Adding delisting returns to monthly data

Ian D. Gow

7 April 2023

This short note uses SAS code by Richard Price to illustrate the process of translating SAS code using WRDS to R code.

1 Introduction

This short note demonstrates how to convert SAS code provided as `delistings.sas` by Richard Price here. This code is presumably closely related to code used in Beaver et al. (2007).

2 Setting up tables

In the R code that folows, we will use a number of packages, including pacakges to connect to a PostgreSQL database containing WRDS data. Instructions for this can be found here.[1]

```
library(dplyr)
library(tidyr)
library(broom)
library(DBI)
library(dbplyr)
library(purrr)
```

```
pg <- dbConnect(RPostgres::Postgres(), bigint = "integer")
```

We will use three tables. The data on "regular" returns come from `crsp.msf`. The tables `crsp.mse` and `crsp.dsedelist` are used for delisting returns.

[1] You don't need to install all the packages listed there, just the ones listed below plus `RPostgres`.

```
crsp.msf <- tbl(pg, sql("SELECT * FROM crsp.msf"))
crsp.mse <- tbl(pg, sql("SELECT * FROM crsp.mse"))
crsp.dsedelist <- tbl(pg, sql("SELECT * FROM crsp.dsedelist"))
```

3 The dataset with monthly return data

Here we get a table of "regular" returns, we will incorporate delisting returns in the returns from this table. The code includes "an arbitrary restriction of the sample for illustration purposes." The data are restricted to 2003 and `permno` values less than 12000.

```
proc sql;
    create table monthlyreturns as
    select permno, date, ret
    from crsp.msf
    where year(date)=2003 and permno<12000;
```

```
monthlyreturns <-
  crsp.msf %>%
  filter(year(date) == 2003, permno < 12000) %>%
    select(permno, date, ret)
```

4 The monthly delisting dataset

4.1 Get the base data for delisting returns

The following code uses `crsp.mse` for the values of `dlret`. It's not clear what this table provides that `crsp.dsedelist` does not, but using `crsp.dsedelist` here—renaming `dlstdt` as `date` to conform with later code—results in small differences in the results below.

The first portion of SAS code is easily translated to R.

```
data delist;
  set crsp.mse;
  where dlstcd > 199;
  keep permno date dlstcd dlpdt dlret;
run;
```

```r
delist <-
  crsp.mse %>%
  filter(dlstcd > 199) %>%
  select(permno, date, dlstcd, dlpdt, dlret)
```

4.2 Calculate replacement values

Compute replacement values for missing delisting returns using daily delisting returns. Richard says modify year range as needed, but I drop this filter in the R code, as it makes little difference to performance and (surprisingly) no difference to the results.

```
proc sql;
  create table rvtemp as
    select * from crsp.dsedelist
    where dlstcd > 199 and 1960 le year(DATE) le 2020
    order by dlstcd;
```

I omit the step of creating `rvtemp`, as it's easy enough to include a single line of code in creating `rv` below.

The following code calculates the mean values of `dlret` by `dlstcd`. Later will use these values to fill missing values of `dlret`. Richard says in a comment "could use `median=median_dlret` and `probm=median_pvalue` if you do not like mean delisting returns as the replacement value."

```
proc univariate data=rvtemp noprint;
    var dlret;
    output out=rv mean=mean_dlret probt=mean_pvalue;
    by dlstcd;
run;

* require replacement values to be statistically significant;
data rv;
    set rv;
      * adjust p-value as desired;
    if mean_pvalue le 0.05 then rv = mean_dlret;
    else rv = 0; * adjust as desired;
    keep dlstcd rv;
run;
```

The SAS code uses `PROC UNIVARIATE`. We just need a small function to return the p-value, which I call `prt()` to match the name of a similar function in SAS. We could use the `t.test()` function, but it's easy enough to calculate the two-sided p-value ourselves.[2]

```
prt <- function(x) {
  p <- pt(mean(x) * sqrt(length(x)) / sd(x), length(x))
  2 * pmin(p, 1 - p)
}
```

As disussed I use `filter(dlstcd > 199)` without `between(year(dlstdt), 1960, 2020)` because it makes no noticeable difference to performance or results. I call the resulting table `rvs` ("replacement values") to distinguish it from the variable `rv` it contains. This makes no functional difference, but (to my mind) makes the code a little easier to read. Note that we bring data from PostgreSQL into R to calculate `p_val` and then return it to PostgreSQL using `copy_inline()`. An alternative might be to calculate t-statistics in PostgreSQL and just bring summary values into R to calculate p-values, but this approach works fine.

```
rvs <-
  crsp.dsedelist %>%
  filter(dlstcd > 199) %>%
  select(dlstcd, dlret) %>%
  filter(!is.na(dlret)) %>%
  collect() %>%
  group_by(dlstcd) %>%
  summarize(mean = mean(dlret),
            p_val = prt(dlret),
            .groups = "drop") %>%
  mutate(rv = if_else(p_val <= 0.05, mean, 0)) %>%
  select(dlstcd, rv) %>%
  copy_inline(pg, .)
```

4.3 Merge replacement values with delisting returns

Again the R code is a simple translation of the SAS code. One difference here is that I use a new table name `delist_rv`, as I find re-using table names to be confusing when debugging code (though no real debugging was required here because I am mimicking someone else's code). Note that I add `month` and `year` variables, as we will use these to merge below data sets below.

[2] It's probably a good thing to do this "by hand" just so you remember your statistics. Of course I checked that I got the same answer as `t.test()`.

```
proc sql;
    create table delist as
    select a.*, b.rv
    from delist a left join rv b
    on a.dlstcd = b.dlstcd;
```

```
delist_rv <-
  delist %>%
  left_join(rvs, by = "dlstcd") %>%
  mutate(month = month(date),
         year = year(date)) %>%
  rename(dldate = date)
```

4.4 Creating a function

Of course, all of the above could be easily be put into a function. Making functions in R is much easier than making macros in SAS.[3] Note that if using the function, we could omit the code creating `crsp.mse` and `crsp.dsedelist` above. Optionally, we could easily use dates taken from `crsp.msi` in the returned table that we could merge with monthly returns without any need to create `month` and `year` fields. Ideally, we would also incorporate the steps to correct `dlret` covered in the next section in the code here.

It would be quite straightforward to add this function to my `farr` package.[4] The only thing needed to be provided to the function is the PostgreSQL database connection (`conn`).

```
get_delist <- function(conn) {
  crsp.mse <- tbl(conn, sql("SELECT * FROM crsp.mse"))
  crsp.dsedelist <- tbl(conn, sql("SELECT * FROM crsp.dsedelist"))

  delist <-
    crsp.mse %>%
    filter(dlstcd > 199) %>%
    select(permno, date, dlstcd, dlpdt, dlret)

  prt <- function(x) {
    p <- pt(mean(x) * sqrt(length(x)) / sd(x), length(x))
    2 * pmin(p, 1 - p)
  }
```

[3] I don't think you'd cover macros in a first SAS class, but I cover making functions here.

[4] Once I understand this code better, I may do this, as it would be good to use returns with delisting returns in the chapter replicating Sloan [1996].

```
  rvs <-
    crsp.dsedelist %>%
    filter(dlstcd > 199) %>%
    select(dlstcd, dlret) %>%
    filter(!is.na(dlret)) %>%
    collect() %>%
    group_by(dlstcd) %>%
    summarize(mean = mean(dlret),
              p_val = prt(dlret),
              .groups = "drop") %>%
    mutate(rv = if_else(p_val <= 0.05, mean, 0)) %>%
    select(dlstcd, rv) %>%
    copy_inline(conn, .)

  delist %>%
    left_join(rvs, by = "dlstcd") %>%
    mutate(month = month(date),
           year = year(date)) %>%
    rename(dldate = date)
}
```

So we can replace the `delist_rv` data table we created above with one produced by the `get_delist()` function.

```
delist_rv <- get_delist(pg)
```

5 Merge monthly returns with delisting data

Translating the SAS code here is pretty straightforward. The SAS code involves `PROC SQL` followed by a data step, but I do it all in one series of pipes.

```
proc sql;
  create table monthlyreturns as
      select a.*, b.dlret, b.dlstcd, b.rv, b.date as dldate, b.dlpdt
      from monthlyreturns a left join delist b
      on (a.permno = b.permno)
      and (month(a.date)= month(b.date))
      and (year(a.date) = year(b.date));
quit;
```

```
data monthlyreturns;
    set monthlyreturns;
    ret_orig = ret;

    if not missing(dlstcd) and missing(dlret) then dlret=rv;
    else if not missing(dlstcd) and dlpdt le dldate and not missing(dlret)
        then dlret=(1+dlret)*(1+rv)-1;

    ** Then, incorporate delistings into monthly return measure;
    if not missing(dlstcd) and missing(ret) then ret=dlret;
    else if not missing(dlstcd) and not missing(ret)
        then ret=(1+ret)*(1+dlret)-1;
run;
```

The code below first uses replacement values where necessary (`is.na(dlret)`). Note, this will happen when the delisting occurs on the last day of the month and `ret` is not missing, but the delisting return is unknown. If the delisting return is a partial month return, CRSP flags it by setting `dlpdt` to a date less than or equal to the delisting date. Richard says that one could use a single replacement value as in Shumway (1997) (-0.35) or Sloan (1996) (-1.0) and that he would only do single replacement value for a subset of delisting codes > 499.

Again I use a new name (`monthlyreturns_delist`) for the resulting table, as I prefer not to reuse table names.

```
monthlyreturns_delist <-
  monthlyreturns %>%
  mutate(month = month(date),
         year = year(date)) %>%
  left_join(delist_rv, join_by(permno, month, year)) %>%
  mutate(ret_orig = ret,
         dlret = case_when(!is.na(dlstcd) & is.na(dlret) ~ rv,
                           !is.na(dlstcd) & dlpdt < dldate & !is.na(dlret) ~
                             (1 + dlret) * (1 + rv) - 1,
                           .default = dlret),
         ret = case_when(!is.na(dlstcd) & is.na(ret) ~ dlret,
                         !is.na(dlstcd) & !is.na(ret) ~
                           (1 + ret)*(1 + dlret) - 1,
                         .default = ret)) %>%
  collect()
```

5.1 Comparison of output with that of SAS code

Richard includes output from SAS's `PROC MEANS` that we can use to compare our results with his. Comparing the numbers in Table 1 with Richard's output confirms that our R code has done the same thing as his.

Table 1: Summary statistics for monthly returns with delisting returns

N	Mean	Std Dev	Minimum	Maximum
3859	0.0418905	0.1820073	-0.991304	5.178572
3842	0.0429354	0.1800152	-0.589041	5.178572
20	-0.1684432	0.3543436	-0.991304	0.333333

5.2 Performance

Running the SAS code above takes between 17 and 20 seconds on the WRDS server. The R code takes 9 seconds using a database on my laptop, and about 20 seconds on the same laptop, but using the remote WRDS database.[5]

I would call this performance comparison in R's favour because the R code is also generating the PDF document you are reading, which takes a few seconds. If using the SAS code, you would likely also need to add time to retrieve the data from the WRDS server if you are running analysis on a local computer.

References

Beaver, W., McNichols, M., Price, R., 2007. Delisting returns and their effect on accounting-based market anomalies. Journal of Accounting and Economics 43, 341–368. https://doi.org/10.1016/j.jacceco.2006.12.002

Shumway, T., 1997. The delisting bias in CRSP data. The Journal of Finance 52, 327–340. https://doi.org/10.1111/j.1540-6261.1997.tb03818.x

Sloan, R.G., 1996. Do stock prices fully reflect information in accruals and cash flows about future earnings? The Accounting Review 71, 289–315.

[5] Running the code above actually takes 27 seconds, but that's because I unnecessarily create `delist_rv` twice. This is the portion of code that takes longer using a remote database because of the need to pull data into R.