



The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search

<http://ageconsearch.umn.edu>

aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.

THE STATA JOURNAL

Editors

H. JOSEPH NEWTON
Department of Statistics
Texas A&M University
College Station, Texas
editors@stata-journal.com

NICHOLAS J. COX
Department of Geography
Durham University
Durham, UK
editors@stata-journal.com

Associate Editors

CHRISTOPHER F. BAUM, Boston College
NATHANIEL BECK, New York University
RINO BELLOCCO, Karolinska Institutet, Sweden, and
University of Milano-Bicocca, Italy
MAARTEN L. BUIS, WZB, Germany
A. COLIN CAMERON, University of California–Davis
MARIO A. CLEVES, University of Arkansas for
Medical Sciences
WILLIAM D. DUPONT, Vanderbilt University
PHILIP ENDER, University of California–Los Angeles
DAVID EPSTEIN, Columbia University
ALLAN GREGORY, Queen's University
JAMES HARDIN, University of South Carolina
BEN JANN, University of Bern, Switzerland
STEPHEN JENKINS, London School of Economics and
Political Science
ULRICH KOHLER, University of Potsdam, Germany

FRAUKE KREUTER, Univ. of Maryland–College Park
PETER A. LACHENBRUCH, Oregon State University
JENS LAURITSEN, Odense University Hospital
STANLEY LEMESHOW, Ohio State University
J. SCOTT LONG, Indiana University
ROGER NEWSON, Imperial College, London
AUSTIN NICHOLS, Urban Institute, Washington DC
MARCELLO PAGANO, Harvard School of Public Health
SOPHIA RABE-HESKETH, Univ. of California–Berkeley
J. PATRICK ROYSTON, MRC Clinical Trials Unit,
London
PHILIP RYAN, University of Adelaide
MARK E. SCHAFER, Heriot-Watt Univ., Edinburgh
JEROEN WEESIE, Utrecht University
IAN WHITE, MRC Biostatistics Unit, Cambridge
NICHOLAS J. G. WINTER, University of Virginia
JEFFREY WOOLDRIDGE, Michigan State University

Stata Press Editorial Manager

LISA GILMORE

Stata Press Copy Editors

DAVID CULWELL, DEIRDRE SKAGGS, and SHELBI SEINER

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go “beyond the Stata manual” in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*), *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

<http://www.stata-journal.com>

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

<http://www.stata.com/bookstore/sj.html>

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
Printed & electronic		Printed & electronic	
1-year subscription	\$ 98	1-year subscription	\$138
2-year subscription	\$165	2-year subscription	\$245
3-year subscription	\$225	3-year subscription	\$345
1-year student subscription	\$ 75	1-year student subscription	\$ 99
1-year institutional subscription	\$245	1-year institutional subscription	\$285
2-year institutional subscription	\$445	2-year institutional subscription	\$525
3-year institutional subscription	\$645	3-year institutional subscription	\$765
Electronic only		Electronic only	
1-year subscription	\$ 75	1-year subscription	\$ 75
2-year subscription	\$125	2-year subscription	\$125
3-year subscription	\$165	3-year subscription	\$165
1-year student subscription	\$ 45	1-year student subscription	\$ 45

Back issues of the *Stata Journal* may be ordered online at

<http://www.stata.com/bookstore/sjj.html>

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

<http://www.stata-journal.com/archives.html>

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.



Copyright © 2014 by StataCorp LP

Copyright Statement: The *Stata Journal* and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp LP. The contents of the supporting files (programs, datasets, and help files) may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

The articles appearing in the *Stata Journal* may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

Written permission must be obtained from StataCorp if you wish to make electronic copies of the insertions. This precludes placing electronic copies of the *Stata Journal*, in whole or in part, on publicly accessible websites, file servers, or other locations where the copy may be accessed by anyone other than the subscriber.

Users of any of the software, ideas, data, or other materials published in the *Stata Journal* or the supporting files understand that such use is made without warranty of any kind, by either the *Stata Journal*, the author, or StataCorp. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the *Stata Journal* is to promote free communication among Stata users.

The *Stata Journal* (ISSN 1536-867X) is a publication of Stata Press. Stata, **stata**, Stata Press, Mata, **mata**, and NetCourse are registered trademarks of StataCorp LP.

Speaking Stata: Self and others

Nicholas J. Cox
Department of Geography
Durham University
Durham, UK
n.j.cox@durham.ac.uk

Abstract. Data for people in a family, firms in a sector, or members of any kind of group are often analyzed using data for the other members in each group. Examples are the number of older children for each child in a family or the mean investment of other firms in the same year. Many such problems yield to simple logic, spelled out here in terms of useful Stata functions and commands. Operations under `by:`, or an equivalent `by()` option, are frequently called for, while the `egen` command offers a key tool for many calculations. Even when looping is needed, some helpful tricks can be identified.

Keywords: dm0075, groups, families, firms, counts, totals, means, by, egen

1 Introduction

Suppose you have data on families and their members or on firms in particular sectors of the economy. A common concern is relating each member of a family to all or some of the other members or relating each firm to all or some of the other firms. Generally, each person or firm is a “self” to be compared with “others”, with those “others” forming a context or environment for each “self”.

Naturally, many operations on such sets are just standard, as when values are ranked or when each value is represented as a deviation from a mean or median or from predictions from some model. In Stata, the `egen` (see [D] `egen`) command is a common starting point for basic descriptive calculations for grouped data, while model postestimation commands yield residuals of differing flavor. However, the problems addressed in this article have two extra twists. The values for “self” and those for “others” are treated separately, and the operation is also repeated for each set member.

We start out with simple examples, made very simple by easy tricks sometimes overlooked, and progress to more challenging examples.

This article is, in a sense, a sequel to Cox (2011), but it is also largely independent of that article.

2 Counting

Counting observations is the first and most straightforward operation relating self and others. The number of other observations satisfying some condition is just the number

of all observations satisfying that condition, minus one if each observation satisfies that condition.

One example of family data in a dataset concerns the 10 children of the scientist Charles Robert Darwin and his wife, Emma Wedgwood. Darwin's contributions to science have received much attention, but the details here on his family are from Berra (2013). Those interested in the Darwin literature should note that this source gives more detail than, and indeed corrects, sources such as Freeman (1978) and Browne (1995).

One of the children, George Howard (Sir George Darwin from 1905), is best remembered for his major work in astronomy and geophysics, but also made notable contributions to statistical science, collected in Darwin (1911), which included two papers on marriages between first cousins (Darwin 1875a,b) and a pioneer contribution on smoothing (Darwin 1877). On the last, see also Stigler (1991).

The dataset comes with the media for this issue as `darwin.dta`. Charles and Emma and their children each define an observation. The children's relation to Charles is defined by the string variable `relation`, which in this dataset takes on the values `self`, `wife`, `son`, and `daughter`. We have dates of birth and death, held as Stata daily dates, and an indicator variable `female`, 1 for female and 0 for male.

Two side comments: I recommend the terminology "indicator variable" over the very common term "dummy variable" if only because I have heard too many stories in which comments of the form "That's just a dummy variable" were wildly misunderstood as implying disdain or disparagement. Similarly, too many presenters have been embarrassed in public for forgetting what codes what for an indicator variable such as `gender`. A convention that indicator variables are named for the category coded 1, here `female`, promotes personal memory and public clarity. In a language such as Stata, it also makes code idioms, for example, `if female` or `if !female`, natural and transparent. Both of these comments are standard but not universal; see Gelman and Hill (2007, 62) on the second comment. Indeed, the indicator variable `foreign` in Stata's standard `auto.dta` dataset is a case in point.

```
. use darwin
. list
```

	name	relation	birth	death	female
1.	Charles Robert	self	12 Feb 1809	19 Apr 1882	male
2.	Emma	wife	2 May 1808	20 Oct 1896	female
3.	William Erasmus	son	27 Dec 1839	1 Sep 1914	male
4.	Anne Elizabeth	daughter	2 Mar 1841	23 Apr 1851	female
5.	Mary Eleanor	daughter	23 Sep 1842	16 Oct 1842	female
6.	Henrietta Emma	daughter	25 Sep 1843	17 Dec 1927	female
7.	George Howard	son	9 Jul 1845	7 Dec 1912	male
8.	Elizabeth	daughter	8 Jul 1847	8 Jun 1926	female
9.	Francis	son	16 Aug 1848	19 Sep 1925	male
10.	Leonard	son	15 Jan 1850	26 Mar 1943	male
11.	Horace	son	13 May 1851	22 Sep 1928	male
12.	Charles Waring	son	5 Dec 1856	26 Jun 1858	male

Although the Darwin dataset includes only one family, a variable `familyid` allows the specification of more general code that will work—and would still work if additional families were included in the same dataset. Here there is an obvious family identifier, but numeric identifiers are much more common in work of this kind if only to protect anonymity and privacy.

```
. generate familyid = "Darwin"
```

As already mentioned, `egen` provides a workhorse for group calculations. If `egen` is relatively new to you, you can study the help file and Cox (2002b) for more information.

With the framework of a family identifier,

```
. by familyid, sort: egen n_female = total(female)
```

and

```
. egen n_female = total(female), by(familyid)
```

are two ways to count females. The first is the documented syntax in recent versions of Stata; the second is an older syntax that still works and is a form you may often see in code written by more experienced users.

Let me emphasize what is happening here. There is only one family in the dataset, but I am giving code that would work if there are many families. Note that code such as

```
. egen n_female = total(female)
```

would produce a variable containing one distinct result, the number of females in the dataset. On occasion, that is what you want to do, but whenever single totals are of interest, it is usually better to use `summarize` (see [R] `summarize`) or `total` (see [R] `total`). With the former, there is a small twist that is often useful, especially

in do-files or programs. Contrary to what you might expect from the option name, **summarize**, **meanonly** will calculate the sum (total); you just have to ask explicitly to see it. See Cox (2007b) if you wish for more detail on this point.

```
. summarize female, meanonly
. display r(sum)
5
```

Back to **egen**: the valuable point about using **by:** or **by()** is that separate results for each group are ensured, here each family as identified by the same identifier value.

That done, how many others in the family were female? For this, you need to subtract one if a particular person is female and zero otherwise, which means you subtract the indicator variable already in the dataset and already used.

```
. generate n_others_female = n_female - female
```

Here we do not need the machinery of **by:**, let alone **by()**, because the calculation is within each observation. It is when we need to work with groups of observations together that such machinery becomes important. A tutorial on **by:** in Cox (2002a) may be helpful if you are new to the idea. See also the manual entry [D] **by** for a concise formal statement.

We do not need to have indicator variables in hand to apply these principles. Recall that **relation** here is a string variable. Then

```
. egen n_dau = total(relation == "daughter"), by(familyid)
```

counts daughters in each family. The true-or-false **relation == "daughter"** is evaluated as 1 if true and 0 if false. Therefore, the total of ones and zeros is just the total of ones and the number of daughters.

If you are surprised that you can do this, consider that the syntax for **egen** indicates that **total()** works on an expression, which need not be a single variable name but can be something more complicated.

As before, we can correct the variable just created if we want a variable counting how many other members of the same family are daughters. It is the same idea: we need to subtract one or zero depending on whether somebody is a daughter.

```
. generate n_others_dau = n_dau - (relation == "daughter")
```

Here many Stata users would want to write something like

```
. generate n_others_dau = n_dau
. replace n_others_dau = n_others_dau - 1 if relation == "daughter"
```

Or they might do this if they were familiar with the **cond()** function:

```
. generate n_others_dau = cond(relation == "daughter", n_dau - 1, n_dau)
```

The `cond()` function offers clear and clean coding once you understand its basic form. In the simplest variant—that shown here—the first argument is a true-or-false condition; the second is the result if the condition is true; and the third is the result if the condition is false. So here the function is told that if `relation` is "daughter", return `n_dau - 1`; otherwise, return `n_dau`. Kantor and Cox (2005) wrote a brief tutorial on this valuable function.

We need not argue about which code is best. In your own work, clarity for yourself—and for any collaborators—may well decide the matter. But getting used to the first form through practice will help make the idiom natural and clear to you.

I stress again that in this calculation, we do not need to create a new indicator variable to correct for those people who are themselves daughters. The expression `relation == "daughter"` is directly evaluated as 1 or 0. But by all means, create an indicator variable if you need one for another purpose.

Counting problems are often more complicated than this, but they need not be more difficult. Suppose we want to know how many daughters were alive on 24 November 1859. Dates of birth and death are in the dataset, so the condition is compound:

```
(relation == "daughter") & inrange(mdy(11, 24, 1859), birth, death)
```

If the function `inrange()` is new to you, the idea is that being alive on a certain date is a matter of being born earlier *and* having died later. Here a particular date, 24 November 1859, is supplied as a constant, but in practice, a date used for comparison may well be, and often is, variable from observation to observation.

Information on what happens if `inrange()` encounters exact equalities or missing values is covered in the documentation or more discursively in Cox (2006). In short, `inrange()` is inclusive when given equalities but not missing values, and the operators are \leq but not with respect to missings.

As conditions get more complicated, code tends to get longer. We could set up an abbreviation using a local macro,

```
. local condition (relation == "daughter") &  
> inrange(mdy(11, 24, 1859), birth, death)
```

and then our code becomes more readable:

```
. egen n_dau_1859 = total(`condition`), by(familyid)  
. generate n_others_dau_1859 = n_dau_1859 - (`condition`)
```

With Stata code—or more generally any code in similar software—there is always the question of what could go wrong. Our dataset is small and complete, but in larger, messier datasets, missing values are much more likely. If the value of `relation` were missing, then zero would be subtracted, which is defensible. If you want missing values in any data to be matched by missing values in generated variables, you should add an extra condition:

```
if !missing(relation)
```


I find it helpful to spell out such qualifier clauses to myself, here as “if *not* missing on the variable `relation`”, to reinforce the syntax.

3 Totals and means

Let’s now consider the calculation of totals and means. We can already guess how to calculate totals in general because we have used the `egen` function `total()` to calculate counts, and the good news should be no surprise: in general, everything carries over to totals.

The mean of other values can be spelled out as

$$\begin{aligned}\text{mean of other values} &= (\text{total of other values}) / (\text{count of other values}) \\ &= (\text{total of all values} - \text{this value}) / (\text{count of all values} - 1)\end{aligned}$$

so we just have to get the ingredients first.

Let’s move to another example. The Grunfeld dataset bundled with Stata is a widely used small-panel dataset on 10 distinct companies observed for 20 years. It is well behaved with no missing values. I say “the” Grunfeld dataset, but that claims too much given its checkered history in economic literature (see Kleiber and Zeileis [2010]). Let’s work out the mean investment of other companies in the dataset for each year.

The total investment is for each year,

```
. egen totalinvest = total(invest), by(year)
```

so we can determine this with

```
. generate meaninvest_others = (totalinvest - invest) / 9
```

because for each company, there are nine others and no missing values to worry about. However, thinking more generally does no harm. `count()` is a helpful `egen` function for counting nonmissing values, so more complete and more general code is

```
. egen totalinvest = total(invest), by(year)
. egen countinvest = count(invest), by(year)
. generate meaninvest_others = (totalinvest - invest) / (countinvest - 1)
```

Are we there yet? Suppose `invest` is missing in a given observation. Then the variable just calculated will also be missing. That might be as you wish—or otherwise put, not a problem—but you might want to use the information about other values wherever it was available. If so, a fix is

```
. replace meaninvest_others = totalinvest / countinvest if missing(invest)
```

In essence, there is nothing to subtract if `invest` is missing in a given observation. A long-winded way to get the same result in one line is

```
. generate meaninvest_others = (totalinvest - cond(missing(invest), 0, invest)) /
> (countinvest - cond(missing(invest), 0, 1))
```

That has a little symmetry about it, which may be appealing, but it can also be made shorter: for example, `cond(missing(invest), 0, 1)` reduces to `!missing(invest)`.

4 Larger, largest, smaller, smallest

A bundle of related problems arises requiring comparison based on inequalities in magnitude. Let's return to my example on families: researchers often want to identify whether a child is the oldest or the youngest or, more generally, how many children are older or younger than a given child (clearly zero children are older than the oldest and younger than the youngest). The key principle is that every such calculation is easy once you have the right sort order. The main practicality is ensuring that irrelevant observations do not get in the way.

With the Darwin data again as an example, it is helpful to create an indicator for being a child. I will give code for several example variables before commenting.

```
. generate child = inlist(relation, "son", "daughter")
. by child familyid (birth), sort: generate first = _n == 1
. by child familyid: generate last = _n == _N
. by child familyid: generate nolder = _n - 1
. by child familyid: generate nyounger = _N - _n
```

Other examples could be added: for instance, once we know the number of children older or younger than a given child, then we can get indicators for there being older and younger siblings.

Note the use of `inlist()` to produce the indicator variable. Other ways to do it include the simple alternative

```
. generate child = (relation == "son") | (relation == "daughter")
```

but `inlist()` should grow on you as a clean way to spell out more complicated definitions. For more information, see Cox (2006), as cited earlier.

The strategic point is to first define the sort order to segregate observations that are not relevant, here the adults for whom `child` is 0, as well as to keep each family together. The framework of `by:` means that the values for observation number `_n` and number of observations `_N` are defined *within groups of observations*. That detail is the detail on which all else here depends.

After we sort on birth date for children within families, the first child has the first, hence smallest, birth date, the last child has the last or largest, and the number of children older and younger can be calculated from the sort order. You can avoid errors

in which the counts are off by one by testing boundary cases. So, for the first child, the observation number `_n` is 1, as used in the definition of `first`. The number for older is $1 - 1 = 0$, which is right. Similarly, for the last child, the number for younger is 0, which is also right. Another kind of boundary case occurs when there is only one child. The definitions imply logical results of no child being younger or older. Also the first child and the last child are one and the same, which makes sense. (If there are no children in a family, no observations are worked on, which is fine too.)

Note that the code does not suppress results for observations that are not children, namely, parents. Results could be left as they are because they have some meaning, such as defining which parent is older and which younger, or they could be suppressed as having no use by using an `if` qualifier. That is, calculations could be conditional on `if child`, with results returned as missing otherwise.

In this example, you might be tempted by the following logic: sort by families and then by birth date. Of necessity, the parents are both older than their children, so the first child has observation number 3, not 1, with corresponding adjustments for the definition of the numbers for older and younger. (The youngest child with the highest birth date remains correct.)

However, writing code depending on empirical values or on a decency assumption about usual cases is not a good idea. Such code is fragile and dangerous. You only have to imagine that some families may have just one parent recorded, or none at all, or that datasets of this kind are often for households containing one or more families. The principle of segregating irrelevant observations is strong by comparison.

Another way to proceed here would be to use the `max()` and `min()` functions of `egen()`. The oldest child then necessarily has the minimum birth date, and the youngest child has the maximum birth date. You could then count older and younger using the approach of section 2. In general, however, this would be more complicated and less direct than the approach of this section.

Let's now address what may seem harder problems. What is the maximum or minimum birth date of the other children?

The problem is easier than it may seem. The maximum birth date of the other children is that of the oldest, unless we are looking at the oldest, in which case it is that of the second oldest. Similarly, the minimum birth date is that of the youngest, or second youngest, depending on who is the focus. Picking up earlier code and emphasizing the sort order again, we type

```
. by child familyid (birth), sort: generate oldest_other =
> cond(_n == 1, birth[2], birth[1])

. by child familyid: generate youngest_other = cond(_n == _N, birth[_N-1],
> birth[_N])
```

The new twist here is yet another virtue of using the framework of `by:.` Subscripts such as `[1]` in `birth[1]` are defined within groups of observations, not within the dataset as a whole. Note that using the term “subscript” refers to role, not typography. The analogy is with subscripts such as 1 in x_1 and is not a matter of exactly how they are written in Stata.

What happens if there is only one child? Then the definitions would point to `birth[2]` and `birth[0]`, respectively. Stata takes such references in stride. Whenever a subscript points outside the data, a missing value is returned, which is how it should be: there is no other child, and we cannot have anything but a missing birth date as a result. Remember: Under the aegis of `by:.`, outside the data means outside the group being considered. Other groups, here other families, are irrelevant.

So what might go wrong? The most likely complication is that all or some of the birth dates are missing. If some are missing, we can either declare the result all missing or ignore the missings and work with what we have. The latter would best entail segregating the missings so that they do not interfere with the calculation.

5 If all else fails

Let’s now take a more pessimistic view and say that no special trick is possible or, more personally, that no special trick is evident to us. The worst scenario is that you must write a loop over the distinct possibilities.

That should not mean a loop like this:

```
initialize results variable

for each observation {
  calculate something for others in the same group
  replace result variable for that member of that group
}
```

A loop like this should often be possible:

```
initialize results variable

for each member of the group {
  calculate something for others in the same group
  replace result variable for that member of that group
}
```

Here we are simultaneously working on members of different groups with the same internal identifier. For convenience, each identifier should be numeric and run over the integers one upward, so the important detail to know is the maximum identifier in the dataset. If no such identifier exists, you can easily create one with a statement like

```

. by familyid, sort: generate personid = _n
. summarize personid, meanonly
. display r(max)
10

```

Let's return to the Grunfeld data. Suppose that we want to know, for each company, the median investment of the other companies in any given year. Here is a way of doing that, exploiting the fact that the `company` identifier is already in the form desired.

```

. use http://www.stata-press.com/data/r13/grunfeld, clear
. generate exclude = .
(200 missing values generated)
. generate median = .
(200 missing values generated)
. generate work = .
(200 missing values generated)
. quietly forval j = 1/10 {
.   replace exclude = missing(invest) | company == `j'
.   by year exclude (invest), sort: replace work =
>     cond(mod(_N, 2), invest[( _N + 1)/2], (invest[_N/2] + invest[_N/2 + 1])/2)
.   by year: replace work = work[_n-1] if exclude
.   replace median = work if exclude
. }

```

The root of the problem lies in excluding each company in turn. We do that by setting up an indicator variable, here called `exclude`, that flags observations excluded from the current calculation. It should also always flag any observations excluded from all calculations, most obviously observations with missing values, because they would complicate calculation.

A subtle detail with the example code just given is that it does matter which way inclusion and exclusion are coded. Tagging included observations by 0 and excluded observations by 1 implies that we can easily copy wanted results downward once they are sorted.

For this problem of medians, we need not invoke `summarize`, `detail`, because it would do much more than we need. Provided that values are sorted by magnitude, we can use a standard recipe from elementary statistics: the median is the middle value when the number of values is odd and the mean of two middle values when the number of values is even. Odd and even are given by the remainder on division by two using the versatile `mod()` function (Cox 2007a). (`mod(x, 2)` for integer x is 1 when x is odd and 0 when x is even, so that expression gives a true-or-false result for being odd.)

Alternatively, using subscripts as in

```
(invest[floor((_N+1)/2)] + invest[ceil((_N+1)/2)])/2
```

would work regardless of whether `_N` was odd or even because the two expressions for subscripts return the same integer if it is odd and return adjacent integers if it is even.

Here is another way to calculate medians that is a little quick and dirty but has some generality.

```
. use http://www.stata-press.com/data/r13/grunfeld, clear
. generate median = .
(200 missing values generated)
. quietly forval j = 1/10 {
.     egen work = median(invest / (company != `j')), by(year)
.     replace median = work if company == `j'
.     drop work
. }
```

The most devious trick here is the way of excluding the company currently under focus. I call it the “divide by zero” trick after its most surprising detail. It was discussed in Cox (2011), but I will explain the main idea again.

Consider the true or false expression

```
company != `j'
```

The reference to the local macro `'j'` is evaluated as 1 to 10 as we cycle over the loop, so the expression starts the first time around the loop as

```
company != 1
```

This expression is 1 whenever `company` is not 1 and 0 whenever `company` is 1, and it behaves similarly for values of `company` from 2 to 10, namely, all the possible values. So the expression

```
invest / (company != 1)
```

becomes `invest / 1` or `invest / 0`. The first is naturally just `invest`. The twist is that the second is treated as missing, and it is then ignored by `egen` in calculations. This is not a problem; it is exactly what we want.

Let us see this in another way. Why is the code not as follows?

```
. egen work = median(invest) if company != `j', by(year)
```

That would produce the same results, but an inevitable side effect is that missing values are returned for observations where `company == 'j'`, precisely those in which we want to store the result. We could fix that by copying results to where they belong, but a better solution is to avoid the problem in the first place.

The same general scheme can be used for other `egen` functions that take an expression as an argument and allow operations under `by:` or `by()`. There is no promise

that such a function exists for your problem, but it may be worth looking for one using `search`. In fact, more `egen` functions have been written outside official Stata than within it, although the official functions cover most of the more common tasks.

6 Conclusions

Researchers working with people in families, firms in sectors, and members of groups of different kinds quite often want to calculate something for the other members of each group. Counts, totals, and means are all easy as variants on the identity:

$$\text{total for others} = \text{total for all} - \text{this value}$$

How many are larger or smaller can be counted from the order of observations in a group once it is sorted. More complicated problems may require a loop, but there are ways to make such loops less painful than you might fear, particularly in terms of time to code or time to execute.

In this article, I have emphasized writing down code to get results. As programming courses often stress, you should get code working before you try to make it faster. Although `egen` is often a convenient framework for group calculations, it is not the fastest machine in town. Even so, whenever speed is an issue, many of the basic small and large tricks discussed here will be useful for your own faster Stata or Mata code.

7 References

- Berra, T. M. 2013. *Darwin and His Children: His Other Legacy*. New York: Oxford University Press.
- Browne, J. 1995. *Charles Darwin: Voyaging. Volume I of a Biography*. London: Jonathan Cape.
- Cox, N. J. 2002a. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86–102.
- . 2002b. Speaking Stata: On getting functions to do the work. *Stata Journal* 2: 411–427.
- . 2006. Stata tip 39: In a list or out? In a range or out? *Stata Journal* 6: 593–595.
- . 2007a. Stata tip 43: Remainders, selections, sequences, extractions: Uses of the modulus. *Stata Journal* 7: 143–145.
- . 2007b. Stata tip 50: Efficient use of `summarize`. *Stata Journal* 7: 438–439.
- . 2011. Speaking Stata: Compared with *Stata Journal* 11: 305–314.
- Darwin, G. H. 1875a. Marriages between first cousins in England and their effects. *Journal of the Statistical Society* 38: 153–182.

- . 1875b. Note on the marriages of first cousins. *Journal of the Statistical Society* 38: 344–348.
- . 1877. On fallible measures of variable quantities, and on the treatment of meteorological observations. *Philosophical Magazine* 4: 1–14.
- . 1911. *The Scientific Papers of Sir George Darwin. Volume 4: Periodic Orbits and Miscellaneous Papers*. Cambridge: Cambridge University Press.
- Freeman, R. B. 1978. *Charles Darwin: A Companion*. Folkestone, UK: Dawson.
- Gelman, A., and J. Hill. 2007. *Data Analysis Using Regression and Multi-level/Hierarchical Models*. Cambridge: Cambridge University Press.
- Kantor, D., and N. J. Cox. 2005. Depending on conditions: A tutorial on the cond() function. *Stata Journal* 5: 413–420.
- Kleiber, C., and A. Zeileis. 2010. The Grunfeld data at 50. *German Economic Review* 11: 404–417.
- Stigler, S. M. 1991. Stochastic simulation in the nineteenth century. *Statistical Science* 6: 89–97.

About the author

Nicholas Cox is a statistically minded geographer at Durham University. He contributes talks, postings, FAQs, and programs to the Stata user community. He has also coauthored 15 commands in official Stata. He was an author of several inserts in the *Stata Technical Bulletin* and is an editor of the *Stata Journal*.