The Stata Journal (2007)
7, Number 4, pp. 571–581

Speaking Stata: Counting groups, especially panels

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

Abstract. Counting panels, and more generally groups, is sometimes possible in Stata through a reduction command (e.g., `collapse`, `contract`, `statsby`) that produces a smaller dataset or through a tabulation command. Yet there are also many problems, especially with irregular sets of observations for varying times, that do not yield easily to this approach. This column focuses on techniques for answering such questions while maintaining the same data structure. Especially useful are the Stata commands `by:` and `egen` and indicator variables constructed for the purpose. With `by:` we often exploit the fact that subscripts are defined within group, not within dataset. `egen` functions are often used to produce group-level statistics. Tagging each group just once ensures that summaries, including counts, are of groups, not individual observations.

Keywords: dm0033, data management, panels

1 Introduction

Hierarchical or multilevel data are the focus of many statistical problems. Some researchers may even deal with nothing else in their daily statistical work. While modeling methods for such data attract the greatest publicity, many questions arising with multilevel structures also call for basic data management. In this column I focus on a fundamental kind of query. Using the language of panel data, the generic question is, "How many panels . . . ?" Recently in the *Stata Journal*, I emphasized the value of the `count` command (Cox 2007a; see [D] **count**) and explained how to calculate the number of events in time intervals (Cox 2007b). The kind of question discussed here is often even simpler. The challenge is to translate it into Stata terms.

Medical research provides examples of panels that are easy to think about. You might have x-ray records with patient identifier, age of patient, and date of x-ray as variables, or you might have drug prescription records with identifier, drug and date of prescription as variables. Similar panels also arise frequently in social sciences or business. You might have sales records with customer identifier, product purchased, and date of purchase.

These problems are not restricted to datasets indexed by identifiers and dates. Those with daily meteorological data might ask, "In how many years . . . ?" Here years play the same role as panels. Furthermore, no time basis is needed. A dataset on individuals organized by families might lead to questions about how many families satisfy certain criteria, such as having two or more children or being all female.

With such examples in mind, let us identify the kind of problem a little more generally:

1. Observations (records) refer to individuals (x-rays, prescriptions, sales, days, family members).
2. Observations are grouped in some way, commonly into panels (patients, customers), but also into years, families, and so forth. For simplicity, we will discuss panels and trust that it is clear that the ideas apply more generally.
3. No time basis is necessary for the ideas here to apply. When date and/or time is an aspect of the data, there is no assumption that data are regularly spaced in time, or even that this is ideal.
4. The researcher's question is about panels. Each panel corresponds to one or more observations. We have to think, therefore, on two levels: that of the panels and that of the individual observations.

Even with that specification, there remain several ways of approaching such questions in Stata. We can seek to produce a reduced dataset, with one observation per panel, that contains the answers to our question. Commands such as `collapse`, `contract`, and `statsby` (see corresponding entries in [D]) feature strongly in this approach, sometimes after some preprocessing. A tabulation may also be the answer after some preprocessing. Both approaches have several attractions, but neither is the main focus here. Once you realize that such commands offer a solution, the problem is typically almost solved—the main issue being creating the precise syntax.

The focus here is on a different strategy in which we maintain the same data structure. The tools that are repeatedly useful include `by:`, `egen` and indicator variables constructed for the purpose. These are discussed in many Stata tutorials (e.g., Cox 2002a, Cox 2002b), but it can take much practice before they become intuitive.

2 Data structure is the problem, but also the solution

Consider again the issue of data structure. This kind of question is a little tricky because a data structure that is in most ways convenient, and indeed natural, is nevertheless awkward for some purposes.

Imagine, once more, data on x-rays or drug prescriptions. Irregular timing and differing numbers of observations in such panels are facts of life. Thus it would usually be artificial and, indeed, inefficient to `reshape` datasets into wide structures with just one observation for each patient. You could represent the first, second, and so forth of several visits by variables such as `date1`, `date2`, and so on. But you would need as many such variables as there were observations in the panel with most observations.

There would then be many missing values. Worst of all, many questions would still be difficult to answer.

Hence the strategy discussed here keeps the same structure. It is likely to be easier to work with, not only for the questions discussed but for other kinds of questions for the same datasets.

3 A first example: panels of firms

Let us work out a Stata answer to a question about a specific dataset. You can read it in yourself, assuming that you have Internet access:

```
. webuse grunfeld
```

This happens to be a well-behaved panel dataset. Each of 10 companies is observed for each of 20 years. There are no gaps and no missing values. However, we do not require such good behavior, as you will see.

How many companies, and which ones, experienced higher than average growth in `invest` over the period? Clearly, this is a question about companies, but it must be answered from data on individual companies in specific years. Under the terms of our self-denying ordinance, we will not resort to a reduction or `reshape` of the data, nor will we exploit the equal lengths of the panels or their regular timing in any way.

There could be discussion on how best to measure growth, but we choose a simple definition of (last − first) / first.

```
. by company (year), sort: gen invest_change = (invest[_N] - invest[1]) / invest[1]
```

The first Stata lesson here is a reminder of the value of `by:`. One basic tutorial was given in an earlier column (Cox 2002a). Alternatively, check out entries on `by:` indexed in the Stata manual. `by:` is the basic tool for calculations by panels or other groups. Those commands that appear to sidestep it will almost always use it internally.

Even if the data are already sorted in the way you want, it does no harm to spell out a sort order. That will also, on occasion, save you from incorrect results when the data are not sorted as you desire. It is also helpful when looking back at past log files that give session transcripts.

A feature we are exploiting here is that under `by:`, subscripts are interpreted within the groups defined by `by:`. We insisted, as part of the command just given, that the observations are sorted by `year` within blocks of observations for each `company`. That being so, `invest[1]` and `invest[_N]` are values of `invest` for the earliest and latest observations for each company.

You should appreciate two possible problems with this calculation. If `invest[1]` is ever 0, the calculation will yield a missing result. Similarly, if either first or last value is missing, we will also get a missing result.

Clearly, the advice is to check your data so that you can identify problems such as these. The second problem is sufficiently common that a more general method of dealing with it is worth knowing about. In related form, it will be familiar to Stata programmers.

Construct an indicator variable that captures missingness:

```
. gen byte invest_miss = missing(invest)
```

Using a `byte` type is not essential unless you are short of memory but is good practice. The new variable will contain 1 whenever `invest` is missing and 0 otherwise. With this in hand, we can tweak the sort order:

```
. by invest_miss company (year), sort: gen invest_change =
>  (invest[_N] - invest[1]) / invest[1]
```

The sort order is now first by `invest_miss`, second by `company`, and third by `year`. Thus all the observations with missing values of `invest` have been put on one side in quarantine, where they cannot infect calculations made with the nonmissing values. `invest[1]` and `invest[_N]` now refer to the first and last years in each group defined jointly by whether `invest` is missing and `company`. Thus we would always use the first and last years for which values are available. Otherwise, missing values of `invest` would inevitably yield missings for `invest_change`.

You might not want this. You might say: I want to use data for the first and last years of the study consistently. If the result is missing, I want to know that and then ignore the result. That is a scientific or practical decision for you to make. The point is to know how to tackle the issue in different ways within Stata. With irregular times and a sprinkling of missings, using the first and last nonmissings could easily be the better choice.

With the Grunfeld data all that would make no difference, so we will not pursue the detail further in this example. Your own data may not be so well behaved.

We now have measures of growth, so what about the average? You could reach for `summarize`. However, that without qualification would summarize 200 observations, and not 10 panels. Although the difference will not bite with equal-length panels, we need a more general method. (Conversely, if you did want summaries over panels to be weighted by number of observations within panel, then `summarize` on the observations is exactly right: just be sure that you choose it consciously rather than accidentally.)

Inspection of the data with `edit` or `list` will confirm what you will have expected: the results for `invest_change` are identical within each panel. Thus we need to pick just one observation from each panel for further use. You could be quite capricious about this and pick any one haphazardly or use your favorite small integer (the third, or the seventh, or whatever), but clearly none of those methods is general. As panels could be as small as a single observation, there are two general methods of selection when all values are identical within panels: selecting either the first or the last. (The last of one is the same as the first of one.)

If we used the first, the technique once more is to produce an indicator variable:

```
. by company: gen byte tag = _n == 1
```

The new variable, `tag`, will be 1 for the first observation in each panel, for which `_n == 1` is true, and 0 otherwise. The new variable, in particular, will never be missing. The name `tag` may suggest that we are tagging some observations for future study, and it is also a prompt for another way of doing essentially the same:

```
. egen tag = tag(company)
```

That alternative is less typing, but a bit more work for Stata. By the way, producing a `byte` variable is wired in to `egen, tag()`, even if you specify otherwise.

`egen` is a major help in this kind of work, and it pays to be familiar with its possibilities (Cox 2002b; [D] **egen**). These include not only those `egen` functions provided with official Stata, but an even larger number of user-written functions: use `findit` to identify what is available from where. In other panel calculations, researchers frequently want to relate data to summaries over panels or over times, and `egen` is then often the tool of choice. It is also more direct than using `collapse` to produce a set of summaries and then using `merge` to put those summaries back into the original dataset, a route surprisingly often followed.

If we decided to tag the last observation in each panel, the syntax would be

```
. by company: gen byte tag = _n == _N
```

Given such an indicator variable, we now have an easy way to pick precisely one observation from each panel—in situations in which every value of a variable of interest is the same within each panel. Do note that stipulation carefully. We can just add `if tag == 1` to commands as appropriate. In fact, we can do better than that. All we need say is `if tag`. The correspondence is simply this: the logical expression `tag == 1` is true, and thus evaluates to 1, precisely when `tag` itself is nonzero, because the only nonzero (true) value possible for `tag` is 1. (Recall the flag earlier that `tag` cannot be missing, a property also wired into `egen, tag()`. There is more on true and false in Stata in Cox [2002a].)

With such tricks, we are now almost finished with this question. The mean of our growth measure across companies is given by

```
. summarize invest_change if tag
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
invest_cha~e |        10    2.843469    1.758583   .8527977   5.666012
```

We will save this mean in a local macro:

```
. local mean = r(mean)
```

How many companies have higher than average growth?

```
. count if invest_change > `mean´ & tag
  5
```

And which are they?

```
. levelsof company if invest_change > `mean´ & tag
1 3 4 6 8
```

The extra condition `& tag` is redundant in the last statement, but it does no harm.

4 A second example: visits to a clinic

How many patients who visited a clinic in 2005 also visited in 2006? For this kind of calculation, we need an identifier variable, say, `id`, and a date variable, say, `date`, which we will take to be a daily date. The logical condition marking a visit in 2005 may be expressed in various ways. Here are two:

```
if year(date) == 2005
```

and

```
if inrange(date, mdy(1,1,2005), mdy(12,31,2005))
```

The first is pleasingly direct for the question asked. The second is very much worth knowing because of the way it can be modified for many other problems, especially those in which the interval is more irregular, usually for some very compelling external reason.

Some Stata morals deserve comment. Beginning Stata users often work through a problem like this by thinking: I need a variable specifying the year to make progress. A variable for year would indeed make the problem look simpler, but often you can move directly to a solution just by using functions such as `year()`. More generally, scanning the documentation giving the list of functions ([D] **functions**) will often make clear some tools that you have been missing. It is one of the driest parts of the Stata documentation, all characters and no plot, but one of the most useful.

`inrange()` is often overlooked. You could always spell out a range using the component inequalities, but using `inrange()` can often be simpler. What is more, it excludes missings, usually the right thing to do. Otherwise, `inrange()` always includes its end-values, and so specifies closed intervals $[a, b]$. Thus `inrange(1, 1, 7)` and `inrange(7, 1, 7)` both evaluate to 1, meaning true. Further examples are given in Cox (2006).

We could construct indicator variables as before

```
. gen byte in2005 = inrange(date, mdy(1,1,2005), mdy(12,31,2005))
. gen byte in2006 = inrange(date, mdy(1,1,2006), mdy(12,31,2006))
```

which flag on the observation level whether (or not) a visit was in 2005 or 2006. But there is a better way to do it. As we want answers for patients (our panels), all we care about is whether there was any visit in 2005 and also any visit in 2006 by each patient. A cumulative sum given by the `sum()` function will serve well. We can calculate, separately by `id`,

```
. sum(inrange(date, mdy(1,1,2005), mdy(12,31,2005)))
```

and its twin

```
. sum(inrange(date, mdy(1,1,2006), mdy(12,31,2006)))
```

which will get us most of the way toward a solution. Here calls to three distinct functions, `mdy()`, `inrange()`, and `sum()`, are nested. As with elementary algebra, Stata works from the inside outward when given an expression with multiple parentheses.

As a detail on the side, we could calculate the fixed dates and feed them to Stata as constants in local macros

```
local b2005 = mdy(1,1,2005)
local e2005 = mdy(12,31,2005)
local b2006 = mdy(1,1,2006)
local e2006 = mdy(12,31,2006)
```

and so work with

```
. sum(inrange(date, `b2005´, `e2005´))
```

and

```
. sum(inrange(date, `b2006´, `e2006´))
```

That should be less work for Stata, as otherwise the `mdy()` calculations are repeated for each observation. However, I would never do this unless I was writing a program or do-file likely to be used repeatedly. Life is too short to optimize all your code.

Now consider what happens for each panel as either cumulative sum is calculated. Sums produced by `sum()` are always initialized as 0. (Occasionally, you have to fight this, as when the sum of several missings is 0, not missing, and that is not what you want.)

For all dates up to and including 2004, both `inrange()` calls will yield 0. Thus the cumulative sums will remain 0. For each date in 2005, the first `inrange()` call will yield 1, and correspondingly for 2006 and the second `inrange()` call. Any dates in 2007, or later, would also yield 0. Thus the cumulative sums yielded by `sum()` are cumulative counts of visits in the two years.

All the question requires is knowing, for each patient, whether there was any visit in 2005 and also any visit in 2006. We can do most of the work in one command:

```
. by id, sort: gen byte both =
> sum(inrange(date, mdy(1,1,2005), mdy(12,31,2005)))
> &
> sum(inrange(date, mdy(1,1,2006), mdy(12,31,2006)))
```

Note the logical "and" (`&`) operator. The result of the logical expression will be true (1 numerically) if and only if both sums are true (nonzero). With this calculation, cumulative sums can never be negative, as they are initialized as zero and we only ever add 1s or 0s as we work through each panel. Thus a cumulative sum can only be zero (no events of the kind specified) or positive (at least one event of the kind specified).

There is more technique that is useful. The answer we want for each panel will be in the last observation for each panel. We can select that too within the same command:

```
. by id, sort: gen byte both = cond(_n == _N,
>  sum(inrange(date, mdy(1,1,2005), mdy(12,31,2005))) &
>  sum(inrange(date, mdy(1,1,2006), mdy(12,31,2006))), .)
```

The `cond()` function here splits the universe in two. If an observation is the last in its panel, that is `_n == _N`, we get the cumulative sum we want, the final result at the end of the panel. For all other observations in each panel, we just get missing. Stata is obliged to work out the cumulative sums for every observation, but only for the last observation in each panel does it put the result in our new variable `both`.

Once more we are exploiting the fact that under `by:` subscripts are defined within group, defined here by `id`, not within dataset. The `cond()` function here lets us do in one statement what would otherwise require two, or an even more complicated statement. For a tutorial, see Kantor and Cox (2005).

Using `cond()` in this way has a nice downstream effect. The result of the previous command, centered on a logical operation using `&`, could be 0 (when either cumulative sum was zero, or both), 1 (when both cumulative sums are nonzero, meaning positive in this problem), or missing (whenever the observation was not the last in its panel). We can therefore proceed directly to counting panels with visits in both 2005 and 2006:

```
. count if both == 1
```

In other words, the effect of the statement was to put nonmissing results for `both` into just one observation in each panel, as we did earlier, but differently, when we defined a tag variable. So we can count what we want without tagging.

5 An egen solution for visits to a clinic

There is another way to do it, which hides some of the details here, and thus has its attractions. It is less efficient in computing terms. However, you might not care about that, especially whenever your personal time taken to figure out Stata code is far greater than any possible machine time to produce the results.

We can use the `egen` function `total()` to get totals or sums. These are not cumulative sums, however, and so will be constant for whatever group of observations is used. (Before Stata 9, `egen`'s `total()` function was called `sum()`. See [D] **egen**.)

The syntax for `egen, total()` reveals that it will feed on an expression, which can be (much) more complicated than one variable name. This detail is, it seems, frequently overlooked. In revisiting the example, we use the `year()` function as a reminder of that solution. As before, the sum or total of a logical expression yielding 0 or 1 yields the count or frequency of observations in which that expression is true.

```
. by id, sort: egen in2005 = total(year(date) == 2005)
. by id: egen in2006 = total(year(date) == 2006)
. gen byte both = in2005 & in2006
. egen tag = tag(id)
. count if both & tag
```

It is characteristic of `egen` solutions that there is no fancy footwork with subscripts such as `_n` or `_N`. Typically, that is what the `egen` functions do on your behalf but out of sight.

There is a more instructive question. Why does the solution not telescope to

```
. by id, sort: egen both = total(year(date) == 2005 & year(date) == 2006)
. egen tag = tag(id)
. count if both & tag
```

as might be thought? `egen` would evaluate

```
year(date) == 2005 & year(date) == 2006
```

for each observation and give a total of the results for each `id`. However, for any observation it can never be true that the year is 2005 and also 2006. It could be one, or the other, or neither, but it cannot be both. Thus the count produced by `egen, total()` would always be zero with this argument. This problem is also not fixed by using the logical 'or' operator (`|`):

```
. by id, sort: egen either = total(year(date) == 2005 | year(date) == 2006)
```

This command counts visits in either 2005 or 2006, without regard to whether visits were in both 2005 and 2006. The result could easily be of interest, but it is an answer to a different question.

This last detail underlines once more the crucial distinction between what is true or false about individual observations and properties of groups of observations, panels in this example.

6 Other possibilities

Techniques for related problems should now be within sight.

Visits in 2005, but not 2006, and vice versa, or patients in the dataset who visited in neither year, could be identified by looking for the appropriate zero sums.

Problems that require knowing the numbers of visits call for looking directly at the sums.

Problems could easily be more complicated given other conditions that you might want to specify. You might want to know not only about visits in 2005 and 2006 but also particular kinds of outcome. Typically, you would just stipulate extra logical expressions. Such problems need not be much more difficult. It is just a matter of spelling out all the restrictions needed.

Problems involving three or more levels of structure also yield to extensions of the techniques here.

Naturally, many related problems do call for either a reduction to a smaller dataset with one observation for each panel or a tabulation, or both. If we wanted a breakdown of visits by patient and year, a year variable would indeed be needed, but a table might be the answer.

7 Conclusions

Counting panels, and more generally groups, is sometimes possible through a reduction command that produces a smaller dataset or through a tabulation command. Yet there are also many problems, especially with irregular sets of observations for varying times, that do not yield easily to this approach. This column has focused on techniques for answering such questions while maintaining the same data structure. Especially useful are `by:`, `egen`, and indicator variables constructed for the purpose. With `by:` we often exploit the fact that subscripts are defined within group, not within dataset. `egen` functions are often used to produce group-level statistics. Indicator variables containing 0s and 1s yield answers to many questions by simple logic and arithmetic. Tagging each group just once ensures that summaries, including counts, are of groups, not individual observations.

8 Acknowledgments

This column stems from various Statalist threads in October 2007. One solution draws on code posted by Kit Baum in response to a similar question.

9 References

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. Speaking Stata: Making it count. *Stata Journal* 7: 117–130.

———. 2007b. Stata tip 51: Events in intervals. *Stata Journal* 7: 440–443.

Kantor, D., and N. J. Cox. 2005. Depending on conditions: A tutorial on the `cond()` function. *Stata Journal* 5: 413–420.

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 wrote several inserts in the *Stata Technical Bulletin* and is an editor of the *Stata Journal*.