The Stata Journal (2008)
8, Number 4, pp. 588–591

Stata tip 71: The problem of split identity, or how to group dyads

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

Many researchers in various disciplines deal with dyadic data, including several who would not use that term. Consider couples in pairings or relationships of any kind, including spouses, lovers, trading partners, countries at war, teams or individuals in sporting encounters, twins, parents and children, owners and pets, firms in mergers or acquisitions, and so on. Dyads can be symmetric or asymmetric; benign, malign, or neutral; exclusive (one partner can be in only one dyad with their unique partner) or not. See Kenny, Kashy, and Cook (2006) for an introduction to the area from a social- and behavioral-science viewpoint.

Behind this intriguing variety lies a basic question: How can we handle dyad identifiers in Stata datasets? A natural data structure reflects the split identity of dyads: each dyad necessarily has two identifiers that researchers will usually read into two variables. This is indeed natural but also often poses a problem that we will need to fix. Suppose that Joanna and Jennifer are twins and that Billy Bob and Peggy Sue are twins. We might have observations looking like this:

```
. list person twin

     +-----------------------+
     |    person        twin |
     |-----------------------|
  1. |    Joanna    Jennifer |
  2. |  Jennifer      Joanna |
  3. | Billy Bob   Peggy Sue |
  4. | Peggy Sue   Billy Bob |
     +-----------------------+
```

And other variables would record data on each person. So the other variables for observation 1 could record the height, weight, number of children, etc., for Joanna, and those variables for observation 2 could record the same data for Jennifer.

In general, the identifiers need not be real names but could be any convenient string or numeric tags. Problems will arise if identifiers are not consistent across the two identifier variables. So with string identifiers, capitalization and other spelling must be identical, and all leading and trailing spaces should be trimmed. See Herrin and Poen (2008) for detailed advice on cleaning up string variables. Also, in general, there is no assumption so far that each person occurs just once in the dataset. Frequently, we will have multiple observations on each person in panel datasets, or we will have similar setups for other dyadic data.

dm0043

If your dataset contains many hundreds or thousands of observations, you need automated methods for handling identifiers. Editing by hand is clearly time consuming, tedious, and error prone.

How do we spell out to Stata that Joanna and Jennifer are a pairing? Here is a simple trick that leads readily to others. We can agree that Joanna and Jennifer have a joint identity, which is (alphabetically) Jennifer Joanna. So we just need to sort those identifiers by observation, or rowwise. This can be done as follows:

```
. generate first = cond(person < twin, person, twin)
. generate second = cond(person < twin, twin, person)
. list person twin first second

     +-----------------------------------------------+
     |    person        twin       first      second |
     |-----------------------------------------------|
  1. |    Joanna    Jennifer    Jennifer      Joanna |
  2. |  Jennifer      Joanna    Jennifer      Joanna |
  3. | Billy Bob   Peggy Sue   Billy Bob   Peggy Sue |
  4. | Peggy Sue   Billy Bob   Billy Bob   Peggy Sue |
     +-----------------------------------------------+
```

This is all breathtakingly simple and obvious once you see the trick. You need to see that inequalities can be resolved for string arguments as well as for numeric arguments, and equally that `cond()` will readily produce string results if instructed.

Let us go through step by step. The `person < twin` string for Stata means that the value of `person` is less than the value of `twin`. For strings, *less than* means *earlier in alphanumeric order*. The precise order is that of `sort`, or of `char()`, not that of your dictionary. So `"a"` is less than `"b"`, but `"B"` is less than `"a"` because all uppercase letters are earlier in alphanumeric order than all lowercase letters. This precise order should only bite you if your identifiers are inconsistent, contrary to advice already given.

You might wonder quite how broad-minded Stata is in this territory. Do the functions `min()` and `max()` show the same generosity? No; `min("a", "b")` fails as a type mismatch.

The `cond()` function assigns results according to the answer to a question. See Kantor and Cox (2005) for a detailed introduction. If `person < twin`, then `first` takes on the values of `person` and `second` takes on the values of `twin`. If that is not true, then it is the other way around. Either way, `first` and `second` end up alphanumerically sorted.

I find it helpful to check through all the logical possibilities with an inequality, if only to reassure myself quickly that every possibility will produce the result I want. An inequality based on `<` will not be true if the operands satisfy `>` or if they satisfy `==`. Here, if the names are the wrong way around, they get swapped in the results for `person` or `twin`, which is as intended. What is easier to overlook is the boundary case of equality. If the names are the same, they will also be swapped, but that makes no difference; no harm is done and no information is lost. In the example of twins, names being the same might seem unlikely, but perhaps someone just used surnames, and the surnames are identical. As usual, however, data-entry errors are another matter. In some other examples of dyads, there may be good reason for the names to be consistently the same; if so, the problem discussed here does not arise at all.

An advantage of using `cond()` is that exactly the same code applies to numeric identifiers. Conversely, if the identifiers were numeric, it would be fine to code

```
. generate first = min(person, twin)
. generate second = max(person, twin)
```

and a quick check shows that this works even if the identifiers are identical.

The problem is now all but solved. We can group observations for each dyad with

```
. by first second: command
```

and if we need a unique identifier for each dyad—it will come in useful sooner or later—we can get that by typing

```
. egen id = group(first second)
```

The `egen, group()` command yields identifiers that are integers 1 and above. Note also its handy `label` option. For more detail on such variables, see Cox (2007).

Tips for dyads should come in pairs, so here is another. This is for the case in which there are precisely two observations for each dyad. Again twins are a clear-cut example. Often we will want to compare each twin with the other, say, by calculating a difference. Then the height difference for each twin is *this twin's height* minus *the other twin's height*, or

```
. by first second: generate diffheight = height - height[3 - _n]
```

Or if we had calculated that identifier variable mentioned earlier, the height difference would be

```
. by id: generate diffheight = height - height[3 - _n]
```

Where does the `[3 - _n]` subscript come from? Recall that under the aegis of `by:`, `_n` is determined *within* groups defined by the *byvarlist*, here, the identifier variable `id`. For more on that, see Cox (2002) or, perhaps more conveniently, the *Speaking Stata* column in this issue (Cox and Longton 2008). So `_n` will be 1 or 2. If it is 1, then $3-1$ is 2, and if it is 2, then $3-2$ is 1.

If that seems too tricky to recall, there are more commonplace ways to do it:

```
. by id: generate diffheight =
> cond(_n == 1, height - height[2], height - height[1])
```

or even

```
. by id: generate diffheight = height - height[2] if _n == 1
. by id: replace diffheight = height - height[1] if _n == 2
```

References

Cox, N. J. 2002. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86–102.

———. 2007. Stata tip 52: Generating composite categorical variables. *Stata Journal* 7: 582–583.

Cox, N. J., and G. M. Longton. 2008. Speaking Stata: Distinct observations. *Stata Journal* 8: ???–??? (this issue).

Herrin, J., and E. Poen. 2008. Stata tip 64: Cleaning up user-entered string variables. *Stata Journal* 8: 444–445.

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

Kenny, D. A., D. A. Kashy, and W. L. Cook. 2006. *Dyadic Data Analysis*. New York: Guilford Press.