The Stata Journal

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 university library subscription	$125	1-year university library subscription	$165
2-year university library subscription	$215	2-year university library subscription	$295
3-year university library subscription	$315	3-year university library subscription	$435
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.

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.

The Stata Journal (2013)
13, Number 1, pp. 217–219

Stata tip 114: Expand paired dates to pairs of dates

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

Often dates for transactions or events come in pairs, for example, start and finish or open and close. People arrive for an appointment at a clinic and later leave; people order goods, which are later delivered; people start work for an employer and later leave. In all cases, when we take a snapshot, we may find people still in the clinic, goods not yet delivered, or (it is hoped) people still working for the employer.

With such events, it is natural that each pair of events is often recorded as an observation (case, row, or record) in a dataset. Such a structure makes some calculations easy. For example, the differences between times of arrivals and departures or of orders and deliveries are key to system performance, although the ideal will be short delays for seeing patients or selling goods and long delays for periods of employment or lifetimes. In Stata, with two variables named, say, `arrival` and `depart`, the delay is just

```
. generate lapse = depart - arrival
```

Precisely how we record time will depend on the problem, but here I am imagining a date or date–time or time variable.

If the closing events have yet to happen, then `depart` may need to be recorded as missing. If so, `lapse` will in turn be missing. Notice, by the way, a simple data quality check that the time lapse can never be negative. Time lapses recorded as zeros might also need checking in some situations: Was the system really that fast and efficient (or ruthless)?

Such a simple data structure—one observation for each time interval—may also be awkward, which leads to the main reason for this tip. Experience with data structures in Stata might lead readers to suggest a `reshape long`, which could be a good idea, but there is an easier alternative, to use `expand`.

We need first a unique or distinct identifier for each interval, which may already exist. The command `isid` allows a simple check of whether an identifier variable indeed matches its purpose. If the identifier variable is broken or nonexistent, then something like

```
. generate long id = _n
```

creates a new identifier fit for the purpose. Specifying a `long` variable type allows over two billion distinct positive identifiers, should they be needed. Otherwise, we use the existing identifier. Then

```
. expand 2
```


is the command needed. We turn each observation into two. The new observations are added at the end of the dataset, so we need to sort them before we can create two new variables that are the keys to other calculations. The first new variable combines the time information:

```
. by id, sort: generate time = cond(_n == 1, arrival, depart)
```

By virtue of the `expand` command, each distinct value of the identifier now occurs precisely twice. We can therefore use the framework provided by the `by:` prefix; see Cox (2002) for a tutorial if desired. Under `by:`, the observation number `_n` is interpreted within groups (here all pairs), and we assign time `arrival` to the first observation of two and time `depart` to the second.

Let's imagine a small section of a toy dataset and apply our expansion method.

```
. list
```

	id	arrival	depart
1.	1	1000	1100
2.	2	1100	1300
3.	3	1200	1400

```
. expand 2
(3 observations created)
. by id, sort: generate time = cond(_n == 1, arrival, depart)
```

The second new variable flags whether each time is an arrival or departure. The usual kind of indicator or dummy variable with values 1 or 0 would serve, but values of 1 and −1 are even better, given that each arrival is an addition and each departure a subtraction.

```
. by id: generate inout = cond(_n == 1, 1, -1)
```

Typing `(_n == 1) - (_n == 2)` is another way to code this.

Now all we need to do is sort and calculate the net results of all events:

```
. sort time
. list, separator(0)
```

	id	arrival	depart	time	inout
1.	1	1000	1100	1000	1
2.	1	1000	1100	1100	-1
3.	2	1100	1300	1100	1
4.	3	1200	1400	1200	1
5.	2	1100	1300	1300	-1
6.	3	1200	1400	1400	-1

Because the flag variable `inout` records additions and subtractions, so also its cumulative or running sum keeps track of the number inside the system. In a jargon common

in economics, flows are used to calculate stocks, the assumption being that any stock from before the start of records would need to be added.

```
. generate present = sum(inout)
. list, separator(0)
```

	id	arrival	depart	time	inout	present
1.	1	1000	1100	1000	1	1
2.	1	1000	1100	1100	-1	0
3.	2	1100	1300	1100	1	1
4.	3	1200	1400	1200	1	2
5.	2	1100	1300	1300	-1	1
6.	3	1200	1400	1400	-1	0

This is only one trick, and others will depend on your problem. For example, if a clinic is only open daily, the number present should drop to zero at the end of each day. More generally, stocks cannot be negative. The logic of how your system operates provides a logic for your code and checks on data quality. If that logic implies separate accounting for each panel of panel or longitudinal data, then that merely implies a different `sort` order and operations under the aegis of `by:` (Cox 2002).

For a different problem arising with paired data, see Cox (2008).

References

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

———. 2008. Stata tip 71: The problem of split identity, or how to group dyads. *Stata Journal* 8: 588–591.