The Stata Journal (2007)
7, Number 3, pp. 440–443

Stata tip 51: Events in intervals

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

Observations in panel or longitudinal datasets are often for irregularly spaced times. Patients may arrive for consultation or treatment, or sites may be visited in the field, at arbitrary times, or other human or natural events may occur with unpredictably uneven gaps. Geophysicists might record earthquakes or political scientists might record incidents of unrest; in either case, events occur with their own irregularity. Such examples could be multiplied. One way researchers seek structure in such data is by counting or summarizing data for each panel over chosen time windows. Usually we look backward: How many times did something happen in the previous 6 months? What was the average of some important variable over observations in the previous 30 days?

To see the precise problem in Stata terms, consider better behaved data in which we have regular observations, say, monthly or daily. Then we can use windows with fixed numbers of observations to calculate the summaries required. `rolling` ([TS] **rolling**) can be especially useful here. This scenario suggests one solution: a dataset with irregular data can be made regular by inserting observations for dates not present in the data. `tsfill` ([TS] **tsfill**) is the key command. In turn the downside of that solution is evident: the bulked-out dataset could be many times larger, even though it carries no extra information.

A more direct solution is possible, typically requiring a few lines of Stata code. Once you grasp the solution, modifying the code for similar problems is easy.

Suppose first that you want to count certain kinds of observations, say, how many times something happened in the previous 30 days. We assume that the data include an identifier (say, `id`) and a daily date (say, `date`) among other variables. A good technique to consider is using the `count` command (Cox 2007a). First, initialize a count variable. Our example will count observations with high blood pressure, so the variable name reflects that:

```
gen n_high_bp = .
```

The idea is to loop over the observations, looking at each one in turn. A basic count will be

```
count if some condition is true &
         observation is in the same panel as this one &
         time is within interval of interest relative to this one
```

The example above is part Stata code, part pseudocode. The parts in *slanted type* are pseudocode. `count` will produce a number in your Results window, but that is less important than `count`'s leaving the result in `r(N)`. We must grab that result before something else overwrites it or it just disappears. We can grab the result and use it:

```
replace n_high_bp = r(N) in this observation
```

We want to repeat this step for each observation. You may know that you can use `forvalues`, often abbreviated `forval`, for automating a loop easily (see Cox 2002 for a tutorial). Suppose that you have 4,567 observations. Then you can type

```
forval i = 1/4567 {
        count if conditions are all satisfied
        replace n_high_bp = r(N) in ‘i’
}
```

Naturally, your having 4,567 observations is unlikely. So, you could just substitute the correct number for 4,567, or you could think more generally. `_N` is the number of observations.

```
local N = _N
forval i = 1/‘N’ {
        count if conditions are all satisfied
        replace nhighbp = r(N) in ‘i’
}
```

`forval` is fussy in its syntax, so we cannot use `_N` directly. The `local` statement sets a local macro, `N`, to contain its value. Once that exists, we can use its contents by referring to `‘N’`. As you might guess, `i` and `‘i’` refer to another local macro, which the `forvalues` loop brings into being. Each time around the loop it takes on values between 1 and the number of observations.

There are three slots to fill in the pseudocode. Here are three examples to match:

```
some condition is true
         inrange(sys_bp, 120, .)
observation is in the same panel as this one
         id == id[‘i’]
time is within interval of interest relative to this one
         inrange(date[‘i’] - date, 1, 30)
```

Our examples use `inrange()` twice. In the first, we suppose that we are counting how often systolic blood pressure was 120 mm Hg or higher. The `inrange()` condition here has one special and useful feature: it excludes missing values. That is, for example, `inrange(., 120, .)` is 0 (false). I do not expect you to find this behavior intuitive, but it is a feature. In the second, the previous 30 days is specified. For more on `inrange()`, see Cox (2006).

Now we can put it all together.

```
gen n_high_bp = .
local N = _N
quietly forval i = 1/`N' {
      count if inrange(sys_bp, 120, .) & ///
               id == id[`i']           & ///
               inrange(date[`i'] - date, 1, 30)
      replace n_high_bp = r(N) in `i'
}
```

A new detail here is the `quietly` added to the loop to stop a long list of results from being shown. Doing so is not essential. Indeed, at a debugging stage, seeing a stream of output, and being able to check that the results are as desired, is useful and reassuring.

Experienced programmers usually reduce that by one line, starting like this:

```
gen n_high_bp = .
quietly forval i = 1/`= _N' {
```

The shortcut here is documented under the help for `macro`. We are evaluating an expression, here just `_N`, and using its result, all within the space of the command line.

Stata users sometimes want to do something like this:

```
quietly forval i = 1/`= _N' {
      count if inrange(sys_bp, 120, .) & ///
               id == id[`i']           & ///
               inrange(date[`i'] - date, 1, 30)
      gen n_high_bp = r(N) in `i'
}
```

That code will fail the second time around the loop. The first time around the loop, when `i` is 1, all will be fine. The new variable `n_high_bp` will be generated. `r(N)` will be put into `n_high_bp[1]`. All the other values of `n_high_bp` will be born as missing. However, the second time around the loop, when `i` is 2, the `generate` command is illegal, as the `n_high_bp` variable already exists, and you cannot `generate` it again.

The consequence is that within the loop we need to use `replace`. In turn, we need to initialize the variable outside and before the loop (because, conversely, you cannot `replace` something that does not yet exist). Initializing it to missing is good practice, even when we know that the program will overwrite the value in each observation.

There are some disadvantages to this approach. Mainly, it will be a bit slow, especially with large datasets. Having to spell out a few lines of code every time you do something similar could also prove tedious. That task could be an incentive to wrap up the code in a do-file or even a program.

More positively, the logic here should seem straightforward and transparent and fairly easy to modify for similar problems. The key will usually be to pick up whatever we need as a saved result. Suppose that we want to record the mean systolic blood pressure over measurements in the last 30 days. The main change is the use of the `summarize` command rather than the `count` command.

```
gen mean_sys_bp = .
quietly forval i = 1/`= _N' {
      summarize sys_bp if id == id[`i'] & ///
                         inrange(date[`i'] - date, 1, 30), meanonly
      replace mean_sys_bp = r(mean) in `i'
}
```

For the `meanonly` option of `summarize` and its advantages, see the previous Stata tip (Cox 2007b).

Naturally, there are occasional problems in which the condition that we are considering only observations in the same panel is inappropriate. For those problems, remove or change code like `id == id[`i']`.

Finally, the technique is readily adaptable to other kinds of windows, say, with regard to intervals of any predictor or controlling variable.

References

Cox, N. J. 2002. Speaking Stata: How to face lists with fortitude. *Stata Journal* 2: 202–222.

———. 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 50: Efficient use of summarize. *Stata Journal* 7: 438–439.

The Stata Journal (2007)
7, Number 3, p. 444

Software Updates

st0015_4: Concordance correlation coefficient and associated measures, tests, and graphs. T. J. Steichen and N. J. Cox. *Stata Journal* 6: 284; 5: 471; 4: 491; 2: 183–189. *Stata Technical Bulletin* 58: 9; 54: 25–26; 45: 21–23; 43: 35–39. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, p. 137; vol. 9, pp. 169–170; vol. 8, pp. 137–145.

A bug exposed by the advent of Stata 10 has been fixed. The help file has been updated to include a new reference.

up0019