The Stata Journal

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press. Stata and Mata are registered trademarks of StataCorp LP.

The Stata Journal (2007)
7, Number 2, pp. 266–267

Stata tip 44: Get a handle on your sample

Ben Jann
ETH Zürich
Zürich, Switzerland
jann@soz.gess.ethz.ch

Researchers producing careful and reproducible statistical analyses need to keep track of precisely which observations are used by commands. Consider `regress` and similar commands as leading examples. The observations used will depend on any `if` or `in` conditions, any weights specified, and the incidence of missing values. Typically, you will want to look at results for `regress` together with those from other commands. For that you want the same observations to be used. Even when there is comparison with results for different subsets, you also need to monitor which observations are used by which commands.

`if` and `in` conditions and the use of weights are explicit in your command syntax, so you have only yourself to blame if you fail to consider their consequences. Stata, however, does not make a great fuss about excluding missing values from your analyses, so more attention is needed to this detail. Since most substantial statistical datasets contain missing values in at least some of the variables, the issue can arise often.

Researchers commonly start with a simple model and then add more predictors or covariates. At each step, some observations may be excluded because values are missing in the extra variables. As long as the proportion of missing values is not too large, you may not care much about them. However, correct interpretation of the results hinges on the subset of observations used remaining identical.

A brute force approach to the problem is to `keep` only those observations being used (or conversely to `drop` the others). But this method can create as many problems as it solves. Any number of different subsets analyzed would mean as many different datasets and consequent awkwardness in setting up comparisons.

A much better way to get a grip on the samples being used is to construct binary indicator or dummy variables that mark the observations used in any analysis. Their values should be 0 for excluded observations and 1 for included observations. With such variables, corresponding `if` conditions may be specified as desired.

Ado-file programmers (see [U] **17 Ado-files**) face a similar problem and have a special command to solve it. If you look at the source code of ado-files (using [P] **viewsource**; Cox 2006), you will often find the command `marksample touse` near the start and many `if 'touse'` qualifiers after. Although `marksample` can be used only within programs, [P] **mark** documents two other "programmer's commands", `mark` and `markout`, that prove to be handy outside ado-files, as I will now show.

Suppose that you are analyzing a dataset containing the variables `x`, `y`, and `z`, all of which contain some missing values; a group variable, `g`; and analytic weights, `w`. The

analysis should be restricted to group `g == 1`. To ensure that the same observations are used throughout the analysis, type

```
. mark touse [aw=w] if g == 1
. markout touse x y z
```

at the beginning of your analysis and include `if touse` in all later commands, as in

```
. reg x y [aw=w] if touse
```

The first command, `mark`, generates a marker variable `touse` (read: "to use") that is set to 1 in observations satisfying the `if` qualifier and having a strictly positive, nonmissing weight and is set to 0 in all other observations.

The second command, `markout`, recodes `touse` to 0 if any of the specified variables contains missing. (If your data are `svyset`, you might want to omit the weights from the first command and add a third line reading `svymarkout touse`; see [SVY] **svymarkout**.)

There are other possible ways to generate the marker variable. You could, for example, use the `missing()` function (see [D] **functions**), or you could code

```
. quietly regress x y z [aw=w] if g==1
. generate byte touse = e(sample)
```

However, using `mark` and `markout` is simple and general. Often it is a good idea to `count if touse` and check that the number of observations used remains the same as that given.

If you need to have a one-liner, then define a program such as

```
program marktouse
version 8
syntax anything(id="markvar") [if] [in] [aw fw iw pw]
gettoken markvar varlist : anything
mark `markvar' `if' `in' [`weight'`exp']
markout `markvar' `varlist'
end
```

and use it as in

```
. marktouse touse x y z [pw=w] if g == 1
```

References

Cox, N. J. 2006. Stata tip 30: May the source be with you. *Stata Journal* 6: 149–150.