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.

For more information on the *Stata Journal*, including information for authors, see the web page

http://www.stata-journal.com

The *Stata Journal* is indexed and abstracted in the following:

- Science Citation Index Expanded (also known as SciSearch®)
- CompuMath Citation Index®

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 (2008)
8, Number 3, pp. 444–445

Stata tip 64: Cleaning up user-entered string variables

Jeph Herrin
Yale School of Medicine
Yale University
New Haven, CT
jeph.herrin@yale.edu

Eva Poen
School of Economics
University of Nottingham
Nottingham, UK
eva.poen@gmail.com

A common problem in data management, especially when using large databases that receive entries from many different people, is that the same name is given in several different forms. This problem can arise in many instances, for example, lists of names of schools, hospitals, drugs, companies, countries, and so forth. Variation can reflect several genuinely different forms of the same name as well as a multitude of small errors or idiosyncrasies in spelling, punctuation, spacing, and use of uppercase or lowercase.

Thus, in context, a person may have no difficulty in recognizing that values of a string variable, such as “New York”, “New York City”, “N Y C”, and so on, all mean the same thing. However, a program like Stata is necessarily literal and will treat them as distinct. How do we massage data so that values with the same meaning are represented in the same way? Several techniques exist for these purposes. Here we outline a simple strategy for ensuring that names are as tidy as possible.

As a preliminary stage, it is useful to try to eliminate small inconsistencies before you look at the individual observations. A good tactic is to keep the original names in one variable, exactly as given, and to work with one or more variables that contain cleaned-up versions. Some common problems are the following:

- Leading and trailing spaces may not be evident but will cause Stata to treat values as distinct. Thus “New York City” and “New York City ” are not considered equal by Stata until `trim()` is used to delete the trailing space.
- Similarly, inconsistencies in internal spacing can cause differences that Stata will register. The `itrim()` function will reduce multiple, consecutive internal blanks to single internal blanks.
- Variations of uppercase and lowercase can also be troublesome. The `upper()`, `lower()`, or `proper()` functions can be used to make names consistent.
- Other common differences include whether hyphens are present, whether accented characters appear with or without accents or in some other form, and whether ampersands are printed as characters or as equivalent words.
- A large class of problems concerns abbreviations.

In the last two cases, `subinstr()` is a useful function for making changes toward consistent conventions. Note a common element here: string functions, documented in [D] **functions**, are invaluable for cleaning up strings.

After a preliminary cleaning, you can create a list of all the names that you have. Usually, this list is shorter than the number of observations. It is worthwhile to inspect the list of names and look for further clean-up possibilities before proceeding. A tabulation of names, say, by using `tabulate, sort`, serves this purpose and provides you with the number of occurrences for each variation.

After the cleaning is completed, you are ready to compile your list of names. Suppose that the `name` variable contains the names.

```
. use mydatafile
. by name, sort: keep if _n == 1
. keep name
```

Now open the Data Editor by typing

```
. edit name
```

and add a second—numeric—variable, say, `code`. In this second variable, give the same number for every observation that represents the same object. This will be moderately time consuming, but because data are sorted on `name`, it may just take a few minutes.

Now exit the Data Editor, sort on `name`, and save the dataset:

```
. sort name
. save codes, replace
```

Some people may prefer to create the code in their favorite spreadsheet or text editor, say, if an outside expert not adept at Stata is recruited to do the coding. The principles are the same: you need to export the data to the other application and then read data back into Stata. You may lose out on an audit trail if the other software does not offer an equivalent to a Stata `.log` file.

Now you have a file (`codes.dta`) that has a list of the names in all their variety, as well as a set of numeric codes, which you can return to later to check your work. The key thing is that you can now `merge` this file into your original file to assign a common code to every value of `name` that is the same:

```
. use mydatafile, clear
. sort name
. merge name using codes
```

As always when using `merge`, examine the `_merge` variable; here, if `_merge` is not always equal to 3, then you have made a mistake somewhere. You should also examine `code`; if there are any missing values, you will need to `edit` the file `codes.dta` again to add them.

Now you can identify the objects by their codes; if you want, you can assign a common name:

```
. by code, sort: replace name = name[1]
```