The Stata Journal

The Stata Journal, electronic version (ISSN 1536-8734) is a publication of Stata Press, and Stata is a registered trademark of Stata Corporation.

The Stata Journal (2001)
1, Number 1, pp. 86–97

Speaking Stata: How to repeat yourself without going mad

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

Abstract. This column will focus on how to improve your fluency in Stata. Over the next issues we will look at Stata problems of intermediate size which turn out to be soluble with a few command lines. As an introduction, systematic ways of repeating the same or similar operations are surveyed to give one overview of the territory to be covered.

Keywords: pr0003, append, by, collapse, contract, do-files, egen, for, foreach, forvalues, log files, merge, naming conventions, programs, repetition, reshape, statsby, subset or group structure, tabulations

1 Introducing this column

This column will focus on Stata as a language. That may sound rather more technical than it is meant to be. Some years ago, at the first Stata User Group meeting held in London, I was expressing my enthusiasm for Stata to another user, and extolling the fact that, unlike certain other popular statistical software, Stata as a language was capable of very much more than was covered in the manuals. In a word, it is, by virtue of being a language, extensible. The reply was that to this user a computer language meant Fortran, C, C++ and the like, to be avoided at all possible cost whenever some package could do the job more painlessly. Calling Stata a "language" did not make it sound appealing!

That user had a fair point. I started computing with a course in Fortran, but have not written a program in that or any other mainstream language for a long time. For what I do, there is no need, and I shudder to think of the time I spent doing low-level stuff; reading in data and writing out results and having to take care of many details. But this column is not going to be about programming in Stata as the alternative. It will focus on increasing your fluency in Stata. The column will try to help users to think more easily in Stata's language, so that they get to where they want to be more quickly, and more effectively. Speaking of "fluency" signals a better comparison, not with learning programming languages, but with improving your skills in a foreign language that you are studying.

It has been said that when traveling in a country whose language you don't know, the words for "yes", "no", "water", "please", "thank you", "that", and as many numbers as possible will get you through about 90% of the situations you encounter. But anyone who picks up Stata's manuals senses the opposite: you are missing most of the power and flexibility of the software if you only know a few commands. Reading the manual,

or the FAQs on *http://www.stata.com*, or answers on Statalist may already have given you a good impression of how often fluent users can solve a tough-looking problem in a very few lines of Stata, and even with remarkably simple code. How do you pick up some of that skill?

2 Problems of different sizes

When you start up Stata, you will usually be tackling a series of questions. At the easiest, a question may be answered by a single Stata command. You must find out what that command is, understand as much of its syntax as you need, and use it to get some results. Naturally, there could be all kinds of small and large difficulties along the way, but most of the detail you need to know about is somewhere in the manual or in the on-line help. Or, at the other extreme, although each question could in principle be answered by a long series of Stata commands, a practical answer can only be produced if somebody—you, another user or a developer at Stata Corp—writes a Stata program. When that's done, all of a sudden a question very difficult, or at least tedious, to tackle step by step is reduced to an easy question.

The middle ground between these extremes is the territory that will concern us. Most of the questions we will use as examples will turn out to have a solution in a few lines of Stata. Many of those questions will turn out, in a broad sense, to be questions about data management. Although many users much of the time, and some users most of the time, are engaged in data management, it is for most Stata users typically a necessary evil that must be performed before the interesting and useful statistical analysis. We want to get it over with as soon as possible. But even with the statistical analysis there are tricks to be learned, ways of getting results more rapidly and more smoothly.

Many readers will know of Statalist as a forum for asking and answering Stata questions. My experience from watching Statalist and from interacting with Stata users all the way from fresh beginners to grizzled veterans is that the ideas to be covered in these columns should be of some use or interest to people at all levels. Those first encountering Stata will get an indication of what can be done using a variety of tools and techniques, especially if they are considering investing the time and effort needed to go further. Even experienced Stata users can become a little inefficient if they are not aware of recent additions to Stata, and we all know dimly of large chunks of Stata we have never used, and which just possibly may contain the solution we need.

3 Repetitive tasks

The next few issues will carry *Speaking Stata* columns looking at a series of worked examples in some detail, but what I want to do in this first column is to try to give you one overview of the terrain to be covered. What ties many examples together is the theme of repeating yourself, or more prosaically, of doing more or less the same thing over and over again.

Much statistically-based research is repetitive to some extent, partly as a consequence of data structure, partly because of the need to consider multiple aspects of a dataset, and partly because the same kinds of question recur with different datasets. The typical Stata dataset has at a minimum a structure of observations (e.g., people) by variables (e.g., their attributes). If, as often happens, there are bundles of related variables, then many analyses may be repeated for those variables. Common complications are variations in time (e.g., people by attributes by time) or variations within and between larger groups (e.g., people within firms by attributes).

In my own field of geography, even with a simple data structure of, say, rainfall measured at several rain gauges for several hundred days, analysts might be interested in aggregations to easily defined months, seasons or years, or to empirically defined spells. (A spell might be, for instance, a series of days in which it rained every day.) Even this relatively simple kind of dataset may need several iterations of analysis, yet at the same time the iterations may be both simple and similar.

Let's now run through some of the ways of repeating the same or similar operations, or of making those easier. We will start at the simplest, and move up. See how far your awareness stretches: marks out of 10?

3.1 Operations are repeated across observations for variables

It is worth emphasizing, especially for Stata users more used to some other software, such as spreadsheets, that many operations are automatically repeated across observations. Simple calculations such as `generate c = a + b` add the variables (columns) `a` and `b` and put the result in new variable (column) `c`; that is, automatically, `c[1]` = `a[1]` + `b[1]`, `c[2]` = `a[2]`+`b[2]`, and so on, for observations (rows) 1, 2, and so on. One jargon is that such calculations are vectorized. Of course, this is also possible in a spreadsheet. Actually, even a background in some language such as Fortran or C can mislead people new to Stata, as that background imprints the idea of an explicit `DO` or `for` loop as the control structure for repeating an operation over observations. Almost all problems for which beginners start trying to write Stata programs that loop over observations turn out to be solvable through normal Stata commands. If you find yourself doing this, think again, and ask!

3.2 Use Page Up/Page Down and clicking in Review window

The Page Up key goes back a command at a time in your command history and the Page Down key goes forward. Once a previous command is in the Command window, you may reissue it. (For the equivalent keys on your keyboard, you may need to consult [U] 13 in the manual.) Alternatively, previous commands may be selected by clicking in the Review window. Most Stata users internalize these habits early on in their experience, but if somehow you have missed these possibilities, or you are explaining Stata to somebody else, make a special note of such features. But be aware that a repeated command is carried out on the data currently in memory: if those data have changed since the previous execution, then naturally the results may differ.

3.3 Log files

For best use of Stata, keep a log file containing a transcript of what appears in each session in the Results window, namely a record of commands typed in the Command window and all (non-graphical) Results. The value of such detailed transcripts is difficult to overstate.

Users who started their Stata experience with earlier versions may not have grasped that the functionality of log files changed drastically with the release of Stata 7. First, it is now possible to keep simultaneously log files as just mentioned and command log files that contain only a transcript of commands issued. With some self-discipline to edit log files while each session is fresh in mind, and to remove mistakes, repetitions, and dead ends, each set of commands can then be converted into a do-file, to be discussed in more detail shortly.

To do this conveniently, some familiarity with a decent text editor helps greatly. Here "decent" implies that you can perform editing easily without being distracted by wanting functions that the editor lacks. The merits of different text editors give rise to occasional fervent discussions on Statalist, and in many other computing circles, but all experienced Stata users do seem to agree on the value of a text editor that you have gotten to know and like. Most people who have spent time using some flavor of Unix, including Linux, prefer some version of vi (now usually vim) or Emacs. Windows users can go beyond the somewhat limited Notepad or Wordpad editors bundled with the operating system to try any number of free or inexpensive editors, such as TextPad, NoteTab, PFE, or vim (again). The Macintosh, as usual, is a special case: BBEdit has many admirers. But let's not ignore that nearest to hand. Stata's own do-file editor `doedit` is fine for many tasks and it has the major advantage that it is well integrated with the rest of Stata. Don't let the name distract you: there is no reason why you should not use it for, say, editing little datasets, or little text files unrelated to Stata. I find it useful in copying and pasting some datasets from email, which I then convert to Stata-readable files. (Naturally, Stata's own data editor can also be invaluable for this.)

I am digressing here, but I do want to underscore the value of a good text editor to Stata users. Depending on your age and career, you may have spent much more time editing documents with a word processor, but for working with text files containing data or Stata commands, a text editor is the better tool. Finding an editor with a congenial style is one of the best things you can do to ease your computing sessions with Stata.

The second change in Stata 7 concerning log files is the new mark-up language SMCL, and logs are now by default in SMCL, although simple unadorned text is easily specified. A detail not documented elsewhere, but which may be of interest, is that log files in SMCL can be converted into HTML: the form of the command is

```
. log html filename.smcl filename.html
```

Incidentally, a common question is why Stata did not in fact use HTML or some other existing mark-up language for logs. The heart of the problem is that Stata needed a language in which logs can be produced a line at a time without foreknowledge of what

is to follow, let alone of the end of a document. That rules out HTML, which is based on the idea that a document file is processed as an integrated whole.

One way in which a log file is frequently invaluable is that it can be, in effect, a first stab at a routine for analyzing a particular kind of dataset. I often work step by step through the management or analysis of a new kind of dataset: once it becomes clear what the routine tasks are, it is then easy to edit a command log file so that the same commands can be repeated. Even if the analysis is never going to be repeated on a different dataset, one excellent practice is to remove all garbage from a command log file, preferably at the end of a Stata session, and then to run the analysis all over again from the log file. This does require self-discipline, but a strong incentive for doing it is that a messy log file will only make later rereading harder than is necessary. A messy log will certainly be more difficult for anyone else to follow.

3.4 Do-files

How is that possible? How can log files be run as a series of commands? The answer is that they can be treated as a do-file. To Stata, a do-file is just a text file containing a series of Stata commands. Conventionally, they are given the extension `.do`, but that is not essential. In Stata, they are then run quietly by `run` *filename*`.do` and noisily by `do` *filename*`.do`. You may omit the extension `.do`. If the file were, say, *filename*`.log`, then `do` *filename*`.log` will work fine, but now you must specify the extension.

So, very simply, instead of typing a whole series of commands again—which is likely to be tedious and error prone—repeating that series is reduced to a single action. One of the most valuable steps a Stata user can take is getting in the habit of putting commands in do-files. The habit is best learned by putting little sequences in little do-files, and eventually it becomes clear that longer sequences can be stored in exactly the same way. Several very experienced Stata users make extensive use of quite long do-files (hundreds of lines long). For substantial data files, they have identified all kinds of checking, reporting and manipulative commands for detailed, yet also routine, analyses.

As a simple example, consider a basic task of converting Fahrenheit temperatures to Celsius and inches of rainfall to mm. Two statements, one on each line, could make up the file `convert.do`:

```
. generate temp_C = 5 * (temp - 32) / 9
. generate rain_mm = rain * 25.4
```

Then the whole is executed by

```
. do convert.do
```

At first sight, the gain seems slight, but very often the do-file grows as one learns to add extra checking and calculation statements, e.g., counting implausible or missing values, checking for reasonable day-to-day changes, and so on. Also, it may seem that this do-file can only be applied to datasets in which the same variable names are used. However, the do-file may be generalized to accept arguments (that is, names or values that are fed to it). So

```
. generate temp_C = 5 * (`1' - 32) / 9
. generate rain_mm = `2' * 25.4
```

is executed by

```
. do convert.do temp rain
```

or

```
. do convert.do Temp Rain
```

or whatever. Here `‘1’` and `‘2’` are simply the first and second arguments fed to the do-file after it is named.

Let's take that more slowly. You direct Stata to the file `convert.do` and the command `do` says: execute the commands in the file specified, and show me the output. In so doing, you also specify `Temp` and `Rain`. Looking inside the `.do` file, you will see that there are two placeholders, `‘1’` and `‘2’`. The mapping follows your order: whatever you name first is treated as `‘1’` and whatever you name second is treated as `‘2’`. So, there is some generality imparted here by the one step towards abstraction, the use of placeholders that can be filled by whatever you supply to the file. Technically, `‘1’` and `‘2’` are so-called local macros, the "local" meaning that they should be thought of as existing inside the do-file, and not at all outside it. If you happen to be dealing with `‘1’` in some other do-file, or in a program, or while interacting with Stata directly, that is just a different beast. The meaning is local, as said, just as "Pat" in one circle will be one person and "Pat" in another circle another person. In Stata's case, however, strict control of namespaces means that there is never a problem of ambiguity, as may sometimes arise with people's names, to our amusement or bemusement.

That one step of abstraction is a step towards Stata programming in a stronger sense. What is lacking includes, in particular, any kind of error checking: that two and only two arguments are provided, that they are the right kind of things for the program to make sense, and so forth. In this example, suppose that you forgot that two arguments were needed, and only specified one. Stata would have no problems with the first statement, but would stop with an error when asked to evaluate `* 25.4`, which is not a legal Stata expression. (Why is it asked to evaluate this? Because if Stata does not know what you want to be the second argument, `‘2’` is treated as blank space; that is, it is ignored.) So, and this happens more widely, what would happen is usually that the do-file would stop part of the way through, and some of your work would be done, but not all, leaving behind a bit of a mess. It is possible to go beyond this and build in much more error checking, and that is much of the business of programs, in the strict sense.

3.5 Programs

What then, strictly, defines a Stata program? Logicians and language mavens should enjoy the answer: a program is defined as such by the `program define` command. In

early versions of Stata, there were very strict divides between typing in commands one by one, running a do-file, and running a program. More recent versions have made this range a smoother transition, so that, for example, several commands that control looping (historically, `while` and then `for`, and now also `foreach` and `forvalues`) can also be used, not only in do-files, but also interactively. This shift puts more power even in the hands of the Stata user who lacks the time or the inclination to start writing Stata programs.

As said earlier, this column is not going to focus much on programming, at least not directly. But right now you may be interested in the possibility of moving further towards writing your own Stata programs. Many are actually very simple indeed, although some are hundreds of lines long and call their own subroutines, not that size is a measure of significance. There is no special mystery here, merely some complications that need to be considered. For example, most commands in Stata allow the user to specify a subset of observations to be used by means of an `if` condition. Hence, this is often also expected behavior in a user-written program, and certain syntax has to be specified to permit it. On the other hand, many people have used Stata almost daily for years without ever writing a full-blown Stata program. Indeed, quite apart from the extensive set of commands within official Stata, many hundreds of Stata programs are now available outside it, so that it is usually worth trying first to find an existing program. Use `findit` (see Allen McDowell's column in this issue) or, if that yields nothing, try posting a question on Statalist.

If you are curious, note that `which` *command* will reveal the file containing the program whenever *command* is defined by a so-called ado-file file, ado meaning automatic do-file. A text editor will then allow scrutiny of the code, which may be instructive. (Sometimes commands are part of the executable program and their source code, in C, is then not accessible to users.)

3.6 Subset or group structure

Many problems in data analysis involve calculations done separately for each of several subsets (groups, more loosely). Note that a group here may be defined for a combination of conditions (e.g., sex is male *and* patient died), which fortunately is not really more difficult to handle. A slow but sure way of doing these separate calculations may be to repeat a command with a series of conditions, say, `if group == 1`, `if group == 2`, etc. As soon as you find yourself doing this again and again, stop! There are several ways of repeating for distinct subsets, which are quite easy to learn and which avoid such repetition. We will look at several in turn.

collapse, contract, statsby

Sometimes what is desired is a condensation of the data into summary statistics, one for each group. The twin commands `collapse` and `contract` are often helpful here. In essence, `collapse` is more general, allowing a variety of summary statistics to be

produced, while `contract` is specialized to counting categories or combinations of categories. Note that both involve destruction of the original data, which therefore should be `save`d first. To put this another way, instead of somehow looping over groups, we just request that Stata produces the group summaries all at once.

The newer command `statsby` helps with similar tasks, especially for producing a set of regression (or regression-like) results.

Tabulations

From that it is one step towards the production of tables of summary statistics (including counts), which leave the original data intact. Here it is easy even for experts to get confused because there are now several tabulation commands with overlapping functions in official Stata (and outside there is, for example, the `tab_chi` package with yet more: typing `findit tab_chi` will pinpoint locations). First, there is `tabulate`, which has long been in Stata, and which is perhaps best at contingency tables (e.g., sex by school-leaving age, with counts, percents, chi-square statistics, etc.). Second, more recently `table` and `tabstat` have been added to official Stata. On the whole, `tabstat` would now often be my first port of call. The key point is that instead of a series of, for example, `summarize` statements, it may be easier to go straight to, say, `tabstat`, which will calculate the summary statistics separately and put them together for you in a more presentable and more customizable form. From that it is then easy to copy the table to some report document that you are preparing.

egen

`egen` stands for `e`xtended `gen`erate and is the home for a library of programs, many of which allow a `by()` option, or prefixing by `by` *varlist*`:`, essentially the same thing in this case. The whole idea of `by` will be picked up again in a moment. The reason `egen` is distinct from `generate` is purely syntactic. `generate` can work with complicated expressions making use of variables, constants and built-in functions of various kinds. `log()` and `sqrt()` are examples of the latter. `egen` has by comparison a simpler and more restricted syntax, but also the great advantage that it can run what are in effect, although not in name, user-written functions. (As a small matter of history, most of the functions bundled with `egen` in official Stata were originally written by users outside StataCorp.) Thus, for example,

```
. egen mean = mean(varname1), by(varname2)
```

calculates the mean of *varname1* separately for each group of *varname2*. A glance at the data editor, or a `list`ing of the data, will show that necessarily each mean is repeated for each observation within a group. On the day that I wrote this, a questioner on Statalist wrote of a do-file containing 500 commands, two for each of 250 item codes. It appears that his question could be largely solved by just two `egen` commands in which the `by()` option is used to iterate across different codes.

More generally, once these means are included in a variable, then they can be tabulated, graphed, saved as a separate dataset, and so forth. Similar applications are possible with other `egen` functions. The on-line help for `egen` will reveal dozens of functions and dozens more can be found on the Internet.

3.7 by varlist:

The `by` construct is in many ways extremely simple, yet with some extremely clever touches. It does seem that many experienced users find it difficult to wrap their heads around it, as I did for some years until enlightenment dawned. One of the distinctive details (and I know of no precise analogues outside Stata) is that `_n` and `_N` are reinterpreted to apply within each distinct group. Recall that `_n` is the observation number, which runs from `1` to `_N`, reflecting the current sort order of the data. The FAQs on data management on the StataCorp web site give several detailed examples of how this can be exploited.

Here is one example. Often there is concern about duplicate observations whenever it is thought that some records may have been entered twice or more. Given a list of variable names (the Stata jargon is *varlist*),

```
. bysort varlist: list varlist if _N > 1
```

lists all observations that are duplicates (pedantically, replicates!) with respect to *varlist*. With many other programs, the solution is much more complicated.

Again, let's look more slowly at how that works. When there is no `by` in sight, `_N` is the number of observations in the whole dataset. One way of seeing this is just to type `display _N` in the Command window. But under the aegis of `by` *varlist*`:`, `_N` changes its meaning to the number of observations in each group defined by *varlist*. In each such group, all observations have identical values for all the variables in *varlist*. The `bysort` command in fact does two things in one: it `sort`s the observations according to their values on *varlist*, and then executes the command following `by` *varlist*`:`, meaning, precisely once for each group of *varlist*. (If you are still on Stata 6, or an earlier version, you will need to `sort` first and then do things `by` *varlist*`:`.) If a group occurs once and once only, then `_N` is 1, while if there are two or more copies, then `_N` is greater than 1, and values are thus listed if there are replicates.

Now, in fact, this is not quite a practical way of looking for duplicates, as Stata does not maintain perfect silence over the singletons, and typically there is too much output, especially for large datasets. Not surprisingly, problems with duplicates are sufficiently common and important that several user-written commands have been written to detect and (if necessary) drop such observations. For one such effort, see Steichen and Cox (1998). Indeed, there has been, almost inevitably, a certain duplication of effort on this problem. But for our purpose, the example shows vividly how the `by` construct yields a remarkably simple and direct solution.

3.8 for, foreach, and forvalues

These commands look a little tricky at first, but grow on users with experience. It is a pity that they are often labeled—especially by Stata Corp!—just as programming commands, when in fact all can be used interactively.

To understand the basic idea, you need to struggle a little with Stata jargon for various kinds of lists, as each of these commands boils down to cycling through a list and doing something for each member of a list. A *varlist*, as mentioned already, is a list of variable names, which can just be spelled out one after another (clicking on variable names in the Variables window is often convenient). In addition, a *varlist* may be expressed more concisely by hyphenation. Thus, the *varlist* `first-last` is all variables between `first` and `last`: the order of variables in the dataset may be seen in the Variables window or through `describe`. Also, a *varlist* may involve wildcards: `a*` consists of all variables beginning with `a`, including `a` if it exists; `*` is all variables. Less greedily, a single question-mark stands for any single character in this position: `weight?` would include `weight1` and `weight2`, but not `weight`.

A *numlist* is just a list of numbers, which again may be spelled out one by one: `1 2 3 4 5` or `2 3 5 7 11` qualifies, as does `11 2 7 5 3`. More commonly, however, some precise notation is used to specify regular sequences: `1/5` is `1 2 3 4 5`, while `10(10)50` is `10 20 30 40 50`.

There are other kinds of lists in Stata, but these will do fine to get the ball rolling.

Suppose we want to take logarithms of a set of variables. Once more we could do this one at a time, one variable in each command, but it may be preferable to use (say) `for`,

```
. for var length width height : gen logX = log(X)
```

which can be construed as follows: `for` the *varlist* `length width height`, take `log( )` of each (think of `X` as a placeholder) and use that to `generate` new variables, generically called `log`*X*, but in practice called `loglength`, `logwidth`, `logheight`. Here the symbol `X` just happens to be the default symbol, but it is not obligatory.

A second example is calculating powers of variables, say, the 2nd through 5th powers:

```
. for num 2/5 : gen xpowX = x^X
```

`foreach` and `forvalues` are more recently added relatives of `for`. At their best they allow small programs to be written on the fly, but to achieve that may take a little effort to master the syntax. Here are two translations of the last two examples:

```
. foreach v in length width height { gen log`v' = log(`v') }

. forval i = 2/5 { gen xpow`i' = x^`i' }
```

One syntax is just as about as complicated as the other. More generally, much could be said on the detailed advantages and disadvantages of these constructs, but a few broad comments must suffice for now. If you are used to `for` from previous versions,

and find it useful, stick with it, but otherwise, check out `foreach` and `forvalues`. For one thing, they are closer in syntax to much of the rest of Stata, which eases any learning of Stata programming. And for another, they are much easier to extend to lengthier programs written on the fly: `for` can be difficult to handle in some such cases, and it is easier to come to grief.

I also find that `for` and its siblings are very useful in non-Stata tasks. As a user of Windows (a matter of institutional choice), I could try doing many manipulations involving several files by repeated clicking, or I could loop over those files within Stata using `for`. You can even `copy` files across the Internet in this way, so long as you know what they are called, and that cuts out a lot of mousing around in your browser.

3.9 Systematic naming conventions

A point tacit here, and important enough to deserve flagging, is the value of systematic naming conventions. There is much benefit in variables that have something in common having some element of their names in common. This may sound obvious, but people are sometimes casual about names. The point applies more generally, as in selecting names for graph files. With commands like `for` it is easier to loop over a list if there is some pattern to the naming. A simple example is that `pop60 pop70 pop80` is easier to handle than `pop1960 pop70 popn80`. One way such inconsistencies may arise is that datasets may be formed by putting together the efforts of one person at many times, or of several people: in fact, discrepancies of naming are quite likely to occur even with careful workers. Such problems can easily be managed with a renaming tool such as `renvars`; see Cox and Weesie (2001).

3.10 append, merge, reshape

There are many commands for more-or-less drastic restructuring of datasets. The `append` command joins datasets by adding observations and `merge` joins them by adding variables. `reshape` allows moving back and forth between long and wide structures. With panel or longitudinal data, in particular, is time represented by separate columns, or by separate observations for each person in the panel? Stata has a marked preference for the second (long) structure over the first (wide) structure. In any case, with long structures it is easier to handle, for example, different times of observations, or different numbers of observations, for different members of the panel.

A moment's reflection points up a simple but fundamental fact. Stata's general data model is asymmetric: operations for variables (along, or more precisely, down columns) are one thing, and operations across variables (across rows) are another, and they are not always equally easy, even when either is appropriate. (Tried taking a row median?) We could debate this at length, but from, say, the point of view of array-based languages such as APL and its successors, such asymmetry could be cited as a design limitation to Stata. Be that as it may, Stata includes many commands that help you get from the data structure you have to the data structure you need to answer your question.

So, for example, joining or reshaping datasets may then allow some other device to be applied. In particular, it is often much easier to work on different groups within one large dataset than on separate smaller datasets. Naturally, much depends on what defines a dataset, and on what you are doing, but many comparative analyses depend on the data being together. In contrast, it is all too common with some software that the file structure and data structure initially chosen may channel all further analyses, as it becomes too awkward or even impracticable for the user to overcome that starting point.

4 What's next?

That's been, quite deliberately, a whistle-stop tour of much of the territory that this column will explore. The commands, devices, habits, tricks, tactics and strategies that make problem solving easier for the Stata user will be our concern. Over the next few columns, I will be looking at more substantial examples and dissecting how they are best tackled in Stata terms.

5 References

Cox, N. J. and J. Weesie. 2001. dm88: Renaming variables, multiply and systematically. *Stata Technical Bulletin* 60: 4–6. In *Stata Technical Bulletin Reprints*, vol. 10, 41–44. College Station, TX: Stata Press.

Steichen, T. J. and N. J. Cox. 1998. dm53: Detection and deletion of duplicate observations. *Stata Technical Bulletin* 41: 2–4. In *Stata Technical Bulletin Reprints*, vol. 7, 52–55. College Station, TX: Stata Press.

About the Author

Nicholas J. Cox is a statistically-minded geographer at the University of Durham. He contributes talks, postings, FAQs, and programs to the Stata user community. He has also coauthored seven commands in official Stata. He was an author of several inserts in the *Stata Technical Bulletin* and is the Executive Editor of *The Stata Journal*.