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 webpage

http://www.stata-journal.com

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

- CompuMath Citation Index®
- Current Contents/Social and Behavioral Sciences®
- RePEc: Research Papers in Economics
- Science Citation Index Expanded (also known as SciSearch®)
- Scopus™
- Social Sciences Citation Index®

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

The Stata Journal (2012)
12, Number 1, pp. 147–158

Speaking Stata: Output to order

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

Abstract. Wanting to present Stata output in a different way is a very common desire that lies behind many substantial user-written programs. Here I start at the beginning with some basic tips and tricks. I discuss using `display` to replay the results you want; putting results into new variables so that they can be listed, tabulated, or plotted; and using existing reduction commands to produce new datasets containing results.

Keywords: pr0053, collapse, contract, display, egen, estimates, foreach, format, forvalues, generate, list, postfile, quietly, replace, saved results, SMCL, statsby, tabdisp

1 Introduction

Stata users often want their results presented in a form that differs from Stata's own output. Many research reports depend on a complicated collation of results from several different commands, models, variables, or datasets. Even at a simpler level, you may need to rearrange a set of results in a different layout, or with different text headings, or with different rounding of numbers. The last in particular is a common need. Like most mathematical and statistical software, Stata typically gives you far more precision than you really need. Spelled out, the argument is elementary: Stata cannot discern exactly what that precision is, and in any case, it is easier to discard detail you do not want than to re-create detail no longer visible.

Sometimes this reordering can all be done within Stata, while at other times the major concern is preparing output in suitable form for other software, ranging from word and text processors to spreadsheets, databases, and browsers. Many of the most popular user-written commands offer tools within this area. Indeed, some of those commands provide highly versatile and well-developed toolkits for reordering your results. Citing some but not others could seem invidious, so instead I invite readers to look through past issues of the *Stata Journal* or to rummage in the Statalist archives for a short while to find out what is available.

In contrast, I concentrate in this column on some of the simpler tips and tricks of a kind that can be very easily used interactively, or in do-files or in simple programs. I am drawing on my experience following Statalist traffic over several years, which has gained me many impressions of what people want to do and what they often do not know about. You may want to look ahead to the concluding section, where all the specific tips are gathered together, to get a more detailed overview of what is explained.

2 Divide, conquer, and display

For a first example, suppose that we wish to show results for several variables or several groups of observations for skewness and kurtosis. This example is chosen partly out of personal interest (Cox 2010a), but more importantly because it is simple but not trivial.

As you probably know, the command `summarize, detail` reports skewness and kurtosis together with several other statistics. So in particular, you can type

```
. sysuse auto
(1978 Automobile Data)
. summarize mpg, detail
                        Mileage (mpg)
-------------------------------------------------------------
      Percentiles      Smallest
 1%           12             12
 5%           14             12
10%           14             14       Obs                  74
25%           18             14       Sum of Wgt.          74
50%           20                      Mean            21.2973
                        Largest       Std. Dev.       5.785503
75%           25             34
90%           29             35       Variance        33.47205
95%           34             35       Skewness        .9487176
99%           41             41       Kurtosis        3.975005
```

However, it is very likely that you do not want all the other results of `summarize`. It is also likely that you do not want all the detail shown (in this example, 7-digit output for skewness and kurtosis). If you were comparing Stata's results with those of some other program, then you might indeed want to keep as much detail as possible, but we will suppose otherwise.

If you are very new to all this, you might guess that you need to copy out results into a different file, say, by copying and pasting from the Results window or a text (log) file. Fortunately, Stata offers better routes to more congenial output.

Tip 1: Stata typically holds major results in memory, at least immediately after the command has run. You can thus access these for presentation in a different form.

The key distinction here is that most of Stata's commands that produce numeric output are either e-class or r-class, and so saved results can be seen immediately after the command in question by typing either `ereturn list` or `return list`. In practice, users quickly learn to abbreviate these to `eret li` and `ret li`.

Broadly, e-class commands (think that "e" stands for estimation) are those that statistical people would think of as fitting a model, while r-class commands are the other commands that produce statistical output, usually descriptive statistics of some kind. `summarize` is an r-class command and so you should type

```
. ret li
scalars:
                  r(N) =  74
              r(sum_w) =  74
               r(mean) =  21.2972972972973
                r(Var) =  33.47204738985561
                 r(sd) =  5.785503209735141
           r(skewness) =  .9487175964588155
           r(kurtosis) =  3.97500459645325
                r(sum) =  1576
                r(min) =  12
                r(max) =  41
                 r(p1) =  12
                 r(p5) =  14
                r(p10) =  14
                r(p25) =  18
                r(p50) =  20
                r(p75) =  25
                r(p90) =  29
                r(p95) =  34
                r(p99) =  41
```

Usually, as here, the names of the results are clear enough, at least when matched with the previous output. We see that `r(skewness)` and `r(kurtosis)` hold the results we want. It is also reassuring to see even more decimal places in evidence here for results with fractional parts, underlining that Stata is holding results to a high degree of numeric precision.

It is not necessary in our running example, but I will underline here that such results also are the ingredients for many further calculations, even of results not shown by the command in question. `summarize` does not show the coefficient of variation, which is standard deviation as a fraction or percentage of mean. Nor does it show measures of skewness based on median and quartiles, or on mean, median, and standard deviation. However, such extra measures are all easy to calculate.

If you are not sure what a particular r-class or e-class result means, or even whether a particular command is e-class or r-class, it is easy enough to find out. The manual entry or other documentation should explain in more detail, or in the latter case, typing both `return list` and `ereturn list` will elicit the flavor of the command.

Tip 2: The `display` *command with specified format is convenient for presenting numeric results as you wish.*

The first step toward doing something different with Stata's results is just to replay them using `display`. Users of `display` quickly learn its abbreviation, `di`.

```
. display r(skewness)
.9487176
```

Note that just 7 decimal places are displayed here, whereas the `return list` results show that Stata is holding more or, more precisely, is holding more detail as a binary number that translates into more decimal places. So what happened to the other decimal places? The answer is that `display`, like `summarize`, has its own default for how much precision it will show if not otherwise instructed.

If you want a different format for `display`, you merely have to ask. The help for `format` gives the small details, but my own decisions usually center on how many decimal places I want to show after the decimal point. For skewness and kurtosis, 3 decimal places seems more than enough, which makes me reach for a format such as `%4.3f`.

```
. display %4.3f r(skewness)
0.949
```

Stata's formats are pretty smart and will often give you more space than you ask for, as in

```
. display %4.3f 1000/3
333.333
```

but you may need to be less cavalier in displaying several results on the same line, at least if you want results nicely aligned, as you will.

Tip 3: To show a set of results for different variables or groups, loop over the possibilities with `foreach` *or* `forvalues`*. Typically, you will need to explain each piece of output with appropriate text.*

Tip 4: To suppress output you do not want, prefix a command with `quietly`*.*

*Tip 5: Stata Markup and Control Language (*SMCL*) offers extra control over presentation with* `display`*.*

Now let us suppose that we want to loop over the numeric variables in the dataset to get the skewness and kurtosis of each. We could issue a `summarize` command, and the separate skewness and kurtosis results would all be displayed, but that won't help us here, because afterward only the r-class results for the last variable analyzed will be accessible. So we need to loop: for each variable, we issue a `summarize` command but then immediately pick up the results we want and put them where we want before they are overwritten.

Stata provides tools that are exactly suited for this need: `forvalues` and, more usually in practice, `foreach`. A detailed tutorial on both was given in an earlier Speaking Stata column (Cox 2002), so I will not repeat an explanation here.

```
. foreach v of var price-foreign {
  2.         quietly summarize `v´, detail
  3.         display "`v´{col 16}"  %10.3f r(skewness)  %10.3f r(kurtosis)
  4. }
price               1.653     4.819
mpg                 0.949     3.975
rep78              -0.057     2.678
headroom            0.141     2.208
trunk               0.029     2.192
weight              0.148     2.118
length             -0.041     2.042
turn                0.124     2.229
displacement        0.592     2.376
gear_ratio          0.219     2.102
foreign             0.887     1.787
```

The `foreach` loop takes the variable list `price-foreign` (conveniently, all the numeric variables in `auto.dta`) and then for each variable does two things.

First, we fire up `summarize, detail` for that variable, but we do so `quietly`. That suppresses the display that `summarize` would give, which is fine because of what we do next.

Second, we will pick up results for skewness and kurtosis. Note the extra detail that the SMCL annotation {`col 16`} shifts the output to column 16. The number 16 was based on a little calculation that is easy in this small dataset: it is evident that the longest variable name, `displacement`, is just 12 characters long, so we will want to accommodate that plus a little space.

For more on SMCL, see `help smcl`, except that usually you will want to consult that documentation as a reference only when you need it.

The formats for skewness and kurtosis are more generous (`%10.3f`) than what we used earlier (`%4.3f`), but as can be seen, one important side-effect of a format more generous than needed is to give extra spaces. Even so, in practice many people might still regard the display just given as rather compressed and so would be tempted by a format like `%12.3f`. The key point is simply that you are in charge and can make your own decision.

The output just given is clearly rather unpolished in some respects. For example, it lacks explanatory column headers and is tied to the particular fact that variable names in this dataset are all no more than 12 characters long. However, it could still be fine for many purposes. `display` should also now be evident as a command that could be used to produce header lines before the main body of the table is shown. Rather than refining this approach further, we will now show a different approach.

3 From data variables to results variables

Tip 6: Use `generate` *and* `replace` *to put results from a command one by one into new variables.*

If we go back to the first variable we used with `summarize`, we could instead place the results in new variables with, say,

```
. generate skewness = r(skewness) in 1
(73 missing values generated)
. generate kurtosis = r(kurtosis) in 1
(73 missing values generated)
```

That is what you might do, but it is not a good idea.

To explain the syntax, however: "`in 1`" flags that we would be putting the results in the first observation only. However, typically we will want to do something like this in a loop. Although a `generate` statement will work fine the first time around the loop, the loop will fail the second time around because `generate` will complain that the variable already exists.

The solution is simple. You first `generate` the variable outside the loop, doing that just once, and change the command within the loop to `replace`. `replace` will not complain about changing what already exists; that is its sole purpose.

While we are exploring this idea, we will see that we can also make the problem more ambitious with very little extra effort. Let's say that we also want to see number of observations, mean, and standard deviation as minimal context for assessing skewness and kurtosis. We also want to record explanatory text for each variable, here variable name and label.

```
. quietly foreach v in n mean sd skewness kurtosis {
.       generate double `v´ = .
. }
. quietly foreach v in varname varlabel {
.       generate `v´ = ""
. }
. local i = 1
. quietly foreach v of var price-foreign {
.       summarize `v´, detail
.       replace varname = "`v´" in `i´
.       replace varlabel = "`: var label `v´´" in `i´
.       replace n = r(N) in `i´
.       replace mean = r(mean) in `i´
.       replace sd = r(sd) in `i´
.       replace skewness = r(skewness) in `i´
.       replace kurtosis = r(kurtosis) in `i´
.       local ++i
. }
```

In essence, there are two phases to this operation. First, we initialize variables to missing; the loops are over the result variables. Here we separate initialization of the numeric and string results. The numeric result variables are initialized to numeric missing `.` and the string result variables are initialized to the empty string `""`. In most problems, specifying `double` for numeric results will cost us little in memory but may save some later embarrassment or frustration in holding values precisely that may be

very large or very small. (People very interested in p-values or their kin will care about that.)

Second, in the last block of code just given, we put the results one by one in the result variables; the loop is over the data variables. There are new small details here:

1. Before the loop, we initialize a counter (the local macro `i`) to 1. Each time around the loop, we add 1 to the counter by using `local ++i`. We are thus looping over both the variables and the counter 1, 2, 3, ...

   If you would like more discussion of looping in general, see Cox (2002), or of looping in parallel, see Cox (2003a).

2. It is easy but often forgotten to put `quietly` on the loop as a whole rather than on each individual statement that would otherwise produce output. That saves typing, produces less cluttered code, and saves on small fixes as you realize that you did not insist that some command remain silent.

3. Because it will be useful for friendlier display later on, we are also putting the variable labels alongside. The syntax `"‘: var label ‘v’’"` instructs Stata to look up the variable label. The nesting of `‘ ’` here should not cause alarm. You need to understand only that Stata works with such nested macro references just as you would with parenthesized expressions in algebra of the form (... (...) ...): what is innermost gets evaluated first. So Stata works out what `‘v’` is referring to, which is the variable name each time around the loop. Then it looks up the variable label for that variable. If there is no variable label defined, an empty string will be returned and no harm done.

   Stata has a bundle of such look-up tools called extended macro functions; even some very experienced users cannot trust themselves to remember how to find the help on such functions quickly, but they do find it easy to remember to look at `help macro` first and then jump to the help for extended macro functions.

4. It does not apply here, but I will add that if double quotes, `"`, appear inside any variable label, you will need compound double quotes, `‘" "’`, to hold the entire label.

5. You might have spotted that the `replace` commands with numeric results all have very similar form and so could be rewritten as another `foreach` loop. That is a matter of programming taste.

The strategy needs to be introduced with caution. There is a tacit assumption that we have no more variables we want to `summarize` than observations—congratulations if you spotted that instantly! If we have 101 variables and 100 observations, where would we put results for the 101st variable? That usually is not a problem, but if it is, then there is a work-around of just increasing the number of observations in the dataset. However, you are now in territory where switching to a more versatile tool is a better idea.

We are also deliberately breaching a convention that whatever is in an observation (in non-Stata terminology, a row, record, or case of a dataset) belongs together. That is, we are putting results for entire variables in single observations and putting results alongside original data with which they are not aligned. If you use spreadsheets frequently, none of this may surprise you because you may do similar things often. If you use relational databases frequently, your reaction to what I am explaining may be some mixture of disbelief and revulsion. What is certain is that Stata won't object to that, and equally, it won't object to your doing things later with the combined dataset that do not make sense. So the counsel can only be: Watch out. If you dislike this strategy on principle, you have admirably refined tastes and should immediately consider the next tip.

What is also crucial in our example is that in looping over variables, we let `summarize` use for each variable as many nonmissing values as exist. If you wished to insist on exactly the same observations being used each time, that would imply an extra `if` condition on the `summarize` command. It could be, for example, that you want to select only observations with nonmissing values for all the variables shown.

Tip 7: For a more general way of posting results, check out `postfile` *and its associated commands.*

I am going to let this tip stand as a flag. The idea is well explained in self-contained documentation. Start with the help for `postfile`, and proceed if desired to the manual entry.

Tip 8: `list` *can be very useful for tabulation of results stored in variables.*

Most Stata users learn `list` early, and many use it often to look at their dataset or at least at small parts of it. In early versions of Stata, before there was a Data Editor, `list` was the main command for looking at data, although many users who have started with Stata with more recent versions may find looking at the Data Editor to be more congenial or convenient.

`list` has some simple but important advantages in looking at results stored in variables. As it often seems overlooked for this purpose, let's spell some of them out briefly.

1. There is no puzzling out of how much space to assign to values, spaces, text headers, and so forth, because `list` does all that for you.

2. The ordering of columns (namely, variables) in the list and the selection of observations using `if` or `in` are totally under your control.

3. Because `list` respects display format, to get (for example) particular numbers of decimal places, you just need to call up `format` beforehand to say what you want. So you will usually want the number of observations, always an integer, to be shown as such. But you might also want skewness and kurtosis rounded, say, to 3 decimal places. Some but by no means all tabulation commands support different formats for different result columns: at the time of writing, it is a major limitation of `tabstat`, for example, that you can have just one numeric format throughout a results table.

4. Because `list` will happily mix numeric and string variables at your command, you can add comment columns or fields easily. That could be a way, for example, of adding stars to match significance levels if they are to your taste, or if your boss or reviewers insist upon them.

5. Because `list` should be a command you already know, learning some more details is likely to be less effort than mastering yet another new command. As hinted above, it seems that many Stata users are aware of the basic syntax of `list` but have not explored its more advanced options for tuning presentation.

6. `list` itself offers no export options, but the user-written command `groups` (Cox 2003c) is a wrapper for `list`, and as of January 2012 it includes a `saving()` option so that what is listed can be saved as a new Stata dataset. If interested, download `groups` using the `ssc` command.

That is quite enough of a small sermon on virtues. Let us see how we can put this to work without too much effort.

```
. format mean sd  %3.2f
. format skew kurt %4.3f
. char varlabel[varname] " "
. char sd[varname] "s.d."
. list varlabel n mean sd skewness kurtosis if n < ., noobs sep(0) subvarname
```

	n	mean	s.d.	skewness	kurtosis
Price	74	6165.26	2949.50	1.653	4.819
Mileage (mpg)	74	21.30	5.79	0.949	3.975
Repair Record 1978	69	3.41	0.99	-0.057	2.678
Headroom (in.)	74	2.99	0.85	0.141	2.208
Trunk space (cu. ft.)	74	13.76	4.28	0.029	2.192
Weight (lbs.)	74	3019.46	777.19	0.148	2.118
Length (in.)	74	187.93	22.27	-0.041	2.042
Turn Circle (ft.)	74	39.65	4.40	0.124	2.229
Displacement (cu. in.)	74	197.30	91.84	0.592	2.376
Gear Ratio	74	3.01	0.46	0.219	2.102
Car type	74	0.30	0.46	0.887	1.787

The preparation needed is minor.

1. We think up reasonable formats for the noninteger numeric result variables. With display of small integers, Stata's default formats will always work well with `list`, but you may need to do more work with very large integers, especially if you dislike displays of large integers in scientific or exponential format, such as `1.000e+09`.

2. An extra detail here is showing how to define headers that could not be legal variable names: we insist on a blank header for the variable label column and spell out `s.d.` rather than `sd`. The `subvarname` option instructs `list` to look for the characteristics that carry the text. Note that variable labels would usually be

too long for this purpose, so it is convenient that `list` lets you define header text using characteristics. That may seem awkward, but having to redefine variable labels for this purpose only to have to redefine them for most other purposes would be even more awkward.

If characteristics are new to you, then `help characteristics` gives a miniature tutorial.

3. Seeing the observation identifiers and separation lines that would appear by default would not help us here, so we override those defaults.

4. Note also that we select observations with nonmissing values with an `if` qualifier; using `if` is less efficient, but when working interactively it is easier to apply than the equivalent `in` qualifier. (Programmers should try to work with `in` rather than `if`, however.)

Tip 9. `tabdisp` *can be useful interactively for preparing simple tables.*

This is going to be another flag. Another earlier column gave detailed discussion (Cox 2003b). Here is a trick worth mentioning, which is obvious once you know it. At first sight `tabdisp` offers the possibility of just one numeric format, which seems very restrictive. However, this is easy to subvert. Just create a string variable version of what you want to show, using the format argument of the `string()` function. `tabdisp` shows string variables exactly as they are and does not look inside. So long as readers of the table see the numeric characters they expect to see, no one need know or care that they are seeing a string variable.

Tip 10. Remember that results in variables may be plotted in graphs, possibly in table-like graphs.

Naturally, that may be turned around too. If you want to plot your results, you need to put them in variables first. (There are some minor exceptions but the principle remains valid.)

See particularly Cox (2008, 2009) for earlier discussions of this utterly basic idea, still far too often neglected.

4 Check out existing reduction commands

Tip 11. Stata commands such as `collapse`, `contract`, `egen`, `estimates`, *and* `statsby` *already offer ways of creating new datasets of results. Be aware of the possibilities before you start reinventing them for yourself.*

What is likely to be the most helpful tip of all has been saved until last for emphasis. The passage from a dataset to a set of results in the form of a new dataset is often much easier than you might imagine. I think of such commands as reduction commands, which reduce a dataset to a smaller one including only key results. There is more than a nod here to a longstanding but still essential idea of "data reduction" (Ehrenberg 1975).

Of the commands singled out above, `collapse` is quite well known, but users often overlook the complementary `contract` command, focusing specifically on saving group frequencies and percentages as new data. `estimates` is also quite well known, but `statsby` is often overlooked (see Cox [2010b] for a tutorial on its graphical application).

The remaining command just mentioned, `egen`, is not so obvious as a reduction command but is made so by simple tricks. The running example in this paper focused on saving results for each of a set of variables. The also common problem of wanting results for each of various groups of observations was neglected, so let us look at such a problem. `collapse` does not support calculations for skewness and kurtosis, but `egen` does offer pertinent functions.

Let's suppose that we want to compare skewness and kurtosis for distinct groups on the categorical variable `foreign`. The syntax would be

```
. by rep78, sort: egen skew = skew(mpg)
. by rep78: egen kurt = kurt(mpg)
```

The same result, say, the skewness for `foreign` being 1 (namely, foreign cars made outside the United States), will be stored repeatedly for all the observations with that value of `foreign`. But now that the variable exists, it is available for `collapse`. We can specify, for example, that we want to keep its minimum or maximum; that's enough to select just one result.

Another trick for reduction is to use `egen`'s `tag()` function to select just one observation in each distinct group. Then reduction to a smaller dataset is effected by `keep if` *tag*, where *tag* is the variable just created.

5 Conclusion

I will now simply gather the specific tips given.

Stata commands such as `collapse`, `contract`, `egen`, `estimates`, and `statsby` already offer ways of creating new datasets of results. Be aware of the possibilities before you start reinventing them for yourself.

Stata typically holds major results in memory, at least immediately after the command has run. You can thus access these for presentation in a different form.

The `display` command with specified format is convenient for presenting numeric results as you wish.

To show a set of results for different variables or groups, loop over the possibilities with `foreach` or `forvalues`. Typically, you will need to explain each piece of output with appropriate text.

To suppress output you do not want, prefix a command with `quietly`.

SMCL offers extra control over presentation with `display`.

Use `generate` and `replace` to put results from a command one by one into new variables.

For a more general way of posting results, check out `postfile` and its associated commands.

`list` can be very useful for tabulation of results stored in variables.

`tabdisp` can be useful interactively for preparing simple tables.

Remember that results in variables may be plotted in graphs, possibly in table-like graphs.

6 References

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

———. 2003a. Speaking Stata: Problems with lists. *Stata Journal* 3: 185–202.

———. 2003b. Speaking Stata: Problems with tables, Part I. *Stata Journal* 3: 309–324.

———. 2003c. Speaking Stata: Problems with tables, Part II. *Stata Journal* 3: 420–439.

———. 2008. Speaking Stata: Between tables and graphs. *Stata Journal* 8: 269–289.

———. 2009. Speaking Stata: Paired, parallel, or profile plots for changes, correlations, and other comparisons. *Stata Journal* 9: 621–639.

———. 2010a. Speaking Stata: The limits of sample skewness and kurtosis. *Stata Journal* 10: 482–495.

———. 2010b. Speaking Stata: The statsby strategy. *Stata Journal* 10: 143–151.

Ehrenberg, A. S. C. 1975. *Data Reduction: Analysing and Interpreting Statistical Data*. London: Wiley.

About the author

Nicholas Cox is a statistically minded geographer at Durham University. He contributes talks, postings, FAQs, and programs to the Stata user community. He has also coauthored 15 commands in official Stata. He wrote several inserts in the *Stata Technical Bulletin* and is an editor of the *Stata Journal*.