



The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search

<http://ageconsearch.umn.edu>

aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.

Speaking Stata: Problems with tables, Part I

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

Abstract. Tables in some form or another are part and parcel of data management and analysis. The main general-purpose tabulation commands, `tabulate`, `table`, and `tabstat`, are reviewed and compared. When these do not provide a tabulation solution, one key strategy is to prepare the material for tabulation as a set of variables, after which the table itself can be presented with `tabdisp` or `list`. This is the first of two papers on this topic.

Keywords: pr0010, tables, tabulate, table, tabstat, tabdisp, list

1 Introduction

Tables are part and parcel of data management and data analysis, as every reader knows. The main Stata data model is a two-dimensional table: a Stata dataset is in essence a table of observations (rows) by variables (columns). In Stata sessions, we consume and produce all kinds of tables of data and tables of results. Even many graphs have near tabular structure. That is most evident given (say) a set of bars or a set of boxes in a regular array. It is also apparent, with a suitably generous definition of rows and columns, even in scatter plots and their kin. Indeed, as has often been argued, the separation between tables and graphs owes much to a division of labor that followed the introduction of printing and is less fundamental than may at first appear. The division was little more than a reflection of the limits of publishing technology. For a few centuries, typesetters prepared “tables” and draftspeople prepared “figures”. This distinction is still encapsulated in our contents lists yet is slowly becoming obsolete. The great statistician John W. Tukey put all the tables and figures in his *Exploratory Data Analysis* (1977) in a single sequence of exhibits, that name being perhaps attributable to his special fondness for detective stories.

In a wider context, therefore, tables and graphs are all reasonably considered as exhibits or displays of some kind. Nevertheless, let us start with those tabulation tasks that Stata makes simple and move to looking at some of the problems that can then arise. (The wider theme of tables and graphs has been trailed partly because it will reappear shortly in this column.) Tabulation, like all tasks, has its easy triumphs and its frustrating tribulations. Setting aside a variety of special-purpose tabulation commands, we are going to look at the general task of producing tables to our own specifications. The decision-making sequence, it is suggested, runs as follows:

1. Is there an official Stata command that will do what we want in one, the details controllable by way of option choices? If any is available, we should use it directly.

2. Can we get there with a few basic steps, preparing one or more variables or putting some values for tabulation in some other structure, and then passing them to a tabulation command, as just mentioned? Applying yourself to this can impart the skills to tackle all sorts of variations on the same underlying problem, without being dependent on whether someone else has codified a solution.
3. Is there a user-written command that can help? This is perfectly sensible if you know that there is a command that works well. However, user-programmers often stop once they have solved their own basic problem (and why not?). A user-written command frequently turns out, therefore, to lack the generality and the flexibility that you may desire.

This is the first of two papers on this topic. The sequel will follow in the next issue of the *Stata Journal*.

2 Official Stata commands for tabulation

Stata has several general-purpose tabulation commands. As said, we are not going to look at special-purpose commands such as [ST] **epitab**, [ST] **ltable**, [SVY] **svytab**, or [XT] **xttab**. If these are important to you, you have probably been using them since shortly after you started with Stata. The existence of several different general commands means that you should spend a little time finding out what each is best at. (On the other hand, imagine that they were combined into fewer commands. We would gain something, but the syntax would almost inevitably become more complicated.)

Let us review the three main commands in turn. The manual entries at [R] **tabulate**, [R] **tabsum**, [R] **table**, and [R] **tabstat** give the crucial details; here, rather, is an overview.

2.1 tabulate

tabulate has the longest Stata history, all the way back to Stata 1.0. In a sense, it is two related commands, as the manual entries [R] **tabulate** and [R] **tabsum** indicate. **tabulate** is part of the executable and written in C, which means that users so inclined cannot look inside and tweak a copy of the code. The corresponding advantage, however, is that **tabulate** is necessarily faster than commands based on interpreted code, notably **table**. **tabulate** is useful for

- one-way and two-way tables of frequencies (options for row, column, and table percents);
- chi-square tests and Fisher's exact test for two-way tables;
- some coefficients of association (Cramér's V , Goodman and Kruskal's gamma, and Kendall's τ_b) (for Somers' D , see Newson 2002);

- summarizing means and standard deviations of another variable for each of the categories shown;
- saving frequencies to a matrix;
- the ability to generate a set of indicator variables, a feature easy to overlook.

Despite its long history, `tabulate` continues to improve, and several small changes were made on the release of Stata 8 (see [U] **1.3.11 What's new in statistics in all fields**). One notable addition in Stata 8 is a `sort` option to specify that frequencies in one-way tables be presented highest first.

Univariate chi-square tests are not provided; for those, see the user-written commands `chitest` and `chitesti` in the package `tab_chi` on SSC.

2.2 table

`table` was introduced in Stata 5.0. It is written as interpreted code and calls `tabdisp`, part of the executable and also introduced in Stata 5.0, to produce the table itself. The more complicated the table you want to produce, the more likely it is that you will prefer `table` to `tabulate`. You can show a wider range of up to five statistics; not just frequencies, percents, means, and standard deviations, but anything computed by `collapse`, which includes weighted and unweighted sums, various quantiles (percentiles), and the interquartile range. Even more useful in practice are `table`'s better support for producing three-dimensional tables and tuning display formats and column widths, alignments, and spacings. By default, missing statistics (empty cells) are left blank; on the whole, this makes it easier to scan and to browse sparse tables, those with lots of zeros, although they may still take up a fair amount of space.

Both `tabulate` and `table` can be combined with `by:` to produce higher-dimensional tables. Thus, `bysort d: table a b c` gives a four-dimensional table of the frequencies of the cross-combinations of variables `a`, `b`, `c`, and `d`. A four-dimensional table is just a set of three-dimensional tables, and that principle extends higher.

2.3 tabstat

`tabstat` was introduced in Stata 7. It also provides tables of summary statistics, and it also is interpreted code. As it emits a line of output at a time and is not dependent on, say, `tabdisp`, it can be used to produce fairly substantial tables.

`tabstat` is especially valuable as a wrapper for `summarize`, giving you a way to customize your own summary statistics command. In particular, `summarize` by default yields count, mean, standard deviation, minimum, and maximum. With the `detail` option, it also yields other statistics, including various percentiles, variance, skewness, and kurtosis. You can pick and choose from these with `tabstat`:

```
. tabstat varlist, s(n mean sd min q max)
```

specifies `q` to add lower, median, and upper quartile to the statistics produced by `summarize` by default, while

```
. tabstat varlist, s(n mean sd skew kurt)
```

gives a moment-based picture of data. Here, as you may guess, the `s()` option specifies statistics. Various statistics easily derived from the results of `summarize` (coefficient of variation and standard error of the mean) are available in addition to a standard list. Three other very useful features are the ability to specify groups (as specified by some other variable) as well as a variable list, to control whether variables or statistics are shown in columns, and to save results to one or more matrices.

2.4 Check out the manual entries

These commands are all excellent in different ways and should take care of many of your basic tabulation tasks. It is mostly surmise on my part, but I guess that even many experienced Stata users may be using the `tabulate` command that they have been familiar with for some years without realizing that `table` or `tabstat` may offer better solutions. Hence, make a Stata resolution to skim the manual entries to see what you have been missing.

3 Doing it yourself

What techniques are available when such commands do not serve? And what techniques will be attractive to you as user? Let us think backwards from finished product to initial production.

3.1 Stata and word or text processing software

The final stage, naturally, will be preparation of tables in whatever other software you use to prepare documents. Most Stata users appear to employ word processing software. A substantial number use some flavor of $\text{T}_{\text{E}}\text{X}$ or $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$. Some prepare documents in HTML or some other web-readable form. Many combine some of these forms of output.

Stata Corporation is characteristically mindful of its cross-platform support (for Stata on Windows, Macintosh, or Unix) and commercial independence and so is particularly reluctant to assume that its users are wedded to any specific proprietary software. Stata itself, therefore, lacks tools tied to any other commercial program, although it does provide some support for public domain mark-up languages, notably HTML. If you use Microsoft Office (Word, Excel, etc.), you will find that many like-minded Stata users share advice on tips and tricks on Statalist. As yet, there appears to be no coordinated compilation of such advice, but a search of the Statalist archives may prove useful. Various user-written Stata programs are aimed directly or indirectly at users of $\text{T}_{\text{E}}\text{X}$, $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, or HTML, or are particularly helpful to them. See, for example, the paper by Newson (2003) in this issue. More generally, use `findit` with the appropriate keyword to locate utilities pertinent to you.

3.2 Downstream or upstream?

But before this final stage, you must produce, at least, the raw material for a table within Stata. Sometimes it can seem easiest to keep a log of output and then identify the elements you need for a table from the separate output of several commands. I think of this as a downstream solution. We select the results we need from among many other results. If the particular task is unlikely to be repeated, this may be as efficient as any other approach. What is more satisfying, naturally, is to be able to produce a table directly. I think of this as an upstream solution: we collect nearer the source. This is also less error-prone. Typically, for example, Stata output carries more significant figures than you want to include in a report, especially if it is an academic paper. The usual argument, as you know, is that it is easy to round to fewer significant figures but impossible to replace details never supplied, so as in similar software Stata's defaults tend to be generous on detail. However, rounding numbers by hand is not a fit job for humans, who can make mistakes or get bored.

3.3 Data reduction commands

Sometimes it is a good idea to use a data reduction command, which takes your data and produces a much smaller dataset, commonly some variables, often categorical, defining groups together with new statistics for each of those groups. Examples of reduction commands are `collapse` ([R] `collapse`), `contract` ([R] `contract`), and `statsby` ([R] `statsby`).

`table` is linked to `collapse` so that, by and large, what you can get by applying `collapse` can also be obtained from `table`. Similarly, as `contract` produces frequencies of combinations, it does not yield anything that you cannot see through `tabulate` or `table`. However, sometimes it is easier to think through a basic tabulation problem if the data have been reduced first. This is largely a matter of psychology, but never mind.

contract as an example of reduction

To make clear the kind of connection being discussed here, take the familiar `auto` dataset

```
. sysuse auto
```

and the canonical Stata twoway table

(Continued on next page)

```
. tabulate rep78
```

Repair Record 1978	Freq.	Percent	Cum.
1	2	2.90	2.90
2	8	11.59	14.49
3	30	43.48	57.97
4	18	26.09	84.06
5	11	15.94	100.00
Total	69	100.00	

Now let us reduce the data to the corresponding bare essentials. But before we do that, here is a “Danger!” flag. You are about to throw away most of your data. In this case, it does not matter, because these are the `auto` dataset bundled with Stata, and as you know you can retrieve them again readily. But for a real problem, with your own data, be warned that the reduction commands can be used to destroy the fruits of a lot of hard work, so make sure that you have **saved** anything important before you invoke one of these commands.

```
. contract foreign rep78
```

```
. list
```

	rep78	foreign	_freq
1.	1	Domestic	2
2.	2	Domestic	8
3.	3	Domestic	27
4.	4	Domestic	9
5.	5	Domestic	2
6.	.	Domestic	4
7.	3	Foreign	3
8.	4	Foreign	9
9.	5	Foreign	9
10.	.	Foreign	1

What `contract` does is produce one observation for every distinct combination of its *varlist* together with a new variable, by default `_freq`, recording the frequency of that combination in the original data. (The original data, in terms of that *varlist*, could be restored by applying `expand`: the names `contract` and `expand` underscore the duality. Other variables, however, are gone forever—except that if you **saved** the data, as advised, then they can be examined once more.)

The reduced dataset is sufficient to reproduce the table, except that to get it we must specify weights:

```
. tabulate foreign rep78 [w = _freq]
```

(What would have happened if we forgot to specify weights? If the answer is not clear to you, try it to be clear on what is happening here.)

Two details arise on close examination.

First, missing values. By default, they are not shown in the table, but their frequencies are kept with the data, given the order to **contract**. Most of the time that is the right way round; in each case, the default can be overturned with an option. The Stata philosophy, most of the time, is to ignore missing values, but it is your decision whether to **drop** them from the dataset.

Second, combinations that do not occur in the dataset. There were no cars out of the original 74 that were foreign with repair record graded 1 or 2. **tabulate** shows that in one legitimate way, by showing 0 in the corresponding cells. **table** shows that in another legitimate way, by showing nothing (a blank, if you like) in the corresponding cells. (This small difference deserves repeated emphasis, as in some circumstances the nuance of presentation may be something you care about.) However, by default, **contract** does not include them in the reduced dataset, which is logical enough, as—this is the whole point—no such cars occurred in the original dataset. Nevertheless, there are circumstances in which an explicit zero is helpful, and **contract** has a **zero** option.

Put more whimsically, Stata behaves, unless encouraged, like an unreconstructed empiricist, trusting only what is observed and to hand, and having no concern for what might have been. **contract** will, however, “fill in” cells, in this sense: the option **zero** forces the addition of extra observations representing zero frequencies of combinations of two or more variables, for which **_freq** is, indeed, 0. (Inside **contract**, this is done by invoking **fillin**, as you may see by inspecting the code. If **fillin** sounds like a tool of general use to you, see [R] **fillin**.)

statsby as an example of reduction

Apart from **collapse** and **contract**, one more reduction command in official Stata should be mentioned here, namely **statsby**. Here is a token example, assuming that the full **auto** dataset have been read in once more,

```
. gen gp100m = 100 / mpg
. statsby "regress gp100m weight" _b[_cons] _b[rep78] e(r2), by(rep78) clear
```

In this particular example, we are guessing that a transformed variable, gallons per 100 miles, is more likely to have a linear relationship with car weight than the original variable, miles per gallon.

The big idea of **statsby** is just to apply a command repeatedly to a set of groups and to gather specified results into a new dataset. To do this, we must look at documentation to find out the syntax for the results we desire. (If the manuals are not to hand, or even if they are, it can be quicker to try the command once and look at saved results to work out the syntax: **return list** and **ereturn list** are helpful for this reverse engineering.)

In our example, we saved the intercept and slope from each regression, together with an R^2 statistic. You will realize that another way to do it would have been to crunch through the regressions one by one. That itself could have been produced easily, as with

```
. bysort rep78: regress gp100m weight
```

The difference, however, is that we need to trawl through all the output to select whatever we wish to report, a fairly trivial task in this case but naturally something we would strongly wish to avoid with more than a few groups. Moreover, having the results in a dataset can be useful in other ways, particularly if they are, literally, to be treated in turn as a dataset, say, by graphing them in some form.

This still leaves open the question of how to present a table. The regression coefficients and R^2 are now in variables and so could just be `listed`. There is at least one attractive alternative, to use `tabdisp`, which we will get to shortly.

3.4 Back to basics: preparing variables for tables

From supplied reduction commands, we turn to basic tools such as `by:` (Cox 2002a), `foreach` and `forvalues` (Cox 2002b, 2003), and `egen` (Cox 2002d). With their aid, many tabulation tasks can be broken down into a preparation step and a presentation step. In fact, most of the work usually lies in the preparation.

Take a type of question that often arises on Statalist. Members ask for minor variations on the theme of `tabulate`. One example will serve to illustrate several little techniques for the remainder of this column. Instead of frequencies, percents, and cumulative percents, you might desire the first two and cumulative frequencies. How can we produce such tables for ourselves? Here is a reminder of the `tabulate` default:

```
. tabulate rep78
```

Repair Record 1978	Freq.	Percent	Cum.
1	2	2.90	2.90
2	8	11.59	14.49
3	30	43.48	57.97
4	18	26.09	84.06
5	11	15.94	100.00
Total	69	100.00	

When you need to think outside the range of possibilities provided by `tabulate`, `table`, or `tabstat`, the first thing to try is to produce what you want tabulated in the form of new variables. If you can do this, the tabulation itself is often easy by comparison. Getting the frequencies will be easy to you if you have studied `by:` (Cox 2002a):

```
. bysort rep78: gen freq = _N
```

remembering that under `by: _N` is interpreted as the number of observations within each distinct group. Or, you can fire up `egen`:

```
. bysort rep78: egen freq = count(1)
```

or

```
. bysort rep78: egen freq = sum(1)
```

The table will look better if we supply a variable label:

```
. label var freq "Freq."
```

The percents are simple, as well, except for one wrinkle: do you want to include the missing values as part of the total or not? That's your decision, as a substantive matter. You can see from the `tabulate` results just given that there are 69 cars with nonmissing values, so

```
. gen Percent = 100 * freq / 69
```

would give you the correct percents if you want to omit missing values. (Capitalizing `Percent` saves us the labor of creating a variable label.) However, for your do-files, and even more for your own programs, you do not want to be dependent on having found out the total, here 69, in advance. Using `_N`, as in

```
. gen Percent = 100 * freq / _N
```

would always base calculations on the total number of observations, whether they were missing or not on any particular variable. More general yet is to get Stata to `count` for you. The most useful syntax looks awkward the first time it is encountered.

```
. count if !missing(rep78)
```

counts how many observations are *not* missing on `rep78`. It is useful to know an abbreviation introduced in Stata 8:

```
. count if !mi(rep78)
```

While we are on this issue, let us examine some details that can bite you if you are not aware of the precise underlying definitions. We will digress a little from tabulation, but this is stuff you need to know for this task and for many others. In Stata 7 or earlier, you may have been accustomed to writing

```
. count if rep78 != .
```

or

```
. count if rep78 < .
```

where the period `.` naturally stands for numeric missing. In Stata 7 or earlier, these were synonyms: for numeric variables, missing is regarded as higher than any nonmissing

value, so being not equal to missing is the same as being less than missing. The second form had the advantage of saving a keystroke (a big advantage if you use Stata a lot).

With Stata 8 come extended missing values, `.a` through `.z`, all of which are regarded as larger than simple `.`—itself now called system missing, or even “sysmiss”. Hence, the condition `!= .` now allows the possibility that values are equal to any of `.a` through `.z`. This strengthens the appeal of `< .` as a simple excluder of all numeric missing values, which you commonly want.

However—and in this area there always seems to be a “however”—none of this applies to string variables, for which missing means, as far as Stata is concerned, the empty or null string `""`. (Exceptions to this rule were discussed in [Cox 2002c](#).) If you want to learn the most general ways of counting,

```
. count
```

counts observations unconditionally, and

```
. count if !mi(varname)
```

counts observations with nonmissing values of *varname*, irrespective of whether *varname* is numeric or string. A neat detail for your do-files or programs is that `count` leaves the number it counted in its wake within `r(N)`. Knowing that, we often put `quietly` in front to suppress the printing of the count and just pick up from `r(N)`. As `r(N)`, like all r-class results, is easily overwritten, make sure that you do that promptly after it has been created. End of digression!

So, with an eye towards more general code

```
. qui count if !mi(rep78)
. gen Percent = 100 * freq / r(N)
```

is, for interactive code, long-winded but, for your files, a better way to do it.

So far, so good: but what about the cumulations? You may know that the function `sum()` produces cumulative sums. It should also be clear that getting the right cumulative sum depends on observations being in the right sort order, but we attended to that earlier: the effect of `bysort rep78` is to leave the data sorted in order of `rep78`. If we now produce `sum(freq)`, the problem is that each value of `freq` will be added `freq` times to this sum. So we must ensure that we use each group frequency only once. One way of doing this is to tag just one observation in each group. There are two systematic ways of doing this, tagging the first or the last, bearing in mind that a group could be represented by only one observation, in which case the first and the last are clearly identical.

```
. bysort rep78: gen tag = _n == 1
```

would tag the first observation in each group with 1 and all others with 0. (The reason? For the first, and the first only, `_n` is 1, so the expression `_n == 1` is true and so is evaluated numerically as 1. For the others, `_n` is greater than 1, and that expression

is false and so is evaluated numerically as 0. If memory were short, we could insist that `tag` be generated as a byte variable. For more explanation and examples, see Cox 2002a.) Similarly,

```
. bysort rep78: gen tag = _n == _N
```

would tag the last. Often it is immaterial which we choose; sometimes there are advantages to one way or the other. What we can now do is

```
. gen cufreq = sum(tag * freq)
```

Where an observation has been tagged so that `tag` has value 1, it contributes $1 \times \text{freq}$ to the cumulative sum. Where it has not been tagged, it contributes $0 \times \text{freq}$, which naturally leaves the sum unchanged, as desired. This is one of several ways in which the fact that `tag` takes only values 1 and 0 leads to neat solutions of various little problems. Adding a variable label,

```
. label var cufreq "Cum."
```

we have enough material to attempt a table. (In this example, there is a small but definite advantage in tagging the first.)

At this point, you may be feeling that although you understand the trick, the issue is how you would reproduce it for yourself or come up with similar tricks for similar problems. (Showing that teachers are smart enough to solve the problems they set themselves is, after all, only a secondary purpose of education.) Like teachers since time immemorial, my response is to show you other ways to do it so that your repertoire of tricks is enlarged. In the Stata exam, being able to solve the problem once is usually enough. Be reminded of a simple pearl of wisdom: in any sufficiently rich language, there's more than one way to do it (Wall and Schwartz 1990, 4).

Strictly, the tagging is unnecessary for the computations, although it has advantages in other circumstances, as we see, for example, in section 3.6. As it turns out, tagging is wired into Stata as an `egen` function, so we could have done it this way:

```
. egen tag = tag(rep78)
. gen cufreq = sum(tag * freq)
```

Alternatively, we could have gone one level down and done everything from first principles. Repeating earlier code to give the correct context,

```
. bysort rep78: gen freq = _N
. qui count if !missing(rep78)
. gen Percent = 100 * freq / r(N)
```

we could do the cumulation this way (1)

```
. by rep78: gen cufreq = _N * (_n == 1)
. replace cufreq = sum(cufreq)
```

or we could do it this way (2)

```
. gen cufreq = sum(1)
. by rep78: replace cufreq = cufreq[_N]
```

or even this way (3)

```
. gen cufreq = _n
. by rep78: replace cufreq = cufreq[_N]
```

A few comments are in order. (3) perhaps gets highest marks as the most Stata-ish solution. To understand it, and indeed (2) and (1), you have to be familiar with the rule that with **by**: `_n` and `_N` are interpreted within groups, and without **by**: they are interpreted within the whole dataset.

(2), however, is as good or better for various reasons. Remembering that counting is just adding 1 and 1 and 1 ... really is going back to basics, but because of that it is a fundamental technique. Also, this solution generalizes nicely to a more complicated problem in which you have a third variable—or even an expression—giving weights: instead of `sum(1)` you just need `sum(exp)`. That small generalization takes all the strain.

(1) looks awkward once you have seen (3) or (2). But it could be a way of putting in Stata code the following idea, already mentioned: I can get each group frequency; I just need to ensure that it is included only once from each group when I cumulate to get the cumulative frequency.

Whatever way we do it, a label will give a better column heading in our table:

```
. label var cufreq "Cum."
```

3.5 tabdisp is your friend

For presentation, we are going to use `tabdisp`. As mentioned earlier, `tabdisp` was introduced in Stata 5.0. It is billed as a programmer's command and as such is documented in [P] `tabdisp`, but it is often the best way to tackle interactive tabulation problems. (We would equally reverse the definitions: if you use `tabdisp`, you qualify as a Stata programmer. Congratulations!)

As our first stab,

```
. tabdisp rep78, c(freq cufreq Percent)
```

Repair Record 1978	Freq.	Cum.	Percent
1	2	2	2.898551
2	8	10	11.5942
3	30	40	43.47826
4	18	58	26.08696
5	11	69	15.94203
.	5	69	7.246377

Not bad! For most circumstances, we need to fix two things: missing values of **rep78** should be excluded from the table, and **Percent** needs a sensible display format, say

```
. format Percent %9.2f
. tabdisp rep78 if !mi(rep78), c(freq cufreq Percent)
```

Repair Record 1978	Freq.	Cum.	Percent
1	2	2	2.90
2	8	10	11.59
3	30	40	43.48
4	18	58	26.09
5	11	69	15.94

tabdisp gives us a great deal of control over the details of presentation. We specified the column order for ourselves, within the **c()** option, thinking that cumulative frequency belonged next to frequency. It is useful to have that ordering under our thumb (or our fingers, more likely).

As the example of **Percent** showed, we can arrange display formats outside **tabdisp**. This allows us more flexibility than using the **format()** option of **tabdisp** to specify a numeric format. That option controls the format within the table of *all* numeric cell variables shown. (Stata has yet to adopt the idea, used in some other languages, of being able to specify, all at once, an array of formats to control an array of results.) Note the exception: in **tabdisp** we are allowed string variables as part or indeed all of what is shown in a cell. Any string formats, however, should be specified outside **tabdisp**.

Other options are available to tune column widths, alignments, and spacings; these are the options inherited by **table** and mentioned in section 2.2.

tabdisp also makes some choices for us, which are typically smart: in particular, variable labels and value labels show up, usually as we would wish. If **rep78** had had value labels attached, they would have been used in the left-hand column.

Two key warnings are needed at this point. First, in terms of our example, **tabdisp rep78, c(whatever)** will show each distinct value of **rep78** just once, together with the corresponding values of *whatever*. If the variables in *whatever* are invariant for groups of **rep78**, as they are in the calculations we have performed, then it is immaterial which observation in each group is shown. But otherwise, it is important to know that **tabdisp** selects the first observation it finds with each distinct value. Alternatively, you can arrange for yourself precisely which observation is shown. **tagging** one in each group and specifying **if tag** is an example of that technique. Conversely, if your groups are all unique (they are identifiers, by design or in effect), then **tabdisp** becomes an alternative to **list**. The small difference that **tabdisp** is inclined to use variable labels for column headers can be helpful, especially as the outside world typically has no interest in or knowledge of the variable names you choose for your Stata datasets.

Second, as the name indicates, `tabdisp` has one aim in life: the display of tables. By itself it does no calculations, either of marginal totals or of any other statistics. There is a way of getting a display of marginal information, explained at [P] `tabdisp`, and it can be useful for do files or programs.

The manual entry [P] `tabdisp` repays careful study. In this issue, Smeeton and Cox (2003) include another example of `tabdisp` being used interactively.

3.6 list is your friend

One reaction to the appearance of `tabdisp` in the drama may have been that it is yet another character to have to understand. If so, consider another possibility: `list`, which is another Stata veteran, with a history all the way back to Stata 1.0, yet it experienced a major overhaul in Stata 8. The more experienced you are as a Stata user, the likelier it is that you have noticed several detailed changes in the output of `list` but have yet to examine its new possibilities carefully. Do not let the name put you off. `list` is, in a strong sense, another tabulation command. It does, naturally, work best when the table you want is in the major form of, surprise, a list, but then many are of this form. In particular, it does not support features such as supercolumns as allowed in `tabdisp` (and therefore `table`). Apart from that, it often does a very good job.

When `list` is used for tabulation, there is one key behavioral trait you need to work with: there is a built-in tendency to show all the observations you have, and you will need to specify when you want matters otherwise. Put plainly, it simply does not know that you want a table whenever you know that. The key to controlling this trait is equally simple and lies in the use of `if`.

Let us replay our running example, from the top, using the variations that let us play the `list` game with success.

```
. bysort rep78: gen freq = _N
. label var freq "Freq."
. count if !mi(rep78)
    69
. gen Percent = 100 * freq / r(N)
. format Percent %9.2f
. egen tag = tag(rep78)
. gen cufreq = sum(tag * freq)
. label var cufreq "Cum."
. list rep78 freq cufreq Percent if tag, noobs
```

rep78	freq	cufreq	Percent
1	2	2	2.90
2	8	10	11.59
3	30	40	43.48
4	18	58	26.09
5	11	69	15.94

Here, the option `noobs` switches off the irrelevant observation numbers. There may be a small surprise here if you have not tried this before: `list` does not honor variable labels, even very short ones. No doubt this reflects its role for listing what may be several variables in columns, when very often space is at a premium. The poor titling may not matter interactively, but if it does, we need a new trick. We define particular characteristics of the variables in question and then specify the option `subvarname`, after which all is well.

```
. char freq[varname] "Freq."
. char cufreq[varname] "Cum."
. list rep78 freq cufreq Percent if tag, noobs subvarname
```

rep78	Freq.	Cum.	Percent
1	2	2	2.90
2	8	10	11.59
3	30	40	43.48
4	18	58	26.09
5	11	69	15.94

Note that, just for once, `varname` is to be typed literally: it is not here a pattern to be substituted by whatever variable name you wish. For more on characteristics, see [U] **15.8 Characteristics**.

4 To be continued . . .

Tables are such an enormous subject that even with another column to follow, we will not be able to touch on more than a few kinds of questions. The aim of this one has been to persuade you, once again, of two key things you can do. First, keep watching out for basic general commands that are the key to many specific problems. You may not have been aware of all the possibilities provided by `tabulate`, `table`, `tabstat`, and `tabdisp`. The perhaps forbidding label “programmer’s command” should not stop you exploiting the power of `tabdisp`. Second, the historical accident that something is not wired into official Stata need not inhibit you from devising your own kinds of tables. Once the preparation has been worked out, the presentation is often straightforward.

5 References

- Cox, N. J. 2002a. Speaking Stata: How to move step by: step. *Stata Journal* 2(1): 86–102.
- . 2002b. Speaking Stata: How to face lists with fortitude. *Stata Journal* 2(2): 202–222.
- . 2002c. Speaking Stata: On numbers and strings. *Stata Journal* 2(3): 314–329.
- . 2002d. Speaking Stata: On getting functions to do the work. *Stata Journal* 2(4): 411–427.

- . 2003. Speaking Stata: Problems with lists. *Stata Journal* 3(2): 185–202.
- Newson, R. 2002. Parameters behind “nonparametric” statistics: Kendall’s tau, Somers’ D and median differences. *Stata Journal* 2(1): 45–64.
- . 2003. Confidence intervals and p-values for delivery to the end user. *Stata Journal* 3(3): 245–269.
- Smeeton, N. and N. J. Cox. 2003. Do-it-yourself shuffling and the number of runs under randomness. *Stata Journal* 3(3): 270–277.
- Tukey, J. W. 1977. *Exploratory Data Analysis*. Reading, MA: Addison–Wesley.
- Wall, L. and R. L. Schwartz. 1990. *Programming Perl*. Sebastopol, CA: O’Reilly.

About the Author

Nicholas 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 co-authored fourteen commands in official Stata. He was an author of several inserts in the *Stata Technical Bulletin* and is Executive Editor of the *Stata Journal*.