The Stata Journal (2020)
20, Number 4, pp. 1016–1027 DOI: 10.1177/1536867X20976341

Stata tip 139: The by() option of graph can work better than graph combine

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

1 by() option of graph or graph combine for paneled figures?

Stata users produce graphs (often called figures) for their presentations and publications. Many such figures are composites with two or more separate panels, so that figure 1 is a composite of figure 1a, 1b, 1c, and 1d. There are two main ways to create such composites in Stata: using a `by()` option or using `graph combine` on previously created graphs. There are also commands such as `graph matrix` whose purpose is to create plots with multiple panels directly.

A `by()` option is likely to seem obvious when a grouping variable is already defined, as when you go something like

```
. sysuse auto
. scatter mpg weight, by(foreign)
```

to get separate scatterplots of `mpg` (miles per gallon) versus `weight` for domestic and foreign cars, domestic cars being those made in the United States and foreign cars being those made outside. The result of those commands may already be familiar to you. If not, running the commands just given is an excellent simple exercise.

Using `graph combine` is most obvious when graphs created separately nevertheless have point in being presented together. There might be some pedagogic or even polemical purpose to comparing a histogram, a box plot, a quantile plot, and a kernel density estimate as ways of showing a univariate distribution, each with advantages and disadvantages. It is worth flagging that paneling that is 2×2, 3×3, and so on can look good, although problems with odd numbers of panels such as 3, 5, or 7 will occur with their own obstinate frequency.

The aim of this tip is to push a little at this distinction, as less clear-cut than it seems. Sometimes, a graph will be improved by seeking a different data layout in which a `by()` option ensures more uniformity and consistency of style than will result from using `graph combine`.

2 Example 1: Confidence intervals for means of different variables

A standard plot in many fields compares sample means of a variable for two or more conditions within a framework of confidence intervals or other indications of uncertainty about those means. The word *confidence* here is, as every good text or course should explain, a term of art, rather than an expression of the conviction or intensity of belief a researcher is entitled or expected to hold about variability. A personal suspicion that *diffidence intervals* would be a better term is far from original. Almost a century ago, Fisher (1925, 10) wrote of "our mental confidence or diffidence" in making inferences, although he was not writing about confidence intervals *avant la lettre*; he was flagging the concept of likelihood.

A natural extension when there are several variables of interest is to show various such plots side by side.

An excellent way to get such plots is through `statsby` and `ci` (Cox 2010). The reduction to a new dataset with means and bounds for confidence intervals is immediate. There is some inefficiency in doing that again and again, as we will do here, but coding otherwise is likely to be more time consuming. The default confidence level with `ci` and more generally is 95%.

Let us suppose that—using the auto data again, and now for real—we want to show, side by side, plots with means and confidence intervals for price, miles per gallon, weight, and length for those domestic and foreign cars.

The plan here is to just loop over four variables, each time reading in the data, producing a reduced dataset and drawing and saving each graph for later combination. For an introduction to loops, and their use of local macros, see Cox (2020).

```
. tokenize "price mpg weight length"
. set scheme sj
. forvalues k = 1/4 {
  2. sysuse auto, clear
  3. if "``k´´" == "price" label var price "Price (USD)"
  4. local which : variable label ``k´´
  5. if "`which´" == "" local which "``k´´"
  6. statsby, by(foreign) clear: ci mean ``k´´
  7. twoway rcap lb ub foreign || scatter mean foreign, ylabel(, angle(h))
> xscale(r(-0.2 1.2)) xlabel(0 1, tlcolor(none) valuelabel) legend(off)
> subtitle("`which´") name(g`k´, replace)
  8. }
  (output omitted)
. graph combine g1 g2 g3 g4
```

The big idea is to loop over variables, producing a graph for each variable to be put together by the final `graph combine`. But I chose to loop directly over integers 1 to 4, which I then used in naming the graphs. That choice is a small matter of style but also informed by experience. If I use similar code for similar problems, there is less code to revise if I cast the problem as a loop over integers rather than as a loop over variable names.

Know that `tokenize` parses its argument into words, parsing in this case a list of variable names on spaces. Then, the distinct words are assigned to local macros named 1 up. Thus, local macro with name `1` has contents `price`, and local macros with names `2`, `3`, and `4` have contents `mpg`, `weight`, and `length`, respectively. So that is how we can, at the same time, loop over 1 to 4 and the first, second, third, and fourth variable names, which are held within local macros named 1 to 4.

The trickiest detail here is the use of nested macro references, specifically `` ``k'' ``. An easy analogy is with nested expressions in elementary algebra, such as $\{a-(b-c)\}$, where one quickly learns to evaluate first what is inside the innermost parentheses and then work outward. As local macro `k` loops over the values `1` to `4`, the nested reference `` ``k'' `` becomes in turn `` `1' ``, `` `2' ``, `` `3' ``, and `` `4' ``, and so a reference to the local macros `1` to `4`, which were earlier assigned as contents the names `price` through `length`.

Some of the details here are specific to the example. It seems to me good practice to show the units of price, here U.S. dollars, especially for an international readership. There are two distinct values of `foreign`, 0 and 1, but we want to see the corresponding value labels instead. Adding a little space on either side with `xscale()` is an aesthetic choice, as is suppressing the (visibility of the) tick by setting its color to none (Cox and Wiggins 2019).

The other details are more general. Each graph shows what it is about in a subtitle. For that subtitle, my preference is to use a variable label. All the variables here do in fact have variable labels, but the code also shows technique for using the variable name, rather than the variable label, when the latter is not defined.

Why did I use the `replace` in naming those graphs? We haven't used those names before in this tip. It is a gesture to human frailty. I almost never get the code exactly right first time round. There is always some detail I have forgotten, or got slightly or very wrong. Indeed, it is better to let Stata tell you quickly what needs changing. I started computing in the middle 1970s, when a job was submitted to the university computer (there was only one such) on a card deck with 80 column cards for data and code you punched yourself. A job would typically be run some hours after submission, and so there was much point to checking your cards repeatedly to try to get them exactly right, because one mistake would mean a delay of several hours before the next version could be tried. Now agonizing about code has less value when Stata, or some other program, will usually find the mistakes much faster than you can.

It is likely that you would prefer a graph scheme different from `sj`, perhaps one that you devised yourself. My personal favorite is `s1color`, although I often tweak away from its defaults.

It would have been entirely possible to write that code directly using variable names. Let's give the code as rewritten that way.

```
. set scheme sj
. foreach v in price mpg weight length {
  2. sysuse auto, clear
  3. if "`v´" == "price" label var price "Price (USD)"
  4. local which : variable label `v´
  5. if "`which´" == "" local which "`v´"
  6. statsby, by(foreign) clear: ci mean `v´
  7. twoway rcap lb ub foreign || scatter mean foreign, ylabel(, angle(h))
> xscale(r(-0.2 1.2)) xlabel(0 1, tlcolor(none) valuelabel) legend(off)
> subtitle("`which´") name(g`v´, replace)
  8. }
```

(*output omitted*)

```
. graph combine gprice gmpg gweight glength
```

Some of the choices here are just conventional, whether widespread (many people do it) or personal (it could just be me). Thus, names like `i`, `j`, `k` for looping macros that often start at 1 have been long used in many programming languages and echo even longer-standing mathematical practices. I often use the local macro name `v` in looping over variable names. If you wanted to use longer local macro names, only taste, how much you have to type, and Stata's limit of 31 characters for such names might stand in your way.

Some of that may have been new to you, or else you skimmed and skipped familiar constructs. To focus now on the main theme of this tip, let us see the result of `graph combine` in figure 1.

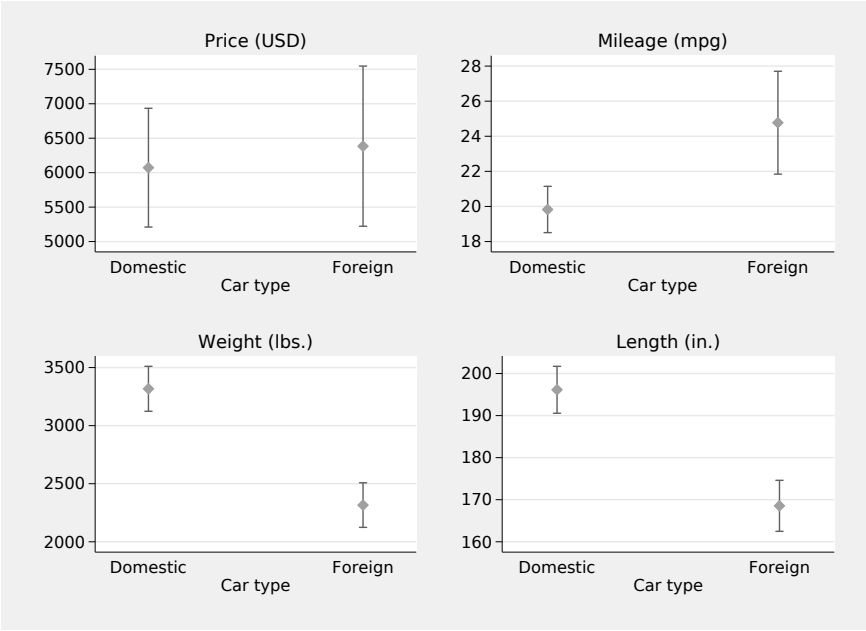

Figure 1. Means and confidence intervals for price, miles per gallon, weight, and length for domestic and foreign cars from the auto dataset. So far, so good, but the y axes stand out as using different spacing.

That does look quite good to me, but it could still be better. The different variables come with different units of measurement and—more crucially—different magnitudes, and the y axes are not exactly aligned. That is the niggling detail that people often spot quickly. Note also that specifying the `ycommon` option with `graph combine` solves this problem, but not helpfully, while specifying the `xcommon` option does no harm, but is no help either.

To solve this problem of unaligned axes, we turn instead to combining the reduced datasets and then drawing a graph just once using the `by()` option. This is more work, but the scope for improved appearance should make it seem worthwhile.

```
. tokenize "price mpg weight length"
. forvalues k = 1/4 {
  2. sysuse auto, clear
  3. if "``k´´" == "price" label var price "Price (USD)"
  4. local which : variable label ``k´´
  5. if "`which´" == "" local which "``k´´"
  6. local labels `labels´ `k´ "`which´"
  7. statsby, by(foreign) clear: ci mean ``k´´
  8. generate which = `k´
  9. save g`k´, replace
 10. }
```

(*output omitted*)

```
. append using g1 g2 g3
```

(*output omitted*)

```
. label define which `labels´
. label values which which
. list, sepby(which)
```

(*output omitted*)

```
. set scheme sj
. twoway rcap lb ub foreign || scatter mean foreign,
> by(which, yrescale note("") legend(off)) ylabel(, angle(h)) xscale(r(-0.2 1.2))
> xlabel(0 1, tlcolor(none) valuelabel)
```

The results of `list` are suppressed here, but having a look at the data is a really good idea if you are working through the code yourself (ideally to adapt the idea to some different dataset you care about).

We need to store information on each variable for later use as we loop through them. We want the variable labels of the four variables concerned (or if they do not exist, the variable names) to become value labels for the grouping variable we are going to use. Naturally, for that to be possible, the grouping variable, here called `which`, must be created first.

For this version of the problem, assigning integers 1 to 4 in the order in which we want the panels is definitely a good idea. Otherwise, if the variable names `price mpg weight length` are used as distinct values of `which`, and that variable is fed to `by()`, then those panels will be shown in alphabetical name order. The `by()` option will sort on the distinct values of the variables it sees, which here means alphabetical order for text that was originally variable names. (There is extra detail, which doesn't apply in this example, for underscore or numeric characters.) Alphabetical order has its uses (Flanders 2020), but alphabetical order of panels is rarely what you want, or more precisely, rarely what you should want. Wainer (1984, 1997) mocked unthinking use of alphabetical order in statistical graphics as Austria first! or Alabama first! Scottish readers will want to add Aberdeen first!

Figure 2 is the result. Isn't it better? What it shows is perhaps banal, but here we go: foreign cars typically are more expensive but have better mileage than domestic cars, and they are lighter and shorter on average too.

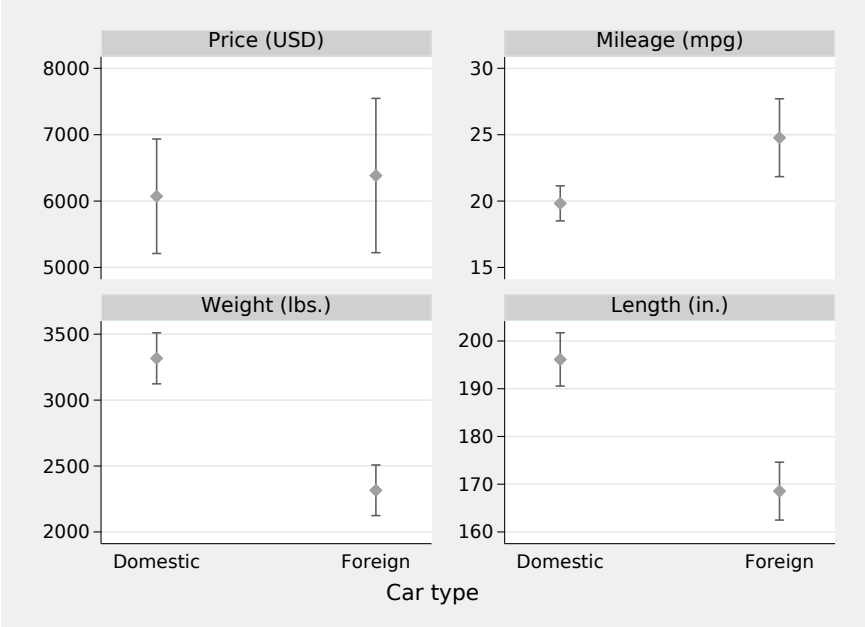

Figure 2. Means and confidence intervals for price, miles per gallon, weight, and length for domestic and foreign cars. It is not obvious from the graph, but a `by()` option was used to improve the presentation of y axes.

3 Example 2: Slicing cyclic time series

Effective graphical representation of intensely cyclical time series can be a challenge, especially if they are long. Many time series from several fields, including economics, epidemiology, and environmental science, are strongly seasonal. Cycles in those and other fields can also be identified with lengths both shorter and longer than a year. Cycles vary from fairly predictable to highly unpredictable, but the same graphical challenge is common to many: do justice to the structure of cycles, which may itself be changing over time, and also reveal any trends or other long-term changes. Stata-based discussion in Cox (2006, 2009) covers several of the graphical ideas that have been floated.

Sunspot numbers are a popular sandbox for enthusiasts in statistical graphics (for example, Cleveland [1993, 1994]; Wilkinson [2005]). Here I use annual averages downloaded on 3 June 2020 from http://www.sidc.be/silso/home with grateful acknowledgment to WDC-SILSO, Royal Observatory of Belgium, Brussels. The data are available with the files for this tip.

A standard line plot is not crazy but not especially helpful either (figure 3).

```
. use sunspots, clear
. line sunspots year
```

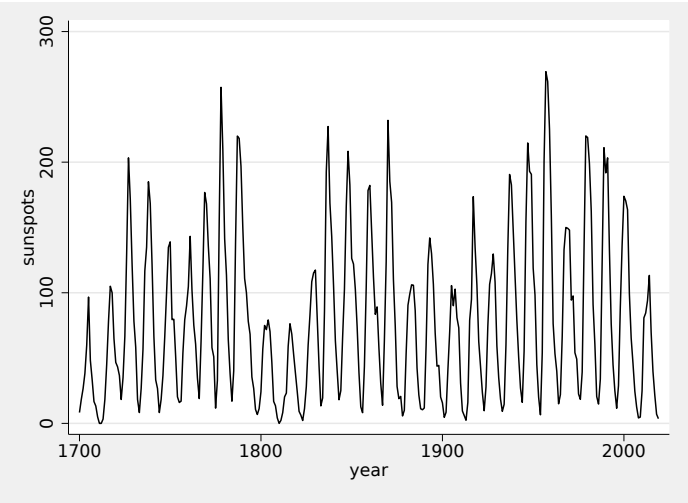

Figure 3. A common or garden line chart for mean sunspot numbers, 1700–2019. Strongly cyclical behavior is evident, but the fine structure of cycles is not especially clear from the roller-coaster display.

Common advice is to change the aspect ratio (ratio of graph height to graph width), which often can work well enough (Cox 2004). Changing the aspect ratio is sometimes discussed as if novel, but it was evidently familiar to Fisher (1925, 31) and doubtless earlier yet. Here the effect is a little disappointing. You may have found low aspect ratios used in long, concertina like graphs inserted in books, but changes in printing have made those much less common.

```
. line sunspots year, aspect(0.2)
```

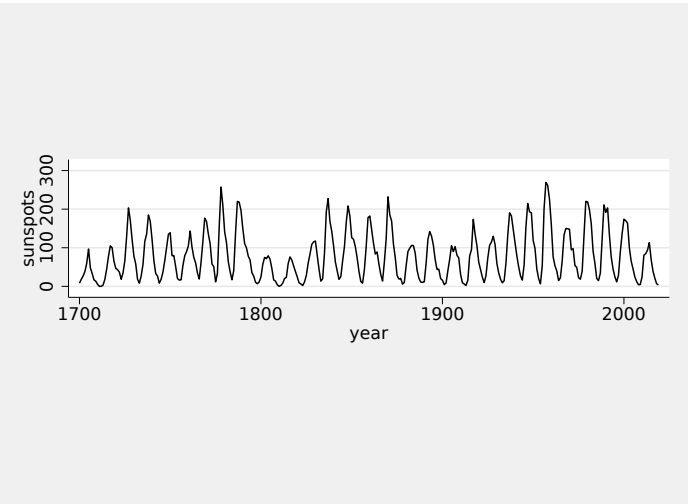

Figure 4. Changing the aspect ratio does not help much here

A twist on the same idea is to slice the series and use several panels for the different slices. Cleveland (1993, 1994) called these "cut-and-stack plots". Cox (2006) reported a Stata command `sliceplot`, but the novelty here is a belated recognition that often no such command is needed. Code comes first and then explanation.

```
. generate slice = ceil(4 *_n/_N)
. line sunspots year, by(slice, note("") cols(1) xrescale)
> ylabel(, angle(h)) xtitle("")
> subtitle("", position(9) nobox nobexpand fcolor(none))
```

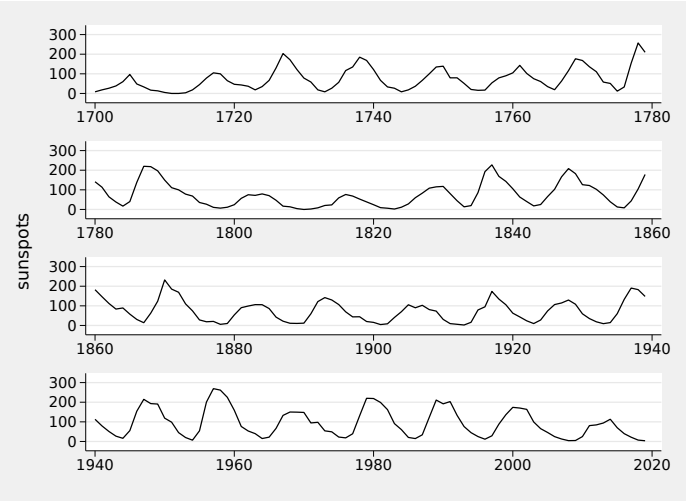

Figure 5. Slicing a long cyclical time-series plot makes the structure of the data clearer. Note the common asymmetry of cycles in which rising limbs are often steeper than falling limbs. A `by()` option was used to create this plot.

We create a slice variable that runs from 1 to 4, but naturally you should use whatever different value of 4 makes sense for your problem. (That recycles a mild joke associated with the great probabilist, William Feller, 1906–1970.)

To spell it out, the presumption is that the time series is already sorted on a time variable. The inbuilt observation number `_n` varies from 1 to the inbuilt number of observations in the dataset `_N`, so `4 * _n/_N` varies from almost 0 to 4. Rounding up with the `ceil()` function produces a variable containing a slice counter with integer values from 1 to 4.

That variable is just a means to an end, and so we suppress all evidence that it exists at all, blanking out both the `note()` that names it and the `subtitle()` that shows its distinct values. The `xrescale` suboption is essential for a readable graph: if you doubt that, see what happens if you omit it. Making the y-axis labels horizontal and removing the default `xtitle()` (which would be `year` in this example) are matters of taste. Aligning in one column of panels is a good idea to keep the aspect ratio low.

The payoff is not just in terms of a clearer graph. It becomes evident that cycles are often asymmetric, with steeper rising limbs than falling limbs. Explaining why is a problem for researchers in solar physics.

4 The art that conceals art

A duty to be clear is a higher virtue than using clever tricks, but clever tricks are always worth knowing about. The trick here implies more work by the user, and the payoff has to be judged case by case. It can be explained as art concealing art, in using a `by()` option but suppressing the visible evidence of doing it that way.

Further uses of the same device can be found in the community-contributed commands `multiline` (Cox 2017b) and `multidot` (Cox 2017a), downloadable from Statistical Software Components Archive (look at `help ssc` if that resource is new to you).

References

Cleveland, W. S. 1993. *Visualizing Data*. Summit, NJ: Hobart.

———. 1994. *The Elements of Graphing Data*. Rev. ed. Summit, NJ: Hobart.

Cox, N. J. 2004. Stata tip 12: Tuning the plot region aspect ratio. *Stata Journal* 4: 357–358. https://doi.org/10.1177/1536867X0400400313.

———. 2006. Speaking Stata: Graphs for all seasons. *Stata Journal* 6: 397–419. https://doi.org/10.1177/1536867X0600600309.

———. 2009. Stata tip 76: Separating seasonal time series. *Stata Journal* 9: 321–326. https://doi.org/0.1177/1536867X0900900211.

———. 2010. Speaking Stata: The statsby strategy. *Stata Journal* 10: 143–151. https://doi.org/10.1177/1536867X1001000112.

———. 2017a. multidot: Stata module for multiple panel dot charts and similar. Statistical Software Components S458376, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458376.html.

———. 2017b. multiline: Stata module for multiple panel line plots. Statistical Software Components S458369, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458369.html.

———. 2020. Speaking Stata: Loops, again and again. *Stata Journal* 20: 999–1015. https://doi.org/10.1177/1536867X20976340.

Cox, N. J., and V. Wiggins. 2019. Stata tip 132: Tiny tricks and tips on ticks. *Stata Journal* 19: 741–747. https://doi.org/10.1177/1536867X19874264.

Fisher, R. A. 1925. *Statistical Methods for Research Workers*. Edinburgh: Oliver & Boyd.

Flanders, J. 2020. *A Place for Everything: The Curious History of Alphabetical Order*. London: Picador.

Wainer, H. 1984. How to display data badly. *American Statistician* 38: 137–147. https://doi.org/10.1080/00031305.1984.10483186.

———. 1997. *Visual Revelations: Graphical Tales of Fate and Deception from Napoleon Bonaparte to Ross Perot*. New York: Copernicus.

Wilkinson, L. 2005. *The Grammar of Graphics*. 2nd ed. New York: Springer.