



*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](mailto: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: Automating axis labels: Nice numbers and transformed scales

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

**Abstract.** Two common problems with graph axis labels are to decide in advance on some “nice” numbers to use on one or both axes and to show particular labels on some transformed scale. In this column, I discuss the `nicelabels` and `mylabels` commands, which address these problems. The first command is new to Stata, and the second is a revision of a previously published command. I also survey the `myticks` command for tick placement. In all commands, the main output is a local macro in the calling program’s space, in the interest of promoting automation in do-files and programs.

**Keywords:** gr0092, `nicelabels`, `mylabels`, `myticks`, axis labels, axis ticks, axis scales, transformations, graphics

## 1 Introduction

Consider a routine Stata graph, such as figure 1 obtained with Stata’s auto data. If you wish, you can read in the dataset yourself and follow the thread actively by typing in commands.

```
. sysuse auto
(1978 automobile data)
. scatter mpg weight, ms(0h)
```

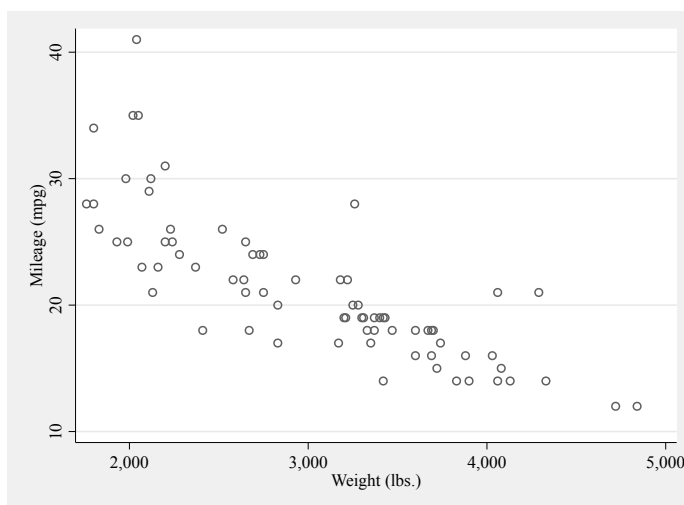


Figure 1. A common or garden scatterplot showing how `graph twoway` chooses “nice” axis labels automatically

Users of Stata’s graphics commands quickly learn that there is an automatic choice of axis labels for such graphs. In `auto.dta`, `mpg` varies from 12 to 41 (miles per gallon), and `weight` varies from 1760 to 4840 (pounds). With `scatter` and many other commands, you get “nice” axis labels by default. For this graph, the *y*-axis labels are 10(10)40 for `mpg`, and the *x*-axis labels are 2000(1000)5000 for `weight`. The notation *start(step)stop* implies numeric values running from *start* to *stop* in increments of *step*.

Here “nice” is a term of art as well as a simple English word. Part of the business of this column is to give some discussion of what it means for axis labels. As in the rest of life, niceness can be easy to recognize but harder to define precisely or universally.

Users also should quickly learn that you can reach in and override the default label choices if you prefer something else. For example, you might want axis labeling to start at 0, include more labels, or differ in some other way from the default. If so, spell out, say, `ylabel(0(10)40)` or `ylabel(0(5)40)`. Such flexibility is helpful and indeed essential for statistical and related purposes. Examples could be multiplied, but let’s just also mention showing critical levels (say, legal or clinical or physical thresholds) or summary values (say, mean or median) as particular axis labels. From this point of view, 32° F as the freezing point on the Fahrenheit scale could be a helpful label if that temperature fell into the range of some data.

Sections 2 and 3 of this column focus on needs where neither the default nor the flexibility of specifying different choices on the fly is quite the answer. A problem often raised by users is wanting a series of graphs to follow the same label choices—because left to follow its own rules, `graph` might show different labels for different subsets of the data. But if we know the limits of a variable in principle, or we can work them out in practice for the entire dataset, then we should be able to identify some standardized

choices. A new command, `nicelabels`, uses simple algorithms to suggest automatically “nice” labels, given some information. The possibilities allowed by `nicelabels` do not reach as far as more subtle (and inevitably more complicated) algorithms. Hence, the area remains wide open for further work.

Users frequently want to show values on transformed scales. By far the most common example is that one or even both axes use logarithmic scales. `graph` provides specific support for such scales, using the `yscale(log)` and `xscale(log)` options. That said, the accompanying choice of labels is often awkward. The `niceloglabels` command (Cox 2018, 2020a) was published in an earlier column as an alternative. Its users do need to specify a style, as label styles such as 1, 2, 5, 10 or 1, 3, 10 or powers of 2 may each be appropriate or appeal otherwise. `niceloglabels` offers a choice between those and yet other styles.

Although less common than logarithmic scales, various other nonlinear scales are possible in statistical applications. Square root, cube root, reciprocal, and logit scales are some leading examples. In principle, you just need to calculate what you want to show before the graph is drawn. The more challenging task is to show text labels on the original scales, just as with `yscale(log)` or `xscale(log)`, values like 1, 10, or 100 would often be shown as axis labels, not the corresponding logarithms.

Sections 4 and 5 here discuss a generic solution for labels on transformed scales, which is to loop over the text to be shown as axis labels and to work out where to put it in each case (Cox 2008b, 2012). To this end, the community-contributed command `mylabels` has been under development since 2003. Early mentions include Cox (2004, 2005a). A revised version is now published formally in this issue. The idea of transformation implemented is quite wide and includes scope to add text as a prefix or suffix to axis labels.

Any readers new to Stata or more familiar with other software might like a confirmation that Stata’s axis labels are not text titles for an axis but any text shown at axis ticks. Other terms that can be found include tick mark label (Cleveland 1985, 23; 1994, 24; Robbins 2013, 156) and tick label (Koponen and Hildén 2019, 206).

## 2 nicelabels in practice

### 2.1 nicelabels fed minimum and maximum

`nicelabels` supports two main syntaxes, one in which a variable is named and one in which two numbers are specified, typically some minimum and maximum. The second syntax seems likely to be less often used, but it is a good place to start the story.

```
. nicelabels 142 233, local(foo)
step:      20
labels:    140 160 180 200 220 240

. nicelabels 142 233, local(foo) tight
step:      20
labels:    160 180 200 220
```

`nicelabels` suggests, by default, what Heckbert (1990) called “loose” labels, which often go beyond the minimum and maximum suggested. Conversely, the `tight` option is not so generous. The choice between them can be regarded as partly technical detail and partly a matter of taste. Using the `tight` option promotes full use of the plot region, typically reducing the amount of space surrounding data elements on a graph.

As the examples show, `nicelabels` suggests power-of-10 multiples of 1, 2, and 5 and suggests “about 5” labels by default, here 6 and 4, respectively, for the `loose` and `tight` options. If you are interested in the algorithm, take a look at Heckbert (1990) or the code revealed after installation by

```
. viewsource nicelabels.ado
```

Stata programmers will appreciate the close correspondence between Heckbert’s original pseudocode and C code and the Mata code used inside `nicelabels`.

The default of “about 5” labels echoes widespread statistical practice, exemplified in the suggestion by Cleveland (1985, 39; 1994, 39) of 3 to 10 labels on any axis.

Naturally, you may ask for more labels. You should guess that, rather than use a step of 20, the code might suggest a step of 10 if you ask for moderately more labels. Here adding the option `nvals(10)` leads to suggestions of 11 and 9 labels for the 2 basic choices. Although those results should seem plausible as being “about 10”, the number of values is a suggestion, not an instruction, so far as the code is concerned.

```
. nicelabels 142 233, local(foo) nvals(10)
step:      10
labels:    140 150 160 170 180 190 200 210 220 230 240
. nicelabels 142 233, local(foo) nvals(10) tight
step:      10
labels:    150 160 170 180 190 200 210 220 230
```

`nicelabels` insists on a `local()` option, naming a local macro to hold the results. If the idea of a local macro is new to you, consider it as a bag or box within your program space that contains text (which, as indicated, could be numeric characters and spaces). Among various lengthier introductions to local macros is a recent Speaking Stata column (Cox 2020b, 2021a).

This insistence on a local macro is intended to be in your own best interests, encouraging the use of that local macro within a later command, either interactively or in a do-file or program. Because `nicelabels` displays the labels suggested, you could also copy and paste them within Stata or even type them out again, but neither is recommended as ideal practice. The goal of self-documenting, reproducible code leads to emphasis on commands that run directly from a script. In this case, we have no data in sight, and a facetious macro name `foo` does no harm. As in the rest of Stata, an existing local macro will readily be overwritten. We kept using the same macro name, and there was no complaint from Stata. Overwriting local macros is usually a feature but may lead to a tiny loss of results if you are a little careless.

## 2.2 nicelabels fed a numeric variable name

Let's now switch to the major syntax of **nicelabels**, based on asking the command to look at a numeric variable. Like **auto.dta**, **census.dta**, now used as a sandbox, is bundled with Stata, so you can follow along by typing the same commands.

For 50 states of the United States in 1980, let's imagine working with median ages. The point is not that **graph** will not do a good job by itself but just to show how to work with **nicelabels**, given a variable of interest. As you might guess, **nicelabels** fires up **summarize** to find the range of a variable from minimum to maximum, but it does not show the results, so we will do that here. We see that the range is maximum – minimum =  $34.7 - 24.2 = 10.5$ .

```
. sysuse census, clear
(1980 Census data by state)
. summarize medage
```

Variable	Obs	Mean	Std. dev.	Min	Max
medage	50	29.54	1.693445	24.2	34.7

You feed the variable name to **nicelabels**, and here it makes various suggestions, depending on choice of the default or of options that you specify.

```
. nicelabels medage, local(agela)
step:      5
labels:    20 25 30 35
. nicelabels medage, local(agela) tight
step:      5
labels:    25 30
. nicelabels medage, local(agela) nvals(10)
step:      2
labels:    24 26 28 30 32 34 36
. nicelabels medage, local(agela) nvals(10) tight
step:      2
labels:    26 28 30 32 34
```

With its default, **nicelabels** is clearly struggling to find many nice numbers in the variable range. My inclinations for this variable run to a step of 2, not 5. You can check for yourself by plotting **medage** against any other variable that, left to itself, **graph twoway** agrees with that step.

Let's use **nicelabels** to make our own version of a quantile plot, using a rank from 1 for the state with the youngest median age to 50 for the state with the oldest. We will run with age labels 24(2)36:

```
. nicelabels medage, local(agela) nvals(10)
step:      2
labels:    24 26 28 30 32 34 36
. egen rank = rank(medage), unique
. scatter medage rank, ylabel(`yla', ang(h)) xlabel(1 5(5)50) xtitle(Rank) ms(Oh)
```

Figure 2 shows the result.

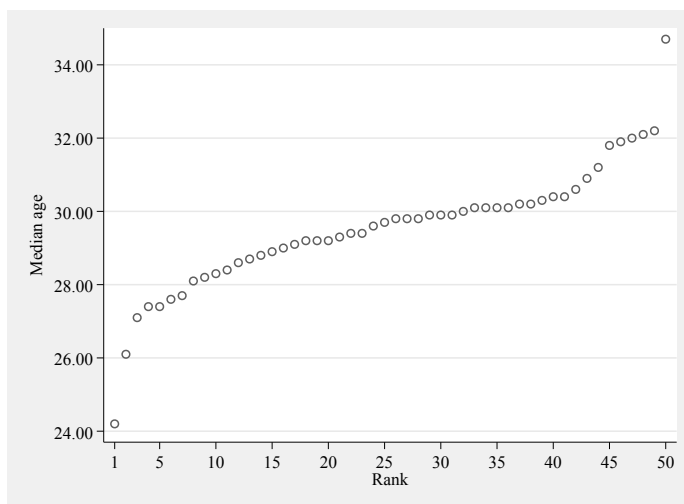


Figure 2. Quantile plot for median ages of 50 U.S. states in 1980. `nicelabels` was used to obtain labels for median age.

A small lesson here is that the output of `nicelabels` and the display format of `medage`, or any other numeric variable, are quite separate matters. For some reason, `medage` has been assigned a display format of `%9.2f`, specifying display of values to two decimal places, which is one more than the resolution of the data themselves. Thus, we may want to spell out display of integers (zero decimal places). At the same time, the distribution deserves a little annotation. To my eye, states with ranks 1, 2, and 50 stand out from other states.

```
. scatter medage rank, ylabel(`yla', ang(h) format(%2.0f)) xlabel(1 5(5)50)
> xtitle(Rank) ms(0h) || scatter medage rank if inlist(rank, 1, 2, 50),
> ms(none) mlabel(state2) xscale(r(. 52)) legend(off)
```

Figure 3 shows the result. Utah, Alaska, and Florida are unsurprising identifications.

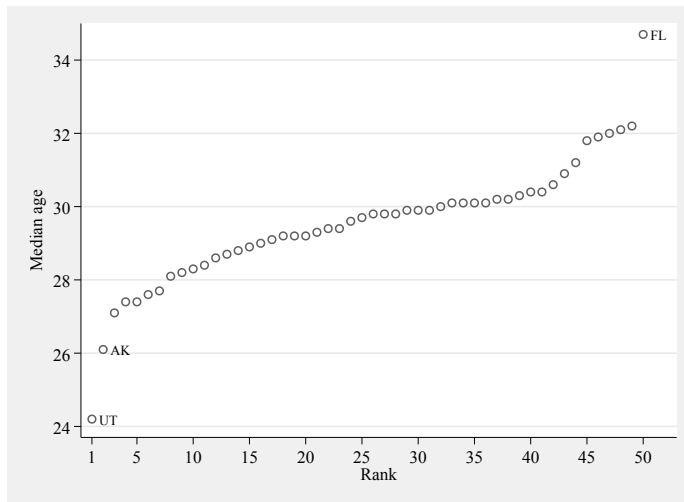


Figure 3. Quantile plot for median ages of 50 U.S. states in 1980. For this version, we specified a display format for the median age labels, and added some annotation.

## 2.3 Further twists

Let's think how to tackle some further twists that may arise. For the sake of variety, examples are taken once more from `auto.dta`.

*An axis should start at zero:* Suppose you have a variable that is always positive, but you want an axis, and so axis labeling, to start at zero. Whether axes should start at zero is sometimes a vexed question in statistical graphics, but for now we will just run with that choice as a decision made. One way to do that is to feed `nicelabels` with 0 as one argument and the empirical maximum from `summarize` as another argument.

```
. sysuse auto, clear
(1978 automobile data)
. summarize mpg, meanonly
. nicelabels 0 `r(max)', local(foo)
step:      10
labels:    0 10 20 30 40 50
```



Note that using the maximum from the `summarize` command with options such as `ylabel(0(10)`max')` or `ylabel(0(20)`max')` or `ysc(0 `max') yla(#6)` may be as or more effective, particularly if you have a clear idea in advance of a good step size. In either case, `local max = r(max)` should be assigned just after `summarize`.

*Axis labels should include empirical minimum and maximum:* A possibility shown by Tufte (2001, 149) is that the lowest and highest labels are the empirical minimum and maximum; the others are all “nice”. The trick here resembles that in the previous example—to get extremes from `summarize` but to let `nicelabels`, `tight` do the rest of the work.

```
. . nicelabels mpg, tight local(yla)
step:      10
labels:    20 30 40
. summarize mpg, meanonly
. local yla `yla' `r(min)' `r(max)'
. nicelabels weight, tight local(xla)
step:      1000
labels:    2000 3000 4000
. summarize weight, meanonly
. local xla `xla' `r(min)' `r(max)'
. scatter mpg weight, xla(`xla') yla(`yla', ang(h)) ms(0h)
```

Figure 4 shows the result. The example is candid in showing a limitation of the idea: it can seem a little awkward at best if a minimum or maximum label is very close to a nice label.

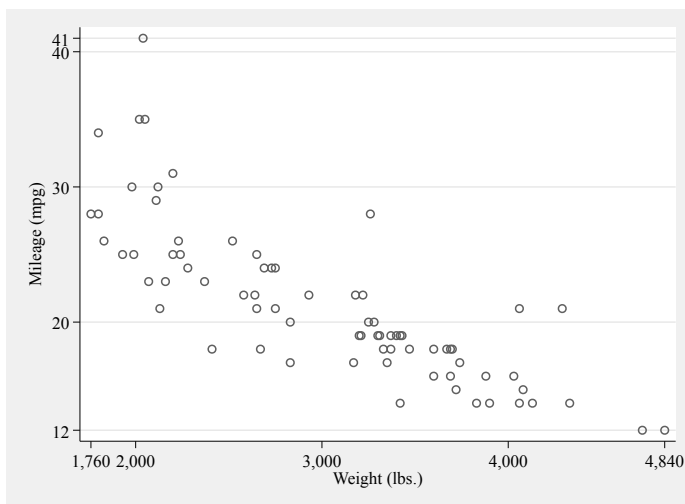


Figure 4. A scatterplot with nice labels together with empirical minimum and maximum on each axis

Here we are exploiting how easy it is to redefine a local macro. `nicelabels` defines a local macro. As mentioned, you can think of such a macro as being like a bag that you may define and redefine as you please, here just by adding other values to the bag. Note that it is crucial to specify the local macro persona of a saved result, for example, ``r(min)'` rather than `r(min)`. Otherwise, you just add the result name `r(min)`, not its current value, to the macro.

*Controlling the number of labels:* Redefining a macro is also crucial to the next twist here. In a do-file or program, we might want to insist on at least a certain number of axis labels. One idea is simple: Try out the `nicelabels` default and count how many labels are suggested; if that is not enough, fire up `nicelabels` again with the `nvals()` option. This idea could be coupled with trying both default and `tight` options or just one of the two choices.

As previously, but now finding that three labels are too few:

```
. nicelabels mpg, tight local(yla)
step:      10
labels:    20 30 40
. if wordcount("`yla'") < 5 nicelabels mpg, tight local(yla) nvals(10)
step:      5
labels:    15 20 25 30 35 40
```

`wordcount()` does what it says—counts words. In Stata, words are separated by white space, so every label is counted as a separate word.

Once again: If you are working interactively, you could see for yourself that only three labels are suggested the first time around and move immediately to asking for more. The point about using `wordcount()` with a local macro result is allowing such choice to be automated.

## 2.4 Further remarks

The aim of `nicelabels` is not to repeat, still less to replace, what is already easy in Stata, or to satisfy all needs. Thus, if quantiles are being shown, it may be that labels 0.5, 0.25, 0.75, 0.125, 0.375, 0.625, 0.875, etc., appeal for a probability scale as corresponding to median, quartiles, octiles, and so forth. These labels are easily produced either by spelling them out or by using step sizes such as 0.25 or 0.125. Those labels are not nice in Heckbert's sense, but they do have some statistical appeal. The same issue arises with a percent scale from 0 to 100 whenever labels like 25, 50, and 75 seem desirable. Even more generally, if you know a good step size, it will often be easiest to specify it directly.

Consider, for example, a graph with two years' worth of monthly data and a monthly date on the *X* axis. `nicelabels` has no special understanding of time variables and will just suggest steps of 5 or 10 months. In contrast, you will find that official commands like `line`, `tsline`, and `xtline` will suggest tailored steps of six months, given only minimal hints through formatting the date variable with a monthly date display format

(and declaring it as time variable to `tsset` or `xtset`). If you need to write code to standardize date label choices, it can help to remember that the first half-year, quarter, or month in a year yields 0 on being fed to `mod(date, 2)`, `mod(date, 4)`, or `mod(date, 12)`, respectively. For more on `mod()`, see Cox (2007a).

Less common—but important to their users—are variables on a circular scale such as compass bearings (in degrees from North, on a scale between 0 and 360°) or times of day (in hours up to 24). Here useful steps are likely to be 90 or 45° or 6 or 3 hours, not (for example) 100 or 50° or 5 or 10 hours.

This implementation of `nicelabels` focuses on the ideas covered by Heckbert (1990). There is a wider literature both before and after that date. Wilkinson (2005, 95–97) and Talbot, Lin, and Hanrahan (2010) have spelled out how fully automated choice entails a delicate tradeoff between several desiderata, chiefly simplicity, coverage, granularity, and legibility. Talbot, Lin, and Hanrahan (2010) give a full survey of the problem.

## 3 nicelabels in principle

### 3.1 Syntax

```
nicelabels varname [if] [in], _local(macname) [tight nvals(#)]
```

```
nicelabels #1 #2, _local(macname) [tight nvals(#)]
```

### 3.2 Description

`nicelabels` suggests axis labels that would look nice on a graph using a general scale. It can help when, for example, you want to choose the same labels for a series of graphs. Results are put in a local macro for later use.

“Nice” is a little hard to define but easier to recognize. This command follows common practice in general, and Heckbert (1990) in particular, in selecting power-of-10 multiples of 1, 2, or 5 that are equally spaced.

See Hardin (1995) for an earlier implementation in Stata of such ideas. Hardin (1995) implemented what Heckbert calls a “loose” definition of labels, while this command allows also a “tight” definition.

There are two syntaxes. In the first, the name of a numeric variable must be given. In the second, two numeric values are given, which will be interpreted as indicating minimum and maximum of an axis range. Those two values can be given in any order.

### 3.3 Options

`local(macname)` inserts the specification of labels in local macro *macname* within the calling program's space. Hence, that macro will be accessible after `nicelabels` has finished. This is helpful for later use with `graph twoway` or other graphics commands. `local()` is required.

`tight` tends to pull in as compared with the default and so not suggest labels beyond the minimum or maximum. This option may often be combined with the `nvals()` option (see below).

`nvals(#)` suggests the number of values to be labeled. The default is `nvals(5)`. Most commonly, you may wish to suggest more values, say, `nvals(10)`. As said, this is a suggestion, not an instruction.

## 4 mylabels in practice

As explained in the introduction, `graph` offers special support for logarithmic scales. That leaves the question of how to support any other nonlinear scales wide open. For example, I make occasional use of square root, cube root, reciprocal, logit, folded root, neglog, and asinh scales, to mention only seven transformations, as well as various quantile (inverse cumulative distribution) scales. Even as a personal list, that is not complete, and others could add yet other scales. The prospect of a separate command or function for each possibility, or even the most common or important according to someone's judgment, is daunting to users and programmers alike.

A better idea, explicit in Royston (1996) in a Stata context, is to expect users to calculate what numbers they want to show and to focus on the question of what axis labels should be shown whenever such numbers are graphed. This is what `mylabels` offers. A defense of the command name is that a bland name mentioning labels might be hard to find in a `search`.

For a first example, we use `auto.dta` again. As commonly remarked, miles per gallon, although a conventional measurement of car performance in the United States, Britain, and some other countries, is mechanically the wrong way round for relating to weight. The reciprocal, gallons per mile, is worth examining. As a matter of convenience, we multiply by 1,000. The calculation is easy, but the harder bit is labeling axes in terms of the original measurements. (Other way round, if gallons per 1,000 miles appeals as a natural scale, then labeling will not be an issue.)

```
. sysuse auto, clear
(1978 automobile data)

. scatter mpg weight

. generate gpm = 1000/mpg

. mylabels 15(5)40, myscale(1000/@) local(myyla)
66.6666666666667 "15" 50 "20" 40 "25" 33.3333333333334 "30" 28.5714285714285
"35" 25 "40"
```

```
. scatter gpm weight, yla(`myyla', ang(h))
> ytitle(Miles per gallon (reciprocal scale)) ms(0h)

. myticks 12/41, myscale(1000/@) local(myyti)
83.33333333333333 76.92307692307692 71.42857142857143 66.66666666666667 62.5
58.8235294117647 55.55555555555556 52.63157894736842 50 47.61904761904762
45.45454545454545 43.47826086956522 41.66666666666666 40 38.46153846153846
37.03703703703704 35.71428571428572 34.48275862068966 33.33333333333334
32.25806451612903 31.25 30.3030303030303 29.41176470588235 28.57142857142857
27.77777777777778 27.02702702702703 26.31578947368421 25.64102564102564 25
24.39024390243902

. scatter gpm weight, yla(`myyla', ang(h))
> ytitle(Miles per gallon (reciprocal scale)) ymtic(`myyti') ms(0h)
```

Figure 5 is the result.

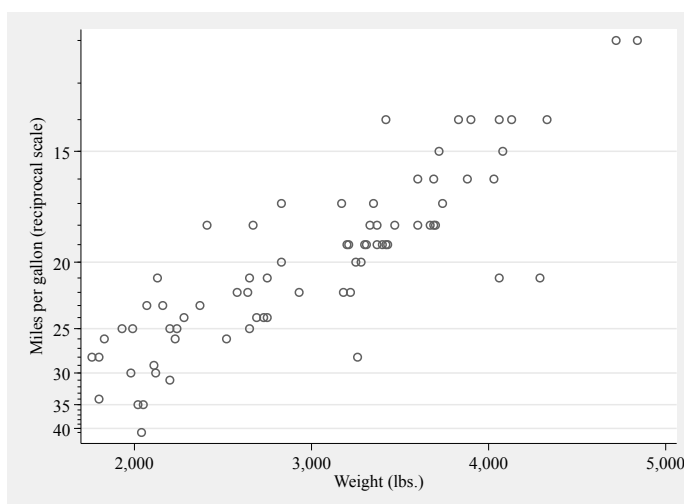


Figure 5. Gallons per 1,000 miles versus weight, but labeled in terms of miles per gallon using `mylabels`

For `mylabels`, there are three essentials.

*The labels you want to see:* Often, but not necessarily, such labels are specified as *numlists*. We have already seen examples of the step notation, used just now for 15(5)40. More information on allowed notation is given at `help numlist`.

*The scale you are working on:* Here the `@` character is a placeholder for the variable in question. You do not need to have a particular variable in mind, or you could be working out good labels that should be useful with several similar variables. Here you are saying that you want 15 20 25 30 35 40 as labels, but in each case the value being plotted is 1,000 divided by each of those.

*A place to put the specification:* As with `nicelabels`, `mylabels` does display the syntax for your desired labels (as text) and where `graph` should put them, but the command insists on a local macro name to encourage good coding practices.

Although it is not so often wanted, `mylabels` has a sister command, `myticks`, to place ticks on an axis. Axis ticks can help in conveying a sense of how far an original scale is being stretched or squeezed when transformed, given that values are shown in the same space.

Transformations do not have to be nonlinear to be useful. A large group of linear transformations concerns mappings between different units of measurement, including simple scalings such as between proportions from 0 and 1 and percents from 0 to 100. The next example takes data for U.S. cities on average temperatures in Fahrenheit but shows labels on a Celsius scale. Recall that Fahrenheit temperatures  $F$  and Celsius temperatures  $C$  are related by  $F = 32 + (9/5)C$ . After a quick look at `summarize` results and some rough calculations, I decided on the labels I wanted to see. The result of the last graph is shown in figure 6.

```
. webuse citytemp, clear
(City temperature data)
. summarize
```

Variable	Obs	Mean	Std. dev.	Min	Max
division	956	5.135983	2.694249	1	9
region	956	2.623431	1.057724	1	4
heatdd	953	4425.533	2199.605	0	10816
cooldd	953	1240.413	937.6679	0	4389
tempjan	954	35.74895	14.18813	2.2	72.6
tempjuly	954	75.05377	5.495036	58.1	93.6

```
. scatter tempjuly tempjan
. mylabels 10(5)35, myscale(32 + (9/5)* @) local(myyla)
50 "10" 59 "15" 68 "20" 77 "25" 86 "30" 95 "35"
. mylabels -15(5)20, myscale(32 + (9/5)* @) local(myxla)
5 "-15" 14 "-10" 23 "-5" 32 "0" 41 "5" 50 "10" 59 "15" 68 "20"
. scatter tempjuly tempjan, yla(`myyla', ang(h)) xla(`myxla') ms(Oh)
> ytitle(Average July temperature ({&degree}C))
> xtitle(Average January temperature ({&degree}C))
```

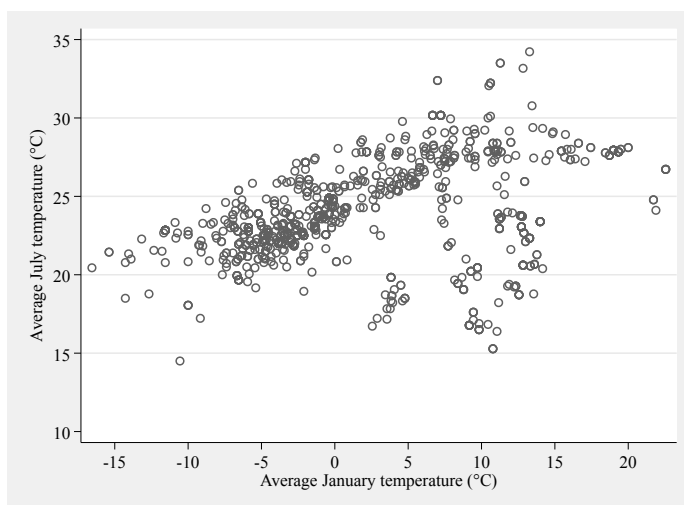


Figure 6. July and January temperatures for various U.S. cities. The original data were on Fahrenheit scale, but `mylabels` is used to show selected labels in Celsius.

Fortuitously, but fortunately, the aspect ratio is about right, namely, that these cities vary much more in January temperature than in July temperature. The ratio of ranges is matched roughly by the aspect ratio of the graph.

Naturally, you could just calculate new variables holding Celsius values. The point is that you do not have to do that. For example, you might have many variables on the Fahrenheit scale, and holding two versions of each variable might seem a little cluttered in that case.

For our next example, let's look at the `ln_wage` variable in `nlswork.dta`. As its name implies, this is already on a natural logarithm scale. A `spikeplot` can be interesting as a way of checking on the fine structure of a distribution, but in turn it can benefit from a root scale for frequencies to pull in long spikes and pull out short spikes. See Cox (2012, 2021b) and references given therein if you want more discussion of root scales. Wilke (2019) is unusual among graphics books in giving root scales any particular attention.

So we want to use `mylabels` in two ways—to show the raw scale for a variable already logged and to show the original scale for a variable we are going to show as square roots.

```
. webuse nlswork, clear
(National Longitudinal Survey of Young Women, 14-24 years old in 1968)
. spikeplot ln_wage, root
. mylabels 1 3 10 30 100 300, myscale(ln(@)) local(myxla)
0 "1" 1.09861228866811 "3" 2.302585092994046 "10" 3.401197381662155 "30"
4.605170185988092 "100" 5.703782474656201 "300"
```

```
. mylabels 0 25 100 225, myscale(sqrt(@)) local(myyla)
0 "0" 5 "25" 10 "100" 15 "225"

. spikeplot ln_wage, root xla(`myxla') yla(`myyla')
> ytitle(frequency (root scale))
> xtitle(wage/GNP deflator (log scale)) xsc(titlegap(*5))
```

The result is shown in figure 7.

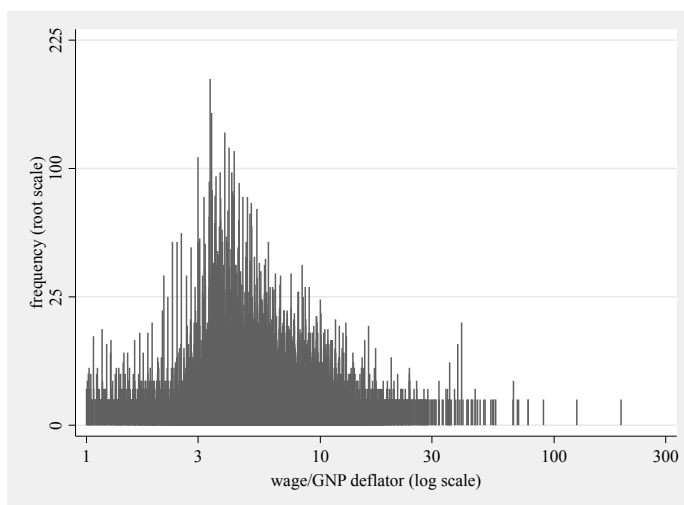


Figure 7. Spikeplot of wage on logarithmic scale with frequencies on a root scale. See text for how **mylabels** was used to get the axis labels.

**mylabels** can be used to add text systematically to axis labels, whether as a prefix or a suffix. Common applications include percent signs, currency symbols, and units of measurement. Let's look at percent of older people (meaning aged 65 or older) in the U.S. census data for 1980.

Suppose first you want to show “%” explicitly by each label. Precedents include Robbins (2013, 188, 250, 278, 318), Knaflitz (2015, 1, 48f, 51, 59, 81f, 156, 209f, 228ff, 238f), Koponen and Hildén (2019, 22, 64, 77, 84, 94, 101, 107, 185, 190, 193ff, 206, 209, 214ff), Wilke (2019, 29, 34, 69, 101f, 104, 107, 139ff, 179, 184f, 234ff, 258f, 261), Tufte (2020, 107), and Gelman, Hill, and Vehtari (2021, 4, 29f, 94, 96, 114, 126f, 166, 292, 467f). The excess of zeal in citation will be explained shortly.

**mylabels** will let you do this. Here, as an easy twist, we first use **nicelabels** to get the labels and then **mylabels** to add a “%” suffix.



Note the way the local macro is just passed from command to command, being modified or just used according to need.

```
. sysuse census, clear
(1980 Census data by state)

. generate pc_older = 100 * pop65p / pop

. nicelabels pc_older, local(yla)
step:      5
labels:    0 5 10 15 20

. mylabels `yla', suffix(%) local(yla)
0 "0%" 5 "5%" 10 "10%" 15 "15%" 20 "20%"

. scatter pc_older medage, ylabel(`yla', ang(h)) xlabel(, format(%2.0f))
> ytitle(% 65 and older) ms(none) mlabel(state2) mlabpos(0)
```

Figure 8 is the result.

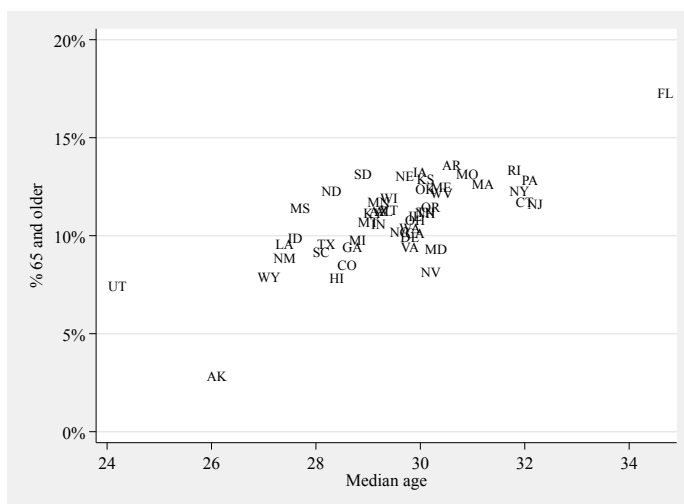


Figure 8. Percent 65 and older and median age for 50 U.S. states in 1980. **mylabels** was used to add the percent sign to each *y*-axis label.

There is a question of taste here. Simply, if you think that adding a percent sign to every label is repetitive clutter, then do not do that. The point of multiplying citations just now was to underline that some excellent authors often do this. Do make your own choice. You might prefer to explain measurement units in an axis title or text caption, if they need explanation at all.

There is a compromise, amply exemplified by Wong (2010, 24f, 50f, 58ff, 64f, 69, 100, 102, 107, 111, 115, 117, 119, 126, 128ff, 141) and Koponen and Hildén (2019, 98f, 181f), which is to add a prefix or a suffix to one label only, say, the first or last on an axis. **mylabels** has options **firstonly** and **lastonly**, added in this 2022 version of the command.

## 5 mylabels in principle

### 5.1 Syntax

```
mylabels lblist, local(macname) [myscale(transformation_syntax)
    format(format) clean prefix(text) suffix(text) firstonly lastonly]
```

```
myticks ticklist, local(macname) [myscale(transformation_syntax)]
```

### 5.2 Description

`mylabels` and `myticks` generate axis labels and ticks on specified scales. Their uses are best explained by examples, as below. Note that the label list, *lblist*, and tick list, *ticklist*, will be expanded if presented as legal *numlists* and left as specified otherwise.

### 5.3 Remarks

You draw a graph, and one axis is on a transformed square-root scale. You wish the axis labels to show untransformed values. For some values, this is easy; for example, `ylabel(0 1 2 "4" 3 "9" 4 "16" 5 "25")` says to use "4" as a label for 2, and so forth. For other values or other transformations, the conversion may be more difficult to do in your head, so a dedicated utility is helpful.

The idea behind `mylabels` is that you feed it the labels (usually, but not necessarily, numeric labels) that you want shown and the transformation being used. It will then place the appropriate specification in a local macro that you name. You may then use that local macro as part of a later `graph` command.

Suppose you want labels 0 1 4 9 16 25 36 49, and your data are square roots of these. Your call is `mylabels 0 1 4 9 16 25 36 49, myscale(sqrt(@)) local(labels)`. Or suppose you want percents shown, and your data are proportions 0–1. Your call is `mylabels 0(25)100, myscale(@/100) local(labels)`. Think of this as follows: My graph labels that I want are 0(25)100, but my data scale is that of the labels divided by 100.

A similar idea may be used for axis ticks.

Further examples are given in the help file.

### 5.4 Options

`local(macname)` inserts the option specification in local macro *macname* within the calling program's space. Hence, that macro will be accessible after `mylabels` or `myticks` has finished. This is helpful for later use with `graph` or other graphics commands. `local()` is required.

`myscale(transformation_syntax)` specifies the transformation used to produce the data you have. Stata syntax should be used with `@` as a placeholder for the original value. To show proportions as percents, specify `myscale(@/100)`. If no transformation is being used, you may specify `myscale(@)`, but that is also the default.

`format(format)` specifies a format controlling the labels. This is an occasionally specified option. Its main use is to enforce the presentation of leading zeros.

`clean` specifies a minimal format eliding trailing zeros and decimal points (whether periods [stops] or commas). This option is most often used together with `format()`. Thus, by itself `format(%03.2f)` would render 0(0.25)1 as 0.00 0.25 0.50 0.75 1.00, but `clean` reduces it to 0 0.25 0.5 0.75 1. Do not use this option with integer labels such as 10(10)40.

`prefix(text)` specifies text to be prepended to all axis labels. Specify any blank spaces within " ".

`suffix(text)` specifies text to be appended to all axis labels. Specify any blank spaces within " ".

`firstonly` and `lastonly` specify that any prefix or suffix should be attached to the first or last label only.

## 6 Conclusion

In graphics, like much else, the devil can lie in the details. Many graphs are for one purpose and can be optimized on the fly to match your personal taste or some compelling prescription. Other graphs may need to follow some overarching choices, worked out systematically. `nicelabels`, `mylabels`, and `myticks` can help either way, but particularly if you are writing scripts to automate graph choices across a range of broadly similar graphs. These commands depend on the convenience of local macros to hold the information produced.

Some other tips and tricks in similar territory can be found in Cox (2005b, 2007b, 2008a) and Cox and Wiggins (2019).

## 7 Acknowledgments

Many questions on Statalist and elsewhere have shown what kinds of needs arise across a wide range of Stata graphics practice. Richard Campbell, Richard Goldstein, John Kim, Clive Nicholas, Gabriel Rossman, and Philippe van Kerm are particularly recalled for challenges, corrections, and suggestions.

## 8 Programs and supplemental materials

To install a snapshot of the corresponding software files as they existed at the time of publication of this article, type

```
. net sj 22-4
. net install gr0092      (to install program files, if available)
. net get gr0092          (to install ancillary files, if available)
```

## 9 References

- Cleveland, W. S. 1985. *The Elements of Graphing Data*. Monterey, CA: Wadsworth.
- . 1994. *The Elements of Graphing Data*. Rev. ed. Summit, NJ: Hobart.
- Cox, N. J. 2004. Speaking Stata: Graphing categorical and compositional data. *Stata Journal* 4: 190–215. <https://doi.org/10.1177/1536867X0400400209>.
- . 2005a. Speaking Stata: Smoothing in various directions. *Stata Journal* 5: 574–593. <https://doi.org/10.1177/1536867X0500500408>.
- . 2005b. Stata tip 24: Axis labels on two or more levels. *Stata Journal* 5: 469. <https://doi.org/10.1177/1536867X0500500316>.
- . 2007a. Stata tip 43: Remainders, selections, sequences, extractions: Uses of the modulus. *Stata Journal* 7: 143–145. <https://doi.org/10.1177/1536867X0700700113>.
- . 2007b. Stata tip 55: Better axis labeling for time points and time intervals. *Stata Journal* 7: 590–592. <https://doi.org/10.1177/1536867X0800700410>.
- . 2008a. Speaking Stata: Between tables and graphs. *Stata Journal* 8: 269–289. <https://doi.org/10.1177/1536867X0800800208>.
- . 2008b. Stata tip 59: Plotting on any transformed scale. *Stata Journal* 8: 142–145. <https://doi.org/10.1177/1536867X0800800113>.
- . 2012. Speaking Stata: Transforming the time axis. *Stata Journal* 12: 332–341. <https://doi.org/10.1177/1536867X1201200210>.
- . 2018. Speaking Stata: Logarithmic binning and labeling. *Stata Journal* 18: 262–286. <https://doi.org/10.1177/1536867X1801800116>.
- . 2020a. Software Updates: gr0072\_1: Speaking Stata: Logarithmic binning and labeling. *Stata Journal* 20: 1028–1030. <https://doi.org/10.1177/1536867X20976342>.
- . 2020b. Speaking Stata: Loops, again and again. *Stata Journal* 20: 999–1015. <https://doi.org/10.1177/1536867X20976340>.
- . 2021a. Erratum: Speaking Stata: Loops, again and again. *Stata Journal* 21: 555. <https://doi.org/10.1177/1536867X211025839>.

- . 2021b. Stata tip 141: Adding marginal spike histograms to quantile and cumulative distribution plots. *Stata Journal* 21: 838–846. <https://doi.org/10.1177/1536867X211045583>.
- 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>.
- Gelman, A., J. Hill, and A. Vehtari. 2021. *Regression and Other Stories*. Cambridge University Press: Cambridge.
- Hardin, J. W. 1995. dm28: Calculate nice numbers for labeling or drawing grid lines. *Stata Technical Bulletin* 25: 2–3. Reprinted in *Stata Technical Bulletin Reprints*. Vol. 5, pp. 19–20. College Station, TX: Stata Press.
- Heckbert, P. S. 1990. Nice numbers for graph labels. In *Graphics Gems*, ed. A. S. Glassner, 61–63. Cambridge, MA: Academic Press.
- Knafllic, C. N. 2015. *Storytelling with Data: A Data Visualization Guide for Business Professionals*. Hoboken, NJ: Wiley.
- Koponen, J., and J. Hildén. 2019. *The Data Visualization Handbook*. Espoo: Aalto ARTS Books.
- Robbins, N. B. 2013. *Creating More Effective Graphs*. Wayne, NJ: Chart House.
- Royston, P. 1996. gr21: Flexible axis scaling. *Stata Technical Bulletin* 34: 9–10. Reprinted in *Stata Technical Bulletin Reprints*. Vol. 6, pp. 34–36. College Station, TX: Stata Press.
- Talbot, J., S. Lin, and P. Hanrahan. 2010. An extension of Wilkinson’s algorithm for positioning tick labels on axes. *IEEE Transactions on Visualization and Computer Graphics* 16: 1036–1043. <https://doi.org/10.1109/TVCG.2010.130>.
- Tufte, E. R. 2001. *The Visual Display of Quantitative Information*. 2nd ed. Cheshire, CT: Graphics Press.
- . 2020. *Seeing with Fresh Eyes: Meaning, Space, Data, Truth*. Cheshire, CT: Graphics Press.
- Wilke, C. O. 2019. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. Sebastopol, CA: O’Reilly.
- Wilkinson, L. 2005. *The Grammar of Graphics*. 2nd ed. New York: Springer.
- Wong, D. M. 2010. *The Wall Street Journal Guide to Information Graphics: The Dos and Don’ts of Presenting Data, Facts, and Figures*. New York: W. W. Norton.

**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 16 commands in official Stata. He was an author of several inserts in the *Stata Technical Bulletin* and is Editor-at-Large of the *Stata Journal*. His “Speaking Stata” articles on graphics from 2004 to 2013 have been collected as *Speaking Stata Graphics* (2014, College Station, TX: Stata Press).