The Stata Journal (2007)
7, Number 3, pp. 413–433

Speaking Stata: Turning over a new leaf

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

Abstract. Stem-and-leaf displays have been widely taught since John W. Tukey publicized them energetically in the 1970s. They remain useful for many distributions of small or modest size, especially for showing fine structure such as digit preference. Stata's implementation `stem` produces typed text displays and has some inevitable limitations, especially for comparison of two or more displays. One can re-create stem-and-leaf displays with a few basic Stata commands as scatterplots of stem variable versus position on line with leaves shown as marker labels. Comparison of displays then becomes easy and natural using `scatter, by()`. Back-to-back presentation of paired displays is also possible. I discuss variants on standard stem-and-leaf displays in which each distinct value is a stem, each distinct value is its own leaf, or axes are swapped. The problem shows how one can, with a few lines of Stata, often produce standard graph forms from first principles, allowing in turn new variants. I also present a new program, `stemplot`, as a convenience tool.

Keywords: gr0028, stemplot, stem-and-leaf, graphics, distributions, digit preference

1 Introduction

I was attracted to stem-and-leaf displays when I first learned about them in the mid-1970s from John Tukey's writings. They were a fixture in my introductory courses from the start of my teaching career. But over the last decade or so, I have come to use them less. As usually implemented, they are less flexible than dot or strip displays. This column is a story of how one can produce stem-and-leaf displays by using Stata's graphics, thus increasing their versatility. Within this story, other questions will emerge: How far can you get using a few basic commands? When is it better to wrap up those commands within a program?

2 Stem-and-leaf displays

Just in case you have missed out on stem-and-leaf displays, let us run through the basic idea and its implementation in Stata as `stem`.

The name *stem-and-leaf* is attributable to Tukey (1972, 1977). Their widespread use is undoubtedly the result of his energetic and enthusiastic advocacy. The basic idea appears in other places, including Dudley (1946, 22), as reproduced by Emerson and

Hoaglin (1983, 19), and Japanese railway timetables, as reproduced by Tufte (1990, 45–46). Among many textbook accounts are those of Griffiths, Stirling, and Weldon (1998); Wild and Seber (2000); and Moore and McCabe (2006).

Let us look at some examples.

```
. sysuse auto
(1978 Automobile Data)

. stem length

Stem-and-leaf plot for length (Length (in.))

  14* | 279
  15* | 45567
  16* | 133455589
  17* | 00002234457999
  18* | 02469
  19* | 23356788889
  20* | 00001113446667
  21* | 224788
  22* | 00112
  23* | 03
```

Each value (e.g., 142) is split into so-called *stem* (e.g., 14) and *leaf* (e.g., 2) parts. Here each distinct stem is used just once, but there is no rule about that. Another example shows how stems are repeated on two or more lines:

```
. stem mpg

Stem-and-leaf plot for mpg (Mileage (mpg))

  1t | 22
  1f | 44444455
  1s | 66667777
  1. | 88888888899999999
  2* | 00011111
  2t | 22222333
  2f | 444455555
  2s | 666
  2. | 8889
  3* | 001
  3t |
  3f | 455
  3s |
  3. |
  4* | 1
```

Here, at least for English speakers, accidents of language make `t`, `f`, and `s` easy labels for two or three, four or five, and six or seven, respectively. In general, a stem may be used most easily one, two, or five times with decimal representations.

2.1 A note on units

In most countries in the world, the units used in examples in this column are not standard, so I will explain. The car lengths in the previous example are measured in inches. An inch is defined to equal 25.4 mm. Twelve inches = 1 foot, 5,280 feet = 1 mile, and 1 U.S. gallon is about 3.785 liters. Hence 1 mile per gallon (mpg) is about 0.425

km per liter or 2.352 liter per km. Twenty miles per gallon, the median in the `auto` dataset, is about 8.50 km per liter. Pennycuick (1988) and Darton and Clark (1994) are two of many useful references on units of measurement.

2.2 Advantages

The display is a graphical condensation of the list of ordered values 142, 147, 149, . . . , 230, 233. Once you realize that, its key features are immediate, which is much of the attraction:

- *A low-technology method.* You can do stem-and-leaf displays by hand, using any convenient piece of paper. (Paper with large squares keeps things tidy.) Evidently, that consideration weighs less and less in an era when many readers take laptops with them wherever they go. But it still has weight, especially as a way of getting children and other learners to look at distributions in small, simple datasets. Similarly, one can easily type or print a stem-and-leaf display. That was an advantage to many in the mid-1970s, but again it counts for little 30 years later. Indeed, handling typed displays in some fixed-width font may now be less familiar and more awkward for you than handling graphic images.

- *Keeps most or all information.* Each value is split into stem and leaf parts. For `length`, there is no more detail, so the stem-and-leaf display with leaves that are single digits loses no information about magnitudes. Whenever that is not true, there are various possibilities, including rounding the data and using leaves that are two or more digits.

- *Shows distributions in histogram style, but including fine structure.* The ordering and grouping yield a histogram on its side, which is, presumably, why stem-and-leafs are usually shown with magnitude increasing down the page. A rotation of 90° counterclockwise yields a histogram. Demonstrating this effect physically by rotating an acetate transparency in the classroom is, unfortunately, now regarded as technologically incorrect. The stem-and-leaf display, however, retains more detail (as seen, often all the detail) of individual data points. That quality is often useful, especially for inspecting outliers and other extremes and any granularity in the distribution, such as may arise from digit preference (Preece 1981). A display need not appear in published reports for it to be useful in exploratory checking of your data.

- *Allows calculation of measures on the basis of ordered values.* The ordering allows fairly easy hand calculation of summary measures on the basis of order statistics (median, quartiles, and so forth). Tukey's low-technology definitions of what later were called letter values (Hoaglin 1983) involved nothing more complicated than splitting the difference between adjacent order statistics. (Doing so makes the definitions trickier to extend to variables associated with weights.) This detail still deserves brief emphasis in introductory texts and courses but is of little importance for researchers able to calculate such measures directly with software such as Stata.

2.3 Limitations

The `stem` ([R] **stem**) command was first introduced in Stata 3.0 in 1992. It does what it does well and has smart defaults based on the distribution of each variable. Its main features include handles for tuning the display and the possibility of repeating displays for distinct groups with a `by()` option.

Given these advantages, why are stem-and-leaf displays not used more? The limitations are just about as obvious:

- *Problems with large datasets.* Stem-and-leaf displays work well for datasets of small or modest size but otherwise may easily be awkward, if not impracticable. The length of the longest line, the number of lines, and the legibility of leaves can quickly become problematic, and there are few easy trade-offs. Using some kind of dot plot (in the sense of [R] **dotplot**, not [G] **graph dot**) is then more practical: we must give up on the idea of a leaf and use some small point symbol instead. Alternatively, many people would turn to histograms, kernel density estimates, quantile plots, or other representations.
- *Are final digits interesting?* The last digit(s) may often not be at all interesting or informative, especially when data were produced by some automated measurement procedure. (Whenever people produce numbers, there is a chance of fine structure, perhaps trivial, as a result of digit preferences or other quirks.) If the detail of leaves is noise to the user, then the simplicity of dot plots is again preferable.
- *Comparison can be awkward.* Comparison of two or more stem-and-leaf displays can be difficult or inefficient unless they are displayed side by side with a common magnitude scale. The action of `stem`, almost inevitably, is to emit displays in sequence, so only reediting in text or word processing software could achieve this. Aligning displays, line length, and legibility of leaves could again prove problematic.

3 Stem-and-leaf displays as scatterplots

These limitations are serious, but they do not rule out the usefulness of stem-and-leaf displays in all circumstances. Moreover, some are largely consequences of using printed text to show stem-and-leaf displays as, in essence, tables rather than graphs. For example, keeping track of various small tables as bundles of characters and trying to emit them properly aligned would be a small nightmare, but getting small graphs aligned as you wish is easy, because you just tell Stata to do it.

A fresh look at the problem grows from the realization that stem-and-leaf displays are essentially a kind of scatterplot. The ingredients include the following:

1. The variable in question, or a rounded variant of it, gives the y coordinate. For that, we may need to learn a little about how to round.

2. The position of each leaf on each line gives the x coordinate. We will have to construct this. A natural start is just to ensure that the first, second, and later leaves on each line are plotted against 1, 2, and so forth. (Only for one purpose will we need to do anything more complicated.)

3. A leaf must be shown at each point (x, y), rather than a point symbol. For that, we may need to learn a little about how to extract the needed digits. Leaves will be shown as marker labels, and marker symbols will be suppressed. If you are not familiar with marker labels, check out [G] ***marker_label_options*** or the equivalent online help.

In following this basic idea, we need not commit ourselves to producing exact replicas of typed or printed displays. Using graphics rather than text may well mean that different design choices are easier, more attractive, or more informative.

3.1 From first principles

The official `dotplot` ([R] **dotplot**) command yields displays similar to stem-and-leaf displays, but showing point symbols rather than leaves. `stripplot` on SSC is a loosely similar user-written command. So, we could try tricking either into showing leaves. A more direct approach is to try building up a command for ourselves from first principles. This strategy strengthens understanding and often shows that a few simple commands will give us the result we desire, thus avoiding the writing of a program.

My habit is to start with small toy examples and get something simple working quickly, not to play with the dataset I really want to work with (which is likely to be much larger), or to sketch an elaborate design in advance. (This is not necessarily good advice for large, complicated programming projects.) In that spirit, consider `mpg` in the `auto` dataset, the second stem-and-leaf example above. The values are all integers, ranging from 12 to 41. The simplest choice, at first sight, is to use the second digits as leaves. One way to do that is to extract the second digit as the remainder on division by 10:

```
. gen leaf = mod(mpg, 10)
```

Another way is to extract it as the last character of the string equivalent of `mpg`:

```
. gen leaf = substr(string(mpg), -1, 1)
```

Here `-1` as the second argument indicates the last character in a string. Naturally,

```
. gen leaf = substr(string(mpg), 2, 1)
```

would produce the same result with `mpg`, but one needs only a little foresight to see that this is less general, applying only to two-digit integers. `scatter` will happily use either numeric or string variables as marker labels, so the choice is for us to make on other grounds.

We need x and y coordinates. We need an x coordinate more, so let's get that first, and just use each distinct value as its own stem:

```
. by mpg, sort: gen x = _n
```

Under `by:`, the observation number `_n` is evaluated separately within distinct groups defined by its argument. Within groups of distinct values of `mpg`, it thus takes on the values 1, 2, and so forth, exactly as desired. If you want a more detailed tutorial on `by:`, see Cox (2002).

Figure 1 shows our initial stem-and-leaf plot:

```
. scatter mpg x, mla(leaf) mlabpos(0) ms(none)
```

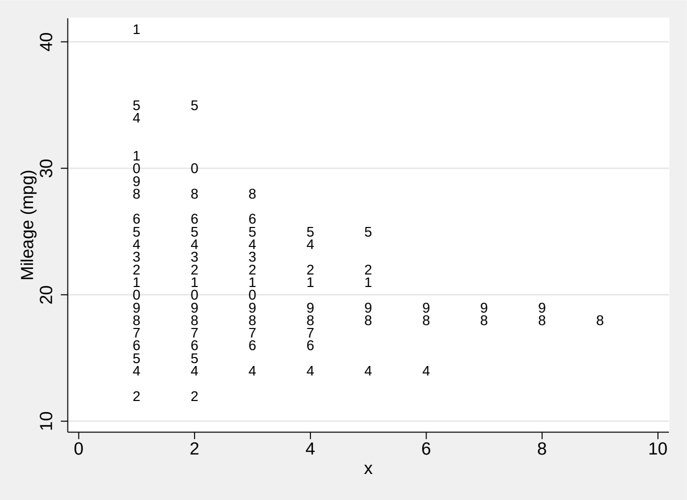

Figure 1: Stem-and-leaf plot for miles per gallon with `scatter`. Each distinct value serves as a stem. Final digits serve as leaves.

When we suppress marker symbols and show marker labels instead, using `mlabpos(0)` to center labels on where symbols would have been is often best.

The result is not especially pretty. More importantly, we are most of the way to where we want to be after just three lines. My first thoughts on this graph are in terms of simple cosmetic improvements. I do not want the x-axis title. I do not want the x-axis labels (although others might want a frequency scale on that axis). I do want horizontal y-axis labels. To match stem-and-leaf conventions, I do not want them to be ticked. I do not want the associated grid lines (although this preference also is a matter of taste). With a few minor changes, we produce figure 2.

```
. scatter mpg x, mla(leaf) mlabpos(0) ms(none) yla(, ang(h) noticks nogrid)
> xla(none) xti("")
```

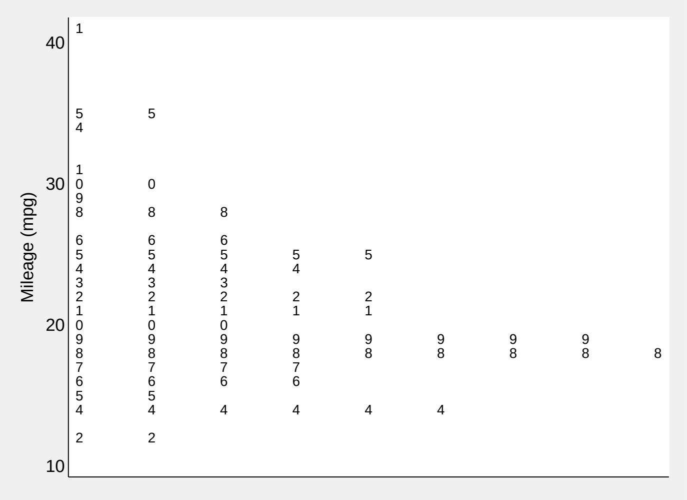

Figure 2: Same as figure 1 but with cosmetic improvements to axis labels and titles

If you are working through this sequence with your copy of Stata, you might be using a graph scheme that sets up a default of `yla(, ang(h))`, but making that choice explicit will do no harm.

My next thoughts are that I have more space to play with than I expected. Figure 3 suggests a variant in which each value is its own leaf:

```
. scatter mpg x, mla(mpg) mlabpos(0) ms(none) yla(, ang(h) noticks nogrid)
xla(none) xti("")
```

(*Continued on next page*)

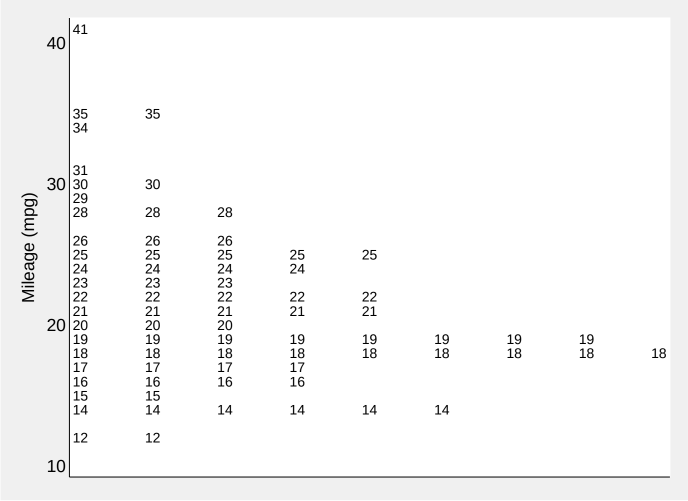

Figure 3: Same as figure 2, but now each value serves as its own leaf.

Now the y-axis labels are redundant, and we can take them off. If we do that, we will probably want to keep a gap in the same space, which is shown in figure 4.

```
. scatter mpg x, mla(mpg) mlabpos(0) ms(none) xla(none) xti("") yla(none)
> ysc(titlegap(4))
```

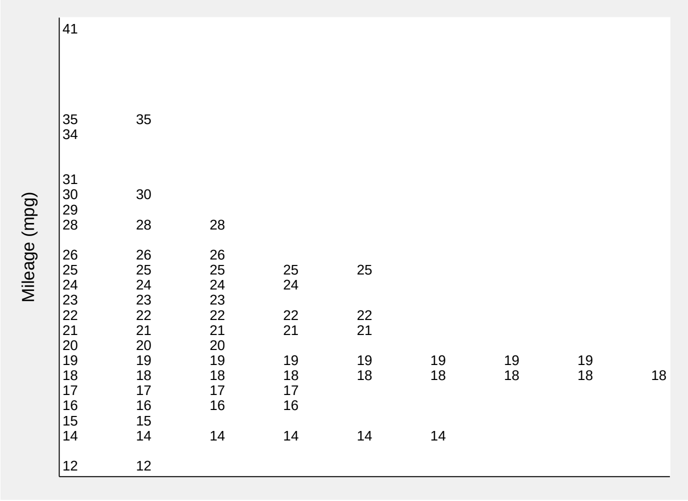

Figure 4: Same as figure 3, but now y-axis labels have been removed.

Our result is no longer a conventional stem-and-leaf display, but no matter: it is showing the same information in readable form.

Using distinct values as stems will work well only if there are enough ties, that is, not too many distinct values. (Some people say *unique values*, but for others that term means those values that occur just once.) You can see the number of distinct values by following a `tabulate` with `display r(r)`, because the number of rows in the table is saved immediately after a `tabulate`. `mpg` has 21 distinct values.

If you try this calculation with the first example above, `length`, the resulting graph (not shown here) is too messy. However, we can reuse most of the code. The main change is the need to recalculate the x coordinate.

```
. by length, sort: replace x = _n
. scatter length x, mla(length) mlabpos(0) ms(none) xla(none) xti("")
> yla(none) ysc(titlegap(4))
```

There are 47 distinct values of `length`. In general, performing `tabulate` beforehand on a potential stem variable will give a foretaste of the number of stems needed and their lengths. My own rule is that more than about 30 stems is usually too messy, unless you are happy to produce a tall graph and can convince those in power over you (advisors, supervisors, reviewers, editors) that it is a good idea.

3.2 Rounded stems

We now need to focus on how to produce rounded stems. `stem`'s choices for `length`, which varies between 142 and 233 inches, are 14(0), . . . , 23(0), which are simple and appear good. Your first thought may be to extract the stem as the first two characters of the string equivalent of `length`, as, say, `substr(string(length), 1, 2)`, but this choice would pose two problems.

First, we need a numeric variable for graphing. That problem is easily soluble: convert the string to numeric by wrapping the expression with `real()` to give `real(substr(string(length), 1, 2))`.

Second, a deeper problem is that extracting the first few digits will give some wrong answers for any variable with different numbers of digits. (Consider the results if values varied between 142 and 2,330 instead.) Approaching the problem directly as a numeric calculation is better in general. Thus the stems for `length` are obtainable by dividing by 10 and rounding down:

```
. gen stem = int(length/10)
```

The `floor()` function would do as well as the `int()` function so long as values are not negative. However, if values are ever negative, `floor()`, which always rounds down, gives the wrong answer. `int()`, which always rounds towards zero, is preferable. The stem for −142 should be −14, not −15.

This rule would work well enough to re-create `stem`'s rendering of `length`, but we need a more general solution for whenever the width (in `stem`'s terminology) is not 10, or even any other exact power of 10. Suppose that we wanted twice as many stems than is given by 10. A rule that was

```
. gen stem = int(length/5)
```

would yield a stem of 28 for values 140–144, 29 for values 145–149, and so forth. But we would not want those stems to be shown as axis labels. A better command for graph purposes would be

```
. gen stem = width * int(varname / width)
```

So, we should start with

```
. gen stem = 10 * int(length/10)
```

However, there is a better approach that will save us work in the long run. Using `clonevar` ([D] **clonevar**) first,

```
. clonevar stem = length
. replace stem = 10 * int(length/10)
```

has a pleasant result. The effect of `clonevar` is that `stem` is born an exact copy of `length`, notably with the same variable label. That is a useful detail for graphing. Naturally, following `clonevar` with `replace` does not affect the variable label.

Let us put together in one place the code for a better stem-and-leaf plot for `length`, starting from nothing. We will need to calculate the x coordinate by using the variable `stem`—ensuring that the leaves are in the right order—and the leaf by using `length` itself. The code will produce figure 5.

```
. sysuse auto, clear
. clonevar stem = length
. replace stem = 10 * int(length/10)
. sort stem length
. by stem: gen x = _n
. gen leaf = substr(string(length), -1, 1)
. scatter stem x, mla(leaf) mlabpos(0) ms(none) yla(, ang(h) noticks nogrid)
> xla(none) xti("")
```

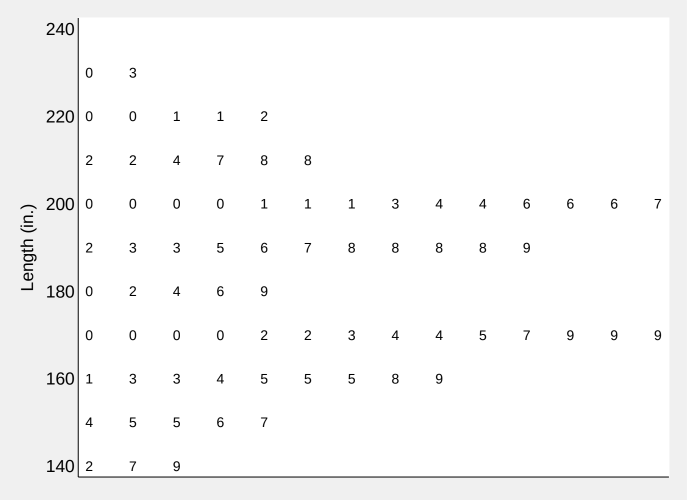

Figure 5: Stem-and-leaf plot for car lengths. Stems are based on intervals of width 10 in. Leaves are final digits.

From here, there are various ways to proceed. One is to make explicit all stems (and follow the convention of showing them truncated) by adding

```
yla(140 "14" 150 "15" 160 "16" 170 "17" 180 "18" 190 "19" 200 "20" 210
"21" 220 "22" 230 "23")
```

Another is to note that there is plenty of white space to spare. So we could use the values as their own leaves and suppress the *y*-axis labels:

```
. scatter stem x, mla(length) mlabpos(0) ms(none) yla(none) xla(none) xti("")
> ysc(titlegap(4))
```

3.3 Comparisons

One of the limitations of the `stem` command is that with a `by()` option, stem-and-leaf displays for different groups can be produced only in vertical sequence. Being able to compare displays that are aligned horizontally would be better, suggesting the use of `scatter` with a `by()` option. The only questions are what difference this might make to preprocessing and what extra details we might need to think about graphically.

Let us return to `mpg` and examine its variation with groups of `foreign` or `rep78`. For simplicity, values will be their own leaves. For clarity, we will start again from the top to show the full sequence of commands, which will produce figures 6 and 7.

```
. sysuse auto, clear
. by foreign mpg, sort: gen x = _n
. scatter mpg x, by(foreign, compact) xla(none) xti("") ms(none) mla(mpg)
> mlabpos(0) yla(none)
```

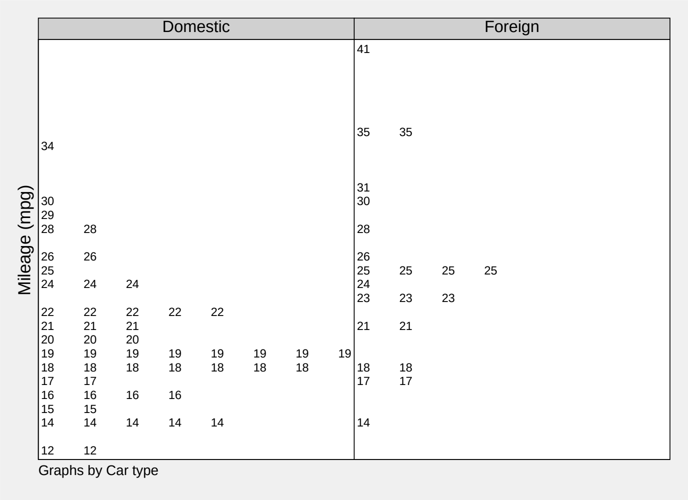

Figure 6: Stem-and-leaf plots of miles per gallon for domestic and foreign cars. Each value serves as its own leaf.

```
. by rep78 mpg, sort: gen X = _n
. scatter mpg X, by(rep78, compact row(1)) xla(none) xti("") ms(none)
> mla(mpg) mlabpos(0) yla(none)
```

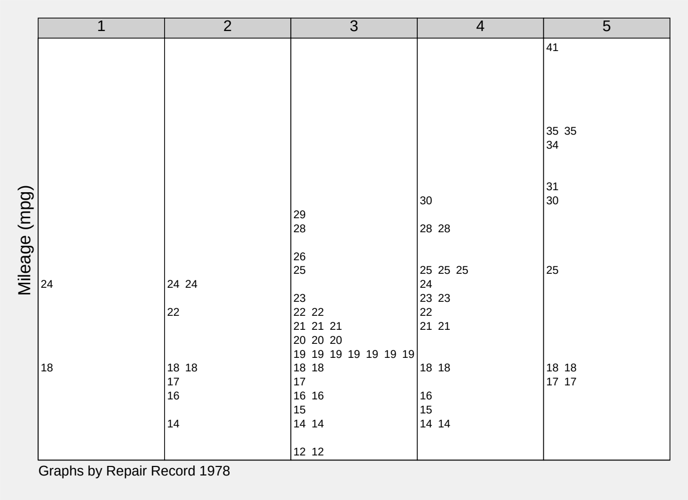

Figure 7: Stem-and-leaf plots of miles per gallon by repair record. Each value serves as its own leaf.

The calculation of the x coordinate needs to be done separately by the grouping variable. Suboptions of `by()` may come into play. `compact` and `row(1)` are useful. However, beware that `total` does not do what you might hope, because it necessarily superimposes different groups.

3.4 Back to back

When two stem-and-leaf plots are compared, some people like to plot them back to back. The result loosely resembles the pyramidal displays often used to compare age structure of populations for males and females. We could discuss whether the result is more attractive or more informative, but let us focus on how it would be done. Look again at figure 6. On the right-hand side the plot for foreign cars is as we would wish for a back-to-back plot. What we need to change is the display on the left-hand side for domestic cars. Each line should be reversed and placed against the right-hand side. `foreign` is 0 for domestic cars (and 1 for foreign cars). Then the command

```
. by foreign mpg, sort: gen bbx = cond(foreign == 0, -_n, _n)
```

assigns negative x coordinates, -1 down, on the left-hand side and positive coordinates, 1 and up, on the right-hand side. (If you want a tutorial on `cond()`, see Kantor and Cox [2005].) We need to ensure that the largest value of the x coordinate appears in the left-hand panel and the smallest value in the right-hand panel. That way, the data points in the left-hand display nearly touch the right-hand edge of their display and vice versa. Some analysis shows that this goal is achieved by a translation (shunt, in plainer English) of the left-hand panel coordinates to the right:

```
. summarize bbx, meanonly
. replace bbx = bbx + max(-r(min), r(max)) + 1 if foreign == 0
```

Let us look at that again. By construction, the x coordinates are negative on the left-hand side and positive on the right-hand side. Thus `r(min)`, which holds the minimum value of the x coordinates `bbx` after a `summarize`, is always negative. Its negation `-r(min)` is the length of the longest stem in the left-hand panel. `r(max)` is the length of the longest stem in the right-hand panel. Our translation is of whichever is larger, plus 1, to the right.

If you care about the detailed logic of that translation, focus on some examples. (Otherwise, skip to the next paragraph.) Suppose that the longest stems have length 4 on the left-hand side and length 3 on the right-hand side. These will have x coordinates -4, -3, -2, -1 and 1, 2, 3, respectively. For the displays to touch as desired, -4 must be translated to a number at least 1 and -1 must be translated to a number at least 3. The gentlest solution is the mapping from -4, -3, -2, -1 to 1, 2, 3, 4, namely, adding 5, the length of the longer stem plus 1. If the stems were reversed, namely, with x coordinates -3, -2, -1 and 1, 2, 3, 4, the same solution would apply. If we had two stems of equal length, say, -3, -2, -1 and 1, 2, 3, the same rule would apply: the longer stem (either one) has length 3, and adding 4 maps -3, -2, -1 to 1, 2, 3. We do not see any of these values, because the x-axis labels are all suppressed, but we do need to do this for the graph to appear as we would wish.

Once the translation is done, we have all we want, as shown in figure 8:

```
. scatter mpg bbx, by(foreign, compact) xla(none) xti("") ms(none) mla(mpg)
> mlabpos(0) yla(none)
```

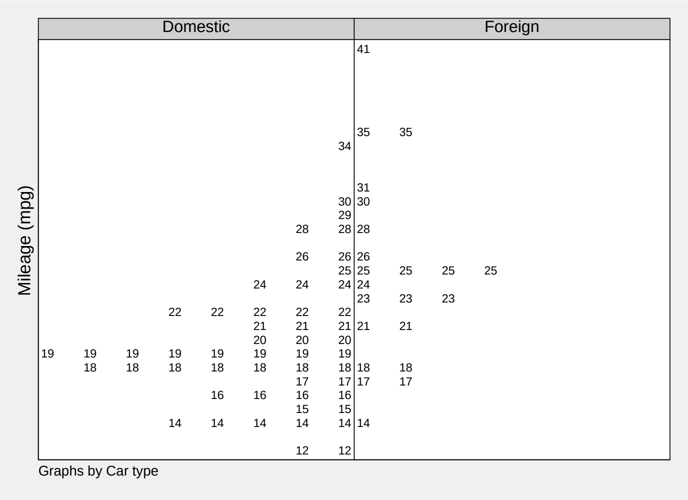

Figure 8: Back-to-back stem-and-leaf plots of miles per gallon for domestic and foreign cars

In typed back-to-back stem-and-leaf displays, stems are shown on a spine between the two parts. Having the stems as axis labels, whether shown or suppressed, is more straightforward when we do it graphically. A second set of labels could be shown on the right-hand axis if desired.

3.5 Other possibilities

By no means have we exhausted the possibilities for simple variations on the basic theme. Here are some more.

- *Reversing y-axis scale.* So far, as you will have noticed, we have followed the usual graph convention in which response magnitude, shown on the y axis, increases upward. The convention with text stem-and-leaf displays is the opposite. The graph option `ysc(reverse)` will produce that.

- *Different leaf choices.* Although we have considered in detail how to produce leaves from some or all of the digits of the variable whose distribution is being shown, doing so is not compulsory. Something different could be used for the leaf, including values coding for some categorical variable. The leaf is whatever is specified as marker label with the `mlabel()` option.

- *Swapping axes.* We can easily swap the y and x axes to produce a display in which magnitude is horizontal and leaves are stacked vertically.

- *Changing leaf size.* Being able to tweak the marker label size with the `mlabsize()` option is useful.

- *Aspect ratio.* We could also change the aspect ratio to use more or less horizontal space, often simply by tuning `xsize()`.

4 stemplot

One of my main aims in these columns is to emphasize how much you can do with a few basic commands to get where you want to be. Here being able to create or re-create basic graphs and to devise your own variations on them gives flexibility born from freedom. You do not depend on whether someone has written a program to do what you want to do or on whether you can find such a program if it exists. Nevertheless, producing even simple results can require mastery of fiddly little details: the manipulation needed to produce back-to-back displays is one example. Many users may not relish the minor acrobatics that may be needed.

Therefore, as an alternative, I publish a `stemplot` command with this column, encapsulating the main ideas so far, and then some more.

4.1 Syntax

`stemplot` *varname* [*if*] [*in*] [`, by(`*byvar*[`,` *byopts*]`)` `back` `digits(`#`)`
`format(`*format*`)` `width(`#`)` `horizontal` *graph_options*]

`stemplot` produces stem-and-leaf plots and similar plots for numeric variable *varname* by using Stata's graphics. For stem-and-leaf plots as typed text displays, see [R] **stem** or `help stem`.

4.2 Options

`by(`*byvar*[`,` *byopts*]`)` specifies that displays are to be shown separately by groups defined by *byvar*. See [G] ***by_option*** or `help by_option`. The suboption `total` is legal but will not work as may be desired, because it superimposes, not juxtaposes, individual displays.

`back` specifies that a two-group display specified using `by()` be shown back to back.

`digits(`#`)` specifies the number of digits to be shown as leaves. The default is `digits(1)`. To be more precise, the rule for leaves is to show

`substr(string(`*varname*`, "`*format*`"), -`*digits*`,` *digits*`)` or

`string(`*varname*`, "`*format*`")`

whenever that is empty. Here *format* is by default the display format of *varname*, as shown by `describe` and set by `format`. See also the `format()` option described below.

`format(`*format*`)` specifies a format other than the display format of *varname* to be shown in determining leaves. See also the `digits()` option described above. This option is seldom specified.

`width(`*#*`)` specifies the width of intervals defining distinct lines.

`horizontal` specifies a horizontal display in which the magnitude axis is horizontal and leaves are stacked vertically. `back` is not allowed with `horizontal`.

graph_options are any of the options documented in [G] **graph twoway scatter**. For example, `mlabel()` overrides the program's choice of leaf, and `mlabsize()` controls the size of leaves.

4.3 Comments

How does the scope of `stemplot` differ from that of combinations of individual commands, as discussed previously? Here we note the most important features.

1. `stemplot` preserves `sort` order, as is standard programming practice for any program not intended to change that order.
2. There are more checks on user input.
3. `stemplot` allows specifying `if` and `in`.
4. Various options allow tuning of the display.

4.4 A last look at those cars

Suppose that we wanted to show some information about the make of cars. The full name within the variable `make` is too long to fit comfortably on a stem-and-leaf display, but we can compromise on the first word:

```
. gen M = word(make, 1)
```

Some experimentation shows that we should `sort` within stems, use a different choice of `mlabpos()`, and stretch the x-axis scale so that the rightmost marker label fits:

```
. sort mpg M
. stemplot mpg, mla(M) mlabpos(3) xsc(r(. 10))
```

To save space, I do not show the result here, but you can experiment for yourself.

5 Choral finale

For a final example, we leave the world of automobiles and turn to choral music. Chambers, Cleveland, Kleiner, and Tukey (1983, 350) listed the heights of singers in the New York Choral Society in 1979. Cleveland (1993) plotted the dataset in various ways; it is downloadable from http://www.stat.purdue.edu/~wsc/visualizing.tables.txt.

Here we focus first on the broad subdivision between soprano, alto, tenor, and bass parts, ignoring the distinction between first (higher pitch) and second (lower pitch) groups for each part.

A general relationship between singer part and height is no surprise. What a stem-and-leaf plot can show is the fine structure of the distributions, based on the singers' own statements. Here the choice of units is important, because of their psychological overtones. Although the data are reported in inches, people in countries like the United States commonly think of their heights in feet and inches, not just in inches. An adult height of 6 feet (72 inches) is moderately tall by most standards (although clearly not, for example, that of basketball players). Similarly, an adult height of 5 feet (60 inches) is rather short by most standards, or petite if you prefer. Fully metric readers might note that 6 feet is about 1.83 meters and 5 feet about 1.52 meters.

Ordinary experience shows that although many adults are happy with their heights, a few would prefer to be taller and a few, shorter. Also, some people may be self-conscious about being distinct from whatever they regard as a norm. Let us bear all this in mind while examining these data.

Using `stemplot` is convenient here to produce figure 9.

```
. stemplot height, by(spart, row(1) compact noiytic
> note(units are inches, place(e))) yla(60 "5'" 66 "5' 6''" 72 "6'")
> yaxis(1 2) yla(60(6)72, axis(2) ang(h))
```

(*Continued on next page*)

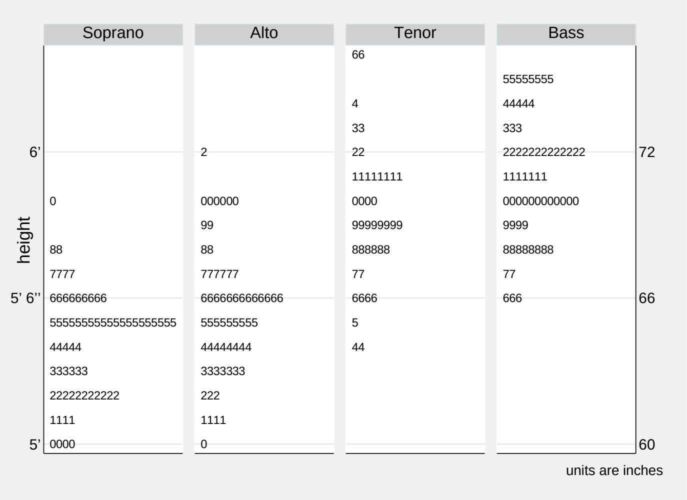

Figure 9: New York Choral Society singers' heights, 1979. Note not only the general relationship between singer part and height but also intriguing hints of fine structure.

`spart` is a numeric variable with value labels, which we need here to show soprano, alto, tenor, and bass in their natural order. A string variable would sort alphabetically from alto to tenor, which would not be a good choice. Two y axes permit two sets of labels, one in feet and inches and one in inches, making the comparison easier, even though all readers can presumably divide small numbers by 12.

The plot does indeed show intriguing, although contradictory, fine structure. No singers admit to being less than 60 inches (5 feet) tall. The modes for sopranos at 55 inches and for altos at 56 inches may be genuinely different, or they may reflect some personal preferences. Why do no altos report themselves as 71 inches? Is there some wishful thinking among some of the basses who claim 72 inches (6 feet)? There are hints of other features that may owe more to psychology than to anatomy.

We can also bring the split into higher and lower groups for each part (first and second sopranos, and so forth). One way to explore that is to use that variable as a new leaf (figure 10).

```
. stemplot height, by(spart, row(1) noiytic
> note(1 = high; 2 = low, place(e))) mla(group)
> yla(60 "5’" 66 "5’ 6’’" 72 "6’") yaxis(1 2) yla(60(6)72, axis(2) ang(h))
```

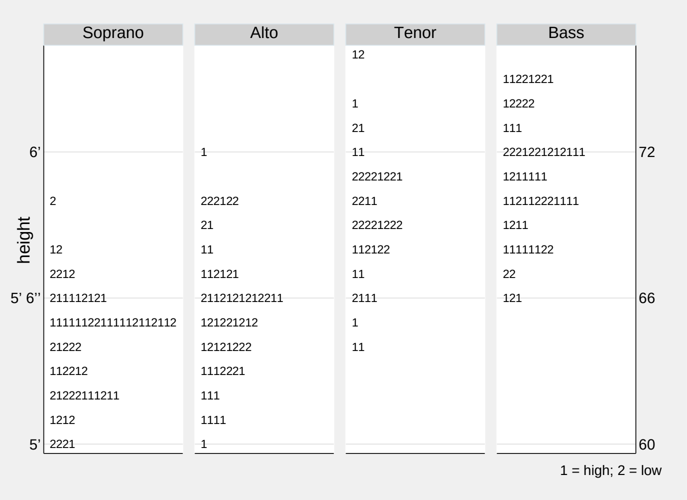

Figure 10: New York Choral Society singers' heights, 1979, by singer part and groups with relatively high or low pitch

Naturally, any fine structure identified in this way should be tested by independent evidence. We need to be clear that we are not seizing on quirks produced by sampling variation. As Harrell (2001, ix) aptly remarks, "Using the data to guide the data analysis is almost as dangerous as not doing so." The opportunity for measuring these singers again is long past, but there may be similar (preferably larger) datasets that could provide some checks. Statistical and scientific caution aside, the stem-and-leaf plot is one of the graphical tools to try when fine structure within distributions is of interest.

6 Conclusion

Stem-and-leaf displays are one of the staples of exploratory statistical graphics. However, producing them as text displays, as with Stata's `stem` command, has inevitable limitations, especially for comparison of distributions across groups. Revisiting stem-and-leaf as full-blown graphs requires no more than the idea of scatterplots with marker labels. A few steps lead not only to acceptable stem-and-leaf displays but also to variants on them. Comparison of groups is then easy by using the standard `by()` option of `twoway scatter`.

This column also illustrates a useful Stata strategy of using basic commands and toy examples and moving step by step toward what you want. Technique is thus strengthened. Serendipitous discoveries of unforeseen possibilities are likely. Although the project followed through to produce a new `stemplot` command, the greater lesson is the flexibility and power of the language available to you.

7 References

Chambers, J. M., W. S. Cleveland, B. Kleiner, and P. A. Tukey. 1983. *Graphical Methods for Data Analysis*. Belmont, CA: Wadsworth.

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

Cox, N. J. 2002. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86–102.

Darton, M., and J. Clark. 1994. *The Dent Dictionary of Measurement*. London: Dent.

Dudley, J. W. 1946. *Examination of Industrial Measurements*. New York: McGraw–Hill.

Emerson, J. D., and D. C. Hoaglin. 1983. Stem-and-leaf displays. In *Understanding Robust and Exploratory Data Analysis*, ed. D. C. Hoaglin, F. Mosteller, and J. W. Tukey, 7–32. New York: Wiley.

Griffiths, D., W. D. Stirling, and K. L. Weldon. 1998. *Understanding Data: Principles and Practice of Statistics*. Brisbane: Wiley.

Harrell, F. E., Jr. 2001. *Regression Modeling Strategies, With Applications to Linear Models, Logistic Regression, and Survival Analysis*. New York: Springer.

Hoaglin, D. C. 1983. Letter values: A set of selected order statistics. In *Understanding Robust and Exploratory Data Analysis*, ed. D. C. Hoaglin, F. Mosteller, and J. W. Tukey, 33–57. New York: Wiley.

Kantor, D., and N. J. Cox. 2005. Depending on conditions: A tutorial on the `cond()` function. *Stata Journal* 5: 413–420.

Moore, D. S., and G. P. McCabe. 2006. *Introduction to the Practice of Statistics*. New York: Freeman.

Pennycuick, C. J. 1988. *Conversion Factors: SI Units and Many Others*. Chicago: University of Chicago Press.

Preece, D. A. 1981. Distribution of final digits in data. *Statistician* 30: 31–60.

Tufte, E. R. 1990. *Envisioning Information*. Cheshire, CT: Graphics Press.

Tukey, J. W. 1972. Some graphic and semigraphic displays. In *Statistical Papers in Honor of George W. Snedecor*, ed. T. A. Bancroft and S. A. Brown, 293–316. Ames, IA: Iowa State University Press.

———. 1977. *Exploratory Data Analysis*. Reading, MA: Addison–Wesley.

Wild, C. J., and G. A. F. Seber. 2000. *Chance Encounters: A First Course in Data Analysis and Inference*. New York: Wiley.

About the author

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