The Stata Journal (2018)
**18**, Number 3, pp. 741–754

# Speaking Stata: From rounding to binning

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

**Abstract.** This is a basic review of how to bin variables in Stata, meaning how to divide their range or support into disjoint intervals. I survey rounding functions with emphasis on floor and ceiling functions as tools to get clearly defined intervals of equal width. Using a specific display format is usually a better idea than rounding to multiples of a fraction. Quantile binning is popular in several fields. I give tips and tricks on how to produce such bins and also on how to show their limitations. Experimentation with the `display` command or Mata is a good way to learn about functions and to test binning rules.

**Keywords:** dm0095, binning, rounding, format, display, quantiles

## 1 Introduction

The previous column (Cox 2018) said much about binning on logarithmic scales. Here I back up to cover some basic ideas about binning and its fraternal twin, rounding. Most of what follows is elementary, meaning fundamental as well as introductory. Questions on Statalist and elsewhere often show minor and even major confusion about what is needed when researchers want to round or bin. How to round or bin in Stata is far from the only question; exactly what is wanted and even whether it is a good idea may be moot.

For the most part, I will steer clear in what follows of the vital statistical question of whether coarsening a numerical scale by binning will help in analysis. The humble histogram illustrates many of the issues. Unless variables are discrete, binning is a means to an end, simplifying a distribution by aggregating. At best, the information lost from the graph is noise. At worst, it is interesting, informative, or important fine structure. You are likely to know many alternatives to histograms that may do better. Beyond visualization of distributions, the gains and losses from binning are harder questions whenever researchers focused on predictive modeling are tempted to bin predictor variables or even outcome variables. For incisive discussion, see Harrell (2015, 18–21) or Bennette and Vickers (2012).

In most projects, exactly how you bin or round is likely to be a trivial detail, or at least you hope that it is. The principle here is that you should still aim to know exactly what you did so that you can understand and reproduce it and so that other researchers can do likewise. Further, exactly how you bin or round becomes a crucial detail whenever binning or rounding is problematic. Thus, there is clear merit in methods that are easy to explain and document.

 

This account cannot claim comprehensiveness on binning schemes. There are many ways to subdivide a range, most of them defensible. Much of what is well known, say, in cartography and geographical information science could be of interest or use to readers (Evans 1977; Kraak and Ormeling 2013; Field 2018). Bins that overlap by design are sometimes useful for graphical exploration (Cleveland 1993) but are not further discussed here.

## 2 Rounding functions

Let's dive straight in and look at the rounding functions available in Stata, which are the first place to look for solutions. The functions in Stata include `int()` (`trunc()` is a synonym), `round()`, `floor()`, and `ceil()`. A little oddly at first sight, Mata has `trunc()` but not `int()`, but more on that point in a moment. Otherwise, the functions are the same in Mata as in Stata.

Puzzlingly or not, I like two of these functions greatly and am wary of the others. It may seem strange to have small passions or prejudices about rounding functions, but I will explain.

`int()` or `trunc()` rounds toward zero, or more precisely to the nearest integer in the direction of zero. So it rounds positive numbers down if it changes them and negative numbers up if it changes them. Otherwise put, decimal parts are lopped off, and each number is truncated to its integer part, hence the name `trunc()`.

Two overarching tips for this column are about how to experiment when you are reaching toward code. Use `display` on simple examples for which you know what you want. Use Mata on slightly more complicated examples.

```
. display int(3.4)
3
. display int(-3.4)
-3
. mata : trunc((3.4, -3.4))
        1    2
  1 |    3   -3 |
```

So the function does what the documentation says. Further, Mata seems comfortable about being given several numbers at once. But there is a small twist that may be new to you. `trunc()` in Mata expects just one argument at a time, or in other words, one thing to be fed to it. It is happy with one vector or one matrix. The inner parentheses and the comma within bind the numbers 3.4 and −3.4 into a vector. Hence, the argument is `(3.4, -3.4)`. The outer parentheses are part of the function syntax, `trunc()`. You need both pairs of parentheses for this example.

Having introduced that pair of functions, I now explain why I will not use them further.

First, I often want to round up or to round down. I do not often find myself wanting to do both at the same time. Admittedly, there are occasions when you do want to use only the integer part. Producing stem-and-leaf plots is one such occasion (Cox 2007).

Second, functions named `int()` or `INT()` are common across computing in various programming languages and applications programs. So the good news is that they can be recognized fairly easily as functions rounding to integers. The bad news is that they do not all work identically! So, for example, Microsoft Excel has `INT()`, but it always rounds down. `INT(-3.4)` yields −4.

Hence, I have some distaste for `int()` because it is far from self-explanatory and dependent on someone knowing exactly what it does (and even in which program it was done). That said, the ambiguity of `int()` does bring up why `trunc()` is a much better name.

Henceforth in this column, I will consider only `round()`, `floor()`, and `ceil()`. Before we look at them, let's sketch a common and concrete problem as context.

## 3 Rounding for binning: Integers only

Stata users often want to average or otherwise summarize variables for disjoint time intervals, such as 10- or 5- or 3-year periods. ("Disjoint" here is useful jargon for "not overlapping".) Clearly, you do not want to issue repeated `summarize` statements or even write a loop if there is an easier way to do it, and there is. `egen` will calculate group means (and many other group summaries); you just need a grouping variable. Consider the Grunfeld data bundled with Stata:

```
. webuse grunfeld
. summarize year
```

| Variable | Obs | Mean | Std. Dev. | Min | Max |
|---|---|---|---|---|---|
| year | 200 | 1944.5 | 5.780751 | 1935 | 1954 |

We have 20 years of observations, so 5 years seems a convenient averaging period.

The problem is rounding integers to other integers. We will not look at rounding numbers with fractional parts until the next section.

I am very fond of `floor()` as a solution here (Cox 2003). First, the name `floor()`, once understood as meaning rounding down, is easy to remember and to explain to others. Just imagine looking down toward the floor to imprint the name. `floor()` by itself just rounds down to the nearest integer, but there is an easy twist to get what we want for integer multiples.

Always experiment when you are uncertain about the code you need: in this territory, silly little errors lie in wait all around. Let's play with the years 1935 to 1944 as enough of a check on any code. Fire up Mata with the command `mata` (no following colon) and type

```
: years = (1935..1944)
: years
        1      2      3      4      5      6      7      8      9     10
  1 |  1935   1936   1937   1938   1939   1940   1941   1942   1943   1944 |

: 5 * floor(years/5)
        1      2      3      4      5      6      7      8      9     10
  1 |  1935   1935   1935   1935   1935   1940   1940   1940   1940   1940 |
```

So, in the easiest case, dividing by 5 before you use `floor()` and then multiplying back by 5 is enough to round in 5-year periods. The years 1935–1939 round down to 1935, 1940–1944 round down to 1940, and so on. A solution with `egen` is now in hand. We have seen that the Grunfeld data have a variable `year`, so we type

```
. generate year5 = 5 * floor(year/5)
. egen mean_mvalue = mean(mvalue), by(year5)
```

Let's suppose that the data were more awkward for that trick to be enough, say that they ran from 1936 to 1945. That is also easy, but we need to switch to the twin function `ceil()`, which is short for ceiling (Cox 2003). You look up to the ceiling, and `ceil()` always rounds up. Once more in Mata, type

```
: years = (1936..1945)
: 5 * ceil(years/5)
        1      2      3      4      5      6      7      8      9     10
  1 |  1940   1940   1940   1940   1940   1945   1945   1945   1945   1945 |
```

Here is a more awkward problem. Suppose that the years included started in 1937 and ran to 1946. As before, looking at the first 10 years makes the difficulty clear.

```
: years = (1937..1946)
: 1 :+ 5 * ceil((years :- 1)/5)
        1      2      3      4      5      6      7      8      9     10
  1 |  1941   1941   1941   1941   1941   1946   1946   1946   1946   1946 |

: 2 :+ 5 * floor((years :- 2)/5)
        1      2      3      4      5      6      7      8      9     10
  1 |  1937   1937   1937   1937   1937   1942   1942   1942   1942   1942 |
```

You get the idea. You may need to shift the vector by subtracting a constant and then adding it back. On the sly, more Mata details are revealed here. Often Mata wants you to spell out that you want elementwise calculation. Colon operators `:+` and `:-` are addition and subtraction operators but with a colon prefix to flag that you want results element by element.

At this point, the problems may seem to require more trickery than you want. But experience underlines that fluency with these and many other functions does pay off. For further discussion on the theme, see a previous column (Cox 2011). It is possible to imagine a general binning function—or more likely command—with arguments or options to indicate origin, width, rounding up or down, and possibly other choices. The tradeoff is delicate: you would need to study its documentation and work out which options you needed. Having a suite of functions to use directly is the better deal.

Historical note: The names "floor" and "ceiling" were introduced by Iverson (1962). Kenneth Eugene Iverson (1920–2004) is remembered as the principal architect behind the programming languages APL and J, neither of which ever achieved many users. Those languages, however, have had an enormous diffuse influence, detectable in any language in which arrays of any kind are entities in their own right that can be used and manipulated directly. Iverson also gave a very big push to the idea—growing out of Boolean logic and used by Stata, Mata, and many other languages—that true or false expressions are evaluated as 1 and 0, respectively.

Floor and ceiling as function names from Iverson's original introduction of the notation that became APL caught on quickly as evocative and easy to remember. Sometimes, it does take a smart person to come up with simple and useful and obvious names. Note in particular the adoption and use of those function names by Donald E. Knuth in successive editions of his magisterial set *The Art of Computer Programming* (1968, 1973, 1997) and in the typesetting languages in the TEX family, used by Stata for its documentation and publications since early days. See also Graham, Knuth, and Patashnik (1994, chap. 3) for an especially thorough account of floor and ceiling functions.

One bonus is worth a small flag. If you bin 1935–1939 to 1950–1954 as 1935(5)1950, no one will care one bit exactly how you did it except other people who want to do that too. But not only is that possible with a one-liner, it also has a mathematical expression as $5\lfloor \text{year}/5 \rfloor$.

So far, so good. But what about `round()`, which may well be better known?

```
: years = (1935..1944)
: round(years, 5)
         1      2      3      4      5      6      7      8      9     10
  1   1935   1935   1935   1940   1940   1940   1940   1940   1945   1945

: round(years :+ 3, 5)
         1      2      3      4      5      6      7      8      9     10
  1   1940   1940   1940   1940   1940   1945   1945   1945   1945   1945

: round(years :- 2, 5)
         1      2      3      4      5      6      7      8      9     10
  1   1935   1935   1935   1935   1935   1940   1940   1940   1940   1940
```

```
: 2 :+ round(years :- 2, 5)
          1      2      3      4      5      6      7      8      9     10
  1 |  1937   1937   1937   1937   1937   1942   1942   1942   1942   1942 |
```

Points from earlier recur. You may need to add or subtract a constant. There is also scope for getting a middle year, which could seem congenial or at least convenient (for example, for graphing). That could be attractive when the number of times being averaged is an odd number like 5 or 3 or 7. Note in passing that `round()` allows two arguments. When there is a second argument, it controls how much rounding takes place.

What happens when you round to multiples of 2? It is obvious that even numbers will round to themselves, but will odd numbers round up or down?

```
: years \ round(years, 2)
          1      2      3      4      5      6      7      8      9     10
  1 |  1935   1936   1937   1938   1939   1940   1941   1942   1943   1944 |
  2 |  1936   1936   1938   1938   1940   1940   1942   1942   1944   1944 |
```

There is an answer for you, but I do not like to have to remember that answer or even re-create it every time I wonder, which is one reason I tend to avoid `round()`. The extra Mata detail here is using the backslash to put one row vector below another.

## 4 Pitfall: Rounding as a poor way to format

Let's bring numbers with decimal parts into the discussion. Now matters become murkier. This section is a warning of a common confusion. People often want to round off fractions to a moderate number of decimal places. Your teachers explained this as good practice early in your education. `round()` allows fractional second arguments, so it may seem to be a way forward.

Suppose you have a correlation that you want to round sensibly. `correlate` yields Pearson correlations and by default shows four decimal places. Conversely, it leaves the r-class stored result in memory, which lets you see even more decimal places. (That could be important if you wanted to check correlation results across programs for correct calculations.) We are still using the Grunfeld data, although any dataset would suffice.

```
. correlate mvalue invest
(obs=200)
             |   mvalue   invest
-------------+------------------
      mvalue |   1.0000
      invest |   0.8569   1.0000

. display r(rho)
.8569329
```

Let's imagine that two decimal places would be enough. Many users approach this kind of problem using `round()` with a second argument, 0.01, for two decimal places (or 0.1 or 0.001 for one or three decimal places, and so on).

```
. display round(r(rho), 0.01)
.86
. display %23.18f round(r(rho), 0.01)
   0.859999999999999987
```

What is going on here? Did that work or not? The short answer is that it seemed to work at first, but then you might wonder. An explanation comes on two levels.

On the surface: If you use `display` without specifying a format, then its default comes into play. The default format of `display` usually does a good job of rounding off trivial detail. So the first command immediately above shows you a combination of what was done to `r(rho)` and what `display` does anyway. If we insist on many decimal places with a specific format, what we usually see is that the rounding was approximate, not exact.

Deeper down: The problem here is that Stata works in binary arithmetic but defaults to showing decimal results. Many simple decimal fractions fail to have exact binary representations. Among the fractions from 0.1 to 0.9 in steps of 0.1, only 0.5 (as 1/2 or $2^{-1}$) can be held exactly in binary. Among the fractions from 0.01 to 0.99 in steps of 0.01, only 0.25, 0.50, and 0.75 can be held exactly in binary.

Resources for understanding this and related questions on how Stata holds and works with numbers are revealed by `search precision` in Stata. I strongly recommend the blog posts by William Gould.

There is a better solution than using `round()` with a fractional second argument: Using a display format suitable for the purpose. If needed, the result of that formatting with an individual correlation or a similar result can be put in a local macro for later use.

```
. display %03.2f r(rho)
0.86
. local wanted: display %03.2f r(rho)
. display "`wanted´"
0.86
. display `wanted´
.86
```

Here we asked for, and got, a display with leading zeros and just two decimal places. Putting that into a local macro allowed us to play it back, which is often useful. Whether in graphics or in other reporting, you often want to round results that will be given as part of a title, table, or other text.

But that still leaves the different results of the last two commands just above to explain. The local macro `wanted` contains the text string `0.86`. That makes sense to people as a numeric result, but as said, it is just a text string to Stata. Using double

quotes in a `display` command emphasizes that you want to see the exact contents of a quoted string, which is just text. If you leave out the double quotes, Stata first sees the macro reference and interprets that immediately. The result is the instruction to

```
. display 0.86
.86
```

As just shown, `display` then uses its default format and does not show the leading zero. As we have seen, we can insist on seeing leading zeros with a suitable display format.

This section is about a problem in which display format is the answer. For the opposite problem in which people expect more of changing the format than it can achieve, see Cox (2012b).

## 5 Rounding for binning: General case

Let's return to binning in general. Everything to be said has already been said. We just need to think through the principles in concrete cases. Thus, from good introductory texts and courses, it should be clear that bins being presented as, say, 0–2, 2–4, 4–6, and so forth is sloppy presentation. Yule (1911) and Freedman, Pisani, and Purves (2007) can stand representative of many such explanations over a century or more. Even earlier, Galton (1885a,b) underlined, although not in these words, that ambiguous bin limits could be misinterpreted; see Pearson (1924, 377) for summary and comment.

In a nutshell, the bounds are ambiguous in such presentation. What happens if any data value is an exact multiple of 2? Do values go up or down from the boundary into the higher or lower class?

The beauty of using `floor()` or `ceil()` to bin is that this ambiguity does not exist. With `floor()`, lower limits or bounds are inclusive; with `ceil()`, upper limits or bounds are inclusive.

## 6 Quantile binning

Researchers in some fields like to bin their variables in groups that are, or should be, equally numerous and whose limits are defined by quantiles equally spaced on a cumulative probability scale. Thus, four quintiles corresponding to cumulative probabilities 0.2(0.2)0.8 delimit five bins, classes, or intervals, subject to small print about how the limits are treated. Or 9 deciles for cumulative probabilities 0.1(0.1)0.9 delimit 10 bins, and so forth. Such binning, on evidence from Statalist, seems especially popular for work on data from business and finance. Performance is relative as well as absolute, although that alone does not imply using bins rather than, say, ranks or percentile ranks (associated cumulative probabilities).

An aside on terminology: A collection of terms for such quantiles was given in Cox (2016), to which can be added "trentile" (Slonim 1958) for division into 30 bins. Historically, the quantile terms such as tertiles, quartiles, quintiles, and their kin, and also the generic term quantile itself, referred to point summaries of a variable, and not

to the bins they delimit. However, it has become common to extend those terms to the bins as well as their limits. Thus, the first quintile might mean the bin below the first quintile as well as that first quintile. Such extension can be criticized as awkward and ambiguous (I agree with that) and defended as now widely familiar and even customary (I agree with that too).

If all values of a variable are distinct (no value occurs twice or more) and missing values are not a problem, then we could produce quantile bins directly with something like this for some generic variable `X`:

```
. sort X
. generate bin5 = ceil(5 * _n/_N)
```

Here we follow a common convention and label quantile bins 1 and above. Wanting to do this groupwise, say, within panels or within time periods, needs only a twist such as (again, a generic identifier `id`)

```
. bysort id (X): generate bin5 = ceil(5 * _n/_N)
```

Recall, or otherwise please learn now, that in Stata, `_n` is the observation number running 1 and above and `_N` is the total number of observations. Under the aegis of `by:`, both are counted within each group defined by `by:`. Hence, the fraction `_n/_N` varies from just above 0 (strictly `1/_N`) to 1 (the observation number `_n` becomes equal to the number of observations `_N` when you reach the last observation). Multiplying by the number of bins multiplies that range so that the upper limit is now the number of bins. As before, the ceiling function `ceil()` rounds up so that 0.123 or 1.456 would be rounded up to 1 and 2.

Using `floor()` would not be a good idea in this case, but not because the bins would be labeled 0 and above. The problem is that the last observation for which `_n/_N` is 1 would always be in a bin by itself.

Either of these commands would be a mildly entertaining one-line solution to quantile bins—except that data are usually more complicated. Only exceptionally is the number of values a multiple (meaning an integer multiple) of the number of bins, so divisibility is usually a small problem. Thus, `auto.dta`, familiar to Stata users, contains 74 observations. A dataset size of 74 is divisible only by 2 and 37 and is otherwise awkward for common choices such as 5 or 10 bins. For one variable, repair record `rep78`, there are only 69 nonmissing values, a number divisible only by 3 and 23. This kind of problem is common, but fortunately trivial once understood.

More crucially, a practical command should handle missing values directly and cope with tied values. To make the problem of ties concrete, consider wanting to split 1, 2, 2, 3, 3, and 4 into 3 (tertile) bins. Here divisibility by 3 is not an issue, but ties are evidently problematic. Any solution of splitting (1,2), (2,3), and (3,4) is no solution, because it defies the principle that equal values belong in the same bin. Even if that were thought dispensable, equal values are unlikely to be equal on other variables. The two values of 2 may be equivalent to each other; the problem is that typically they correspond to different values on other variables.

Stata has a built-in command, `xtile`, to produce quantile bins. The rules it follows are that 1) each quantile calculated is the upper limit of its bin and 2) the maximum is the upper limit of the last bin. Denote the $k$th quantile of $y$ for $K$ bins by $y_k$, and define $y_0$ and $y_K$ as the minimum and maximum of $y$. (Fastidious readers queasy at my hijacking notation commonly used otherwise may wish to invent their own alternative.) Then bins as calculated by `xtile` run $[y_0, y_1], (y_1, y_2], \ldots, (y_{K-1}, y_K]$ and the formula for all but the first is $(y_{k-1}, y_k]$. Here we use a common notation for intervals: $(a, b]$ means for $y$ that $a < y \leq b$ and $[a, b]$ means for $y$ that $a \leq y \leq b$.

To make this concrete, we turn to `auto.dta` and ask for five bins of `mpg`:

```
. sysuse auto, clear
(1978 Automobile Data)
. xtile mpg5 = mpg, nquantiles(5)
. _pctile mpg, nquantiles(5)
. return list
scalars:
                  r(r1) =  17
                  r(r2) =  19
                  r(r3) =  22
                  r(r4) =  25
. tabstat mpg, by(mpg5) statistics(n min max)
Summary for variables: mpg
     by categories of: mpg5 (5 quantiles of mpg)
```

| mpg5 | N | min | max |
|---|---|---|---|
| 1 | 18 | 12 | 17 |
| 2 | 17 | 18 | 19 |
| 3 | 13 | 20 | 22 |
| 4 | 12 | 23 | 25 |
| 5 | 14 | 26 | 41 |
| Total | 74 | 12 | 41 |

At the time of writing, `xtile` has various limitations. First, it does not display the quantiles it uses, nor does it leave them as stored results. To discover what they are, you could, for example, type `_pctile` or `tabstat`. The latter has the advantage, as just shown, of showing bin frequencies directly. Second, `xtile` does not support the `by:` prefix, so you need some kind of work-around for groupwise binning. Third, `xtile` can be slow for large datasets. The first limitation is easily addressed, as already shown. Community-contributed commands can be found addressing the other two limitations, but it seems likely that StataCorp will address these problems within the useful life of this column. Otherwise, I suggest that you search Statalist for recommendations.

As the table just given emphasizes, quantile bins produced by `xtile` are necessarily disjoint. Any values equal to a calculated quantile are all assigned to the same bin, and this can lead to bins with markedly unequal frequency, especially if there are relatively few distinct values. An absolute limit that sometimes puzzles people is that the number of bins cannot possibly exceed the number of distinct values in the data!

A quantile plot (Cox 2005, fig. 1) can help emphasize how quantile binning works, especially when the example dataset is small to moderate in size. I have found this plot useful with people quite new to the idea. It does emphasize where the limits lie and also does show directly how much variability there is within each bin. In this case, and indeed commonly, the uppermost bin is the most variable. As the `tabstat` results also underline, its range even exceeds the range of the other bins combined. Notice, as a Stata detail, how we suppress the default marker symbol and put a marker label where it would have been.

```
. quantile mpg, msymbol(none) mlabel(mpg5) mlabposition(0) rlopts(lcolor(none))
```

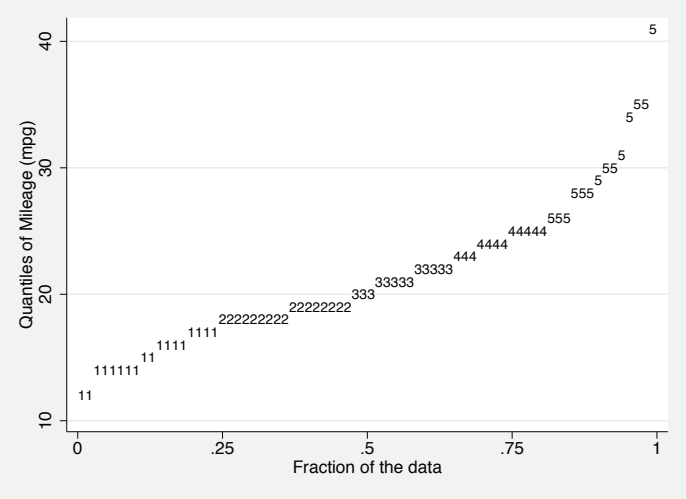


Figure 1. Quantile plot showing five bins for `mpg` in `auto.dta`

Unequal bin frequencies raise the question of whether a different binning convention might produce a better result (Cox 2012a). The title of that column unfortunately hides the fact that it includes a section on binning. Conversely, the column shows how matrices used as look-up tables can be used for quantile binning and indeed more widely.

We could defensibly have bins $[y_0, y_1), [y_1, y_2), \ldots, [y_{K-1}, y_K]$ so that the notation for all but the last is $[y_{k-1}, y_k)$. As it turns out, the convention that lower limits are inclusive is in essence that used by `histogram` and `twoway histogram` in Stata. According to Venables and Ripley (2002, 112), it is "the convention that most people prefer" for histograms. (Statisticians, like anybody else, can give confident generalizations without showing or even needing empirical data.) If you want the same kind of binning, you can check out `twoway_histogram_gen` (Harrison 2005); note that there are three underscores in that command name, not two.

`xtile` does not offer this kind of binning as an option, but one easy way to subvert the command is to negate the variable and then reverse the bins. See the column just cited (Cox 2012a) for another example and another way to do it.

```
. generate negmpg = -mpg
. xtile mpg5rev = negmpg, nquantiles(5)
. replace mpg5rev = 6 - mpg5rev
(58 real changes made)
. tabstat mpg, by(mpg5rev) statistics(n min max)
Summary for variables: mpg
     by categories of: mpg5rev (5 quantiles of negmpg)
```

| mpg5rev | N | min | max |
|---|---|---|---|
| 1 | 14 | 12 | 16 |
| 2 | 13 | 17 | 18 |
| 3 | 16 | 19 | 21 |
| 4 | 12 | 22 | 24 |
| 5 | 19 | 25 | 41 |
| Total | 74 | 12 | 41 |

# 7 Conclusion

Binning is a basic device in statistical analysis, but it needs to be applied clearly, carefully, and critically. Knowing useful functions and commands to bin variables is the essential first step, but checking the results to see that they are fit for purpose is just as necessary. In the larger scheme of statistical science, exactly how to bin is just a bundle of small details, but any aim of reproducible research requires knowing and stating explicit and unambiguous rules.

# 8 References

Bennette, C., and A. Vickers. 2012. Against quantiles: Categorization of continuous variables in epidemiologic research, and its discontents. *BMC Medical Research Methodology* 12: 21.

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

Cox, N. J. 2003. Stata tip 2: Building with floors and ceilings. *Stata Journal* 3: 446–447.

———. 2005. Speaking Stata: The protean quantile plot. *Stata Journal* 5: 442–460.

———. 2007. Speaking Stata: Turning over a new leaf. *Stata Journal* 7: 413–433.

———. 2011. Speaking Stata: Fun and fluency with functions. *Stata Journal* 11: 460–471.

———. 2012a. Speaking Stata: Matrices as look-up tables. *Stata Journal* 12: 748–758.

———. 2012b. Stata tip 113: Changing a variable's format: What it does and does not mean. *Stata Journal* 12: 761–764.

———. 2016. Speaking Stata: Letter values as selected quantiles. *Stata Journal* 16: 1058–1071.

———. 2018. Speaking Stata: Logarithmic binning and labeling. *Stata Journal* 18: 262–286.

Evans, I. S. 1977. The selection of class intervals. *Transactions of the Institute of British Geographers* 2: 98–124.

Field, K. 2018. *Cartography*. Redlands, CA: Esri Press.

Freedman, D., R. Pisani, and R. Purves. 2007. *Statistics*. 4th ed. New York: W. W. Norton.

Galton, F. 1885a. A common error in statistics. In *Jubilee Volume of the Statistical Society*, 261. Charing Cross London: Edward Stanford.

———. 1885b. Some results of the anthropometric laboratory. *Journal of the Anthropological Institute of Great Britain and Ireland* 14: 275–287.

Graham, R. L., D. E. Knuth, and O. Patashnik. 1994. *Concrete Mathematics: A Foundation for Computer Science*. 2nd ed. Reading, MA: Addison–Wesley.

Harrell, F. E., Jr. 2015. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*. 2nd ed. Cham, Switzerland: Springer.

Harrison, D. A. 2005. Stata tip 20: Generating histogram bin variables. *Stata Journal* 5: 280–281.

Iverson, K. E. 1962. *A Programming Language*. New York: Wiley.

Knuth, D. E. 1968. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Reading, MA: Addison–Wesley.

———. 1973. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. 2nd ed. Reading, MA: Addison–Wesley.

———. 1997. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. 3rd ed. Reading, MA: Addison–Wesley.

Kraak, M.-J., and F. Ormeling. 2013. *Cartography: Visualization of Spatial Data*. New York: Routledge.

Pearson, K. 1924. *The Life, Letters and Labours of Francis Galton. Volume 2: Researches of Middle Life*. Cambridge: Cambridge University Press.

Slonim, M. J. 1958. The trentile deviation method of weather forecast evaluation. *Journal of the American Statistical Association* 53: 398–407.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with S*. 4th ed. New York: Springer.

Yule, G. U. 1911. *An Introduction to the Theory of Statistics*. London: Griffin.

**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 an editor 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).