



*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: How best to generate indicator or dummy variables

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

Clyde B. Schechter  
Albert Einstein College of Medicine  
Bronx, NY  
clyde.schechter@einstein.yu.edu

**Abstract.** Indicator or dummy variables record whether some condition is true or false in each observation by a value of 1 or 0. Values may also be missing if truth or falsity is not known, and that fact should be flagged. Such indicators may be created on the fly by using factor-variable notation. `tabulate` also offers one method for automating the generation of indicators. In this column, we discuss in detail how otherwise to best generate such variables directly, with comments here and there on what not to do.

**Keywords:** dm0099, indicator variable, dummy variable, true or false, any, all, missing values, logical and relational operators, functions, merge

## 1 Introduction

We define indicator or dummy variables at the outset as numeric variables taking on a value of 1 when some condition is true and a value of 0 when that condition is false. We leave open the possibility of the definition being a little more complicated, which we discuss later. We also leave open the possibility of true and false states being coded in some other way, which we also discuss later, although we will fall far short of a complete discussion of the possibilities.

After a fairly broad introduction to the subject, we will explain good ways to produce such variables. We further highlight some unnecessarily incomplete or indirect methods that we often encounter.

For more on the idea that 1 and 0 may encode true and false conditions, including some historical notes and references, see Cox (2016b).

## 2 What's in a name?

Some discussion of terminology is needed before we get to the main business. There is a simple, wider use of indicator variables for anything quantitative that indicates, say, health, well-being, or general system state or operation. Systolic blood pressure, stock price, or soil moisture content can be key indicators for people in particular fields. In practice, that does not conflict with the specialized use here.

The specialized sense of a variable with values of 0 and 1 seems to grow out of the mathematical idea of an indicator function. See, for example, Zeitz (2007) for a friendly explanation of that and much else. In some parts of mathematics, the term “characteristic function” has the same meaning, but for statistical people, that term is already used for a quite different function arising in the analysis of probability distributions. Both indicator function and indicator variable are key ideas in probability theory, which is perhaps the main way in which they entered statistical science generally. Feller (1971, 104) attributed the term to Michel Loève. Other references include Moran (1968, 7), Whittle (2000, 7), and Blitzstein and Hwang (2015, 100).

The term “dummy variable” for a variable encoded with values of 0 and 1 seems to have arisen around 1950 in statistical science. Freedman (2009, 113) cites Oakland (1950) and Klein (1951). We can add a reference to Quenouille (1950). Whatever the roots, “dummy variable” seems especially popular at present as a term in economics and other social sciences. As you may recall from your mathematical education, there is a different and older sense of the term (for example, in calculus) as an arbitrary variable playing a filler role in some formalism.

Training and taste affect what terms people use. We try to avoid the term “dummy variable” because it is at least a little awkward. We have heard too many stories of the term being horribly misunderstood as pejorative or disparaging. That has sometimes happened when quantitatively minded researchers try to explain what they are doing to people with different backgrounds. Misunderstanding can run in any direction, which is why terms that can possibly offend should be avoided.

We should mention other terms used for variables indicating which of two possible states occurs in each observation, such as binary, dichotomous,<sup>1</sup> Boolean, or quantal. “Binary” is an excellent term likely to be familiar already. The other words have interesting histories but also a need to explain to beginners how to spell them and what they mean.

### 3 Why use indicator variables?

We can identify three main reasons why indicators might be useful:

- *Coding a response or outcome or dependent variable.* Many responses of interest are binary, such as survived or not, present or absent, or success or failure. Many standard models, such as logit and probit regression and various more exotic relatives, are dedicated to analysis for such response variables.

---

1. Having mentioned it, it may be helpful to explain that “dichotomous” should be parsed using its Greek roots as *dicho*-*tomous*, not *di*-*chotomous*. Hence, *polytomous*, not *polychotomous*, is a correct analogous term for many states rather than two. So if fruits can be apple, orange, banana, or any other named fruit, we have passed beyond a binary variable to one with many possible states that some are tempted to call *polytomous*. Better advice is to use a word like “multistate” unless your audience knows Greek.

- *Coding a predictor or descriptive variable.* Many variables that are descriptive or possibly predictive can be coded 0 or 1, and this can be useful too. You may already be familiar with simple or even complicated uses of binary or indicator predictors.

If this is new to you, here is a first example. Given  $x_1$  that is counted or measured and  $x_2$  that is 0 or 1, a regression of  $y$  on those predictors,

$$y = b_0 + b_1x_1 + b_2x_2$$

reduces to two parallel functions:

$$y = b_0 + b_1x_1 + b_2$$

when  $x_2 = 1$  and

$$y = b_0 + b_1x_1$$

when  $x_2 = 0$ . This is a neat but simple way to combine a quantitative and a binary predictor in a model. Hence,  $b_2$  is the estimate of the difference that it makes to  $y$  if  $x_2$  is 1 rather than 0. The idea can be taken further to include interactions as most good texts on regression or econometrics do explain these.

In general, this is the territory of categorical variables. Even multistate categorical variables can be coded using indicators. For example, collectively being an orange, apple, or banana, or some other fruit can be coded using two or more indicator variables for orange or not, apple or not, and so forth. One of those variables will be redundant. In this example, if the only possibilities are orange, apple, or banana, then not being an orange or apple is enough information to identify a banana.

- *Other uses, especially for data management.* This reason is a catch-all for all the other possibilities. We often find that an indicator is a useful stepping stone for data management. It might flag which observations have missing values, deserve keeping, deserve dropping, and so on. A small but often overlooked trick to get proportions is to take the mean of an indicator that is 0 or 1, often with [D] **egen**. Essentially the same trick to get a percent is to take the mean of 100 times an indicator or to multiply a proportion by 100; the result is identical either way.

## 4 How to produce indicator variables: Automation

This section includes the most important advice in this column. **tabulate** (see [R] **tabulate oneway**) can produce indicator variables for you. This is particularly helpful in producing several indicator variables at once from a multistate categorical variable. For example, we can use **auto.dta** and type the following:

```
. sysuse auto
(1978 Automobile Data)
. tabulate rep78, generate(rep78_)
```

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

We created the new variables `rep78_1` to `rep78_5`. `rep78_1` is 1 if `rep78` is 1 and is 0 otherwise; `rep78_2` is 1 if `rep78` is 2 and is 0 otherwise; and so on.

Once you know how to do it, factor-variable notation can easily be used to specify that numerically coded categorical predictor variables should be treated as one or more indicators. There are many advantages to that and no real disadvantage beyond needing to learn the syntax. For an introduction, see [U] **11.4.3 Factor variables**.

The rest of this column applies to situations where you have the need or inclination to generate indicators directly.

## 5 How to produce indicator variables: Do it yourself

Let us continue to work with Stata's `auto.dta`.

We quite often see something like this on Statalist:

```
. generate himpg = 1 if mpg > 30
```

This creates a variable that is 1 if `mpg > 30` and that is missing (system missing, or `.`) otherwise. Sometimes people follow with

```
. replace himpg = . if mpg <= 30
```

That has the merit of spelling out—to those reading a log file as a transcript of a Stata session, which could include the researcher who produced the log at a later date—that values that are not 1 are all missing. Otherwise, the statement is redundant because it changes nothing.

Such a variable with values of 1 and missing is of limited use in Stata. It cannot be used in statistical modeling because the observations with missing values will be omitted, and the observations with a value of 1 will define a constant variable, which itself will be omitted from any model as a predictor. At the end of this column, we will mention an exception to our assertion.

This is better:

```
. generate himpg = 1 if mpg > 30
. replace himpg = 0 if mpg <= 30
```

Now we will have a variable that is 1 or 0, assuming naturally that there are some observations satisfying each of the two states, as indeed there are in `auto.dta`.

There is a better way still. Did you know that you can do this?

```
. generate himpg = mpg > 30
```

Given an expression that can be true or false when evaluated, Stata will return 1 when the expression is true and 0 when it is false. So, with such a **generate** statement, Stata automatically loops over observations, returning 1 for observations for which the expression is true and 0 for the others.

Incidentally, many programming languages (we include Stata in that group) are way ahead of general mathematical practice in allowing such expressions. It is possible to specify indicator functions in mathematical writing, but no one seems to have thought up a notation that everyone else likes. For  $x > 30$ , we have seen  $I(x > 30)$ ,  $i(x > 30)$ ,  $1(x > 30)$ , and others yet, to say nothing of different choices of fonts or pushing those expressions into subscripts. Iverson (1962), de Finetti (1967, 1972, 1974, 1975), and Knuth (see, particularly, his 1992 paper) encouraged  $x > 30$  as acceptable notation, although sometimes with surrounding parentheses or square brackets.

Also incidentally, but a crucial question for researchers: examples like this where we split a measured variable into two intervals raise the key question of whether that splitting is a good idea. Cox (2018) airs this question and points to detailed critical discussions elsewhere.

## 6 What about missing values?

We have not told the whole truth yet. Most importantly, the `mpg` variable is well behaved with no missing values. If there were any missing values, then they too would count as greater than 30. Missing values of numeric variables are treated in Stata as arbitrarily large. Occasionally, you know independently that missing really does mean “larger than any measured value”, but more commonly, missing just means that values are unknown or even not applicable. The trick we have just explained can be extended easily:

```
. generate himpg = mpg > 30 if mpg < .
```

Now there are three possible values: 1 or 0 as before, and missing if `mpg` is missing, because observations with missing values on `mpg` are excluded from the calculation and the consequence is a result of missing.

We may exclude missing values with the condition `mpg < .` for two reasons.

- To save a keystroke over `mpg != .`. Do not laugh or scoff: over many years of intensive Stata use, the saving may be worthwhile. This trick was written up by Lachenbruch (1992).
- To exclude extended missing values `.a` to `.z` too, each of which is regarded as larger than system missing. If you do not know about them, check out [U] **12.2.1 Missing values** at your convenience.

That said, many people may find

```
. generate himpg = mpg > 30 if !missing(mpg)
```

more transparent. Your collaborators or reviewers may know little or no Stata but still want to look at your code. If so, you should want to avoid tricks specific to Stata. Indeed, people you work with may be more familiar with some alternative software with the opposite convention: missing values are treated as arbitrarily large negative numbers. Naturally, it remains true that `!` has to be understood as logical negation (not), but that is syntax used in many programs or environments.

## 7 In a subset?

We can say that indicator variables take on a value of 1 if observations lie in a subset of possibilities and 0 if they lie in the complementary subset. We now mention some other problems in which this idea arises, with references to discussions elsewhere.

In this section, the possibility of missing values is set aside. If you want a small challenge, think through what happens in these commands if missing values are present and so what might need to be changed.

### 7.1 Compound conditions

Sometimes true-or-false conditions involve compound conditions using two or more relational and logical operators. See `help operators` as desired for a list of such operators.

The flavor is given by

```
. generate foreign_himp = (foreign == 1) & (mpg > 30)
```

because being a foreign car with high mileage requires both a condition of being foreign and a condition of high mileage. Here, the parentheses are dispensable but may help to underline the logic.

## 7.2 Any and all problems

Do you know these twin tricks?

- Finding whether any values satisfy a condition corresponds to looking for a maximum over indicator values.
- Finding whether all values satisfy a condition corresponds to a looking for a minimum.

Let's imagine data on people (observations) within households (groups of observations). Suppose we are trying to classify households into those with and without children. To be concrete, we take children to mean **age** 17 or younger. Given an identifier **hhid**, then typing

```
. egen any_children = max(age <= 17), by(hhid)
```

is an answer in one line. The expression **age <= 17** is a true-or-false expression of the kind now familiar. After first evaluating the expression as 1 or 0 for each observation, **egen** works out the maximum within groups defined by **hhid**. If any value in a household is 1, then so is the maximum. If all values are 0, then so is the maximum. So we now have our indicator defined for households.

We could flip the problem around and look for households in which every person is an adult (18 or over in **age**).

```
. egen all_adult = min(age > 18), by(hhid)
```

If all values are 1, then so is the minimum; otherwise, 0 is returned. Or to translate all to any: if any value is 0, so is the minimum.

Various types of problems can arise when the comparison is not between the values of a single variable in a group of observations, but between the values of two or more variables. Solutions for this version of the problem are both easy and plentiful. The functions **max()** and **min()** are available for rowwise comparisons, as are the **egen** functions **rowmax()** and **rowmin()**. On the surface, the difference is minor: **max()** and **min()** require comma-separated lists of arguments, while the **egen** functions will accept a wildcarded *varlist*. Hence, if **x1** and **x2** were indicators and the compound condition is that either is true, then

```
. generate either = max(x1, x2)
```

and

```
. generate either = x1 | x2
```

are equivalents so long as the possible values are just 0 and 1. Similarly,

```
. generate both = min(x1, x2)
```



and

```
. generate both = x1 & x2
```

are equivalents with the same reservation.

However, if the condition involved, say, 42 variables of a similar kind (for example,  $x_1, \dots, x_{42}$ ), then you are likely to prefer something like these commands:

```
. egen all = rowmin(x1-x42)
. egen any = rowmax(x1-x42)
```

For more on any and all problems, see Cox (2016a). For more on working rowwise, see Cox (2009).

### 7.3 `inlist()` and `inrange()`

Two further functions that can be extremely useful in producing indicators are `inlist()` and `inrange()`. For more, see Cox (2006, 2011). We particularly note the twist that

```
. generate anyx = inlist(1, x1, x2, x3, x4, x5)
```

is another way to write

```
. generate anyx = (x1 == 1) | (x2 == 1) | (x3 == 1) | (x4 == 1) | (x5 == 1)
```

In between those statements, and also equivalent, would be

```
. generate anyx = (1 == x1) | (1 == x2) | (1 == x3) | (1 == x4) | (1 == x5)
```

Different ways of doing the calculation in Stata match different ways that we could write down the corresponding algebra. That is,  $1 = x_1$  or  $1 = x_2$  is another (if unconventional) way to write  $x_1 = 1$  or  $x_2 = 1$ . Despite the apparent symmetry implied by the equals sign, there is a strong sense in which  $a = 1$  (think  $a$  is 1) does not have the same connotation as  $1 = a$  (think 1 is  $a$ ). See de Bruijn (1958) for instructive and entertaining discussion of the ways in which notation connoting equality is handled, or mishandled, in mathematics.

### 7.4 Intersection alongside union

Sometimes a subset is specified by inclusion in quite a long list of possible values. You might have a dataset of many countries (perhaps 200 or so, depending on your definition, purposes, and data availability) and wish to indicate which belong to the Organisation for Economic Co-operation and Development (OECD). At the time of writing (27 November 2018), there are 36 countries in OECD. So you might just grimace and write out lengthy code mentioning them all.

There is an alternative that is increasingly attractive as the list of possibilities grows longer, so long as somehow you can put that list into a data file. You can use the `merge`

command (see [D] **merge**) to merge your main dataset with a dataset containing the list. Then what interests you is the intersection of the datasets.

Even the OECD countries as a subset of countries is too lengthy an example to be comfortable. No matter; the principle can be shown by a tiny sandbox. Let's imagine a panel dataset with string identifiers and (being realistic about a common complication) a time variable too. Our `main.dta` has values like these (and other variables too, but we can keep those on one side):

```
. list in 1/6, sepby(id)
```

	id	time
1.	A	1
2.	A	2
3.	B	1
4.	B	2
5.	C	1
6.	C	3

Now we need a dataset with just the wanted values, say, like this:

```
. clear
. input str1 id
      id
1. A
2. end
```

The key trick is then just a one-to-many **merge**:

```
. merge 1:m id using main
```

Result	# of obs.
not matched	4
from master	0 (_merge==1)
from using	4 (_merge==2)
matched	2 (_merge==3)

The `wanted` indicator variable is then 1 if and only if there was matching and 0 otherwise:

```
. generate wanted = _merge == 3
. drop _merge
. list id time wanted, sepby(id)
```

	id	time	wanted
1.	A	1	1
2.	A	2	1
3.	B	1	0
4.	B	2	0
5.	C	1	0
6.	C	3	0

There is naturally no rule that you must type the identifiers into a file or into Stata yourself. The ideal will be that they are already in a separate data file of some kind. For the main idea and another example, see Baum (2011).

## 8 Storage types

We have not specified variable or storage type. Many careful Stata users insist to Stata that indicator variables can and should be stored in byte variables:

```
. generate byte himpg = mpg > 30 if mpg < .
```

We will confess to being a little careless about this point on occasion. A defense for such caprice is that we can always type

```
. compress
```

if memory looks short or we need to make a dataset as small as possible. Then Stata will let us catch up on being careless about storage by using more efficient storage types whenever possible.

## 9 Fond of `cond()`?

Next comes a matter of taste. One of us (more than the other) is quite fond of `cond()` as making all possibilities explicit. Thus, consider

```
. generate byte himpg = cond(missing(mpg), ., mpg > 30)
```

`cond()` takes some getting used to at first, but there is a short tutorial to help (Kantor and Cox 2005). You may be familiar with the same idea expressed in other software using a syntax in terms of `if` and `else`. Mata (following other languages) uses the form

$TF?a:b$ , where Stata would have `cond(TF, a, b)`. Here  $TF$  is an expression evaluated as true or false,  $a$  is returned if the statement is true, and  $b$  is returned if the statement is false.

Even when familiar with the construct, we find it helpful to spell it out mentally (even aloud when alone!) as informal pseudocode.

```
if mpg is missing, then return missing;
else return mpg > 30
```

One reason for using `cond()` is to make all the possibilities explicit. We could go further:

```
. generate byte himpg = cond(missing(mpg), ., cond(mpg > 30, 1, 0))
```

That can be translated to

```
if mpg is missing, then return missing;
else if mpg > 30 return 1;
else return 0
```

Only you will know if this syntax appeals to you. The tutorial in Kantor and Cox (2005) has more tips on how to use the function.<sup>2</sup>

## 10 Choose informative variable names

One point has remained implicit so far, but now should be brought forward. A variable name like `himp` is descriptive, indicating “hi[gh] mpg”. Naturally, longer and even more descriptive names such as `high_mpg` or `mpg_gt_30` are entirely possible. Further, a variable label can and should carry explanatory detail on a new variable.

Making variable names explanatory is standard and good advice and perhaps too obvious to deserve mention, except that daily we see variable names in researchers’ datasets that carry no evident meaning. In the case of indicators, we can add further emphasis: choose a name that describes the state coded as 1. `himp` does that, as does `foreign`, an indicator variable already present in `auto.dta`. Beware the fate, or at least the embarrassment, of researchers who could not remember in presentations which way their indicators were coded. So, never use `gender` as an indicator variable name; always use `female` or `male`, as the case may be.

---

2. But you should be in agreement that one hallmark of good code is that it is readable, making as much sense as possible to anyone who has to read it. That could be you later, so whatever seems tricky now may seem cryptic later if you do not remember the trick. It could be others in your group, or examiners or reviewers of your work, looking for evidence of reproducible research. The theme of (readable and) reproducible research is too large to be discussed well here, but we remind you that you can always sprinkle your code with comments.

## 11 Alternatives to 0 and 1 as codes for binary states

Some Stata users seem to find it more convenient or congenial to code binary states as 1 and 2. That can be fine except that you will need to use factor-variable notation to use such variables as predictors and will need to subtract 1 before you can use such variables as responses or outcomes. Similar remarks apply to any other choice.

As a small programming trick, we sometimes negate indicator variables so that true becomes  $-1$  and false remains 0. The point of doing so is that if we next **sort** on the indicator variable, observations with values of  $-1$  are moved to the beginning of the dataset. We can then manage them a little more easily than if they were at the end of the dataset.

We finally redeem an earlier promise. Indicators that are 1 or missing can sometimes be useful in data management. One example is looking for the first nonmissing value of some variable in a panel, a problem discussed at greater length by Cox (2016b). Here we acknowledge gratefully an example with similar flavor by Romalpa Akzo on 7 November 2018 on Statalist; see <https://www.statalist.org/forums/forum/general-stata-discussion/general/1469251-select-first-event-or-last-non-event>.

We could type

```
. generate byte wanted = 1 if !missing(whatever)
. bysort id (wanted year): generate first = whatever[1]
```

so that the indicator variable **wanted** is 1 if **whatever** is not missing and is missing otherwise. When we sort each panel first on **whatever**, the observations with 1 on **wanted** sort before those with missing on that variable. Then ties are broken by **year** (or some other time variable), so the first nonmissing value rises to the start of each panel.

How could we extend that idea to get at the last nonmissing value in each panel?

```
. replace wanted = !missing(whatever)
. bysort id (wanted year): generate last = whatever[_N]
```

Now the observations with missing values rise to the start of each panel. After breaking ties on the time variable, the observation we want to use is at the end of that panel.

If all values of a variable are missing in a panel, then each new variable, **first** and **last**, will be returned as missing, which is only right.

## 12 Conclusions

Far from being trivial as might be supposed on first acquaintance, indicator variables are fundamental and extremely useful. Without pretense at being encyclopedic, we have here tried to bring together the main possibilities—and some of the pitfalls—for the Stata user wanting to create and use indicator variables. We have also tried to discourage the dismal and disparaging term “dummy variables”.

## 13 References

- Baum, C. F. 2011. FAQ: How do you define group characteristics in your data in order to create subsets? <https://www.stata.com/support/faqs/data-management/group-characteristics-for-subsets/>.
- Blitzstein, J. K., and J. Hwang. 2015. *Introduction to Probability*. Boca Raton, FL: CRC Press.
- Cox, N. J. 2006. Stata tip 39: In a list or out? In a range or out? *Stata Journal* 6: 593–595.
- . 2009. Speaking Stata: Rowwise. *Stata Journal* 9: 137–157.
- . 2011. Speaking Stata: Fun and fluency with functions. *Stata Journal* 11: 460–471.
- . 2016a. FAQ: How do I create a variable recording whether any members of a group (or all members of a group) possess some characteristic? <https://www.stata.com/support/faqs/data-management/create-variable-recording/>.
- . 2016b. Speaking Stata: Truth, falsity, indication, and negation. *Stata Journal* 16: 229–236.
- . 2018. Speaking Stata: From rounding to binning. *Stata Journal* 18: 741–754.
- de Bruijn, N. G. 1958. *Asymptotic Methods in Analysis*. Amsterdam: North-Holland.
- de Finetti, B. 1967. Sur quelques conventions qui semblent utiles. *Revue Roumaine de Mathématiques Pures et Appliquées* 12: 1227–1233.
- . 1972. *Probability, Induction and Statistics: The Art of Guessing*. London: Wiley.
- . 1974. *Theory of Probability: A Critical Introductory Treatment*, vol. 1. London: Wiley.
- . 1975. *Theory of Probability: A Critical Introductory Treatment*, vol. 2. London: Wiley.
- Feller, W. 1971. *An Introduction to Probability Theory and Its Applications*, vol. 2. 2nd ed. New York: Wiley.
- Freedman, D. 2009. *Statistical Models: Theory and Practice*. Rev. ed. New York: Cambridge University Press.
- Iverson, K. E. 1962. *A Programming Language*. New York: Wiley.
- Kantor, D., and N. J. Cox. 2005. Depending on conditions: A tutorial on the `cond()` function. *Stata Journal* 5: 413–420.

- Klein, L. R. 1951. Estimating patterns of savings behavior from sample survey data. *Econometrica* 19: 438–454.
- Knuth, D. E. 1992. Two notes on notation. *American Mathematical Monthly* 99: 403–422.
- Lachenbruch, P. A. 1992. ip2: A keyboard shortcut. *Stata Technical Bulletin* 9: 9. Reprinted in *Stata Technical Bulletin Reprints*, vol. 2, p. 46. College Station, TX: Stata Press.
- Moran, P. A. P. 1968. *An Introduction to Probability Theory*. Oxford: Oxford University Press.
- Oakland, G. B. 1950. An application of sequential analysis to whitefish sampling. *Biometrics* 6: 59–67.
- Quenouille, M. H. 1950. Computational devices in the application of least squares. *Journal of the Royal Statistical Society, Series B* 12: 256–272.
- Whittle, P. 2000. *Probability via Expectation*. 4th ed. New York: Springer.
- Zeitz, P. 2007. *The Art and Craft of Problem Solving*. 2nd ed. Hoboken, NJ: Wiley.

#### About the authors

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).

Clyde Schechter is a professor of family and social medicine at the Albert Einstein College of Medicine, where he works as an epidemiologist and focuses on clinical and health services research projects involving many specialties and disciplines. He has used Stata since version 4, is an active participant in the Statalist forum, and is an occasional contributor to the *Stata Journal*.