One Hundred Nineteen Stata Tips

Third Edition

NICHOLAS J. COX, Editor
Durham University
Department of Geography

H. JOSEPH NEWTON, Editor
Texas A & M University
Department of Statistics

A Stata Press Publication
StataCorp LP
College Station, Texas

First edition 2006
Second edition 2009
Third edition 2014

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845
Typeset in LaTeX 2_ε
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN-10: 1-59718-143-9
ISBN-13: 978-1-59718-143-3

Library of Congress Control Number: 2014936591

Contents

(Pages omitted)

Editors' preface

The book you are reading reprints 119 Stata Tips from the *Stata Journal*, with thanks to their original authors. We, the *Journal* editors, began publishing tips in 2003, beginning with volume 3, number 4. It pleases us now to reprint them in this book.

The *Stata Journal* publishes substantive and peer-reviewed articles ranging from reports of original work to tutorials on statistical methods and models implemented in Stata, and indeed on Stata itself. Other features include regular columns such as "Speaking Stata", book reviews, and announcements.

We are pleased by the external recognition that the *Journal* has achieved. The *Stata Journal* is indexed and abstracted by CompuMath Citation Index, Current Contents/Social and Behavioral Sciences, RePEc: Research Papers in Economics, Science Citation Index Expanded (also known as SciSearch), Scopus, and Social Sciences Citation Index.

But back to the Tips. There was little need for tips in the early days. Stata 1.0 was released in 1985. The original program had 44 commands, and its documentation totaled 175 pages. Stata 13, on the other hand, has more than 1,000 commands—including an embedded matrix language called Mata—and Stata's official documentation now totals more than 11,000 pages. Beyond that, the user community has added several hundred more commands.

The pluses and the minuses of this growth are evident. As Stata expands, it is increasingly likely that users' needs can be met by available code. But at the same time, learning how to use Stata and even learning what is available become larger and larger tasks.

The Tips are intended to help. The ground rules for Stata Tips, as found in the original 2003 statement, are laid out as the next item in this book. We have violated one original rule in the letter, if not the spirit: Some Stata Tips have been as long as six pages. However, the intention of producing concise tips that are easy to pick up remains as it was.

The Tips grew from many discussions and postings on Statalist, at Users Group meetings, and elsewhere, which underscores a simple fact: Stata is now so big that it is easy to miss even simple features that can streamline and enhance your sessions with Stata. This applies not just to new users, who understandably may quake nervously before the manual mountain, but also to longtime users, who too are faced with a mass of new features in every release.

Tips have come from Stata users as well as from StataCorp employees. Many discuss new features of Stata, or features not documented fully or even at all. We hope that you enjoy the Stata Tips reprinted here and can share them with your fellow Stata users. If you have tips that you would like to write, or comments on the kinds of tips that are helpful, do get in touch with us, as we are eager to continue the series.

Nicholas J. Cox, Editor
H. Joseph Newton, Editor
April 2014

(Pages omitted)

The Stata Journal (2004)
4, Number 1, p. 93

Stata tip 4: Using display as an online calculator

Philip Ryan, University of Adelaide
philip.ryan@adelaide.edu.au

Do you use Stata for your data management, graphics, and statistical analysis but switch to a separate device for quick calculations? If so, you might consider the advantages of using Stata's built-in `display` command:

1. It is always at hand on your computer.
2. As with all Stata calculations, double precision is used.
3. You can specify the format of results.
4. It uses and reinforces your grasp of Stata's full set of built-in functions.
5. You can keep an audit trail of results and the operations that produced those results, as part of a log file. You can also add extra comments to the output.
6. Editing of complex expressions is easy, without having to re-enter lengthy expressions after a typo.
7. You can copy and paste results elsewhere whenever your platform supports that.
8. It is available via the menu interface (select **Data—Other utilities—Hand calculator**).
9. It can be abbreviated to `di`.

To be fair, there are some disadvantages, such as its lack of support for Reverse Polish Notation or complex number arithmetic, but in total, `display` provides you with a powerful but easy-to-use calculator.

```
. di _pi
3.1415927

. di %12.10f _pi
3.1415926536

. * probability of 2 heads in 6 tosses of a fair coin
. di comb(6,2) * 0.5^2 * 0.5^4
.234375

. di "chi-square (1 df) cutting off 5% in upper tail is " invchi2tail(1, .05)
chi-square (1 df) cutting off 5% in upper tail is 3.8414588

. * Euler-Mascheroni gamma
. di %12.10f -digamma(1)
0.5772156649
```

(Pages omitted)

The Stata Journal (2005)
5, Number 4, pp. 605–607

Stata tip 27: Classifying data points on scatter plots

Nicholas J. Cox
Durham University
n.j.cox@durham.ac.uk

When you have scatter plots of counted or measured variables, you may often wish to classify data points according to the values of a further categorical variable. There are several ways to do this. Here we focus on the use of `separate`, gray-scale gradation, and text characters as class symbols. If different categories really do plot as distinct clusters, it should not matter too much how you show them, but knowing some Stata tricks should also help.

One starting point is that differing markers may be used on the plot whenever there are several variables plotted on the y-axis. With the `auto.dta` dataset, you can imagine

```
. sysuse auto
. gen mpg0 = mpg if foreign == 0
. gen mpg1 = mpg if foreign == 1
. scatter mpg? weight
```

Note the use of the wildcard `mpg?`, which picks up any variable names that have `mpg` followed by just one other character. Once the two variables `mpg0` and `mpg1` have been generated, different markers are automatic. This process still raises two questions. To get an acceptable graph, we need self-explanatory variable labels or at least self-explanatory text in the graph legend. Moreover, two categories are easy enough, but do we have to do this for each of say 5, 7, or 9 categories?

In fact, it would have been better to type

```
. separate mpg, by(foreign) veryshortlabel
. scatter mpg? weight
```

The command `separate` (see [D] **separate**) generates all the variables we need in one command and has a stab at labeling them intelligibly. In this case, we use the (undocumented) `veryshortlabel` option, which was implemented with graphics especially in mind. You may prefer the results of the documented `shortlabel` option. Note that the `by()` option can take true-or-false conditions, such as `price < 6000`, as well as categorical variables.

If your categorical variable consists of qualitatively different categories, you are likely to want to use qualitatively different symbols. Alternatively, if that variable is ordered or graded, the coding you use should also be ordered. One possibility is to use symbols colored in a sequence of gray scales.

Some data on landforms illustrate the point: Ian S. Evans kindly supplied measurements of 260 cirques in Wales, armchair-shaped hollows formerly occupied by small glaciers. Length tends to increase with width, approximately as a power function, but qualitative aspects of form, particularly how closely they approach a classic, well-developed shape, are also coded in a grade variable.

```
. separate length, by(grade) veryshortlabel
. scatter length? width, xsc(log) ysc(log) ms(O ..)
> mcolor(gs1 gs4 gs7 gs10 gs13) mlcolor(black ..) msize(*1.5 ..)
> yti("`: variable label length´") yla(200 500 1000 2000, ang(h))
> xla(200 500 1000 2000) legend(pos(11) ring(0) col(1))
```

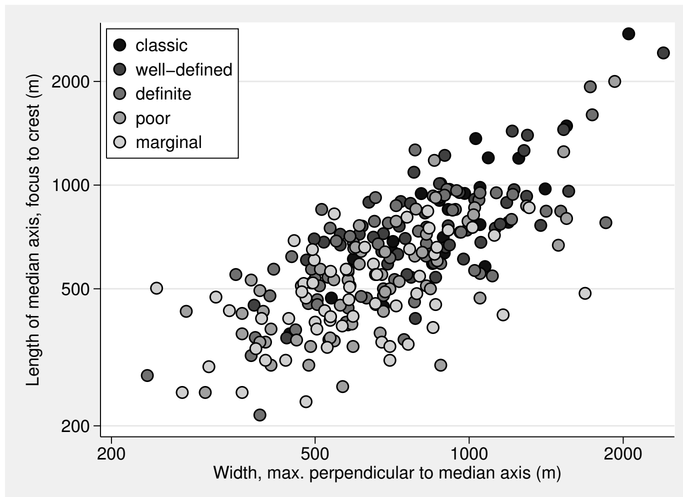
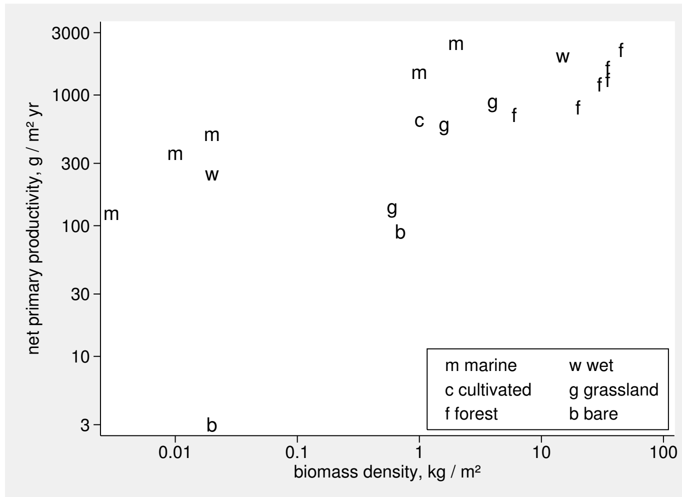


Figure 1 shows length versus width, subdivided by grade. Some practical details deserve emphasis. Gray scales near 16 (white) may be difficult to spot against a light background, including any printed page. Therefore, a dark outline color is recommended. Bigger symbols than the default are needed to do the coloring justice, but as a consequence, this approach is less likely to be useful with thousands of data points. A `by()` option showing different categories separately might work better. With the coding here, it so happens that the darkest category is plotted first and is thus liable to be overplotted by lighter categories wherever data points are dense. Some experimentation with the opposite order of plotting might be a good idea to see which works better.

An alternative that sometimes works nicely is to use ordinary text characters as different markers. One clean style is to suppress the marker symbols completely, using instead the contents of a `str1` variable as marker labels. Whittaker (1975, 224) gave data on net primary productivity and biomass density for various ecosystem types. Figure 2 shows the subdivision.

```
. scatter npp bd, xsc(log) ysc(log) ms(i) mlabpos(0) mlabsize(*1.4)
> mla(c) yla(3000 1000 300 100 30 10 3, nogrid ang(h))
> xla(0.01 "0.01" 0.1 "0.1" 1 10 100)
> legend(on ring(0) pos(5) order( - "m marine" - "w wet" - "c cultivated" -
> "g grassland" - "f forest" - "b bare"))
```



With three or four orders of magnitude variation in each variable, log scales are advisable. On those scales, there is a broad correlation whereby more biomass means higher productivity, but also considerable variation, much of which can be rationalized in terms of very different cover types. For the same biomass density, marine and other wet ecosystems have higher productivity than land ecosystems.

On the Stata side, remember `mlabpos(0)` and note that the `legend` must be set `on` explicitly. For different purposes, or for different tastes, what is here given as the legend might go better as text in a caption in a printed report. Behind the practice here lies general advice that lowercase letters, such as `abc`, work better than uppercase, such as `ABC`, as they are easier to distinguish from each other, and they are less likely to impart an synaesthetic sense in readers that the graph designer is shouting at them.

Reference

Whittaker, R. H. 1975. *Communities and Ecosystems*. New York: Macmillan.

(Pages omitted)

The Stata Journal (2007)
7, Number 2, pp. 268–271

Stata tip 45: Getting those data into shape[1]

Christopher F. Baum
Department of Economics
Boston College
Chestnut Hill, MA 02467
baum@bc.edu

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

Are your data in shape? That is, are they in the structure that you need to conduct the analysis you have in mind? Data sources often provide the data in a structure that is suitable for presentation but clumsy for statistical analysis. One of the key data management tools that Stata provides is `reshape`; see [D] **reshape**. If you need to modify the structure of your data, you should be familiar with `reshape` and its two functions: `reshape wide` and `reshape long`. In this tip, we discuss how two applications of `reshape` may be the solution to some knotty data management problems.

As a first example, consider this question posted on Statalist by an individual who has a dataset in the wide form:

`country`	`tradeflow`	`Yr1990`	`Yr1991`
Armenia	imports	105	120
Armenia	exports	90	100
Bolivia	imports	200	230
Bolivia	exports	80	115
Colombia	imports	100	105
Colombia	exports	70	71

He would like to reshape the data into long form:

`country`	`year`	`imports`	`exports`
Armenia	1990	105	90
Armenia	1991	120	100
Bolivia	1990	200	80
Bolivia	1991	230	115
Colombia	1990	100	70
Colombia	1991	105	71

1. This tip was updated to use the new command `import delimited` rather than `insheet`.—Ed.

We must exchange the roles of years and tradeflows in the original data to arrive at the desired structure, suitable for analysis as `xt` data. This exchange can be handled by two successive applications of `reshape`:

```
. reshape long Yr, i(country tradeflow)
(note: j = 1990 1991)
Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                        6   ->      12
Number of variables                   4   ->       4
j variable (2 values)                     ->   _j
xij variables:
                          Yr1990 Yr1991   ->   Yr
-----------------------------------------------------------------------------
```

This transformation swings the data into long form with each observation identified by `country`, `tradeflow`, and the new variable `_j`, taking on the values of year. We now perform `reshape wide` to make imports and exports into separate variables:

```
. rename _j year
. reshape wide Yr, i(country year) j(tradeflow) string
(note: j = exports imports)
Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                       12   ->       6
Number of variables                   4   ->       4
j variable (2 values)         tradeflow   ->   (dropped)
xij variables:
                                     Yr   ->   Yrexports Yrimports
-----------------------------------------------------------------------------
```

If we transform the data to wide form once again, the `i()` option contains `country` and `year`, as those are the desired identifiers on each observation of the target dataset. We specify that `tradeflow` is the `j()` variable for `reshape`, indicating that it is a `string` variable. The data now have the desired structure. Although we have illustrated this double-reshape transformation with only a few countries, years, and variables, the technique generalizes to any number of each.

As a second example of successive applications of `reshape`, consider the World Bank's World Development Indicators (WDI) dataset.[2] Their extract program generates a comma-separated value (CSV) database extract, readable by Excel or Stata, but the structure of those data hinders analysis as panel data. For a recent year, the header line of the CSV file is

```
"Series code","Country Code","Country Name","1960","1961","1962","1963",
"1964","1965","1966","1967","1968","1969","1970","1971","1972","1973",
"1974","1975","1976","1977","1978","1979","1980","1981","1982","1983",
"1984","1985","1986","1987","1988","1989","1990","1991","1992","1993",
"1994","1995","1996","1997","1998","1999","2000","2001","2002","2003","2004"
```

2. See http://econ.worldbank.org.

That is, each row of the CSV file contains a *variable* and *country* combination, with the columns representing the elements of the time series.[3]

Our target dataset structure is that appropriate for panel-data modeling, with the variables as columns and rows labeled by country and year. Two applications of `reshape` will again be needed to reach the target format. We first `import delimited` (see [D] **import delimited**) the data and transform the triliteral country code into a numeric code with the country codes as labels:

```
. import delimited using wdiex
. encode countrycode, generate(cc)
. drop countrycode
```

We then must address that the time-series variables are named `var4-var48`, as the header line provided invalid Stata variable names (numeric values) for those columns. We use `rename` (see [D] **rename**) to change `v4` to `d1960`, `v5` to `d1961`, and so on:

```
forvalues i=4/48 {
        rename v`i´ d`=1956+`i´´
}
```

We now are ready to carry out the first `reshape`. We want to identify the rows of the reshaped dataset by both country code (`cc`) and `seriescode`, the variable name. The `reshape long` will transform a fragment of the WDI dataset containing two series and four countries:

```
. reshape long d, i(cc seriescode) j(year)
(note: j = 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972
> 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987
> 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002
> 2003 2004)
Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                        7   ->     315
Number of variables                  48   ->       5
j variable (45 values)                    ->   year
xij variables:
                  d1960 d1961 ... d2004   ->   d
-----------------------------------------------------------------------------
```

3. A variation occasionally encountered will resemble this structure, but with periods in reverse chronological order. The solution here can be used to deal with that problem as well.

```
. list in 1/15

     +-------------------------------------------------------+
     |  cc   seriesc~e   year   countryname                d |
     |-------------------------------------------------------|
  1. | AFG   adjnetsav   1960   Afghanistan                . |
  2. | AFG   adjnetsav   1961   Afghanistan                . |
  3. | AFG   adjnetsav   1962   Afghanistan                . |
  4. | AFG   adjnetsav   1963   Afghanistan                . |
  5. | AFG   adjnetsav   1964   Afghanistan                . |
     |-------------------------------------------------------|
  6. | AFG   adjnetsav   1965   Afghanistan                . |
  7. | AFG   adjnetsav   1966   Afghanistan                . |
  8. | AFG   adjnetsav   1967   Afghanistan                . |
  9. | AFG   adjnetsav   1968   Afghanistan                . |
 10. | AFG   adjnetsav   1969   Afghanistan                . |
     |-------------------------------------------------------|
 11. | AFG   adjnetsav   1970   Afghanistan        -2.97129 |
 12. | AFG   adjnetsav   1971   Afghanistan        -5.54518 |
 13. | AFG   adjnetsav   1972   Afghanistan        -2.40726 |
 14. | AFG   adjnetsav   1973   Afghanistan        -.188281 |
 15. | AFG   adjnetsav   1974   Afghanistan         1.39753 |
     +-------------------------------------------------------+
```

The rows of the data are now labeled by year, but one problem remains: all variables for a given country are stacked vertically. To unstack the variables and put them in shape for `xtreg` (see [XT] **`xtreg`**), we must carry out a second `reshape` that spreads the variables across the columns, specifying `cc` and `year` as the i variables and `seriescode` as the j variable. Since that variable has string content, we use the `string` option.

```
. reshape wide d, i(cc year) j(seriescode) string
(note: j = adjnetsav adjsavC02)
Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                      315   ->     180
Number of variables                   5   ->       5
j variable (2 values)        seriescode   ->   (dropped)
xij variables:
                                      d   ->   dadjnetsav dadjsavC02
-----------------------------------------------------------------------------

. order cc countryname

. tsset cc year
       panel variable:  cc (strongly balanced)
        time variable:  year, 1960 to 2004
```

After this transformation, the data are now in shape for `xt` modeling, tabulation, or graphics.

As illustrated here, the `reshape` command can transform even the most inconvenient data structure into the structure needed for your research. It may take more than one application of `reshape` to get there from here, but it can do the job.