

THE STATA JOURNAL

data, citation and similar papers at core.ac.uk

brought to you by

provided by Research Papers in

Department of Statistics
Texas A & M University
College Station, Texas 77843
979-845-3142; FAX 979-845-3144
jnewton@stata-journal.com

Department of Geography
University of Durham
South Road
Durham City DH1 3LE UK
n.j.cox@stata-journal.com

Associate Editors

Christopher Baum
Boston College

Rino Bellocco
Karolinska Institutet

David Clayton
Cambridge Inst. for Medical Research

Mario A. Cleves
Univ. of Arkansas for Medical Sciences

William D. Dupont
Vanderbilt University

Charles Franklin
University of Wisconsin, Madison

Joanne M. Garrett
University of North Carolina

Allan Gregory
Queen's University

James Hardin
University of South Carolina

Stephen Jenkins
University of Essex

Ulrich Kohler
WZB, Berlin

Jens Lauritsen
Odense University Hospital

Stanley Lemeshow
Ohio State University

J. Scott Long
Indiana University

Thomas Lumley
University of Washington, Seattle

Roger Newson
King's College, London

Marcello Pagano
Harvard School of Public Health

Sophia Rabe-Hesketh
University of California, Berkeley

J. Patrick Royston
MRC Clinical Trials Unit, London

Philip Ryan
University of Adelaide

Mark E. Schaffer
Heriot-Watt University, Edinburgh

Jeroen Weesie
Utrecht University

Nicholas J. G. Winter
Cornell University

Jeffrey Wooldridge
Michigan State University

Stata Press Production Manager

Lisa Gilmore

Copyright Statement: The Stata Journal and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp LP. The contents of the supporting files (programs, datasets, and help files) may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the Stata Journal.

The articles appearing in the Stata Journal may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the Stata Journal.

Written permission must be obtained from StataCorp if you wish to make electronic copies of the insertions. This precludes placing electronic copies of the Stata Journal, in whole or in part, on publicly accessible web sites, file servers, or other locations where the copy may be accessed by anyone other than the subscriber.

Users of any of the software, ideas, data, or other materials published in the Stata Journal or the supporting files understand that such use is made without warranty of any kind, by either the Stata Journal, the author, or StataCorp. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the Stata Journal is to promote free communication among Stata users.

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press, and Stata is a registered trademark of StataCorp LP.

Stata tip 13: generate and replace use the current sort order

Roger Newson
King's College London, UK
roger.newson@kcl.ac.uk

Did you know that `generate` and `replace` use the current sort order? You might have guessed this because otherwise the `sum()` function could work as designed only with difficulty. However, this fact is not documented in the manuals, but only in the Stata web site FAQs. The consequence is that, given a particular desired `sort` order, you can be sure that values of a variable are calculated in that order and can use them to calculate subsequent values of the same variable.

A simple example is filling in missing values by copying the previous nonmissing value. The syntax for this is simply

```
. replace myvar = myvar[_n-1] if missing(myvar)
```

Here the subscript `[_n-1]`, based on the built-in variable `_n`, refers to the previous observation in the present sort order. To find more about subscripts, see [U] **16.7 Explicit subscripting** or the online help for **subscripting**.

Suppose that values of `myvar` are present for observations 1, 2, and 5 but missing in observations 3, 4, and 6. `replace` starts by replacing `myvar[3]` with the nonmissing `myvar[2]`. It then replaces `myvar[4]` with `myvar[3]`, which now contains (just in time) a copy of the nonmissing `myvar[2]`. Finally, `replace` puts a copy of `myvar[5]` into `myvar[6]`. As said, this all requires that data are in the desired sort order, commonly that of some time variable. If not, reach for the `sort` command.

There are numerous variations on this idea. Suppose that a sequence of years contains nonmissing values only for years like 1980, 1990, and 2000. This is common in data derived from spreadsheet files. A simple fix would be

```
. replace year = year[_n-1] + 1 if mi(year)
```

That way, changes cascade down the observations.

More exotic examples concern recurrence relations, as found in probability theory and elsewhere in mathematics. We typically use `generate` to define the first value (or the first few values) and `replace` to define the other values.

Consider the famous “birthday problem”: what is the probability that no two out of n people have the same birthday? Assuming equal probabilities of birth on each of 365 days, and so ignoring leap years and seasonal fertility variation, this probability is $\prod_{j=1}^n x_j$, where $x_j = (365 - j + 1)/365$. We can put these probabilities into a variable `palldiff` by typing

```

. set obs 370
. generate double palldiff = 1
. replace palldiff = palldiff[_n-1] * (365 - _n + 1) / 365 in 2/1
. label var palldiff "Pr(All birthdays are different)"
. list palldiff

```

To illustrate, the probability that all birthdays are different is below 0.5 for 23 people, below one-millionth for 97 people, and zero for over 365 people. An alternative solution (based on a suggestion by Roberto Gutierrez) is to replace the second and third lines of the above program with

```

. generate double palldiff = 0
. replace palldiff = exp(sum(ln(366 - _n) - ln(365))) in 1/365

```

which works because the product of positive numbers is the sum of their logarithms, exponentiated.

Another example is the Fibonacci sequence, defined by $y_1 = y_2 = 1$ and otherwise by $y_n = y_{n-1} + y_{n-2}$. The first 20 numbers are given by

```

. set obs 20
. generate y = 1
. replace y = y[_n-1] + y[_n-2] in 3/1
. list y

```

If you ever want to work backwards by referring to later observations, it is often easiest to reverse the order of observations and then to use tricks like these.