The Stata Journal (2001) **1**, Number 1, pp. 1--2

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 website 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 non-missing 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 on-line 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]` by the non-missing `myvar[2]`. It then replaces `myvar[4]` by `myvar[3]`, which now contains (just in time) a copy of the non-missing `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 non-missing 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 met in probability theory and elsewhere in mathematics. We typically use `generate` to define the first value (or the first few values), and then `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, then 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/l
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 (suggested by R. G. 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 a 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/l
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.