The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go “beyond the Stata manual” in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

For more information on the *Stata Journal*, including information for authors, see the web page

http://www.stata-journal.com

The *Stata Journal* is indexed and abstracted in the following:

- CompuMath Citation Index®
- RePEc: Research Papers in Economics
- Science Citation Index Expanded (also known as SciSearch®)

**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*.

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, fileservers, 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. Stata and Mata are registered trademarks of StataCorp LP.
**Stata tip 80: Constructing a group variable with specified group sizes**

Martin Weiss  
Department of Economics  
Tübingen University  
Tübingen, Germany  
martin.weiss@uni-tuebingen.de

Quite often, Stata users wish to construct a variable denoting group membership with different group sizes. This could be part of a simulation study in which a discrete variable with a certain distribution is required. Two cases can be distinguished: one in which the desired group sizes should be hit exactly and one in which group size should vary randomly around the desired proportion. In either case, group membership is to be determined randomly. This problem differs from grouping on the basis of one or more existing categories, which usually is best accomplished through `egen, group()` (see [D] `egen` and Cox [2007]).

The first goal can be achieved by obtaining uniformly distributed random numbers with Stata’s `runiform()` function (see Buis [2007] for an expanded discussion of the usefulness of uniform random numbers) and sorting on these numbers. Because this uniform random variable is of no interest as such, we construct it as a temporary variable. The groups are subsequently determined in one fell swoop by conditioning on the position of an observation after the random sorting. The way to achieve this is to use an expression, as described in [U] 13 Functions and expressions, featuring the system variable `_n` (see [U] 13.4 System variables (_variables)), which denotes the running number of the observation. Say that you want groups with proportions 50%, 40%, and 10% in a sample of 10,000: whenever `_n` is smaller than or equal to 5,000, the expression `inrange(_n,1,5000)` evaluates to 1 and the other two evaluate to zero. Thus the entire sum evaluates to 1.

```
. clear
. set obs 10000
   obs was 0, now 10000
. set seed 12345
. tempvar aux
. generate byte `aux'=runiform()
. sort `aux'
. generate byte group = inrange(_n,1,5000)*1 +
   > inrange(_n,5001,9000)*2 +
   > inrange(_n,9001,10000)*3
. tabulate group
```

<table>
<thead>
<tr>
<th>group</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5,000</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>2</td>
<td>4,000</td>
<td>40.00</td>
<td>90.00</td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>10.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Total</td>
<td>10,000</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

© 2009 StataCorp LP  
st0181
The `set seed` command is used to make the draws from `runiform()`, and thus the division into groups, reproducible (for further information, see [R] `set seed`). This practice is also maintained in the examples below.

A feasible way to achieve the second goal is the use of nested `cond()` functions, as introduced in Kantor and Cox (2005):

```stata
. clear
. set obs 10000
  obs was 0, now 10000
. set seed 12345
. generate byte group = cond(runiform()<0.5, 1, cond(runiform()<0.8, 2, 3))
. tabulate group

<table>
<thead>
<tr>
<th>group</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,947</td>
<td>49.47%</td>
<td>49.47%</td>
</tr>
<tr>
<td>2</td>
<td>4,048</td>
<td>40.48%</td>
<td>89.95%</td>
</tr>
<tr>
<td>3</td>
<td>1,005</td>
<td>10.05%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
```

The outer `cond()` function takes care of the 50% group, and the inner `cond()` function splits the remaining 50% in the ratio 4 to 1 so that the end result coincides with the user’s intention. The two—separate—draws from `runiform()` are not related in any way.

This process can be shortened further with the help of the lesser-known `irecode()` function (see [D] `functions` for the full array of functions).

```stata
. clear
. set obs 10000
  obs was 0, now 10000
. set seed 12345
. generate byte group = irecode(runiform(), 0, 0.5, 0.9, 1)
. tabulate group

<table>
<thead>
<tr>
<th>group</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,957</td>
<td>49.57%</td>
<td>49.57%</td>
</tr>
<tr>
<td>2</td>
<td>4,049</td>
<td>40.49%</td>
<td>90.06%</td>
</tr>
<tr>
<td>3</td>
<td>994</td>
<td>9.94%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
```

The `irecode()` function takes \( n + 1 \) arguments. The first argument is evaluated against the remaining \( n \) arguments. If it is smaller than the second, “0” is returned; if it is larger than the second but smaller than the third, “1” is returned; and so on. Because the draws from `runiform()` must lie between 0 and 1, any draw will be larger than the second argument, and thus “0” will never be returned. Approximately 50% of the time, the draw will lie between the second and third argument, and the results will be “1”, and so on. The desired proportions for the groups must be translated into differences between the arguments of the `irecode()` function: to get the 40% group, for instance, the third and fourth argument must be \( 0.9 - 0.5 = 0.4 \) units apart.
As a final thought, consider the slight difference between the group sizes in the second and third example—although we did set seed to “12345” in both cases. As mentioned above, in the second example, two draws from the uniform distribution are conducted, one for each cond() function. The irecode() function in the third example, however, is fed one draw from runiform(). The function compares this one draw with the cutpoints provided by the user and assigns group membership accordingly.

References

