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](http://www.stata-journal.com)

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

- CompuMath Citation Index®
- Current Contents/Social and Behavioral Sciences®
- RePEc: Research Papers in Economics
- Science Citation Index Expanded (also known as SciSearch®)
- Social Sciences Citation Index®

**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* (ISSN 1536-867X) is a publication of Stata Press. Stata, Mata, NetCourse, and Stata Press are registered trademarks of StataCorp LP.
Stata tip 86: The missing() function

Bill Rising
StataCorp
College Station, TX
brising@stata.com

Stata’s treatment of missing numeric values in expressions is clear: numeric values behave like infinity. However, some care is needed whenever numeric missing values could appear in a relational expression. Typically, missing values are included or excluded explicitly by a segment of Stata code. But there is a better way to deal with missing values: the missing() function.

Suppose that we have a dataset containing the body mass index (BMI) and age for a sample of people. As happens in real-world datasets, some observations contain missing values for one or both of the variables. We would like to create an indicator variable that marks all the obese adults, meaning those who are at least 20 years old and have a BMI of at least 30. On the surface, the command we should type is

```
. generate obese_adult = age>=20 & bmi>=30
```

This will not yield the desired results, however, because the relational operators do not treat missing values as anything special—they are just considered to be very large numbers and so are still included. Hence, the values for `obese_adult` will be filled according to the following table:

<table>
<thead>
<tr>
<th>age&lt;20</th>
<th>age&gt;=20, nonmissing</th>
<th>age missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmi&lt;30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bmi&gt;=30, nonmissing</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>bmi missing</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The solution to this problem is to assign the value only for those observations that have no missing values. Because missing values are large, we could type

```
. generate obese_adult = age>=20 & bmi>=30 if !missing(age, bmi)
```

Although correct, this is not the best solution. In particular, it is only readable to Stata users—other people would have no clue what `age<.` means.

A better way to work with missing values is to use the missing() function. Its syntax and behavior are simple: missing(exp1[, exp2, ..., expN]) evaluates to 1 if any of the expressions is missing and 0 if none is missing. Thus we could rewrite our previous correct `generate` command as

```
. generate obese_adult = age>=20 & bmi>=30 if !missing(age, bmi)
```

This is much more readable.

Another advantage of using the missing() function is that it treats string and numeric variables in the same fashion. Now suppose that we have a dataset containing
Stata tip 86: The missing() function

A string-valued identifier named `ssn` and a numeric identifier named `whodat`. If we want to mark all the observations for which at least one identifier is missing, we can do this quite easily:

```
. generate byte badid = missing(ssn, whodat)
```

Not only is this readable but it also does not require knowing the data type of the two variables.

One thing that can trip up users new to `missing()` is its arguments: they are a comma-separated list of expressions, not a varlist. If you are interested in using varlists, you can write your own ado-file:

```
*! version 1.0.0 February 18, 2010 @ 08:48:04
*! makes a comma-separated varlist
program define vcomma, rclass
version 11
syntax [varlist]
local varlist : subinstr local varlist " " ",", all
return local varlist "'varlist'"
end
```

You can then give `vcomma` a varlist, and it will return the comma-separated list in `r(varlist)`. So, for example, if you were interested in finding any observation in a dataset that had missing values for any variable, you could do this with the following two lines:

```
vcomma
list if missing(r(varlist))
```

This is certainly much easier than using loops or a long `if` qualifier.

Working with missing values is necessary in most datasets. As you have learned in this tip, using the `missing()` function makes working with missing values less onerous.