The Stata Journal (2010)
10, Number 2, pp. 303–304

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:

	`age<20`	`age>=20`, nonmissing	`age` missing
`bmi<30`	0	0	0
`bmi>=30`, nonmissing	0	1	1
`bmi` missing	0	1	1

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 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

© 2010 StataCorp LP

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.