The Stata Journal

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.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*), *Scopus*, and *Social Sciences Citation Index*.

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

http://www.stata-journal.com

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
Printed & electronic		**Printed & electronic**	
1-year subscription	$115	1-year subscription	$145
2-year subscription	$210	2-year subscription	$270
3-year subscription	$285	3-year subscription	$375
1-year student subscription	$ 85	1-year student subscription	$115
1-year institutional subscription	$345	1-year institutional subscription	$375
2-year institutional subscription	$625	2-year institutional subscription	$685
3-year institutional subscription	$875	3-year institutional subscription	$965
Electronic only		**Electronic only**	
1-year subscription	$ 85	1-year subscription	$ 85
2-year subscription	$155	2-year subscription	$155
3-year subscription	$215	3-year subscription	$215
1-year student subscription	$ 55	1-year student subscription	$ 55

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

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

The Stata Journal (2015)
15, Number 4, pp. 1174–1185

Speaking Stata: A set of utilities for managing missing values

Nicholas J. Cox
Department of Geography
Durham University
Durham, UK
n.j.cox@durham.ac.uk

Abstract. Identifying frequencies and patterns of missing values and preliminary cleaning of missing data are common and fundamental parts of statistical data management. I offer a new command, `missings`, as a replacement for, and extension of, my previous commands `nmissing` (Cox 1999, *Stata Technical Bulletin* 49: 7–8; 2001a, *Stata Technical Bulletin* 60: 2–3; 2003, *Stata Journal* 3: 449; 2005, *Stata Journal* 5: 607) and `dropmiss` (Cox 2001b, *Stata Technical Bulletin* 60: 7–8; 2008, *Stata Journal* 8: 594). I also make comparisons with similar commands.

Keywords: dm0085, missing, missing values, system missing, extended missing values, data management, numeric variables, string variables, drop, list, table, tag

1 Introduction

A crucial hallmark of statistical software is support for missing values. Extensive real datasets, even those of high quality, can be speckled and spattered with missing data, for reasons both good and bad. People do not answer all the questions put to them, instruments can break down or otherwise fail to record, the observer or experimenter may be unable to take some measurements: reasons for being missing are legion; the fact of being missing is pervasive. The ability to hold those missing values within the dataset is essential, even if that means ignoring missings in statistical or graphical analysis (except perhaps for showing their frequencies or patterns of occurrence).

Good support means special codes for missing values. The alternative of adopting ad hoc conventions (missing ages are recorded as −99, or whatever) is too awkward and error-prone to be taken seriously. Eternal vigilance is the price of such ad hockery: the researcher would have to remember always to work around such conventions, which may differ from problem to problem and may not even be documented explicitly. The scope for data to be separated from their metadata is unlimited, especially if the metadata reside in the understanding of the few researchers who were in charge of data production at some past time and some different place.

Hence, the better solution of special codes for missing values has long been used by writers of statistical software. Generically, the first principle is that missing values must always be distinguishable from nonmissing values. A second principle is that it can help greatly to be able to separate different reasons for missing values. Specifically, in Stata, we use the system missing value `.` and extended missing values `.a` to `.z` for numeric values and the empty string value `""` for string values.

Caution with missing values often seems to be considered too obvious or too mundane to deserve discussion in texts. Among the splendid exceptions are Altman (1991), with clear warnings of various pitfalls, and Long (2009), with much good detail linked to advice on Stata.

Several commands in Stata can provide help in dealing with missing values. These include

- `codebook` ([D] **codebook**), which counts missing values when present and optionally can provide more detail;
- `egen` ([D] **egen**), which can count missing values, and indeed nonmissing values too—quite likely more to the point;
- `ipolate` ([D] **ipolate**), used for interpolation given missing values within sequences;
- `misstable` ([R] **misstable**) (added in Stata 11), used for reporting on missing values;
- `mvdecode` ([D] **mvencode**) and `recode` ([D] **recode**), mentioned here mainly for their use in recoding nonmissing values to missing, particularly to handle imported data, where (say) −99 might mean missing; and
- `tabulate oneway` ([R] **tabulate oneway**) and `tabulate twoway` ([R] **tabulate twoway**) and their siblings `tab1` and `tab2`, which have useful `missing` options that insist on missings being counted too.

The most substantial aid in dealing with missing values is the `mi` ([MI] **intro**) suite for multiple imputation added in Stata 11 and enhanced in later releases.

Among user-written commands that may also help with missing values, I capriciously single out `findname` (Cox 2010a,b, 2012, 2015), whose abilities include finding variables with all, some, or no missing values, and `mipolate`, an extension of `ipolate` downloadable at the time of writing from the Statistical Software Components archive.

Here I focus on `missings`, a set of utilities for managing missing values, which I offer as a replacement for, and extension of, my previous commands `nmissing` (Cox 1999, 2001a, 2003, 2005) and `dropmiss` (Cox 2001b, 2008). I have tinkered with these commands over several years. Reflection on recent developments in Stata and on continued experiences in data management has led to a new integrated command with several subcommands. `missings` includes new functionality but excludes some details that now seem of limited value. (Anyone seeking any of those details will find that the previous commands remain accessible and downloadable.) Although no connection is needed or implied, the design of `missings` was influenced by the `duplicates` ([D] **duplicates**) command, which offers a modest but serviceable bundle of utilities for dealing with duplicate observations.

I will outline the functionality of `missings` with examples and close with a formal syntax statement.

2 Command goals

`missings` combines various basic utilities for managing variables that may have missing values. The syntax starts

`missings` *subcommand*

with a choice of six subcommands. By default, "missing" means missing in all senses, as defined in the previous section. Later, we will learn how to change this default.

If a *varlist* (variable list) is not further specified, then it is interpreted by default as all variables.

`missings report` issues a report on the number of missing values in *varlist*. By default, counts of missings are given by variables; optionally, counts are given by observations.

`missings list` lists observations with missing values in *varlist*.

`missings table` tabulates observations by the number of missing values in *varlist*.

`missings tag` generates a variable containing the number of missing values in each observation in *varlist*.

`missings dropvars` drops any variables in *varlist* that are missing on all values.

`missings dropobs` drops any observations that are missing on all values in *varlist*.

Let us consider how missings across entire observations or variables might arise. Creating entirely empty observations (rows) and variables (columns) is a habit of many spreadsheet users. The main reason is just to separate blocks of data to make them easier to understand and work with. Such blank rows and columns may well be copied unchanged even when the spreadsheet contents are exported to quite different file formats. However, neither kind of blank is helpful in Stata datasets. At worst, such variables or observations consume precious memory. So the subcommands `dropobs` and `dropvars` should help users clean up.

Note that there is no explicit support in `missings` for dropping observations or variables with some missing but also some nonmissing values. Researchers are often tempted to reduce datasets to a minimum that is completely nonmissing on both variables and observations. That strategy can be severely reductive. As a program author, I want users to take on complete responsibility for any damage they might inflict on datasets. Users intent on destruction will find other subcommands of `missings` of some use as an intermediate step. Modern statistical thinking encourages consideration of multiple imputation as a better way forward. Any kind of statistical thinking encourages use of all the information you have, despite its deficiencies.

Users may have their own ideas of what is missing that go beyond Stata's formal definitions. For example, string values that are just one or more spaces are unlikely to be informative. However, because they are not empty, they do not qualify as missing. Should you wish to treat them as missing, the simplest way forward is to note that ap-

plying `trim()` to one or more spaces produces an empty string. `findname`, as mentioned in the first section, can find all variables with any such trimmable values by means of

```
. findname, any(trim(@) == "" & @ != "")
```

The simpler criterion

```
. findname, any(trim(@) == "")
```

finds both empty strings and values that are just spaces. `findname` also has an `all()` option to find variables with all values satisfying some criterion. Once such variables are identified, you should use `trim()` to remove the spaces. `findname` does not do the trimming itself; it just reports the results. `missings` quite deliberately does not stretch the definition of missing values beyond Stata's own definitions, so some cleanup may be needed before it is invoked.

Another similar example would be the use of `"."` to indicate string missing values. Although Stata does give special meaning to system missing `.` for numeric values, a string consisting of just a period (stop, dot) has no special meaning. That or any other personal convention for missing values would need special action before `missings` could be used.

I reiterate that `mvdecode` ([D] **mvencode**) is custom designed to replace numeric codes indicating missing with Stata's own missing values.

3 Examples of missings

After that overview, let us look at a few examples of `missings`. We can read in a fairly complicated dataset as a sandbox in which to play and first get a concise report on missings. We can specify a threshold to see which variables look particularly poor.

```
. webuse nlswork
(National Longitudinal Survey.  Young Women 14-26 years of age in 1968)
. missings report
Checking missings in all variables:
15082 observations with missing values
age           24
msp           16
nev_mar       16
grade          2
not_smsa       8
c_city         8
south          8
ind_code     341
occ_code     121
union       9296
wks_ue      5704
tenure       433
hours         67
wks_work     703
```

```
. missings report, minimum(1000)
Checking missings in all variables:
15082 observations with missing values

union      9296
wks_ue     5704
```

Although not shown here, `missings report` also supports the display of percentages as well as counts.

We can hone in on the observations with missing values by asking for a list. Again, we can specify a threshold:

```
. missings list, minimum(5)
Checking missings in all variables:
15082 observations with missing values
```

6924.

age	msp	nev_mar	not_smsa	c_city	south	ind_code	occ_code
26	.	.	0	1	0	11	8

union	wks_ue	tenure	hours	wks_work
.	.	.0833333	40	.

19002.

age	msp	nev_mar	not_smsa	c_city	south	ind_code	occ_code
31	.	.	0	1	1	7	8

union	wks_ue	tenure	hours	wks_work
.	.	.3333333	.	.

21220.

age	msp	nev_mar	not_smsa	c_city	south	ind_code	occ_code
26	1	0	.	.	.	7	3

union	wks_ue	tenure	hours	wks_work
.	.	.0833333	40	.

22493.

age	msp	nev_mar	not_smsa	c_city	south	ind_code	occ_code
32	1	0	1	0	0	.	.

union	wks_ue	tenure	hours	wks_work
.	.	.75	.	.

22628.

age	msp	nev_mar	not_smsa	c_city	south	ind_code	occ_code
39	0	0	0	1	0	.	3

union	wks_ue	tenure	hours	wks_work
.	.	1.5	.	.

For the sake of a short example, we just set the threshold high. In this example, only 5 observations have 5 or more missing values. Naturally, the threshold you need will depend on your dataset and goals. Similarly, the produced listing might deserve long and careful scrutiny.

Another basic command, `missings table`, reports counts of missings across all variables specified or, alternatively, all variables:

```
. missings table
Checking missings in all variables:
15082 observations with missing values
```

# of missing values	Freq.	Percent	Cum.
0	13,452	47.14	47.14
1	13,790	48.33	95.47
2	964	3.38	98.85
3	291	1.02	99.87
4	32	0.11	99.98
5	2	0.01	99.99
6	3	0.01	100.00
Total	28,534	100.00	

Where appropriate, you can specify a `by:` prefix to get a breakdown by other variables:

```
. bysort race: missings table
-------------------------------------------------------------------------------
-> race = white
Checking missings in all variables:
10576 observations with missing values
```

# of missing values	Freq.	Percent	Cum.
0	9,604	47.59	47.59
1	9,672	47.93	95.52
2	677	3.35	98.88
3	199	0.99	99.86
4	25	0.12	99.99
5	1	0.00	99.99
6	2	0.01	100.00
Total	20,180	100.00	

```
-> race = black
Checking missings in all variables:
4342 observations with missing values
```

# of missing values	Freq.	Percent	Cum.
0	3,709	46.07	46.07
1	3,966	49.26	95.33
2	278	3.45	98.78
3	89	1.11	99.89
4	7	0.09	99.98
5	1	0.01	99.99
6	1	0.01	100.00
Total	8,051	100.00	

```
-> race = other
Checking missings in all variables:
164 observations with missing values
```

# of missing values	Freq.	Percent	Cum.
0	139	45.87	45.87
1	152	50.17	96.04
2	9	2.97	99.01
3	3	0.99	100.00
Total	303	100.00	

It is sometimes helpful to have a count of `missings` in a variable. As mentioned in section 1, there is already an `egen` function for this, but a little duplication does no harm.

```
. missings tag, generate(nmissing)
Checking missings in all variables:
15082 observations with missing values
```

The variable generated by `missing tag`, like that generated by the `egen` function `rowmiss()`, can be zero only when there are no missing values in an observation for the *varlist* specified or implied or positive when there are some such missing values. In Stata, zero is treated as false and positive as true when arguments are offered in true-or-false decisions. (The frequently asked question http://www.stata.com/support/faqs/data-management/true-and-false/ expands on this point.) Hence, something such as

```
... if tag
```

would select observations with any missing values, and its negation

```
... if !tag
```

would select observations with no missing values. Alternatively, you could write

```
... if tag > 0
```

or

```
... if tag == 0
```

if you prefer. Note that you can map such a variable to a $(0, 1)$ indicator if you desire by writing

```
. generate anymissing = nmissing > 0
```

This dataset does not have any variables or observations that are entirely missing, but let's see how `missings` could be used to remove any. We will extend the sandbox in which we play by adding some nonsense variables.

```
. generate newt = ""
(28,534 missing values generated)
. generate frog = .
(28,534 missing values generated)
. generate toad = .a
(28,534 missing values generated)
```

These new variables are simple examples showing different kinds of missing, as containing, respectively, empty strings only, system missing only, and one of the extended missing values only.

Suppose we assume that extended missing values are informative and should be included but that empty strings and system missing values are not informative and may be dropped (so long as no nonmissing values are observed in the same variables). For this task, we need the `sysmiss` option, which is ignored with regard to empty strings. (Equivalently, we are stretching Stata's usual definition of "system missing" just a little to include empty strings. The alternative would be introducing some new terminology that could seem arbitrary or mysterious, such as "really missing" or "fully missing".)

```
. missings dropvars newt frog toad, force sysmiss
Checking missings in newt frog toad:
28534 observations with system missing values
note: newt frog dropped
```

In our example, `newt` and `frog` are entirely uninformative and can be dropped. However, note the extra option `force`. When the dataset in memory has changed from its last saved version, you need the `force` option to drop any variables or observations, even though they are entirely missing. This may seem excessively cautious, but prudence requires that any destructive change to datasets be confirmed.

`missings` would not drop `toad`, containing `.a` only, because it is protected by the `sysmiss` option. That will be true if we try again:

```
. missings dropvars toad, force sysmiss
Checking missings in toad:
0 observations with system missing values
note: no variables qualify
```

If we remove that protection, `toad` can be removed from the dataset:

```
. missings dropvars toad, force
Checking missings in toad:
28534 observations with missing values
note: toad dropped
```

Let's continue with our sandbox and simulate observations that are all missing. If we enlarge the dataset by setting the number of observations to 30,000, then 1,466 extra observations spring into being, all born missing on each variable.

```
. set obs 30000
number of observations (_N) was 28,534, now 30,000
. missings dropobs, force
Checking missings in idcode year birth_yr age race msp nev_mar grade collgrad
    not_smsa c_city south ind_code occ_code union wks_ue ttl_exp tenure hours
    wks_work ln_wage nmissing anymissing:
16548 observations with missing values
(1,466 observations deleted)
```

As before, various safety features are engaged here. Most obviously, if the dataset in memory has not been saved, then you must recognize that you are exerting brute force. In addition, extended missing values `.a` to `.z` are supposed to be informative, so you can specify `sysmiss` to spell out that only system missings may be dropped. In this case, `expand` never creates extended missing values, so the problem does not arise.

It is fairly rare, at least in my experience, to have so many observations that are entirely missing. But the principle applies to small cleanups as well as large—and indeed to dropping missing observations regardless of whether they were deliberate at some point or merely inserted by accident.

4 Command syntax

4.1 Syntax diagram

`missings report` [*varlist*] [*if*] [*in*] [`,` *common_options* `observations` `minimum(`#`)` `percent` `format(`*format*`)` *list_options*]

`missings list` [*varlist*] [*if*] [*in*] [`,` *common_options* `minimum(`#`)` *list_options*]

`missings table` [*varlist*] [*if*] [*in*] [`,` *common_options* `minimum(`#`)` *tabulate_options*]

`missings tag` [*varlist*] [*if*] [*in*] `, generate(`*newvar*`)` [*common_options*]

`missings dropvars` [*varlist*] [`,` *common_options* `force`]

`missings dropobs` [*varlist*] [*if*] [*in*] [`,` *common_options* `force`]

common_options are `numeric`, `string`, and `sysmiss`.

`by:` may be used with any of `missings report`, `missings list`, or `missings table`. See [D] **by**.

4.2 Description

`missings` is a set of utility commands for managing variables that may have missing values. By default, "missing" means numeric missing (that is, the system missing value `.` or one of the extended missing values `.a` to `.z`) for numeric variables and empty or `""` for string variables. See [U] **12.2.1 Missing values** for further information.

If *varlist* is not specified, it is interpreted by default as all variables.

4.3 Options

`numeric` (all subcommands) indicates to include numeric variables only. If any string variables are named explicitly, such variables will be ignored.

`string` (all subcommands) indicates to include string variables only. If any numeric variables are named explicitly, such variables will be ignored.

`sysmiss` (all subcommands) indicates to include system missing `.` only. This option has no effect with string variables, for which missing is deemed to be the empty string `""`, regardless.

`observations` (`missings report`) indicates counting of missing values by observations, not the default of counting by variables.

`minimum(#)` (`missings report`, `missings list`, and `missings table`) specifies the minimum number of missings to be shown explicitly. With `missings table`, the default is `minimum(0)`; otherwise, it is `minimum(1)`.

`percent` (`missings report`) reports percents missing as well as counts. Percents are calculated relative to the number of observations or variables specified.

`format(`*format*`)` (`missings report`) specifies a display format for percents. The default is `format(%5.2f)`. This option has no effect unless `percent` is also specified.

list_options (`missings report` and `missings list`) are options listed in [D] **list** that may be specified when `list` is used to show results.

tabulate_options (`missings table`) are options listed in [R] **tabulate oneway** that may be specified when `tabulate` is used to show results.

`generate(`*newvar*`)` (`missings tag`) specifies the name of a new variable. `generate()` is required.

`force` (`missings dropvars` and `missings dropobs`) signals that the dataset in memory is being changed and is a required option when data are being dropped and the dataset in memory has not been saved as such.

5 Conclusion

When you look for and at missing values, several built-in commands can provide a first step. The aim of `missings` is to help in the second step so that early decisions can be made about what to do—all the way from dropping variables or observations that are not informative to considering multiple imputation. Minimally, knowing about missings can thus inform strategies for data management and data analysis. Not knowing about missings can just produce much puzzlement and many misleading results, so the benefits are clear.

6 Acknowledgments

Jeroen Weesie, Eric Uslaner, and Estie Sid Hudes contributed to the earlier development of `nmissing` and `dropmiss`.

7 References

Altman, D. G. 1991. *Practical Statistics for Medical Research*. London: Chapman & Hall.

Cox, N. J. 1999. dm67: Numbers of missing and present values. *Stata Technical Bulletin* 49: 7–8. Reprinted in *Stata Technical Bulletin Reprints*, vol. 9, pp. 26–27. College Station, TX: Stata Press.

———. 2001a. dm67.1: Enhancements to numbers of missing and present values. *Stata Technical Bulletin* 60: 2–3. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 7–9. College Station, TX: Stata Press.

———. 2001b. dm89: Dropping variables or observations with missing values. *Stata Technical Bulletin* 60: 7–8. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 44–46. College Station, TX: Stata Press.

———. 2003. Software Updates: dm67_2: Numbers of missing and present values. *Stata Journal* 3: 449.

———. 2005. Software Updates: dm67_3: Numbers of missing and present values. *Stata Journal* 5: 607.

———. 2008. Software Updates: dm89_1: Dropping variables or observations with missing values. *Stata Journal* 8: 594.

———. 2010a. Software Updates: dm0048_1: Finding variables. *Stata Journal* 10: 691–692.

———. 2010b. Speaking Stata: Finding variables. *Stata Journal* 10: 281–296.

———. 2012. Software Updates: dm0048_2: Finding variables. *Stata Journal* 12: 167.

———. 2015. Software Updates: dm0048_3: Finding variables. *Stata Journal* 15: 605–606.

Long, J. S. 2009. *The Workflow of Data Analysis Using Stata*. College Station, TX: Stata Press.

About the author

Nicholas Cox is a statistically minded geographer at Durham University. He contributes talks, postings, FAQs, and programs to the Stata user community. He has also coauthored 15 commands in official Stata. He was an author of several inserts in the *Stata Technical Bulletin* and is an editor of the *Stata Journal*. His "Speaking Stata" articles on graphics from 2004 to 2013 have been collected as *Speaking Stata Graphics* (College Station, TX: Stata Press, 2014).