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 3, pp. 607–626

precombine: A command to examine n $\geq$ 2 datasets before combining

Mark D. Chatfield
Menzies School of Health Research
Charles Darwin University
Darwin, Australia
Mark.Chatfield@menzies.edu.au

Abstract. In this article, I present a new command, `precombine`, that alerts the user to, and provides assurance concerning, some problems that can occur when multiple Stata datasets are merged and appended. It describes variables that are common to multiple datasets as well as variables that are unique to one dataset. Where value labels are attached to variables, it checks whether code sets are identical across datasets. Summary statistics for values of all variables can also be listed and left in memory along with the descriptions of each variable of each dataset.

Keywords: dm0081, precombine, data management, merge, append, joinby, cross, describe, uselabel, key, overlapping variables, common variables, same-named variables, label, value labels, code sets, codings, cfvars, vlc, cf, tabstat

1 Introduction

In his excellent book *Data Management Using Stata: A Practical Handbook*, Mitchell (2010, 215) writes, "There are several problems that can arise when merging datasets. Some problems produce no error messages and some produce innocent-looking messages. Only upon deeper inspection are these underlying problems revealed. Knowing about these problems can help you anticipate and avoid them". Mitchell's (2010) book contains a chapter describing how to combine datasets using Stata. It covers problems with `merge` and `append` that can arise when combining datasets, how the problems can be detected, and how to resolve them. Problems can occur when merging datasets that have variables with the same names (common variables), when appending datasets that have variables that are not common, when storage types or variable labels differ for common variables, and when value labels differ between datasets.

The `precombine` command alerts the user to, and provides assurance concerning, some problems that can occur when multiple Stata datasets are combined. It describes variables that are common to multiple datasets as well as variables that are unique to one dataset. Where value labels are attached to variables, the command checks whether code sets are identical across datasets.

Problems can also occur in relation to the values of variables that are common to multiple datasets. For example, a merge results in unmatched observations because of a data entry error. A program cannot be expected to detect problems of this nature.

That said, `precombine` can summarize the values of each variable of each dataset, which can help the user to get a quick feel for the data before combining the datasets.

2 Background

2.1 Common variables

Merging

`merge` is used to add new variables from a second dataset (the using dataset) to the dataset in Stata's memory (the master dataset) by joining observations through matching one or more specified variables (key variables). Before merging two datasets, the user should be familiar with the contents of each dataset and how the datasets relate to each other. The key variables must have the same variable names in both the master and using datasets before they can be used for merging. Other variables may also be common to the datasets (overlapping variables). For instance, say the variables in the master and using datasets are

- master: `id`, `seq`, `x1`, `x2`
- using: `id`, `bar`, `x1`, `x2`

and `id` is the key variable. Here the overlapping variables are `x1` and `x2`.

Before proceeding with a merge, the user should identify the common variables between the datasets. If there are any overlapping variables, the user can choose how to handle them by choosing, for each overlapping variable, one of the following four options:

1. Retain the overlapping variable (this is the default option). The values of overlapping variables will be combined in a way that depends on whether the user performs a "standard merge" (this is the default option) or an "update merge".
    a. In a standard merge, the data in the master dataset are the authority and are inviolable. Matched observations will contain values of the overlapping variables from the master dataset, while unmatched observations will contain values from the using dataset.
    b. An update merge differs from a standard merge in that matched observations will update missing values from the master dataset with values from the using dataset. Optionally, where there are conflicting nonmissing values, values from the using dataset can also take precedence over values from the master dataset. (See [D] **merge** for examples.)
2. Plan to merge on the overlapping variables as well as the key variables. Gould (2011) recommends merging on all overlapping variables that are expected to be

constant within the key variables—as well as on the key variables—to prevent wrong observations from being combined.

3. Rename the overlapping variable in the master and using datasets before merging (this would be a popular option). This option obviously makes sense if the meaning of the variable differs between datasets. It also makes sense when the meaning of the variable is the same, but the user wishes to compare the contents of the overlapping variable. Renaming the variable in both datasets (using previously unused variable names) is encouraged to prevent possible confusion of the origin (and hence the interpretation) of the variables.

4. Drop the overlapping variable from one of the datasets before merging. This option risks possible confusion of the origin (and hence the interpretation) unless the variable has a new name in the combined dataset.

If the two datasets in the above example are merged using any of options 1–3 for `x1` and `x2`, the resulting merged dataset will contain variables named `id`, `seq`, `x1`, `x2`, and `bar` and (optionally) a new variable to mark (merge) results (for example, `_merge`). Except for when the storage type of overlapping variables differs between datasets, `merge` currently provides no clues as to what the overlapping variables are. The user may miss some existing overlapping variables and not be aware of exactly how the overlapping variables have mixed. This is quite plausible, for example, when merging datasets that contain many variables.

Appending

`append` is used to add new observations from a dataset or datasets to the master dataset. Usually, all variable names need to be common between the datasets. Variable names that are unique to a dataset may need to be renamed so that values of these variables are combined with the values of corresponding variables in the other datasets.

Storage type

By default, `merge` and `append` issue error messages when there are mismatches in the generic storage type of common variables (that is, when a variable is a string in one dataset and numeric in another). If the `force` option is specified (and there are no such generic storage type mismatches in the key variables for `merge`), Stata issues a warning message before combining the data and treats values of these common variables in the using dataset as if they were missing.

If there are conflicts in numeric storage types of common variables (Stata has five numeric storage types), the storage type that is more precise will be used regardless of whether this storage type was in the master or using dataset. For string variables of different lengths, the longer string storage type will prevail. Stata will issue a warning if the more precise numeric or longer string storage type is in the using dataset; Stata will not do so if this is the case for variables in the master dataset.

Variable labels

When combining datasets, Stata will keep the variable labels for common variables in the master dataset. Stata will not inform you of differences in variable labels of common variables.

The user-written command `cfvars` (Cox 2011) identifies variable names that are common to two datasets (if any) and variable names that are unique for each dataset. `precombine` not only identifies these variables but also describes each variable's storage type, variable label, and associated value label (if any) as recorded by Stata's `describe` command. `precombine` also works for more than two datasets.

2.2 Value labels

A value label comprises a value-label name and an associated code set, and it is attached to one or more variables in a dataset.

Where a value-label name is used in multiple datasets and the multiple datasets are combined, Stata will issue a warning message—for example, `(label yesnolbl already defined)`—to alert the user that the code set associated with the value-label name in the master dataset will be the code set used in the combined dataset. At this point, the user should verify that the code sets are the same between datasets. This can be a somewhat laborious task, and a user may choose not to bother checking this and instead proceed assuming there are no problems.

Suppose now that a common variable is labeled with *valuelabel1* (*lblname1*+*codeset1*) in the master dataset and *valuelabel2* (*lblname2*+ *codeset2*) in the using dataset. When the datasets are combined, no warning is currently produced, and the common variable is labeled with *valuelabel1*. This could lead to some incorrectly labeled data if *codeset1* and *codeset2* are not compatible and to some data no longer being labeled if *codeset2* is an expansion of *codeset1*. The user should check for this potential problem before combining labeled datasets.

The user-written command `vlc` (Nichols 2008) enables, for one or more variables, a quick comparison of the code set used in a specified dataset with the code sets attached to the value-label name if also used in another dataset. `precombine` is recommended over `vlc` because it compares the code sets for all value-label names in the datasets, it works in the previously mentioned *lblname1*-versus-*lblname2* scenario, and it makes comparisons easy by specifying where there are differences in the code sets (if there are any).

2.3 Values of variables

Users usually look at, and perhaps summarize, the values of variables in each dataset to get a feel for the data before combining the datasets. Stata's `codebook` command is a very helpful tool for this. `codebook` produces a summary of each variable's values that differs according to whether a variable is string or numeric (although the number of missing values and the number of unique nonmissing values are reported for both). If

the variable is string, example values are listed. If the variable is numeric, `codebook` has a rule for deciding whether to produce summary statistics (as would be desirable for a continuous variable) or a frequency tabulation (which is nice when data are categorical or discrete) when there are not many values. To simplify quick comparisons of common variables between datasets, `precombine` can output some of the summaries (such as summary statistics that are returned by Stata's `summarize` command) conveniently on a separate line for each variable of each dataset.[1]

3 The precombine command

3.1 Syntax

`precombine` [*filename*] [*filename* ...] [`, currentdataset uniquevars describe(`*variabledetaillist*`) summarize(`*statnamelist*`) clear`]

You may enclose *filename* in double quotes and must do so if *filename* contains blanks or other special characters.

3.2 Options

`currentdataset` examines the dataset currently in memory together with the specified *filename*s.

`uniquevars` lists variables that appear in one dataset.

`describe(`*variabledetaillist*`)` lists details of variables that appear in multiple datasets. When `uniquevars` is specified, the option lists details of variables that appear in one dataset. Most of the details come from Stata's `describe, replace` command. *variabledetaillist* is a sequence of *variabledetail*s separated by spaces.

variabledetail	Description
`type`	storage type
`format`	display format
`vallabname`	value-label name
`varlab`	variable label
`ndta`	number of datasets that include variable
`isnumeric`	whether numeric or string (this can be inferred from type)
`position`	variable number in dataset

1. It is difficult to list examples of a string variable and tabulate frequency distributions of a numeric variable on one line of output, so `precombine` does not do this. `vlc` can tabulate, and save as a dataset, the frequency distribution of coded numerical variables provided they appear in every dataset. A more general method, however, would be to append datasets and, after one studies output from `precombine` and takes any necessary action, to generate a variable to distinguish the datasets the observations came from if need be and to then create two-way tables of variables with the dataset name.

`summarize(`*statnamelist*`)` lists summary statistics of variables that appear in multiple datasets. When `uniquevars` is specified, the option lists summary statistics of variables that appear in one dataset. Most of the summary statistics come from Stata's `summarize` command for numeric variables. *statnamelist* is a sequence of *statname*s separated by spaces.

statname	Description
`nmiss`	number of observations with missing values (also works for string variables)
`nobs`	number of observations with nonmissing values (also works for string variables)
`nuniq`	number of unique values (also works for string variables)
`mean`	mean
`sd`	standard deviation
`min`	minimum
`max`	maximum
`p1`	1st percentile
`p5`	5th percentile
`p10`	10th percentile
`p25`	25th percentile
`p50`	50th percentile
`p75`	75th percentile
`p90`	90th percentile
`p95`	95th percentile
`p99`	99th percentile
`sum`	sum of variable
`skewness`	skewness
`kurtosis`	kurtosis

`clear` specifies that the data in memory be cleared and replaced with all possible descriptions and summary statistics for each variable in each dataset, even if the original data have not been saved to disk.

3.3 Description

`precombine` produces reports to assist users in safely merging and appending datasets. If no datasets have been specified, all datasets in the current working directory are examined. The following reports are produced:

- For variables that appear in multiple datasets,
  - a list of the datasets that the variables appear in (optionally including details and summary statistics of the variables)

  - where the variables are numeric in one dataset and string in another, a list of the storage types (Stata will issue an error or a warning message if you attempt to merge and append these datasets)
  - where a value label is attached, a list of code sets if they are not identical

- For variables that appear in one dataset (and when the `uniquevars` option is specified),
  - a list of the variables and the dataset that they appear in (optionally including details and summary statistics of the variables)

- Where the same value-label names are used in multiple datasets,
  - a list of the datasets that the value-label names are used in (Stata will issue a warning message when you merge and append these datasets)
  - confirmation that the associated code sets are identical between datasets or a list of code sets when not identical

The above code sets are the sets of value labels associated with a variable. If a label is too long (for example, greater than 2,045 characters; see [D] **labelbook**), it may be truncated so that it fits in a string variable. Note that this definition completely ignores the value-label name. In addition to these reports, all details and summary statistics about each variable of each dataset can be left in memory.

4 Examples

Example 1

I construct the following example to illustrate every reporting feature of `precombine`. Suppose there are two Stata datasets: `_examination.dta` and `_medical_records.dta`. To show how `precombine` improves on the status quo, let's also suppose that there are two Stata users: Catherine (who has downloaded `precombine`) and Adam (who is not aware of `precombine`). Both users expect that a child's ID uniquely identifies observations in each dataset, and both users wish to merge the two datasets to achieve the same analysis objectives. In this example, Catherine and Adam each can uncover the same problems (though their approaches differ), and their response to the problem they both uncover happens to be identical (despite there being other valid responses). The users are followed up with until they have merged the datasets and determined whether they can safely ignore any warning messages given. Of course, the users would proceed to consider other features of the merge—such as the merge results—but this is not the focus of this article. For clarity, the (labeled and formatted) values in each dataset relating to the first child are listed below. It is very likely that the users would look at and perhaps summarize (for example, using `codebook`) the values of variables in each dataset to get a feel for the data before merging the datasets.

```
. use _examination, clear
. list if id==1, noobs abbreviate(12)
```

id	sex	exam_anaemic	exam_date	region
1	Female	Not sure	04apr2014	Darwin

```
. use _medical_records, clear
. list if id==1, noobs abbreviate(12)
```

id	sex	hx_anaemia	hx_date	region
1	F	Yes	01jan2013	Alice Springs

Adam begins by examining each dataset and studying the output.

```
. use _examination, clear
. describe
Contains data from _examination.dta
  obs:           533
 vars:             5                          21 Nov 2014 15:35
 size:         9,594
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
id              int     %9.0g                 Child ID
sex             float   %9.0g      sexlbl     Sex of child
exam_anaemic    float   %9.0g      yesnolbl   Child anaemic on examination
exam_date       float   %td                   Examination date
region          float   %9.0g      darwinlbl
                                              Region where exam took place
-------------------------------------------------------------------------------
Sorted by: id
. label list
sexlbl:
           1 Male
           2 Female
yesnolbl:
           0 No
           1 Yes
           9 Not sure
darwinlbl:
           1 Darwin
```

```
. use _medical_records, clear
. describe
Contains data from _medical_records.dta
  obs:           524
 vars:             5                          21 Nov 2014 15:35
 size:         7,860
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
id              int     %9.0g                 Child ID
sex             str1    %9s                   Sex of child
hx_anaemia      float   %9.0g      yesnolbl   Ever diagnosed as anaemic?
hx_date         float   %td                   Date of 1st anaemia diagnosis
region          float   %13.0g     regionlbl
                                              Region where first anaemic
-------------------------------------------------------------------------------
Sorted by: id
. label list
regionlbl:
           1 Darwin
           2 Alice Springs
yesnolbl:
           0 No
           1 Yes
```

Meanwhile, Catherine begins by using `precombine`. Immediately, six reports are produced. (Note that some uninteresting output has been suppressed. Note also that the reports appear nicer when the Results window is stretched wide.)

```
. precombine _examination _medical_records,
> describe(type format vallabname varlab) uniquevars
Reports relevant to the combining of the following datasets:
[vars: 5     obs: 533]      _examination
[vars: 5     obs: 524]      _medical_records
Variables that appear in multiple datasets:
```

variable	dataset	type	format	vallabname
varlab				

id	_examination	int	%9.0g	
Child ID				

id	_medical_records	int	%9.0g	
Child ID				

region	_examination	float	%9.0g	darwinlbl
Region where exam took place				

region	_medical_records	float	%13.0g	regionlbl
Region where first anaemic				

sex	_examination	float	%9.0g	sexlbl
Sex of child				

sex	_medical_records	str1	%9s	
Sex of child				

First report: Catherine sees that if she merges on `id`, she will have two overlapping variables, `region` and `sex`. The variable labels associated with region indicate that the meaning of the variable `region` differs between datasets, and Catherine decides to rename the variable in each dataset before merging so that it is no longer an overlapping variable. Catherine expects the information concerning the sex of the child to be constant within `id`, and she decides to merge on both `id` and `sex`. Catherine sees that while `id` is stored and labeled the same in the datasets, this is not the case for `sex`. `sex` is numeric and coded in `_examination.dta` but stored as a string in `_medical_records.dta`. She decides to change the storage type of `sex` in `_medical_records.dta` to match that of `sex` in `_examination.dta`.

```
When merging/appending, expect an error/warning message
 for the following variable(s) being a string in one dataset and numeric in
 another:
```

variable	dataset	type
sex	_examination	float
sex	_medical_records	str1

Second report: Because Catherine specified that she wanted the storage type to be listed for variables, she learns nothing new with this report.

```
Differences exist in the code sets* for these variables between datasets:
(*The set of {value, label} ignoring any differences in vallabname)
```

variable	dataset	value	label	vallabname
region	_examination	1	Darwin	darwinlbl
region	_medical_records	1	Darwin	regionlbl
region	_medical_records	2	Alice Springs	regionlbl

Third report: Catherine learns that differences exist in the code sets for `region` between datasets (and the value-label names are different). In this example, Catherine can ignore this because she has already decided to rename `region` in each dataset so

that it is no longer an overlapping variable. (In some situations, this particular report would be useful to show that a common variable has different value-label names and code sets between datasets.)

```
Variables that appear in only one dataset:
```

variable	dataset	type	format	vallabname
varlab				
exam_anaemic	_examination	float	%9.0g	yesnolbl
Child anaemic on examination				
exam_date	_examination	float	%td	
Examination date				
hx_anaemia	_medical_records	float	%9.0g	yesnolbl
Ever diagnosed as anaemic?				
hx_date	_medical_records	float	%td	
Date of 1st anaemia diagnosis				

Fourth report: Catherine learns the names of variables that are unique to each dataset as well as the other variable details she specifically asked for, such as the associated value-label names. In this example, this helps Catherine when renaming `region` to avoid existing variable names and similarly to avoid changing the name of an existing value-label name to another that already exists. Catherine can also now see which value-label names are associated with which variables. (When one merges in general, this report may not be of much help, especially if values are not labeled.)

```
When the following datasets are combined, expect warning message(s) of the form
(label <vallabname> already defined) ...
```

vallabname	dataset
yesnolbl	_examination
yesnolbl	_medical_records

Fifth report: Catherine learns that when the datasets are combined, she can expect to see the warning message: `(label yesnolbl already defined)`. This warning means that a value label named `yesnolbl` is associated with some variables in `_examination.dta` (`exam_anaemic`, here) and that a value label also named `yesnolbl` is associated with some variables in `_medical_records.dta` (`hx_anaemia`, here). The warning informs the user that the value labels in each dataset may differ.

```
... ignore the warning message(s) for the following vallabnames AT YOUR PERIL
    as differences in the code sets exist:
```

vallabname	dataset	value	label
yesnolbl	_examination	0	No
yesnolbl	_examination	1	Yes
yesnolbl	_examination	9	Not sure
yesnolbl	_medical_records	0	No
yesnolbl	_medical_records	1	Yes

Sixth report: Catherine learns that there are differences in the code sets associated with `yesnolbl` in each dataset. She can see from the list that the `value 9` is labeled `Not sure` in `_examination.dta`. This difference may not be that important in this example, but Catherine decides to rename the value-label name in `_examination.dta` before merging.

After investigating how `sex` is coded in `_examination.dta` and identifying what (string) values `sex` can take in `_medical_records.dta` (not shown), Catherine follows through with her previously determined changes.

```
. use _medical_records, clear
. rename region hx_region
. generate sex_tmp = 1 if sex=="M"
(260 missing values generated)
. replace sex_tmp = 2 if sex=="F"
(260 real changes made)
. drop sex
. rename sex_tmp sex
. label define sexlbl 1 "Male" 2 "Female"
. label values sex sexlbl
. label variable sex "Sex of child"
. save medical_records2, replace
(note: file medical_records2.dta not found)
file medical_records2.dta saved
. use _examination.dta, clear
. rename region exam_region
. label copy yesnolbl yesnounsurelbl
. label drop yesnolbl
. label values exam_anaemic yesnounsurelbl
```

She makes changes to `_medical_records.dta` and saves it as a new dataset called `medical_records2.dta`. She then makes changes to `_examination.dta` and, believing she may now be ready to merge (1:1 on `id` and `sex` using `medical_records2.dta`), uses the `precombine` command again to verify that things are as she anticipates they will be and to help her decide whether to ignore warning messages of the form `(label <vallabname> already defined)`.

```
. precombine medical_records2, currentdataset
> describe(type format vallabname varlab) uniquevars

Reports relevant to the combining of the following datasets:
[vars: 5     obs: 533]      <current>
[vars: 5     obs: 524]      medical_records2

Variables that appear in multiple datasets:
```

variable	dataset	type	format	vallabname	varlab
id	<current>	int	%9.0g		Child ID
id	medical_records2	int	%9.0g		Child ID
sex	<current>	float	%9.0g	sexlbl	Sex of child
sex	medical_records2	float	%9.0g	sexlbl	Sex of child

```
Variables that appear in only one dataset:
```

variable	dataset	type	format	vallabname
	varlab			
exam_anaemic	<current>	float	%9.0g	yesnounsurelbl
	Child anaemic on examination			
exam_date	<current>	float	%td	
	Examination date			
exam_region	<current>	float	%9.0g	darwinlbl
	Region where exam took place			
hx_anaemia	medical_records2	float	%9.0g	yesnolbl
	Ever diagnosed as anaemic?			
hx_date	medical_records2	float	%td	
	Date of 1st anaemia diagnosis			
hx_region	medical_records2	float	%13.0g	regionlbl
	Region where first anaemic			

```
When the following datasets are combined, expect warning message(s) of the form
(label <vallabname> already defined) ...
```

vallabname	dataset
sexlbl	<current>
sexlbl	medical_records2

```
... however you CAN safely ignore the warning message(s)
    as the code sets are consistent between datasets
```

Things are just as she hoped, and she now performs the merge. As anticipated by `precombine`, the warning message `(label sexlbl already defined)` appears, but Catherine knows she can safely ignore it because she has been told by `precombine` that the code sets are consistent between datasets (confirming that she did indeed define `sexlbl` on one dataset so that it matched the definition in the other). Catherine finishes with a good merged dataset.

```
. merge 1:1 id sex using medical_records2
(label sexlbl already defined)
```

(*output omitted*)

```
. drop _merge
. describe
Contains data from _examination.dta
  obs:           533
 vars:             8                          21 Nov 2014 15:35
 size:        15,990
```

variable name	storage type	display format	value label	variable label
id	int	%9.0g		Child ID
sex	float	%9.0g	sexlbl	Sex of child
exam_anaemic	float	%9.0g	yesnounsurelbl	Child anaemic on examination
exam_date	float	%td		Examination date
exam_region	float	%9.0g	darwinlbl	Region where exam took place
hx_anaemia	float	%9.0g	yesnolbl	Ever diagnosed as anaemic?
hx_date	float	%td		Date of 1st anaemia diagnosis
hx_region	float	%13.0g	regionlbl	Region where first anaemic

(*output omitted*)

```
. list if id==1, noobs abbreviate(12)
```

id	sex	exam_anaemic	exam_date	exam_region	hx_anaemia
1	Female	Not sure	04apr2014	Darwin	Yes

hx_date	hx_region
01jan2013	Alice Springs

Adam finished studying his list of variables and value labels in each dataset. He sees that if he merges on `id`, `sex` will be an overlapping variable. He expects the information concerning the sex of the child to be constant within `id`, and he decides to merge on both `id` and `sex`. Adam also sees that while `id` is stored and labeled the same in both datasets, this is not the case for `sex`. `sex` is numeric and coded in `_examination.dta`, but stored as a string in `_medical_records.dta`. He decides to change the storage type of `sex` in `_medical_records.dta` to match that of `sex` in `_examination.dta`. However, Adam did not see that `region` is also an overlapping variable or that there are differences in the code sets for `yesnolbl`. After investigating what (string) values `sex` can take in `_medical_records.dta` (not shown), he proceeds to merge the datasets as follows:

```
. use _medical_records, clear
. generate sex_tmp = 1 if sex=="M"
(260 missing values generated)
. replace sex_tmp = 2 if sex=="F"
(260 real changes made)
. drop sex
. rename sex_tmp sex
. label define sexlbl 1 "Male" 2 "Female"
. label values sex sexlbl
. label variable sex "Sex of child"
. merge 1:1 id sex using _examination
(label sexlbl already defined)
(label yesnolbl already defined)
  (output omitted)
. drop _merge
```

Adam gets two warning messages concerning value labels, and he considers double checking that the code sets are consistent between datasets. However, because he defined `sexlbl` on one dataset so that it would match the definition in the other, he trusts that he did this correctly and ignores the warning message. He also thinks he has already checked that the code sets associated with `yesnolbl` are identical. Adam finishes with a different merged dataset. He no longer has information on the region where the exam took place (because a variable called `region` exists in `_medical_records.dta`—this variable describes the region where the child was first anemic). He also does not have the label associated with the code `9` for the variable `exam_anaemic`: the code set associated with `yesnolbl` from the master dataset (`_medical_records.dta`) was used, and this code set was not exactly the same as the code set associated with `yesnolbl` from the using dataset (`_examination.dta`).

```
. list if id==1, noobs abbreviate(12)
```

id	hx_anaemia	hx_date	region	sex	exam_anaemic
1	Yes	01jan2013	Alice Springs	Female	9
exam_date					
04apr2014					

Example 2

A user might like to examine a summary of the values of certain variables between datasets. `precombine, clear` can create a dataset of summary statistics of each variable (along with descriptives such as variable storage type) from each dataset, and then the user can select what details he or she would like to see, on which variables, in what format, sorted as specified.

```
. precombine _examination _medical_records, clear
```

(*output omitted*)

```
. list variable dataset type nmiss nobs nuniq mean min max, sepby(variable) noobs
```

variable	dataset	type	nmiss	nobs	nuniq	mean
id	_examination	int	0	533	533	267
	min		max			
	1		533			

variable	dataset	type	nmiss	nobs	nuniq	mean
id	_medical_records	int	0	524	524	270.7271
	min		max			
	1		533			

variable	dataset	type	nmiss	nobs	nuniq	mean
region	_examination	float	0	533	1	1
	min		max			
	1		1			

variable	dataset	type	nmiss	nobs	nuniq	mean
region	_medical_records	float	0	524	2	1.080153
	min		max			
	1		2			

variable	dataset	type	nmiss	nobs	nuniq	mean
sex	_examination	float	0	533	2	1.497186
	min		max			
	1		2			

variable	dataset	type	nmiss	nobs	nuniq	mean
sex	_medical_records	str1	0	524	2	.
	min		max			
	.		.			

variable	dataset	type	nmiss	nobs	nuniq	mean
exam_anaemic	_examination	float	0	533	3	1.20075
min		max				
0		9				

variable	dataset	type	nmiss	nobs	nuniq	mean
exam_date	_examination	float	0	533	2	19848.85
min		max				
19817		19899				

variable	dataset	type	nmiss	nobs	nuniq	mean
hx_anaemia	_medical_records	float	0	524	2	.25
min		max				
0		1				

variable	dataset	type	nmiss	nobs	nuniq	mean
hx_date	_medical_records	float	410	114	104	19390.46
min		max				
19004		19689				

```
. format min max %td
. list variable dataset nmiss nobs nuniq min max if regexm(variable, "date")==1,
> sepby(variable) noobs
```

variable	dataset	nmiss	nobs	nuniq	min
exam_date	_examination	0	533	2	04apr2014
max					
25jun2014					

variable	dataset	nmiss	nobs	nuniq	min
hx_date	_medical_records	410	114	104	12jan2012
max					
27nov2013					

Example 3

When one prepares to append datasets, `precombine` with the `uniquevars` option can be helpful. In this example, the command ensures that the datasets will be appended nicely—all variables appear in all datasets. Summary statistics can be listed at the same time.

```
. precombine http://www.stata-press.com/data/r14/capop
> http://www.stata-press.com/data/r14/ilpop
> http://www.stata-press.com/data/r14/txpop, uniquevars describe(type)
> summarize(nmiss min max)

Reports relevant to the combining of the following datasets:
[vars: 2     obs: 3]        http://www.stata-press.com/data/r14/capop
[vars: 2     obs: 3]        http://www.stata-press.com/data/r14/ilpop
[vars: 2     obs: 3]        http://www.stata-press.com/data/r14/txpop

Variables that appear in multiple datasets:
```

variable	dataset	type	nmiss
min	max		

county	http://www.stata-press.com/data/r14/capop	str11	0
.	.		

county	http://www.stata-press.com/data/r14/ilpop	str7	0
.	.		

county	http://www.stata-press.com/data/r14/txpop	str7	0
.	.		

pop	http://www.stata-press.com/data/r14/capop	long	0
798364	9878554		

pop	http://www.stata-press.com/data/r14/ilpop	long	0
103729	5285107		

pop	http://www.stata-press.com/data/r14/txpop	long	0
149797	4011475		

```
There are no variables that appear in only one dataset,
 i.e. every variable appears in multiple datasets.

. use http://www.stata-press.com/data/r14/capop, clear

. append using http://www.stata-press.com/data/r14/ilpop
> http://www.stata-press.com/data/r14/txpop
```

Example 4

`precombine` can clarify how more than two datasets relate to one another and can indicate what unwanted overlapping variables exist. It can automatically look at all Stata datasets in a folder.

```
. cd "M:\Respiratory Study\Tables imported from database into Stata"
. precombine, describe(type format vallabname varlab)
```

5 Concluding remarks

`precombine` is a new command that enables users to examine multiple Stata datasets before combining them. It describes variables common to multiple datasets as well as variables unique to one dataset. Where value labels are attached to variables, `precombine` checks whether code sets are identical across datasets. It can also help users to get a quick feel for the data by listing some summary statistics for the values of variables.

While `precombine` was written to alert the user to, and provide assurance concerning, some problems that can occur when multiple datasets are merged and appended, `precombine` can also help when a user is interested in the distribution of certain variables in various datasets (whether the user wishes to combine the datasets). Users can use the command with the `clear` option to summarize the distribution of values of some (especially continuous) variables, to examine amounts of missing data, or to simply describe aspects of the variables (for example, storage type) over multiple datasets. The names of all variables (and the datasets they occur in) will then be known (and searchable), and summary statistics for a specified subset of variables can be seen by issuing another Stata command (for example, `browse if`). Of course, other approaches can also help in this more general setting (for example, using the `cf` command when there are two nearly identical datasets).

6 Acknowledgments

I am grateful for suggestions on drafts of the article from Matthew Stevens and Michael Binks (Menzies), Suzanna Vidmar (Murdoch Children's Research Institute, Australia), and an anonymous reviewer who also inspired me to add the `summarize` and `clear` options to the command.

7 References

Cox, N. J. 2011. cfvars: Stata module to compare variable name lists in two data sets. Statistical Software Components S457004, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457004.html.

Gould, W. 2011. Merging data, part 1: Merges gone bad. The Stata Blog: Not Elsewhere Classified. http://blog.stata.com/2011/04/18/merging-data-part-1-merges-gone-bad/.

Mitchell, M. N. 2010. *Data Management Using Stata: A Practical Handbook*. College Station, TX: Stata Press.

Nichols, A. 2008. vlc: Stata module to compare value labels across datasets. Statistical Software Components S456907, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s456907.html.

About the author

Mark Chatfield is a senior biostatistician and is Head of Biostatistics at Menzies School of Health Research, Charles Darwin University. He has been a Stata user for 13 years.