Data Management Using Stata: A Practical Handbook

MICHAEL N. MITCHELL

A Stata Press Publication
StataCorp LP
College Station, Texas

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845
Typeset in LaTeX 2ε
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN-10: 1-59718-076-9
ISBN-13: 978-1-59718-076-4

Library of Congress Control Number: 2010926561

(Pages omitted)

Contents

(Pages omitted)

Preface

There is a gap between raw data and statistical analysis. That gap, called data management, is often filled with a mix of pesky and strenuous tasks that stand between you and your data analysis. I find that data management usually involves some of the most challenging aspects of a data analysis project. I wanted to write a book showing how to use Stata to tackle these pesky and challenging data-management tasks.

One of the reasons I wanted to write such a book was to be able to show how useful Stata is for data management. Sometimes people think that Stata's strengths lie solely in its statistical capabilities. I have been using Stata and teaching it to others for over 10 years, and I continue to be impressed with the way that it combines power with ease of use for data management. For example, take the `reshape` command. This simple command makes it a snap to convert a wide file to a long file and vice versa (for examples, see section 8.3). Furthermore, `reshape` is partly based on the work of a Stata user, illustrating that Stata's power for data management is augmented by user-written programs that you can easily download (as illustrated in section 10.2).

Each section of this book generally stands on its own, showing you how you can do a particular data-management task in Stata. Take, for example, section 2.4, which shows how you can read a comma-delimited file into Stata. This is not a book you need to read cover to cover, and I would encourage you to jump around to the topics that are most relevant for you.

Data management is a big (and sometimes daunting) task. I have written this book in an informal fashion, like we were sitting down together at the computer and I was showing you some tips about data management. My aim with this book is to help you easily and quickly learn what you need to know to skillfully use Stata for your data-management tasks. But if you need further assistance solving a problem, section 10.3 describes the rich array of online Stata resources available to you. I would especially recommend the Statalist listserver, which allows you to tap into the knowledge of Stata users around the world.

If you would like to contact me with comments or suggestions, I would love to hear from you. You can write me at MichaelNormanMitchell@gmail.com, or visit me on the web at http://www.MichaelNormanMitchell.com. Writing this book has been both a challenge and a pleasure. I hope that you like it!

Simi Valley, CA
April 2010 — Michael N. Mitchell

(Pages omitted)

6.1 Introduction

This chapter describes how to combine datasets using Stata. It also covers problems that can arise when combining datasets, how you can detect them, and how to resolve them. This chapter covers four general methods of combining datasets: appending, merging, joining, and crossing. Section 6.2 covers the basics of how to append datasets, and section 6.3 illustrates problems that can arise when appending datasets. The next four sections cover four different kinds of merging—one-to-one match-merging (section 6.4), one-to-many match-merging (section 6.5), merging multiple datasets (section 6.6), and update merges (see section 6.7). Then section 6.8 discusses options that are common to each of these merging situations, and section 6.9 illustrates problems that can arise when merging datasets. The concluding sections cover joining datasets (section 6.10) and crossing datasets (section 6.11).

I should note that a new syntax was introduced in Stata 11 for the `merge` command. This new syntax introduces several new safeguards and features. This chapter exclusively illustrates this new syntax for the `merge` command, and thus these examples will not work in versions of Stata prior to version 11. Although not presented here, the syntax for the `merge` command from earlier versions of Stata continues to work using Stata 11.

6.2 Appending: Appending datasets

Consider `moms.dta` and `dad.dta`, presented below. Each dataset has four observations, the first about four moms and the second about four dads. Each dataset contains a family ID, the age of the person, his or her race, and whether he or she is a high school graduate.

```
. use moms

. list

     +---------------------------+
     | famid   age   race   hs   |
     |---------------------------|
  1. |     3    24      2    1   |
  2. |     2    28      1    1   |
  3. |     4    21      1    0   |
  4. |     1    33      2    1   |
     +---------------------------+

. use dads

. list

     +---------------------------+
     | famid   age   race   hs   |
     |---------------------------|
  1. |     1    21      1    0   |
  2. |     4    25      2    1   |
  3. |     2    25      1    1   |
  4. |     3    31      2    1   |
     +---------------------------+
```

Suppose that we wanted to stack these datasets on top of each other so that we would have a total of eight observations in the combined dataset. The `append` command is used for combining datasets like this, as illustrated below. First, we clear any data from memory. Then, after the `append` command, we list all the datasets we want to append together. Although we specified only two datasets, we could have specified more than two datasets on the `append` command.

```
. clear
. append using moms dads
```

The `list` command below shows us that these two files were appended successfully.

```
. list

     ┌───────────────────────────┐
     │ famid   age   race   hs   │
     ├───────────────────────────┤
  1. │     3    24      2    1   │
  2. │     2    28      1    1   │
  3. │     4    21      1    0   │
  4. │     1    33      2    1   │
  5. │     1    21      1    0   │
     ├───────────────────────────┤
  6. │     4    25      2    1   │
  7. │     2    25      1    1   │
  8. │     3    31      2    1   │
     └───────────────────────────┘
```

Suppose that you already had `moms.dta` loaded in memory, as shown below.

```
. use moms
```

At this point, you can append `dads.dta` like this:

```
. append using dads
. list

     ┌───────────────────────────┐
     │ famid   age   race   hs   │
     ├───────────────────────────┤
  1. │     3    24      2    1   │
  2. │     2    28      1    1   │
  3. │     4    21      1    0   │
  4. │     1    33      2    1   │
  5. │     1    21      1    0   │
     ├───────────────────────────┤
  6. │     4    25      2    1   │
  7. │     2    25      1    1   │
  8. │     3    31      2    1   │
     └───────────────────────────┘
```

(*Continued on next page*)

Tip! Appending jargon

In the last example, we call `moms.dta` the *master* dataset because it is the dataset in memory when the append is initiated. `dads.dta` is called the *using* dataset because it is specified after the `using` keyword.

However we `append` these datasets, the combined file does not identify the source of the data. We cannot tell whether an observation originated from `moms.dta` or from `dads.dta`. To solve this, we can add the `generate()` option, which will create a new variable that tells us from which dataset each observation came. You can name this variable anything you like; I called it `datasrc`.

```
. clear
. append using moms dads, generate(datasrc)
. list, sepby(datasrc)

     +------------------------------------+
     | datasrc   famid   age   race   hs |
     |------------------------------------|
  1. |       1       3    24      2    1 |
  2. |       1       2    28      1    1 |
  3. |       1       4    21      1    0 |
  4. |       1       1    33      2    1 |
     |------------------------------------|
  5. |       2       1    21      1    0 |
  6. |       2       4    25      2    1 |
  7. |       2       2    25      1    1 |
  8. |       2       3    31      2    1 |
     +------------------------------------+
```

Looking back at the original data, we can see that when `datasrc` is 1, the data originate from `moms.dta`. When `datasrc` is 2, the data originate from `dads.dta`. If we had a third dataset on the `append` command, `datasrc` would have been 3 for the observations from that dataset.

Contrast this with the strategy where we first `use` the `moms.dta` dataset and then `append` the dataset `dads.dta`, as shown below.

```
. use moms
. append using dads, generate(datasrc)
. list, sepby(datasrc)

     ┌──────────────────────────────────┐
     │ famid   age   race   hs   datasrc │
     ├──────────────────────────────────┤
  1. │     3    24      2    1         0 │
  2. │     2    28      1    1         0 │
  3. │     4    21      1    0         0 │
  4. │     1    33      2    1         0 │
     ├──────────────────────────────────┤
  5. │     1    21      1    0         1 │
  6. │     4    25      2    1         1 │
  7. │     2    25      1    1         1 │
  8. │     3    31      2    1         1 │
     └──────────────────────────────────┘
```

Here a 0 means that the data came from the master dataset (i.e., `moms.dta`), and having a 1 means that the data came from the first using dataset (i.e., `dads.dta`). Had a second dataset been added after `dads` on the `append` command, the value for `datasrc` for those observations would have been 2.

The `label define` and `label values` commands below are used to label the values of `datasrc` (as described in section 4.4). Although I think labeling values is useful, it is optional.

```
. label define source 0 "From moms.dta" 1 "From dads.dta"
. label values datasrc source
. list, sepby(datasrc)

     ┌────────────────────────────────────────┐
     │ famid   age   race   hs         datasrc │
     ├────────────────────────────────────────┤
  1. │     3    24      2    1   From moms.dta │
  2. │     2    28      1    1   From moms.dta │
  3. │     4    21      1    0   From moms.dta │
  4. │     1    33      2    1   From moms.dta │
     ├────────────────────────────────────────┤
  5. │     1    21      1    0   From dads.dta │
  6. │     4    25      2    1   From dads.dta │
  7. │     2    25      1    1   From dads.dta │
  8. │     3    31      2    1   From dads.dta │
     └────────────────────────────────────────┘
```

As mentioned earlier, you can append multiple datasets at one time. For example, we have three datasets that contain book review information from three different reviewers: Clarence, Isaac, and Sally. The datasets are listed below using the `dir` command.

```
. dir br*.dta
   0.8k   2/02/10 18:48  br_clarence.dta
   0.8k   2/02/10 18:48  br_isaac.dta
   0.8k   2/02/10 18:48  br_sally.dta
```

The datasets all have the same variables in them. Below we can see the dataset containing the reviews from Clarence. This includes a variable identifying the book number (booknum), the name of the book (book), and the rating of the book (rating).

```
. use br_clarence
. list

     ┌───────────────────────────────────────────────────────────────┐
     │ booknum                                      book      rating │
     ├───────────────────────────────────────────────────────────────┤
  1. │       1                 A Fistful of Significance         5   │
  2. │       2    For Whom the Null Hypothesis is Rejected        10   │
  3. │       3   Journey to the Center of the Normal Curve         6   │
     └───────────────────────────────────────────────────────────────┘
```

Let's use the append command to combine all three datasets together. In doing so, we will use the generate() option to create a variable named rev that indicates the source of the data (i.e., the reviewer).

```
. clear
. append using br_clarence br_isaac br_sally, generate(rev)
. list, sepby(rev)

     ┌─────────────────────────────────────────────────────────────────────┐
     │ rev   booknum                                      book      rating │
     ├─────────────────────────────────────────────────────────────────────┤
  1. │   1         1                 A Fistful of Significance           5 │
  2. │   1         2    For Whom the Null Hypothesis is Rejected          10 │
  3. │   1         3   Journey to the Center of the Normal Curve           6 │
     ├─────────────────────────────────────────────────────────────────────┤
  4. │   2         1                     The Dreaded Type I Error          6 │
  5. │   2         2                            How to Find Power          9 │
  6. │   2         3                                 The Outliers          8 │
     ├─────────────────────────────────────────────────────────────────────┤
  7. │   3         1             Random Effects for Fun and Profit         6 │
  8. │   3         2                             A Tale of t-tests         9 │
  9. │   3         3             Days of Correlation and Regression        8 │
     └─────────────────────────────────────────────────────────────────────┘
```

The value of rev is 1, 2, or 3 for the observations that came from br_clarence, br_isaac, or br_sally, respectively.

This covers the basics of using the append command. The next section covers some of the problems that can arise when appending datasets.

6.3 Appending: Problems

The last section showed how easy it is to append datasets, but it ignored some of the problems that can arise when appending datasets. This section describes five problems that can arise when appending datasets: differing variable names across datasets, conflicting variable labels, conflicting value labels, inconsistent variable coding, and mixing variable types across datasets. These are discussed one at a time below.

(Pages omitted)

6.4 Merging: One-to-one match-merging

A match-merge combines two datasets using one (or more) key variables to link observations between the two datasets. In a one-to-one match-merge, the key variable(s) uniquely identifies each observation in each dataset. Consider the moms1.dta and dads1.dta datasets, below. The key variable, famid, uniquely identifies each observation in each dataset and can be used to link the observations from moms.dta with the observations from dads.dta. Because these datasets are so small, you can see that each observation from moms.dta has a match in dads.dta based on famid.

```
. use moms1

. list

     ┌───────────────────────────────┐
     │ famid   mage   mrace   mhs    │
     ├───────────────────────────────┤
  1. │     1     33       2     1    │
  2. │     2     28       1     1    │
  3. │     3     24       2     1    │
  4. │     4     21       1     0    │
     └───────────────────────────────┘

. use dads1

. list

     ┌───────────────────────────────┐
     │ famid   dage   drace   dhs    │
     ├───────────────────────────────┤
  1. │     1     21       1     0    │
  2. │     2     25       1     1    │
  3. │     3     31       2     1    │
  4. │     4     25       2     1    │
     └───────────────────────────────┘
```

We perform a 1:1 merge between moms1.dta and dads1.dta, linking them based on famid.

```
. use moms1

. merge 1:1 famid using dads1
    Result                           # of obs.
    ─────────────────────────────────────────
    not matched                             0
    matched                                 4  (_merge==3)
    ─────────────────────────────────────────
```

The output from the merge command confirms our expectations that each observation from moms.dta has a matched observation in dads.dta (and vice versa). We can see this for ourselves by listing the merged dataset.

(Continued on next page)

```
. list

     +-----------------------------------------------------------------+
     | famid   mage   mrace   mhs   dage   drace   dhs          _merge |
     |-----------------------------------------------------------------|
  1. |     1     33       2     1     21       1     0     matched (3) |
  2. |     2     28       1     1     25       1     1     matched (3) |
  3. |     3     24       2     1     31       2     1     matched (3) |
  4. |     4     21       1     0     25       2     1     matched (3) |
     +-----------------------------------------------------------------+
```

The listing shows the `famid` variable followed by the variables from `moms.dta` and then the variables from `dads.dta`. The last variable, `_merge`, was created by the `merge` command to show the matching status for each observation. In this example, every observation shows `matched (3)`, indicating that a match was found between the master and using dataset for every observation.

Tip! Merging jargon

In this example, `moms1.dta` is the *master* dataset because it is the dataset in memory when the `merge` command is issued. `dads1.dta` is called the *using* dataset because it is specified after the `using` keyword. The variable `famid` is called the *key variable* because it holds the key to linking the master and using files.

Let's consider a second example that involves some observations that do not match. Let's merge and inspect the datasets `moms2.dta` and `dads2.dta`.

```
. use moms2

. list

     +------------------------------------------+
     | famid   mage   mrace   mhs   fr_moms2 |
     |------------------------------------------|
  1. |     1     33       2     1          1 |
  2. |     3     24       2     1          1 |
  3. |     4     21       1     0          1 |
  4. |     5     39       2     0          1 |
     +------------------------------------------+

. use dads2

. list

     +------------------------------------------+
     | famid   dage   drace   dhs   fr_dads2 |
     |------------------------------------------|
  1. |     1     21       1     0          1 |
  2. |     2     25       1     1          1 |
  3. |     4     25       2     1          1 |
     +------------------------------------------+
```

Note how `moms2.dta` has an observation for family 3 and an observation for family 5 with no corresponding observations in `dads2.dta`. Likewise, `dads2.dta` has an observation for family 2, but there is no corresponding observation in `moms2.dta`. These observations will not be matched. When we merge these files, Stata will tell us about these nonmatched observations and help us track them, as we can see below.

```
. use moms2

. merge 1:1 famid using dads2
    Result                           # of obs.
    -----------------------------------------
    not matched                             3
        from master                         2  (_merge==1)
        from using                          1  (_merge==2)

    matched                                 2  (_merge==3)
    -----------------------------------------
```

The merge command summarizes how the matching went. Two observations were matched and three observations were not matched. Among the nonmatched observations, two observations originated from the master (moms2.dta) dataset, and one nonmatched observation originated from the using (dads2.dta) dataset. Let's now list the resulting merged dataset. (I first sorted the dataset on famid to make the listing easier to follow.)

```
. sort famid

. list famid mage mrace dage drace _merge

     +-------------------------------------------------------+
     | famid   mage   mrace   dage   drace            _merge |
     |-------------------------------------------------------|
  1. |     1     33       2     21       1       matched (3) |
  2. |     2      .       .     25       1    using only (2) |
  3. |     3     24       2      .       .   master only (1) |
  4. |     4     21       1     25       2       matched (3) |
  5. |     5     39       2      .       .   master only (1) |
     +-------------------------------------------------------+
```

Families 3 and 5 have data from moms2.dta (master) but not dads2.dta (using). The _merge variable confirms this by displaying master only (1). Family 2 has data from dads2.dta (using) but not moms2.dta (master). The _merge variable informs us of this by displaying using only (2) for this observation. Families 1 and 4 had matched observations between the master and using datasets, and this is also indicated in the _merge variable, which shows matched (3).

Let's look more closely at the _merge variable. This variable, which tells us about the matching status for each observation, might appear to be a string variable, but it is a numeric variable. We can see this using the codebook command.

(*Continued on next page*)

```
. codebook _merge
--------------------------------------------------------------------------------
_merge                                                               (unlabeled)
--------------------------------------------------------------------------------

                  type:  numeric (byte)
                 label:  _merge

                 range:  [1,3]                        units:  1
         unique values:  3                        missing .:  0/5

            tabulation:  Freq.   Numeric  Label
                             2         1  master only (1)
                             1         2  using only (2)
                             2         3  matched (3)
```

The value for the `_merge` variable is just the number 1, 2, or 3 with a value label providing a more descriptive label. If we want to list just the matched observations, we can specify `if _merge == 3` with the `list` command, as shown below.

```
. list famid mage mrace dage drace _merge if _merge == 3

     +-----------------------------------------------------+
     | famid   mage   mrace   dage   drace          _merge |
     |-----------------------------------------------------|
  1. |     1     33       2     21       1     matched (3) |
  4. |     4     21       1     25       2     matched (3) |
     +-----------------------------------------------------+
```

Or we could list the observations that only originated from the master dataset (`moms2.dta`) like this:

```
. list famid mage mrace dage drace _merge if _merge == 1

     +---------------------------------------------------------+
     | famid   mage   mrace   dage   drace              _merge |
     |---------------------------------------------------------|
  3. |     3     24       2      .       .     master only (1) |
  5. |     5     39       2      .       .     master only (1) |
     +---------------------------------------------------------+
```

We could keep just the matched observations by using the `keep` command, as shown below.[2]

```
. keep if _merge == 3
(3 observations deleted)

. list famid mage mrace dage drace _merge

     +-----------------------------------------------------+
     | famid   mage   mrace   dage   drace          _merge |
     |-----------------------------------------------------|
  1. |     1     33       2     21       1     matched (3) |
  2. |     4     21       1     25       2     matched (3) |
     +-----------------------------------------------------+
```

When merging `moms2.dta` and `dads2.dta`, we called this a one-to-one merge because we assumed that `moms2.dta` contained one observation per `famid` and, likewise, `dads2.dta` contained one observation per `famid`. Suppose that one of the datasets

[2] This could also be done using the `keep()` option, as illustrated in section 6.8.

had more than one observation per `famid`. `momsdup.dta` is such a dataset. This value of `famid` is accidentally repeated for the last observation (it shows as 4 for the last observation but should be 5).

```
. use momsdup

. list

     ┌────────────────────────────────────────┐
     │ famid   mage   mrace   mhs   fr_moms2 │
     ├────────────────────────────────────────┤
  1. │     1     33       2     1          1 │
  2. │     3     24       2     1          1 │
  3. │     4     21       1     0          1 │
  4. │     4     39       2     0          1 │
     └────────────────────────────────────────┘
```

This mistake should have been caught as a part of checking for duplicates (as described in section 3.8) on the `famid` variable, but suppose that we did not notice this. Fortunately, Stata catches this when we perform a one-to-one merge between `momsdup.dta` and `dads2.dta`, as shown below.

```
. use momsdup

. merge 1:1 famid using dads2
variable famid does not uniquely identify observations in the master data
r(459);
```

The error message is alerting us that `famid` does not uniquely identify observations in the master dataset (`momsdup.dta`). For a one-to-one merge, Stata checks both the master and the using datasets to make sure that the key variable(s) uniquely identifies the observations in each dataset. If not, an error message like the one above is displayed.

So far, all the examples have used one key variable for linking the master and using datasets, but it is possible to have two or more key variables that are used to link the master and using datasets. For example, consider `kids1.dta`, below.

```
. use kids1

. sort famid kidid

. list

     ┌──────────────────────────────┐
     │ famid   kidid   kage   kfem │
     ├──────────────────────────────┤
  1. │     1       1      3      1 │
  2. │     2       1      8      0 │
  3. │     2       2      3      1 │
  4. │     3       1      4      1 │
  5. │     3       2      7      0 │
     ├──────────────────────────────┤
  6. │     4       1      1      0 │
  7. │     4       2      3      0 │
  8. │     4       3      7      0 │
     └──────────────────────────────┘
```

It takes two variables to identify each kid: `famid` and `kidid`. Let's merge this dataset with another dataset named `kidname.dta` (shown below).

```
. use kidname
. sort famid kidid
. list

     +-----------------------+
     | famid   kidid   kname |
     |-----------------------|
  1. |     1       1     Sue |
  2. |     2       1     Vic |
  3. |     2       2     Flo |
  4. |     3       1     Ivy |
  5. |     3       2     Abe |
     |-----------------------|
  6. |     4       1     Tom |
  7. |     4       2     Bob |
  8. |     4       3     Cam |
     +-----------------------+
```

The kids in these two files can be uniquely identified and linked based on the combination of `famid` and `kidid`. We can use these two variables together as the key variables for merging these two files, as shown below.

```
. use kids1
. merge 1:1 famid kidid using kidname
    Result                           # of obs.
    -----------------------------------------
    not matched                             0
    matched                                 8  (_merge==3)
    -----------------------------------------
```

The output from the `merge` command shows that all the observations in the merged file were matched. Below we can see the merged dataset.

```
. list

     +----------------------------------------------------------+
     | famid   kidid   kage   kfem   kname            _merge |
     |----------------------------------------------------------|
  1. |     1       1      3      1     Sue   matched (3) |
  2. |     2       1      8      0     Vic   matched (3) |
  3. |     2       2      3      1     Flo   matched (3) |
  4. |     3       1      4      1     Ivy   matched (3) |
  5. |     3       2      7      0     Abe   matched (3) |
     |----------------------------------------------------------|
  6. |     4       1      1      0     Tom   matched (3) |
  7. |     4       2      3      0     Bob   matched (3) |
  8. |     4       3      7      0     Cam   matched (3) |
     +----------------------------------------------------------+
```

This concludes this section on one-to-one merging. This section did not address any of the problems that can arise in such merges. Section 6.9 discusses problems that can arise when merging datasets, how to discover them, and how to deal with them.

6.5 Merging: One-to-many match-merging

Section 6.4 showed a 1:1 merge that merged moms with dads. This was called a 1:1 merge because the key variable(s) uniquely identified each observation within each dataset. By contrast, when matching moms to kids, a mom could match with more than one kid (a one-to-many merge). The moms dataset is the 1 dataset and the kids dataset is the m dataset. Despite this difference, the process of performing a 1:m merge is virtually identical to the process of performing a 1:1 merge. This is illustrated by merging moms1.dta with kids1.dta. These two datasets are shown below.

```
. use moms1
. list

     +-----------------------------+
     | famid   mage   mrace   mhs  |
     |-----------------------------|
  1. |     1     33       2     1  |
  2. |     2     28       1     1  |
  3. |     3     24       2     1  |
  4. |     4     21       1     0  |
     +-----------------------------+

. use kids1
. list

     +------------------------------+
     | famid   kidid   kage   kfem  |
     |------------------------------|
  1. |     3       1      4      1  |
  2. |     3       2      7      0  |
  3. |     2       1      8      0  |
  4. |     2       2      3      1  |
  5. |     4       1      1      0  |
     |------------------------------|
  6. |     4       2      3      0  |
  7. |     4       3      7      0  |
  8. |     1       1      3      1  |
     +------------------------------+
```

The variable famid links the moms with the kids. You can see that the mom in family 1 will match to one child, but the mom in family 4 will match to three children. You can also see that for every mom, there is at least one matched child, and every child has a matching mom. We merge these two datasets below.

```
. use moms1
. merge 1:m famid using kids1
    Result                           # of obs.
    -----------------------------------------
    not matched                             0
    matched                                 8  (_merge==3)
    -----------------------------------------
```

The report shows that all observations were matched.

We can see the resulting merged dataset below. The dataset is sorted on famid and kidid to make the listing easier to follow.