SOCY498C—Introduction to Computing for Sociologists
Neustadtl

Combining Data

Combining data taken from several datasets is a common data management task. The datasets are either "appended" or "merged". Appending means added to the bottom of an existing dataset—adding cases. Merging datasets means adding data side-by-side to an existing dataset—adding variables. There are several types of merges including one-to-one merges and match merging (one-to-many and many-to-one merges). Merging is often used with the `collapse` command which creates aggregate statistical data. A final complication is how missing values and differently named variables in the different datasets are managed when combining data.

It is common for data, especially survey data, to come in multiple datasets (there are practical reasons for distributing datasets this way). When data are distributed in multiple files, the variables you want to use will often be scattered across several datasets. In order to work with information contained in two or more data files it is necessary to merge the segments into a new file that contains all of the variables you intend to work with.

First, you'll need to determine which variables you need and which datasets contain them. You can do this by consulting the codebook. Additionally, you need to know the name of the *id* variable (or variables). An *id* variable is a variable that is unique to a case (observation) in the dataset. For a given individual, the *id* should be the same across all datasets allowing you to match the data from different datasets to the correct observation. For cross-sectional data, this will typically be a single variable, in other cases, two or more variables are needed, this is commonly seen in panel data where subject id and date or wave are often needed to uniquely identify an observation. In order for Stata to merge the datasets, the *id* variables have to have the same name across all files. If the variable is a string in one dataset, it must also be a string in all other datasets, and the same is true of numeric variables (the specific storage type is not important, as long as they are numerical). Once you have identified the variables you want, and know what the *id* variable(s) are, you can begin to merge the datasets.

`append` (`help append`)

- This command appends a Stata-format dataset stored on disk to the end of the dataset in memory.
- You can specify the variables to append.
- You can control if labels and notes are retained.

`merge` (`help merge`)

- `merge` joins corresponding observations from the dataset currently in memory (called the master dataset) with those from Stata-format datasets stored as filename (called the using datasets) into single observations.
- You can specify the variables to append.
- You can control if labels and notes are retained.

append

Sometimes parallel datasets are created during data collection. For example if there are multiple data collection sites. The following data measure patient id (*pid*), the date a blood glucose measurement was taken (*datestamp*), the BG reading, and the data collection site (*site*). Note that there are two patients in each dataset with different numbers of records for each patient. The append command will create a new dataset that contains all of these cases.

Data Site #1

1634	"04/22/05"	213	1
1634	"04/27/05"	117	1
1701	"04/09/08"	232	1
1701	"04/09/08"	324	1
1701	"04/10/08"	250	1
1701	"04/11/08"	213	1

Data Site #4

1525	"03/17/06"	64	4
1525	"03/17/06"	165	4
1525	"03/17/06"	72	4
1525	"03/17/06"	123	4
1683	"05/05/08"	156	4
1683	"05/05/08"	141	4
1683	"05/06/08"	121	4

The data need to be stored in Stata dataset format. Let's assume that we have two files called "site1.dta" and "site4.dta". To append then we would do the following:

```
. use site1, clear
. append using site4
. list, sepby(pid site) noobs
```

pid	datest~p	bglevel	site
1634	04/22/05	213	1
1634	04/27/05	117	1
1701	04/09/08	232	1
1701	04/09/08	324	1
1701	04/10/08	250	1
1701	04/11/08	213	1
1525	03/17/06	64	4
1525	03/17/06	165	4
1525	03/17/06	72	4
1525	03/17/06	123	4
1683	05/05/08	156	4
1683	05/05/08	141	4
1683	05/06/08	121	4

In this example the people at site 1 also collected a blood based measure called HbA1c—these measurements were not taken at site 4. That is, the variable *hba1c* is in the dataset "site1" but is not present in the dataset "site4". In this case Stata adds the new variable but assigns missing values to *hba1c* for people from site 4.

Notice the append command does not require the data to be sorted in any particular way and that you may want to order your data logically after appending different datasets. In this example it might make sense to sort the data by *site* and within site by *pid*. Which happened in this example without sorting because 1) each dataset was already sorted by *pid*, and 2) site4 was appended to site1 preserving the site order.

```
. use site1, clear
. append using site4
. list, sepby(pid site) noobs
```

pid	datest~p	bglevel	site	hba1c
1634	04/22/05	213	1	8.4
1634	04/27/05	117	1	7.3
1701	04/09/08	232	1	8.1
1701	04/09/08	324	1	12.4
1701	04/10/08	250	1	9
1701	04/11/08	213	1	8.4
1525	03/17/06	64	4	.
1525	03/17/06	165	4	.
1525	03/17/06	72	4	.
1525	03/17/06	123	4	.
1683	05/05/08	156	4	.
1683	05/05/08	141	4	.
1683	05/06/08	121	4	.

merge

Sometimes data, especially survey data, are distributed in multiple datasets to keep individual data files sizes smaller. In this situation variables will often be scattered across several datasets. In order to work with information contained in two or more data files it is necessary to merge the variables into a new file.

Typically you will use the data codebook to determine which variables you need, and which datasets contain them. In addition to finding analytically important variables you need to know the name of an identifying variable. This variable is sometimes called a *key* or *ID* variable. Regardless of the name this variable must be unique to a case (observation) in the dataset. For a given data record, the key should be the same across all datasets to allow matching the data from different datasets to the right record. For cross sectional data, this will typically be a single variable, in other cases, two or more variables are needed, this is commonly seen in panel data where subject ID and date or wave are often needed to uniquely identify an observation.

In order for Stata to merge the datasets, the ID variable, or variables, have to have the same name across all files. Additionally, if the variable is a string in one dataset, it must also be a string in all other datasets, and the same is true of numeric variables (the specific storage type is not important, as long as they are numerical). Once you have identified all the variables you need, and know what the ID variable(s) are, you can begin to merge the datasets.

One-to-one match merge

This example was taken from the Stata 10 manual on data management [D] and demonstrates a simple one-to-one merge using a key variable.

The variable *_merge* was created by Stata during the merge and keeps track of where the data in the final dataset come from. When *_merge* equals:

1 obs. from master data
2 obs. from at least two datasets, master or using
3 obs. from only one using dataset

The data storage type for numeric variables is not important because Stata will store the data with sufficient precision so no information will be lost.

```
. use even, clear

. list

     +---------------+
     | number   even |
     |---------------|
  1. |      5     10 |
  2. |      6     12 |
  3. |      7     14 |
  4. |      8     16 |
     +---------------+

.
. use odd, clear

. list

     +--------------+
     | number   odd |
     |--------------|
  1. |      1     1 |
  2. |      2     3 |
  3. |      3     5 |
  4. |      4     7 |
  5. |      5     9 |
     +--------------+

.
. merge 1:1 number using even

    Result                           # of obs.
    -----------------------------------------
    not matched                             7
        from master                         4  (_merge==1)
        from using                          3  (_merge==2)

    matched                                 1  (_merge==3)
    -----------------------------------------

. list

     +-------------------------------------+
     | number   odd   even          _merge |
     |-------------------------------------|
  1. |      1     1      .   master only (1) |
  2. |      2     3      .   master only (1) |
  3. |      3     5      .   master only (1) |
  4. |      4     7      .   master only (1) |
  5. |      5     9     10       matched (3) |
     |-------------------------------------|
  6. |      6     .     12    using only (2) |
  7. |      7     .     14    using only (2) |
  8. |      8     .     16    using only (2) |
     +-------------------------------------+
```

Many to one match-merge

A key variable is used in match merging where observations are joined or merged if the values of the key variable(s) are the same or match. The dataset in memory is called the *master* dataset and the other dataset is called the *using* dataset. An observation is read in the *master* dataset and in the *using* dataset. If the values of the key variable(s) match the observations are joined. If the key values do not match then the data values of the smaller of the two values is merged with missing data for the other, merged variables. Match-merges require that both datasets are sorted by the key variable(s) or that the sort option is specified.

One dataset has measurements of individuals, their sex and number of years of education. The other dataset has household level data—the total family income for three consecutive years. There is also a household ID number (*hhid*) that ties these two datasets together.

There are three records in each dataset, one for each household. Looking at the variable *_merge* we see that the *hhid* variable was matched in each dataset. It is important to look at the frequency distribution for this variable (before dropping it) to be certain the merge worked as anticipated.

.

```
. use ind_data, clear
. list

     +-------------------+
     | hhid   sex   educ |
     |-------------------|
  1. |    2     0     12 |
  2. |    1     1     15 |
  3. |    3     0      9 |
     +-------------------+

. use hh_data, clear
. list

     +-------------------------------------+
     | hhid   faminc96   faminc97   faminc98 |
     |-------------------------------------|
  1. |    3      75000      76000      77000 |
  2. |    1      40000      40500      41000 |
  3. |    2      45000      45400      45800 |
     +-------------------------------------+

.
. use ind_data, clear
. merge 1:1 hhid using hh_data

    Result                           # of obs.
    -----------------------------------------
    not matched                             0
    matched                                 3  (_merge==3)
    -----------------------------------------

. list

     +-------------------------------------------------------------------------+
     | hhid   sex   educ   faminc96   faminc97   faminc98          _merge |
     |-------------------------------------------------------------------------|
  1. |    1     1     15      40000      40500      41000   matched (3) |
  2. |    2     0     12      45000      45400      45800   matched (3) |
  3. |    3     0      9      75000      76000      77000   matched (3) |
     +-------------------------------------------------------------------------+
```

This example is very similar except that there is a duplicate household ID in the individual level dataset representing the presence of a man and a woman in the same household.

This merge is a many-to-1 merge because the individual data (*n*=4, the many) is being merged with the household data (*n*=3, the one).

If the order of the datasets was reversed the merge command would need to reflect this:

```
use hh_data, clear
merge 1:m hhid using ind_data

use ind_data, clear
merge m:1 hhid using hh_data
```

Looking at the variable *_merge* we see that the *hhid* variable was matched in each dataset and each dataset contributed to each new observation.

```
. use ind_data, clear
. list

     +-------------------+
     | hhid   sex   educ |
     |-------------------|
  1. |    2     0     12 |
  2. |    1     1     15 |
  3. |    1     0     14 |
  4. |    3     0      9 |
     +-------------------+

. use hh_data, clear
. list

     +-------------------------------------+
     | hhid   faminc96   faminc97   faminc98 |
     |-------------------------------------|
  1. |    3      75000      76000      77000 |
  2. |    1      40000      40500      41000 |
  3. |    2      45000      45400      45800 |
     +-------------------------------------+

.
. use ind_data, clear
. merge m:1 hhid using hh_data

    Result                           # of obs.
    -----------------------------------------
    not matched                             0
    matched                                 4  (_merge==3)
    -----------------------------------------

. list

     +-------------------------------------------------------------------------+
     | hhid   sex   educ   faminc96   faminc97   faminc98          _merge |
     |-------------------------------------------------------------------------|
  1. |    1     1     15      40000      40500      41000   matched (3) |
  2. |    1     0     14      40000      40500      41000   matched (3) |
  3. |    2     0     12      45000      45400      45800   matched (3) |
  4. |    3     0      9      75000      76000      77000   matched (3) |
     +-------------------------------------------------------------------------+
```

In this example there are more households in the *using* dataset than in the *master* dataset. Household #4 is in the *using* dataset but not in the *master* dataset.

This may not be useful and should probably be dropped. We can detect situations like this by looking at the variable *_merge* which in this case indicates that the fourth record was contributed from the *using* dataset with nothing from the *master* dataset. Once you confirm that this behavior is appropriate you can drop all of the records where *_merge* is equal to 2 (`drop if _merge==2`).

```
. use ind_data, clear

. list

     +-------------------+
     | hhid   sex   educ |
     |-------------------|
  1. |    2     0     12 |
  2. |    1     1     15 |
  3. |    3     0      9 |
     +-------------------+

. use hh_data, clear

. list

     +-------------------------------------+
     | hhid   faminc96   faminc97   faminc98 |
     |-------------------------------------|
  1. |    3      75000      76000      77000 |
  2. |    1      40000      40500      41000 |
  3. |    2      45000      45400      45800 |
  4. |    4      35000      37400      42100 |
     +-------------------------------------+

.
. use ind_data, clear

. merge m:1 hhid using hh_data

    Result                           # of obs.
    -----------------------------------------
    not matched                             1
        from master                         0  (_merge==1)
        from using                          1  (_merge==2)

    matched                                 3  (_merge==3)
    -----------------------------------------

. list

     +-------------------------------------------------------------+
     | hhid   sex   educ   faminc96   faminc97   faminc98   _merge |
     |-------------------------------------------------------------|
  1. |    1     1     15      40000      40500      41000   matched (3) |
  2. |    2     0     12      45000      45400      45800   matched (3) |
  3. |    3     0      9      75000      76000      77000   matched (3) |
  4. |    4     .      .      35000      37400      42100   using only (2) |
     +-------------------------------------------------------------+
```

Finally, here is a situation where there is a household ID in *master* but not in *using* as well as a household ID in *using* but not in *master*. Both of these records are detected using the variable *_merge*.

These records are probably not useful and should probably be dropped. As before, once you confirm that this behavior is appropriate you can drop all of the records where *_merge* is equal to 1 or 2 (`drop if _merge==1 | _merge==2`).

```
. use ind_data, clear

. list

     +-------------------+
     | hhid   sex   educ |
     |-------------------|
  1. |    2     0     12 |
  2. |    1     1     15 |
  3. |    3     0      9 |
  4. |    5     0     10 |
     +-------------------+

. use hh_data, clear

. list

     +-------------------------------------+
     | hhid   faminc96   faminc97   faminc98 |
     |-------------------------------------|
  1. |    3      75000      76000      77000 |
  2. |    1      40000      40500      41000 |
  3. |    2      45000      45400      45800 |
  4. |    4      35000      37400      42100 |
     +-------------------------------------+

.
. use ind_data, clear

. merge 1:1 hhid using hh_data

    Result                           # of obs.
    -----------------------------------------
    not matched                             2
        from master                         1  (_merge==1)
        from using                          1  (_merge==2)

    matched                                 3  (_merge==3)
    -----------------------------------------

. sort hhid

. list

     +-------------------------------------------------------------+
     | hhid   sex   educ   faminc96   faminc97   faminc98   _merge |
     |-------------------------------------------------------------|
  1. |    1     1     15      40000      40500      41000   matched (3) |
  2. |    2     0     12      45000      45400      45800   matched (3) |
  3. |    3     0      9      75000      76000      77000   matched (3) |
  4. |    4     .      .      35000      37400      42100   using only (2) |
  5. |    5     0     10          .          .          .   master only (1) |
     +-------------------------------------------------------------+
```

The *_merge* variables

The *_merge* variable(s) created by the merge command are easy to miss, but are very important. As discussed above, they tell us which dataset(s) each case came from. This is important because a lot of values that came from only one dataset may suggest a problem in the merge process. However, it is not uncommon for some cases to be in one dataset, but not another. In panel data this can occur when a given respondent did not participate in all the waves of the study. It can also occur for a number of other reasons. For example, a female respondent might appear in the subset of the data with demographic information, but be completely absent from the subset of data with information on female respondents' children, because she does not have children.

Because cases that are not present in all datasets are not necessarily a problem, in order for the information in *_merge* variables to be useful you need to know what to expect if the datasets merged correctly. Having too many, or all, of the cases in your merged dataset come from one, or only a few of the datasets you've merged is a sign that the ID variable does not match correctly across datasets.

Once we have examined and sorted the datasets we can merge them. The syntax below does this, note that the command is the same as in the first example. By default, Stata will allow cases to come from any of the three datasets. There are options that will allow you to control which datasets the cases come from, you can find out about them by typing "help merge" (without the quotes) in Stata.

Using `collapse` with `merge`

The `collapse` command is very powerful and creates a new dataset of summary or aggregate statistics. But, it can be more useful when used with the `merge` command. In this example we will use a dataset with measurements on 3,141 counties to create a dataset with an aggregate measurement at the state level (*n*=51). Further, we will merge this aggregate data back into the county level dataset.

Here we create a new dataset (statetemp.dta) that contains one record per state with a variable called *statecrim* which is the average of the crime rates from all of the counties.

The key variable here is *state*, a numeric code unique to each state. This variable ties the two datasets together.

This dataset can be used to examine county level crime within the context of the overall state average crime rate.

```
use SOCY401-Neustadtl-County-Crime.dta, clear

preserve
  collapse statecrim=crimerate04, by(state)
  sort state
  save statetemp, replace
  list
restore

merge m:1 state using statetemp
erase statetemp.dta
```

The `preserve` and `restore` commands deal with the programming problem where the data must be changed to achieve the desired result (e.g. using `collapse`) but, when the program concludes, you want to undo the damage done to the data. See `help preserve` for more details.

Exercise

1. The Stata data three files *md.dta, dc.dta,* and *va.dta* county-level demographic data for these states. Create a new, combined Stata data file called *merged-md-dc-va.dta* by appending these data files.

2. Six Stata files (*mdpop00.dta* through *mdpop05.dta*) contain county-level population variables for each year. That is, the 2000 population variable for Maryland is in the file *mdpop00.dta*, the 2001 data is in *mdpop01.dta*, and so forth. The Maryland data files have twenty-four observations (i.e. twenty-four counties).

   Merge these six files so that the resulting dataset has one variable for each year (you could use this dataset to examine trends). Save the new dataset as *merged-md00-05*. The data should look like this:

```
Contains data from mdpop00.dta
  obs:            24                          County Characteristics, 2000-2007, Dataset 0001
 vars:            10                          3 Mar 2010 11:07
 size:           888 (99.9% of memory free)
-------------------------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------------------------
county          int    %8.0g                  County FIPS Code
fips            long   %12.0g                 State and county FIPS Code
statename       byte   %20.0g      statenametmp
                                               State name
countyname      int    %43.0g      countynametmp
                                               County name
pop00           long   %12.0g                 Total resident population, 7/1/00
pop01           long   %12.0g                 Total resident population, 7/1/01
pop02           long   %12.0g                 Total resident population, 7/1/02
pop03           long   %12.0g                 Total resident population, 7/1/03
pop04           long   %12.0g                 Total resident population, 7/1/04
pop05           long   %12.0g                 Total resident population, 7/1/05
-------------------------------------------------------------------------------------------------
Sorted by:  county
     Note:  dataset has changed since last saved
```

3. The Stata dataset *countybirths.dta* contains birth data for each county in the United States (*n*=3,141). The dataset *merged-md-dc-va.dta* that you created earlier contains population data for the 159 counties in Maryland, the District of Columbia, and Virginia.

   Create a new dataset called *merged-births-deaths.dta* that has both the population and birth data for the 159 counties by merging the two data files.