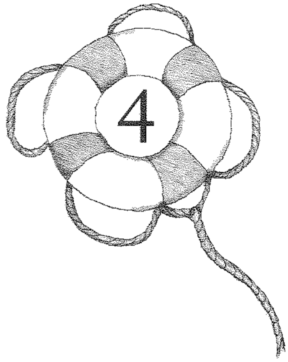


Manipulating data

SORTING DATA

Often you need to sort data; that is, to organize the cases or rows according to the categories of one or more variables. There are a number of reasons for wanting to do this, the main ones being: (1) preparing data to be merged with other data sets (more on that later); and (2) if you want to produce statistics separately for different groups – men and women or different countries, for example.

To sort your data, type **sort** followed by the variable or variables you want to sort by. If you want to sort by sex:

```
sort sex
```

The command will sort the data by the categories of the variable *sex* from the lowest to the highest. Now if you want to know the descriptive statistics of some variables by *sex*, you can examine them by using the prefix command **by**. A colon (**:**) must follow the variable or variables by which you sorted (and by which you want to have your output organized). For example, if you wanted the mean age for men and women you would use the command:

```
by sex: su age
```

If you have not sorted your data and try to use the **by** command then Stata will return the error message (in red): `not sorted`.

There is more detail on the command **su** to produce descriptive data and alternative commands in Chapter 5.

In the latest versions of Stata, the **by** and **sort** prefix commands can be combined into one prefix command, **bysort**, which can be used when organizing results by groups. For the same example of the mean ages of men and women you could use

bysort sex:su age

As with the **by** command, it is necessary to place a colon (:) after the variable(s) to sort on.

```
. bysort sex:su age

---------------------------------------------------------------
-> sex = male

    Variable |     Obs        Mean    Std. Dev.    Min    Max
-------------+-----------------------------------------------
         age |    4833    43.37099    17.98608     16     94

---------------------------------------------------------------
-> sex = female

    Variable |     Obs        Mean    Std. Dev.    Min    Max
-------------+-----------------------------------------------
         age |    5431     45.5511    18.82718     16     97
```

The **bysort** command can also be used with as many variables as you need to create the subgroups you are interested in. The **bysort** variable(s) should generally refer to a categorical variable or variables. For example, if you were interested in extending the previous example and wanted to know the mean ages of men and women but further broken down by their marital status you would use

bysort sex mastat: su age

The order of the two variables after **bysort** determines how the output is presented. The above command produces the (partial) results shown below with the mean ages of men by their marital status; this would be followed by the mean ages of women by their marital status. This is because in the variable *sex*, men are coded 1 and women coded 2.

```
. bysort sex mastat:su age

---------------------------------------------------------------
-> sex = male, mastat = married

    Variable |     Obs        Mean    Std. Dev.    Min    Max
-------------+-----------------------------------------------
         age |    2947    48.60333    15.08131     18     94
```

```
------------------------------------------------------------
-> sex = male, mastat = living a

    Variable |     Obs        Mean    Std. Dev.     Min     Max
-------------+-------------------------------------------------
         age |     334    33.45808    12.36032      17      91

------------------------------------------------------------
-> sex = male, mastat = widowed

    Variable |     Obs        Mean    Std. Dev.     Min     Max
-------------+-------------------------------------------------
         age |     169    73.70414    10.53543      35      91

------------------------------------------------------------
-> sex = male, mastat = divorced

    Variable |     Obs        Mean    Std. Dev.     Min     Max
-------------+-------------------------------------------------
         age |     150    47.88667    12.97056      23      82
etc
```

If you wanted to organize your results so that you could more easily compare the mean ages of men and women within each category of marital status, then you may find it better to use

bysort mastat sex: su age

This produces the (partial) results shown below where you can see that the mean ages for both men and women are shown for those who are married, cohabiting, etc.

```
. bysort mastat sex :su age

------------------------------------------------------------
-> mastat = married, sex = male
    Variable |     Obs        Mean    Std. Dev.     Min     Max
-------------+-------------------------------------------------
         age |    2947    48.60333    15.08131      18      94

------------------------------------------------------------
-> mastat = married, sex = female
    Variable |     Obs        Mean    Std. Dev.     Min     Max
-------------+-------------------------------------------------
         age |    3062    45.84651    14.68746      18      89
etc
```

The command after **bysort mastat sex:** could be a range of operations that we haven't covered yet. It is possible to have separate estimations for groups you have specified using many statistical functions, such as correlations, regressions, and *t*-tests, just to name a few. But it is worth noting that using the **bysort** command does mean some restrictions on other commands you can use.

MERGING AND APPENDING DATA

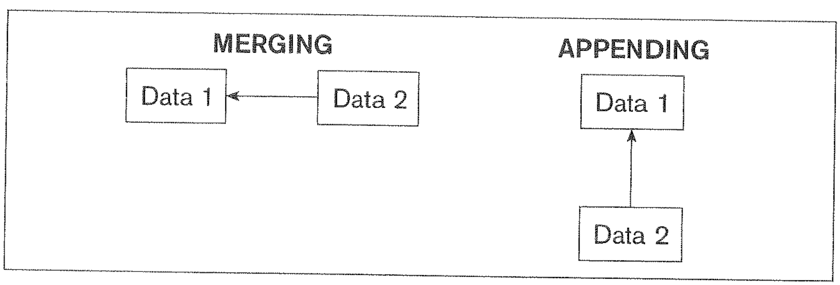
There may be times when you will want to join two or more data files with one another. There are two main ways – merging and appending. Figure 4.1 illustrates the basic difference between them. Merging adds one file horizontally to the right of another file in the spreadsheet, and appending adds one file vertically to the bottom of another file. The data file that is already open in Stata is referred to as the *master* data and the data file to be added is referred to as the *using* data. In both the **merge** and **append** commands you will specify the data to be added by **using data.dta** part of the command.

For example, you might need to merge your data files if you have follow-up data on the same set of respondents, and this would add more variables to the rows containing the master data. If your study was collecting data from two or more locations where the data are entered then you might need to append all the data files together into one large data file.

Merging data

The **merge** command is best illustrated with an example. Here, we have a small data file from 2004 (called 2004data.dta) that contains information on five people's age (*age04*), income in units of 10,000 (*inc04*) and marital status (*mstat04*); each person has an identifying number (*id*).

Figure 4.1
Merging and appending data



id	*age04*	*inc04*	*mstat04*
1	25	55	married
2	24	66	single
3	23	45	divorced
4	24	27	married
5	32	100	cohabiting

You have data on age, income and marital status from the same five people in 2005 (called 2005data.dta) and want to merge these data with the 2004 data.

id	*age05*	*inc05*	*mstat05*
1	26	57	married
2	25	78	single
3	24	32	divorced
4	25	59	divorced
5	33	200	married

You would end up with a file that was organized like this:

id	*age04*	*inc04*	*mstat04*	*age05*	*inc05*	*mstat05*
1	25	55	married	26	57	married
2	24	66	single	25	78	single
3	23	45	divorced	24	32	divorced
4	24	27	married	25	59	divorced
5	32	100	cohabiting	33	200	married

The data from the year 2005 is added to the right of the 2004 data for all cases. Another thing to note when you merge files is the naming of the variables. If the variables in both the 2004 and 2005 data had simply been called *age*, *inc* and *mstat*, then when you attempted to merge them Stata would not return an error message but use the values for the variable from the master data! So you must ensure that your variables have unique names before you merge data files.

To do this step-by-step proceed as follows:

1. Make sure both data files are sorted on the variable you wish to merge by – in this example, the *id* variable is used as it is common to both data files. If you regularly merge data then we recommend that you get into the habit of sorting on the merge variable before you save your data. This way you know the data is sorted and then can omit the first stage of the do file example below.
2. Open the data file that is to be the master data – in this case the 2004 data.
3. Merge the 2005 data on to the 2004 data.
4. Save the new data file under a new name.

The do file to do this would look like this:

```
cd datafolder                 /* set default folder */

use 2005data,clear            /* open 2005 data */
sort id                       /* sort by id ready
                                 to merge */
save 2005data,replace         /* save sorted 2005
                                 data */

use 2004data,clear            /* open 2004 data */
sort id                       /* sort by id ready
                                 to merge */
merge id using 2005data       /* merge on 2005 data
                                 by id */

sort id                       /* ensure sorted by
                                 id */
save newdata,replace          /* save as new file */
```

Alternatively, you can use the **sort** option in the **merge** command which means the do file would be:

```
cd datafolder
use 2004data,clear
merge id using 2005data,sort
save newdata,replace
```

Rarely is real-world data as straightforward as the above example, and it is more than likely that any follow-up data will have respondents who have dropped out of the study. To add a further twist, it is also possible that new people have entered the study. This is the case in many of the household-based surveys when a child reaches a certain age to be included in the study. Stata produces a variable called *_merge* that will help you determine how many cases have dropped out, entered or remained in the data.

To extend the previous example, suppose you now have two sets of data – one from 2001 and one from 2002 – with the same variables as before, but not all of the people are in both years. The 2001 data are:

id	*age01*	*inc01*	*mstat01*
1	35	45	married
2	24	66	single
3	28	25	divorced
4	24	27	married
5	32	100	cohabiting
6	42	35	married
7	26	14	single
8	38	23	married
9	40	85	cohabiting
10	44	27	divorced

The 2002 data are:

id	*age02*	*inc02*	*mstat02*
1	36	57	married
3	29	32	divorced
4	25	59	divorced
5	33	200	married
7	27	14	single
8	39	23	married
9	41	85	cohabiting
11	18	10	single
12	21	15	single

If you merged these two files you would end up with a file that was organized like this:

id	*age01*	*inc01*	*mstat01*	*age02*	*inc02*	*mstat02*
1	35	45	married	36	57	married
2	24	66	single	.	.	.
3	28	25	divorced	29	32	divorced
4	24	27	married	25	25	divorced
5	32	100	cohabiting	33	120	married
6	42	35	married	.	.	.
7	26	14	single	27	14	single
8	38	23	married	39	24	married
9	40	85	cohabiting	41	80	cohabiting
10	44	27	divorced	.	.	.
11	.	.	.	18	10	single
12	.	.	.	21	15	single

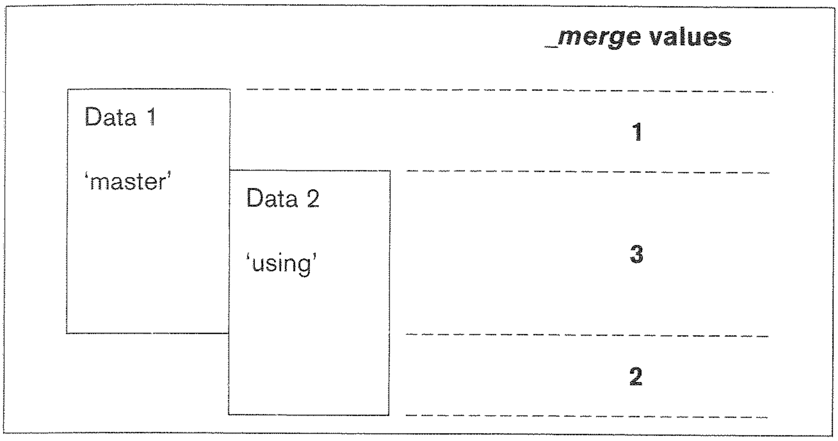


Figure 4.2 The *_merge* variable

In this case, you can make use of the new variable *_merge* which will have been created and placed at the end of your data file after merging the two data files – the bottom of your list of variables. The variable *_merge* is very important. It gives you information on the success of your merge. There are three codes associated with *_merge* (see Figure 4.2):

_merge = 1, observations/cases from your master data;
_merge = 2, observations/cases from your using data;
_merge = 3, observations/cases from both your master and using data.

You should be very careful with merges that do not equal 3. Those coded 1 in this case refer to cases lost from 2001 to 2002, while those coded 2 refer to cases that appear only in 2002 and not in 2001. Merges that equal 3 mean that the observation/case was present in both years.

The do file for this merge would look like this:

```
cd datafolder                /* set default folder */

use 2001data,clear           /* open 2001 data */
merge id using 2002data,sort    /* merge on 2002
                                   data by id */

ta _merge                    /* inspect _merge
                                variable */
drop _merge                  /* drop _merge not
                                needed? */

sort id                      /* ensure sorted by id */
save newdata,replace         /* save as new file */
```

If you run the above do file, part of your results should look like this:

```
. ta _merge

     _merge |      Freq.     percent        Cum.
------------+-----------------------------------
          1 |          3       25.00       25.00
          2 |          2       16.67       41.67
          3 |          7       58.33      100.00
------------+-----------------------------------
      Total |         12      100.00
```

Seven cases were in both the master and using data (*_merge* = 3), 3 cases were in the master data only (*_merge* = 1), and 2 cases were in the using data only (*_merge* = 2). We recommend that you tabulate the *_merge* variable to check your merges, especially when you first attempt this type of data manipulation. The **merge** command has a number of options that allow you to keep only the cases for some of the *_merge* values, but to start with you should view the tabulated results and then decide if you wish to delete any of the cases. Remember to drop the *_merge* variable if it is no longer needed or rename it if you need to keep it but have other merges to perform, because if you don't Stata will return an error saying that the *_merge* variable already exists.

In these simple examples we have only used one variable (*id*) to uniquely identify cases in both data sets, but the **merge** command can specify more than one variable if a combination of variables is what uniquely identifies each case (which is often the case in large-scale data sets). For example:

```
merge id_1 id_2 id_3 using otherdata,sort
```

Using the **sort** option in this way implies that the matching variables (*id_1*, *id_2* and *id_3*) uniquely identify the same cases in both data sets. There are other options to the **merge** command when this situation doesn't apply, but these are quite advanced techniques.

Merging data files of different levels (hierarchical data structures)

Merging files of different levels is quite common when, for example, you have data from individuals (in an individual level data file) and data from the households in which the individuals live (in a household level file). Another example would be an individual level data file on people's experiences of and attitudes to crime and data at neighbourhood level on crime rates and police efficiency. Again, this process is best illustrated with an example. In this one you have an individual level data file (each row is data from a person) and a household level data file (each row refers to data about the household).

In the individual level file (individual.dta), there is a person identifier (*id*) which uniquely identifies each individual and a household identifier (*hid*) which uniquely identifies each household; as you can see, most households have more than one person. In these data, three people live in household 1, two people in household 2, one person in household 3, one person in household 4, and three people in household 5.

id	*hid*	*age*	*inc*	*mstat*
1	1	25	55	married
2	1	24	66	single
3	1	23	45	divorced
4	2	24	27	married
5	2	32	100	married
6	3	54	78	single
7	4	33	0	single
8	5	21	74	single
9	5	33	0	single
10	5	21	77	single

The household level file (household.dta) has information on household size and region of the country. The variable that is common to the individual level file and the household level file is *hid*. In order to match data from the household level file onto the individual level file, you would need to make sure that a common identifier is present in each file type.

hid	*hhsize*	*region*
1	3	south
2	2	north
3	1	east
4	1	west
5	3	north

When you merge these files by their common identifier (*hid*) you will end up with a file that has the household characteristics merged on to each individual:

id	*hid*	*age*	*inc*	*mstat*	*hhsize*	*region*
1	1	25	55	married	3	south
2	1	24	66	single	3	south
3	1	23	45	divorced	3	south
4	2	24	27	married	2	north
5	2	32	100	married	2	north
6	3	54	78	single	1	east
7	4	33	0	single	1	west
8	5	21	74	single	3	north
9	5	33	0	single	3	north
10	5	21	77	single	3	north

The do file for this merge would look similar to the one for merging the individual data files, but this time sorting on *hid* as it is the common variable to merge on.

To have Stata merge on multiple rows of the master data, the commands **merge** or **joinby** can be used. Both commands tell Stata to match on all possible pairs between the master and using data on the common variable(s) (in this case *hid*) and give pretty much the same results. The **joinby** command, however, does not automatically generate the *_merge* variable as with the **merge** command. This is because Stata only matches the possible rows of data. If you specify the option **unmatched(both)** (shortened to **unm(b)** below) to the **joinby** command then a *_merge* variable is generated with the same features as with the **merge** command. There are a number of other options to the **joinby** command for more complex matching situations.

```
cd datafolder                      /*set default folder*/

use household,clear                /*open household
                                     data*/
sort hid                           /*sort by hid*/
save household,replace             /*save sorted data*/
use individual,clear               /*open individual
                                     data*/
sort hid                           /*sort by hid for
                                     merge*/
joinby hid using household,unm(b)  /*join hhold
                                          data by hid*/
ta _merge                          /*inspect _merge*/
drop _merge                        /*drop _merge*/

sort hid id                        /*sort by hid and id*/
save newdata,replace               /*save as new file*/
```

In this example, all cases will have a value of 3 on the *_merge* variable, but often that is not the case in real-world data. Some respondents may have provided individual data but not household data. In this case, they would be *_merge* = 1 as they are in the master data (the individual level file) but not in the using file (the household level file). Conversely, if there is household data but no matching individual level data then these households would be *_merge* = 2 as they are in the using data but not in the master data.

For merging housing and individual data using the **merge** command, see the demonstration exercise later in this chapter.

Appending data

The **append** command is also best illustrated with an example. In this example you have collected a small set of data from town A (called townAdata.dta) and another researcher has collected similar data from town B (called townBdata.dta). Both sets of data contain information on five people's year of birth (*yob*), employment status (*empstat*) and income in units of 10,000 (*inc*), and each person has an identifying number (*id*). Here are the town A data:

id	*yob*	*empstat*	*inc*
101	1968	employed	45
102	1973	not working	66
103	1974	employed	45
104	1980	student	14
105	1963	employed	80

Here are the town B data:

id	*yob*	*empstat*	*inc*
201	1977	employed	57
202	1960	employed	78
203	1982	not working	32
204	1975	not working	59
205	1968	employed	91

There are three issues to deal with before you can append these data.

1. You need to make sure that the person identifier variable (*id*) has different values in both sets of data. It would have been easy for both data collectors to use the numbers 1 to 5. In

this example, the people in town A have identifiers 101 to 105 and those in town B have 201 to 205.

2. You need to make sure that the variable names are identical in both sets of data, otherwise Stata will treat them as different variables and put the data in different columns (see Box 4.1).
3. You should consider adding a new variable to each set of data that indicates what town the people come from. In each set of data this technically will not be a variable as the values are constant for all people in the data, but after the data files are appended it will vary depending on which town the people are from.

Box 4.1: Variable names when appending data

If the variable names, for the same variable, are different in the data sets then Stata treats them as different variables and puts them in different columns. For example, if income is called *inc* in the town A data and *income* in town B data, then after you append the data it will look like this:

id	*yob*	*empstat*	*inc*	*income*	*town*
101	1968	employed	45	.	1
102	1973	not working	66	.	1
103	1974	employed	45	.	1
104	1980	student	14	.	1
105	1963	employed	80	.	1
201	1977	employed	.	57	2
202	1960	employed	.	78	2
203	1982	not working	.	32	2
204	1975	not working	.	59	2
205	1968	employed	.	91	2

After you create a new variable for the town (where 1 = town A and 2 = town B) and append the sets of data you end up with a file like this:

id	*yob*	*empstat*	*inc*	*town*
101	1968	employed	45	1
102	1973	not working	66	1
103	1974	employed	45	1
104	1980	student	14	1
105	1963	employed	80	1
201	1977	employed	57	2
202	1960	employed	78	2
203	1982	not working	32	2
204	1975	not working	59	2
205	1968	employed	91	2

The do file to append these sets of data would look like:

```
cd datafolder                 /* set default
                                 folder */

use townBdata,clear           /* open town B data */
gen town=2                    /* new variable
                                 town */
save townBdata,replace        /* save data */

use townAdata,clear           /* open individual
                                 data */
gen town=1                    /* new variable
                                 town */
append using townBdata        /* append town B
                                 data */
sort id                       /* ensure sorted
                                 by id */
save newdata,replace          /* save as new file */
```

LONGITUDINAL DATA

Longitudinal data comes from surveys or studies that have collected information from the same source on more that one occasion. This type of data is also regularly referred to in the social sciences as 'panel' data because the data is observational rather than experimental. The definitions of 'longitudinal' and 'panel' data are often blurred at the edges and sometimes the terms are used interchangeably as the surveys tend to be more complex than a simple panel. For a fuller discussion on types of longitudinal data, see Frees (2004), Singer and Willett (2003) and Wooldridge (2002).

One important concept to emphasize, before we tackle data manipulation, is the difference between balanced and unbalanced panel data. In a balanced panel, there are data at all points of time for all individuals (or other source of data) in the panel. In an unbalanced panel, the number of individuals at each point of time may change as some drop out or some new people enter the study. We have illustrated this in Figure 4.3. In the top half of the figure we show a balanced panel of five people who are present at all three data collection points. In the lower half we show an unbalanced panel that starts with five people at the first time point. By the time of the second data collection there are four left in the

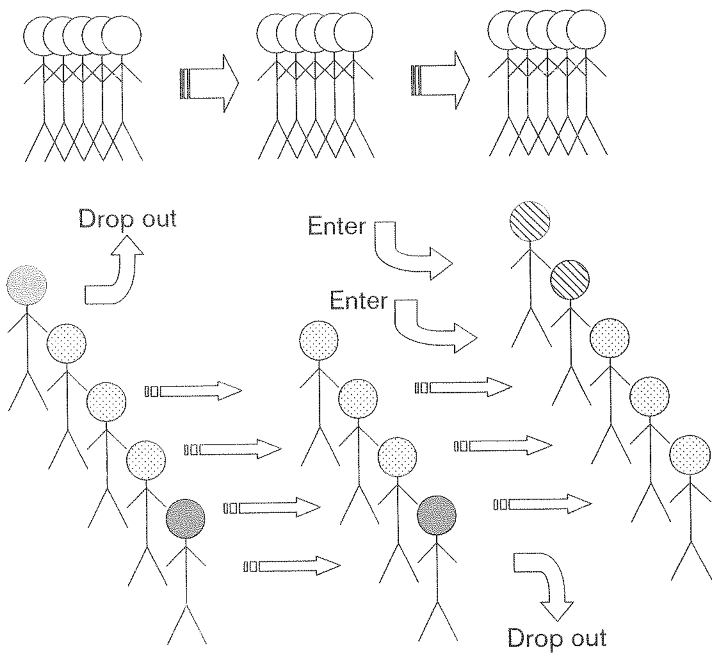


Figure 4.3 Balanced and unbalanced panels

panel with one person (shaded light grey) dropping out of the study. At the third data collection, there are again five people in the study but only three are original members (dotted) as one more person (shaded dark grey) has dropped out but two new people (stripes) have joined the study.

Most large longitudinal studies have fairly complex rules about who stays in, leaves or enters the study as time goes on. These are sometimes known as 'following rules' and need to be thoroughly understood before using the data for any project. As a general rule, panel studies based on families or households have more complex following rules than those based on a cohort with some common characteristic, such as a birth cohort that are born in the same week. The dynamics of modern family life – partnering, marriage, births, deaths, separation and divorce – naturally require more complex following rules than a tightly defined cohort of individuals. However, even what may appear to be straightforward cohort studies have developed into far more complex studies.

Longitudinal data may come from a variety of data sources. We often see graphs or charts of a country's economic indicators such as gross national product, unemployment or public spending for a number of years to show a trend over time. These are longitudinal data, and a data set could very well contain similar information from a number of countries. Similarly, within a given country, data over time could be collected from county or state level, hospitals, schools or police forces.

Longitudinal data need some consideration to enable them to be used for analysis. This primarily rests with the structure of the data required by Stata. For many types of analysis, Stata requires the data to be in unit/time format (although some types of analysis can be done in wide format). Unit is the source of the data – person, country, etc. – and time is the interval between data collection. So, for example, for the individual data from the British Household Panel Survey (collected yearly) the data format would be person/year and for monthly country level data it would be country/month format. In the first example, each row in the data spreadsheet would contain data from one person for one year of data collection. Therefore, each person has as many rows of data as the number of years they have been in the survey.

In the example data below there are four people, with *id* = 001, 002, 003 and 004. Person 001 has three years' worth of data, which can be seen from the identifier appearing in three rows of data, and the time variable (*year*) shows that these data are for 2001, 2002 and 2003. Person 002 has only one year's data, for 2002. Person 003 has five years' data, but with data missing for 2002. Person 004 has three years' data, but not from consecutive years.

id	*year*	*age*	*mstat*
001	2001	16	single
001	2002	17	single
001	2003	18	single
002	2002	67	widowed
003	1999	27	single
003	2000	28	married
003	2001	29	married
003	2003	31	separated
003	2004	32	separated
004	1999	44	married
004	2002	47	married
004	2004	49	divorced

One important thing to note when your data are organized in person/year (or more generally unit/time) format is that the *id* number does not uniquely identify a particular row of data. It is the combination of the *id* and the *year* variable that is unique to each row of data.

When your data are arranged in person/year format it is possible to easily see the changes to other variables over time. In the above example, you can see that person 001 is 16 years old in 2001 and is single. They stay in the study for two more years, 2002 and 2003, as they age to 17 and 18 years old and they stay single at all three time points. It is worth noting here that in real survey data the age variable may not increase as uniformly as we have shown here, as survey dates may differ from year to year. Person 003 stays in the study for five time points from 1999 to 2004, but data from 2002 is missing. In that time they age from 27 to 32 and their marital status (*mstat*) shows that they married between the 1999 and 2000 interviews, then separated between the 2001 and 2003 interviews.

Stata has a number of features that make it easy to manipulate longitudinal data and to gain insights into your data structure. Here, we demonstrate some of the basic features you may need to start getting a handle on manipulating longitudinal data. Stata contains many more complex features for use with longitudinal data which can be employed when you are comfortable with the basic techniques. More details on these advanced techniques can be found in Rabe-Hesketh and Skrondal (2008).

`_n` AND `_N`

These two features are commonly used in conjunction with a **`generate`** command to give you some information about the structure of your data. They will help you answer questions such as 'How many people are in the study at all time points?' and 'What is the distribution of the number of times people are in the study?'

To start, the data from each time point need to be appended to each other with a time variable specified similar to that described earlier in the chapter with the example for the town A and town B data, but instead of generating the *town* variable you need to generate a time variable. To illustrate the use of **`_n`** and **`_N`** we will continue with the example data described immediately above. Before the **`_n`** and **`_N`** features are used to generate new variables, the data must be sorted on the *id* and *year* variables.

We then use the **`generate`** command and the **`_n`** and **`_N`** features to make two new variables (*seq* and *tot*) in the data:

```
sort id year
by id: gen seq = _n
by id: gen tot = _N
```

These commands would produce the following results:

id	*year*	*age*	*mstat*	*seq*	*tot*
001	2001	16	single	1	3
001	2002	17	single	2	3
001	2003	18	single	3	3
002	2002	67	widowed	1	1
003	1999	27	single	1	5
003	2000	28	married	2	5
003	2001	29	married	3	5
003	2003	31	separated	4	5
003	2004	32	separated	5	5
004	1999	44	married	1	3
004	2002	47	married	2	3
004	2004	49	divorced	3	3

Using combinations of the *year*, *seq* and *tot* variables, you can find out some core information about your data. For example, from the above small data set, you can **tabulate** the *year* variable conditional on the *seq* variable equalling 1 and you will get the number of people by their first year in the study.

ta year if seq==1

```
. ta year if seq==1

       year |      Freq.     Percent        Cum.
------------+-----------------------------------
       1999 |          2       50.00       50.00
       2001 |          1       25.00       75.00
       2002 |          1       25.00      100.00
------------+-----------------------------------
      Total |          4      100.00
```

This shows that two people were first observed in 1999 then one each in 2001 and 2002. Other combinations can tell you the distribution of the individuals' number of years in the study:

ta tot if seq==1

```
. ta tot if seq==1

        tot |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          1       25.00       25.00
          3 |          2       50.00       75.00
          5 |          1       25.00      100.00
------------+-----------------------------------
      Total |          4      100.00
```

This shows that one person was in the study at only one time point, two people were observed three times and one person was observed five times.

The following command identifies the last year in the study for each person:

ta year if seq==tot

```
. ta year if seq==tot

       year |      Freq.     Percent        Cum.
------------+-----------------------------------
       2002 |          1       25.00       25.00
       2003 |          1       25.00       50.00
       2004 |          2       50.00      100.00
------------+-----------------------------------
      Total |          4      100.00
```

The last year in the study for each person is identified when *seq* = *tot*, so this tells you that one person was last observed in 2002, another in 2003 and the other two people last observed in 2004.

While all this information is useful, none of it tells us what years the people were in the study between their first and last observation. There is a potential maximum of six observations in these data as the earliest date is 1999 and the latest is 2004, but no one was observed six times. One person (*id* 003) was observed five times, but is missing for 2002. There are ways of using the *year*

Box 4.2: Long and wide files

Longitudinal data or data with repeated measures need to be organized in a way that allows Stata to compute the appropriate tests some of which we cover in Chapter 7. The most common way of referring to the two main types of data organization is as 'long' and 'wide' files.

Long files are where the data are 'stacked', usually constructed by using the **append** command. In this way each row contains data from a person at a particular time point. In panel data this could mean each row represents a person/year but in data from a pre- and post-test experiment each row would represent a person/test. Then the number of rows for each person matches the number of times they have been interviewed or tested.

Wide files are where each row contains all the variables for that person. For example, the data may be a set of variables collected in 2002 then to the right of them would be variables from 2003 and so on. So each person has a row and the number of variables depends on the number of times interviewed or tested. Wide files are usually constructed using the **merge** command.

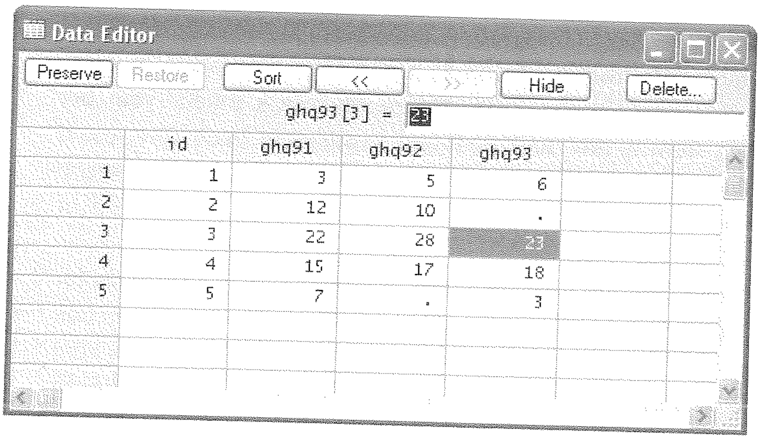
variable in conjunction with the **_n** and **_N** functions to identify these breaks in observations as Stata can look within the rows with the same *id* value to see if numbers are sequential or not.

However, these techniques are beyond the scope of this book and we just want to draw your attention to them and to some other capabilities that you may want to progress on to. The comparative ease of handling longitudinal data is one of the main reasons why many people change to using Stata. If you too take the route to longitudinal data analysis then there are many books available; we would also suggest taking a course as some of the issues are best worked out in the classroom as they can take a little time to get your head around.

MORE ADVANCED DATA HANDLING

Two other more advanced data handling commands to be aware of are **reshape** and **expand**. These are both very powerful commands. Very briefly, **reshape** allows you to change data from wide to long format and vice versa. The most common use is to change wide files into long files as Stata has far more capability using long files. You need to invest some time in ensuring the data and variables are correctly formatted, because then the command is rather simple.

For example, we have a small data set as seen in the Data Editor:



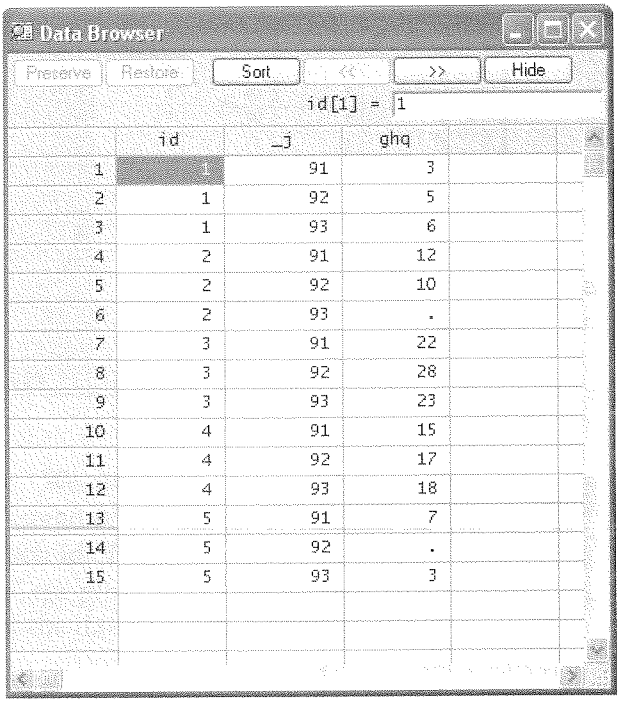
There is the *id* variable and then three variables *ghq91*, *ghq92* and *ghq93*. This is the format of the variable names that Stata needs to perform the **reshape** command; a common prefix and a numeric suffix. Stata will interpret this as the same measure (*ghq*) taken at times 91, 92 and 93. Then we use the **reshape** command and specify we want these data changed to long format. After **long** we put the common variable prefix, *ghq*, then after the comma tell Stata that the identifier variable is *id*.

```
reshape long ghq,i(id)
```

```
. reshape long ghq,i(id)
(note: j = 91 92 93)

Data                               wide   ->   long
-----------------------------------------------------
Number of obs.                        5   ->     15
Number of variables                   4   ->      3
j variable (3 values)                     ->     _j
xij variables:
                        ghq91 ghq92 ghq93  ->    ghq
```

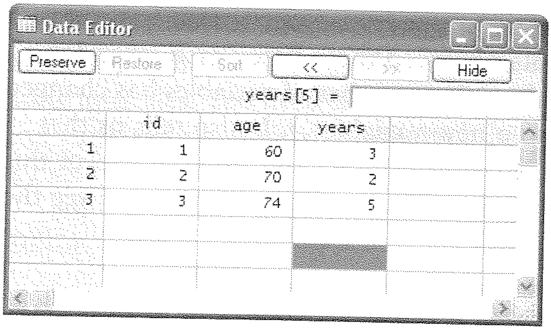
The output tells us that the number of observations has changed from 5 to 15: five people observed three times each, even taking into account missing values. The number of variables has changed from 4 to 3: *id*, *ghq* and a new variable *_j*. It also tells us that *_j* has three values. This is the 'time' variable, and we expect this to be 91, 92 and 93. If we look in the Data Browser we see the following:



	id	_j	ghq
1	1	91	3
2	1	92	5
3	1	93	6
4	2	91	12
5	2	92	10
6	2	93	.
7	3	91	22
8	3	92	28
9	3	93	23
10	4	91	15
11	4	92	17
12	4	93	18
13	5	91	7
14	5	92	.
15	5	93	3

Obviously these reshaping techniques can get very complicated, but at this stage we would just like you to be aware of some of Stata's capabilities. Compare **reshape** with the **xpose** command that we mention in Chapter 7.

The other command is **expand**. This command tells Stata to add rows of data for each case so that there are as many rows for each case as in the specified variable. The specified variable is usually a time variable so that after the expansion each row represents a time point. In this simple example we use a variable *years*. The first case (*id*=1) has a value of 3 in years, so after the expansion there will be three rows of data. If the value of *years* was 1, less than 1 or missing then only the original single row of data would remain.



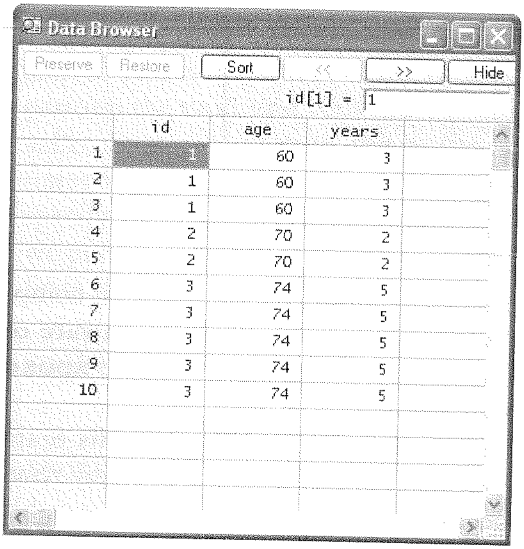
	id	age	years
1	1	60	3
2	2	70	2
3	3	74	5

The expansion is done by:

```
expand years
sort id
```

```
. expand years
(7 observations created)
```

Now the data look like this:

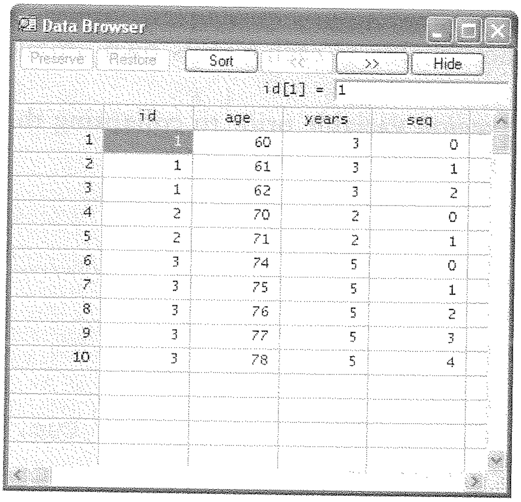


	id	age	years
1	1	60	3
2	1	60	3
3	1	60	3
4	2	70	2
5	2	70	2
6	3	74	5
7	3	74	5
8	3	74	5
9	3	74	5
10	3	74	5

You can see that each case now has the same number of rows as the value of the *years* variable. You could now use this to change the age at each observation to match the one-year increase in time by using:

```
bysort id: gen seq=_n-1
replace age=age+seq
```

Now the data look like this:



	id	age	years	seq
1	1	60	3	0
2	1	61	3	1
3	1	62	3	2
4	2	70	2	0
5	2	71	2	1
6	3	74	5	0
7	3	75	5	1
8	3	76	5	2
9	3	77	5	3
10	3	78	5	4

We used the **_n** function to make a new variable *seq* but instead of starting at 1 we used **_n-1** so the count started at 0 for each id. Then we replaced the old values for *age* with new values of *age* plus the *seq* value. Again, this is just a very brief indication of the data manipulation capabilities of Stata.

DEMONSTRATION EXERCISE

In Chapter 3 we manipulated the individual level variables and saved a new data set called demodata1.dta. In this part of the exercise we merge the individual data file with household level data in the hhexampledata.dta data set to add the region of country variable.

First, we need to ensure that the household level data set is correctly sorted ready for the merge as we are using a step-by-step approach rather than the **sort** option in the **merge** command. The data are opened and then inspected.

```
use hhexampledata.dta, clear
keep hid region
su hid
```

```
. su hid

Variable |     Obs       Mean   Std. Dev.       Min        Max
---------+----------------------------------------------------
     hid |    5511    1396155   219476.6    1000209    1761811
```

From this output you can see that in the hhexampledata.dta file there are data on 5511 households as each has its own unique identifier (*hid*).

Next we examine the *region* variable prior to collapsing the categories into the ones we want to use in our analyses.

```
ta region
ta region, nol
```

```
. tab region

    region / metropolitan |
                     area |      Freq.     Percent        Cum.
--------------------------+-----------------------------------
             inner london |        247        4.48        4.48
             outer london |        348        6.31       10.80
         r. of south east |        990       17.96       28.76
               south west |        493        8.95       37.71
              east anglia |        208        3.77       41.48
            east midlands |        399        7.24       48.72
west midlands conurbation |        240        4.35       53.08
      r. of west midlands |        263        4.77       57.85
       greater manchester |        242        4.39       62.24
               merseyside |        131        2.38       64.62
         r. of north west |        247        4.48       69.10
          south yorkshire |        151        2.74       71.84
           west yorkshire |        205        3.72       75.56
 r. of yorks & humberside |        175        3.18       78.73
              tyne & wear |        144        2.61       81.35
              r. of north |        216        3.92       85.27
                    wales |        281        5.10       90.36
                 scotland |        531        9.64      100.00
--------------------------+-----------------------------------
                    Total |      5,511      100.00
```

```
. tab region,nol

   region / |
metropolita |
     n area |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |        247        4.48        4.48
          2 |        348        6.31       10.80
          3 |        990       17.96       28.76
          4 |        493        8.95       37.71
          5 |        208        3.77       41.48
          6 |        399        7.24       48.72
          7 |        240        4.35       53.08
          8 |        263        4.77       57.85
          9 |        242        4.39       62.24
         10 |        131        2.38       64.62
         11 |        247        4.48       69.10
         12 |        151        2.74       71.84
         13 |        205        3.72       75.56
         14 |        175        3.18       78.73
         15 |        144        2.61       81.35
         16 |        216        3.92       85.27
         17 |        281        5.10       90.36
         18 |        531        9.64      100.00
------------+-----------------------------------
      Total |      5,511      100.00
```

As you can see from the output, there are 18 categories in the *region* variable. We recode this variable into a new variable (*region2*) which has seven categories: London, South, Midlands, Northwest, North and Northeast, Wales, and Scotland.

```
recode region (1/2=1) (3/5=2) (6/7=3) ///
   (9/11=4) (12/16=5) (17=6) (18=7), ///
   gen(region2)
lab var region2 "regions 7 categories"
lab def region 1 "London" 2 "South" ///
   3 "Midlands" 4 "Northwest" 5 "North and ///
   Northeast" 6 "Wales" 7 "Scotland"
lab val region2 region
tab region2
```

```
. tab region2

          regions 7 |
         categories |      Freq.     Percent        Cum.
--------------------+-----------------------------------
             London |        595       10.80       10.80
              South |      1,691       30.68       41.48
           Midlands |        902       16.37       57.85
          Northwest |        620       11.25       69.10
North and Northeast |        891       16.17       85.27
              Wales |        281        5.10       90.36
           Scotland |        531        9.64      100.00
--------------------+-----------------------------------
              Total |      5,511      100.00
```

The two variables *hid* and *region2* are kept then sorted on the *hid* (household identifier) variable and then saved with a new file name.

```
keep hid region2
sort hid
save hhdata1, replace
```

Next, we open the individual level data set saved from Chapter 3 (demodata1.dta) and sort by the matching variable (*hid*) before merging the household level data.

```
use demodata1, clear
sort hid
merge hid using hhdata1
ta _merge
```

```
. merge hid using hhdata1.dta
variable hid does not uniquely identify
observations in the master data

. ta _merge
     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          2 |      1,092       11.80       11.80
          3 |      8,163       88.20      100.00
------------+-----------------------------------
      Total |      9,255      100.00
```

Stata gives a warning that the *hid* variable does not uniquely identify cases in the master (individual) data. We would expect this because individuals in the same household will have the same *hid* value. The **merge** command creates a new variable *_merge* in the data. To inspect the cases involved in the merge process we tabulate the *_merge* variable (see Figure 4.2). From this output you can see that none of the cases were only in the master data (the individual level file) as there is no value 1 in the *_merge* variable. *_merge*=2 indicates how many cases were only in the using data file, which means there were no cases in the individual file to match onto. This is to be expected as we dropped cases from the individual file as we are only concerned with working age respondents. *_merge*=3 indicates that there were 8163 cases in both the master and using files. This corresponds to the number of individuals in the demodata1.dta file – see the output from this demonstration exercise in Chapter 3 or summarize the *pid* variable to check.

```
. su pid

Variable |     Obs       Mean   Std. Dev.        Min        Max
---------+-----------------------------------------------------
     pid |    8163   1.47e+07    2640230   1.00e+07   1.91e+07
```

Now we only want to keep the 8163 individual cases with matched household data and then we have no more use for the *_merge* variable so we drop it from the data set.

```
keep if _merge==3
drop _merge
```

Check the new data set before saving under a new name

```
su _all
compress
save demodata2.dta, replace
```

```
. su _all

Variable |       Obs        Mean    Std. Dev.         Min          Max
---------+-----------------------------------------------------------
     pid |      8163    1.47e+07     2640230    1.00e+07     1.91e+07
     hid |      8163     1393652    220058.3     1000381      1761811
ghqscale |      7714    10.76407    4.987117           0           36
   d_ghq |      7714    .1870625     .389987           0            1
  female |      8163    .5205194    .4996094           0            1
---------+-----------------------------------------------------------
     age |      8163    39.32733    13.08993          18           65
  agecat |      8163    1.867083    .7574497           1            3
  marst2 |      8163    1.917677    .5949101           1            4
 empstat |      7864     1.93235     1.64757           1            6
  numchd |      8163    1.431092    .6140945           1            3
---------+-----------------------------------------------------------
 region2 |      8163    3.435869    1.816138           1            7
. compress

. save demodata2.dta, replace
(note: file demodata2.dta not found)
file demodata2.dta saved
```