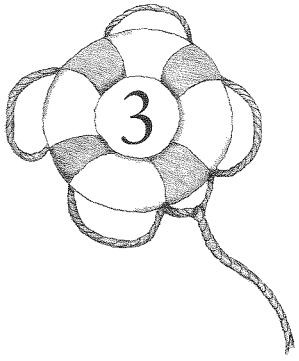

Manipulating variables

MANIPULATING VARIABLES – WHY?

Often, when you are working with data, you need to make changes to your variables. This may be for a number of reasons. For example, you may have an income variable that is measured in pounds, but you would prefer to have income grouped into a number of categories, or even percentiles, instead. You may be examining educational qualifications, but you are only really interested in studying whether people went to university or not, and therefore you would collapse a variable into two categories (i.e. went to university, didn't go to university). You may have a very detailed ethnicity or race variable that was collected as an open-ended question – that is, the survey respondents could write in whatever they pleased. Clearly a categorical variable with hundreds of possible responses would be very nearly impossible to work with and therefore you may want to collapse these into more manageable categories. There are many reasons you would want to change the way your variables are presented. One special instance is the case of missing values.

MISSING VALUES

When people answer surveys, they don't always answer all the questions, either because they would rather not or because not all the questions may be relevant to them. Survey researchers use a variety of techniques to capture the instances where a respondent may refuse to answer a question or that question isn't appropriate for the particular respondent. Often these values are recorded in the data spreadsheet as negative numbers, such as –1 for 'refused' or –9 for 'inapplicable'. Sometimes, they are stored as high numbers,

such as 99 or 999. There is no rule about this – it depends on the conventions of the particular researcher or research institute who collected the data.

You can recode the missing values to one or more of the 27 values Stata recognizes as meaning 'missing'. These 27 are a dot (.) and a dot followed by a letter (.a to .z). Stata actually uses the dot to recode the value to an extremely high number, which is why you have to be careful when using the greater than symbol (>) if the variable has missing values.

In our example data the data collectors decided to use negative numbers (i.e. –7, –8) to indicate non-response for a variety of reasons. The table below shows the output of the variable *educ* using the **tabulate** command (see Chapter 5):

```
. tabulate educ
```

highest educational qualification	Freq.	Percent	Cum.
missing	19	0.19	0.19
proxy respondent	352	3.43	3.61
higher degree	122	1.19	4.80
first degree	598	5.83	10.63
teaching qf	225	2.19	12.82
other higher qf	1,207	11.76	24.58
nursing qf	215	2.09	26.68
gce a levels	985	9.60	36.27
gce o levels or equiv	2,086	20.32	56.60
commercial qf, no o levels	349	3.40	60.00
cse grade 2-5,scot grade 4-5	411	4.00	64.00
apprenticeship	262	2.55	66.55
other qf	84	0.82	67.37
no qf	3,349	32.63	100.00
Total	10,264	100.00	

As you can see from the frequency table there are 19 cases with 'missing' values and 352 cases where this question was asked of a 'proxy respondent'. A proxy respondent is someone in the household who answers questions on the respondent's behalf if he or she is not able to participate. If we tabulate again, with the option **nol** (nolabel) asking for the values rather than the labels of the categories, we get:

```
. tabulate educ,nol
```

highest educational qualification	Freq.	Percent	Cum.
-9	19	0.19	0.19
-7	352	3.43	3.61
1	122	1.19	4.80
2	598	5.83	10.63
3	225	2.19	12.82
4	1,207	11.76	24.58
5	215	2.09	26.68
6	985	9.60	36.27
7	2,086	20.32	56.60
8	349	3.40	60.00
9	411	4.00	64.00
10	262	2.55	66.55
11	84	0.82	67.37
12	3,349	32.63	100.00
Total	10,264	100.00	

We see that –9 is the value given to 'missing' cases and –7 to 'proxy respondent' cases. For this variable we would choose to recode both –9 and –7 to be missing values.

There are a number of ways to specify missing values. The first uses the **recode** command, where we recode both the negative numbers to a dot.

```
recode educ -9=. -7=.
```

or

```
recode educ -9/-7=.
```

The forward slash / is Stata notation for a range of values – in the above example, values –9 through –7. Remember to put the lowest value first. There are no cases with a value of –8, but that doesn't matter.

If you wanted to keep the reasons for the missing values separate then you could use the dot followed by a letter values such as

```
recode educ -9=.a -7=.b
```

Also, you can recode more than one variable at a time if the variables in your list have the same values to change. For example, if our variables education (*educ*) and housing tenure (*tenure*) had the same range of missing values to recode, we could use

```
recode educ tenure (-9/-1=.)
```

or

```
recode educ tenure (-9=.a) (-7=.b)
```

Where there is more than one variable in the list, the recode values must be in parentheses. The **recode** command is also used to manipulate the non-missing values of variables, and this is covered in the next section.

You must be sure that the 'missing' values do not have a *meaningful value*. That is, in our example data, when the negative values you may think to recode to missing values can represent a real value, such as zero, for a variable. To illustrate this we shall look at the variables *ncigs*, which measures how many cigarettes a respondent smoked per day.

First, let us get some descriptive information on the variable.

```
. ta ncigs
```

number of cigarettes smoked	Freq.	Percent	Cum.
missing or wild	20	0.19	0.19
inapplicable	6,949	67.70	67.90
proxy respondent	352	3.43	71.33
refused	1	0.01	71.34
don't know	1	0.01	71.35
less than 1 per day	41	0.40	71.75
1	39	0.38	72.13
2	55	0.54	72.66
3	50	0.49	73.15
4	31	0.30	73.45
5	162	1.58	75.03

```
                  6 |         59       0.57     75.60
                  7 |         53       0.52     76.12
                  8 |         62       0.60     76.72
                  9 |         14       0.14     76.86
                 10 |        501       4.88     81.74

                etc
                 60 |          9       0.09     99.99
                 80 |          1       0.01    100.00
--------------------+-----------------------------------
              Total |     10,264     100.00
```

We can see that several respondents smoked less than one per day. We can find out the variable values as well.

```
. ta ncigs,nol

          number of |
        cigarettes |
            smoked |      Freq.     Percent        Cum.
--------------------+-----------------------------------
                -9 |         20        0.19        0.19
                -8 |      6,949       67.70       67.90
                -7 |        352        3.43       71.33
                -2 |          1        0.01       71.34
                -1 |          1        0.01       71.35
                 0 |         41        0.40       71.75
                 1 |         39        0.38       72.13
                 2 |         55        0.54       72.66
                 3 |         50        0.49       73.15
                 4 |         31        0.30       73.45
                 5 |        162        1.58       75.03
                 6 |         59        0.57       75.60
                 7 |         53        0.52       76.12
                 8 |         62        0.60       76.72
                 9 |         14        0.14       76.86
                10 |        501        4.88       81.74

               etc
                60 |          9        0.09       99.99
                80 |          1        0.01      100.00
--------------------+-----------------------------------
              Total |     10,264      100.00
```

All respondents were asked the question 'Do you smoke?'. If they answered 'Yes', then they were asked the second question, 'How many cigarettes do you smoke a day?'. If they answered 'No' to the first question then they were not asked the second question. This can be seen by the number of respondents with a value of –8 in the variable *ncigs*. This number should correspond to the number of respondents who answered 'No' to the first question. As well, there is a value of 0, which is labelled as `less than 1 per day`. But these are occasional smokers, so are they really zeros?

If you wanted a variable that indicated the number of cigarettes smoked per day then you could recode the negative values of *ncigs* in this way:

```
recode ncigs 0=1 -9=. -8=0 -7=. -2=. -1=.
```

Or, if you wanted to keep the reasons for missing values separate:

```
recode ncigs 0=1 -9=.a -8=0 -7=.b -2=.c -1=.d
```

In the second way we have kept **-9=.a** and **-7=.b** as they are for the *educ* variable, as in our data –9 and –7 denote the same reason for missing values in all of the variables. Both of these recodes will produce the frequency table of the *ncigs* variable below. The differences in coding the missing values are only shown when the **missing** option is used in the **tabulate** command which we cover in Chapter 5.

```
. ta ncigs,nol
```

number of cigarettes smoked	Freq.	Percent	Cum.
0	6,949	70.26	70.26
1	80	0.81	71.07
2	55	0.56	71.63
3	50	0.51	72.13
4	31	0.31	72.45
5	162	1.64	74.08
6	59	0.60	74.68
7	53	0.54	75.22
8	62	0.63	75.84

```
          9 |         14       0.14     75.99
         10 |        501       5.07     81.05
        etc
         60 |          9       0.09     99.99
         80 |          1       0.01    100.00
------------+-----------------------------------
      Total |      9,890     100.00
```

There is a useful command that will convert all the negative values to missing values in one step. But before you use this global recode, you must make sure that all the values that you are going to specify are *indeed logically deemed to be system missing* and you want to recode all of the missing values to a dot (.) only rather than keep the reasons for missing values separate. The command is **mvdecode** (missing value decode). The subcommand is **_all** (for all variables) and the further option **mv** specifies which value(s) to treat as missing.

```
mvdecode _all,mv(-9/-1)
```

The shortened range for the missing values (-9/-1) does not work on some Stata configurations. If this is the case with your set-up, then you can enter the missing values one at a time:

```
mvdecode _all,mv(-9)
mvdecode _all,mv(-8)
```
etc.

Also, the **mvdecode** command can be used with single variables or lists of variables and does not have to be used with **_all**:

```
mvdecode mastat,mv(-9/-1)
mvdecode tenure-fimn,mv(-9/-7)
```

The use of negative numbers to indicate values likely to be changed to missing is good practice. It has its limitations, as we have shown above. Another time when negative numbers are not appropriate is when you have standardized scores, as these variables have negative numbers as legitimate values. As we mentioned previously, some data sets use high numbers such as 99 or 999 to indicate non-response (no answer). Additional care is

needed when recoding these to missing and especially using the global recode command **mvdecode**. For example, your data may have a variable for age that uses 999 as the indicator for non-response as well as a variable for five categories of marital status that uses 9 as the value for non-response. If you simply used the global **mvdecode _all** command for the range 9/999 then all respondents aged 9 and over would be given a missing value in the age variable. Whatever system is used to indicate non-response, you need to exercise care when declaring values to be missing.

It is very important to note that there is no way to undo the **mvdecode** command. You are well advised to be absolutely sure you want to run this command. And you should *never* overwrite your original data sets – always work with copies.

As you may have gathered by now – there are usually half a dozen different ways to do what you want to do in Stata. As you become more familiar with its possibilities you will find the way that suits you best. In the numerous courses that we have taught on Stata, not one has passed without a student showing us something that we didn't know!

CREATING NEW VARIABLES OUT OF EXISTING VARIABLES

You may want to make new variables out of one or more existing variables. For example, you might want a variable that denotes women who are self-employed. To make this variable, you would need information from two variables that already existed in your data set: gender (*sex*) and job status (*jbstat*).

When you are creating variables, you often need to use mathematical expressions. The following are recognized by Stata:

+	addition
-	subtraction
*	multiplication
/	division
^	raise to the power

Parentheses are used to control the order in which calculations are done. Parentheses may be nested within other parentheses, many layers deep.

Before moving on to more advanced variable manipulation, it is necessary to understand the logical operators used in Stata.

~	not	>	greater than
==	equal (two equals signs)	>=	greater than or equal to
!=	not equal	<	less than
&	and	<=	less than or equal to
|	or		

Stata also recognizes a myriad of mathematical functions. The ones you probably use most are:

sqrt(x)	square root	log(x) or ln(x)	natural logarithm
exp(x)	exponent (*e* to power)	int(x)	integer

Type **help functions** for the full range of functions supported by Stata.

Box 3.1: Shortening commands

The shortening of commands can be used on almost all commands. The manuals or the **help** command will tell you what is the shortest accepted form of the command as shown by the underlined part of the word. For example, **tabulate** can be shortened to **ta**. Some commands cannot be shortened.

However, with **help**, you must spell the name of the command completely, so **help tabulate** will yield you results, while **help tab** will produce a message stating that the command could not be found. Help brings up the online help system for the command. Think of it as the computer version of the Stata manuals, as it often contains the same information! As well, the **search** command looks for the term you indicate in help files, Stata Technical Bulletins, and Stata FAQ if your Stata is web active.

Probably the most common command used to create new variables from pre-existing variables is **generate**. The command **generate** can be shortened to **gen**, **ge** or even **g**. To use the **generate** command, you type **generate** followed by the name of the new variable and then the set of conditions that equal the value of the new variable. For example, if we wanted to create a variable that measured yearly income, we could multiply our monthly income variable (*fimn*) by 12:

```
gen yearlyincome=fimn*12
```

Likewise, we could generate a variable that roughly equated to weekly income by dividing monthly income by, say, 4.2:

```
ge weeklyincome=fimn/4.2
```

We can use the **recode** command to make a new variable out of an existing variable. Suppose we wanted to make a simplified variable where the only categories were 'partnered', 'never married', and 'previously married' from the original marital status variable (*mastat*). 'Partnered' would include married and cohabiting couples, while 'previously married' would include all divorced, separated, and widowed people. We could do this by:

```
recode mastat 1/2=1 3/5=2 6=3
```

But this would mean that we lose the original variable categories, so we advise that you create new variables when recoding. Let's call our new variable *mastat2*, so now the command will be:

```
recode mastat 1/2=1 3/5=2 6=3,gen(mastat2)
```

The **gen** option to the **recode** command creates a new variable with the recoded categories, so you will keep the original variable and its categories as well.

```
. ta mastat2
mastat2 |      Freq.     Percent        Cum.
--------+-----------------------------------
      1 |      6,683       65.11       65.11
      2 |      1,489       14.51       79.62
      3 |      2,092       20.38      100.00
--------+-----------------------------------
  Total |     10,264      100.00
```

You should always check the variables you create. One way is to crosstabulate them, provided the number of categories is manageable:

```
. ta mastat mastat2
```

marital status	mastat2 1	2	3	Total
married	6,009	0	0	6,009
living as couple	674	0	0	674
widowed	0	866	0	866
divorced	0	434	0	434
separated	0	189	0	189
never married	0	0	2,081	2,081
Total	6,683	1,489	2,081	10,253

Generating a new variable with information from one existing variable is the simplest type of variable creation. Returning to our example of a variable that indicates women who are self-employed, we could do this in a number of different ways; we first use step-by-step approaches so you can see how the different commands work, and then we use a single command to demonstrate Stata's use of logic in constructing new variables. The self-employed have the value 1 in the *jbstat* variable (10 categories) and the missing values have been recoded to a dot (.), while women have the value 2 in the *sex* variable and there are no missing values on the *sex* variable.

The step-by-step way is to generate a new variable (*fem1*) and set all values to missing (.), and then to **replace** the values as they meet specific conditions. The command **replace** allows you to change the contents of a variable, so to speak.

```
gen fem1=.
replace fem1=1 if jbstat==1 & sex==2

replace fem1=0 if jbstat>=2 & jbstat<=12
replace fem1=0 if sex==1 & jbstat==1
```

This will create a value of 1 for all women who are self-employed. You are then left to decide what to do with the rest of the values. A variable must take more than one value to vary. Often, a researcher will use dummy variables to denote 1 for the category

of interest and 0 for cases that do not meet this condition, so we replace the new variable with zeros where the cases are women who are not self-employed and all men. The third line replaces those who have an employment status but are not self employed with a 0. The fourth line replaces men who are self-employed with a 0. The step-by-step way of creating this new variable is cumbersome, but it does help to understand what is actually happening in your data with each command. However, we hope that you are able to move quickly on to use more efficient commands.

We can first reduce the four commands to three:

```
gen fem2=1 if jbstat==1 & sex==2
replace fem2=0 if jbstat>=2 & jbstat<=12
replace fem2=0 if sex==1 & jbstat==1
```

In the first command line the **generate** (**gen**) command creates a new variable *fem2* with cases equal to 1 for women who are self-employed, and then the rest of the cases are automatically set as missing (.). The last two command lines are the same as in the first way, replacing all cases that are not self-employed or are self-employed men with a zero. Note here that we do not use **jbstat!=1** (*jbstat* not equal to one) as this would code the cases who are missing on *jbstat* to zero.

We can now reduce the three commands to two:

```
gen fem3=1 if jbstat==1 & sex==2
replace fem3=0 if ((jbstat>=2&jbstat<=10)| ///
   sex==1) & !missing(jbstat)
```

In this way the last two lines are combined to replace cases who are not self-employed or are men as zeros. It is necessary to include the last element of the command line – **& !missing(jbstat)** – in order not to include men who have missing values on the *jbstat* variable in the zeros. This element means 'not missing values on the *jbstat* variable'.

The most efficient way to create this new variable is to use Stata's logic for constructing new variables which then means it can be done in one command line:

```
gen fem4=jbstat==1 & sex==2 if ///
   !missing(jbstat,sex)
```

In this form of the **gen** command, cases who are self-employed (**jbstat==1**) and women (**sex==2**) are automatically given the value 1 while all other cases are given zeros except for those who are missing on either *jbstat* or *sex* – **if !missing (jbstat,sex)** – in which case they are given a missing value (.). Note the use of the **if** in this way rather than the **&** in the last line of making the *fem3* variable. Including the variable **sex** in the parentheses after **!missing** is not strictly necessary as we know in our data that there are no missing values on that variable.

A check on the summary statistics (see Chapter 5) of the new variables shows that they are identical:

```
. su fem*

Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
    fem1 |    9912    .0204802    .1416433         0          1
    fem2 |    9912    .0204802    .1416433         0          1
    fem3 |    9912    .0204802    .1416433         0          1
    fem4 |    9912    .0204802    .1416433         0          1
```

DUMMY VARIABLES

One very common reason to manipulate pre-existing variables is to create dummy variables, or dichotomous variables, with the values of 0 and 1. Dummy variables are a special case of nominal variables. Remember, nominal variables are those which measure qualitative characteristics such as sex, religion, or marital status. There are two reasons why we use dummy variables, and they are based upon what you should already know about nominal variables:

- The only measure of central tendency that makes sense for a nominal variable is the mode. The mean and median for nominal variables are not meaningful. Although you can get Stata or any other software program to produce a mean and a median for nominal variables, that doesn't mean they have any meaningful value.
- There are lots of nominal-type characteristics that are important predictors of the things that social scientists are interested in – but because we can't determine a 'meaningful mean' or median, we need to make a special adjustment so that we can use nominal variables in multivariate statistics.

See Box 3.2 if you've heard of dummy variables but aren't exactly sure what they are or how to interpret them.

Box 3.2: Dummy variables

Let us start with a typical dummy variable: sex. Sex is a nominal variable and a predictor of many dependent variables in which we are interested: pay, poverty, family structure, etc. But sex is nominal, which limits what we can do with it in terms of statistics. However, there are ways of including nominal variables in statistical analysis through the use of these dummy variables. If we take the example of sex, a transformation into a dummy variable would mean that we would pick one (category of) sex and code it 1 (e.g. female), leaving the other sex (in this case, male) to be coded 0. Why would we do this?

Let's look at it in a spreadsheet:

ID	Sex (1 = female)
1	1
2	1
3	0
4	0
5	0
6	1
7	1
8	0
9	1
10	1

In this example, ID just refers to the personal identifier of the respondent, and sex is the dummy variable, where 1 = female and 0 = male. When we transform our variable into a dummy variable like this, it becomes possible to calculate a mean that we can interpret. If we add up the values of the variable sex above and divide by the number of cases, we get:

$$(1 + 1 + 0 + 0 + 0 + 1 + 1 + 0 + 1 + 0)/10 = 0.6$$

Because we've transformed the variable into 1s and 0s, what this number represents now is the proportion of the sample that takes the value 1 on the dummy variable; that is, the proportion of the sample that is female. So typically, you would see a dummy variable such as this presented in a table of descriptive statistics like this:

Variable	Mean	Standard Dev.	Minimum	Maximum
Sex (1 = female)	0.6	–	0	1

Sometimes the note '(1 = female)' isn't presented and the authors just put 'Female' instead of 'Sex' as the variable name, assuming that readers will understand that it is dummy variable:

Variable	Mean	Standard Dev.	Minimum	Maximum
Female	0.6	–	0	1

So, both of these examples mean exactly the same thing: 60% of the sample is female. There is also the additional hint that the variable is a 'dummy' because the minimum is reported as 0 and the maximum is reported as 1. The standard deviation for dummy variables is not interpretable, although some authors occasionally put it in.

Often, dummy variables are created out of yes/no questionnaire items as well (which are also dichotomous). Suppose you saw something like this:

Variable	Mean	Standard Dev.	Minimum	Maximum
Capital punishment should be reinstated	0.15	–	0	1
Single parent	0.45	–	0	1

You could understand it to mean that 15% of the sample answered 'Yes' to a questionnaire item which asked if they think capital punishment should be reinstated. Similarly, 45% of this sample were single parents when the only possible categories in the variable were 1 = yes, 0 = no.

Nominal variables with more than two categories

Nominal variables often have more than two categories. Let us consider the example of a measure of where people live or their region of residence with a variable that has the following categories: 1= North, 2 = South, 3 = East, 4 = West. If you want to include region as an independent variable in your study you need to make some adjustments to it. You must make dummy variables for as many categories as there are in the original variable. In other words, we need to make four dummy variables to measure region in this example. So what we do is:

- make a variable *North* where 1 = North 0 = all other responses;
- make a variable *South* where 1 = South 0 = all other responses;
- make a variable *East* where 1 = East 0 = all other responses;
- make a variable *West* where 1 = West 0 = all other responses.

Let's look at our original variable in a spreadsheet:

ID	Region
1	3
2	2
3	4
4	1
5	1
6	1
7	2
8	3
9	2
10	2

So we have 3 cases in the North, 4 in the South, 2 in the East, and 1 in the West. When we convert region into dummy variables, the spreadsheet will look like this:

ID	Region	North	South	East	West
1	**3**	0	0	**1**	0
2	**2**	0	**1**	0	0
3	**4**	0	0	0	**1**
4	**1**	**1**	0	0	0
5	**1**	**1**	0	0	0
6	**1**	**1**	0	0	0
7	**2**	0	**1**	0	0
8	**3**	0	0	**1**	0
9	**2**	0	**1**	0	0
10	**2**	0	**1**	0	0

We have the original variable here and the four dummy variables. If we compute the mean of the dummy variables we will get:

North, $(0 + 0 + 0 + 1 + 1 + 1 + 0 + 0 + 0 + 0)/10 = 0.3$;
South, $(0 + 1 + 0 + 0 + 0 + 0 + 1 + 0 + 1 + 1)/10 = 0.4$;
East, $(1 + 0 + 0 + 0 + 0 + 0 + 0 + 1 + 0 + 0)/10 = 0.2$;
West, $(0 + 0 + 1 + 0 + 0 + 0 + 0 + 0 + 0 + 0 +)/10 = 0.1$.

These means represent the proportion of those who had the value 1 on each of the dummy variables. In a table, you would see this presented typically as:

Variable	Mean	Standard Dev	Minimum	Maximum
Region				
North	0.3	–	0	1
South	0.4	–	0	1
East	0.2	–	0	1
West	0.1	–	0	1

▶ We have four dummy variables measuring region. Of our sample, 30% live in the North, 40% in the South, 20% in the East, and 10% in the West. Note that if you add up these percentages, you get 100%, and likewise if you add up the numbers in the 'mean' column, you will end up with 1.0.

You may wonder why the variable sex can be converted into just one dummy variable and region required four. The reason is that when we convert a dichotomous variable (which only has two categories in the original) we only need to present one variable for the category we choose to code as '1'. It is assumed that whatever the other category for the dichotomous variable was will be represented as the zero.

You will want to note that when you use dummy variables in a multivariate analysis, one group always is omitted as the 'reference category'. In the case of sex, male is omitted and the coefficients for 'female' would be compared to those for 'male'. In the example of region, one category would have to be omitted (e.g. North) and the other categories compared to the omitted category when interpreting the results. This will be discussed further when we are including dummy variables in multivariate estimations in Chapters 8 and 9.

As we've said before, and will say again, there is often more than one way to complete a task in Stata that gets you to the right answer. The creation of variables is a case in point. Let us show you some different ways of creating dummy variables.

Suppose we wanted to create a dummy variable for those who have a university degree called *degree* from the *educ* variable where there are 12 categories of qualifications, with 1 and 2 being degrees and the missing values recoded to a dot (.). We could do this in a number of more or less efficient ways.

(a) Create a new variable and recode it:

```
generate degree1=educ
recode degree1 1/2=1 3/max=0
```

(b) Recode *educ* and make the new variable at the same time:

```
recode educ (1/2=1) (3/max=0), ///
   gen (degree2)
```

(c) Use **generate** and **replace** commands:

```
gen degree3=1 if educ<=2
replace degree3=0 if educ>=3 & ///
   !missing(educ)
```

(d) Use the **generate** command in one line:

```
gen degree4=educ<=2 if !missing(educ)
```

It is very important in (c) and (d) to have the **!missing(educ)** element at the end – note that one is a **&** and the other is an **if**. If you make another variable (*degree5*) where you omit this part from (d), and then do a frequency distribution on both, you will see that they are different.

```
gen degree5=educ<=2
tab1 degree4 degree5
```

```
. tab1 degree4 degree5

-> tabulation of degree4

degree4 |      Freq.     Percent        Cum.
--------+-----------------------------------
      0 |      9,173       92.72       92.72
      1 |        720        7.28      100.00
--------+-----------------------------------
  Total |      9,893      100.00

-> tabulation of degree5

degree5 |      Freq.     Percent        Cum.
--------+-----------------------------------
      0 |      9,544       92.99       92.99
      1 |        720        7.01      100.00
--------+-----------------------------------
  Total |     10,264      100.00
```

Where have the missing values gone? Without the **!missing(educ)** element of the command, Stata has put all the missing values into the 0 category of the dummy variable *degree 5*. This is because Stata recognizes missing values (dot, .a, etc.) as a very high number and every time we use < or > we must ensure that we tell Stata not to include missing values!

You may wonder why, in (d), Stata knows to make a variable coded 1 and 0. Nowhere in the code are 0 and 1 explicitly stated, like in the other examples. This has to do with how Stata understands double and single equal signs. In (d), we are setting a condition and if it is met, Stata logically assigns 1 to the condition and 0 to the cases were the condition is not met. See Box 3.3 on double and single equal signs for more information on this topic.

Now we have created five variables measuring whether or not someone has a degree. We don't need all these. We can get rid of the redundant variables using the command **drop**.

```
drop degree1 degree2 degree3 degree5
```

Box 3.3: Single and double equals signs

New users to Stata often get confused as to when single (=) or double (==) equals signs should be used. A simple rule of thumb is that a single equals sign is used to assign the value at the right of it the value to the left. For example, in the command

```
generate x=educ
```

a new variable called *x* has been created which is the same as the variable *educ*. We have told Stata that *x* is going to be the same as *educ*.

A double equal sign, on the other hand, is used to check whether the value to the right is the same as the value to the left. It is used for comparison purposes, rather than 'assignment' (like the single equal sign). The command

```
summarize if sex==2
```

would return the summary statistics of all variables in the data set for females only.

So if you are going to set the values of a variable to something, use a single equals sign, and if you are testing for equality, use the double equals signs. As with anything else, the more you practice, the more it makes sense. Often, single and double equals signs are used within the same operation.

Often when we want to include nominal (categorical) variables in statistical estimations, it is necessary to create a series of dummy variables from the original variable. This is done easily by combining the **tab** and **gen** commands. For example, if we wanted to create a series of dummies for the job status variable:

```
tab jbstat, gen(jobstat)
```

If you look at the bottom of your variable list, you will see that a number of dummy variables have been created (*jobstat1*, *jobstat2*, *jobstat3*, etc.). You can do a frequency distribution of these variables like any other variable. You will see that they are all dichotomized into a 0/1 format.

LABELLING VARIABLES

From our previous example on page 50 we have a new marital status variable (*mastat2*). Having done the frequency distribution (**tab**) of the variable, we can see that there are no labels attached to the values of the variable. Now we want to label the variable and values so that we don't forget which each number stands for. Labeling in Stata is a bit different for those used to labeling variables in other statistical software packages. There are essentially three steps:

1. Create a label for the variable itself.
2. Define a label for values.
3. Attach that label to a specific variable.

We first label our variable:

```
label var mastat2 "simpler marital status"
```

Now our variable *mastat2* has a label 'simpler marital status'. If you **tab mastat2** you will see this new label and it will also be in your variable list.

Now we want to label the values of the categories of *mastat2*. To do this use the command **label define**. First, we have to define a name for our labels. We are calling these labels 'marital'.

```
lab def marital 1 "partnered" 2 "prev married" ///
   3 "single"
```

If you make an error when you are using this command and try to do it again, you will likely get a message that the labels have already been defined. Stata will not let you overwrite labels unless it is sure that you actually want to! You can assure Stata that you intend to overwrite labels by adding the option **modify**. Try:

```
lab def marital 1 "partnered" 2 "prev married" ///
   3 "single", modify
```

It is important to realize that at this point you have defined a label, but this label is not yet attached to any variable categories. The label exists separately from any variable. If you type **label list** you will see all the labels in your data set. The label for 'marital' will be there with all the category values defined as we have entered them above.

Now we must attach these category labels to the variable using the procedure **label value**.

```
lab val mastat2 marital
```

This tells Stata to attach the values we assigned to the label 'marital' to the variable *mastat2*. Once value labels have been defined they can be applied to other variables. If we now tabulate the new marital status variable we get:

```
. ta mastat2

     simpler |
     marital |
      status |      Freq.     Percent        Cum.
-------------+-----------------------------------
   partnered |      6,683       65.18       65.18
prev married |      1,489       14.52       79.70
      single |      2,081       20.30      100.00
-------------+-----------------------------------
       Total |     10,253      100.00
```

This may seem like a nonsense way of doing labels, but it does have one really great benefit. If you have numerous variables with the same codes (e.g. 'yes' and 'no'), it is very easy to label them.

We have created some dummy variables earlier in the chapter: one for self-employed women and one for degrees. To label them all, we would just define a label:

```
lab def yesno 1 "yes" 0 "no"
```

and then attach these values to our dummy variables:

```
lab val fem1 yesno
lab val degree4 yesno
```
. . . etc

If you were using multiple dummy variables in your data set, you can easily see how quick it would be to label them all.

For ways of changing a number of variable names at once, see the commands in Box 3.4.

Box 3.4: Renaming variables

If a variable already has a name you might want to change it. It could be that the original one isn't as informative as it should be; some of the older data sets were very restricted in the number of characters they could use to name variables, so it's not uncommon to see variable names such as *sdt0068* for the number of children in the house which it would more convenient to change to *numchd*. For this example you would use:

```
rename sdt0068 numchd
```

The **rename** command is suitable when you only have a few variables to change as you can only change one variable per **rename** command. If you have more than half a dozen variables to change it is worth installing the **renvars** command. Type:

```
findit renvars
```

Then follow the instructions to install the command. ▶

There are a number of options with **renvars** but mostly it allows you to rename multiple variables in one command. Let's continue with our example above where a data set has variable names *std0068*, *std0069* and *std0070* and you want to change these to more meaningful names. Typing

```
renvars std0068 std0069 std0070 \ numchd ///
    chdage chdsex
```

would change the variable names listed to the left of the backslash (\) to the new variable names to the right of the backslash in the respective order. If the original variables are consecutive in the data set then you can just put the first and last separated by a dash (as with many other Stata commands), for example:

```
renvars std0068-std0070 \ numchd chdage chdsex
```

Another useful command to be aware of is **renpfix**. This command can either change variable name stubs, such as *sdt* in our example, or remove them completely. If you wanted to change *sdt* stub to *chd* for our three example variables, you would type

```
renpfix std chd
```

You would now have three variables: *chd0068*, *chd0069* and *chd0070*. If you typed

```
renpfix st
```

Stata would remove the *st* stub from all variables that start with that stub, in this case leaving three variables named *d0068*, *d0069* and *d0070*. All variables that start with the stub will be changed, so be careful with this command. Note that you cannot use variable names that start with a number.

A common use for **renpfix** is if you have panel data and each wave of data has variables that start with the same letter. In the British Household Panel Survey all the variables in the first wave start with *a*, second wave start with *b*, third wave start with *c*, and so on. When you put data sets together in a panel format the variables need to be named the same in each so **renpfix** can be used to remove the starting letter (stub) from all variable names.

MORE ON GENERATING NEW VARIABLES

Stata has a second, 'extended' generate command that allows you to create numerous types of variables from your existing data both for analysis and for data management. The command is **egen** and as you get more familiar with Stata then you will probably find yourself using this command more and more as it is very powerful. There are a considerable number of uses for **egen** so we cannot cover all of them here. Instead we use a few examples to illustrate its utility. Remember that you can type **help egen** to find out all of the ways to use the command.

The **group** function of **egen** creates a new variable by combining categories of two or more variables. Let's take a simple example of two variables both with only two categories. First we make a variable that indicates those who are married compared to all others:

```
recode mastat (1=1) (2/max=0), gen(married)
```

We have included the **(1=1)** for clarity but it is not strictly necessary for this recode. Now we want to create a new variable that has the following four categories: married men, married women, unmarried men, and unmarried women. If we crosstabulated the variables *married* and *sex* we would see that:

```
ta sex married
```

```
. ta sex married

           | RECODE of mastat
           | (marital status)
       sex |         0         1 |     Total
-----------+---------------------+----------
      male |     1,886     2,947 |     4,833
    female |     2,369     3,062 |     5,431
-----------+---------------------+----------
     Total |     4,255     6,009 |    10,264
```

From this table we can see the numbers in each of the four categories we are interested in, but we don't have them in a single variable. The long way to do this would be to use the **gen** and **replace** commands with **if** statements such as:

```
gen sexmar1=1 if sex==1&married==0
replace sexmar1=2 if sex==1&married==1
replace sexmar1=3 if sex==2&married==0
replace sexmar1=4 if sex==2&married==1
```

```
. gen sexmar1=1 if sex==1&married==0
(8378 missing values generated)

. replace sexmar1=2 if sex==1&married==1
(2947 real changes made)

. replace sexmar1=3 if sex==2&married==0
(2369 real changes made)

. replace sexmar1=4 if sex==2&married==1
(3062 real changes made)
```

Then if we label the categories and tabulate the new variable:

```
lab def sexmar 1 "unmarried men" ///
   2 "married men" 3 "unmarried women" ///
   4 "married women"
lab val sexmar1 sexmar
ta sexmar1
```

```
. lab def sexmar 1 "unmarried men" ///
   2 "married men" 3 "unmarried women" ///
   4 "married women"

. lab val sexmar1 sexmar

. ta sexmar1
```

sexmar1	Freq.	Percent	Cum.
unmarried men	1,886	18.37	18.37
married men	2,947	28.71	47.09
unmarried women	2,369	23.08	70.17
married women	3,062	29.83	100.00
Total	10,264	100.00	

If you compare the categories of this new variable with the crosstabulation of *sex* and *married* above you can see the new categories represent each cell of the crosstabulation. However, using the **egen** command considerably shortens this process:

```
egen sexmar2=group(sex married)
ta sexmar2
```

```
. ta sexmar2

group(sex |
 married) |      Freq.     Percent        Cum.
----------+-----------------------------------
        1 |      1,886       18.37       18.37
        2 |      2,947       28.71       47.09
        3 |      2,369       23.08       70.17
        4 |      3,062       29.83      100.00
----------+-----------------------------------
    Total |     10,264      100.00
```

We ordered our **replace** commands to re-create the process that the **egen group** command uses to make its categories so that you can easily compare the results. This is that the first category of the first variable in the list is taken first and then groups are made with that and the categories of the second variable. So, in the *sex* variable 1 = men and 2 = women, and in the *married* variable 0 = unmarried and 1 = married. Therefore the categories of the new variable are 1 = men unmarried, 2 = men married, 3 = women unmarried, 4 = women married. So we can use the label 'sexmar' with this variable as well:

```
lab val sexmar2 sexmar
ta sexmar2
```

```
. ta sexmar2

      group(sex |
       married) |      Freq.     Percent        Cum.
----------------+-----------------------------------
  unmarried men |      1,886       18.37       18.37
    married men |      2,947       28.71       47.09
unmarried women |      2,369       23.08       70.17
  married women |      3,062       29.83      100.00
----------------+-----------------------------------
          Total |     10,264      100.00
```

Obviously, this gets more complicated if the variables have more than two categories and if you use more than two variables. It's

important to understand how Stata orders categories of variables in a list; not only for this command but for many others as well such as the **by** and **bysort** commands that we cover in Chapter 4.

The second function of **egen** we cover here is **rownonmiss**. This function tells Stata to look across a number of variables and, for each case (row), create a new variable that shows how many non-missing values there are. Non-missing means any value which has not being designated as missing (. or .a, .b, etc.). Remember that you need to have coded the values to missing prior to using this command. In this example, we take three General Health Questionnaire (GHQ) items, *ghqa*, *ghqb* and *ghqd*, to show how this command works by creating a new variable called *obs*. Then to show the frequencies of the new variable we need to tabulate it:

```
egen obs=rownonmiss(ghqa ghqb ghqd)
ta obs
```

```
. egen obs=rownonmiss(ghqa ghqb ghqd)
. ta obs
```

obs	Freq.	Percent	Cum.
0	530	5.16	5.16
1	1	0.01	5.17
2	14	0.14	5.31
3	9,719	94.69	100.00
Total	10,264	100.00	

This table shows us that 9719 people answered all three questions, 14 answered two (note that this doesn't show you which two), 1 answered only one question, and 530 did not answer any of the three questions. Another way of looking at this is that those given a zero on the variable *obs* are missing on all three questions. To show this, we tabulate the three variables for those who have a zero:

```
tab1 ghqa ghqb ghqd if obs==0
```

```
. tab1 ghqa ghqb ghqd if obs==0

-> tabulation of ghqa if obs==0
no observations
```

```
-> tabulation of ghqb if obs==0
no observations

-> tabulation of ghqd if obs==0
no observations
```

This confirms that those 530 cases have not answered any of the three questions.

The values of *obs* can be used in a number of ways when creating and manipulating variables, especially when creating scales (see below) and deciding how many questions need to be answered to be included. It may also be used to select cases for analysis or to manage data sets when some cases are kept or dropped.

We carry on using the three GHQ items to demonstrate another function of **egen**. This function, **rowmean**, creates a new variable with a value of the mean of the variables specified. The default method for this function creates a mean for any cases that have at least one non-missing value. So, a mean might be created for someone who has only answered one of the questions. You need to decide if that is something you wish to do. So, we will use the *obs* variable created by the **rownonmiss** function to refine the variable creation. We start with the default method:

```
egen mean1=rowmean(ghqa ghqb ghqd)
```

```
. egen mean1=rowmean(ghqa ghqb ghqd)
(530 missing values generated)
```

Stata tells us that the new variable, *mean1*, has been created with 530 missing values. Refer back to the table of *obs* and see that 530 cases were missing on all three questions and these are the ones that are missing on this new variable. While you are looking at the table of *obs*, you can see that one person answered only one of the three questions. If we wanted to exclude that person from having a mean then we can use the values of *obs* to condition the **rowmean** function by restricting it to only those who answered two or three questions:

```
egen mean2=rowmean(ghqa ghqb ghqd) if obs>1
```

```
. egen mean2=rowmean(ghqa ghqb ghqd)if obs>1
(531 missing values generated)
```

The output shows 531 missing values in the new variable, *mean2*, which reflects the inclusion of the person who only answered one question. To follow this example through to its logical conclusion we now use the **rowmean** function and the values of the *obs* variable to calculate a mean for only those who answered all three questions:

```
egen mean3=rowmean(ghqa ghqb ghqd) if obs==3
```

```
. egen mean3=rowmean(ghqa ghqb ghqd) if obs==3
(545 missing values generated)
```

Now for the new variable, *mean3*, there are 545 missing values, 14 more than in *mean2*, which shows that the 14 people who only answered two questions are now given missing values. We now show the descriptives of all three newly created mean value variables:

```
su mean*
```

```
. su mean*

Variable |     Obs        Mean    Std. Dev.       Min        Max
-------------------------------------------------------------
   mean1 |    9734    1.976645    .4311008         1          4
   mean2 |    9733    1.976643    .4311229         1          4
   mean3 |    9719    1.976198    .4302754         1          4
```

This output shows the different number of non-missing observations in each of the mean score variables. The relatively small number of cases that did not answer all three questions means that their exclusion has very little effect on the mean scores for the sample. However, there are other reasons why you may want to exclude these cases or you may be happy to include them.

CREATING A SCALE

It is often useful to create a scale out of a number of variables in a data set to measure a more general concept. You may, for example, have 10 different variables that measure depression, but find that they can be grouped together to form a single overall measure of depression. There are, mathematically, many ways to create a scale, but one common way is to simply add up the variables to create a summed scale.

We have a number of GHQ (Goldberg and Williams 1988) measures in our sample data set. If we add these up and create a summed measure, we would do it like this:

```
gen ghq= ghqa+ ghqb+ ghqc+ ghqd+ ghqe+ ///
   ghqf+ ghqg+ ghqh+ ghqi+ ghqj+ ghqk+ ghql
```

This command might look a little cumbersome, so let's use the **egen** command to do the same thing:

```
egen obs2=rownonmiss(ghqa-ghql)
egen ghq2=rowtotal(ghqa-ghql) if obs==12
```

Compare the two:

```
su ghq ghq2
```

```
. su ghq ghq2

Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
     ghq |    9613    22.77125    4.914182         12         48
    ghq2 |    9613    22.77125    4.914182         12         48
```

Be careful how you use the **rowtotal** function in **egen** as it does different things with missing values in some circumstances between version 9 and 10, so it would be best to check using **help egen**. In this example it's not a problem as we are using the *obs2* variable to determine which cases to add across the variables.

The variables used to make up this scale all have values from 1 to 4 with high values that are associated with an undesirable characteristic, such as being depressed, being under strain, or having lessened ability to face problems. So the resulting scale has a minimum of 12 and a maximum of 48. You should note that individuals who are missing on even one of the composite items are given a 'missing' value in the overall scale because of the condition **obs2==12**.

The GHQ is a well-established scale and we can be fairly confident of its reliability and validity. But what if we are working with data that are less familiar?

It is very important that you are familiar with the variables in your proposed scale and that they are all in the 'same direction'.

The difference between Stata and other statistical software programs is that if you were making a satisfaction scale in SPSS, for example, it would be important that all the variables used to create this scale all have the same 'direction' of measurement. If some are coded in the opposite order (to measure dissatisfaction, for example), you would have to recode these items so that all the measures were in the same direction (i.e. measuring the extent of satisfaction). Stata, however, has the ability to examine variables and 'reverse score' items that it thinks are reverse coded using the command **alpha**. The default setting in Stata is to empirically determine the relationship and reverse the scorings for any that enter (i.e. are correlated with other items) negatively. We will return to this topic shortly.

But how do you know if it is a good scale? There are several ways of assessing the reliability of a scale. One of the most common is the Cronbach's alpha, which is what the scaling command **alpha** is based upon. For all proposed scale items, **alpha** computes the inter-item correlations (or covariances) for all pairs of variables in the variable list. The command will also return a Cronbach's alpha statistic for the new scale. Cronbach's alpha statistic ranges from 0 to 1, with values closer to one indicating a 'better', more internally consistent scale. It should be noted, however, that scales comprised of a high number of variables will have higher alpha values than scales with the same inter-item correlations but with fewer variables. So the number of items in a scale must be kept in mind when assessing alpha value. However, in general, a value of about 0.70 or higher is generally considered acceptable for a scale.

Let's try a different set of variables – those that assess opinions on women and their role in the home and workplace. To find out how good our measure is, we can ask Stata to report a reliability coefficient for us:

```
alpha opfama opfamb opfamc opfamd opfame ///
   opfamf opfamg opfamh opfami
```

or

```
alpha opfama-opfami
```

Note that if you use the latter of these two **alpha** commands, the variables must be in this order in your variable list.

```
. alpha opfama-opfami

Test scale = mean(unstandardized items)
Reversed items: opfamc opfamd opfame opfamh
opfami

Average interitem covariance:     .204454
Number of items in the scale:           9
Scale reliability coefficient:     0.6958
```

We are given an alpha of 0.6958, which is a little on the low side for a nine-item scale. We can also see that Stata has listed a number of reversed items. This means that Stata has decided that these five items are 'negatively' scored compared to the other items in the scale, and as such, has 'reverse coded' them to fit in with the theme of the scale.

Let's examine the items as they were asked to the survey respondents.

opfama: A pre-school child is likely to suffer if his or her mother works.
opfamb: All in all, family life suffers when the woman has a full-time job.
opfamc: A woman and her family would all be happier if she goes out to work.
opfamd: Both the husband and wife should both contribute to the household income.
opfame: Having a full-time job is the best way for a woman to be an independent person.
opfamf: A husband's job is to earn money; a wife's job is to look after the home and family.
opfamg: Children need a father to be as closely involved in their upbringing as the mother.
opfamh: Employers should make special arrangements to help mothers combine jobs and childcare.
opfami: A single parent can bring up children as well as a couple.

The response categories for all items were: (1) strongly agree, (2) agree, (3) neither agree nor disagree, (4) disagree and (5) strongly disagree.

As the items are, if a person strongly agreed that pre-school children suffered if a mother worked (*opfama*), he or she would

get a score of 1. However, such a person would be unlikely to strongly agree with the statement that a woman and her family would be happier if she worked (*opfamc*). In this case, the respondent might strongly disagree with such a statement, giving a score of 5. However, both these opinions reflect a tendency to be 'conservative' in opinions about gender roles in a family. Stata has picked up that people who answered in certain ways on items like *opfama* and *opfamb* (where high scores reflect more 'liberal' opinions about gender roles) were likely to have 'reversed' scores on items *opfamc*, *opfamd*, *opfame*, *opfamh*, and *opfami* (where high scores reflect a more 'conservative' orientation). As the items that were 'reverse scored' were more conservative, this means that higher scores on our new scale are associated with more liberal opinions.

But does Stata always get it right? It is the case that the five items highlighted by Stata as being reverse coded are 'opposite' in direction to *opfama*, *opfamb*, and *opfamf*. But you could argue that agreeing with *opfamg* may also indicate more liberal views. Stata, however, hasn't picked this up.

The problem isn't something with Stata. The algorithm with which Stata works to decide which items should be reverse coded relies on the item's correlations with the other scale items. If it is negatively correlated, it becomes reverse coded.

Let's create a correlation matrix of these items:

```
. corr opfam*
(obs=9510)

        | opfama  opfamb  opfamc  opfamd  opfame  opfamf  opfamg opfamh opfami
--------+---------------------------------------------------------------------
opfama  |  1.0000
opfamb  |  0.6465  1.0000
opfamc  | -0.2798 -0.3550  1.0000
opfamd  | -0.1230 -0.1620  0.3192  1.0000
opfame  | -0.1059 -0.1797  0.3327  0.3610  1.0000
opfamf  |  0.4375  0.5438 -0.2455 -0.1043 -0.0756  1.0000
opfamg  |  0.1456  0.1215 -0.0225  0.0883  0.0496  0.0959  1.0000
opfamh  | -0.1892 -0.1825  0.1746  0.1729  0.1606 -0.2193  0.1356 1.0000
opfami  | -0.2871 -0.2698  0.1406  0.1312  0.0860 -0.1895 -0.1476 0.2153 1.0000
```

You can see from this matrix that the reverse coded items (*opfamc*, *opfamd*, *opfame*, *opfamh*, *opfami*) are negatively correlated with *opfama*, *opfamb* and *opfamf* (and positively with the other 'reverse coded' items). On the other hand, *opfamg* does not follow this general pattern, which is why it wasn't reverse coded by Stata.

There are a couple possible reasons for this. The most likely is that it is not a good item for your scale – that it somehow does not

'fit' as well as the other items. You can check this by using the option **item**:

```
alpha opfama-opfami,item
```

```
. alpha opfama-opfami,item

Test scale = mean(unstandardized items)
```

Item	Obs	Sign	item-test correlation	item-rest correlation	average inter-item covariance	alpha
opfama	9628	+	0.6949	0.5465	.172436	0.6301
opfamb	9662	+	0.7466	0.6127	.1618418	0.6131
opfamc	9640	-	0.5667	0.4340	.2053443	0.6604
opfamd	9646	-	0.4510	0.2786	.2201311	0.6863
opfame	9648	-	0.4481	0.2639	.2198141	0.6896
opfamf	9652	+	0.6375	0.4588	.1808333	0.6498
opfamg	9657	+	0.2283	0.0726	.2533817	0.7152
opfamh	9645	-	0.4424	0.2746	.2222064	0.6866
opfami	9648	-	0.5297	0.3319	.2040992	0.6788
Test scale					.204454	0.6958

Of particular interest is the last column labelled 'alpha'. It would be better labelled as 'alpha if item removed', because that is what it is telling us. If we look down to *opfamg*, we can see that the alpha of the scale would improve to 0.7152 if we took this item out. Alpha can only tell us this information for items one at a time – it won't tell us the effect on alpha of removing two or three items at once.

So it seems that *opfamg* isn't such a great measure for our proposed scale. And if we look at it, we can see that the item is somewhat different from the other items because it is asking about general child upbringing, rather than issues pertaining specifically to employment.

It should be stressed that you should always check the items in your scale and make sure that they make theoretical sense. The mathematical techniques involved in scale construction do not do this for you!

So what happens if we take out *opfamg*?

```
. alpha opfama opfamb opfamc opfamd opfame opfamf opfamh ///
    opfami,item

Test scale = mean(unstandardized items)
```

Item	Obs	Sign	item-test correlation	item-rest correlation	average inter-item covariance	alpha
opfama	9628	+	0.6880	0.5338	.2231625	0.6574
opfamb	9662	+	0.7448	0.6076	.2081992	0.6384
opfamc	9640	-	0.5771	0.4419	.2618421	0.6824
opfamd	9646	-	0.4773	0.3036	.2777641	0.7062
opfame	9648	-	0.4684	0.2815	.2786443	0.7112
opfamf	9652	+	0.6393	0.4534	.2327838	0.6764
opfamh	9645	-	0.4754	0.3076	.2790421	0.7050
opfami	9648	-	0.5191	0.3129	.2656042	0.7083
Test scale					.2533817	0.7152

We can see that the alpha has improved and that no further removal of individual items will increase our alpha.

As we haven't created the scale variable yet, we could ask Stata to create it for us after it computed the reliability using the **gen** option.

```
alpha opfama opfamb opfamc opfamd opfame ///
   opfamf opfamh opfami, gen(famscale1)
```

The **gen** option tells Stata to create a new variable (we have named it *famscale1*). Unless we tell Stata otherwise, it will go ahead and reverse code items. If you are ever in a position where you don't want Stata to do this, typing the option **asis** will tell Stata not to reverse code any items. You can also use the option **std** to get Stata to create the new variable in standardized form (a mean of 0 and a standard deviation of 1).

It is very important to note how Stata handles missing data in the **alpha** command. In other commands that we have talked about so far in this book, cases are deleted from the analysis if they are missing on at least one of the variables under consideration. So when we **tab sex age**, if a case is missing on the age variable, it will not be included in the tabulation. It is deleted 'casewise'. Similarly, we could create the scale just by simply adding up all the variables using the **gen** command (after reverse coding manually). Now, the cases where there was missing data on one or more of the scale items would not be included. The default in the **gen** option, however, is to create a score for every observation where there is a response to at least one scale item. In other words, a scale value could be created for someone who

answered only one of the eight *opfam* variables. The score is calculated by dividing the summative score over the total number of items available for the specific case.

If you find this objectionable (we do!), you may want to employ the option **casewise** so that cases with any missing values on the scale items are deleted. Let's see how this changes our results.

```
alpha opfama opfamb opfamc opfamd opfame ///
   opfamf opfamh opfami,gen(famscale2) casewise
```

```
. alpha opfama opfamb opfamc opfamd opfame ///
   opfamf opfamh opfami,gen(famscale2) casewise

Test scale = mean(unstandardized items)
Reversed items: opfamc opfamd opfame opfamh
   opfami

Average interitem covariance:     .2540916
Number of items in the scale:            8
Scale reliability coefficient:      0.7166
```

The results suggest a slightly better alpha. But we also know that cases are only included if respondents answered all the items in our scale.

If you don't want to be so stringent, you can set an alternative minimum number of items that must be non-missing in order for the scale to be constructed. Let's say we decided that a person must have answered at least five of the eight items in order to be included in the scale, we would use the option **min**.

```
alpha opfama opfamb opfamc opfamd opfame ///
   opfamf opfamh opfami, gen(famscale3) min(5)
```

When we create a scale, we are just really adding up items and, as such, we would rightly expect that if we are adding up 8 items, all of which have values of 1 to 5, our scale will have a minimum of 8 and a maximum of 40. People who score 8 would be very conservative, while those around the 40 mark would be rather liberal.

```
recode opfamc opfamd opfame opfamh opfami ///
   (1=5) (2=4) (3=3) (4=2) (5=1)
```

```
gen famscale4= opfama+ opfamb+ opfamc+ ///
   opfamd+ opfame+ opfamf+ opfamh+ opfami
```

However, if we **summarize** our new variables (*famscale1*, *famscale2* and *famscale3*) and compare them with *famscale4* created by manually reverse coding and using **gen** to add the items (note the use of the wildcard * that saves typing all the variables). We get:

```
. su famscale*

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
   famscale1 |      9718   -.5544141    .6009038        -4          2
   famscale2 |      9515   -.5526143    .5954785     -2.75       1.25
   famscale3 |      9657   -.5529802    .5968845     -2.75          2
   famscale4 |      9515    25.57909    4.763828         8         40
```

What is happening?

In order to understand these values, you need to understand how Stata constructs the scale with the **alpha** command. When an item is 'reverse scored' it isn't recoded so that 1 becomes 5, 2 becomes 4, etc. What happens is that a negative sign is placed in front of all the original values so that the original values are changed to:

Variable	Direction	New values
opfama	+	1 2 3 4 5
opfamb	+	1 2 3 4 5
opfamc	–	–5 –4 –3 –2 –1
opfamd	–	–5 –4 –3 –2 –1
opfame	–	–5 –4 –3 –2 –1
opfamf	+	1 2 3 4 5
opfamh	–	–5 –4 –3 –2 –1
opfami	–	–5 –4 –3 –2 –1

This logic produces an overall total, for those who answered all eight items, with a minimum of –22 and a maximum of 10. If we divide these scores by 8 (the total number of items), we get –2.75 and 1.25 which match the minimum and maximum values in the data shown for *famscale2* which used the **casewise** option so that only cases with data on all eight items were included.

The theoretical minimum and maximum values for *famscale1*, created using the default settings for the **gen** option, are –5 and 5 as it is possible for respondents to answer just one item and be included in the scale. You can see from the Obs column that there are about 200 more cases in the *famscale1* variable than in the *famscale2* variable.

Determining the theoretical minimum and maximum values of the scale is a little more complex for *famscale3*, which used the option **min(5)** to tell Stata to use only cases that have answers to five or more items. If we take the five lowest possible scores which are the five reverse coded items then we get $-25/5 = -5$ as a minimum. The five highest possible scores are the three unaltered items and two reverse coded items: 5+5+5–1–1 so $13/5 = 2.6$ is the maximum. You can see that the actual minimum and maximum values in the data fall short of these extremes.

However, if we correlate all four scales we see that they are mathematically equivalent for the cases that were included on each pair of scales. The decision rests with you, as the analyst, as to how many answers you need for someone to obtain an overall scale score.

```
. pwcorr famscale*,obs

             |  famsca~1  famsca~2  famsca~3  famsca~4
-------------+------------------------------------------
famscale1    |   1.0000
             |     9718
famscale2    |   1.0000   1.0000
             |     9515     9515
famscale3    |   1.0000   1.0000   1.0000
             |     9657     9515     9657
famscale4    |   1.0000   1.0000   1.0000   1.0000
             |     9515     9515     9515     9515
```

For an account of using a scale in an applied research project, see Box 3.5.

Box 3.5: An example of using a scale in a project

In a project funded by the Department of Health we conducted a series of analyses investigating the consequences of early childbearing on outcomes later in life. These were later published as papers examining housing,[1] partners and partnerships,[2] and gender differences in outcomes.[3] The data we used were from the British Cohort Study of 1970 which has followed approximately 15,000 children from birth in 1970 until the present day. Part of the analysis was to construct a measure of childhood behaviour prior to any childbearing. The cohort was interviewed at age 10 in 1980. At this time their teachers were also asked about their behaviour in school. We decided to use teacher reported behaviour as well as parent reported behaviour. We took all the answers from the teachers and examined the correlations between the variables and, after some thought and analysis, settled on a scale of five items that measured the child's concentration on educational tasks, their popularity with peers, the number of friends, their level of co-operation with peers, and the extent that the teachers were able to negotiate with the child. All items were measured using a 'thermometer' type response ranging from 1 to 47.

These five items had correlations between 0.33 and 0.84, which suggested that they might make a reasonable scale without all measuring the same thing. We standardized all five items to a mean of 0 and a standard deviation of 1 using the **egen** command to make five new variables: *zone*, *ztwo*, *zthree*, *zfour*, *zfive*. Then we used the **alpha** command to determine that the resulting scale had a Cronbach's alpha of 0.82, which was very satisfactory for a five-item scale. We used the **item** option on the **alpha** command to see if omitting items would improve the alpha value, and the results indicated that no significant improvement could be gained. To create the final score for each child we used the **egen**

[1] Ermisch, J.F. and Pevalin, D.J. (2004) Early childbearing and housing choices. *Journal of Housing Economics*, 13: 170–194.

[2] Ermisch, J.F. and Pevalin, D.J. (2005) Early motherhood and later partnerships. *Journal of Population Economics*, 18: 469–489.

[3] Robson, K. and Pevalin, D.J. (2007) Gender differences in the predictors and outcomes of young parenthood. *Research in Social Stratification and Mobility*, 25: 205–218.

command again to take a mean of the five items where a low score indicated poor behaviour.

```
. alpha zone- zfive,item

Test scale = mean(unstandardized items)

                                                        average
                              item-test    item-rest inter-item
Item          |   Obs Sign  correlation correlation covariance  alpha
--------------+-----------------------------------------------------
zone          | 12550    +       0.6689      0.4770   .5438279 0.8266
ztwo          | 12606    +       0.8546      0.7514   .4237435 0.7463
zthree        | 12575    +       0.8186      0.6949   .4474017 0.7641
zfour         | 12565    +       0.8164      0.6919   .4489863 0.7652
zfive         | 12506    +       0.6714      0.4816   .5412759 0.8252
--------------+-----------------------------------------------------
Test scale    |                                         .481128 0.8226
------------------------------------------------------------------
```

The final measure of childhood behaviour was a significant predictor of early parenthood, but the regression models indicated an interaction effect with gender (see Chapters 8 and 9 for regression and interaction effects) where this behaviour predicted early motherhood but not early fatherhood. Then we looked at when the cohort was 30 years old and we found that this behaviour measure was significantly associated with a range of outcomes including their educational attainment, labour force participation, pay, social class, house ownership, and receipt of benefits.

DEMONSTRATION EXERCISE

In this demonstration exercise we use some of the techniques covered in this and each of the subsequent chapters to conduct a series of data analyses exploring the question of social variations in mental health among working age adults. Our measure of mental health is the 12-item General Health Questionnaire, which is a scale that can be constructed in a number of ways. In this demonstration we start by using the GHQ in the form of a scale that ranges from 0 to 36, with higher scores indicating poorer mental health. The factors we are interested in using are sex, age, marital status, employment status, number of own children in the household and region of the country.

We start by opening the example data file (exampledata.dta) from our default directory. Before opening the data file we increase the memory available to Stata to 50 Mb using the **set mem** command:

```
version 10
set mem 50m
cd "C:\project folder"
use exampledata.dta
```

Next we use the **keep** command to retain only the individual level variables we need for this analysis and then recode all the negative values to missing. We keep the individual identifier (*pid*) and the household identifier (*hid*) so that we can match on household level information in the next chapter.

```
keep pid hid ghq* sex age mastat jbstat nchild
mvdecode _all,mv(-9/-1)
```

Stata returns the output below. You can see that after the **mvdecode** command, Stata tells you how many missing values were assigned for each variable. For example, the variable *jbstat* had 352 missing values while the variable *ghql* had 580 missing values. From this you can deduce that the variables *pid*, *hid*, *sex*, *age*, *mastat* and *nchild* do not have any missing values.

```
. mvdecode _all,mv(-9/-1)
      jbstat: 352 missing values generated
        ghqa: 536 missing values generated
        ghqb: 536 missing values generated
        ghqc: 545 missing values generated
        ghqd: 534 missing values generated
        ghqe: 535 missing values generated
        ghqf: 546 missing values generated
        ghqg: 534 missing values generated
        ghqh: 532 missing values generated
        ghqi: 534 missing values generated
        ghqj: 577 missing values generated
        ghqk: 587 missing values generated
        ghql: 580 missing values generated
```

We now create the GHQ scale from the 12 items in the data set. The items are coded from 1 to 4 but we want to make a scale that goes from 0 to 36, so we need to recode all the 12 GHQ items

to go from 0 to 3. Note the use of the wildcard (*) to save listing every item. Another way to do this would be to use the dash as the items are in order in the data set (i.e. **recode ghqa-ghql**). We then check the internal consistency of the scale using the **alpha** command. In this example we use the **item** option to give us some more information.

```
recode ghq* (4=3) (3=2) (2=1) (1=0)
alpha ghq*,item
```

The output shows the changes made to each of the GHQ items then the details of the scale internal reliability check. The overall alpha value (0.8631) is reported at the bottom right of the table; all the signs are positive and all items have similar item–rest correlations. The right-hand column shows that the overall alpha value would not be increased by dropping any item. We should be reasonably happy with the internal reliability of this scale.

```
. recode ghq* (4=3) (3=2) (2=1) (1=0)
(ghqa: 9728 changes made)
(ghqb: 9728 changes made)
(ghqc: 9719 changes made)
(ghqd: 9730 changes made)
(ghqe: 9729 changes made)
(ghqf: 9718 changes made)
(ghqg: 9730 changes made)
(ghqh: 9732 changes made)
(ghqi: 9730 changes made)
(ghqj: 9687 changes made)
(ghqk: 9677 changes made)
(ghql: 9684 changes made)
. alpha ghq*,item
Test scale = mean(unstandardized items)
```

Item	Obs	Sign	item-test correlation	item-rest correlation	average inter-item covariance	alpha
ghqa	9728	+	0.5891	0.5119	.1517366	0.8548
ghqb	9728	+	0.6413	0.5325	.1408634	0.8541
ghqc	9719	+	0.5092	0.4116	.1539663	0.8603
ghqd	9730	+	0.4798	0.3983	.1580361	0.8607
ghqe	9729	+	0.6981	0.5990	.1361276	0.8489
ghqf	9718	+	0.6835	0.5943	.1403709	0.8489
ghqg	9730	+	0.6220	0.5429	.1487009	0.8527
ghqh	9732	+	0.5680	0.4970	.154532	0.8561
ghqi	9730	+	0.7721	0.6895	.1299578	0.8415
ghqj	9687	+	0.7324	0.6497	.135947	0.8447
ghqk	9677	+	0.6483	0.5638	.1450927	0.8511
ghql	9684	+	0.6220	0.5479	.1499576	0.8528
Test scale					.1454405	0.8631

The command creates the scale using the **gen** command. As we have shown in this chapter, you could use the **alpha** command with a **gen** option but we prefer to construct the scale manually in this example.

```
gen ghqscale=ghqa+ghqb+ghqc+ghqd+ghqe+ghqf ///
   +ghqg+ghqh+ghqi+ghqj+ghqk+ghql
lab var ghqscale "ghq 0-36"
su ghqscale
```

In this part of the output, Stata lets us know that in creating the scale 651 missing values have been generated in the new variable (*ghqscale*). This is because the **gen** command only creates a new variable for those cases that have non-missing values on all 12 items. The next line labels the new variable and then we use the **su** command to display the descriptive statistics of the new variable, which shows that we have 9613 cases with a new scale score. There is further discussion of descriptive statistics commands in Chapter 5.

```
. gen ghqscale = ghqa+ghqb+ghqc+ghqd+ghqe ///
>      +ghqf+ghqg+ghqh+ghqi+ghqj+ghqk+ghql
(651 missing values generated)

. lab var ghqscale "ghq 0-36"

. su ghqscale

Variable |   Obs        Mean    Std. Dev.       Min  Max
---------+--------------------------------------------------
ghqscale |  9613    10.77125    4.914182         0   36
```

We now construct another variable based on the GHQ items. This one uses a coding of 0–0–1–1 for each of the items and then adds up the items to a maximum of 12. Then a threshold of 4 or more is used to make a dichotomous indicator. First we recode all the 12 GHQ items and then sum them to create a new variable called *d_ghq*.

```
recode ghq* (0/1=0) (2/3=1)
gen d_ghq=ghqa+ghqb+ghqc+ghqd+ghqe+ghqf ///
   +ghqg+ghqh+ghqi+ghqj+ghqk+ghql
ta d_ghq
```

```
. ta d_ghq
```

d_ghq	Freq.	Percent	Cum.
0	4,933	51.32	51.32
1	1,423	14.80	66.12
2	873	9.08	75.20
3	600	6.24	81.44
4	447	4.65	86.09
5	341	3.55	89.64
6	260	2.70	92.34
7	210	2.18	94.53
8	162	1.69	96.21
9	103	1.07	97.28
10	112	1.17	98.45
11	94	0.98	99.43
12	55	0.57	100.00
Total	9,613	100.00	

The tabulation shows us that the recode and summing have been done correctly. Now we recode the *d_ghq* variable into a dichotomous indicator where 1 equals those with a GHQ score of 4 or more:

```
recode d_ghq 0/3=0 4/12=1
ta d_ghq
```

```
. ta d_ghq
```

d_ghq	Freq.	Percent	Cum.
0	7,829	81.44	81.44
1	1,784	18.56	100.00
Total	9,613	100.00	

The tabulation of the dichotomous GHQ indicator shows that 18.56% of the current cases in the data set are over the threshold.

As we are interested in variations of mental health for those aged 18 to 65, we use the **keep** command to retain only those cases within that age range. Compare this use of the **keep** command – keeping cases – with the other use earlier in this

example when it was used to keep variables. Similarly, the **drop** command can be used to drop variables or cases depending on how the command is formatted. We then produce descriptive statistics of the *age* variable to see how many cases we have left in our data.

```
keep if age>=18 & age<=65
su age
```

The output shows how many cases (observations) are deleted from the data set after implementing the **keep** command. From the descriptive statistics for the variable *age*, we see that now we only have 8163 cases in our data. Remember that there are no missing cases in the *age* variable.

```
. keep if age>=18 & age<=65
(2101 observations deleted)

. su age

Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
     age |    8163    39.32733    13.08993        18         65
```

Next we recode the *age* variable into three categories and use the **gen** option in the **recode** command to create a new variable called *agecat*. We then label the new variable and its categories. We then use the **tab** command to produce a frequency table of the new variable to check if our recode and labelling have come out correctly.

```
recode age (18/32=1) (33/50=2) ///
   (51/65=3),gen(agecat)
lab var agecat "age categories"
lab def agelab 1 "18-32 years" ///
   2 "33-50 years" 3 "51-65 years"
lab val agecat agelab
tab agecat
```

After the **recode** command, Stata tells us that there are 8163 (all cases) differences between the original variable *age* and the newly created variable *agecat*. As we numbered the categories of *agecat* 1, 2, and 3 and there is no one in the data under 18 years of age, it

is not surprising that for all cases the values of *age* and *agecat* are different. The frequency table produced from the **tab** command shows the number and percentage of the cases in each of the three age categories.

```
. recode age (18/32=1) (33/50=2) ///
   (51/65=3),gen(agecat)
(8163 differences between age and agecat)

. lab var agecat "age categories"

. lab def agelab 1 "18-32 years" ///
   2 "33-50 years" 3 "51-65 years"

. lab val agecat agelab

. tab agecat

        age |
 categories |      Freq.     Percent        Cum.
------------+-----------------------------------
18-32 years |      2,956       36.21       36.21
33-50 years |      3,336       40.87       77.08
51-65 years |      1,871       22.92      100.00
------------+-----------------------------------
      Total |      8,163      100.00
```

Our next step is to recode the *sex* variable into a dummy variable that indicates female cases. We use the **recode** command with the **gen** option again. We label the new variable and its categories, then produce a frequency table to check our recode.

```
tab sex
tab sex,nol
recode sex (1=0) (2=1),gen(female)
lab var female "female indicator"
lab def sexlab 0 "male" 1 "female"
lab val female sexlab
tab female
```

We see the frequency table of the *sex* variable but we need to see what numbers lie underneath the category labels of male and female. We use the **tab** command with the **nol** option.

```
. ta sex

      sex |      Freq.     Percent        Cum.
----------+-----------------------------------
     male |      3,914       47.95       47.95
   female |      4,249       52.05      100.00
----------+-----------------------------------
    Total |      8,163      100.00

. ta sex,nol

      sex |      Freq.     Percent        Cum.
----------+-----------------------------------
        1 |      3,914       47.95       47.95
        2 |      4,249       52.05      100.00
----------+-----------------------------------
    Total |      8,163      100.00

. recode sex (1=0) (2=1),gen(female)
(8163 differences between sex and female)

. lab var female "female indicator"

. lab def sexlab 0 "male" 1 "female"

. lab val female sexlab

. ta female

   female |
indicator |      Freq.     Percent        Cum.
----------+-----------------------------------
     male |      3,914       47.95       47.95
   female |      4,249       52.05      100.00
----------+-----------------------------------
    Total |      8,163      100.00
```

To reduce the number of marital status categories, we recode the marital status variable (*mastat*) into a new variable called *marst2* and have four categories, where 1 = single, 2 = married/cohabiting, 3 = separated/divorced and 4 = widowed. We need to see what the categories and the numbering are in the original marital status variable (*mastat*). We do this by using the **tab** command with the **nol** option. We then recode, create the new variable and label the new variable and its categories.

```
tab mastat
tab mastat,nol
recode mastat (6=1) (1/2=2) (4/5=3) ///
   (3=4),gen(marst2)
lab var marst2 "marital status 4 categories"
lab def marlab 1 "single" 2 "married" ///
   3 "sep/div" 4 "widowed"
lab val marst2 marlab
tab marst2
```

The output for these commands is similar to that above. The exact process and commands you use to recode variables may vary from this, but we strongly advise you to have a system that allows you to check your recoding as you go along.

```
. tab mastat

  marital status |      Freq.     Percent        Cum.
-----------------+-----------------------------------
         married |      5,132       62.87       62.87
living as couple |        654        8.01       70.88
         widowed |        189        2.32       73.20
        divorced |        397        4.86       78.06
       separated |        172        2.11       80.17
   never married |      1,619       19.83      100.00
-----------------+-----------------------------------
           Total |      8,163      100.00

. tab mastat,nol

         marital |
          status |      Freq.     Percent        Cum.
-----------------+-----------------------------------
               1 |      5,132       62.87       62.87
               2 |        654        8.01       70.88
               3 |        189        2.32       73.20
               4 |        397        4.86       78.06
               5 |        172        2.11       80.17
               6 |      1,619       19.83      100.00
-----------------+-----------------------------------
           Total |      8,163      100.00
```

```
. recode mastat (6=1) (1/2=2) (4/5=3) ///
    (3=4),gen(marst2)
(7509 differences between mastat and marst2)

. lab var marst2 "marital status 4 categories"

. lab def marlab 1 "single" 2 "married" ///
    3 "sep/div" 4 "widowed"

. lab val marst2 marlab

. tab marst2
```

marital status 4 categories	Freq.	Percent	Cum.
single	1,619	19.83	19.83
married	5,786	70.88	90.71
sep/div	569	6.97	97.68
widowed	189	2.32	100.00
Total	8,163	100.00	

We now create the employment status variable:

```
ta jbstat
ta jbstat,nol
recode jbstat (1/2=1) (3=2) (7=3) (6=4) ///
   (9=4) (5=5) (8=5) (4=6) (10=.), gen(empstat)
lab var empstat "employment status"
lab def emplab 1 "employed" 2 "unemployed" ///
   3 "longterm sick" 4 "studying" ///
   5 "family care" 6 "retired"
lab val empstat emplab
ta empstat
```

```
. ta jbstat
```

current labour force status	Freq.	Percent	Cum.
self employed	731	9.26	9.26
in paid employ	4,844	61.39	70.65

```
                unemployed |    505     6.40   77.05
                   retired |    403     5.11   82.16
                family care |    900    11.41   93.56
                ft student |    202     2.56   96.12
long term sick/disabled |    244     3.09   99.21
           on matern leave |     13     0.16   99.38
          govt trng scheme |     22     0.28   99.66
            something else |     27     0.34  100.00
---------------------------+-----------------------
                     Total | 7,891   100.00
```

```
. ta jbstat,nol
```

current labour force status	Freq.	Percent	Cum.
1	731	9.26	9.26
2	4,844	61.39	70.65
3	505	6.40	77.05
4	403	5.11	82.16
5	900	11.41	93.56
6	202	2.56	96.12
7	244	3.09	99.21
8	13	0.16	99.38
9	22	0.28	99.66
10	27	0.34	100.00
Total	7,891	100.00	

```
. recode jbstat (1/2=1) (3=2) (7=3) (6=4) ///
>   (9=4) (5=5) (8=5) (4=6) (10=.),gen(empstat)
(6260 differences between jbstat and empstat)

. lab var empstat "employment status"

. lab def emplab 1 "employed" 2 "unemployed" ///
>   3 "longterm sick" 4 "studying" ///
>   5 "family care" 6 "retired"

. lab val empstat emplab
```

```
. ta empstat

   employment |
       status |      Freq.     Percent        Cum.
--------------+-----------------------------------
     employed |      5,575       70.89       70.89
   unemployed |        505        6.42       77.31
longterm sick |        244        3.10       80.42
     studying |        224        2.85       83.27
  family care |        913       11.61       94.88
      retired |        403        5.12      100.00
--------------+-----------------------------------
        Total |      7,864      100.00
```

Next we collapse the variable for number of children into fewer categories:

```
su nchild
recode nchild (0=1) (1/2=2) (3/9=3), ///
   gen(numchd)
lab var numchd "children 3 categories"
lab def chdlab 1 "none" 2 "one or two" ///
   3 "three or more"
lab val numchd chdlab
```

```
. su nchild

Variable |       Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
  nchild |      8163   .6659316    1.019895         0          9

. recode nchild (0=1) (1/2=2) (3/9=3), ///
   gen(numchd)
(6508 differences between nchild and numchd)

. lab var numchd "children 3 categories"

. lab def chdlab 1 "none" 2 "one or two" ///
   3 "three or more"

. lab val numchd chdlab

. ta numchd
```

```
   children 3 |
   categories |      Freq.     Percent        Cum.
--------------+-----------------------------------
         none |      5,182       63.48       63.48
   one or two |      2,443       29.93       93.41
three or more |        538        6.59      100.00
--------------+-----------------------------------
        Total |      8,163      100.00
```

When we have completed our recoding we produce descriptive statistics for all the variables that we will use in our future analyses.

```
su ghqscale d_ghq female age agecat marst2 ///
   empstat numchd
```

We can use the output of descriptive statistics to see if our variables have the right number of categories and cases (observations). The output below shows that some of the variables have fewer valid observations than the 8163 in our total sample. This is due to the GHQ items, employment status and marital status variables having a number of people who did not respond to the questions and so have been coded as missing values.

```
. su ghqscale d_ghq female age agecat marst2 ///
   empstat numchd

Variable |       Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
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
```

Finally, we use the **keep** command again to retain only the variables we wish to use in further analyses. The **order** command lets us order the variables in the data set if this is something you prefer. The **compress** command stores the data set in the smallest

amount of space, and then the **save** command saves our new data set to our default directory for future use.

```
keep pid hid ghqscale d_ghq female age ///
   agecat marst2 empstat numchd
order pid hid ghqscale d_ghq female age ///
   agecat marst2 empstat numchd
compress
save demodata1.dta,replace
```

```
. keep pid hid ghqscale d_ghq female age ///
   agecat marst2 empstat numchd

. order pid hid ghqscale d_ghq female age ///
   agecat marst2 empstat numchd

. compress
ghqscale was float now byte

. save demodata1.dta,replace
(note: file demodata1.dta not found)
file demodata1.dta saved
```