You're not lost. We have a new look but the same content.

Stata Learning Module
Labeling data

This module will show how to create labels for your data. Stata allows you to label your data file (**data label**), to label the variables within your data file (**variable labels**), and to label the values for your variables (**value labels**). Let's use a file called autolab that does not have any labels.

```
use http://www.ats.ucla.edu/stat/stata/modules/autolab.dta, clear
```

Let's use the **describe** command to verify that indeed this file does not have any labels.

```
describe
Contains data from autolab.dta
 obs:            74                          1978 Automobile Data
 vars:           12                          23 Oct 2008 13:36
 size:        3,478 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
-----------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
-----------------------------------------
make            str18  %-18s
price           int    %8.0gc
mpg             int    %8.0g
rep78           int    %8.0g
headroom        float  %6.1f
trunk           int    %8.0g
weight          int    %8.0gc
length          int    %8.0g
turn            int    %8.0g
displacement    int    %8.0g
gear_ratio      float  %6.2f
foreign         byte   %8.0g
-------------------------------------------------------------------------------
--
Sorted by:
```

Let's use the **label data** command to add a label describing the data file. This label can be up to 80 characters long.

```
label data "This file contains auto data for the year 1978"
```

The **describe** command shows that this label has been applied to the version that is currently in memory.

```
describe
Contains data from autolab.dta
 obs:            74                          This file contains auto data for
the year 1978
```

```
 vars:            12                          23 Oct 2008 13:36
 size:         3,478 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
-----------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
-----------------------------------------
make            str18  %-18s
price           int    %8.0gc
mpg             int    %8.0g
rep78           int    %8.0g
headroom        float  %6.1f
trunk           int    %8.0g
weight          int    %8.0gc
length          int    %8.0g
turn            int    %8.0g
displacement    int    %8.0g
gear_ratio      float  %6.2f
foreign         byte   %8.0g
-------------------------------------------------------------------------------
--
Sorted by:
```

Let's use the **label variable** command to assign labels to the variables **rep78 price**, **mpg** and **foreign**.

```
label variable rep78   "the repair record from 1978"
label variable price   "the price of the car in 1978"
label variable mpg     "the miles per gallon for the car"
label variable foreign "the origin of the car, foreign or domestic"
```

The **describe** command shows these labels have been applied to the variables.

```
describe
Contains data from autolab.dta
 obs:            74                          This file contains auto data for
the year 1978
 vars:            12                          23 Oct 2008 13:36
 size:         3,478 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
-----------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
-----------------------------------------
make            str18  %-18s
price           int    %8.0gc                the price of the car in 1978
mpg             int    %8.0g                 the miles per gallon for the
car
rep78           int    %8.0g                 the repair record from 1978
headroom        float  %6.1f
trunk           int    %8.0g
weight          int    %8.0gc
length          int    %8.0g
```

```
turn            int    %8.0g
displacement    int    %8.0g
gear_ratio      float  %6.2f
foreign         byte   %8.0g                 the origin of the car, foreign
or domestic
------------------------------------------------------------------------------
--
Sorted by:
```

Let's make a value label called **foreignl** to label the values of the variable **foreign**. This is a two step process where you first define the label, and then you assign the label to the variable. The **label define** command below creates the value label called **foreignl** that associates 0 with **domestic car** and 1 with **foreign car**.

```
label define foreignl 0 "domestic car" 1 "foreign car"
```

The **label values** command below associates the variable **foreign** with the label **foreignl**.

```
label values foreign foreignl
```

If we use the describe command, we can see that the variable **foreign** has a value label called **foreignl** assigned to it.

```
describe
Contains data from autolab.dta
 obs:            74                          This file contains auto data for
the year 1978
 vars:           12                          23 Oct 2008 13:36
 size:        3,478 (99.9% of memory free)   (_dta has notes)
------------------------------------------------------------------------------
------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
------------------------------------------
make            str18  %-18s
price           int    %8.0gc                the price of the car in 1978
mpg             int    %8.0g                 the miles per gallon for the
car
rep78           int    %8.0g                 the repair record from 1978
headroom        float  %6.1f
trunk           int    %8.0g
weight          int    %8.0gc
length          int    %8.0g
turn            int    %8.0g
displacement    int    %8.0g
gear_ratio      float  %6.2f
foreign         byte   %12.0g      foreignl   the origin of the car, foreign
or domestic
------------------------------------------------------------------------------
--
Sorted by:
```

Now when we use the **tabulate foreign** command, it shows the labels **domestic car** and **foreign car** instead of just 0 and 1.

```
table foreign
-------------+-----------
the origin   |
of the car,  |
foreign or   |
domestic     |      Freq.
-------------+-----------
domestic car |         52
 foreign car |         22
-------------+-----------
```

Value labels are used in other commands as well. For example, below we issue the **ttest , by(foreign)** command, and the output labels the groups as **domestic** and **foreign** (instead of 0 and 1).

```
ttest mpg , by(foreign)
Two-sample t test with equal variances

------------------------------------------------------------------------------
-
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf.
Interval]
---------+--------------------------------------------------------------------
-
domestic |      52    19.82692     .657777    4.743297    18.50638
21.14747
 foreign |      22    24.77273     1.40951    6.611187    21.84149
27.70396
---------+--------------------------------------------------------------------
-
combined |      74     21.2973    .6725511    5.785503     19.9569
22.63769
---------+--------------------------------------------------------------------
-
    diff |           -4.945804    1.362162               -7.661225   -
2.230384
------------------------------------------------------------------------------
-
Degrees of freedom: 72

                Ho: mean(domestic) - mean(foreign) = diff = 0

     Ha: diff <0 Ha: diff ~="0" Ha: diff> 0
       t =  -3.6308                t =  -3.6308              t =  -3.6308
   P < t =   0.0003          P > |t| =   0.0005          P > t =   0.9997
```

One very important note: These labels are assigned to the data that is currently in memory. To make these changes permanent, you need to **save** the data. When you **save** the data, all of the labels (data labels, variable labels, value labels) will be saved with the data file.

Summary

Assign a label to the data file currently in memory.

```
label data "1978 auto  data"
```

Assign a label to the variable foreign.

```
label variable foreign "the origin  of the car, foreign or domestic"
```

Create the value label **foreignl** and assign it to the variable **foreign**.

```
label define foreignl 0 "domestic  car"  1 "foreign  car"
label values foreign foreignl
```

How to cite this page

Report an error on this page or leave a comment

You're not lost. We have a new look but the same content.

Stata Learning Module
Creating and recoding variables

This module shows how to create and recode variables. In Stata you can create new variables with **generate** and you can modify the values of an existing variable with **replace** and with **recode**.

Computing new variables using generate and replace

Let's **use** the **auto** data for our examples. In this section we will see how to compute variables with **generate** and **replace**.

```
use auto
```

The variable **length** contains the length of the car in inches. Below we see summary statistics for **length**.

```
summarize length
Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+--------------------------------------------------------
  length |      74    187.9324   22.26634        142        233
```

Let's use the **generate** command to make a new variable that has the length in feet instead of inches, called **len_ft**.

```
generate len_ft = length / 12
```

We should emphasize that **generate** is for creating a new variable. For an existing variable, you need to use the **replace** command (not **generate**). As shown below, we use **replace** to repeat the assignment to **len_ft**.

```
replace len_ft = length / 12
```

```
 (49 real changes made)
```

```
summarize length len_ft
```

```
 Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
   length |      74    187.9324    22.26634       142        233
   len_ft |      74    15.66104    1.855528   11.83333   19.41667
```

The syntax of **generate** and **replace** are identical, except:
- **generate** works when the variable does not yet exist and will give an error if the variable already exists.
- **replace** works when the variable already exists, and will give an error if the variable does not yet exist.

Suppose we wanted to make a variable called **length2** which has **length** squared.

```
generate length2 = length^2
```

```
summarize length2
```

```
Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
 length2 |      74    35807.69    8364.045      20164      54289
```

Or we might want to make **loglen** which is the natural log of **length**.

```
generate loglen = log(length)
```

```
summarize loglen
```

```
 Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
   loglen |      74    5.229035    .1201383   4.955827   5.451038
```

Let's get the mean and standard deviation of **length** and we can make Z-scores of **length**.

```
summarize length
```

```
 Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
   length |      74    187.9324    22.26634       142        233
```

The mean is 187.93 and the standard deviation is 22.27, so **zlength** can be computed as shown below.

```
generate zlength = (length - 187.93) / 22.27

summarize zlength

 Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
 zlength |      74    .0001092    .9998357  -2.062416   2.023799
```

With **generate** and **replace**
you can use + - for addition and subtraction
you can use * / for multiplication and division
you can use ^ for exponents (e.g., length^2)
you can use ( ) for controlling order of operations.

Recoding new variables using generate and replace

Suppose that we wanted to break **mpg** down into three categories. Let's look at a table of **mpg** to see where we might draw the lines for such categories.

```
tabulate mpg

        mpg |      Freq.     Percent        Cum.
------------+-----------------------------------
         12 |          2        2.70        2.70
         14 |          6        8.11       10.81
         15 |          2        2.70       13.51
         16 |          4        5.41       18.92
         17 |          4        5.41       24.32
         18 |          9       12.16       36.49
         19 |          8       10.81       47.30
         20 |          3        4.05       51.35
         21 |          5        6.76       58.11
         22 |          5        6.76       64.86
         23 |          3        4.05       68.92
         24 |          4        5.41       74.32
         25 |          5        6.76       81.08
         26 |          3        4.05       85.14
         28 |          3        4.05       89.19
         29 |          1        1.35       90.54
         30 |          2        2.70       93.24
         31 |          1        1.35       94.59
         34 |          1        1.35       95.95
         35 |          2        2.70       98.65
         41 |          1        1.35      100.00
------------+-----------------------------------
      Total |         74      100.00
```

Let's convert **mpg** into three categories to help make this more readable. Here we convert **mpg** into three categories using **generate** and **replace**.

```
generate mpg3    = .

 (74 missing values generated)
```

```
replace  mpg3    = 1 if (mpg <= 18)

 (27 real changes made)

replace  mpg3    = 2 if (mpg >= 19) & (mpg <=23)

 (24 real changes made)

replace  mpg3    = 3 if (mpg >= 24) & (mpg <.)

 (23 real changes made)
```

Let's use **tabulate** to check that this worked correctly. Indeed, you can see that a value of 1 for **mpg3** goes from 12-18, a value of 2 goes from 19-23, and a value of 3 goes from 24-41.

```
tabulate mpg mpg3
```

mpg	mpg3 1	2	3	Total
12	2	0	0	2
14	6	0	0	6
15	2	0	0	2
16	4	0	0	4
17	4	0	0	4
18	9	0	0	9
19	0	8	0	8
20	0	3	0	3
21	0	5	0	5
22	0	5	0	5
23	0	3	0	3
24	0	0	4	4
25	0	0	5	5
26	0	0	3	3
28	0	0	3	3
29	0	0	1	1
30	0	0	2	2
31	0	0	1	1
34	0	0	1	1
35	0	0	2	2
41	0	0	1	1
Total	27	24	23	74

Now, we could use **mpg3** to show a crosstab of **mpg3** by **foreign** to contrast the mileage of the foreign and domestic cars.

```
tabulate mpg3 foreign, column
```

mpg3	foreign 0	1	Total
1	22	5	27
	42.31	22.73	36.49

```
-----------+----------------------+----------
         2 |         19           5 |        24
           |      36.54       22.73 |     32.43
-----------+----------------------+----------
         3 |         11          12 |        23
           |      21.15       54.55 |     31.08
-----------+----------------------+----------
     Total |         52          22 |        74
           |     100.00      100.00 |    100.00
```

The crosstab above shows that 21% of the domestic cars fall into the **high mileage** category, while 55% of the foreign cars fit into this category.

Recoding variables using recode

There is an easier way to recode **mpg** to three categories using **generate** and **recode**. First, we make a copy of **mpg**, calling it **mpg3a**. Then, we use **recode** to convert **mpg3a** into three categories: min-18 into 1, 19-23 into 2, and 24-max into 3.

```
generate mpg3a = mpg

recode   mpg3a (min/18=1) (19/23=2) (24/max=3)

 (74 changes made)
```

Let's double check to see that this worked correctly. We see that it worked perfectly.

```
tabulate mpg mpg3a
```

```
           |                mpg3a
       mpg |         1           2           3 |     Total
-----------+---------------------------------+----------
        12 |         2           0           0 |         2
        14 |         6           0           0 |         6
        15 |         2           0           0 |         2
        16 |         4           0           0 |         4
        17 |         4           0           0 |         4
        18 |         9           0           0 |         9
        19 |         0           8           0 |         8
        20 |         0           3           0 |         3
        21 |         0           5           0 |         5
        22 |         0           5           0 |         5
        23 |         0           3           0 |         3
        24 |         0           0           4 |         4
        25 |         0           0           5 |         5
        26 |         0           0           3 |         3
        28 |         0           0           3 |         3
        29 |         0           0           1 |         1
        30 |         0           0           2 |         2
        31 |         0           0           1 |         1
        34 |         0           0           1 |         1
        35 |         0           0           2 |         2
        41 |         0           0           1 |         1
-----------+---------------------------------+----------
```

```
     Total |        27         24         23 |        74
```

Recodes with if

Let's create a variable called **mpgfd** that assesses the mileage of the cars with respect to their origin. Let this be a 0/1 variable called **mpgfd** which is:
0 if below the median mpg for its group (foreign/domestic)
1 if at/above the median mpg for its group (foreign/domestic).

```
sort foreign

by foreign: summarize mpg, detail
```

```
 -> foreign=        0
                            mpg
-------------------------------------------------------------
      Percentiles      Smallest
 1%           12             12
 5%           14             12
10%           14             14       Obs                  52
25%         16.5             14       Sum of Wgt.          52

50%           19                      Mean           19.82692
                        Largest       Std. Dev.      4.743297
75%           22             28
90%           26             29       Variance       22.49887
95%           29             30       Skewness       .7712432
99%           34             34       Kurtosis       3.441459

-> foreign=        1
                            mpg
-------------------------------------------------------------
      Percentiles      Smallest
 1%           14             14
 5%           17             17
10%           17             17       Obs                  22
25%           21             18       Sum of Wgt.          22

50%         24.5                      Mean           24.77273
                        Largest       Std. Dev.      6.611187
75%           28             31
90%           35             35       Variance       43.70779
95%           35             35       Skewness        .657329
99%           41             41       Kurtosis        3.10734
```

We see that the median is 19 for the domestic (foreign==0) cars and 24.5 for the foreign (foreign==1) cars. The **generate** and **recode** commands below recode **mpg** into **mpgfd** based on the domestic car median for the domestic cars, and based on the foreign car median for the foreign cars.

```
generate mpgfd = mpg

recode   mpgfd (min/18=0) (19/max=1) if foreign==0
```

```
 (52 changes made)
```

```
recode   mpgfd (min/24=0) (25/max=1) if foreign==1
```

```
 (22 changes made)
```

We can check using this below, and the recoded value **mpgfd** looks correct.

```
by foreign: tabulate mpg mpgfd
```

```
 -> foreign=        0
           |         mpgfd
       mpg |         0          1 |     Total
-----------+----------------------+----------
        12 |         2          0 |         2
        14 |         5          0 |         5
        15 |         2          0 |         2
        16 |         4          0 |         4
        17 |         2          0 |         2
        18 |         7          0 |         7
        19 |         0          8 |         8
        20 |         0          3 |         3
        21 |         0          3 |         3
        22 |         0          5 |         5
        24 |         0          3 |         3
        25 |         0          1 |         1
        26 |         0          2 |         2
        28 |         0          2 |         2
        29 |         0          1 |         1
        30 |         0          1 |         1
        34 |         0          1 |         1
-----------+----------------------+----------
     Total |        22         30 |        52


-> foreign=        1
           |         mpgfd
       mpg |         0          1 |     Total
-----------+----------------------+----------
        14 |         1          0 |         1
        17 |         2          0 |         2
        18 |         2          0 |         2
        21 |         2          0 |         2
        23 |         3          0 |         3
        24 |         1          0 |         1
        25 |         0          4 |         4
        26 |         0          1 |         1
        28 |         0          1 |         1
        30 |         0          1 |         1
        31 |         0          1 |         1
        35 |         0          2 |         2
        41 |         0          1 |         1
-----------+----------------------+----------
     Total |        11         11 |        22
```

Summary

Create a new variable **len_ft** which is **length** divided by 12.

```
generate len_ft = length / 12
```

Change values of an existing variable named **len_ft**.

```
replace len_ft = length / 12
```

Recode **mpg** into **mpg3**, having three categories using **generate** and **replace if.**

```
generate mpg3 = .
replace  mpg3 = 1 if (mpg <=18)
replace  mpg3 = 2 if (mpg >=19) & (mpg <=23)
replace  mpg3 = 3 if (mpg >=24) & (mpg <.)
```

Recode **mpg** into **mpg3a**, having three categories, 1 2 3, using **generate** and **recode**.

```
generate mpg3a = mpg
recode   mpg3a (min/18=1) (19/23=2) (24/max=3)
```

Recode **mpg** into **mpgfd**, having two categories, but using different cutoffs for foreign and domestic cars.

```
generate mpgfd = mpg
recode   mpgfd (min/18=0) (19/max=1) if foreign==0
recode   mpgfd (min/24=0) (25/max=1) if foreign==1
```


Stata Learning Module
Subsetting data

This module shows how you can subset data in Stata. You can subset data by keeping or dropping variables, and you can subset data by keeping or dropping observations. You can also subset data as you **use** a data file if you are trying to read a file that is too big to fit into the memory on your computer.

Keeping and dropping variables

Sometimes you do not want all of the variables in a data file. You can use the **keep** and **drop** commands to subset variables. If we think of your data like a spreadsheet, this section will show how you can remove columns (variables) from your data. Let's illustrate this with the **auto** data file.

```
sysuse auto
```

We can use the **describe** command to see its variables.

```
describe
```

```
Contains data from C:\Program Files\Stata10\ado\base/a/auto.dta
  obs:            74                          1978 Automobile Data
 vars:            12                          13 Apr 2007 17:45
 size:         3,478 (99.7% of memory free)    (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 Price
mpg             int    %8.0g                  Mileage (mpg)
rep78           int    %8.0g                  Repair Record 1978
headroom        float  %6.1f                  Headroom (in.)
trunk           int    %8.0g                  Trunk space (cu. ft.)
weight          int    %8.0gc                 Weight (lbs.)
length          int    %8.0g                  Length (in.)
turn            int    %8.0g                  Turn Circle (ft.)
displacement    int    %8.0g                  Displacement (cu. in.)
gear_ratio      float  %6.2f                  Gear Ratio
foreign         byte   %8.0g       origin     Car type
-------------------------------------------------------------------------------
Sorted by:  foreign
```

Suppose we want to just have **make mpg** and **price**, we can **keep** just those variables, as shown below.

```
keep make mpg price
```

If we issue the **describe** command again, we see that indeed those are the only variables left.

```
describe
```

```
Contains data from C:\Program Files\Stata10\ado\base/a/auto.dta
obs:            74                          1978 Automobile Data
vars:            3                           13 Apr 2007 17:45
size:         1,924 (99.8% of memory free)    (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
```

```
variable name   type   format      label      variable label
------------------------------------------------------------------------------
--
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 Price
mpg             int    %8.0g                  Mileage (mpg)
------------------------------------------------------------------------------
--
Sorted by:
     Note:  dataset has changed since last saved
```

Remember, this has not changed the file on disk, but only the copy we have in memory. If we saved this file calling it **auto**, it would mean that we would replace the existing file (with all the variables) with this file which just has **make**, **mpg** and **price**. In effect, we would permanently lose all of the other variables in the data file. It is important to be careful when using the **save** command after you have eliminated variables, and it is recommended that you save such files to a file with a new name, e.g., **save auto2**. Let's show how to use the **drop** command to drop variables. First, let's clear out the data in memory and **use** the auto data file.

```
sysuse auto, clear
```

perhaps we are not interested in the variables **displ** and **gear_ratio**. We can get rid of them using the **drop** command shown below.

```
drop displ gear_ratio
```

Again, using **describe** shows that the variables have been eliminated.

```
describe
```

```
Contains data from C:\Program Files\Stata10\ado\base/a/auto.dta
obs:            74                          1978 Automobile Data
vars:           10                          13 Apr 2007 17:45
size:        3,034 (99.7% of memory free)   (_dta has notes)
------------------------------------------------------------------------------
--
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
--
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 Price
mpg             int    %8.0g                  Mileage (mpg)
rep78           int    %8.0g                  Repair Record 1978
headroom        float  %6.1f                  Headroom (in.)
trunk           int    %8.0g                  Trunk space (cu. ft.)
weight          int    %8.0gc                 Weight (lbs.)
length          int    %8.0g                  Length (in.)
turn            int    %8.0g                  Turn Circle (ft.)
foreign         byte   %8.0g       origin     Car type
------------------------------------------------------------------------------
--
Sorted by:  foreign
```

```
     Note:  dataset has changed since last save
```

If we wanted to make this change permanent, we could save the file as **auto2.dta** as shown below.

```
save auto2

file auto2.dta saved
```

Keeping and dropping observations

The above showed how to use **keep** and **drop** variables to eliminate variables from your data file. The **keep if** and **drop if** commands can be used to keep and drop observations. Thinking of your data like a spreadsheet, the **keep if** and **drop if** commands can be used to eliminate rows of your data. Let's illustrate this with the auto data. Let's use the **auto** file and **clear** out the data currently in memory.

```
sysuse auto , clear
```

The variable **rep78** has values 1 to 5, and also has some missing values, as shown below.

```
tabulate rep78 , missing
    Repair |
Record 1978 |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2        2.70        2.70
          2 |          8       10.81       13.51
          3 |         30       40.54       54.05
          4 |         18       24.32       78.38
          5 |         11       14.86       93.24
          . |          5        6.76      100.00
------------+-----------------------------------
      Total |         74      100.00
```

We may want to eliminate the observations which have missing values using **drop if** as shown below. The portion after the **drop if** specifies which observations that should be eliminated.

```
drop if missing(rep78)

 (5 observations deleted)
```

Using the **tabulate** command again shows that these observations have been eliminated.

```
tabulate rep78 , missing

      rep78 |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2        2.90        2.90
          2 |          8       11.59       14.49
          3 |         30       43.48       57.97
          4 |         18       26.09       84.06
```

```
          5 |         11       15.94      100.00
------------+-----------------------------------
      Total |         69      100.00
```

We could make this change permanent by using the **save** command to save the file. Let's illustrate using **keep if** to eliminate observations. First let's clear out the current file and **use** the **auto** data file.

```
sysuse auto , clear
```

The **keep if** command can be used to eliminate observations, except that the part after the **keep if** specifies which observations should be kept. Suppose we want to keep just the cars which had a repair rating of 3 or less. The easiest way to do this would be using the **keep if** command, as shown below.

```
keep if (rep78 <= 3)

 (34 observations deleted)
```

The **tabulate** command shows that this was successful.

```
tabulate rep78, missing
```

```
      rep78 |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2        5.00        5.00
          2 |          8       20.00       25.00
          3 |         30       75.00      100.00
------------+-----------------------------------
      Total |         40      100.00
```

Before we go on to the next section, let's clear out the data that is currently in memory.

```
clear
```

Selecting variables and observations with "use"

The above sections showed how to use **keep**, **drop**, **keep if**, and **drop if** for eliminating variables and observations. Sometimes, you may want to use a data file which is bigger than you can fit into memory and you would wish to eliminate variables and/or observations as you use the file. This is illustrated below with the **auto** data file. Selecting variables. You can specify just the variables you wish to bring in on the **use** command. For example, let's **use** the **auto** data file with just **make price** and **mpg**.

```
use make price mpg using http://www.stata-press.com/data/r10/auto
```

The **describe** command shows us that this worked.

```
describe
```

```
Contains data from http://www.stata-press.com/data/r10/auto.dta
 obs:            74                          1978 Automobile Data
vars:             3                          13 Apr 2007 17:45
size:         1,924 (99.8% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
--
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
--
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 Price
mpg             int    %8.0g                  Mileage (mpg)
-------------------------------------------------------------------------------
--
Sorted by:
```

Let's clear out the data before the next example.

```
clear
```

Suppose we want to just bring in the observations where **rep78** is 3 or less. We can do this as shown below.

```
use http://www.stata-press.com/data/r10/auto if (rep78 <= 3)
```

We can use tabulate to double check that this worked.

```
tabulate rep78, missing
```

```
       rep78 |      Freq.     Percent        Cum.
-------------+-----------------------------------
           1 |          2        5.00        5.00
           2 |          8       20.00       25.00
           3 |         30       75.00      100.00
-------------+-----------------------------------
       Total |         40      100.00
```

Let's clear out the data before the next example.

```
clear
```

Let's show another example. Lets read in just the cars that had a rating of 4 or higher.

```
use http://www.stata-press.com/data/r10/auto if (rep78 >= 4) & (rep78 <.)
```

Let's check this using the **tabulate** command.

```
tabulate rep78, missing
```

```
      rep78 |      Freq.     Percent        Cum.
```

```
------------+-----------------------------------
          4 |         18       62.07       62.07
          5 |         11       37.93      100.00
------------+-----------------------------------
      Total |         29      100.00
```

Let's clear out the data before the next example.

```
clear
```

You can both eliminate variables and observations with the **use** command. Let's read in just **make mpg price** and **rep78** for the cars with a repair record of 3 or lower.

```
use make mpg price rep78 if (rep78 <= 3) using http://www.stata-
press.com/data/r10/auto
```

Let's check this using **describe** and **tabulate**.

```
describe
```

```
Contains data from http://www.stata-press.com/data/r10/auto.dta
  obs:            40                          1978 Automobile Data
 vars:             4                          13 Apr 2007 17:45
 size:         1,120 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
--
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
--
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 Price
mpg             int    %8.0g                  Mileage (mpg)
rep78           int    %8.0g                  Repair Record 1978
-------------------------------------------------------------------------------
--
Sorted by:
```

```
tabulate rep78
```

```
       rep78 |      Freq.     Percent        Cum.
-------------+-----------------------------------
           1 |          2        5.00        5.00
           2 |          8       20.00       25.00
           3 |         30       75.00      100.00
-------------+-----------------------------------
       Total |         40      100.00
```

Let's clear out the data before the next example.

```
clear
```

Note that the ordering of **if** and **using** is arbitrary.

```
use make mpg price rep78 using http://www.stata-press.com/data/r10/auto if
(rep78 <= 3)
```

Let's check this using **describe** and **tabulate**.

```
describe

Contains data from http://www.stata-press.com/data/r10/auto.dta
  obs:            40                          1978 Automobile Data
 vars:             4                          13 Apr 2007 17:45
 size:         1,120 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
--
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
--
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 Price
mpg             int    %8.0g                  Mileage (mpg)
rep78           int    %8.0g                  Repair Record 1978
-------------------------------------------------------------------------------
--
Sorted by:
tabulate rep78

       rep78 |      Freq.     Percent        Cum.
-------------+-----------------------------------
           1 |          2        5.00        5.00
           2 |          8       20.00       25.00
           3 |         30       75.00      100.00
-------------+-----------------------------------
       Total |         40      100.00
```

Have a look at this command. Do you think it will work?

```
use make mpg if (rep78 <= 3) using http://www.stata-press.com/data/r10/auto
rep78 not found
r(111);
```

You see, **rep78** was not one of the variables read in, so it could not be used in the **if** portion. To use a variable in the **if** portion, it has to be one of the variables that is read in.

Summary

Using keep/drop to eliminate variables
keep make price mpg

drop displ gear_ratio

Using keep if/drop if to eliminate observations
drop if missing(rep78)

keep if (rep78 <= 3)

Eliminating variables and/or observations with use
use make mpg price rep78 using auto

use auto if (rep78 <= 3)

use make mpg price rep78 using auto if (rep78 <= 3)

Stata Learning Modules
Collapsing data across observations

Sometimes you have data files that need to be **collapsed** to be useful to you. For example, you might have student data but you really want classroom data, or you might have weekly data but you want monthly data, etc. We will illustrate this using an example showing how you can collapse data across kids to make family level data.

Here is a file containing information about the kids in three families. There is one record per kid. **Birth** is the order of birth (i.e., 1 is first), **age wt** and **sex** are the child's age, weight and sex. We will use this file for showing how to collapse data across observations.

```
use http://www.ats.ucla.edu/stat/stata/modules/kids, clear
list
```

	famid	kidname	birth	age	wt	sex
1.	1	Beth	1	9	60	f
2.	1	Bob	2	6	40	m
3.	1	Barb	3	3	20	f
4.	2	Andy	1	8	80	m
5.	2	Al	2	6	50	m
6.	2	Ann	3	2	20	f
7.	3	Pete	1	6	60	m
8.	3	Pam	2	4	40	f
9.	3	Phil	3	2	20	m

Consider the **collapse** command below. It collapses across all of the observations to make a single record with the average age of the kids.

```
collapse age
list
          age
  1.  5.111111
```

The above **collapse** command was not very useful, but you can combine it with the **by(famid)** option, and then it creates one record for each family that contains the average age of the kids in the family.

```
use http://www.ats.ucla.edu/stat/stata/modules/kids, clear
collapse age, by(famid)
list
         famid        age
  1.         1          6
  2.         2   5.333333
  3.         3          4
```

The following **collapse** command does the exact same thing as above, except that the average of **age** is named **avgage** and we have explicitly told the **collapse** command that we want it to compute the **mean**.

```
use http://www.ats.ucla.edu/stat/stata/modules/kids, clear
collapse (mean) avgage=age, by(famid)
list
         famid     avgage
  1.         1          6
  2.         2   5.333333
  3.         3          4
```

We can request averages for more than one variable. Here we get the average for **age** and for **wt** all in the same command.

```
use http://www.ats.ucla.edu/stat/stata/modules/kids, clear
collapse (mean) avgage=age avgwt=wt, by(famid)
list
         famid     avgage      avgwt
  1.         1          6         40
  2.         2   5.333333         50
  3.         3          4         40
```

This command gets the average of **age** and **wt** like the command above, and also computes **numkids** which is the count of the number of kids in each family (obtained by counting the number of observations with valid values of **birth**).

```
use http://www.ats.ucla.edu/stat/stata/modules/kids, clear
collapse (mean) avgage=age avgwt=wt (count) numkids=birth, by(famid)
list
         famid     avgage      avgwt    numkids
  1.         1          6         40          3
  2.         2   5.333333         50          3
  3.         3          4         40          3
```

Suppose you wanted a count of the number of boys and girls in the family. We can do that with one extra step. We will create a dummy variable that is 1 if the kid is a boy (0 if not), and a dummy variable that is 1 if the kid is a girl (and 0 if not). The sum of the **boy** dummy variable is the number of boys and the sum of the **girl** dummy variable is the number of girls.

First, let's use the kids file (and clear out the existing data).

```
use http://www.ats.ucla.edu/stat/stata/modules/kids, clear
```

We use **tabulate** with the **generate** option to make the dummy variables.

```
tabulate sex, generate(sexdum)
        sex |      Freq.     Percent        Cum.
------------+-----------------------------------
          f |          4       44.44       44.44
          m |          5       55.56      100.00
------------+-----------------------------------
      Total |          9      100.00
```

We can look at the dummy variables. **Sexdum1** is the dummy variable for girls. **Sexdum2** is the dummy variable for boys. The sum of **sexdum1** is the number of girls in the family. The sum of **sexdum2** is the number of boys in the family.

```
list famid sex sexdum1 sexdum2
          famid        sex   sexdum1   sexdum2
  1.         1          f         1         0
  2.         1          m         0         1
  3.         1          f         1         0
  4.         2          m         0         1
  5.         2          m         0         1
  6.         2          f         1         0
  7.         3          m         0         1
  8.         3          f         1         0
  9.         3          m         0         1
```

The command below creates **girls** which is the number of girls in the family, and **boys** which is the number of boys in the family.

```
collapse (count) numkids=birth (sum) girls=sexdum1 boys=sexdum2, by(famid)
```

We can list out the data to confirm that it worked correctly.

```
list famid boys girls numkids
         famid      boys     girls    numkids
  1.         1         1         2          3
  2.         2         2         1          3
  3.         3         2         1          3
```

Summary

To create one record per family (**famid**) with the average of age within each family.

```
collapse age, by(famid)
```

To create one record per family (**famid**) with the average of age (called avgage) and average weight (called avgwt) within each family.

```
collapse (mean) avgage=age avgwt=wt,  by(famid)
```

Same as above example, but also counts the number of kids within each family calling that **numkids**.

```
collapse (mean) avgage=age  avgwt=wt (count) numkids=birth, by(famid)
```

Counts the number of boys and girls in each family by using tabulate to create dummy variables based on sex and then summing the dummy variables within each family.

```
tabulate sex, generate(sexdum)
collapse (sum) girls=sexdum1 boys=sexdum2, by(famid)
```

How to cite this page

Report an error on this page or leave a comment

You're not lost. We have a new look but the same content.

Stata Learning Module
Working across variables using foreach

1. Introduction

This module illustrates (1) how to create and recode variables manually and (2) how to use **foreach** to ease the process of creating and recoding variables.

Consider the sample program below, which reads in income data for twelve months.

```
input famid inc1-inc12
1 3281 3413 3114 2500 2700 3500 3114 3319 3514 1282 2434 2818
2 4042 3084 3108 3150 3800 3100 1531 2914 3819 4124 4274 4471
3 6015 6123 6113 6100 6100 6200 6186 6132 3123 4231 6039 6215
end

list
```

The output is shown below

```
list famid inc1-inc12, clean
```

```
famid inc1    inc2    inc3    inc4    inc5    inc6    inc7    inc8    inc9
inc10    inc11    inc12
1     3281    3413    3114    2500    2700    3500    3114    3319    3514
1282     2434     2818
2     4042    3084    3108    3150    3800    3100    1531    2914    3819
4124     4274     4471
3     6015    6123    6113    6100    6100    6200    6186    6132    3123
4231     6039     6215
```

2. Computing variables (manually)

Say that we wanted to compute the amount of tax (10%) paid for each month, the simplest way to do this is to compute 12 variables (**taxinc1-taxinc12**) by multiplying each of the (**inc1-inc12**) by .10 as illustrated below. As you see, this requires entering a command computing the tax for each month of data (for months 1 to 12) via the **generate** command.

```
generate taxinc1 = inc1 * .10
generate taxinc2 = inc2 * .10
generate taxinc3 = inc3 * .10
generate taxinc4 = inc4 * .10
generate taxinc5 = inc5 * .10
generate taxinc6 = inc6 * .10
generate taxinc7 = inc7 * .10
generate taxinc8 = inc8 * .10
generate taxinc9 = inc9 * .10
generate taxinc10= inc10 * .10
generate taxinc11= inc11 * .10
generate taxinc12= inc12 * .10
```

The output is shown below.

```
     +-------------------------------------------------------------------------
----------------------+
  1. | famid | inc1 | inc2 | inc3 | inc4 | inc5 | inc6 | inc7 | inc8 | inc9 |
inc10 | inc11 | inc12 |
     |     1 | 3281 | 3413 | 3114 | 2500 | 2700 | 3500 | 3114 | 3319 | 3514 |
1282 |  2434 |  2818 |
     |-------------------------------------------------------------------------
----------------------|
     | taxinc1 | taxinc2 | taxinc3 | taxinc4 | taxinc5  | taxinc6  | taxinc7
| taxinc8  | taxinc9  |
     |   328.1 |   341.3 |   311.4 |     250 |     270  |     350  |   311.4
|   331.9  |   351.4  |
     |-------------------------------------------------------------------------
----------------------|
     |           taxinc10           |           taxinc11            |
taxinc12            |
     |              128.2           |              243.4            |
281.8            |
     +-------------------------------------------------------------------------
----------------------+

     +-------------------------------------------------------------------------
----------------------+
```

```
  2. | famid | inc1 | inc2 | inc3 | inc4 | inc5 | inc6 | inc7 | inc8 | inc9 |
inc10 | inc11 | inc12 |
     |     2 | 4042 | 3084 | 3108 | 3150 | 3800 | 3100 | 1531 | 2914 | 3819 |
4124 |  4274 |  4471 |
     |-----------------------------------------------------------------------
-----------------------|
     | taxinc1 | taxinc2 | taxinc3 | taxinc4 | taxinc5  | taxinc6  | taxinc7
| taxinc8  | taxinc9  |
     |   404.2 |   308.4 |   310.8 |     315 |     380  |     310  |   153.1
|   291.4  |   381.9  |
     |-----------------------------------------------------------------------
-----------------------|
     |           taxinc10           |           taxinc11            |
taxinc12           |
     |              412.4           |              427.4            |
447.1            |
     +-----------------------------------------------------------------------
-----------------------+

     +-----------------------------------------------------------------------
-----------------------+
  3. | famid | inc1 | inc2 | inc3 | inc4 | inc5 | inc6 | inc7 | inc8 | inc9 |
inc10 | inc11 | inc12 |
     |     3 | 6015 | 6123 | 6113 | 6100 | 6100 | 6200 | 6186 | 6132 | 3123 |
4231 |  6039 |  6215 |
     |-----------------------------------------------------------------------
-----------------------|
     | taxinc1 | taxinc2 | taxinc3 | taxinc4 | taxinc5  | taxinc6  | taxinc7
| taxinc8  | taxinc9  |
     |   601.5 |   612.3 |   611.3 |     610 |     610  |     620  |   618.6
|   613.2  |   312.3  |
     |-----------------------------------------------------------------------
-----------------------|
     |           taxinc10           |           taxinc11            |
taxinc12           |
     |              423.1           |              603.9            |
621.5            |
     +-----------------------------------------------------------------------
-----------------------+
```

3. Computing variables (using the foreach command)

Another way to compute 12 variables representing the amount of tax paid (10%) for each month is to use the **foreach** command. In the example below we use the **foreach** command to cycle through the variables **inc1** to **inc12** and compute the taxable income as **taxinc1** - **taxinc12**.

```
foreach var of varlist inc1-inc12 {
  generate tax`var' = `var' * .10
}
```

The initial **foreach** statement tells Stata that we want to cycle through the variables **inc1** to **inc12** using the statements that are surrounded by the curly braces. The first time we cycle through the statements, the value of **var** will be **inc1** and the second time the value of **var** will be **inc2** and so on until the final iteration where the value of **var** will be **inc12**. Each statement within the loop

(in this case, just the one generate statement) is evaluated and executed. When we are inside the **foreach** loop, we can access the value of **var** by surrounding it with the funny quotation marks like this **`var'** . The ` is the quote right below the ~ on your keyborad and the ' is the quote below the " on your keyboard. The first time through the loop, **`var'** is replaced with **inc1**, so the statement

```
generate tax`var' = `var' * .10
```

becomes

```
generate taxinc1 = inc1 * .10
```

This is repeated for **inc2** and then **inc3** and so on until **inc12.** So, this **foreach** loop is the equivalent of executing the 12 **generate** statements manually, but much easier and less error prone.

4. Collapsing across variables (manually)

Often one needs to sum across variables (also known as collapsing across variables). For example, let's say the quarterly income for each observation is desired. In order to get this information, four quarterly variables **incqtr1-incqtr4** need to be computed. Again, this can be achieved manually or by using the **forea**ch command. Below is an example of how to compute 4 quarterly income variables **incqtr1-incqtr4** by simply adding together the months that comprise a quarter.

```
generate incqtr1 = inc1 + inc2 + inc3
generate incqtr2 = inc4 + inc5 + inc6
generate incqtr3 = inc7 + inc8 + inc9
generate incqtr4 = inc10+ inc11+ inc12

list incqtr1 - incqtr4
```

The output is shown below.

```
     +-----------------------------------------+
     | incqtr1   incqtr2   incqtr3   incqtr4 |
     |-----------------------------------------|
  1. |    9808      8700      9947      6534 |
  2. |   10234     10050      8264     12869 |
  3. |   18251     18400     15441     16485 |
     +-----------------------------------------+
```

5. Collapsing across variables (using the foreach command)

This same result as above can be achieved using the **foreach** command. The example below illustrates how to compute the quarterly income variables **incqtr1-incqtr4** using the **foreach** command.

```
foreach qtr of numlist 1/4 {
```

```
  local m3 = `qtr'*3
  local m2 = (`qtr'*3)-1
  local m1 = (`qtr'*3)-2
  generate incqtr`qtr' = inc`m1' + inc`m2' + inc`m3'
}
list incqtr1 - incqtr4
```

The output is shown below.

```
     +---------------------------------------+
     | incqtr1   incqtr2   incqtr3   incqtr4 |
     |---------------------------------------|
  1. |    9808      8700      9947      6534 |
  2. |   10234     10050      8264     12869 |
  3. |   18251     18400     15441     16485 |
     +---------------------------------------+
```

In this example, instead of cycling across variables, the **foreach** command is cycling across numbers, 1, 2, 3 then 4 which we refer to as **qtr** which represent the 4 quarters of variables that we wish to create. The trick is the relationship between the quarter and the month numbers that compose the quarter and to create a kind of formula that relates the quarters to the months. For example, quarter 1 of data corresponds to months 3, 2 and 1, so we can say that when the quarter (qtr) is 1 we want the months represented by qtr*3, (qtr*3)-1 and (qtr*3)-2, yielding 3, 2, and 1. This is what the statements below from the **foreach** loop are doing. They are relating the quarter to the months.

```
  local m3 = `qtr'*3
  local m2 = (`qtr'*3)-1
  local m1 = (`qtr'*3)-2
```

So, when **qtr** is 1**,** the value for **m3** is 1*3, the value for **m2** is (1*3)-1 and the value for **m1** is (1*3)-2. Then, imagine all of those values being substituted into the following statement from the **foreach** loop.

```
  generate incqtr`qtr' = inc`m1' + inc`m2' + inc`m3'
```

This then becomes

```
  generate incqtr1 = inc3 + inc2 + inc1
```

and for the next quarter (when **qtr** becomes 2) the statement would become

```
  generate incqtr2 = inc6 + inc5 + inc4
```

In this example, with only 4 quarters of data, it would probably be easier to simply write out the 4 **generate** statements manually, however if you had 40 quarters of data, then the **foreach** loop can save you considerable time, effort and mistakes.

6. Identifying patterns across variables (using the foreach command)

The **foreach** command can also be used to identify patterns across variables of a dataset. Let's say, for example, that one needs to know which months had income that was less than the income of the previous month. To obtain this information, dummy indicators can be created to indicate in which months this occurred. Note that only 11 dummy indicators are needed for a 12 month period because the interest is in the change from one month to the next. When a month has income that is less than the income of the previous month, the dummy indicators **lowinc2-lowinc12** get assigned a "1". When this is not the case, they are assigned a "0". This program is illustrated below (note for simplicity we assume no missing data on income).

```
foreach curmon of numlist 2/12 {
  local lastmon = `curmon' - 1
  generate lowinc`curmon' = 1 if ( inc`curmon' <  inc`lastmon' )
  replace  lowinc`curmon' = 0 if ( inc`curmon' >= inc`lastmon' )
}
```

We can list out the original values of **inc** and **lowinc** and verify that this worked properly

```
list famid inc1-inc12, clean noobs
```

```
famid inc1    inc2    inc3    inc4    inc5    inc6    inc7    inc8    inc9
inc10    inc11    inc12
1     3281    3413    3114    2500    2700    3500    3114    3319    3514
1282     2434     2818
2     4042    3084    3108    3150    3800    3100    1531    2914    3819
4124     4274     4471
3     6015    6123    6113    6100    6100    6200    6186    6132    3123
4231     6039     6215
```

```
list famid lowinc2-lowinc12, clean noobs
```

```
famid lowinc2 lowinc3 lowinc4 lowinc5 lowinc6 lowinc7 lowinc8 lowinc9
lowinc10 lowinc11 lowinc12
    1       0       1       1       0       0       1       0       0
1        0        0
    2       1       0       0       0       1       1       0       0
0        0        0
    3       0       1       1       0       0       1       1       1
0        0        0
```

This time we used the **foreach** loop to compare the current month, represented by **curmon,** and the prior month, computed as **`curmon'-1** creating **lastmon**. So, for the first pass through the **foreach** loop the value for **curmon** is 2 and the value for **lastmon** is 1, so the **generate** and **replace** statements become

```
generate lowinc2 = 1 if ( inc2 <  inc1 )
replace  lowinc2 = 0 if ( inc2 >= inc1 )
```

The process is repeated until **curmon** is 12, and then the **generate** and **replace** statements become

```
generate lowinc12 = 1 if ( inc12 <  inc11 )
replace  lowinc12 = 0 if ( inc12 >= inc11 )
```

If you were using **foreach** to span a large range of values (say 1/1000) then it is more effcient to use **forvalues** since it is designed to quickly increment through a sequential list, for example

```
forvalues curmon = 2/12 {
  local lastmon = `curmon' - 1
  generate lowinc`curmon' = 1 if ( inc`curmon' <  inc`lastmon' )
  replace  lowinc`curmon' = 0 if ( inc`curmon' >= inc`lastmon' )
}
```


You're not lost. We have a new look but the same content.

Stata Learning Module
Combining data

This module will illustrate how you can combine files in Stata. Examples will include appending files, one to one match merging, and one to many match merging.

Appending data files

When you have two data files, you may want to combine them by stacking them one on top of the other. For example, we have a file containing **dads** and a file containing **moms** as shown below.

```
input famid str4 name inc
2 "Art" 22000
1 "Bill" 30000
3 "Paul" 25000
end
save dads, replace
list
         famid       name        inc
```

```
  1.         2        Art      22000
  2.         1       Bill      30000
  3.         3       Paul      25000
clear
input famid str4 name inc
1 "Bess" 15000
3 "Pat" 50000
2 "Amy" 18000
end
save moms, replace
list
         famid       name        inc
  1.         1       Bess      15000
  2.         3        Pat      50000
  3.         2        Amy      18000
```

If we wanted to combine these files by stacking them one atop the other, we can use the **append** command as shown below.

```
use dads, clear
append using moms
```

We can use the **list** command to see if this worked correctly.

```
list
         famid       name        inc
  1.         2        Art      22000
  2.         1       Bill      30000
  3.         3       Paul      25000
  4.         1       Bess      15000
  5.         3        Pat      50000
  6.         2        Amy      18000
```

The **append** worked properly... the **dads** and **moms** are stacked together in one file. But, there is a little problem. We can't tell the dads from the moms. Let's try doing this again, but first we will create a variable called **momdad** in the **dads** and **moms** data file which will contain **dad** for the dads data file and **mom** for the moms data file. When we combine the two files together, the **momdad** variable will tell us who the moms and dads are.

Here we make **momdad** variable for the **dads** data file. We **save** the file calling it **dads1**.

```
use dads, clear
generate str3 momdad = "dad"
save dads1
 file dads1.dta saved
```

Here we make **momdad** variable for the **moms** data file. We **save** the file calling it **moms1**.

```
use moms, clear
generate str3 momdad = "mom"
save moms1
 file moms1.dta saved
```

Now, let's append **dads1** and **moms1** together.

```
use dads1, clear
append using moms1
```

Now, when we list the data the **momdad** variable shows who the moms and dads are.

```
list
         famid       name        inc     momdad
  1.         2        Art      22000        dad
  2.         1       Bill      30000        dad
  3.         3       Paul      25000        dad
  4.         1       Bess      15000        mom
  5.         3        Pat      50000        mom
  6.         2        Amy      18000        mom
```

Match merging

Another way of combining data files is match merging. Say that we wanted to combine the **dads** with the **faminc** data file, having the dads information and the family information side by side. We can do this with a match merge.

Let's have a look at the **dads** and **faminc** file.

```
use dads, clear
list
         famid       name        inc
  1.         2        Art      22000
  2.         1       Bill      30000
  3.         3       Paul      25000
clear
input famid faminc96 faminc97 faminc98
3 75000 76000 77000
1 40000 40500 41000
2 45000 45400 45800
end
save faminc, replace
list
         famid   faminc96   faminc97   faminc98
  1.         3      75000      76000      77000
  2.         1      40000      40500      41000
  3.         2      45000      45400      45800
```

We want to combine the data files so they look like this.

```
famid name  inc    faminc96 faminc97 faminc98
 1     Bill  30000  40000    40500    41000
 2     Art   22000  45000    45400    45800
 3     Paul  25000  75000    76000    77000
```

Notice that the **famid** variable is used to associate the observation from the **dads** file with the appropriate observation from the **faminc** file. The strategy for merging the files goes like this.

1. sort **dads** on **famid** and **save** that file (calling it **dads2**).
2. sort **faminc** on **famid** and **save** that file (calling it **faminc2**).
3. use the **dads2** file.
4. merge the **dads2** file with the **faminc2** file using **famid** to match them.

Here are those four steps.

1. Sort the **dads** file by **famid** and **save** it as **dads2**

```
use dads, clear
sort famid
save dads2
 file dads2.dta saved
```

2. Sort the **faminc** file by **famid** and **save** it as **faminc2**.

```
use faminc, clear
sort famid
save faminc2
 file faminc2.dta saved
```

3. Use the **dads2** file

```
use dads2, clear
```

4. Merge with the **faminc2** file using **famid** as the key variable.

```
merge famid using faminc2
```

It seems like this worked just fine, but what is that **_merge** variable?

```
list, nodisplay noobs
```

famid	name	inc	faminc96	faminc97	faminc98	_merge
1	Bill	30000	40000	40500	41000	3
2	Art	22000	45000	45400	45800	3
3	Paul	25000	75000	76000	77000	3

The **_merge** variable indicates, for each observation, how the merge went. This is useful for identifying mismatched records. **_merge** can have one of three values
1 - The record contains information from file1 only (e.g., a **dad2** record with no corresponding **faminc2** record.
2 - The record contains information from file2 only (e.g., a **faminc2** record with no corresponding **dad2** record.
3 - The record contains information from both files (e.g., the **dad2** and **faminc2** records matched up).

When you have many records, tabulating **_merge** is very useful to summarize how many mismatched you have. In our case, all of the records match so the value for **_merge** was always 3.

```
tabulate _merge
     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          3 |          3      100.00      100.00
------------+-----------------------------------
      Total |          3      100.00
```

One-to-many match merging

Another kind of merge is called a **one to many** merge. Our **one to one** merge matched up **dads** and **faminc** and there was a **one to one** matching of the files. If we merge **dads** with **kids**, there can be multiple kids per dad and hence this is a **one to many** merge.

As you see below, the strategy for the **one to many** merge is really the same as the **one to one** merge.

1. sort **dads** on **famid** and **save** that file as **dads3**
2. sort **kids** on **famid** and **save** that file as **kids3**
3. use the **dads3** file
4. merge the **dads3** file with the **kids3** file using **famid** to match them.

The 4 steps are shown below.

1. Sort the **dads** data file on **famid** and **save** that file as **dads3**.

```
use dads, clear
sort famid
save dads3
 file dads3.dta saved
list
         famid       name        inc
  1.         1       Bill      30000
  2.         2        Art      22000
  3.         3       Paul      25000
```

2. Sort the **kids** data file on **famid** and **save** that file as **kids3**.

```
clear
input famid str4 kidname birth age wt str1 sex
1 "Beth" 1 9 60 "f"
2 "Andy" 1 8 40 "m"
3 "Pete" 1 6 20 "f"
1 "Bob" 2 6 80 "m"
1 "Barb" 3 3 50 "m"
2 "Al" 2 6 20 "f"
2 "Ann" 3 2 60 "m"
3 "Pam" 2 4 40 "f"
3 "Phil" 3 2 20 "m"
end

sort famid
```

```
save kids3
file kids3.dta saved

list
        famid    kidname      birth        age         wt        sex
  1.        1       Beth          1          9         60          f
  2.        1        Bob          2          6         40          m
  3.        1       Barb          3          3         20          f
  4.        2       Andy          1          8         80          m
  5.        2         Al          2          6         50          m
  6.        2        Ann          3          2         20          f
  7.        3       Pete          1          6         60          m
  8.        3        Pam          2          4         40          f
  9.        3       Phil          3          2         20          m
```

3. Use the **dads3** file.

```
use dads3, clear
```

4. Merge the **dads3** file with the **kids3** file using **famid** to match them.

```
merge famid using kids3
```

Let's list out the results.

```
list famid name kidname birth age _merge
       famid   name   kidname   birth   age   _merge
  1.       1   Bill      Barb       3     3        3
  2.       2    Art        Al       2     6        3
  3.       3   Paul       Pam       2     4        3
  4.       1   Bill       Bob       2     6        3
  5.       1   Bill      Beth       1     9        3
  6.       2    Art      Andy       1     8        3
  7.       2    Art       Ann       3     2        3
  8.       3   Paul      Phil       3     2        3
  9.       3   Paul      Pete       1     6        3
```

The results are a bit easier to read if we sort the data on **famid** and **birth**.

```
sort famid birth
list famid name kidname birth age _merge
        famid       name    kidname      birth        age    _merge
  1.        1       Bill       Beth          1          9         3
  2.        1       Bill        Bob          2          6         3
  3.        1       Bill       Barb          3          3         3
  4.        2        Art       Andy          1          8         3
  5.        2        Art         Al          2          6         3
  6.        2        Art        Ann          3          2         3
  7.        3       Paul       Pete          1          6         3
  8.        3       Paul        Pam          2          4         3
  9.        3       Paul       Phil          3          2         3
```

As you see, this is basically the same as a **one to one** merge. You may wonder if the order of the files on the merge statement is relevant. Here, we switch the order of the files and the results are the same. The only difference is the order of the records after the merge.

```
use kids3, clear
merge famid using dads3
list famid name kidname birth age
```

	famid	name	kidname	birth	age
1.	1	Bill	Beth	1	9
2.	1	Bill	Bob	2	6
3.	1	Bill	Barb	3	3
4.	2	Art	Andy	1	8
5.	2	Art	Al	2	6
6.	2	Art	Ann	3	2
7.	3	Paul	Pete	1	6
8.	3	Paul	Pam	2	4
9.	3	Paul	Phil	3	2

Summary

Appending data example

```
use dads, clear
append using moms
```

Match merge example steps (one-to-one and one-to-many)

```
1. sort dads on famid and save that file
2. sort kids on famid and save that file
3. use the dads file
4. merge the dads file with the kids file using  famid to match them.
```

Match merge example program

```
use dads, clear
sort famid
save dads2

use faminc, clear
sort famid
save faminc2

use dads2, clear
merge famid using faminc2
```

You're not lost. We have a new look but the same content.

Stata Learning Module
Reshaping data wide to long

This module illustrates the power (and simplicity) of Stata in its ability to reshape data files. These examples take **wide** data files and reshape them into **long** form. These show common examples of reshaping data, but do not exhaustively demonstrate the different kinds of data reshaping that you could encounter.

Example #1: Reshaping data wide to long

Consider the family income data file below.

```
use http://www.ats.ucla.edu/stat/stata/modules/faminc, clear
list
         famid   faminc96   faminc97   faminc98
  1.         3      75000      76000      77000
  2.         1      40000      40500      41000
  3.         2      45000      45400      45800
```

This is called a **wide** format since the years of data are wide. We may want the data to be **long**, where each year of data is in a separate observation. The **reshape** command can accomplish this, as shown below.

```
reshape long faminc, i(famid) j(year)
(note:  j = 96 97 98)

Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                        3   ->       9
Number of variables                   4   ->       3
j variable (3 values)                     ->   year
xij variables:
             faminc96 faminc97 faminc98   ->   faminc
-----------------------------------------------------------------------------
```

The **list** command shows that the data are now in **long** form, where each **year** is represented as its own observation.

```
list
         famid       year     faminc
  1.         1         96      40000
  2.         1         97      40500
  3.         1         98      41000
  4.         2         96      45000
  5.         2         97      45400
  6.         2         98      45800
  7.         3         96      75000
  8.         3         97      76000
```

```
  9.         3         98      77000
```

Let's look at the **wide** format and contrast it with the **long** format.

The **reshape wide** command puts the data back into **wide** format. We then list out the **wide** file.

```
reshape wide
(note:  j = 96 97 98)

Data                                long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                         9   ->       3
Number of variables                    3   ->       4
j variable (3 values)               year   ->   (dropped)
xij variables:
                                  faminc   ->   faminc96 faminc97 faminc98
-----------------------------------------------------------------------------
list
         famid   faminc96   faminc97   faminc98
  1.         1      40000      40500      41000
  2.         2      45000      45400      45800
  3.         3      75000      76000      77000
```

The **reshape long** command puts the data back into **long** format. We then list out the **long** file.

```
reshape long
(note:  j = 96 97 98)

Data                                wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                         3   ->       9
Number of variables                    4   ->       3
j variable (3 values)                      ->   year
xij variables:
             faminc96 faminc97 faminc98   ->   faminc
-----------------------------------------------------------------------------
list
         famid       year     faminc
  1.         1         96      40000
  2.         1         97      40500
  3.         1         98      41000
  4.         2         96      45000
  5.         2         97      45400
  6.         2         98      45800
  7.         3         96      75000
  8.         3         97      76000
  9.         3         98      77000
```

Now let's look at the pieces of the original **reshape** command.

```
reshape long faminc, i(famid) j(year)
```

long tells reshape that we want to go from **wide** to **long**
faminc tells Stata that the **stem** of the variable to be converted from **wide** to **long** is **faminc**

i(famid) option tells reshape that **famid** is the unique identifier for records in their **wide** format **j(year)** tells reshape that the suffix of **faminc** (i.e., 96 97 98) should be placed in a variable called **year**

Example #2: Reshaping data wide to long

Consider the file containing the kids and their heights at 1 year of age (ht1) and at 2 years of age (ht2).

```
use http://www.ats.ucla.edu/stat/stata/modules/kidshtwt, clear
list famid birth ht1 ht2
          famid      birth        ht1        ht2
  1.          1          1        2.8        3.4
  2.          1          2        2.9        3.8
  3.          1          3        2.2        2.9
  4.          2          1          2        3.2
  5.          2          2        1.8        2.8
  6.          2          3        1.9        2.4
  7.          3          1        2.2        3.3
  8.          3          2        2.3        3.4
  9.          3          3        2.1        2.9
```

Lets reshape this data into a long format. The critical questions are:
Q: What is the stem of the variable going from **wide** to **long**.
A: The stem is **ht**
Q: What variable uniquely identifies an observation when it is in the **wide** form.
A: **famid** and **birth** together uniquely identify the **wide** observations.
Q: What do we want to call the variable which contains the suffix of **ht**, i.e., 1 and 2.
A: Lets call the suffix **age**.

With the answers to these questions, the reshape command will look like this.

```
reshape long ht, i(famid birth) j(age)
```

Let's look at the **wide** data, and then the data reshaped to be **long**.

```
list famid birth ht1 ht2
         famid      birth        ht1        ht2
  1.         1          1        2.8        3.4
  2.         1          2        2.9        3.8
  3.         1          3        2.2        2.9
  4.         2          1          2        3.2
  5.         2          2        1.8        2.8
  6.         2          3        1.9        2.4
  7.         3          1        2.2        3.3
  8.         3          2        2.3        3.4
  9.         3          3        2.1        2.9
reshape long ht, i(famid birth) j(age)
(note:  j = 1 2)

Data                               wide   ->   long
-----------------------------------------------------------------------------
```

```
Number of obs.                        9   ->      18
Number of variables                   7   ->       7
j variable (2 values)                     ->   age
xij variables:
                               ht1 ht2   ->   ht
-----------------------------------------------------------------------------
```

```
list famid birth age ht
```

	famid	birth	age	ht
1.	1	1	1	2.8
2.	1	1	2	3.4
3.	1	2	1	2.9
4.	1	2	2	3.8
5.	1	3	1	2.2
6.	1	3	2	2.9
7.	2	1	1	2
8.	2	1	2	3.2
9.	2	2	1	1.8
10.	2	2	2	2.8
11.	2	3	1	1.9
12.	2	3	2	2.4
13.	3	1	1	2.2
14.	3	1	2	3.3
15.	3	2	1	2.3
16.	3	2	2	3.4
17.	3	3	1	2.1
18.	3	3	2	2.9

Example #3: Reshaping data wide to long

The file with the kids heights at **age** 1 and **age** 2 also contains their weights at **age** 1 and **age** 2 (called **wt1** and **wt2**).

```
use http://www.ats.ucla.edu/stat/stata/modules/kidshtwt, clear
list famid birth ht1 ht2 wt1 wt2
```

	famid	birth	ht1	ht2	wt1	wt2
1.	1	1	2.8	3.4	19	28
2.	1	2	2.9	3.8	21	28
3.	1	3	2.2	2.9	20	23
4.	2	1	2	3.2	25	30
5.	2	2	1.8	2.8	20	33
6.	2	3	1.9	2.4	22	33
7.	3	1	2.2	3.3	22	28
8.	3	2	2.3	3.4	20	30
9.	3	3	2.1	2.9	22	31

Let's reshape this data into a **long** format. This is basically the same as the previous command except that **ht** is replaced with **ht wt**.

```
reshape long ht wt, i(famid birth) j(age)
```

Let's look at the **wide** data, and then the data reshaped to be **long**.

```
list famid birth ht1 ht2 wt1 wt2
```

	famid	birth	ht1	ht2	wt1	wt2

```
  1.         1          1        2.8        3.4         19         28
  2.         1          2        2.9        3.8         21         28
  3.         1          3        2.2        2.9         20         23
  4.         2          1          2        3.2         25         30
  5.         2          2        1.8        2.8         20         33
  6.         2          3        1.9        2.4         22         33
  7.         3          1        2.2        3.3         22         28
  8.         3          2        2.3        3.4         20         30
  9.         3          3        2.1        2.9         22         31
reshape long ht wt, i(famid birth) j(age)
(note:  j = 1 2)

Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                        9   ->      18
Number of variables                   7   ->       6
j variable (2 values)                     ->   age
xij variables:
                                ht1 ht2   ->   ht
                                wt1 wt2   ->   wt
-----------------------------------------------------------------------------
list famid birth age ht wt
          famid      birth        age         ht         wt
  1.         1          1          1        2.8         19
  2.         1          1          2        3.4         28
  3.         1          2          1        2.9         21
  4.         1          2          2        3.8         28
  5.         1          3          1        2.2         20
  6.         1          3          2        2.9         23
  7.         2          1          1          2         25
  8.         2          1          2        3.2         30
  9.         2          2          1        1.8         20
 10.         2          2          2        2.8         33
 11.         2          3          1        1.9         22
 12.         2          3          2        2.4         33
 13.         3          1          1        2.2         22
 14.         3          1          2        3.3         28
 15.         3          2          1        2.3         20
 16.         3          2          2        3.4         30
 17.         3          3          1        2.1         22
 18.         3          3          2        2.9         31
```

Example #4: Reshaping data wide to long with character suffixes

It also is possible to reshape a wide data file to be long when there are character suffixes. Look at the **dadmomw** file below.

```
use http://www.ats.ucla.edu/stat/stata/modules/dadmomw, clear
list
         famid      named       incd      namem       incm
  1.         1       Bill      30000       Bess      15000
  2.         2        Art      22000        Amy      18000
  3.         3       Paul      25000        Pat      50000
```

We would like to make **name** and **inc** into **long** formats but their suffixes are characters (d & m) instead of numbers. Stata can handle that as long as you use **string** in the command to indicate that the suffix is a character.

```
reshape long name  inc, i(famid) j(dadmom) string
```

Let's look at the data before and after reshaping.

```
list
         famid      named       incd      namem       incm
  1.         1       Bill      30000       Bess      15000
  2.         2        Art      22000        Amy      18000
  3.         3       Paul      25000        Pat      50000
reshape long name inc, i(famid) j(dadmom) string
(note:  j = d m)

Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                        3   ->       6
Number of variables                   5   ->       4
j variable (2 values)                     ->   dadmom
xij variables:
                            named namem   ->   name
                              incd incm   ->   inc
-----------------------------------------------------------------------------
list
         famid     dadmom       name        inc
  1.         1          d       Bill      30000
  2.         1          m       Bess      15000
  3.         2          d        Art      22000
  4.         2          m        Amy      18000
  5.         3          d       Paul      25000
  6.         3          m        Pat      50000
```

Summary reshaping data wide to long

```
 Wide format
          famid   faminc96   faminc97   faminc98
  1.         1      40000      40500      41000
  2.         2      45000      45400      45800
  3.         3      75000      76000      77000
```

```
reshape long faminc, i(famid) j(year)
```

```
 Long Format
         famid       year     faminc
  1.         1         96      40000
  2.         1         97      40500
  3.         1         98      41000
  4.         2         96      45000
  5.         2         97      45400
  6.         2         98      45800
  7.         3         96      75000
  8.         3         97      76000
```

```
  9.         3         98      77000
```

The general syntax of **reshape long** can be expressed as...

```
reshape long stem-of-wide-vars, i(wide-id-var)  j(var-for-suffix)
```

where

```
stem-of-wide-vars  is the stem of the wide variables, e.g., faminc
wide-id-var        is the variable that uniquely identifies wide
                   observations, e.g., famid
var-for-suffix     is the variable that will contain the suffix of
                   the wide variables, e.g., year
```


Stata Learning Module
Reshaping data long to wide

This module illustrates the power (and simplicity) of Stata in its ability to reshape data files. These examples take **long** data files and reshape them into **wide** form. These examples cover some common examples, but this is only part of the features and options of the Stata **reshape** command.

Example #1: Reshaping data long to wide

The reshape command can be used to make data from a **long** format to a **wide** format. Consider the **kids** file (to make things simple at first, we will drop the variables **kidname, sex** and **wt**).

```
use kids, clear

drop  kidname sex wt

list
```

```
          famid      birth        age
  1.         1          1          9
  2.         1          2          6
  3.         1          3          3
  4.         2          1          8
  5.         2          2          6
  6.         2          3          2
  7.         3          1          6
  8.         3          2          4
  9.         3          3          2
```

Let's make **age** in this file wide, making one record per family which would contain **age1 age2 age3**, the ages of the kids in the family (**age2** would be missing if there is only one kid, and **age3** would be missing if there are only two kids). Let's look at the data before and after reshaping.

```
list
```

```
          famid      birth        age
  1.         1          1          9
  2.         1          2          6
  3.         1          3          3
  4.         2          1          8
  5.         2          2          6
  6.         2          3          2
  7.         3          1          6
  8.         3          2          4
  9.         3          3          2
```

```
reshape wide age, i(famid)  j(birth)

(note:  j = 1 2 3)

Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                        9   ->       3
Number of variables                   3   ->       4
j variable (3 values)             birth   ->   (dropped)
xij variables:
                                    age   ->   age1 age2 age3
-----------------------------------------------------------------------------
```

```
list
```

```
          famid       age1       age2       age3
  1.         1          9          6          3
  2.         2          8          6          2
  3.         3          6          4          2
```

Let's look at the pieces of the **reshape** command.

```
reshape wide age, j(birth) i(famid)
```

wide tells reshape that we want to go from long to wide
age tells Stata that the variable to be converted from long to wide is **age**
i(famid) tells reshape that **famid** uniquely identifies observations in the wide form
j(birth) tells reshape that the suffix of **age** (1 2 3) should be taken from the variable **birth**

Example #2: Reshaping data long to wide with more than one variable

The reshape command can work on more than one variable at a time. In the example above, we just reshaped the **age** variable. In the example below, we reshape the variables age, wt and sex like this

```
reshape wide age wt sex,  i(famid) j(birth)
```

Let's look at the data before and after reshaping.

```
use kids, clear
```

```
list
```

```
          famid    kidname      birth         age          wt         sex
  1.          1       Beth          1           9          60           f
  2.          1        Bob          2           6          40           m
  3.          1       Barb          3           3          20           f
  4.          2       Andy          1           8          80           m
  5.          2         Al          2           6          50           m
  6.          2        Ann          3           2          20           f
  7.          3       Pete          1           6          60           m
  8.          3        Pam          2           4          40           f
  9.          3       Phil          3           2          20           m
```

```
reshape wide kidname age wt sex, i(famid) j(birth)
```

```
 (note:  j = 1 2 3)

Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                        9   ->       3
Number of variables                   6   ->      13
j variable (3 values)             birth   ->   (dropped)
xij variables:
                                kidname   ->   kidname1 kidname2 kidname3
                                    age   ->   age1 age2 age3
                                     wt   ->   wt1 wt2 wt3
                                    sex   ->   sex1 sex2 sex3
-----------------------------------------------------------------------------
```

```
list
```

```
 Observation 1

       famid             1    kidname1          Beth         age1             9
         wt1            60        sex1             f     kidname2           Bob
        age2             6          wt2            40         sex2             m
    kidname3          Barb         age3             3          wt3            20
        sex3             f


Observation 2

       famid             2    kidname1          Andy         age1             8
         wt1            80        sex1             m     kidname2            Al
        age2             6          wt2            50         sex2             m
    kidname3           Ann         age3             2          wt3            20
        sex3             f


Observation 3

       famid             3    kidname1          Pete         age1             6
```

```
        wt1            60        sex1             m    kidname2           Pam
       age2             4         wt2            40        sex2             f
    kidname3          Phil        age3             2         wt3            20
       sex3             m
```

Example #3: Reshaping wide with character suffixes

The examples above showed how to reshape data using numeric suffixes, but **reshape** can handle character suffixes as well. Consider the **dadmoml** data file shown below.

```
use dadmoml, clear
```

```
list
```

```
          famid       name        inc     dadmom
  1.         2        Art      22000        dad
  2.         1       Bill      30000        dad
  3.         3       Paul      25000        dad
  4.         1       Bess      15000        mom
  5.         3        Pat      50000        mom
  6.         2        Amy      18000        mom
```

Let's reshape this to be in a wide format, containing one record per family. The **reshape** command below uses **string** to tell reshape that the suffix is character.

```
reshape wide name inc,  i(famid) j(dadmom) string
```

Let's look at the data before and after reshaping.

```
list
```

```
          famid       name        inc     dadmom
  1.         2        Art      22000        dad
  2.         1       Bill      30000        dad
  3.         3       Paul      25000        dad
  4.         1       Bess      15000        mom
  5.         3        Pat      50000        mom
  6.         2        Amy      18000        mom
```

```
reshape wide name inc, i(famid) j(dadmom) string
```

```
(note:  j = dad mom)

Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                        6   ->       3
Number of variables                   4   ->       5
j variable (2 values)            dadmom   ->   (dropped)
xij variables:
                                   name   ->   namedad namemom
                                    inc   ->   incdad incmom
-----------------------------------------------------------------------------
```

```
list
```

```
          famid    namedad     incdad    namemom     incmom
  1.          1       Bill      30000       Bess      15000
  2.          2        Art      22000        Amy      18000
  3.          3       Paul      25000        Pat      50000
```

Summary

Reshaping data long to wide

```
 Long format
        famid      birth          age
  1.          1          1            9
  2.          1          2            6
  3.          1          3            3
  4.          2          1            8
  5.          2          2            6
  6.          2          3            2
  7.          3          1            6
  8.          3          2            4
  9.          3          3            2
```

```
reshape wide age, j(birth) i(famid)
```

```
 Wide format
         famid       age1       age2       age3
  1.          1          9          6          3
  2.          2          8          6          2
  3.          3          6          4          2
```

The general syntax of **reshape wide** can be expressed as:

```
reshape wide long-var(s),  i( wide-id-var ) j( var-with-suffix )
where
long-var(s)      is the name of the long variable(s) to be made wide e.g. age
wide-id-var      is the variable that uniquely identifies wide
                 observations, e.g. famid
var-with-suffix  is the variable from the long file that contains
                 the suffix for the wide variables, e.g. age
```