Working with large data files

Stata requires that the data file you want to analyze fits into memory. This means that working with files approaching the size of memory on your computer can be a challenge. Fortunately, Stata has supplied a number of nice tools for dealing with large data files. We review them here.

describe using

Sometimes you may just want to see what variables are in the large file. You don't need to **use** the entire file just to see a list of variables and their labels. Instead, you can type

```
describe using "bigfile.dta"
```

where "bigfile.dta" is that name of the file you want to describe. Stata will give you all the information about the variables that you would expect from the describe command. Ideally, you'll be able to select a subset of variables, or a subset of observations, just by looking at describe.

lookfor and lookfor_all

If the big file has a lot of variables, the **describe using** command will give you a lot of text to search. The lookfor command will search the variable names and labels for any character string you supply and list the variable names/labels containing that string. If you have several files to search, try **lookfor_all**. This command is available from the SSC archives. It searches through all Stata data files in the current directory (and its subdirectories if you ask for it) for any string you want to find. The string may be in the variable name or label. For example, you may want to find the variable containing the sampling weight, so you try searching for the string "weight". First, change directories (**cd**) to the directory containing the file or files you want to search, then "lookfor" the string:

```
cd "c:\big_file_directory"
lookfor_all weight, subdir
```

The command lists the name of each file containing that string along with the names of all the variables containing that string in their name or label. It then gives you a clickable link to each file with a match. This command has lots of nice features. See **help lookfor_all** at CPC, or you can download it to your standalone computer with **ssc install lookfor_all**.

use list_of_variables using

You can bring a subset of variables from bigfile.dta into memory using this form of the **use** command:

```
use list_of_variables using "bigfile.dta"
```

After looking at the results of **describe using** or **lookfor**, decide which variables you need for your analysis, and list them in the **use** command.

use in

You can bring in a small sample of observations from a large file with this version of the command:

```
use in 1/20 using "bigfile.dta"
```

This allows you to look at a sample of the variables more carefully, perhaps learning more than you could glean from the **describe** command.

use if

Suppose you're only interested in studying people in a certain age range.

```
use if age >= 1 & age < 5 using "bigfile.dta"
```

Of course, you can combine any or all of these features in the same command.

random sample

You might want to test your model on a small number of observations. Selecting those observations randomly can help you get a somewhat more representative set than selecting those from the beginning of the file, for example. You can use the **runiform** function to select any percent of observations you choose. The function returns a value between 0 and 1, so to get a 10% sample, you might use observations when runiform returns values between 0 and 0.1, or any other range of length 0.1, like this:

```
use if inrange(runiform(),0,.1) using "bigfile.dta"
```

Review Again?

Another topic?

A simple program

Writing a simple program

The term "program" has a special meaning in Stata. It is a set of commands that starts with **program define** and ends with **end**. In between you can put any of the commands you're used to using in Stata, and you can use many other commands that are specific to programs. These are discussed in detail in the PDF document "Stata Programming Reference Manual".

The purpose of this page is to give you a simple example of a program and to show you the power of a program to save you time and effort.

Suppose that you want to add household income and assets, variables in the household-level data, to each individual in the household. Using the methods discussed in One-to-many merging you would merge the household data onto the individual data using houseid as the merge key. Now let's suppose you want to do this for 50 countries to look at time and regional differences. It would be cumbersome to copy and paste the code 50 times. But more importantly, if you decided to add some code, you'd have to add it 50 times.

Instead, you can write a program and run it 50 times. Each time you run it you only need to change the name of the country and the year of the survey. If you want to add a command, you only add it once, and it is automatically run on all 50 countries when you run the program. Here's an example:

```
capture program drop mergedata
program define mergedata
   use "c:/data/`1'/`2'/individual.dta",clear
   merge m:1 houseid using "c:/data/`1'/`2'/household.dta", ///
      keepusing(houseid hhincome assets)
   drop if _merge == 2 // households without individuals in sample
   drop _merge
   save "c:/data/`1'/`2'/merged.dta", replace
end

mergedata Peru 2000
mergedata Peru 2005
mergedata "Costa Rica" 2001
mergedata "Costa Rica" 2004
mergedata Brazil 1998
mergedata Brazil 2003
mergedata Brazil 2008
 (etc.)
```

Questions:

1. Which part is the "program"? Answer.

2. What does "capture program drop mergedata" do? Answer.

3. There are numbers "1" and "2" scattered around the program. What do they do? Answer.

4. So how do I send values to these local macros `1' and `2'? Answer.

5. Why does "Costa Rica" have quotes around it? Answer.

6. The slashes ("/") in the paths are backward from the way I usually type them in Windows. Is that necessary? Answer.

7. How to I run a program? Answer.

8. What if I have an error in my program? How do I see what values went into `1' and `2'? Answer.

Answers:

1. Each command between and including **program define mergedata** and **end** is part of the program. Here we've chosen to name the program "mergedata", but you can name it almost anything. It's best to avoid names that are already taken, like "merge". You can check whether **help** turns up a command when selecting a name for your program, and then you'll know you need to try a different name.

Back to question

---

2. The **capture** command allows you to give a command that might otherwise fail and continue anyway. In this case, the command is **program drop mergedata**. If there is no program in memory already called "mergedata", the command to drop the program from memory will fail. By capturing the error message from that situation, you can continue working.

Back to question

---

3. The numbers **`1'** and **`2'** are called "local macros". They are temporary variables that hold values you send to them. The first value you send goes into `1' and the second value you send goes into `2'. You can have many more local macros numbered `3', `4', etc. if you need them.

Notice that these numbers have a backward apostrophe to their left side. This character is on the key in the upper left corner of the English keyboard along with the tilda (~) character. The character to the right of the number is an apostrophe (also called "single quote"), which shares a key with the quote (").

This is one way to send values to a program, but not the only way. Others are discussed in the programming reference.

Back to question

---

4. The lines starting with **mergedata Peru 2000** that come after **end** are the commands to run the program. The first word is the new command **mergedata** which we defined above it. Following the command are the two values we want to send to the local macros `1' and `2'.

Back to question

---

5. "Costa Rica" has quotes because it has a space in the middle. We want **mergedata** to understand that the value going into local macro `1' is two words. Without the quotes, "Costa" will go into `1', "Rica" will go into `2', and 2001 will have nowhere to go because we didn't put a `3' in our program. Back to question.

---

6. Stata doesn't normally care which way you type the slashes. In this case, though, where we have a local macro in the file path, we must type the slashes as shown in the example.

Back to question

---

7. To run your program you need to first define it to Stata, that is highlight and execute the commands from **capture program drop** through **end**. You will see the commands echoed in the Results Window, but you won't otherwise get any indication from Stata that it has stored your program in memory. Next, you can highlight and execute one line at a time that calls the program (in this case the commands like **mergedata Peru 2000**) so you can check the results one merge at a time. Or, if you're feeling very self-confident, you can execute all 50 of them at once!

Back to question

---

8. Debugging a program can be tricky. The best tool available is **set trace on** in combination with **set tracedepth**

```
    set tracedepth 1
    set trace on
```

set trace on shows each of your commands followed by the values that were substituted in your local macros. It takes a minute to figure out how to read this, but it's your best tool and well worth that minute of staring at it.

set tracedepth 1 (its smallest value) allows you to see only the substitutions of values in your program. If you forget to set tracedepth, you'll see deep down into all the commands you're calling as the substitutions scroll by in the Results Window. It's slow and not particularly useful.

Put the **set tracedepth** command anywhere before the set trace command. You can put the **set trace** command in front of the program, or in front of your first line that calls the program, to have it apply to all lines of the program. If you have a long program and know approximately where your problem is, you can turn trace on for just the one or two lines of code that you think have an error, then turn it off again like this:

```
set trace on
   merge m:1 houseid using ...
set trace off
```

That way you have fewer lines of code to sift through in your Results Window.

Once you figure out how to fix your program, remember to **set trace off** so you don't have to look at the extra stuff scrolling by in the Results Window.

Back to question

Review again?

Another topic?

Adding summary statistics to a data file

Adding a statistic to each observation, reducing the file to summary statistics.

Stata has commands that allow you to either add a summary statistic to each observation in memory, or to reduce the file according to values of a group so that each resulting observation is the summary statistic for that group.

```
clear
use "q:\utilities\statatut\exampfac.dta"

/* Add the mean facility age to each observation. */

egen mage= mean(age)
su mage
list facid age mage in 1/5

/* Create a file containing the mean facility age by authority. */

collapse (mean) age, by(authorit)
list
```

Stata offers a large number of statistics with both the egen and collapse commands. See the manual or the on-line help for a full list.

Questions:

1. The **egen** command adds a new variable, in this case called mage, to every observation in the data. What happens to the original observations and variables? Answer.

2. The name "egen" stands for "extensions to generate." What's the difference between **generate** and **egen**? Answer.

3. The **collapse** command calculated the mean age for each value of authority. How many observations and variables did the resulting data file in memory contain? Answer.

Answers:

1. The original observations and variables are unchanged. The egen command simply adds another column to the data in memory.

Back to question

---

2. The **generate** command has functions, such as "log" below, to create unique values on each observation. The **egen** command has a different set of functions. Some of its functions put unique values on each observation, while others put summary statistics across all observations (or groups) on each observation.

For example, the following generate command would calculate the natural log of the age of each facility in exampfac.dta and add that value to each facility's record:

```
generate lage= log(age)
```

In contrast, the following egen command would calculate the median age of all facilities of each type (authorit), and it would add that value to each facility's record:

```
sort authorit
by authorit: egen medage= median(age)
```

By the way, you can use **bysort** to combine the above two commands and reduce your typing:

```
bysort authorit: egen medage= median(age)
```

See the manual or online help for **generate**, **egen**, and **functions**

for more information.

Back to question

---

3. After the collapse command, the resulting file had 13 observations and 2 variables. The number of observations is determined by the number of distinct values in the by variable. The number of variables is: one for each summary statistic calculated, and one for each by variable.

Back to question

Review again?

Another topic?

Analyzing data from sample surveys

A sample survey is conducted to obtain information about the characteristics of a population. To reduce the cost and time necessary to collect the data, this task is often handled by selecting a subset (a sample) from the target population of interest to the researchers. The sample design adds certain characteristics to the data that may bias the analysis. The general term for this potential bias is the "sample survey design effect." Methods are available to adjust the analysis to account for some of these characteristics. There should be variables for each observation in your data set to identify each of the characteristics that are described next.

Stata has a well-developed and straightforward set of commands that allow you to adjust your analysis for the sample survey design effect. This section of the tutorial discusses how to use these survey data commands. The discussion is divided into:

- Data Characteristics
- Choosing the Correct Weight Syntax
- Commands to Analyze Survey Data
- Logistic Regression Analysis
- Common Errors and How to Avoid Them

Most of the information in this section on analyzing survey data was provided by Kim Chantala. However, please direct questions and comments to Phil Bardsley as noted below.

Another topic?

Appending data files

Adding observations with the same variables

The need to append doesn't arise often in surveys. Usually it is needed during data entry when new questionnaires arrive in the survey office in batches, are entered, and the resulting files must be combined with files from previous batches of surveys.

For this example we'll simulate the situation in which a new batch of questionnaires has been entered. Facility data from 19 of 20 regions have been entered and combined in one file, named fac19.dta. We've received the remaining facilities' data and need to append it to the original 19.

Copy these commands into a do-file editor and run them.

```
/* Use the original facility file with 19 regions */

clear
use "q:\utilities\statatut\fac19.dta"
tabulate region

/* Append the file with one remaining region (number 7) */

append using "q:\utilities\statatut\newfac.dta"
tabulate region
```

Questions:

1. Why don't we need to sort before appending? Answer.

2. Several notes appear in the Stata log about labels already being defined. Is this a problem? Answer.

3. When using append, the two files should have identical variable names. How would I know if, by mistake, a variable in one file had a different name from a variable in the other file being appended? Answer.

Answers:

1. We don't sort before appending because we do not need to match on any identifiers. We use append when we want to add new observations, so no matching is involved.

Back to question

---

2. No, these notes do not indicate a problem with the append. They mean that the file in memory has label definitions with the same names as those in the "using" file on disk. We expect that, since both files have identical variables and, therefore, identical labels defined for their values.

Back to question

---

3. The only way to catch this situation is to **describe** both files before the append. They should have the same number of variables. After the append command, the resulting file should also have that number of variables. If the files have the same number before but one more variable after the append, you know that a variable in one of the input files had a different name.

Back to question

Review again?

Another topic?

Article

Labor-Saving Techniques

Stata offers several techniques that will save you time and reduce the chances that you'll make a mistake in your code out of sheer boredom. The techniques discussed here are:

- Looping Over Variables and Values
- Dummy variables
- A Simple Program

Changing data values

Replacing values, recoding variables, editing data, labelling data.

If you haven't read about relational, logical, and arithmetic operators in the previous page on Groups and Subsets of Data, click here to read a brief summary.

We'll continue using the 1999 Tanzania Facility Survey data.

```
   clear
   use "q:\utilities\statatut\exampfac.dta"

/* as a precaution, make a copy of the variable you want to recode */

   gen factype2=factype

/* change 8 values of factype2 into 4 */

   replace factype2= 1 if factype2 >= 2 & factype2 <= 4
   replace factype2= 2 if factype2 == 5
   replace factype2= 3 if factype2 == 6
   replace factype2= 4 if factype2 == 7 | factype2 == 9

/* the recode command does this more efficiently */

/* recode factype2 2/4=1 5=2 6=3 7 9=4 */

/* change a variable's name */

   rename factype2 type

/* give the new variable some labels */

   label variable type "Recoded facility type"
/* Click here for more on the label variable command **/

   label define fac 1 "Hospital" 2 "HealthCenter" 3 "Dispensary" 4 "Other"
   label values type fac
/* Click here for more on the label value command **/

/* the data file already has a label, but this is how it was done: */

   label data "1999 Tanzania Facility Survey"

/* delete the original factype variable */

   drop factype

/* delete observations with missing facility type */

   drop if missing(type)
   browse
   edit
```

Questions:

1. The **replace** command changes specific values in an existing variable. What would happen if you reversed the order of these replace commands:

```
replace factype2= 4 if factype2 == 7 | factype2 == 9
replace factype2= 3 if factype2 == 6
replace factype2= 2 if factype2 == 5
replace factype2= 1 if factype2 >= 2 & factype2 <= 4
```

Answer.

2. What happens to missing values of factype2? Answer.

3. How many variable names can you change with one **rename** command? Answer.

4. The **label values** command assigns a set of labels to the values of one variable. If I have several variables needing the same value labels, can I define one label and assign it to all the variables? Answer.

5. How can I change existing variable and value labels? Answer.

6. What if I want to **drop** most of my variables? That's a lot of typing! Answer.

7. What is the difference between **browse** and **edit**? Answer.

Answers:

1. All observations would have a value of 1 for factype2. Remember that each Stata command is executed for every observation before the next command is executed. The **recode** command makes all these changes in a single command, so the order of the changes doesn't matter.

Back to question.

---

2. We didn't change the missing values in either the replace or recode examples, so they remain missing.

Back to question.

---

3. Only one. Later we'll see how to rename or make just about any other change to a lot of variables easily using the **foreach** command.

Back to question.

---

4. Yes, for example the last 10 variables, pill-natural, have a yes/no answer format:

```
label def yesno 1 "Yes" 2 "No"
label val pill yesno
label val inject yesno
etc.
label val natural yesno
```

Back to question.

---

5. To change a variable label, simply type a new **label var** statement:

```
label var factype "Type of facility"
```

To change a value label, you have two choices:

- drop the existing label and recreate it with the changes, or
- create a new label and reassign the variable to the new label.

The **describe** command shows you which variables have value labels attached to them and the names of the labels. Using that command, we see that the variable urbrur has the label urb. Here's how to drop urb and recreate it with new values:

```
label drop urb
label def urb 1 "RURAL" 2 "URBAN" 3 "MIXED"
label val urbrur urb
```

Back to question.

---

6. Sometimes it's easier to use the **keep** or the **keep if** command than the drop or drop if command.

Back to question.

---

7. Both commands display the data in spreadsheet format. The edit command allows you to make changes directly in the data, just as you would in a spreadsheet. The browse command allows you to view the data but not to make changes.

Note: the edit command does make entries in the Stata log, but they are not a very useful record of the changes you've made. The best record to keep of data editing uses a case identifier, but the Edit command uses _n, which is relative to the current sort order of the data in memory. **Because Edit leaves you with no record of what you've done, we recommend that you never use it.**

Back to question.

Review again?

Another topic?

Choosing the Correct Weight Syntax

One of the most common mistakes made when analyzing data from sample surveys is specifying an incorrect type of weight for the sampling weights. Only one of the four weight keywords provided by Stata, pweight, is correct to use for sampling weights. The purpose of each type of weight follows.

Sampling or Probability weights: pweight

Stata has a special term **pweight** to specify probability weights. Probability weights are another name for sampling weights. The pweight option causes Stata to use the sampling weight as the number of subjects in the population that each observation represents when computing estimates such as proportions, means, and regressions parameters. A robust variance estimation technique will automatically be used to adjust for the design characteristics so that variances, standard errors and confidence intervals are correct.

Run the following commands to demonstrate the difference between unweighted and weighted results, and to see that Stata automatically uses the robust estimation technique when you use pweights. As before, these data are from the 1999 Tanzania DHS women's survey. Here we predict having 0-2 kids using a woman's education and controlling for her age.

```
clear
use "q:\utilities\statatut\svysamp.dta"
logit twokids age educat
logit twokids age educat [pweight=sampwt]
logit twokids age educat [pweight=sampwt], robust cluster(earea)
```

Note that the coefficient associated with education changes only slightly with the use of pweights. However, the standard error increases quite a bit, with a corresponding decrease in z. Most notably, p increases from .001 to .030 with the addition of weights. If we had not included probability weights, we would have assigned too much importance to the role education plays in the number of children these women have. Adding the cluster(earea) option makes only a slight adjustment in this case, but is recommended.

The discussion below of other weight commands is included as general information. In most cases, these commands are ***NOT APPROPRIATE for use with sample survey data.***

Frequency Weights: fweight

Frequency weights are integers that indicate the number of times the observation was actually observed. It is used when your data set has been collapsed and contains a variable that tells the frequency each record occurred. For example, if the original data was:

```
x1    x2     y
16     3     1
16     3     1
19     2     0
19     2     0
19     2     0
```

and the estimation command would be

```
logit y x1 x2
```

then the collapsed data would look like:

```
x1    x2     y    count
16     3     1       2
19     2     0       3
```

and the estimation command would be

```
logit y x1 x2 [fweight=count]
```

Do not use fweights to specify sampling weights. Your variance of estimates, p-values and standard errors will be computed incorrectly.

Analytic Weights: aweight

Analytic weights are used when you want to compute a linear regression on data that are observed means. For example, instead of having data that looks like:

```
group   x      y
  1     3     22
  1     4     30
  2     8     25
  2     2     19
  2     5     16
```

suppose the data has been condensed with only the averages being available:

```
group    x      y    n
  1     3.5   26.0   2
  2     5.0   20.0   3
```

and a linear regression could be done by using the command:

```
regress y x [aweight=n]
```

Do not use aweights to specify sampling weights. This is because the formulas that use aweights assume that larger weights designate more accurately measured observations. Conversely, one observation from a sample survey is no more accurately measured than any other observation. Hence, using the aweight command to specify sampling weights will cause Stata to estimate incorrect values of the variance and standard errors of estimates, and p-values for hypothesis tests.

Importance Weights: iweight

Stata has a special weight command, iweight, which can be used by programmers who need to implement their own analytical techniques by using some of the available estimation commands. Special care should be taken when using importance weights to understand how they are used in the formulas for estimates and variance. This information is available in the Methods and Formulas section in the Stata manual for each estimation command. In general, these formulas will be incorrect for computing the variance for data from a sample survey.

Review again?
Another topic?

Combining data files

Introduction to merging

Merging is the process of adding variables from a permanent file on disk to the data in memory. The observations in the two files may be on the same level, for example, they may both be from surveys of the same people who were interviewed at different times. Or, the observations may be from surveys on different levels, such as mothers and their children. In either case, the files have one or more identifying variables in common.

Many people confuse **merge** with **append**. Append combines two files with completely different observations but the same variables, while merge combines files with the same or related observations but different variables. The **append** command is explained fully in a later example.

The **merge** command is simple to use, but there are a wide variety of situations, and corresponding pitfalls, associated with merging. The following three examples cover the more common situations.

One-to-one merging

Match merging

One-to-many merging

Caution (if using a version of Stata prior to 12)! Merging and appending both add data to the data already in Stata's memory. It is easy to ask Stata to put more data in memory than you have allowed room for. Add together the sizes of all the files you want to merge or append before you combine them, **clear** and **set memory** if necessary, then combine the files. If not, you may get the message "No room to add more variables/observations."

Another topic?

Commands to Analyze Survey Data

Stata provides two ways to analyze survey data. After a description of the two ways, there is a table to help you decide which one to choose.

The survey Commands

The preferred way is to use the family of commands that begin with **svy:**. (See **help survey** in Stata for a list of commands that can be run after svy:) These commands were designed especially for analyzing data from sample surveys. Before any of the survey estimation commands can be used, the **svyset** command should be used to specify one or more of the variables that describe the stratification, sampling weight, and/or primary sampling unit variables. You can try **svyset** by running the following commands:

```
    clear
    use "q:\utilities\statatut\svysamp.dta"
    svyset earea [pweight=sampwt], strata(urbrur)
```

In this example from the 1999 Tanzania DHS data, the variable earea ("enumeration area") is the PSU, sampwt is the probability weight, and urbrur (urban-rural) is the stratum identifier.

These values stay in effect until they are cleared or reset. If you save the data, these values are saved with the data and will be in effect the next time you use the data file.

We could now use any of the survey estimation commands. For example, the mean of a variable from the data set could be estimated as follows:

```
    svy: mean numkids

(running mean on estimation sample)

Survey: Mean estimation

Number of strata =       2          Number of obs    =    4029
Number of PSUs   =     176          Population size  =    4029
                                    Design df        =     174

--------------------------------------------------------------
             |             Linearized
             |       Mean   Std. Err.     [95% Conf. Interval]
-------------+------------------------------------------------
    numkids |   2.409699   .0617263       2.28787    2.531528
--------------------------------------------------------------
```

Stata first reports the names of variables that were defined with the svyset command and some statistics about the data used in the computation. It is a good idea to make sure the names of the variables, the number of strata and the number of PSU's reported are correct. The number of observations with non-missing data (4029) and the size of the population represented by the observations (4029) are also reported (these weights are normalized).

After any of the survey estimation commands, you can use the **test** command to test linear hypotheses and **lincom** to compute linear combinations of estimations. These special commands adjust the test statistics properly for the sample design. For example, to test whether urban women have fewer children than rural women:

```
    svy: mean numkids, over(urbrur)
    test [numkids]Urban = [numkids]Rural
```

Subpopulation Analysis

When using the svy commands to analyze only a portion of the sample (a sub-population), it is important to analyze the entire data set and to use the **subpop** option to identify those observations you want included in the estimate. This is because Stata needs to have information from every observation in the sample to compute the variance, standard error, and confidence intervals even though only the observations in the sub-sample are needed to compute means, proportions, and regression coefficients.

To use the subpop option, you need to generate a variable that has a value of 1 for the observations in your sub-population and a value of 0 for those that should be excluded. Here is an example where we compute the mean of numkids for the people living on Zanzibar (the variable zanzibar has a value of 1):

```
    svy, subpop(zanzibar): mean numkids  // CORRECT SUBPOPULATION ANALYSIS

(running mean on estimation sample)

Survey: Mean estimation

Number of strata =       2          Number of obs    =    4029
Number of PSUs   =     176          Population size  = 4.0e+09
                                    Subpop. no. obs  =     969
                                    Subpop. size     = 1.0e+08
                                    Design df        =     174

--------------------------------------------------------------
             |             Linearized
             |       Mean   Std. Err.     [95% Conf. Interval]
-------------+------------------------------------------------
    numkids |   2.858553   .1157252      2.630147    3.086959
--------------------------------------------------------------
```

Note that the subpopulation number of observations is listed as 969. It's a good idea to check that number to make sure your subpop variable is working as expected.

It would be **incorrect** to use the **if** option to subset the data:

```
    svy: mean numkids if zanzibar==1   // INCORRECT - DO NOT DO THIS

(running mean on estimation sample)

Survey: Mean estimation

Number of strata =       2          Number of obs    =     969
Number of PSUs   =      30          Population size  = 1.0e+08
                         |          Design df        =      28
                       WRONG
--------------------------------------------------------------
             |             Linearized
             |       Mean   Std. Err.     [95% Conf. Interval]
-------------+------------------------------------------------
    numkids |   2.858553   .1037723      2.645985    3.071121
--------------------------------------------------------------
                               |            |           |
                             WRONG         WRONG       WRONG
```

The number of PSUs is incorrect, so the standard error and the confidence interval are also incorrect. Note that the estimate of the mean is the same. This is just one example of how different the results can be when you subset the data. For some variables the difference might

be much smaller or much larger. It is best to always use the subpop option when analyzing a sub-population with the svy command.

Using the pweight and robust cluster() Options

The second way to analyze survey data is to use the estimation commands that allow the pweight and robust cluster options. The estimation commands when used with the pweight and robust cluster options handle the sampling weights and clustering properly. However, there is no option for specifying the stratification variable. As a result, the standard error may be larger than it would be using and svy command.

The following set of commands demonstrates the difference between logit (without stratum) and svylogit (with stratum):

```
clear
use "q:\utilities\statatut\svysamp.dta"
svyset earea [pweight=sampwt], strata(urbrur)
logit twokids age educat [pweight=sampwt], robust cluster(earea)
svy: logit twokids age educat
```

Choosing a method

The following table compares the two methods available for analyzing data from a sample survey:

Method	Strengths	Limitations
	test and **lincom** commands used after estimation adjust the test statistics correctly for the sample design.	
The survey commands	Can make finite population corrections for without-replacement samples. Option available on svyset command to specify the stratification variable.	The analysis command you want may not support the **svy** prefix: see **help svy_estimation** for a current list of those commands
Commands that allow pweight and robust cluster() options.	There may be an estimation command that supports **cluster** but does not support **svy**.	Should have at least 40 clusters available. Option for specifying a stratification variable is not available.

It is best to use the survey commands to analyze survey data. These commands incorporate the effect of clustering and stratification as well as the effect of sampling weights when computing the variance, standard error, and confidence intervals, and they allow you to perform analyses on a subset of the data using the subpop option. If the analysis technique you need is not available with the survey commands, then using the estimation commands with pweights and robust cluster() options would be a good choice.

Review again?
Another topic?

Common Errors and How to Avoid Them

Here are some common errors that you can avoid.

Ignoring Clustering

Ignoring clustering and unequal probability of selection of participants in your analyses. This results in biased estimates and false-positive hypothesis test results. *Avoid this error by using the **svy** commands for your analysis. If your analysis technique is not available with the svy commands, then use a command that allows **pweight** with the **robust cluster()** option.*

Using the Wrong Weight Command

Using the wrong weight specification in Stata. *For data from a sample survey, you should use the **pweight** option to define the sampling weight.* Using any of the other weight options (aweight, fweight, or iweight) can result in incorrect variance, standard errors, confidence intervals, and p-values.

Subsetting the Sample

Subsetting the sample when using the svy commands in Stata. These commands use the Taylor Series approximation for the variance estimation and must be able to correctly count the number of primary sampling units (PSUs) that were originally sampled. Subsetting the data may cause an incorrect number of PSU's to be used in the variance computation formula. *Do not subset the data from a sample survey and always use the **subpop** option when using the svy commands to do sub-population analysis.*

Stratum with only one PSU detected

You may get an error message when you try to run an svy command: "stratum with only one PSU detected". This happens when observations have values missing for variables in your model, resulting in their being dropped. An entire PSU may disappear as a result of missing values. Use the **svydes** command to identify the problem strata. A common fix is to combine a small stratum with an adjoining stratum. See the manual entry on svydes, or in Stata type **findit svydes**, or see http://www.stata.com/support/faqs/stat/stratum.html for details.

What set of observations is Stata analyzing?"

Using a subpop variable does not do the same thing as an -if-. In fact that's why the subpop option was invented. The -svy- commands use the whole dataset to help determine the standard error even if you are only looking at a subset of it (with a subpop var). During the time Stata is analyzing your data, Stata subsets to only those observations where ALL the following variables are non-missing:

- strata (if using one)
- psu (if using one)
- sample weight (if using one)
- subpop (if using one)
- analysis variable(s)**

If any one of them is missing then Stata drops the obs where any of those variables are missing. ** svymean with more than one variable will not subset to obs where all analysis variables are non-missing unless the "complete" option is specified.

Review again?
Another topic?

Data Characteristics

A sample survey is conducted to obtain information about the characteristics of a population. To reduce the cost and time necessary to collect the data, this task is often handled by selecting a subset (a sample) from the set of all measurements (the target population) of interest to the researchers. The methods that are used to select the sample add certain characteristics to the data. These characteristics must be incorporated into your analysis to get estimates concerning the entire population. There should be variables for each observation in your data set to identify each of the characteristics that are described next.

Clustering

A truly random selection of households would involve listing all the households in the country and randomly selecting the desired number of households from that list. Using this approach, each household would have an independent and equal chance of being included in the survey. While this approach is statistically ideal, the cost of first enumerating all households and then visiting the selected households that would be scattered all over the country make this approach impractical.

A more practical approach is cluster sampling. For example, in the Demographic and Health Surveys, "enumeration areas" from the Census or similar national surveys are first selected randomly from a list of all such areas in the country (or within strata if stratification is being used). These areas are often referred to as "clusters" or as "primary sampling units" (PSU's). They may be towns or villages, or they may be census tracts in cities. Generally each cluster contains roughly the same number of households.

The next step is to enumerate (count and label) all the households in the cluster. Then a random-selection process is used to select households within each cluster. This is the sample of households that will be visited for the survey.

While cluster sampling is much more practical, it also means that the households are not statistically independent. Instead, the characteristics of a given household (and its household members) are more like those of other households in the same cluster, and are less like households in other clusters. This effect of a non-independent sampling process, called the "sample survey design effect", shows up in the standard error of estimation statistics (means, regression coefficients). Clustering tends to decrease the size of standard errors, leading to a greater likelihood of rejecting the null hypothesis. In other words, it's more conservative to correct statistically for the design effect.

Stratification

The population can be divided into sections (the strata) that are internally more homogeneous. This may be done in order to over-sample smaller groups in a target population. Examples of strata are region of country, urban/rural residence, or education level. A separate sample is selected from each stratum. Like clustering, the observations within strata are not statistically independent, and adjusting for stratification leads to more conservative inferences about statistical significance.

Sampling Weights

Each observation in the sample is chosen using a method of random selection. An important property of this method is that the probability of selection may not be equal for all members of

the population. The sampling weight for each observation is computed as the inverse of the selection probability. Additional adjustments (such as non-response) may be made to the sampling weights. An observation with a sampling weight of 1000 represents one thousand individuals from the target population while another observation with a sampling weight of 50 represents only fifty individuals. Your analysis technique will need to use the sampling weights to estimate the characteristics of the target population from the reports of the sample. Thus, the sampling weights are needed in computing both the population estimates (such as means and regression coefficients) and their standard errors.

Population Number of PSUs in Each Stratum

Sampling with replacement means that once a PSU was chosen, it remains eligible to be selected again. Without replacement means that once the unit is selected, it is no longer eligible for selection and a finite population correction will need to be made in your analysis. Data sets for without replacement samples will need to have a variable that tells how many PSUs per stratum are in the population. Stata will use the data set to count how many PSUs were selected per stratum and compute a sampling fraction to use in analysis.

If the proportion of PSUs selected from each stratum (the sampling fraction) is small, then your sample can be analyzed as if it was selected with replacement and you do not need this variable. This simplifies your analysis since you can ignore the finite population correction. According to Cochran, "In practice the fpc can be ignored whenever the sampling fraction does not exceed 5% and for many purposes even if it is as high as 10%. The effect of ignoring the correction is to overestimate the standard error of the estimate." (William G. Cochran, Sampling Techniques, 3rd Edition, 1977, John Wiley & Sons)

Questions

What characteristics of the sampling design affect estimates such as totals, means, proportions, and regression coefficients? **Answer:** Sampling weights.
What characteristics of the sampling design affect standard errors, p-values, and confidence intervals? **Answer:** Sampling weights, clustering, and stratification.

Review again?
Another topic?

Data cleaning

Working with do-files, documenting, outliers, duplicate ids.

Unless you're lucky enough to be working with perfectly clean data, you'll need to at least check for outliers and, the bane of all survey work, duplicate identifiers. In the previous section on changing data, the **edit** command was introduced. We don't recommend using it for data cleaning, because it does not keep a record of what you've done, and you can't easily repeat data cleaning steps you've taken.

Stata do-files, which are collections of commands that you write, are an easy way to clean and document your data. Copy and paste the following commands into a text editor (e.g., Wordpad, Notepad, or the Stata do-file editor), and save the file as a *text* (ascii) file named test.do. The suffix should be ".do" (not required, but strongly recommended). The file should be in the same directory that you're using for your Stata session. If you're using a different directory, remember where you saved it.

```
clear
use "q:\utilities\statatut\exampfac.dta"

/* Notice the blank line above this line. Blanks are fine in do-files. */
/* Comment lines, like these, can be anywhere in a do-file, including */
/* at the end of a command line. */

// You can also use double-slashes anywhere in a command line
// to write a comment. Everything from there to the end of the
// line will be treated as a comment.

gen factype2= factype  /* comment at the end of a command */
gen factype3= factype  // this works, too

* Which facilities have an odd value for age?

ta age
list facid facname factype authorit if age == 1998
replace age= 1 if age == 1998

* Look for duplicate values of facid using duplicates command.

duplicates list facid
list facid factype authorit if facid == 1001

* Look for duplicate values of facid using _n.

sort facid
list facid factype authorit if facid == facid[_n-1]
#delimit ;
list facid factype authorit if facid == facid[_n-1]
  | facid == facid[_n+1];
#delimit cr

* Drop the duplicate observation where factype and authorit are missing.

drop if facid == 1001 & missing(factype)
```

If you're using the Stata do-file editor, click on the button "Execute (do)" (the icon looks like a page with writing on it and an arrow pointing to the right). If you used another editor, open Stata and type:

```
do test.do
```

or

```
do "driveletter/directory/etc/test.do"
```

depending on where you saved the file test.do. Press Enter and watch the results. When you see **--more--** press the Space Bar.

Questions:

1. Comments can begin with an asterisk (*) and end with a carriage return (Enter key), or they can begin with two slashes (//) and end with a carriage return, or they can be bracketed by /* */ and span an many lines as needed. All are shown in the example above. What happens if commands are enclosed in comments, like this:

```
	/*
	ta age
	list facid facname factype authorit if age == 1998
	replace age= 1 if age == 1998
	*/
```

Answer.

2. When **ta age** was executed, the screen stopped scrolling and **--more--** appeared at the bottom. What does that mean? Answer.

3. What does the **duplicates** command do? Answer.

4. Why are you showing a second way to check for duplicates? Answer.

5. The character **_n** is the sequence number of the current observation in memory. What does **facid[_n-1]** mean? Answer.

6. Why do we need to sort the data before testing for duplicate values of facid? Answer.

7. Why list both observations **_n-1** and **_n+1**? Answer.

8. What if there is more than one nested id variable? Answer.

9. What do the commands **#delimit ;** and **#delimit cr** do? Answer.

Answers:

1. Everything that's enclosed in comments is treated as a comment and is not executed. This is

a handy way to block out portions of a do-file that you don't want to run.

Back to question

---

2. When the results of a Stata command fill up more than one screen, Stata pauses to let you review what's currently on the screen. When you want to continue, press the Space Bar to see another page, or press the Enter key to see another line.

You have other options for how to handle this situation.

- put the command **set more off** in the beginning of your do-file
- click on the green "Clear --more-- Condition" button to see another page
- press Ctrl-Break to cancel the command
- press the "Q" letter key to cancel the command
- click on the "Break" button (red circle with white "X")

The **set more off** command allows the do-file to continue uninterrupted to the finish. This is a good option if you have turned on logging, so you'll have a record of what scrolled by. Use **set more on**(the default) when you need to stop and check each screen.

The other options are useful if you're debugging a do-file by executing one command at a time, or if you're working interactively (not using a do-file). If you Break or "Q" in the middle of a do-file, the do-file execution is canceled along with the command that produced the --more-- condition.

Back to question

---

3. The duplicates command counts, lists, tags, or drops duplicates. In this case we're just listing the duplicates so you can explore them further. See "One-to-one merging" for an example of dropping duplicates using the duplicates command. See **help duplicates** for details.

Back to question

---

4. Before the duplicates command, you needed to use [_n] to check for duplicates. Even though you no longer need it to check for duplicates, it's useful to know about [_n] for more advanced programming.

Back to question

---

5. You can think of each variable as a column in Excel, and _n is the row number. For example, facid is the first variable, which would be column A in Excel. To refer to the cell in the third row, you would type A3 in Excel. Similarly, in Stata facid[3] is the value of facid for the third observation in memory.

Since Stata executes a command on every observation from top to bottom, facid[_n] is the current observation it's looking at, facid[_n-1] is the observation before the current observation, and facid[_n+1] is the next observation.

Back to question

---

6. If duplicate values of facid exist, sorting the data will place them next to each other in the data

in memory. So, the array element reference **[_n-1]** will refer to the element immediately prior to the current observation, and the duplicate value will match the current value.

Back to question

---

7. Actually, we've asked to list the current observation (_n). If the current observation has the same value of facid as either the previous or the next observation, it's a duplicate. This method catches 2 or more observations with the same value for any variable.

Back to question

---

8. If there is more than one nested id variable, you need to sort in the proper nest order and check all ids for duplicates using the logical operator & (ampersand) between them. Suppose there are 3 nested ids:

```
      sort id1 id2 id3
      list id1 id2 id3  if id1 == id1[_n-1] & id2 == id2[_n-1] & id3 == id3[_n-1]
```

Again, we'd like to point out that the **duplicates** command is much easier to use than this method:

```
      duplicates list id1 id2 id3
```

We've shown you the method using _n because there are many instances in programming where you'll want to refer to observations before or after a particular case. This implicit array is powerful and well worth understanding and remembering for other applications.

Back to question.

---

9. In a do-file, Stata assumes that each command is no more than 1 line long, and that each line ends in a carriage return (when you press the Enter key, a text editor inserts a carriage return symbol). If you want to type a command that is more than one line long, you can use **#delimit ;** to tell Stata to look for a semi-colon instead of a carriage return. From that point on, you must end each command, whether one or more lines long, with a semi-colon. To switch back to carriage return, use **#delimit cr**.

There are other ways to continue a single command across more than one line. One way is to comment out the carriage return - type /* at the end of one line, and */ at the beginning of the next line (to end the comment):

```
list facid factype authorit if facid == facid[_n-1] /*
  */  | facid == facid[_n+1]
```

Another way is to end a line with ///, which tells Stata to continue reading the next line as a continuation of this line:

```
list facid factype authorit if facid == facid[_n-1] ///
   | facid == facid[_n+1]
```

There is no one "correct" method, so use the one you prefer.

Back to question

Review again?

Another topic?

Describing the data

Describing the variables: means, univariates, frequencies, and data types.

Now we'll use some real data. The data are from a health facility survey conducted in Tanzania in 1999. Copy each of these commands into the Command window, press Enter, and observe the results. Note that the letters in bold on each command are acceptable abbreviations. If you see "-more-" at the bottom of your Results screen, press the Space Bar to see a new page of data, or press "q" to quit the command.

```
clear
use "q:\utilities\statatut\exampfac.dta"
describe
summarize
su pill-natural
su fphour*
codebook urbrur facname
tabulate urbrur
tab urbrur, nolabel missing plot
tab factype urbrur
tab1 factype urbrur
tab2 factype urbrur
tab2 factype urbrur, row col cell
```

Questions:

1. The **describe** command lists each variable in Stata's memory. What do the terms "double," "str42," "byte," etc. in the second column refer to? Answer.

2. How do I specify which data type I want to use? Answer.

3. The **summarize** command lists the number of observations, mean, standard deviation, min, and max for a variable. Why is the number of observations different for some variables, and is even 0 for facname? Answer.

4. How can I summarize a specific set of variables? Answer.

5. When is the **codebook** command useful? Answer.

6. The **tabulate** command gives frequencies (counts), and is most useful with categorical variables. What are the two ways to specify a one-way frequency? Answer.

7. What do the **nolabel missing plot** options do on the **tabulate** command? Answer.

8. How can I get two-way frequencies (cross-tabulations)? Answer.

Answers:

1. That is the data type for each variable. Each data type handles a different kind of data. The following table describes the data types used by Stata:

```
Type            Min          Max   Precision   Bytes   Type
-------------------------------------------------------------------
byte      -2 digits     2 digits    2 digits       1   integer
int       -4 digits     4 digits    4 digits       2   integer
long      -9 digits     9 digits    9 digits       4   integer
float       -10**38       10**36      10**-8       4   real
double     -10**307      10**307     10**-16      8   real
str1              1            1                 1   string
str2              2            2                 2   string
...               .            .                 .   ...
str2045           1         2045              2045   string
strL     2000000000   2000000000        2000000000   long string
-------------------------------------------------------------------
```

Prior to Stata version 13, strings were limited to 2045 characters. Starting with Stata 13 a new data type **strL** can hold strings up to *2 billion characters.* You can see this and the limits on just about everything else in Stata by typing the command **help limits,** and there's a detailed explanation of data types in the PDF documentation and under **help data types.**

Back to question.

---

2. You can specify a data type on the **generate** command:

```
      gen byte a=0
```

If you don't specify a data type, by default Stata uses type float (4 bytes). Using an efficient data type reduces the file size. This is important for very large files or for computers with little memory (RAM). The **compress** command selects the most efficient data type after variables have been generated. See compress in Miscellaneous Tips and Tricks for details on compress.

Back to question.

---

3. The "Obs" column displays the number of non-missing observations for numeric variables.

For string variables, like facname, it is always 0.

Back to question.

---

4. There are two ways to specify a variable list, both shown in the example:

- pill-natural (first variable - last variable)
- fphour* (root variable name plus *)

These two methods work with all Stata commands. To use the first method, you need to know the position of each variable in the Stata data file. Use the describe command to see those positions, or look for them in the Variables window.

Back to question.

---

5. The codebook command gives univariate statistics about numeric variables, and it is a handy way to get information about string variables.

Back to question.

---

6. The two ways to get one-way frequencies are:

- tab factype (for a single variable)
- tab1 pill-natural (necessary for lists of variables)

Another handy command is **fre** written by Ben Jan at the University of Bern. It is not built into Stata, but we have installed it on all terminal servers at CPC. LIke all user-contributed Stata commands, it is available for free from the SSC archives at Boston College. You can install it on your desktop or laptop by typing:

ssc install fre

Back to question.

---

7. These three options give extra information about the variable urbrur:

- **nolabel** displays the numeric values instead of the value labels
- **missing** shows how many observations have missing values
- **plot** gives a graphical comparison of the frequencies

For more information on how Stata handles missing values, see missing values in the

Miscellaneous Tips and Tricks section of this tutorial.

Back to question.

---

8. The two ways to get two-way frequencies are:

- tab factype urbrur
- tab2 factype urbrur

These two commands are equivalent.

Back to question.

Review again?

Another topic?

Documenting your work

Stata provides several opportunities to document your work. You can describe your variables in more detail than is possible with the variable name using the **label variable** command. You can attach one or more words to each value of a variable using the **label values** command. These variable and value labels will be displayed in the results of Stata commands that support them.

The do-file is the best place to document how you created your variables or cleaned your data. Not only is the code there, allowing you or others to reproduce your work, but you can also add explanations and keep a record of decisions about how decisions were made. You can start with this **do-file template** to create a format that suits your needs.

After creating your analysis file, or if you have a data file for public dissemination, you might want to create a codebook using the **cb2html** command. This command allows you to add much more information to your codebook about each variable, the data set, and even the questionnaire, using the Stata **note** command. Such information might include the algorithm you chose and the decisions you made in creating a new variable. For survey data, they might include the skip instructions to the interviewer to help the data user understand the pattern of missing values. If you are not using CPC's network, you'll need to download this command from the SSC archive:

```
ssc install cb2html
```

Another topic?

Dummy variables

Creating Indicator (Dummy) Variables

Shortcuts to save you time.

Sometimes we need to create a number of indicator, or dummy, variables from a single categorical variable. These indicators usually take the value of 1 if the observation has the attribute and 0 if the observation does not. Most Stata commands support a syntax that creates indicator variables for you automatically. They call that syntax **factor variables**. There's a clear and detailed discussion in Chapter 25 of the Users Guide.

Here's a simple example to give you the flavor of factor variables syntax. Suppose each respondent has a value for their age group in a variable called agegroup that has 5 values. You want to create 5 indicator variables with values 0/1 that describe whether the respondent is in age group 1 or not, age group 2 or not, etc., and use them in a regression on the correlates of body mass index (bmi). The factor variable syntax is:

```
regress bmi i.agegroup
```

It's that simple! Age group 1 is automatically treated as the base level and omitted from the equation.

Factor variables syntax is much more powerful than this simple example illustrates. For example, it will create interactions for you with continuous as well as categorical variables. See the Users Guide for a complete discussion.

Unfortunately, not all commands support the factor variable syntax. Below are some alternatives in case you need to use a command that doesn't support factor variables.

The most obvious way to do this is the **generate** command. Suppose we want to create 5 indicator variables from agegroup, a variable with 5 values:

```
gen age1=0
replace age1= 1 if agegroup == 1
gen age2=0
replace age2= 1 if agegroup == 2
etc.
```

This can be tedious if you're creating a lot of indicator variables. A few shortcuts are available, including **recode**, **autocode**, and **egen**, all of which are discussed in the Users Guide referred to above. Here are a few more alternatives.

The first shortcut is the **forvalues** command. See Looping over variables and values in this tutorial to learn the basics of this command.

```
forvalues n=1/5 {
   gen byte age`n' = 0
   replace age`n' = 1 if agegroup==`n'
```

```
}
```

This use of **forvalues** command simply generates the two commands in the first example 5 times for us, eliminating all that typing and opportunity for error. Note that we've added the data storage type **byte** to the generate command. Since the indicators only contain the values 0 and 1, they easily fit in a single byte of storage, so this option saves megabytes of storage. See Describing the data in this tutorial for an explanation of Stata's storage types.

The second shortcut is the **tabulate** command, which is the easiest to use.

```
tab agegroup, gen(age)
```

The gen option on tabulate creates a new dummy variable for each value of agegroup. It names each dummy using the prefix you assign in parentheses, in this case "age". Note that the dummies are named age1 through age5, which may or may not correspond to their value. However, the values are recorded in the variable labels.

The third shortcut is the **xi** command. This command is really intended to feed indicator variables into another Stata command, such as a regression. It has largely been replaced by factor variables, but it will create dummy variables.

```
rename agegroup age
xi, prefix(i) noomit i.age
```

First we rename agegroup to age so that the indicator variables have a shorter name. In the xi command, the **prefix(i)**option gets rid of Stata's default prefix, "_I", which it adds to each dummy variable name. We use the **noomit** option because xi does not create a dummy for the lowest value (remember, it's designed to feed these dummies into a multivariate procedure, so one category must be dropped). The result is 5 indicator variables named iage_1, iage_2, ..., iage_5.

The fourth shortcut is the **margins** command. This command is used in postestimation after a previously fitted model at fixed values of some covariates. The command is powerful and full of options, and it is much more than a "shortcut". See the full help in the PDF Base Reference.

Review again?

Another topic?

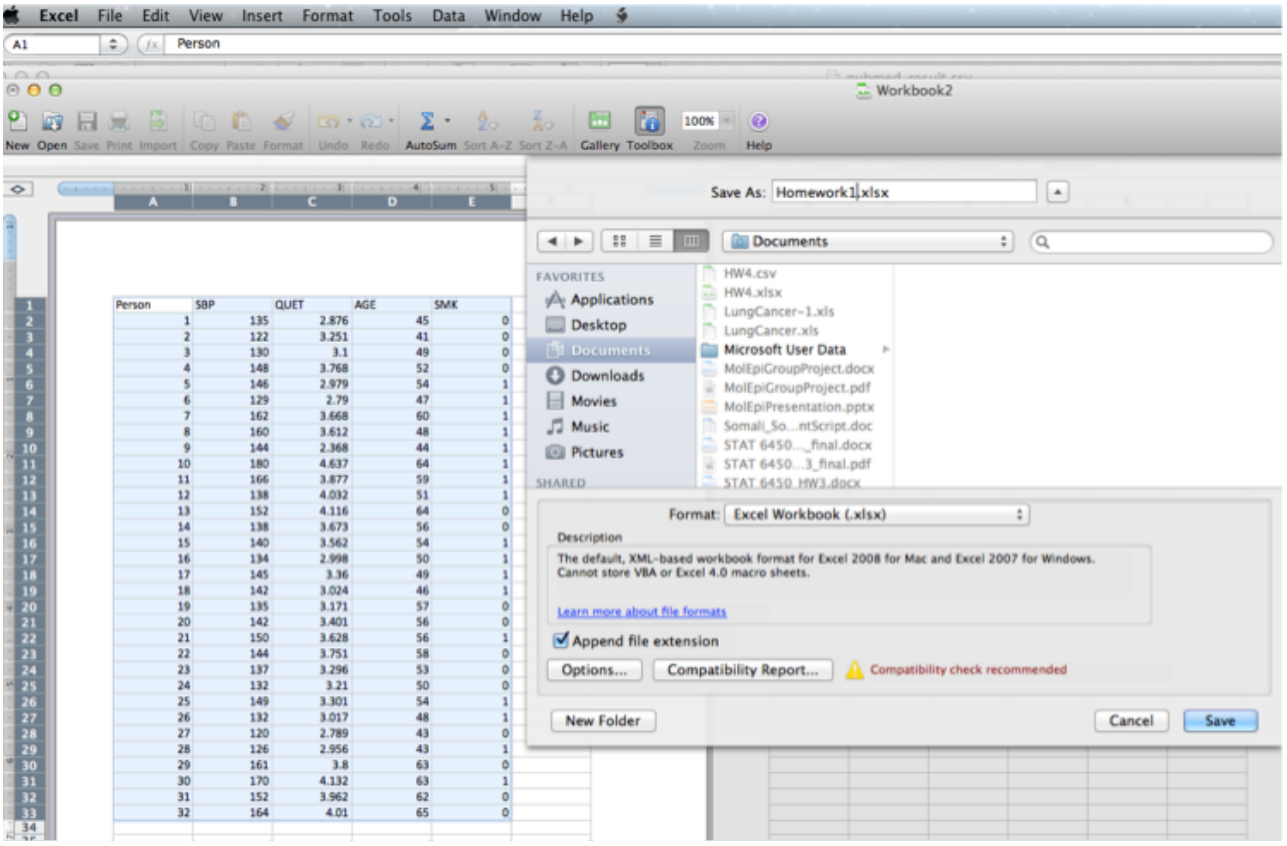
Entering Data into STATA

Help Center

Here are some steps that you could follow to enter datasets into STATA

Step One:

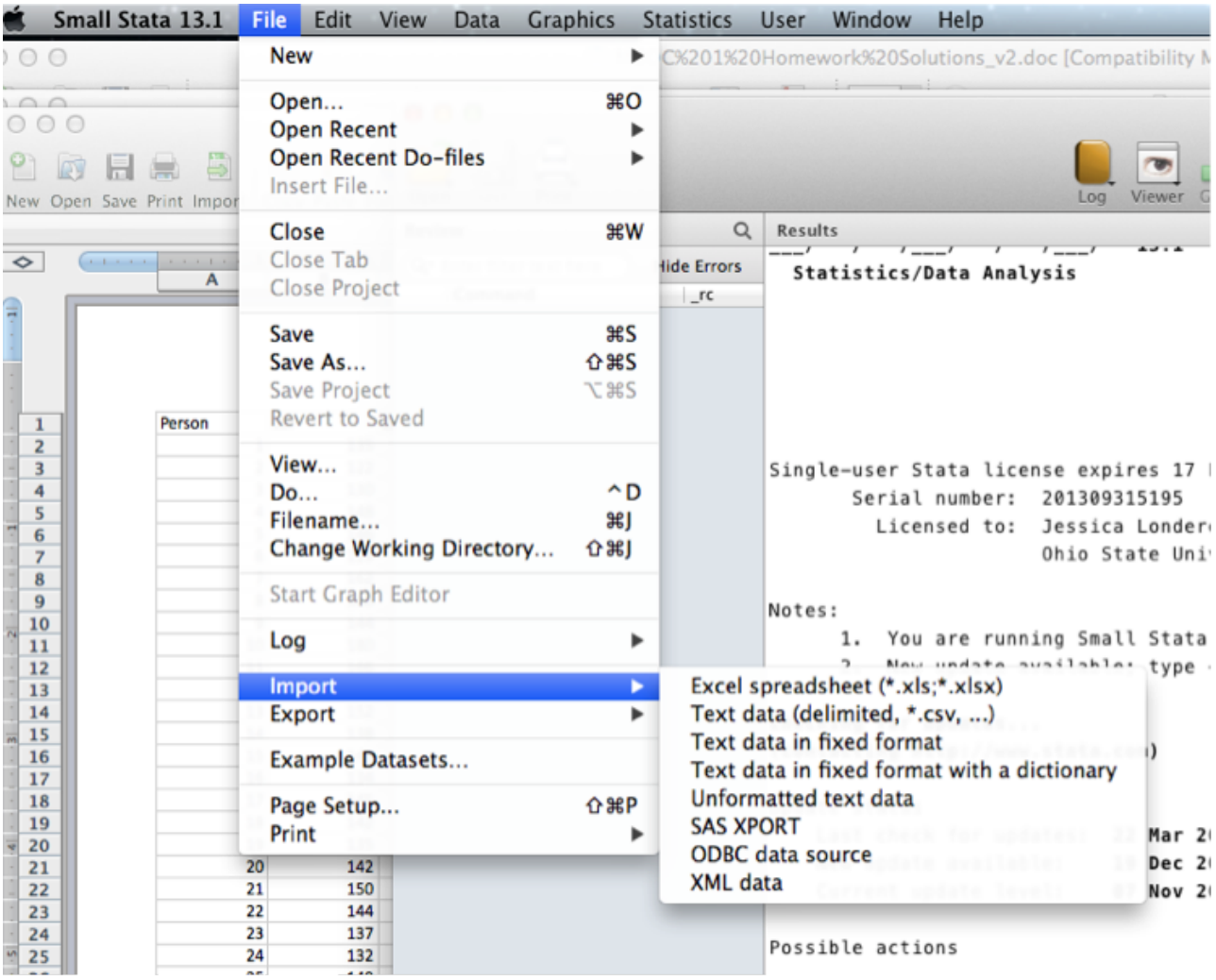
Copy and paste the table into a Microsoft Excel spreadsheet. Save the spreadsheet.



Person	SBP	QUET	AGE	SMK
1	135	2.876	45	0
2	122	3.251	41	0
3	130	3.1	49	0
4	148	3.768	52	0
5	146	2.979	54	1
6	129	2.79	47	1
7	162	3.668	60	1
8	160	3.612	48	1
9	144	2.368	44	1
10	180	4.637	64	1
11	166	3.877	59	1
12	138	4.032	51	1
13	152	4.116	64	0
14	138	3.673	56	0
15	140	3.562	54	1
16	134	2.998	50	1
17	145	3.36	49	1
18	142	3.024	46	1
19	135	3.171	57	0
20	142	3.401	56	0
21	150	3.628	56	1
22	144	3.751	58	0
23	137	3.296	53	0
24	132	3.21	50	0
25	149	3.301	54	1
26	132	3.017	48	1
27	120	2.789	43	0
28	126	2.956	43	1
29	161	3.8	63	0
30	170	4.132	63	1
31	152	3.962	62	0
32	164	4.01	65	0

Step Two:

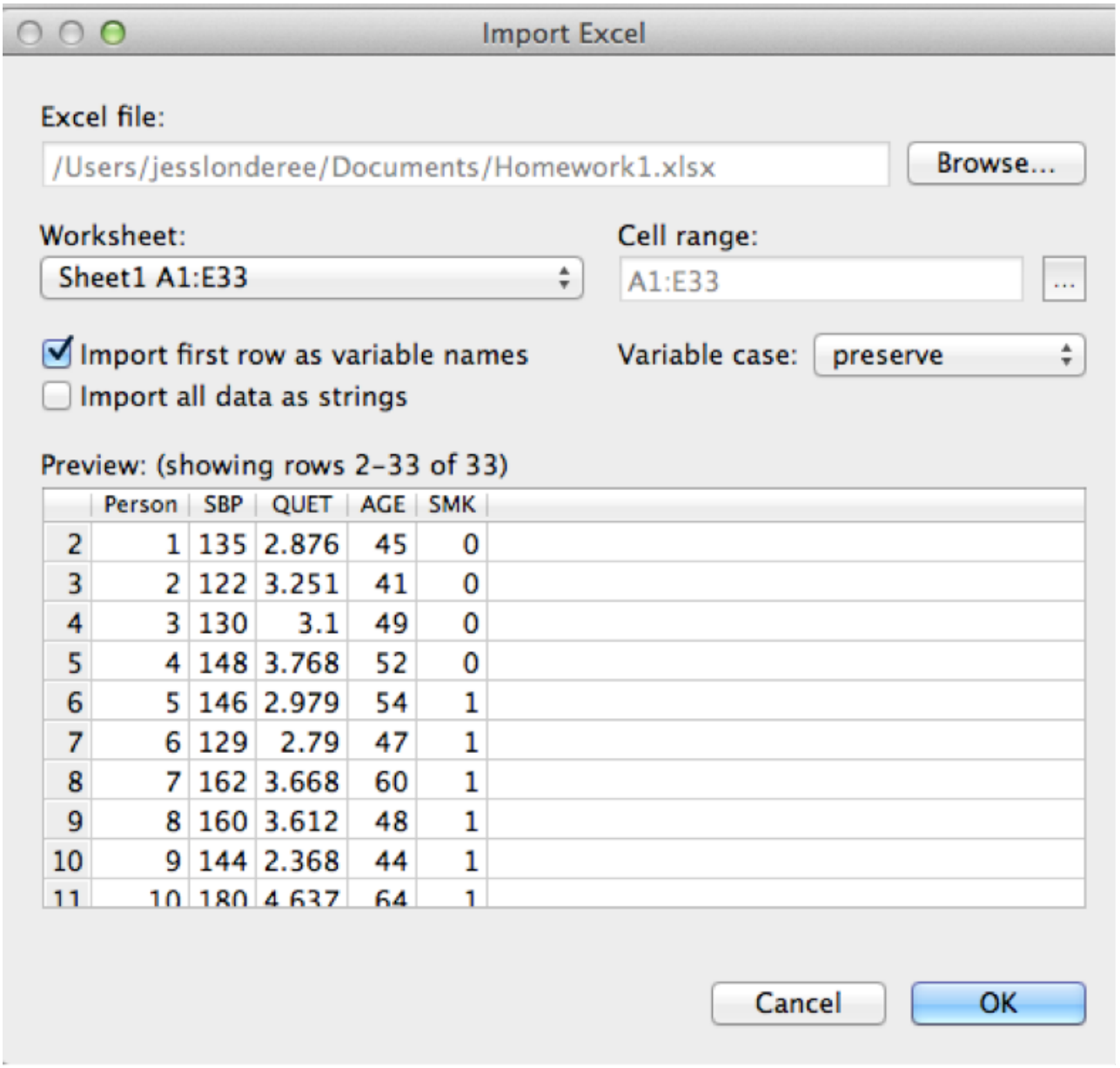
Open STATA, Click on File>Import>Excel Spreadsheet. *This will bring you to the "Import Excel" module*



Step Three:

To the right of where it says "Excel File", click "Browse" then navigate to find your spreadsheet.

Make sure the box next to "Import first row as variable names" is checkmarked, then click "Ok".



You should be all set to get started!

*You can also **download this guide**.*

Created Tue 24 Mar 2015 12:34 PM PDT

Last Modified Tue 24 Mar 2015 12:46 PM PDT

Error messages

Error messages and what they mean

Stata's error messages are usually clear, but it's not always obvious what you can do about them. Here are a few that could use a little explanation.

```
1.     too many values
```

This error message can result from a 2-way tabulation of variables that each have a lot of values. If you really want to cross-tabulate these two variables, you can do it in pieces, limiting the range of one or the other variable in each tabulation:

```
      tab2 var1 var2  if var1 <= 10
      tab2 var1 var2  if var1 > 10 & var1 <= 20
         etc.
```

```
2.     no: data in memory would be lost
```

If you make any changes in the data currently in memory, then try to exit Stata, you'll get this message. Stata is reminding you that you have not saved the data in memory since making the changes. You can either **save** the data, if you want to keep the changes, or **clear** the data from memory, if you don't want to keep the changes. Then exit.

Review again?

Another topic?

Exporting Stata Results to MS Office

Stata results can be exported in a wide variety of ways. We've divided the approaches into two groups: working with the Results Window and working with estimated parameters directly.

From the Results Window

Copy Table into Excel, Word, or PowerPoint

Suppose you've run svy:mean on 3 variables, and Stata has produced a nice table in the Results Window that you want to capture.

```
. svy, subpop(subpop): mean var1 var2 var3
(running mean on estimation sample)

Survey: Mean estimation

Number of strata =      43       Number of obs    =      20607
Number of PSUs   =      86       Population size  =  261897236
                                  Subpop. no. obs  =      18081
                                  Subpop. size     =  253890607
                                  Design df        =         43

--------------------------------------------------------------
             |             Linearized
             |       Mean   Std. Err.     [95% Conf. Interval]
-------------+------------------------------------------------
        var1 |   289.6824   4.175324      281.2621    298.1028
        var2 |   106.6606   1.311485      104.0157    109.3055
        var3 |    396.343   4.804086      386.6547    406.0314
--------------------------------------------------------------
```

Copy Table

You can copy the table into Excel or into a Word table and retain much of its formatting:

- Highlight the table in the Results Window.
- Right click on the table and select Copy Table.
- In **Excel**, click on the cell where you want the upper-left corner of the table.
- Right click and select Paste.

You'll need to adjust column widths to see all the text. We've found that copying the upper half of the table separately from the lower half helps Excel select the right number of columns to use.

To copy into **Word**, count the number of rows and columns you'll need, create a table of that size, highlight all the cells in the table, and then paste. In the above example, the table of parameter estimates requires 5 columns and 6 rows if you do not include the top and bottom horizontal lines in your highlighting.

Copy Picture

You can copy the highlighted table as a picture directly into a Word document. It's a perfect snapshot of the table in the Results Window.

- Highlight the table in the Results Window.
- Drag the left border of the Results Window to the right until the highlighted table just fills the width of the Window.
- Right click on the table and select Copy as Picture.
- In **Word** or **PowerPoint**, right click and paste where you want the picture.

You can resize the picture, and you can right click to edit it.

Using the Parameter Estimates

Suppose that we want to make a table of just the means in the above example. We can do this be taking advantage of a very nice feature of Stata. Most Stata commands produce temporary variables containing the key results. These temporary variables will continue to store these values until you use another Stata command that replaces those results, or until you end your Stata session.

You can look at what results have been stored using either **return list** or for estimation commands **ereturn list**. Here's a subset of the estimate parameters stored temporarily after the svy:mean command that produced the table above:

```
ereturn list

scalars:
               e(df_r) =  43
     e(N_strata_omit) =  0
         e(singleton) =  0
            e(census) =  0
          e(N_subpop) =  253890607
             e(N_sub) =  18081
             e(N_pop) =  261897236
             e(N_psu) =  86
          e(N_strata) =  43
            e(N_over) =  1
                 e(N) =  20607

    (lots of results omitted here)

matrices:
                 e(b) :  1 x 3
                 e(V) :  3 x 3
           e(_N_subp) :  1 x 3
          e(V_srssub) :  3 x 3
             e(V_srs) :  3 x 3
              e(_N) :  1 x 3

    (mores results omitted here)
```

For purposes of this example, we're only interested in the means. The estimates of the means are contained in a matrix called e(b), which has dimensions 1 x 3.

The matrix e(b) is temporary. We want to create variables that we can export to Excel, so we first need to put the means into variables that won't go away when we run another estimation command. There are lots of ways to do this. We'll demonstrate one way that uses official Stata commands and is flexible, but it requires you to manage the data a bit more than you might care to do. After that we'll talk about a number of user-contributed commands that work with results from estimation commands.

First, let's verify that we have in fact selected the right matrix of estimates:

```
matrix list e(b)
```

```
e(b)[1,3]
         var1       var2       var3
y1  289.68245   106.6606  396.34305
```

We can compare these values with the means printed in the table above and see that we have the estimates of the means. Next put them into a matrix, and then create a variable for each cell in the matrix:

```
matrix means = e(b)
gen meanvar1= means[1,1]
gen meanvar2= means[1,2]
gen meanvar3= means[1,3]
list meanvar*  in 1

     +--------------------------------+
     | meanvar1   meanvar2   meanvar3 |
     |--------------------------------|
  1. | 289.6824   106.6606    396.343 |
     +--------------------------------+

keep in 1
(21661 observations deleted)
```

Since every observation has the same values for the three means, we can keep just one observation.

At this point we can **browse** the data, copy, and paste it to an empty Excel file template we've created with column and row labels, colors, and all the other formatting we want. We find this to be the easiest approach.

Another approach is to **export** the results to Excel.

```
export excel "c:\tables\means.xlsx", firstrow(variables)
```

Here is an example of how to write a program to build a more complex table with many rows.

putexcel

New in Stata 13 is the command putexcel. It reads the parameter estimates described above and writes them directly to an Excel spreadsheet. You can specify the starting cell (upper left) and even write column headers. Here's an example that accomplishes the task described above.

```
svy, subpop(subpop): mean var1 var2 var3
putexcel C4=("Means") B5=matrix(e(b)) using "c:\tables\means.xlsx"
```

As we saw above, the **svy: mean** command puts the means into the matrix e(b). We then instructed **putexcel** to write those 3 means into a row in a spreadsheet, starting with cell B5. We also wrote the label "Means" centered above the 3 means in cell C4. The **putexcel** command is very powerful and flexible. See the PDF help for full details.

User-Contributed Commands

It is a testament to both the cleverness and generosity of the Stata user community that so

many powerful open-source commands exist outside the set shipped by Stata Corporation, and further that the authors maintain these commands and continue to offer improved versions. Most of these commands are archived at Boston College in the SSC (Statistical Software Components) website. To read about and install these commands, you can search the SSC website, or you can use the **ssc** command in Stata.

```
ssc describe parmest
ssc install parmest
```

CPC maintains the most current version of each of the commands listed below. You will need to install them locally if you run Stata on a standalone computer.

The discussion below is brief, intended only to point you to the command you might need. See the help for each command for details.

tabout

The **tabout** command produces publication-quality cross tabulations. Lots of features are available to customize the table. The output file may be in tab-delimited or html format. Tab-delimited format files may be copied into Word and converted into a table using the Table menu (Table, Convert, Text to table). Html format files can be opened in your browser and copied. Inside Word, use Paste Special and paste the table as Formatted Text (rtf). There's a tutorial on the command as well.

outreg

The **outreg** command uses the saved results after an estimation command to create a text file of the results you select. You can then turn this text file into a Word table. This command has lots of formatting features, and you can combine the results from several regressions into a single table. Here's a simple example to get you started.

```
cd "c:\tables\"
sysuse auto, clear
regress mpg foreign weight headroom trunk length turn displacement
outreg using outtab1, replace
```

The output file is tab-delimited and has the extension .out. Open the file in Word, highlight the rows of results, click on Insert, Table, Convert text to table.

outreg2

This is an extension of outreg to enhance its capabilities with multiple models and to provide more format options. Here's an example from the help. Follow the instructions above to convert the results to a Word table.

```
    cd "c:\tables\"
    sysuse auto, clear
    regress mpg foreign weight headroom trunk length turn displacement
    outreg2 using outtab2, replace cttop(full)
    regress mpg foreign weight headroom
    outreg2 using outtab2, see
```

esttab

This command is part of the **estout** package. The **esttab** command produces a table of regression results that have been stored by the **eststor** command, or the current results if nothing has been stored. The output table may be tab, csv, rtf, html, or other formats. Tab-delimited format files may be imported into Excel using the Import Wizard. Rtf files go directly into Word. There's a steep learning curve to this powerful command, but if you produce publication-quality tables often that have to look "just so", it's worth the time to study this command. Here's a simple example that outputs to .rtf.

```
eststo: svy, subpop(subpop  if childage == 1): mean var1 var2 var3
esttab using "c:\tables\esttab_means.rtf", replace
```

The command gives you a clickable link to esttab_means.rtf in the Results Window. There's a tutorial available for esttab here.

xml_tab

Like esttab, the **xml_tab** command works from stored estimates. The output is in Excel's xml format. This format allows for a very feature-rich table that takes advantage of many of Excel's capabilities. Here's a simple example.

```
svy, subpop(subpop): mean enerbev1 enerfood1 enertot1
xml_tab e(b), save("d:\statatemp\xml_tab.xml") replace
```

The command gives you a clickable link to xml_tab.xml. You can store multiple sets of estimates using **estimates store** and use xml_tab to combine them in a single table.

parmest

The parmest command comes in a package of four modules: parmest, parmby, parmcip, and metaparm. The **parmest** and **parmby** commands put each estimation result into a separate observation in an output dataset. **parmcip** inputs these variables and adds new variables containing confidence intervals and p-values. **metaparm** does a metanalysis on sets of estimation results.You can see an example here.

logout

The **logout** command captures the results that are printed to the Results Window (log) and writes them to Excel, Word, or other formats. The success of the formatting in the output file depends on the complexity of the results table you are exporting. Compared with copying from the log by hand and pasting into a spreadsheet, this approach may not produce as well formatted tables, but it can be automated. Here's an example:

```
logout, save(c:\tables\logout_means) excel replace: ///
   svy, subpop(subpop): mean var1 var2 var3
```

This command produces a file call logout_means.xml, and it gives you a clickable link to this file in the Results Window.

Review Again?

Another topic?

Getting help for Stata

There are several ways to get help for Stata commands within Stata:

- help
- menus
- search

The **Help** menu links you to the PDF Documentation, which is the full set of official manuals for Stata. The **help** command gives most or all of the same information, but you must know the command.

The Data, Graphics, and Statistics **menus** build Stata commands for you. They are function-oriented, so you do not need to know the command or syntax.

The **search** command and link in the Help menu allows a keyword search for related words that are not themselves Stata commands. They will point you to help for the related commands. It searches not only the help files that come with Stata but also a wide range of web-based resources at Stata Corp and elsewhere. Type **help search** for details.

Web-based Help

The search command searches most web-based resources for specific commands and keywords. However, if you want more general help, such as a tutorial, a good resource is the UCLA's IDRE Stat website. It has links to resources for many statistical computing programs, including Stata, and has a search feature. You may also want to explore Stata's Resources for learning Stata.

Manuals

CPC no longer maintains current printed copies of the Stata manual set, since the full set is available in PDF in the Help menu. However, we do have other Stata and third-party books that focus on specific topics in Stata programming. These books are shelved in the CPC library and can be checked out.

User-Supplied Stata Commands

Many more commands are available for Stata than are shipped in the official version. Stata Corporation encourages users to develop new commands, and at times adopts these commands in some form into their official version. These include statistical, data management, and graphics commands. If you don't see a command in the official Stata manuals or help, the search command will search the user-supplied command archives. You can also search the main repository at the Boston College Department of Economics: http://ideas.repec.org/s/boc/bocode.html

---

Review again?

Another topic?

Graphics

Creating simple graphs to visualize your data

Stata produces publication-quality graphs that can be modified to fit publishers' formats, and *schemes* (templates) are available for many journals to automate this process.

This page gives a few examples of graphs that you might want to create, but it by no means does justice to the broad capability of Stata graphics. In addition to the Stata *Graphics* PDF manual and the online help, other sources of information include *A Visual Guide to Stata Graphics* by Michael Mitchell (available from Stata Press), the drop-down Graphics menu in the Windows version of Stata, and a couple of online learning aids:

- www.stata.com/help.cgi?graph
- www.ats.ucla.edu/stat/stata/topics/graphics.htm

Each of the commands below creates an example of a commonly used graph. Use the Tanzania 1999 facility data file, then run the following commands to see the examples.

```
clear
use "q:\utilities\statatut\exampfac.dta"

/* Create Government vs Non-govt authority */

gen govt= .
replace govt= 1 if authorit == 1
replace govt= 0 if authorit > 1 & !missing(authorit)
label define g 0 "Non-govt" 1 "Govt"
label value govt g

/* Count FP staff-hours/week at each facility */

gen fphrs= 0
foreach v of varlist fphour* {
   replace fphrs= fphrs + `v'  if !missing(`v')
}

/* Count FP methods available at each facility */

gen methods= 0
foreach v of varlist pill-natural {
   replace methods= methods + 1  if `v' == 1
}

/* Histogram of number of methods with normal curve superimposed */

histogram methods, normal

/* Box plot of methods by whether govt or non-govt facility */

graph box methods, by(govt)

/* Scatter plot of FP hours by number of methods */

scatter fphrs methods

/* Add a linear regression line to the scatter plot */

twoway (scatter fphrs methods) (lfit fphrs methods)
```

Another topic?

Groups and subsets of data

Groups and subsets of data, comparison, missing values.

We'll continue using the 1999 Tanzania Facility Survey data. Copy and paste the following commands into the Command window, press Enter, and see what happens. No need to copy the comments (surrounded by /* */). Note the double equal signs (==) in the **tab** and **su** commands.

```
   clear
   use "q:\utilities\statatut\exampfac.dta"

/* types of government facilities */

   tab factype if authorit==1

/* mean age of hospitals */

   su age if factype<=4

/* availability of condoms at religious facilities */

   tab malecond if authorit==4 | (authorit>=7 & authorit<=9), missing

/* mean age of facilities by urban-rural location */

   sort urbrur
   by urbrur: su age

/* mean age of facilities that offer family planning by urban-rural location */

   by urbrur: su age if pill<.

/* display the first 10 observations in memory */

   list factype authorit urbrur in 1/10
```

For details on subgroup processing, see by command in the Miscellaneous Tips and Tricks section of this tutorial.

Questions:

1. The **if** option on the **tab** command restricts the command to observations that meet the following qualifications. In this case, the qualification is that authority is 1 (government). What is the difference between

```
authorit=1
```

and

```
authorit==1
```

in the **tab** command? Answer.

2. The **==** sign is called a relational operator. What relational, logical, and arithmetic operators are available in Stata? Answer.

3. The **sort** command puts the observations in memory in ascending order of the values of one or more variables. In this case, the variable is authority. What does the command **sort factype urbrur** do? Answer.

4. The **by** option executes the following command once for each value of the by variable. When would you use **by** instead of **tab2** to see data separately by groups. Answer.

5. What does the phrase **pill<.** mean? Answer.

6. How are missing values stored in Stata data? Answer.

7. The **in** option allows you to specify which specific observations you want. What do the numbers "1" and "10" refer to in this example? Answer.

8. What is another way, using _n, to write "in 1/10"? Answer.

Answers:

1. A single equal sign means give the value on the right to the variable on the left, in other words it means "assignment." A double equal sign means check whether the variable on the left has the value on the right, in other words "comparison."

Back to question.

---

2. Here is the full list of relational, logical, and arithmetic operators:

```
      ==   equal to
      >    greater than
      >=   greater than or equal to
      <    less than
      <=   less than or equal to
      ~    not
```

```
!    not
&    and
|    or
+    addition
-    subtraction
*    multiplication
/    division
^    power
```

Back to question.

---

3. It puts the data in ascending order of factype, and within each value of factype it orders the data in ascending order of urbrur. The **gsort** command allows you to sort in descending as well as ascending order.

Back to question.

---

4. The tab2 command works best with categorical data, while the by option works best with continuous variables. For example, if a and b are categorical variables:

```
by a: tab b
```

is the same as:

```
tab2 a b
```

The tab2 command gives more concise output and offers chi-square and other statistics.

For n-way analyses of continuous data, consider the **tabsum**, **tabstat**, or **table** commands instead, such as:

```
tabulate urbrur, summarize(age)
tabstat age, by(urbrur) stats(mean n)
table urbrur, contents(mean age n age)
```

The table command is particularly powerful and handles multiple levels of conditioning variables. See help for details.

Back to question.

---

5. It means "pill is less than missing."

Back to question.

---

6. Missing values are stored as a number larger than the largest allowable value for the data type. So, you need to be careful when using the "greater than" operator: This command includes missing values of age:

```
tab factype if age>=25
```

If you don't want missing, you need to specifically exclude it:

```
tab factype if age>=25 & age<.
```

You can specify up to 27 different types of missing values. They are: ".", ".a", ".b", ... ,".z". (During data entry, you can use these to differentiate among Refused, Not Applicable, Don't Know, and other possible reasons for missing values.) These are the largest values allowed by the data type, so you can use "<." to exclude all 27 missing values for a variable. Back to question.

---

7. The numbers refer to the temporary variable "_n" that Stata creates for each observation in memory. This number is not saved if you save a permanent data file. Furthermore, this number changes if you change the sort order of the data.

Back to question.

---

8. You can write:

```
list factype authorit urbrur  if _n <= 10
```

Back to question.

Review again?

Another topic?

Importing and exporting data files

We often find data on the Web in formats other than Stata, or we need to share data with our collaborators who (unfortunately) are not Stata users.

There are several ways to get data into and out of Stata. While this list is probably not exhaustive, it covers most of the formats that we encounter at CPC. Some methods are available from within Stata, while others require shareware or commercial software.

Stata commands

import and export

These are native Stata commands to read and write several data file formats. The commands **import excel** and **export excel** read and write Excel spreadsheets (.xls and .xlsx). The command **export excel** is quite flexible, allowing you to write to a specific cell in a specific sheet in an Excel workbook if you need to. The commands **import sasxport** and **export sasxport** read and write SAS xport (transport) format files and will even read and write value labels from/to a SAS formats.xpf file. See **help import** and **help export** for details. See Exporting Stata Results to MS Office for an example of **export excel.** Here's a simple example of importing an Excel spreadsheet:

```
import excel "C:\tables\workbook.xlsx", firstrow
```

usespss

This command, written by Sergiy Radyakin, is available from the SSC archive. It handles Windows SPSS data files with the .sav suffix (called "system" files in SPSS). It does not handle SPSS portable files with the .por suffix. See **usesas** and **Stat/Transfer** below for software that can deal with SPSS portable files.

outsheet and insheet

The **outsheet** command is a way to write the variable names and values (but no labels) into tab-delimited text files, comma-separated values (.csv) files, or files with other delimiters. Most software will read a text file with one of these delimiters between the values. Stata also reads these text files with the **insheet** command. So, you can export data from something like MySQL to a .csv file and read it into Stata with **insheet**.

xmluse and xmlsave

These commands transfer data between Stata and MS Excel's xml format. (For most purposes you'll probably find the **import excel** and **export excel** commands more useful.) The xml format is a portable text version of Excel's .xls or .xlsx format files. **xmlsave** produces a file that Excel reads directly. In order to use an Excel file in Stata, though, you need to open it in Excel and save it as type xml. There are two xml "Save As" formats in Excel. The "XML spreadsheet 2003 (*.xml)" option seems to work better.

Other software

usesas and savasas

While these are implemented as Stata commands, they are listed here under "Other software" because they both require you to have SAS installed locally as well as Stata. At CPC we have both of these commands installed and they work wonderfully.

If you have a standalone computer with both SAS and Stata installed, you may want to download these commands from the SSC archive. You may need to edit the code to tell Stata where to find the SAS command, but that is all explained in the help and in comments inside the code.

Stat/Transfer

This is not intended as an advertisement for a commercial product, but it really is useful. It converts data files between statistical software, spreadsheets, databases, etc. with ease. If you find yourself needing to import or export data across software formats often, it is worth the investment. It is distributed by Stata Corp as well as by the developer Circle Systems. At CPC, send an email to CPC Help for access to Stat/Transfer.

use13, saveold

At CPC we keep Stata up to date with the most recent version. Currently, that is version 13. You may be working with colleagues who have an earlier version that cannot read version 13 files. When they try to use one of your files, they will get the message that the file is not in Stata format.

One option is the command **use13**, written by Sergiy Radyakin at the World Bank and available from the SSC archives. You can ask your colleagues to install **use13** and run it in place of the **use** command to read your files.

Another option is Stata's **saveold** command. You can use it to save your file in version 12 format, which version 11 can also read, and send this older version of the file to your colleagues.

Review Again?

Another topic?

Introduction to Stata

This tutorial is function-oriented, focusing on the data-management tasks most needed by data analysts working with sample survey data. It works up from basic tasks, such as how to drop variables, to the tasks needed for complex file organization, such as how to reshape and merge data files.

There is also a section on Analyzing Data from Sample Surveys. It explains which sampling weight command to use and whether to use svy or robust cluster to adjust for survey design effects.

These web pages assume that you are using Stata Version 13 for Windows.

If you would like to run the example commands, you need to copy the example Stata data files to your local PC. Click download sample data for instructions. If you're using a computer at the Carolina Population Center, the data are available to you on Q:\temp\statatut\.

See Stata Windows environment below for an orientation to the Windows interface. It also gives you sources of help beyond this tutorial.

Other resources available to help you learn Stata include the UCLA's IDRE Stat website, several introductory guides in the CPC library and others available from Stata Press, and Stata Corporation's Resources for learning Stata.

SAS Users: the SAS User's Guide to Stata may help you make the transition from SAS to Stata.

A simple example

- **input:** putting data into Stata
- **generate:** creating a new variable
- **list (or browse):** viewing the contents of memory
- **save:** saving memory in a permanent Stata-format file
- **log:** capturing the results of Stata commands for printing
- Stata's default actions
- how data are stored in RAM

Using permanent Stata data files

- **clear:** clearing Stata's memory
- **set memory:** allowing enough space for the data
- **use:** copying the file into memory
- **save,replace:** saving changes

Describing the data

- **describe:** names of variables
- **summarize:** the mean, min, and max of variables
- **codebook:** more univariate statistics
- **tabulate:** frequencies and cross-tabulations

- data types and data storage

Groups and subsets of data

- **if:** do command for a subset of observations
- **sort:** order observations by the values of a variable
- **by:** do command for groups of observations (requires sort)
- **in:** do command for a range of observations
- relational, logical, and arithmetic operators
- missing values

Changing the data

- **replace:** change the values of a variable
- **recode:** change the values of a variable
- **rename:** change a variable name
- **label:** labeling variables, values, and data files
- **drop:** drop one or more variables
- **drop if:** drop observations conditional on one or more variables
- **edit:** editing the data file directly

Data cleaning

- **do:** storing and executing commands in do-files
- **#delimit:** writing long commands in do-files
- **/* */:** documenting your do-files
- finding and fixing outliers
- **duplicates:** finding duplicate ids

Adding summary statistics to a data file

- **egen:** add summary statistics to each observation
- **collapse:** create file of summary statistics by groups

Combining data files

- **one-to-one:** same observations in each file
- **match merge:** many observations in each file match, but some don't
- **one-to-many:** hierarchical data, analysis at the **lower** level
- **merging summary statistics:** hierarchical data, analysis at the **higher** level
- **appending:** adding observations with the same variables

Reshaping a data file

- **reshape long:** change variables to observations
- **reshape wide:** change observations to variables

Documenting Your Work

- **variable labels**
- **value labels**

- **do-file template**

Graphics

- **histogram** with normal curve fitted to it
- **graph box** plot displayed for two groups
- **scatter** plot
- **twoway** scatter plot with regression line
- other resources for learning graphics in Stata

Analyzing Data from Sample Surveys

- **Data characteristics:** stratification, clustering, sampling weights
- **Choosing the correct weight syntax:** pweight, aweight, fweight, or iweight?
- **Commands to analyze survey data:** svy, robust cluster, subpop
- **Logistic Regression Example:** adjust, svylc, svytest
- **Common errors** and how to avoid them

Labor-Saving Techniques

- **looping over variables and values**
- **dummy variables**
- **a simple program**

Miscellaneous Tips and Tricks

- **getting help for Stata**
- **updating Stata**
- **importing and exporting data files**
- **working with large files**
- **shrinking large data files**
- **error messages** and what they mean
- **missing values** and how to work with them
- **the by command** in detail
- **exporting results** to MS Office
- **the parmest command:** saving Stata results
- **temporary files**
- **looping:** foreach in detail
- **looping with while**
- **precision** and data storage

Authors: Phil Bardsley, Kim Chantala, and Dan Blanchette

Logistic Regression Analysis

The Logit Model

Logistic Regression is used to model dichotomous (0 or 1) outcomes. This technique models the log odds of an outcome defined by the values of covariates in your model. In addition to covering how to model sub-populations, we will use both the svy commands and the robust cluster commands. The following example comes from The National Longitudinal Study of Adolescent to Adult Health.

Research Question: How is being in the upper quartile of the Vocabulary test score (PVT_Q4) influenced by a boy's grade in English (ENGL_GPA) and Family composition (BIOMAPA)?

Predictive Model:

$$\log\left[\frac{\Pr(PVT\_Q4 = 1)}{1 - \Pr(PVT\_Q4 = 1)}\right] = b_0 + b_1\, AGE\_KID + b_2\, BIOMAPA + b_3\, ENGL\_GPA$$

Where

b_0 = Intercept

b_1 = Change in log odds of being in upper quartile for one year increment in age

b_2 = Change in log odds of being in upper quartile for living with Biological Parents

b_3 = Change in log odds of being in upper quartile for increase in one grade level

The model predicted log-odds for the categorical subpopulations will be:

BIOMAPA	ENGL_GPA	Ln(odds)
0 = No	4 = A	$b_0 + b_1 AGE\_KID + 4b_3$
0 = No	3 = B	$b_0 + b_1 AGE\_KID + 3b_3$
0 = No	2 = C	$b_0 + b_1 AGE\_KID + 2b_3$
0 = No	1 = D/F	$b_0 + b_1 AGE\_KID + b_3$
1 = Yes	4 = A	$b_0 + b_1 AGE\_KID + b_2 + 4b_3$
1 = Yes	3 = B	$b_0 + b_1 AGE\_KID + b_2 + 3b_3$
1 = Yes	2 = C	$b_0 + b_1 AGE\_KID + b_2 + 2b_3$
1 = Yes	1 = D/F	$b_0 + b_1 AGE\_KID + b_2 + b_3$

We are assuming a model with a common slope for age of the boy, but different intercepts defined by grade in English and living with both biological parents.

The relationship between probability and odds

The odds of an outcome is related to the probability of the outcome by the following relation:

$$\text{odds} = \frac{\text{probability}}{1 - \text{probability}}$$

An odds ratio is just the ratio of the odds of the outcome evaluated at two different sets of values for your covariates. It is easy to show that to test the hypothesis that p1 = p2 you can test that the hypothesis that an odds ratio comparing group 1 to group 2 is equal to 1. However, you cannot easily put a confidence interval on the difference between the two probabilities.

SVY: LOGIT

The **svyset** command is used to specify the design information for analysis. Use the **strata** keyword to specify the stratification variable (region), the **pweight** keyword to specify the probability weight variable (gswgt1), and specify the primary sampling unit (psuscid).

```
svyset psuscid [pweight=gswgt1], strata(region)
```

The **svy: logit** command states the model being tested. The first variable following **svy: logit** denotes the outcome (pvt_q4) of our model, and the following variables are the covariates. The option **subpop**

is used to specify the sub-population we want to be used to compute parameter estimates. All 18,924 observations are needed for the variance computation because Stata determines the design information (number of primary sampling units) used in the formula variance computation.

```
svy, subpop(male): logit pvt_q4 age_kid biomapa engl_gpa
```

Stata lists the number of observations with no missing values for the variables in the model (N=17,191) and has summed the corresponding sample weights to estimate 19,955,620 adolescents in the U.S. are represented by these observations. The number of observations with complete data in the sub-population is 8,366 representing 10,084,117 boys. Note that the number of strata (4) and primary sampling units (132) has been correctly counted.

```
Survey: Logistic regression

Number of strata   =         4                  Number of obs      =     17191
Number of PSUs     =       132                  Population size    =  19955620
                                                Subpop. no. of obs =      8366
                                                Subpop. size       =  10084117
                                                Design df          =       128
                                                F(   3,    126)    =     49.14
                                                Prob > F           =    0.0000

------------------------------------------------------------------------------
             |             Linearized
     twokids |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     age_kid |  -.0451845   .0278879    -1.62   0.108    -.1003656    .0099965
     biomapa |   .4273138   .0820139     5.21   0.000     .2650354    .5895923
    engl_gpa |   .4258579   .0423055    10.07   0.000     .3421493    .5095664
       _cons |  -1.886177   .4411884    -4.28   0.000    -2.759144   -1.013211
------------------------------------------------------------------------------
```

The **adjust** command can be used to estimate a linear combination of the coefficients estimated for the variables in our model. If you do not specify a value for a variable when using adjust, Stata will incorrectly substitute the sample mean rather than an estimate of the population

mean. ***This is because adjust ignores any weights used by the estimation commands.*** (See *Stata Reference Manual, Release 9, Vol 1 A-G*, page 10.) To correctly compute a linear combination, it is necessary to specify a value for all variables in the model. For example, the following statement:

```
    adjust age_kid=17 engl_gpa=3, by(biomapa) xb se ci
```

produces an estimate of the log odds of scoring above the 75th percentile for boys at age 17 with a grade of B in English for both categories of living with both biological parents:

```
------------------------------------------------------------------------------
    Dependent variable: pvt_q4      Command: logit
Covariates set to value: age_kid = 17, engl_gpa = 3
------------------------------------------------------------------------------

-------------------------------------------------------
Live with |
Bio Mom & |
Dad       |
0=N/1=Y   |         xb        stdp          lb          ub
----------+--------------------------------------------
        0 |   -1.37674   (.100564)   [-1.57572   -1.17776]
        1 |   -.949427   (.094665)   [-1.13674   -.762115]
-------------------------------------------------------
    Key:  xb         =  Linear Prediction
          stdp       =  Standard Error
          [lb , ub]  =  [95% Confidence Interval]
```

You can also include the **exp** option at the end of the adjust command to get exponentiated linear combinations of the coefficients. The **pr** option on adjust is not available after using the svylogit command.

The **lincom** command can also be used to produce linear combinations of the coefficients:

```
    lincom 17*age_kid + 1*biomapa + 3*engl_gpa + _cons

 ( 1)  17.0 age_kid + biomapa + 3.0 engl_gpa + _cons = 0.0

------------------------------------------------------------------------------
     pvt_q4 |      Coef.    Std. Err.      t    P>|t|    [95% Conf. Interval]
------------+-----------------------------------------------------------------
        (1) |  -.9494267    .0946653   -10.03   0.000   -1.136738   -.7621154
------------------------------------------------------------------------------
```

The results from lincom match those from adjust. The advantage of using lincom is that a hypothesis test can also be performed. For example, suppose you want to compute the odds ratio comparing 17 year-old boys not living with both biological parents to12 year-old boys living with both biological parents. Assume both boys make the same grade in English. We would want to estimate the difference in log odds for these to:

$(b_0 + 17*b_1 + GRADE*b_3) - (b_0 + 12*b_1 + b_2 + GRADE*b_3) = 5*b_1 - b_2$

Since b_1 is the coefficient for AGE_KID and b_2 is the coefficient for BIOMAPA, the lincom command would be:

```
    lincom 5*age_kid - 1*biomapa
```

This produces the desired difference in log odds:

```
( 1)  5.0 age_kid - biomapa = 0.0

------------------------------------------------------------------------------
     pvt_q4 |      Coef.    Std. Err.      t    P>|t|    [95% Conf. Interval]
------------+-----------------------------------------------------------------
        (1) |  -.6532364    .1641137    -3.98   0.000   -.9779635   -.3285094
------------------------------------------------------------------------------
```

The **or** option can be added to the lincom command to get the odds ratio ($e^{5*b_1-b_2}$):

```
    lincom 5*age_kid - 1*biomapa , or
```

The following table will be printed:

```
( 1)  5.0 age_kid - biomapa = 0.0

------------------------------------------------------------------------------
     pvt_q4 | Odds Ratio    Std. Err.      t    P>|t|    [95% Conf. Interval]
------------+-----------------------------------------------------------------
        (1) |   .5203589     .085398    -3.98   0.000    .3760762    .7199962
------------------------------------------------------------------------------
```

Thus, assuming equal grades in English, the odds of a 17 year-old boy not living with both biological parents is only half that of a 12 year boy who lives with his biological parents.

The **test** command can be used to test joint hypothesis about variables. For example, testing that the coefficient for age_kid and biomapa are both equal to zero can be done with the following stata command:

```
    test age_kid biomapa
```

which produces the following output:

```
Adjusted Wald test

 ( 1)  age_kid = 0.0
 ( 2)  biomapa = 0.0

       F(  2,   127) =   14.51
            Prob > F =    0.0000
```

logit with pweight and robust cluster

Note that we can subset the data (**if male == 1**) when using the **robust cluster( )** options in Stata and still have the variance computed with an acceptable technique. The primary sampling unit (psuscid) is used as the argument to the **cluster** option and the sample weights (gswgt1) are specified by **[pweight=gswgt1]**.

```
    logit pvt_q4 age_kid biomapa engl_gpa  if male == 1 [pweight=gswgt1], robust cluster(psuscid)
```

The results and interpretation in the following output are identical to the results from svylogit.

```
Logistic regression                                 Number of obs   =       8366
                                                    Wald chi2(3)    =     142.44
                                                    Prob > chi2     =     0.0000
Log pseudolikelihood = -4429.7883                   Pseudo R2       =     0.0384

                              (Std. Err. adjusted for 132 clusters in psuscid)
------------------------------------------------------------------------------
             |               Robust
      pvt_q4 |      Coef.   Std. Err.       z     P>|z|       [95% Conf. Interval]
-------------+----------------------------------------------------------------
     age_kid |  -.0451845   .0277835     -1.626   0.104      -.0996393    .0092702
     biomapa |   .4273138   .0817512      5.227   0.000       .2670844    .5875433
    engl_gpa |   .4258579   .0430438      9.894   0.000       .3414936    .5102222
       _cons |  -1.886177   .4414855     -4.272   0.000      -2.751473   -1.020881
------------------------------------------------------------------------------
```

Review again?
Another topic?

Looping over variables and values

We often need to run the same command for a large number of variables. For example, we might want to change the value 9 to missing for 200 variables in a data file. We can type the recode command 200 times, or we can type the **foreach** command once and let it create 200 copies of recode for us.

The **foreach** and **forvalues** commands are convenient ways to save you typing. In addition to the above recode situation, they can also rename a group of variables for us, saving us typing many rename commands. In fact, any command, or set of commands, that you need to repeat over a group of variables, is a likely candidate for one of these labor-saving commands.

Two examples of **foreach** and **forvalues** are shown below.

To see an engaging discussion of the topic by Nicholas Cox of the University of Durham, UK, see this PDF file: How To Face Lists With Fortitude. This article was also published in Stata Journal 2(2), 2002.

```
clear


use "q:\utilities\statatut\examfac2.dta"


su q102_* q103_*
su q102_* q103_*

/* Replace all values of 99 with missing in q102_* and q103_* */

foreach x of varlist q102_* q103_* {
   replace `x'= .  if `x' == 99
}


/* Rename q102_* to title* and rename q103_* to fphour* */

forval x=1/20 {
   rename q102_`x' title`x'
   rename q103_`x' fphour`x'
}
exit
```

TIP: To see how each pass through the loop is resolved, do the following.

```
clear
use "q:\utilities\statatut\examfac2.dta"
set tracedepth 1   // only show one level down
set trace on  // turn on Stata's trace option

foreach x of varlist q102_* q103_* {
   replace `x'= . if `x' == 99
}
set trace off  // return to normal Stata mode
```

Questions:

1. What is the "*" in the foreach and forvalues commands? Answer.

2. What purpose does the word "varlist" serve in the **foreach** command? Answer.

3. What is the "x" in each of the commands? Answer.

4. The **foreach** and **forval** commands have a couple characters I haven't seen before in this tutorial: {} and `. What are they? Answer.

5. Why does the **forval** command take up more than one line? Answer.

Answers:

1. The asterisk (*) after a variable name is a shortcut in Stata. You can use it in any Stata command, not just this one. It tells Stata to look for all variable names that begin with "q102_" and "q103_" and end with anything. We know that they end with the numbers 1-20. It's the same as typing all 40 variable names:

```
foreach x of varlist q102_1 q102_2 q102_3 ... q103_19 q103_20 {
```

Back to question

---

2. The word "varlist" in the **foreach** command tells Stata that we are referring to a list of existing variables. The foreach command has other options, such as "newlist" for generating a list of new variables.

Back to question

---

3. The "x" is itself a variable that stands for each variable in the foreach variable list, or each value in the forvalues number list. In the first example, the command following the "{" is repeated once for each variable in the variable list, substituting the real variable name for the "x" in the replace command. In the second example, "x" is substituted for each number in the list 1/20. Note that it need not be "x" - any variable name will do, but "x" is quick and easy to type.

The net result of the **foreach** command is the same as typing the following 40 times:

```
replace q102_1=. if q102_1==99
replace q102_2=. if q102_2==99
  etc.
```

Back to question

---

4. The braces {} in the **foreach** and **forval** commands surround the commands that you want to execute for each variable. In the **foreach** command there is one command (replace). But in the **forval** command we have inserted two commands between the braces. The other character is an accent mark. On a standard US keyboard it is on the same key with the tilda (~) on the left side next to the 1 key. This character tells Stata that the character following is a special kind of variable known as a "local macro" in Stata. A macro (local or global) is temporary and does not become part of the data in memory. A macro must be surrounded by an accent on the left and an apostrophe (also called a "single quote") on the right like this: `m'

Back to question

---

5. Stata requires separate lines for each part of these commands. Here is how Stata's help lays out the syntax rules:

- the open brace must appear on the same line as "foreach" or "forvalues"
- nothing may follow the open brace except, of course, comments
- the first command to be executed must appear on a new line
- the close brace must appear on a line by itself

Alternatively, we can change the command delimiter to semicolon (;) and put the entire command on one line like this:

```
#delimit ;
forval x=1/20 {; rename q102_`x' title`x'; rename q103_`x' fphour`x'; };
```

Frankly, that's pretty hard to read. Using separate lines and indentation looks much nicer. In fact, if you copy the above lines into your do-file editor and run them, you'll see in the Results window that Stata will improve your code by using separate lines and indentation - at least your log will be easy to read!

Back to question

Review again?

Another topic?

Looping with while

while: another way to repeat commands.

The **while** command can be used in much the same way as **foreach** or **forvalues**, but it has greater flexibility. While it's generally used by programmers (writing commands), it is a tool available for do-files as well.

You can copy the following text into a do-file and run it to see how **while** works.

```
clear

use "q:\utilities\statatut\examfac2.dta"

su

list q102_*  in 1/10

/************************ from log ****************************************
            q102_1         q102_2         q102_3         q102_4         q102_5
  1.     PH NurseB          Other        ClinOff             99             99
>
  2.       ClinOff             99             99             99             99
>
  3.      NurseOff      PH NurseB      PH NurseB       MCH Aide       MCH Aide
>
  4.       ClinOff        ClinOff  Nurse/Midwife       MCH Aide             99
>
  5.      MCH Aide    NurseAssist        ClinOff             99             99
>
  6. Nurse/Midwife        ClinOff  Nurse/Midwife             99             99
>
  7.   NurseAssist    NurseAssist             99             99             99
>
  8.       ClinOff    NurseAssist    NurseAssist             99             99
>
  9.       ClinOff    NurseAssist    NurseAssist             99             99
>
 10. Nurse/Midwife             99             99             99             99
>
*******************************************************************************/

/**
Notice that q102_* vars appear to be character data not numeric.
       The value label "title" is associated to these variables formats
       the numeric data to look like character data.  Click here to see webpages about
       variable labels and value labels.
   **/


/** Say you want to disassociate the value label title from the q102_* vars.
    You could type: label value q102_1 ;
                    label value q102_2 ;
                    label value q102_3 ;
                    label value q102_4 ;
                    label value q102_5 ;

    Or you could use a while command to disassociate
    the value label title with the q102_* vars.  **/

local i= 1
while `i' <= 5 {
  label value q102_`i'
  local i= `i' + 1
 }
```

```
/** It is possible to write a while command without using a local macro variable.
    Users generally use local macro variables because they are not associated
    to the data set.  It's not possible for their values to be different from
    one observation to the next and they can hold a number or a string of text.

    In this example, the while command uses the local macro variable i to step
    through the list of variables q102_1 q102_2 q102_3 q102_4 q102_5.  `i' is
    first evaluated to be 1 and then 2 and so on until the expression `i'<=5 is
    not true (when `i'=6). The brackets "{" and  "}" enclose all commands to be
    processed by the while command.  **/

list q102_*  in 1/10


/* Create string variables that can hold up to 13 characters. */

local j=1  /** local i=1 would also work.  i is not the only local macro variable name allowed **/
while `j' <= 5 {
 gen str13 titlec`j'= ""
 local j= `j' + 1
 }

/** Rename q102_* vars **/
local hi= 1
while `hi' <= 5 {
 rename q102_`hi' title`hi'
 local hi= `hi' + 1
 }

list title1-title5 in 1/10

local hi= 1
while `hi' <= 10 {
 replace titlec1= "not missing"  if title1 == `hi'
 local hi= `hi' + 1
 }

/* More than 1 list of variables can used in while command. */

/* Replace all values of 99 with missing in title1-title5 and q103_* */

su title5-q103_1

local i= 1
while `i' <= 5 {
 replace title`i'= .  if title`i' == 99
 replace q103_`i'= .  if q103_`i' == 99
 local i= `i' + 1
 }

su title5-q103_1

/** Create new variables from existing variables. **/
local i=1
quietly while `i' <= 5 {
 gen mch_h`i'= q103_`i'  if title`i' == 8
 label var mch_h`i' "MCH Aid's hours"
 local i= `i' + 1
}

/** NOTE: Adding "quietly" to a STATA command tells STATA not to print any output for the command.
         All lists of variable have to have the same number of elements.
*********************************************************************/


list mch_h1-mch_h5  in 11/15


/** NOTE: STATA reads the shorthand varlist of mch_h1-mch_h5 to be all
         variables positionally in the data set between mch_h1 and mch_h5 (not
         necessarily mch_h1 mch_h2 mch_h3 mch_h4 mch_h5).  When using STATA
         interactively, the variable list window in the lower left-hand side
         shows the list of variables in the data set in the order of their position.  **/
```

```
/* Here is an example using while to run the decode command
   with all title1-title5 variables:

   local i= 1
   while `i' <= 5 {
       decode title`i', gen(title`i'c)
       local `i'= `i'+1
   }

*************************************************************************/
```

Note: Dan Blanchette contributed this web page, however please direct questions to Phil Bardsley as noted below.

Review again?

Another topic?

Looping: foreach in detail

More examples of foreach and forvalues commands.

Stata processes all observations of the data set for each element in the list of the **foreach** and **forvalues** commands.

This page shows 4 different types of lists that can be used in the **foreach** command: varlist, newlist, numlist, and anylist and how to use the **forvalues** command.

Both **foreach** and **forvalues** commands use "local macro" variables in processing whatever list they loop through. A local macro variable is not a variable in the data set but rather a variable available to Stata for programming purposes. To learn more about local macro variables use Stata's help by searching for "macro". Here's a quick example:

```
. local X= "MyVarName"  // create local macro variable X and set it equal to the text MyVarName.
. display "My variable name is: `X'"

"My variable name is: MyVarName"
```

Notice that the local macro variable X is enclosed in a left and right single quote. The left quote is under the tilde (~) key on the left side of your keyboard and the right quote is under the double quote (") key on the right side of your keyboard. Enclosing a letter or variable name in left and right quotes tells Stata to evaluate it as a local macro variable.

NOTE: the following is a **do-file**. The **foreach** and **forvalues** commands are better suited to a **do-file** than to interactive use of Stata. Highlight and copy all text between the horizontal bars and paste into the Stata interactive do-file window so that you can run this do-file.

```
clear

use "q:\utilities\statatut\examfac2.dta"

summarize


list q102_*  in 1/10

/*********************** from log ****************************************
           q102_1         q102_2         q102_3         q102_4         q102_5
  1.     PH NurseB          Other        ClinOff             99             99
>
  2.       ClinOff             99             99             99             99
>
  3.      NurseOff      PH NurseB      PH NurseB       MCH Aide       MCH Aide
>
  4.       ClinOff        ClinOff  Nurse/Midwife       MCH Aide             99
>
  5.      MCH Aide    NurseAssist        ClinOff             99             99
>
  6. Nurse/Midwife        ClinOff  Nurse/Midwife             99             99
>
  7.   NurseAssist    NurseAssist             99             99             99
>
  8.       ClinOff    NurseAssist    NurseAssist             99             99
>
  9.       ClinOff    NurseAssist    NurseAssist             99             99
>
 10. Nurse/Midwife             99             99             99             99
>
*******************************************************************************/
```

```
/**
Notice that q102_* vars appear to be character data not numeric.
       The value label "title" is associated to these variables formats
       the numeric data to look like character data.  Click here to see webpages about
       variable labels and value labels    **/



/** use varlist for a list of variables that already exist in a data set */

/** Say you want to disassociate the value label title from the q102_* variables.
    You could type: label value q102_1
                    label value q102_2
                    label value q102_3
                    label value q102_4
                    label value q102_5

    Or you could use a varlist in a foreach command to disassociate
    the value label title with the q102_* vars.  **/


foreach X of varlist q102_1 q102_2 q102_3 q102_4 q102_5{
 label value `X'   /* Notice that the capital letter X is is enclosed in left and right quotes.
                    * This tells Stata to evaluate the local macro X. */
 display "label value `X' "  // Displays in the results window/log file what commands Stata processed.
                             // This is completely unnecessary but helpful in explaining the looping.
}


list q102_*  in 1/10


/***************************************
 In both cases the value label title will still be available for use
 with other variables or even to be re-used again on the q102_1-q102_5 variables.
 If the value label title is not associated to any variable when the data set
 is saved then it will be dropped.
***************************************/




/* Now re-assign the title value label to each q102_* variable.
   This time use a local macro variable in the foreach command.  **/

local titles "q102_1 q102_2 q102_3 q102_4 q102_5"

foreach X of varlist `titles' {
 label val `X' title
}

/** NOTE: The apostrophe on the left is a left apostrophe.
         It is on your keyboard next to the "1" key.
         When a local macro variable is enclosed in a left
         and right apostrophe, it is evaluated as the
         contents of that local macro variable.  **/

list `titles'  in 1/10

/** NOTE: A local macro variable can be used many times throughout a program and
         thus can save a lot of typing.  It can also help keep a foeach command
         from looking very messy, like if you wanted to pass through a foreach
         command 10 or more variables that could not be represented in shorthand
         (like q103_1-q103_10 can be).


    NOTE: `titles' represents the text "q102_1 q102_2 q102_3 q102_4 q102_5".
         Local macro variable names can be up to 7 (not 8) characters long in Stata 6.
         Stata 7 allows them to be up to 31 characters long.

***************************************/




/* Use newlist for variables that are created/generated by the foreach command. */

/* Create string variables title1c, title2c, title3c, title4c and title5c that can hold up to 13 characters. */
```

```
foreach X of newlist title1-title5{
 gen str13 `X'c= ""
}

/** NOTE: A foreach command like:
           foreach X of newlist title* {
             gen str13 `X'c= ""
           }
         would not work because Stata wouldn't know how many title variables to create.  **/



/** Use numlist to insert numbers in place of the local macro var X into a foreach command. **/

/** Use numlist to rename q102_* vars in a foreach command  **/

foreach X of numlist 1/5 {
  rename q102_`X' title`X'
               /** The local macro variable X is replaced with
                   the numbers 1, 2, 3, 4, 5  ***/
}


list title1-title5 in 1/10

/**  NOTE: The above foreach command could have been written with a forvalues command:
 *   forval X= 1/5 {
 *    rename q102_`X' title`X'
 *   }
 *   The capital letter X and could be any letter, upper or lower case.
 *   A good rule of thumb is have all variable names in lowercase.
 **/



/** The anylist is a list of words.  Shorthand notation like title* or
title1-title5 does not work.  **/

foreach X in 1 2 3 4 5 6 7 8 9 10 {
  replace title1c="not missing" if title1==`X'
}

/* The anylist is the most universal type of foreach command since it is not
   restricted to pre-existing variables, new variables, or just numbers.  */



/* More than 1 list of variables can processed at a time. */

/* Replace all values of 99 with missing in title1-title5 and q103_*  */

summarize title5-q103_1

foreach X of varlist title1-title5 q103_* {
  replace `X'= .  if `X' == 99
}

summarize title5-q103_1




/** Two or more lists or types of lists can be processed at the same time. **/

/** Create new variables from existing variables in a foreach command. **/

 /* First create 2 variable list and store them in local macro variables using the
  * the unab Stata command: */
unab varlist1 : title1-title5
unab varlist2 : q103_1-q103_5

 /* Use the display command to see that local macro varlist1 contains the string:
  *  q103_1 q103_2 q103_3 q103_4 q103_5 */
di "`varlist1'"

foreach X of newlist mch_h1-mch_h5 {
 local n= `n' + 1   /* keep count of how many times the loop is processed */
 /* Use the extended macro function ": word # of" to set the local macro variable Y to the n'th variable name */
```

```
 local Y : word `n' of `varlist1'
 local Z : word `n' of `varlist2'
 gen `X'= `Z'  if `Y' == 8
 label var `X' "MCH Aid's hours"
}


/** NOTE: All lists have to have the same number of elements. **/

/* Since all the variables involved have numbers involved in their name, using the forval command would be simpler:
  forvalues X= 1/5 {
   gen mch_h`X'= q103_1`X'  if title`X' == 8
   label var mch_h1`X' "MCH Aid's hours"
  }
*****************************************************************/

list mch_h1-mch_h5 in 23/31


/** NOTE: Stata reads the shorthand varlist of mch_h1-mch_h5 to be all
          variables positionally in the data set between mch_h1 and mch_h5 (not
          necessarily mch_h1 mch_h2 mch_h3 mch_h4 mch_h5).  When using Stata
          interactively, the variable list window in the lower left-hand side
          shows the list of variables in the data set in the order of their position.
          The describe command also shows variable order. **/
```

Note: Dan Blanchette contributed this web page, however please direct questions to Phil Bardsley as noted below.

Review again?

Another topic?

Match merging

Many observations match, but others don't

In this example, we have data from the 1996 Tanzania Facility Survey that we want to merge with data from the 1999 survey. This will allow us to examine trends. The 1999 survey was not designed as a followup of the 1996 survey, so not all the same facilities were visited in the two surveys. To simplify the programming, we've only included facilities in the 1999 file that were also surveyed in 1996.

For this example, we'll look at trends in the availability of the 4 major contraceptive methods, so the two files contain only those 4 variables in addition to the single identifier: facid.

Prior to Stata 11, the data had to be sorted on the identifying variables before merging. Now you have a choice of syntax for merging. We've shown you both methods below, the old which requires sorting, and the new which does not (Stata will sort the data for you when you use the new syntax). Both the old and new syntax work in Stata 11 and after, but the new syntax is better.

The syntax is the same as the "One-to-one merging" example, and in fact this is one-to-one merging. The difference is that in the earlier example we expected a perfect match, but here we don't.

You can type the following commands into Stata, or copy them into a do-file and run them.

Merge using New Syntax (Stata 11 and later)

```
clear
use "q:\utilities\statatut\merge96.dta"
merge 1:1 facid using "q:\temp\statatut\merge99.dta"
drop _merge
/* Look at the Results Window - the merge is a mess! */
```

Merge using Old Syntax (Stata 10 and earlier)

```
/* Sort the 1999 file and save it temporarily */

clear
use "q:\utilities\statatut\merge99.dta"
sort facid
save "d:\statatemp\temp99.dta"

/* Sort the 1996 file */

clear
use "q:\utilities\statatut\merge96.dta"
```

```
sort facid

/* Merge the 1999 data onto the 1996 data */

merge facid using "d:\statatemp\temp99.dta"

/* Check how well the merge went - we don't expect all "3's" */

tab _merge
drop _merge
su

/* Clean up */

erase "d:\statatemp\temp99.dta"
```

Questions:

1. This merge is a mess! Less than half of the facilities in 1996 have a match in 1999. How can we get rid of the non-matching observations? Answer.

2. The 1996 and 1999 files each have 5 variables. How many variables do we expect the merged file to have? Answer.

3. This is a simple example with only 5 variables in each input file. So, it's easy to check whether I've used different variable names in each file. What would happen if I accidentally had a variable in each file with the same name? Answer.

4. How would I know if I had a variable with the same name on each file? Answer.

5. How many data files can I merge with one command? Answer.

Answers:

1. Only 207 of the facilities matched (_merge==3). The remainder were only in the 1996 "master" file (_merge == 1) except for one facility in the 1999 "using" file (_merge == 2). To get rid of the non-matches, we need to use _merge: **keep if _merge == 3**

You can type this command now and see the result. Stata drops 274 observations from the 1996 data in memory and one observation from the 1999 data. The remaining 207 observations are the matched 1996-1999 facilities. Now we can drop _merge, since we no longer need it.

Back to question

---

2. The resulting file should have facid (common to both files), 4 more variables from each input file, and _merge, for a total of **10 variables**

.

Back to question

---

3. Stata by default keeps the values of the "master" file when a merge involves variables with the same name in both files. We planned for this default by renaming the 4 method variables to include the survey year in both files. If we had not done that, the values in the 1999 "using" file would have been lost for matching facilities.

The merge command includes an **update** option to override this default. However, you probably won't need that. It's best to always rename your variables so that the master and using files have unique variable names.

Back to question

---

4. The merged file would have one fewer variables than you expected. That is the only way you would know; Stata does not issue a warning.

Back to question

---

5. In prior versions (8-10) of Stata, you could merge 3 or more files in a single merge command. Frankly, we thought that merging more than 2 files at a time was error prone. There are many ways to miss problems in a merge, and these are compounded when more files are merged at the same time.

So, we're happy to see that in Stata 11 and after you can only merge 2 files together at a time (using the new syntax).

Back to question

Review again?

Another topic?

Merging summary statistics onto each observation

Hierarchical data, analysis at the higher level

This example combines the **collapse** command learned in Adding summary statistics to a data file with the **merge** command covered earlier. We'll use the same data here as the previous example: 503 facilities (which we call "higher" level) in one file, and 2,584 service providers (which we call "lower" level) in another file. But this time we want to combine them in such a way that each observation is a facility, i.e., we want to do our analysis at the higher level.

Our analysis question for this example might be: is the mean number of trained providers at government facilities higher than at private facilities? We need to count trained providers in the provider file (using the **collapse** command) to summarize their data at the facility level, and merge the resulting file onto the facility data.

We use only the new merge syntax here. Please refer to the previous examples of **merge** for Stata 10 and earlier syntax.

Copy these commands into a do-file editor and run them.

```
/* Use the facility file and drop the duplicate. */

clear
use "q:\utilities\statatut\exampfac.dta"
keep facid authorit
duplicates list facid
drop  if facid == 1001 & missing(authorit) // get rid of the duplicate
su
save "d:\statatemp\tempfac.dta", replace

/* Collapse the service provider file to count the trained providers per facility */

clear
use "q:\utilities\statatut\exampro.dta"
keep facid bcs ccs
su
replace bcs=0 if bcs==2
replace ccs=0 if ccs==2
sort facid
collapse (sum) bcs ccs, by (facid)
list in 1/10
su

/* Merge the collapsed provider data with the facility data */

merge 1:1 facid using "d:\statatemp\tempfac.dta"

/* Check how well the merge went - we don't expect all "3's" */

keep  if _merge == 3
su

/* Recode authorit to give 2 groups: govt and non-govt */
/* Note that the private group is labeled "MarieStopes" */
/* because that's the label on value 2. */
```

```
recode authorit 3/12=2
ta authorit

/* Run a t-test to check difference in mean training levels */

ttest bcs, by(authorit)

/* Clean up */

erase "d:\statatemp\tempfac.dta"
drop _merge
```

Questions:

1. Why recode bcs and ccs to 0/1 before the collapse command? Answer.

2. After collapsing the provider data to the facility level, we merged a facility-level file with another facility-level file. Could we have merged them without using facid, in other words, could we have done a one-to-one merge instead of a match-merge? Answer.

3. What were the results of the match-merge? Answer.

4. Do we need to drop the non-matching observations before running the t-test? Answer.

5. Could we use **egen** instead of collapse in this situation? Answer.

Answers:

1. We need to count the number of trained providers at each facility. The easiest way to do that is to add the people who are trained, so we recode them to 1=trained and 0=not trained.

Back to question

---

2. No. In the earlier example, where we merged two files of DHS women's data, the two files originally came from the same combined file. So, we were confident that by sorting first, the women would be in the same order and match one-to-one without using their identifying variables. (We matched on identifiers anyway just to be safe.)

In this case, though, the provider data were collected separately from the facility data. Theoretically, all facilities should be represented in the provider data. But practically, given all the ways field surveys can and do go wrong, it didn't work out that cleanly. Therefore, we needed to match-merge.

Back to question

---

3. The results were the same as in our previous example using these two files. 468 matches, 34 facilities with no matching provider data, and 9 (facility-level) provider observations with no matching facilities. These 9 provider observations are collapsed from the 22 providers we saw unmatched in the previous example.

Back to question

---

4. No we don't. The t-test, and most other Stata statistical commands, drop observations with missing values on any one of the analysis variables. In this case we have 467 observations in the analysis instead of 468, because 1 facility has a missing value for authority.

In fact, if we were to do this analysis using the **svy** commands, we would need to leave all facilities in the sample. The svy commands correct the standard error for the effects of clustering and stratification, and they require the full set of observations in order to make this correction accurately. See Commands to Analyze Survey Data for more information on this topic.

Back to question

---

5. Yes, we could use **egen** to count the number of trained providers, drop the duplicates in the provider file so there was only one observation per facility, and merge the result onto the facility file:

```
/* Use egen to count the trained providers per facility */

clear
use "q:\utilities\statatut\exampro.dta"
keep facid bcs ccs
replace bcs= 0 if bcs == 2
replace ccs= 0 if ccs == 2
sort facid
by facid: egen numbcs= total(bcs)
by facid: egen numccs= total(ccs)
list in 1/10
duplicates drop facid, force
drop bcs ccs
su

/* Merge the collapsed provider data onto the facility data */

merge 1:1 facid using "tempfac.dta"
```

There are a couple of advantages to the egen approach. First, you can easily keep additional variables that may be of interest on the provider file. While this is possible with collapse, it can be cumbersome. Second, you can browse the data after egen to see whether you created the count correctly. Then, if it looks good, you can drop the duplicates.

Back to question

Review again?

Another topic?

Miscellaneous Tips and Tricks

You might find it useful to browse these topics from time to time to see whether they're useful for your data management work.

Getting help for Stata
Updating Stata
Importing and exporting data files
Working with large data files
Shrinking large data files
Error messages
Missing values
The by command in detail
Exporting Stata Results to MS Office
The parmest command
Temporary files
Looping: foreach in detail
Looping with while
Precision and data storage

Missing values

Missing Values and how to work with them

Stata represents missing values with a "." (period) in its results window. If the file has special missing values, Stata represents them as ".a", ".b", ... , ".z". Missing values are stored as values larger than the largest allowable number for the data type. For example, a variable stored as data type **byte** can take the following values:

```
You see:        Stored as:      Treated as:
   1               1               1
   2               2               2
  ...             ...             ...
 100             100             100
   .             101            missing
   .a            102            missing
   .b            103            missing
  ...             ...             ...
   .z            127            missing
```

Most commands ignore missing values by default. Some commands, such as **tabulate**, have an option to display missing if you want to see how many missing observations there are. Other commands, however, may use missing values in a way that will surprise you. For example, the **replace** command does not ignore missing values. Here is a simple example to demonstrate how replace and recode handle missing values differently. In this example, we have three variables with the values of 1, 2, and missing. We want to change all values of 2 to 1.

```
clear
input a b c
1 1 1
2 2 2
. . .
end
list
replace a=1 if a>1
replace b=1 if b>1 & b<.
recode c 1/max=1
list
```

The first **replace** command changes every value that's greater than 1 to 1. This command does not ignore missing values, so both 2 and missing are changed to 1. This probably is not what we would normally want to do, since missing values should remain missing.

The second **replace** command changes all values greater than 1 but less than missing to 1. In this case values of 2 are changed and missing values are not changed, which is our intention.

The **recode** command automatically ignores missing values, so we don't have to think about it. The results are the same as the second replace command.

Review again?

Another topic?

One-to-many merging

Hierarchical data, analysis at the lower level

In this example, we have data from the 1999 Tanzania Facility Survey. We have data from 503 facilities (which we call "higher" level) in one file, and data from 2,584 service providers (which we call "lower" level) in another file. We want to combine them so that each observation is a service provider, i.e., we want to do our analysis at the lower level.

Our analysis question for this example might be: are providers from government facilities more likely to be trained than those from private facilities?

Copy these commands into a do-file editor and run them.

Merge using New Syntax (Stata 11 and later)

```
clear
use "q:\utilities\statatut\exampfac.dta"
keep facid authorit
merge 1:m facid using "q:\utilities\statatut\exampro.dta", keepusing(facid bcs ccs)
drop _merge
/* Check how well the merge went - we do not expect all "3's" */
```

Merge using Old Syntax (Stata 10 and earlier)

```
/* Use the provider file and sort it */

clear
use "q:\utilities\statatut\exampro.dta"
keep facid bcs ccs
sort facid
save "d:\statatemp\tempro.dta", replace

/* Use the facility file and sort it */

clear
use "q:\utilities\statatut\exampfac.dta"
keep facid authorit
sort facid

/* Merge the provider data onto the facility data */

merge facid using "d:\statatemp\tempro.dta"

/* Check how well the merge went - we don't expect all "3's" */
```

```
tab _merge
keep if _merge == 3
drop _merge
su

/* Recode authority to form government vs private */

recode authorit 3/12=2

/* Run chi-square analysis */

tab authorit bcs, row chi

/* Clean up */

erase "d:\statatemp\tempro.dta"
```

Questions:

1. I used the new Stata syntax, and the merge failed! I got an error message:

```
variable facid does not uniquely identify observations in the master data
```

What does this mean, and how can I fix it? Answer.

2. Using the Stata 11 and later syntax, can I first use providers and then merge facilities onto them? Answer.

3. I used the Stata 10 and earlier syntax, and the merge worked, but I got two notes:

```
variable facid does not uniquely identify observations in the master data
variable facid does not uniquely identify observations in tempro.dta
```

What does this mean, and do I need to worry about it? Answer.

4. This merge is not too bad - about 98% of the service providers were matched with their appropriate facilities. How many providers had no matching facility data, and how many facilities weren't paired with providers? Answer.

5. Is it possible to tell from the tabulation of _merge that we're getting the correct number of providers in the merge? How about the correct number of facilities? Answer.

Answers:

1. The problem is that the facility ("master") data has duplicates, that is, it contains more than one observation with the same value of facid. The good news is that Stata stopped you from continuing with the merge, getting the wrong answer, and possibly never discovering the error. (Earlier versions of Stata allowed you to continue, but with a warning - see Answer 2.) The bad news is that you should have checked for duplicates before doing this merge!

Earlier, you saw how to check directly for duplicate identifiers:

```
use "q:\utilities\statatut\exampfac.dta", clear
duplicates list facid
```

That is the easiest way to check for duplicates, and it should always be done prior to a merge.

The error message in our results window refers to the master file, which in this case is the facilities. There may be multiple providers from each facility, so we expect duplicates of facid in the provider (using) file. We told Stata to expect duplicate providers by putting "m" on the right side (corresponding to the using file) of "1:m" in the merge command. Similarly, we told Stata to expect unique values of facid in the facility file by putting "1" on the left side (corresponding to the master file).

Using the duplicates list command, we discovered that our facility data has two facilities with facid == 1001. One of these is a mistake, because it's value of authorit is missing:

```
    +------------------+
    | facid   authorit |
    |------------------|
12. |  1001        Gov |
13. |  1001          . |
    +------------------+
```

To fix the problem we drop that facility and continue with the merge. Back to question.

---

2. Yes, you can merge facilities and providers together in either order. To merge in the other order, the syntax (after fixing the duplicates problem) is:

```
clear
use "q:\utilities\statatut\exampro.dta"
keep facid bcs ccs
merge m:1 facid using "q:\utilities\statatut\exampfac.dta", keepusing(facid authorit)
```

The terms "1:m" and "m:1" tell Stata what to expect when it tries to match observations in the two files. If it finds what you tell it to expect, you're merge will be successful. But, if Stata doesn't find what you expect, it will stop the merge. The only case where it will continue is where you tell it there are many observations with the same identifiers, but in fact there is only 1 instance of each identifier. Since the resulting merged files will be correct, there's no reason for Stata to stop or even to warn you.

Stata also allows a "many-to-many" merge (m:m), but we recommend that you forget you ever heard this. In over 30 years of managing survey data, we have never found an instance where Stata's implementation of this concept fit. In most situations it would result in the wrong answer, and as seen in question 3, that is the case here as well. Back to question.

3. Please read the answer to Question 1 first, then continue here.

As we see here, it's possible to ignore these notes and continue as though the merge worked properly, when in fact we got the wrong answer. (We could even use the new syntax "merge m:m" and continue as though all is well, but the merge would be incorrect.)

Specifically, here's what happens when we ignore the warnings:

As we saw earlier, one of the facilities with facid 1001 is a mistake, because it's value of authorit is missing:

```
    +------------------+
    | facid   authorit |
    |------------------|
12. |  1001        Gov |
13. |  1001          . |
    +------------------+
```

Our provider data contains 5 observations from facility 1001:

```
     +-------------------+
     | facid   bcs   ccs |
     |-------------------|
 29. |  1001     1     1 |
 30. |  1001     2     2 |
 31. |  1001     2     2 |
 32. |  1001     2     1 |
 33. |  1001     1     2 |
     +-------------------+
```

When we allow the merge to continue, our mistake will be hidden from us. Look at the result of the merge by facid:

```
     +---------------------------------------+
     | facid   authorit   bcs   ccs   _merge |
     |---------------------------------------|
 12. |  1001        Gov     1     1        3 |
 13. |  1001          .     2     2        3 |
521. |  1001          .     2     2        3 |
522. |  1001          .     2     1        3 |
523. |  1001          .     1     2        3 |
     +---------------------------------------+
```

Stata matches the first facility, which has a value of "Gov" for authorit, with the first provider. This match is correct. Then it matches the second facility, which has a value of "." (missing) for authorit, with the second provider. It has no new facilities to match with the third provider, so it uses the second facility again, repeating this mistake for the remaining providers from facility 1001.

Despite these mistakes, we think the merge is fine when we see all those 3's in the _merge column. The best way to prevent the error is to check for duplicate facilities **before the merge**. Back to question.

---

4. 22 providers had no facility data, and 34 facilities had no provider data. The facilities were in memory for the merge (the "master" data), so if they had no matches they received a value of 1 for _merge. The providers were on disk (the "using" data), so if they had no matches they received a value of 2 for _merge. Back to question.

---

5. The tabulation of _merge tells us that we have 22 providers with no matching facilities (_merge==2) and 2562 providers with matching facilities (_merge==3). The total is 2584, which is the number of providers in the original file, so all are accounted for.

There's no simple check like this for the number of facilities. Back to question.

Review again?

Another topic?

One-to-one merging

Same observations in each file being merged

The first example is a one-to-one merge of DHS data. The women's file in DHS has more variables than the 2,047 limit in Intercooled Stata. This is not a problem at CPC, where we are running Stata/SE with a limit of 32,767 variables. However, many people run Intercooled Stata, so we are taking a conservative approach in this example. (Type **help limits** to see how many variables your version of Stata can handle.)

Because of this variable limit, we broke the women's file into 2 pieces. Each file has exactly the same women (8,781 observations), but each file has a different set of variables (about 1500 variables in each). When we split the original file in two, we were careful to keep the 3 identifying variables in each file for later merging.

For this example, we selected age, education, and number of children from the first file, and marital status, age at first marriage, and desired family size from the second file. We need to merge these two sample files into a single file for analysis. The identifying variables for this merge are cluster (v001), household within cluster (v002), and line number within household (v003). The order in which the identifying variables are nested determines the sort order.

Prior to Stata 11, the data had to be sorted by the identifying variables before merging. Now you have a choice of syntax for merging. We have shown you both methods below, the old which requires sorting, and the new which does not (starting with version 11 Stata will sort the data for you when you use the new syntax). Both the old and new syntax work in Stata 11, but the new syntax is better for several reasons explained below.

Regardless of which version you are using, it is always a good idea to check your identifiers for duplicates before a merge. These are observations that have the same identifying variables, in this case v001 v002 v003. We expect each combination of these three variables to uniquely identify a woman, but as we see below, two duplicates crept into our data somehow.

Before trying this example, create a folder somewhere on your computer to store intermediate data files. These are files that you need for only for a short time while creating an analysis file, and then you can erase them. At CPC I put them into a folder I named **d:\statatemp**.

Then, copy the following commands into a do-file and submit them.

Check for, examine, and drop duplicates

```
/* Check for duplicates in exampw1 */

clear
use "q:\utilities\statatut\exampw1.dta"
duplicates report v001 v002 v003

/* List the duplicates */

duplicates tag v001 v002 v003, gen(duptag)
list if duptag > 0
```

```
/* Drop the duplicates */

duplicates drop v001 v002 v003, force
drop duptag
save "d:\statatemp\tempw1.dta"

/* Drop duplicates from exampw2 */

clear
use "q:\utilities\statatut\exampw2.dta"
duplicates drop v001 v002 v003, force
save "d:\statatemp\tempw2.dta"
```

Merge using New Syntax (Stata 11 and later)

```
clear
use "d:\statatemp\tempw1.dta"
merge 1:1 v001 v002 v003 using "d:\statatemp\tempw2.dta"

/* Look at the Results Window - should see only matches */

/* Clean up */

drop _merge
```

Merge using Old Syntax (Stata 10 and earlier)

```
/* Sort the first file and save it temporarily */

clear
use "d:\statatemp\tempw1.dta"
sort v001 v002 v003
save "d:\statatemp\tempw1.dta", replace

/* Sort the second file and save it temporarily */

clear
use "d:\statatemp\tempw2.dta"
sort v001 v002 v003
save "d:\statatemp\tempw2.dta", replace

/* Use the first file, merge on the second file */

clear
use "d:\statatemp\tempw1.dta"
merge v001 v002 v003 using "d:\statatemp\tempw2.dta"

/* Check how well the merge went - should see only "3" for _merge */

tab _merge
su

/* Clean up */

erase "d:\statatemp\tempw1.dta"
erase "d:\statatemp\tempw2.dta"
drop _merge
```

Questions:

1. What does the **duplicates** command do? Answer.

2. The file **tempw1.dta** has 8,779 observations (after dropping two duplicates) and 6 variables. The file tempw2.dta also has 8,779 observations and 6 variables. After merging them, how many observations and variables do you expect the merged file to contain? Answer.

3. There is a 10th variable on the merged file, named **_merge.** What is its role? Answer.

4. What does the command **erase** do? Answer.

5. Why **drop _merge**? Answer.

6. Is there an easier way to work with intermediate data files? Answer.

Answers:

1. The **duplicates** command checks whether the variables you list identify observations in your data uniquely. If more than one observation has the same combination of identifying variables, **duplicates report** will tell you. You can then tag the duplicates and examine them using **duplicates tag**.

To examine the duplicates, you can list them as we show above. Or you might use the Data Editor in browse mode (**browse**), so you don't accidentally change the data:

```
browse  if duptag > 0
```

At this point, if you have access to the questionnaires or other information about the raw data, you may be able to resolve the duplicates problem. In this case, though, we do not have access to the questionnaires, so we used **duplicates drop** to arbitrarily drop all but the first instance of each duplicate. (The command drops all but the first instance, given the current sort order of the data.)

Back to question

---

2. It is a good idea to anticipate the results of a merge, so that when you see the actual results, you will know immediately whether the merge worked the way you intended it.

In this case, a one-to-one merge, we expect the resulting file to have the same number of observations as the two original files: 8,779. The two files have 3 ID variables in common, so the merged file will have 3 ID variables. The two files have 3 additional variables each, so the

merged file will have 6 variables in addition to the IDs, for a total of **9 variables**.

Back to question

---

3. Stata creates the variable **_merge** when you ask it to merge two files. The variable can have 3 different values as follows:

- 1= this observation comes from the "master" file only (the file in memory)
- 2= this observation comes from the "using" file only (the file on disk)
- 3= this observation contains variables from both the master and using files (a match)

In this case, all observations have variables from both the master and using files, so they all have a value of 3 for _merge.

Back to question

---

4. After sorting the master and using files, we saved each one on disk with the names tempw1.dta and tempw2.dta. When we no longer needed them, we deleted them from disk with the **erase** command.

Back to question

---

5. At some time in the future, we may want to merge another file onto this new file. During the merge, Stata will again want to create the variable _merge. Since the variable already exists, Stata will stop the merge command and write an error message to the log.

The easiest way to handle this is to always **drop _merge** after looking at it. If you want to keep it around, you can rename it, or you can ask Stata to create it with a different name like "merge1" (using the **generate** option on the merge command).

Back to question

---

6. Intermediate data files can be handled in two ways. One way is shown here, where we create a permanent file and then erase it when we no longer need it. The second way is to use Stata temporary files, which Stata automatically erases at the end of your interactive Stata session. This second method is described in temporary files in the Miscellaneous Tips and Tricks section of this tutorial.

It is up to you to decide which way is easier!

Back to question.

Review again?

Another topic?

Precision and data storage

Two precision issues come up repeatedly when using Stata (and other similar analysis packages). One is how decimal values are represented in the computer's memory. The other is how large an integer you can store in a given Stata data type.

When 0.7 doesn't equal 0.7

Computers use a binary (0's and 1's) system to store decimal numbers. This leads to some inaccuracy, since some decimal values can't be stored exactly in binary. Try this:

```
clear all
set obs 1
gen x= 0.7
list
list if x == 0.7   // 0.7 doesn't equal 0.7
browse
list if x == float(0.7)   // now they are equal
```

You'll notice that the command **list if x == 0.7**results in nothing being listed! When you browse the data, you'll see that 0.7 is being stored as the value 0.69999999. Since that value isn't 0.7, your command to list x results in no matches.

The **float** function takes care of this problem - it rounds the value 0.69999999 to 0.7. Many decimal values are stored accurately in binary, for example 0.5, but many are not. Rather than trying to memorize which are and which are not, we suggest always using the float function.

Big integers

The other precision issue has to do with Stata's data types. Stata offers 3 data types for integers (byte, int, and long) and 2 for floating point (float and double). For character data, Stata offers data type string. If you type **help data_types**you'll see a table that lists the 5 numeric data types that Stata uses, and the string data type, along with the minimum and maximum values that can be stored in each data type, and the maximum value that can be stored precisely.

Why data types?When Stata was first being developed, computers had very little random-access memory, and RAM was expensive. So, there was a benefit to storing values in as little memory as possible. While it would be convenient to create all variables with the highest precision available, which for numeric data is type double, this would waste a lot of memory. For example, a typical yes-no (1,2) variable can be stored accurately in a single byte, so storing it in type double would waste 7 bytes per observation. In a data file of survey results with thousands of variables and thousands of observations, this adds up many megabytes of wasted storage.

Now that computer memory is less expensive, we tend to pay less attention to it. But we need to pay attention when creating values in Stata that are relatively large. Typically, this occurs when the researcher decides to create a single numeric identifier out of multiple, nested identifying variables. In the Demographic and Health Survey data, one can often use three variables (the sampling cluster, the household identifier within the cluster, and the person identifier within the household) jointly to identify an individual respondent. For example:

```
    duplicates report v001 v002 v003
```

usually demonstrates that these 3 nested variables create a unique identifier (but not always - be sure to check). But, if you try to combine them into a single numeric variable, you may run into trouble:

```
gen id= (v001*1000000) + (v002*1000) + v003
```

This example creates an 8- or 9-digit number, depending on the values of the cluster identifier v001. But Stata will store id **by default in type float**. Data type float begins to lose precision above 7 digits. We might be lucky and create a unique identifier, but probably not.

There are two ways to get around this. First, specify **double** when generating large integers. Data type double can accurately represent integers up to 15 digits:

```
gen double id= (v001*1000000) + (v002*1000) + v003
```

While double is discussed in the data type help page in terms of floating point precision, it works well for integers too, and it's the only way to store big integers precisely.

The other way to do this is using data type string for the identifier. In fact, the DHS data includes a string identifier, called caseid for the individual respondent. Strings are a bit of a pain to work with, but they precisely hold integers up to 244 characters in length.

We recommend that you always check for duplicates when creating a composite identifier. And always use **data type double** for these big integers. Some people even recommend that you always use data type double, which you can do with:

```
set type double, permanently
```

This asks Stata to create all new variables with data type double, greatly reducing your need to worry about precision. When you're finished creating an analysis file, you can **compress**the file. This command asks Stata to decide how each variable can be stored most efficiently.

The Stata Corp. web site offers many FAQ's on topics like this. Here's one on the precision of floating point storage that you might find useful: The accuracy of the float data type

Note: Dan Blanchette contributed this web page, however please direct questions to Phil Bardsley as noted below.

Review again?

Another topic?

Reshaping a data file

Converting variables to observations, observations to variables.

Most population surveys gather information about household members in a household roster. The data are organized with one observation being a household. Information about each member of a household is in different variables on the same observation. For example, the age of the first household member might be in the variable age01, the second in variable age02, and so forth.

We may need to summarize the data about household members. Or we may need to merge the household roster information with other information collected from selected members on different forms. In either case, it would be more convenient if the household roster variables were organized differently, with each observation being a member (instead of a household).

Stata offers the **reshape** command to make this transformation easier. Stata calls the original data format "wide", where household members are represented as variables on a household observation. They call the new format "long", where each household member is a separate observation. So, the command to transform the data from wide to long is **reshape long**, and from long to wide is **reshape wide**.

The example below uses a household roster with up to 10 members. **Browse** the data before and after each reshape to help visualize the transformation.

```
clear
use "q:\utilities\statatut\hhwide.dta"
summarize

/* Reshape from wide form to long form (each member becomes an observation). */

reshape long age edu rel sex, i(hhid) j(lineno)
summarize

/* Reshape from long form to wide form (each member becomes a set of 4 variables). */

reshape wide age edu rel sex, i(hhid) j(lineno)
summarize
```

Questions:

1. What do the terms "age", "edu", "rel", and "sex" stand for in the reshape command? Answer.

2. What is the term "i(hhid)"? Answer.

3. What is the term "j(lineno)"? Answer.

4. Why are there 5000 values for lineno in the long file, but only 2564 values for age and the

other variables? Answer.

5. What happens to the variable "lineno" when we convert from long to wide? Answer.

6. The variable "electric" is a household-level variable. We want it to be added unchanged to each observation in the long form. Is that what happens? Answer.

Answers:

1. These are the prefixes for the variable names in the original (wide) form of the data. Reshape works most easily if the variable names have a prefix followed by a number. In this case we have the variable name prefix "age" followed by a number 1-10 corresponding to the household member's order in the roster. If your variable names don't have this prefix-number form, you may want to **rename** them before doing reshape. See the explanation of **foreach** elsewhere in this tutorial for an easy way to rename multiple variables. Alternatively, you may want to explore the advanced features of reshape that allow you work with other naming conventions without renaming the variables. See the Stata manual for details.

Back to question

---

2. The reshape command needs a common identifier for each member of a household. **Reshape long** assigns this identifier to each household member, while **reshape wide** uses this identifier to assign each member to a household. In this case, that identifier is named hhid in the original data. We would need this identifier in order to collapse the data to the household level or to merge data from other files onto the new long file.

Back to question

---

3. The **reshape** command needs an identifier for each individual in the household. Typically, this identifier is the line number in the household roster. The "j(lineno)" term tells **reshape long** to create a new variable named lineno. This variable captures the position of the household member in each household observation in the original (wide) file. The **reshape wide** command uses this variable to assign a suffix to the new household-level age, education, relationship, and sex variables it is creating. In **reshape long**, we can choose any name we want for the "j" variable, so we chose lineno, while in reshape wide we must use the variable that actually identifies each individual in the household. We would need this line number, together with the hhid variable, to merge individual-level data with the new long file.

Back to question

---

4. **Reshape** creates one new observation for each set of 4 variables in the wide file. We have 10 sets of 4 variables, so we get 10 observations. The wide file has 500 households. Multiplied by 10 members, that's 5,000 new observations in the long file. However, not every household in the wide file has 10 members - some have fewer. Nevertheless, **reshape** creates a new observation for each set of variables, missing or not. The **summarize** command tells us that these 500 households have 2,564 members distributed among them, with anywhere from 1-10 members in each household. We probably would **drop** any observation with all missing data before doing anything further with the long file. The command would be:

```
drop  if missing(age)
```

Note: you should not use edu to drop observations with missing values, because 8 people are missing education but have values for all the other variables. They shouldn't be dropped.

Back to question.

---

5. The variable lineno disappears going from long to wide. Its values become the suffix in the variable names age, edu, rel, and sex.

Back to question

---

6. Yes, the variable electric is automatically added to each new observation generated by the reshape long command. We do not name it in the reshape command, and Stata carries it along unchanged.

Back to question

Review again?

Another topic?

Shrinking large data files

In an earlier example, we mentioned the **compress** command as a way to reduce the size of Stata data files. It works by choosing the most efficient data type that is necessary to store each variable and still maintain the precision of your data.

When working with very large files, or on computers with limited RAM, you need to compress the data. Below is an example you can try to see how compress works.

```
clear
use "q:\utilities\statatut\excomp.dta"
set more off
compress
```

Questions:

1. All these variables started as data type "float" or as short strings. How much is the memory need reduced when compress changes floats to bytes? Answer.

2. What does **set more off** do? Answer.

3. How can I tell whether compress will make much difference in my RAM requirements? Answer.

4. This is a "Catch-22"! I can't fit my Stata data into memory on my computer, but I can't compress it unless I can get it into Stata. Answer.

5. I have a Windows compression program on my PC. Can I use that instead? Answer.

Answers:

1. A float variable requires 4 bytes, while a byte variable requires 1 byte. See **help data types**

in Stata for details on the amount of storage required by each data type.

Back to question

---

2. You'll recall from an earlier example that **--more--** shows on the bottom of the Stata results window when a command generates more lines than can fit in that window. When you see --more-- you need to press the space bar to continue viewing results.

You can tell Stata to continue scrolling the results and not stop when the screen fills up. The command is **set more off**. To turn more back on, use **set more on**. That way you can go get a cup of coffee while Stata compresses your giant survey file.

Back to question

---

3. You can look at the current data type of the variables using the describe command. If you see a lot of doubles (8 bytes) or floats (4 bytes) and you know your data are mostly 1 or 2 digit values, you'll know that compress will make a great deal of difference. If your data are already stored in bytes, compress won't help much.

Back to question

---

4. There are two ways to get around this problem. You can find another computer with more RAM than is available on yours, and compress the file there. Or, you can use the **varlist** option on the **use** command to bring a subset of variables into memory. This will allow you to split the file into two or more pieces, each of which is small enough to fit in RAM and compress. Then you can recombine them, if necessary, into a single file. An example of this was shown in the discussion of One-to-one merging

. Remember to include the necessary identification variables in the smaller files you create so that you can merge if you have to.

Back to question

---

5. Compression programs like WinZip and PKZIP use a different compression method. Stata cannot read data files that have been compressed by those programs, or by Unix commands such as compress, gzip, or tar.

Back to question

Review again?

Another topic?

Temporary files

Saving and using temporary files on disk

In previous examples we often created a file on disk that we needed only for the duration of the example. We erased the file at the end of the example to clean up after ourselves.

Stata supplies the **tempfile** command to get around creating and erasing permanent files. However, temporary files are a bit tricky to work with, and I don't recommend using them in everyday work. They're best suited to programs, that is sets of Stata commands that are executed together. Most researchers work with a few commands at a time in a do-file, which makes temporary files cumbersome. For completeness, however, here is an example of the tempfile command:

```
clear
use "t:\statatut\exampw1.dta"
sort v001 v002 v003
tempfile temp1    /* create a temporary file */
save "`temp1'"      /* save memory into the temporary file */
use "t:\statatut\exampw2.dta"
sort v001 v002 v003
merge v001 v002 v003 using "`temp1'" /* use the temporary file */
```

Questions:

1. The name temp1 in the save and merge commands is enclosed in single quotes. Why is that? Answer.

2. Why don't we need to erase temp1 after we're finished with it? Answer.

Answers:

1. The name "temp1" is actually a temporary variable name, which is called a "macro" in Stata. In this case it refers to a temporary file. Looking in the log you'll see that the actual temporary file is named something like:

```
    C:\TEMP\ST_040001.tmp
```

The macro "temp1" contains that value. To get that value, we need to put the accent mark (`) in

front of the macro name and a single quote after the macro name.

Note that you must use the "backward" single quote (accent mark on the tilda key) at the beginning of your macro name, and the "forward" single quote (apostrophe on the double-quotes key) at the end of your macro name when you refer to the file.

Back to question

---

2. We don't need to erase temp1 because Stata takes care of that for us.

Back to question

Review again?

Another topic?

The by command in detail

More examples of the by command.

This page is intended to show how to use the **by** command to create variables and access specific observations in sub-groups. Sub-group processing breaks down the dataset into multiple mini-datasets. The way to think about it is as if each sub-group is the entire dataset.

The way to access data from different observations is with internal Stata variables **_n** and **_N**. The variable **_n** is equal to the numeric position of an observation and **_N** is equal to the total number of observations. For example, in a dataset of 10 observations, in the first observations **_n** is equal to 1 and **_N** is equal to 10. In the second observation **_n** is equal to 2 and **_N** is equal to 10, etc.

```
v001 _n _N
---- -- --
1002  1 10
1002  2 10
1002  3 10
1002  4 10
1002  5 10
1003  6 10
1003  7 10
1003  8 10
1004  9 10
1004 10 10
```

When using the **by** command, **_n** is equal to 1 for the first obs of the by-group and **_N** is equal to the total number of observations for the by-group. So if a person has 5 observations in the data set and **v001** is the id variable, "**by v001:** " creates by-groups where the first observation in the by-group **_n=1** and **_N=5**. The second observation **_n=2** and **_N=5** , etc. The last observation for that person **_n** will equal **_N**.

Note: **if** conditions in the command being run by the **by** command do not subset the by group so **_n** and **_N** are not affected by the **if**condition.

```
by v001:
```

makes the data look like this:

```
v001 _n _N
---- -- --
1002  1  5
1002  2  5
1002  3  5
1002  4  5
1002  5  5
1003  1  3
1003  2  3
1003  3  3
1004  1  2
```

```
1004  2  2
```

If you want to keep the first obervation per person do:

```
by v001: keep  if _n == 1
```

second observation is:

```
by v001: keep  if _n == 2
```

second to the last observation is:

```
by v001: keep  if _n == _N - 1
```

last observation is:

```
by v001: keep  if _n == _N
```

The following is Stata code that plays with this concept.

NOTE: the following is a **do-file**. Highlight and copy all text between the horizontal bars and paste into the Stata interactive do-file editor window so that you can submit this do-file.

```
use "q:\utilities\statatut\examfac2.dta",clear

keep facid factype q102_1-q103_20
** q103_1-q103_20 are # hours/week usually worked **

** facid is the id var **

** q102_1-q102_20 are titles of facility workers **
label list title type

/** The following code figures out how many total hours per week
  * different types of workers at a health facility work. **/
sort facid

/* Get rid of duplicate ids. */
drop  if facid == facid[_n-1]

/* This would also keep the first obs in a duplicate id situation:
 * by facid: keep  if facid[_n] == 1 */

/* Make the data set multiple obs per facility. */
reshape long q102_ q103_, i(facid) j(position)

/** Save the data set as a temporary data set in case you want to try another idea. **/
preserve
gen ttl_h1= 0
replace ttl_h1=q103_  if q103_ != 99 & q102_ == 1

by facid: gen ttl_th1=ttl_h1[1]+ttl_h1[2]+ttl_h1[3]+ttl_h1[4]+ttl_h1[5]+ ///
   ttl_h1[6]+ttl_h1[7]+ttl_h1[8]+ttl_h1[9]+ttl_h1[10]+                   ///
   ttl_h1[11]+ttl_h1[12]+ttl_h1[13]+ttl_h1[14]+ttl_h1[15]+               ///
   ttl_h1[16]+ttl_h1[17]+ttl_h1[18]+ttl_h1[19]+ttl_h1[20]

list facid factype q102_ q103_ ttl_h1 ttl_th1 in 1/500 if factype==3

/* or you could use the -forvalues- command */
replace ttl_th1= 0
```

```
forvalues num= 1/20 {
 by facid: replace ttl_th1= ttl_h1[`num'] + ttl_th1
}

/* If you wanted totals for all the other titles you'd have
 * to repeat the above 19 more times. */

/* You could use the -egen- command to sum up work hours per title for each facilty
 * and for commands to reduce the amount of typing. */

clear
restore
preserve

foreach var of newlist ttl_h1-ttl_h10 {
 gen `var'= .
}

forvalues num = 1/10 {
 replace ttl_h`num'= q103_  if q103_ != 99 & q102_ == `num'
 egen ttl_th`num'= sum(ttl_h`num'), by(facid)
}

// examples of when factype == 3 in the first 500 obs
list facid factype q102_ q103_ ttl_h1 ttl_th1  in 1/500  if factype == 3

/** Keep only the variables that represent all obs of the facility. **/
keep facid factype ttl_th1-ttl_th10

/** Keep only the last obs per facid, though any of it's obs would be just as good. **/
by facid: keep  if _n == _N

list  in 1/500  if factype == 3
```

This page was contributed by Dan Blanchette, however please direct questions to Phil Bardsley as noted below.

Review again?

Another topic?

The parmest command

The **parmest** command follows any Stata command that produces parameter estimates, and saves the results in a Stata dataset, the Results window, and/or the log file. It saves the results as one observation per parameter in a Stata dataset. You can format the results for printing and publication, e.g. rounding estimates to 2 decimal places.

Parmest does not come with the Stata software but is available through the SSC archive. If you are not on the CPC network, you can download the latest parmest package by typing at a Stata command line:

```
ssc install parmest
```

You need to have an active connection to the internet to do this. If you have a copy of parmest but are unsure how recently it was downloaded, install it again using the "replace" option.

Here's an example showing how to save the results of a regress command:

```
use "t:\statatut\examfac2.dta"
reg age factype authorit urbrur femster
parmest, saving("c:\tables\reg1.dta", replace)
```

You now have a copy of the results saved in a dataset. To list the results in the Results Window in a format that allows you to copy them to Excel:

```
use "c:\tables\reg1.dta", clear
list parm estimate t p, noobs clean

       parm     estimate            t           p
    factype   -3.5400133     -4.16595   .00004732
   authorit   -.56049792   -1.1524406   .25061116
     urbrur   -2.2121531   -1.4140991   .15899552
    femster   -3.2143824   -.98375474   .32650721
      _cons    51.359101    8.4572685   7.705e-15
```

You can now highlight the table of results, right click, copy table, and paste into Excel.

Type **help parmest** for more information. See Exporting Stata Results to MS Office for another way to put results into a Stata dataset.

Review Again?
Another topic?

Updating Stata

How to update Stata

Stata Corp. makes updates to your current version available for free on their website. If you have your own Stata license, such as on a standalone PC or laptop, it's a good idea to make a habit of checking once a month for updates. (At CPC, we keep the network copies up-to-date for you.)

To check whether your copy is up-to-date, first make sure that your computer is connected to the Internet, then click on Check for Updates in the Help menu, or type the command:

```
update query
```

This instructs Stata to first check when you last updated it, then connect to www.stata.com and check whether a newer version is available. If updates are available for you, it will instruct you how to get the updates.

You can also ask Stata to check for updates automatically. Go to the Edit menu and click on Preferences, General Preferences, Internet, and check the box for Enable automatic update checking.

If you haven't updated your copy for some time, or if your connection speed is slow, your connection may time out before all the updates have been downloaded. The default connection period is 5 minutes (300 seconds). If you need to increase this, type:

```
set timeout2 sss
```

where "sss" is the number of seconds. The maximum is 32,000.

If you don't have Internet connectivity for your computer, you can download the updates at CPC, copy them to a flash drive or other portable storage, and install them on your computer at a later date. See Keeping Stata up to date for complete instructions.

Review again?

Another topic?

Using permanent Stata data files

Clearing memory, using a Stata file, saving changes to the file.

In Example1 you created a permanent Stata data file called myfile.dta. This example uses that file. Type each of these commands and observe the results:

```
clear
use myfile
drop d
save myfile
save myfile,replace
```

Questions:

1. The **clear** command removes data from Stata's memory. Why do we need to use this command before copying a new data file into Stata's memory? Answer.

2. The **use** command copies a Stata data file into Stata's memory. How would you change the use command if the Stata data file was not in the present working directory? Assume it was in the following path:

```
e:\student\jdoe\myproject\data\myfile.dta
```

Answer.

3. What does the **drop** command do? Answer.

4. Why do you get the error message "file myfile.dta already exists" when you type the **save** command? Answer.

Answers:

1. Stata can only keep one set of data in memory at a time. In order to protect you from accidentally writing over and destroying your data in memory, Stata requires that you clear memory before either of these tasks.

Back to question.

---

2. Change the use command to include the full path:

```
use "e:\student\jdoe\myproject\data\myfile.dta"
```

By the way, if you make a mistake typing a Stata command, especially a long one like this, you don't have to retype the whole command. You can press the **Page Up** key to display your last command, edit it, and press Enter to resubmit it. Or, you can click on the command in the Review window to display it for editing.

Back to question.

---

3. The drop command drops one or more variables. We cover it in detail in the next example. It's included here only to make this example a little less artificial.

Back to question.

---

4. Stata is warning you that you are about to write over an existing file on disk. It's giving you a chance to think about whether that's what you really should be doing. The replace option reassures Stata that you know what you're doing:

```
save myfile,replace
```

Back to question.

Review again?

Another topic?

Value labels

Value labels: words in place of numbers.

Value labels give more description than the integer values of a variable and can make it more user-friendly.

Stata does *not* allow value labels to be associated to a range of values (eg. 1-10 "low" 11-20 "med" 21-30 "high"). It is possible to get the same results by defining the same value label one-by-one to all integer values in the range. The foreach command may come to your rescue if the range is long. In this case, though, it would make more sense to create a recoded variable with 3 values (1="low", 2="med", 3="high").

Stata does *not* allow value labels to be associated with non-integer data (e.g., 1.5 "low" 2.1 "med" 3.3 "high") or with character strings (e.g., "Yes" "No").

Examples of the **label** command are shown below in a **do-file**.

```
clear
use "q:\utilities\statatut\examfac2.dta"
keep facid q102_1-q102_5

* Confirm that the variables are numeric (in this case "byte").
* Value labels named "title" are associated with the variables.

describe

* Many commands, like browse, show the labels rather than the values.

browse

* To see the values, use the "no label" option.

browse, nolabel

* List all value labels stored in examfac2.dta.

label dir

* Show what the value label title is.

label list title

* These numbers are the actual data that STATA stores for each variable.
* Notice that there is no value label for responses of 99.  Stata prints
*   the actual data for un-formatted responses.

* Remove the value label title.

label drop title
browse

* Re-create the title value label.

#delimit ;
 label define title
            1 "Doctor"
            2 "AssistMedOff"
            3 "ClinOff"
```

```
            4 "AssistClinOff"
            5 "NurseOff"
            6 "Nurse/Midwife";

* Add more to the title value label.;

label define title
            7 "PH NurseB"
            8 "MCH Aide"
            9 "NurseAssist"
           10 "Other"
           11 "Dr. Doolittle", add;
#delimit cr

* Change the title value label.

label define title 11 "", modify

* Re-assign the title value label to each q102_* variable.

foreach v of varlist q102* {
   label val `v' title
}
browse
```

Questions:

1. How many characters long can a value label be? Answer.

2. Can I see both the values and the value labels at the same time? Answer.

3. Can value labels have the same name as the variable? Answer.

Answers:

1. The maximum length for a value label is 32,000 characters. See **help limits** for this and almost all other limits in Stata.

Back to question

---

2. The **label list** command in the above example gives a kind of codebook for the values and labels. Otherwise, most commands give you either the value or the label but not both. One exception is the **fre** command, written by Ben Jann and available on CPC computers or by download from the SSC archive: **ssc install fre** It works like **tab** but shows both values and value labels in the tabulation.

Back to question

---

3. Yes. For example, you could have a value label **gender** assigned to the variable **gender**

.

Back to question

Review again?

Another topic?

Variable labels

Variable labels: describe your variables.

Variable labels document your data and make it more user-friendly.

Examples of the **label variable** command are shown below in a **do-file**.

```
use "q:\utilities\statatut\examfac2.dta", clear

keep facid q102_1-q102_5

/** show variable labels **/
describe

/** remove variable labels **/

label variable q102_1 ""

label variable q102_2 ""

label variable q102_3 ""

label variable q102_4 ""

label variable q102_5 ""

describe

/** re-create variable labels */
label var q102_1 "Staff member 1 job title"

label var q102_2 "Staff member 2 job title"

label var q102_3 "Staff member 3 job title"

label var q102_4 "Staff member 4 job title"

label var q102_5 "Staff member 5 job title"

describe
```

Questions:

1. How many characters long can a variable label be? Answer.

2. Can more than one variable be assigned a variable label in one label command? Answer.

Answers:

1. The maximum length for a variable label is 80 characters. See **help limits** for this and almost every other limit in Stata.

Back to question

---

2. No. Only one variable can be processed in a label variable command. However, you can use a command like **foreach** or **forvalues** to create labels for many similar variables in a single loop, which saves a lot of typing.

Back to question

Review again?

Another topic?

A simple example

Inputting data, generating a new variable, listing and saving data, log files.

Start Stata by double-clicking on the desktop icon. Type each of these commands in the Stata Command window, one at a time, pressing Enter after each line:

```
input a b c
1 5 10
0 8 7
1 4 6
end
generate d=c-b
list
browse
save myfile
```

Questions:

1. The **input** command puts data into Stata's memory. What is the result of the input command? In other words, where are those numbers stored? Answer.

2. What does the **generate** command do? Answer.

3. The **list** and **browse** commands list the contents of Stata's memory. Where are the results of all Stata commands displayed? Answer.

4. The **save** command writes a permanent disk file. Where is the file "myfile" created? Answer.

5. What does the file "myfile" contain? Answer.

6. What are the three most important default actions of a simple Stata program like the one above? Answer.

7. What would you change in the program to make it read one thousand records placed after the input command instead of three? Answer.

8. What would you change in the program if your data records were in a file separate from the program? Assume your data file has the following path:

```
e:\student\jdoe\myproject\data\part1.dat
```

Answer.

9. Describe the result if you typed the generate command before the input command:

```
generate d=c-b
input a b c
1 5 10
0 8 7
1 4 6
end
```

Answer.

10. Describe the result if you typed the variable name "C" in caps in the generate command.

```
input a b c
1 5 10
0 8 7
1 4 6
end
generate d=C-b
```

Answer.

Answers:

1. The numbers are stored in the desktop PC memory (RAM). There is no file, temporary or permanent, created on disk. The data are stored conceptually as 3 rows (observations) and 3 columns (variables), like an Excel spreadsheet.

Back to question.

---

2. It creates a 4th column (variable) named "d" with one value for each observation.

Back to question.

---

3. Stata echos each command that you type and displays the results in the Stata Results window. An exception is **browse**, which opens a new window. The **log** command will create a permanent file on disk with the contents of the Stata Results window while you type commands.

```
input a b c
1 5 10
0 8 7
1 4 6
end
generate d=c-b
log using mylog.log
list
log close
save myfile
```

Here the log command creates a file named mylog.log containing the results of the list command but none of the other contents of the Stata Results window.

Back to question.

---

4. The save and log commands create files in the present working directory (pwd). This directory is displayed in the lower left corner of the Stata window. You can change it with the cd command. You can also specify a full path name in the save and log commands. Stata automatically adds the suffix ".dta" to files created with the save command.

Back to question.

---

5. The save command creates a Stata-format file. It contains the contents of Stata's memory at the time you type the save command. In this case, the file would have 3 obs and 4 variables.

Back to question.

---

6. Default actions:

- all file-related commands (save, log) work in the present working directory
- all variable-related commands (generate) affect every observation in Stata's memory
- commands are executed in order from the top down

Back to question.

---

7. Nothing.

Back to question.

---

8. Replace the **input** command with the **infile** command:

```
infile a b c using "e:/student/jdoe/myproject/data/part1.dat"
```

Note the quotes around the path and file name. Quotes are only necessary if a directory has blank spaces in it, but it's a good habit to get into. Also, note that the "/" (slashes) can be either forward or backward - Stata doesn't distinguish between them.

Back to question.

---

9. The generate command results in an error message "c not found", since Stata doesn't yet know about the variable c.

Back to question.

---

10. The generate command results in an error message "C not found", since Stata distinguishes between upper and lower case.

Back to question.

Review again?

Another topic?
