Review of Stata
AERC Technical Workshop
Nairobi, 20-24 May 2002

This note is designed to give you a review of the basic tools available in the statistical package called Stata (pronounced "stay-tuh"). Stata has many built in commands to do such things as simple regression, two- and three-stage least squares, probit/logit/tobit models, Heckman selection models (maximum likelihood and two-stage methods), and many more. As such, most of an analyst's time spent using this program involves manipulating data and output from the various procedures. Of course, you can also write your own estimation procedures, but we'll concentrate on the basics of data management.

Data analysis with Stata is essentially a simple three-step process:

1. Load the dataset
2. Issue a set of commands
3. Examine/save the output

With this in mind, the outline for this note is as follows:

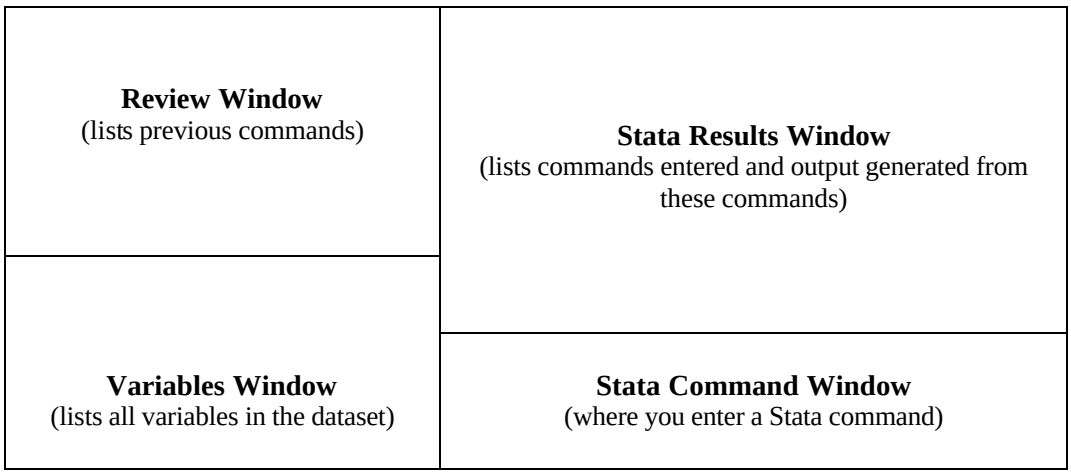
A. Stata Environment – opening & closing Stata
B. Creating and Understanding Stata Datasets
C. Data Modification and Editing
D. Merging Datasets

The next note will continue with additional commands, estimation and post-estimation output.

A. The Stata Environment

To open Stata, double click on the **Stata icon**.

The Stata environment is made up of four main windows which are located as follows:



Stata Command Window: Where all commands given to Stata are typed.

Stata Results Window: Where results of commands appear.

Review Window: Where a list of all previous commands appears.

Variables Window: Where a list of all the variables contained in the dataset currently loaded appears. This window is empty when you first start a Stata session because a dataset has yet to be loaded into memory.

To exit Stata, type `exit` in the **Stata Command Window** and then press "Enter", or click on the "X" box in the upper-right-hand-corner of the Stata window.

B. Creating and Understanding Stata Datasets

Before any type of analysis can be performed in Stata, it is necessary to access a Stata dataset. We will go through an exercise of one of the many ways of creating a dataset to get a better understanding of how Stata handles data. At the end of this discussion, we list the other means of loading data into Stata.

Step 1: Open a small dataset in Excel

Open Excel and open the following spreadsheet:

```
c:\aerc\stata_review\data\exercise.xls
```

in this spreadsheet you will find the following…

	A	B	C	D	E
1	name	sex	age	from	kids
2	Winnie	2	30	Nairobi	0
3	Amare	1	33	Addis	2
4	Leopold	1	28	Dakar	1
5	Kristin	2	40	Cleveland	4
6	Christy	2	50	Lagos	3
7	Jean Luc	1	35	Antananarivo	0

(Note: In this example, sex is coded 1 for males and 2 for females)

Step 2: Copy the dataset in Stata

a. Open Stata

b. To facilitate our work, change the working directory for Stata by typing the following command in the **Command Window** and then pressing "Enter":

```
cd c:\aerc\stata_review\temp
```

c. Now go back to the Excel spreadsheet. Highlight the data and copy it (press `ctrl` and `c` at the same time).

d. Go back to Stata and type the following into the command line:

```
edit
```

(or you can click on **Window**, and then on **Data Editor**). The data editor will open and you can paste the data into this editor (press `ctrl` and `c` at the same time, *or* click on **Edit**, and then on **Paste**). Now close the data editor by clicking on the **X** in the upper right hand corner of the **Stata Editor Window**.

Note that all Stata commands are in lower case type, and Stata is case sensitive when it comes to names (e.g. the variable `Name` is not the same as `name`).

You will notice that a list of variables will appear in the **Variables Window** to indicate that the dataset is now loaded into memory. You will also notice that the commands that you just entered appear in the **Review Window**. The following will appear in the **Stata Results Window**:

```
. edit
(5 vars, 6 obs pasted into editor)
```

This indicates that the Stata dataset has 5 variables and 6 observations.

Step 3: View the Data

We will now go through some basic commands that give us information about our dataset.

a. In the command line, enter the following command

```
describe
```

The following should appear in the **Stata Results Window**

```
. describe

Contains data
  obs:             6
 vars:             5
 size:           162 (100.0% of memory free)
------------------------------------------------------------------------------
   1. name      str8   %9s
   2. sex       byte   %8.0g
   3. age       byte   %8.0g
   4. from      str12  %12s
   5. kids      byte   %8.0g
------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

The output from the command `describe` provides us with such useful information as

1. Number of observations
2. Listing of the variables in the dataset
3. *Storage type* assigned to each variable
4. *Display format* for each variable (i.e. how the variable is displayed – number of spaces allocated in the display of the actual underlying data)
5. A "Note", which in this case reminds the user that we are working with a dataset that has changes that have not been saved

We will comment briefly about the *storage type*:

The variable *storage type* indicates the data type assigned to a variable. Stata can store variables in *numeric* (numbers) and *string* (characters including numbers) formats. From the results above, `describe` tells us that "name" and "from" are *strings* that are up to 8 and 12 characters long (`str8` and `str12`), respectively. *Numeric* variables (such as "sex", "age" and "kids") can be one of several data types: bytes, integers, long, float, and double. Each of the 5 data types has a different level of precision. In our case, all of the *numeric* variables above are bytes, since they are small and don't need more than the smallest level of precision.

b. Let us actually look at the data by typing

```
list
```

Your output should appear in the **Stata Results Window** as follows:

```
. list

        name       sex       age          from      kids
  1.    Winnie       2        30       Nairobi         0
  2.     Amare       1        33         Addis         2
  3.   Leopold       1        28         Dakar         1
  4.   Kristin       2        40     Cleveland         4
  5.   Christy       2        50         Lagos         3
  6.  Jean Luc       1        35  Antananarivo         0
```

(Note: You can also look directly at the data by typing `browse` and scrolling around the **Data Browser**. This is a very convenient way to view your data. Nonetheless, because you may want to print out some or all of your data later, we will proceed with a discussion of the `list` command.)

Typically, you will be working with a dataset that has many observation (much more than the 6 that we have) and will be interested in listing only some of them. Similarly, you may want to list only some of the variables, not all of them. Or you may want to list your data by different groups. To do this, we can use some of the variations for `list`, such as

```
list name age

list name kids sex if age > 30

list name if sex == 2

list name age kids in 1/3

list name age kids in 3/6 if sex==1

list from name if kids<= 2 &sex~=2

list if name==”Winnie” | name == “Jean Luc”

list if sex==2 | (name==”Jean Luc” & kids<1)

sort sex
by sex: list
```

In these commands, we modified our `list` command with some *options:* `if`, `in`, and `by`. These *options* can be used with many Stata commands, so let's briefly review them:

i. `if` allows us to restrict the sample to those cases that satisfy the expression that follows. For example, in the second command above, we restricted our listing to those people who are older than 30 years of age. In the sixth command, we restricted our listing to those who have at the most 2 children, and who are not female. When we want to specify a *string* variable, we must put the characters in quotes. When we use "&" (and) and "|" (or)

together, we have to be careful to appropriately put our specifications in parentheses in order to make sure that we are clear about our restriction.

There are a couple of notes to make here on syntax in Stata

1. Equal signs: Notice that there two "=" (==) appear in the `if` statements above. Stata distinguishes between assigning values and evaluating expressions by requiring that the former have one equal sign (=) and the latter have two (==). The rule of thumb is that whenever you use `if`, always use "==".
2. Spaces: Spaces within expressions do not matter
3. Other logical operators used in evaluating expressions:

Logical operators used in Stata
~ not
= = equal
~ = not equal
! = not equal
> greater than
> = greater than or equal
< less than
< = less than or equal
& and
| or

ii. `in` allows us to restrict the sample based on the number of observations in the dataset. In the fourth command above, we restricted our listing to the first, second and third observations in the dataset. In the fifth command, we restricted our listing to the third, fourth, fifth and sixth (as well as to only males).

iii. `by` allows us to divide the data into groups based on the different values for the `by` variable that we specify. In order to use `by`, we must first `sort` the data by the variable (or set of variables) that we use in our `by` command. Unlike other *options* in Stata, `by` precedes the command.

c. Label the datasets and variables and view them again.

Datasets can contain labels on the data, the variables, and the values. Labels are ***extremely*** useful for users of Stata datasets. They help explain and document the data and the variables. They are a very important component of a well-documented and user-friendly dataset. The data we are using, however, have no labels…… Let's assign some by entering the following five commands:

```
label variable name "Name of participant"
label variable from "Where the participant is from"
label variable kids "Number of participant's kids"
label data "TRAINING PARTICIPANTS"
describe
```

The following output should appear in the **Stata Results Window**:

```
. label variable name "Name of participant"

. label variable from "Where the participant is from"

. label variable kids "Number of participant's kids"

. label data "TRAINING PARTICIPANTS"

. describe

Contains data
  obs:             6                          TRAINING PARTICIPANTS
 vars:             5
 size:           162 (100.0% of memory free)
-------------------------------------------------------------------------------
   1. name      str8   %9s                    Name of participant
   2. sex       byte   %8.0g
   3. age       byte   %8.0g
   4. from      str12  %12s                   Where the participant
   is from
   5. kids      byte   %8.0g                  Number of
   participant's kids
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
```

You will notice that the variable labels also appear in the **Variables Window** next to the variable names.

We can also attach labels to values of variables that are numeric and categorical. For example, in this dataset, we know that a 1 for "sex" means a person who is a man, and a 2 means that she is a woman. We can build this information into our dataset using a *value label*. To do this, we first define a *value label* and then attach this label to the variable as follows:

```
label define sexlabel 1 “Man” 2 “Woman”
label values sex sexlabel
describe
list sex
```

The following output should appear in the **Stata Results Window**:

```
. label define sexlabel 1 "Man" 2 "Woman"

. label values sex sexlabel

. describe

Contains data
  obs:             6                          TRAINING PARTICIPANTS
 vars:             5
 size:           162 (100.0% of memory free)
-------------------------------------------------------------------------
   1. name      str8   %9s                    Name of participant
   2. sex       byte   %8.0g       sexlabel
   3. age       byte   %8.0g
   4. from      str12  %12s                   Where the participant
   is from
   5. kids      byte   %8.0g                  Number of
   participant's kids
-------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. list sex

         sex
  1.   Woman
  2.     Man
  3.     Man
  4.   Woman
  5.   Woman
  6.     Man
```

By attaching the *value label* to “sex”, we have <u>not</u> replaced the actual values of the variable. They are still 1 and 2. To assure yourself of this, type the following command:

```
list sex, nolabel
```

`nolabel` is a special option that exists for the list command (and certain others). It is specified after the comma sign.

d. Summary statistics give us another means of “viewing” our data.

In the command line, type:

```
summarize
```

The following output should appear in the **Stata Results Window**:

```
. summarize

Variable |     Obs        Mean   Std. Dev.       Min        Max
---------+--------------------------------------------------------
    name |       0
     sex |       6         1.5   .5477226          1          2
     age |       6          36   8.024961         28         50
    from |       0
    kids |       6    1.666667   1.632993          0          4
```

You will notice that the "name" and "from" variables have 0 observations. This is because they are *strings* – calculating a mean is undefined for these types of variables. This is why knowing the *storage type* is so important. The 0's for "name" and "from" follow because they are *strings*, not because they have missing values.

We can get more information on our variable by adding the *option* `detail`, as well by using `if`, `by`, and `in`. To get an idea of this, type the following commands and look at the results they produce:

```
sum kids, detail

sum age if kids<4, detail

sort sex
by sex: summ
```

`summarize` (which can also by typed as `sum`[1] rather than the entire `summarize`) is not very useful for categorical variables. So now we turn to `tabulate`.

e. To produce a frequency distribution for a variable, or a cross-tabulation for frequencies for two variables, we use the `tabulate` command (abbreviated `tab`). `tabulate` is particularly useful for categorical variable and for string variables.

Type the following commands and take a look at the results that they produce[2]:

```
tab sex

tab kids if age > 30

tab kids sex
```

[1] To be more specific. The following all represent the `summarize` command: `sum, summ, summa, summar, summari, summariz` and summarize. In the Stata manuals, the entry for `summarize` will have the following summarize. The underline represents the minimum number of characters that you need to type to for Stata to recognize the command.

[2] In the sixth command, because there are no other variables that start with `na`, Stata recognizes `na` to represent `name`.

```
tab1 kids sex age

tab2 kids sex age

tab na
```

There are many options that exist for the tabulate command. For example, we might want to know the percent of women and men with 0, 1, 2, 3 and 4 kids in the dataset. We do this by typing the following:

```
tab sex kids, row
```

and if we only want the percentages and not the frequencies as well, we can type:

```
tab sex kids, row nofreq
```

Now suppose we want to know the average (mean) number of kids by sex of participants, then we can type:

```
tab sex, sum(kids)
```

We could also find the average age by sex of participants, excluding Leopold:

```
tab sex if name ~= “Leopold”, sum(age)
```

Step 4: Create a new Stata dataset

Thus far we haven't actually created a Stata dataset. We have loaded data into Stata's memory, but haven't saved it as a Stata dataset. Let's do so now by typing

```
save exercise
```

Since we did not specify a suffix, Stata automatically assigns a ".dta". Also, since we did not specify where to put the new dataset, Stata will automatically save these data in our present working directory (`c:\aerc\stata_review\temp`). We can see this by typing[3]

```
dir
```

or

```
dir c:\aerc\stata_review\temp
```

If we wanted to specify the saving location, we could have used the following command

```
save c:\aerc\stata_review\temp\exercise
```

Now suppose that we want to add one more label to our data and then re-save it. We have to explicitly tell Stata to `replace` the old dataset with the revised (new) one.

```
label var age "Age of participant"
save exercise, replace
```

If we had not included the `replace`, then the following error statement would have appeared in the **Stata Results Window** (after also typing `lookup 602`):

```
. save exercise
file exercise.dta already exists
r(602);

. lookup 602

[R]     error messages . .  . . . . . . . . . . . . . . . Return code 602
        file ___________ already exists;
        You attempted to write over a file that already exists.  Stata
        will never let you do this accidentally.  If you really intend
        to overwrite the previous file, reissue the last command
        specifying the replace option.
```

A very important thing to consider when working with data is how we manage our files. As I'm sure you know very well, we will quickly create many files. With good file management, you can avoid frustration in trying to understand what you have already created! More on this later.

[3] Stata recognizes DOS commands.

Other ways of reading data into Stata:

Here is a list of some of the possible ways of reading data into Stata include…

- Loading Stata datasets (we'll see how this is done in the next section)
- Using insheet
- Using infile
- Using infix
- Cutting from Excel and pasting into the **Stata Data Editor**
- Using data conversion programs (e.g. StatTransfer and DBMS/COPY) to convert other data formats to Stata datasets and then load them into Stata.

C. Data Modification & Editing

We often want to manipulate raw household data for purposes of analysis. In this section we illustrate how to create new variables and how to modify them. The three most common commands used to generate new variables are

1. `generate`
2. `egen` ("extensions to generate")
3. `collapse`

1. `generate` and `replace`:

The `generate` command has the following basic syntax

```
[by varlist:] generate newvar = exp [if exp] [in range]
```

where *`varlist`* indicates a list of variables, and *`exp`* indicates an expression. Naming convention in Stata 6.0 (and earlier versions) requires that variable names (`newvar`) have no more than 8 characters. Note that the `g` in the command `generate`, is underlined. This indicates that you can type only `g`, and Stata will know that you mean `generate`. The syntax for `generate` should become clearer after we work through some examples. Let's start by loading some data:

```
use c:\aerc\stata_review\data\memsex.dta
```

In this dataset, we have information on the number of men, women and kids in the 4,800 households in our sample and their area of residence. Suppose that we want to calculate a variable that indicates the squared number of household members. Enter the following command

```
generate memb2 = memb*memb
```

Of course, we could have also squared the value. Try typing

```
generate memb2 = memb^2
```

You should have received the following error statement:

```
. generate memb2 = memb^2
memb2 already defined
r(110);
```

This occurs because Stata prevents us from inadvertently writing over variables that already exist. We *can* overwrite existing variables, but Stata just makes us be sure that we want to, and thus the rationale for the `replace` command. Let us now re-`generate memb2` by using the `replace` command:

```
replace memb2 = memb^2
```

Now Stata accepts it and provides the following output:

```
. replace memb2 = memb^2
(0 real changes made)
```

which is what we would expect since nothing has really changed to `memb2`. Note that the basic syntax for `replace` is the same as for `generate` (except that `replace` cannot be abbreviated).

Now suppose that we want to calculate the number of household members from the information that we are given on the number of men and women in the household. Then we can use the following command:

```
generate members = men + women
```

This new variable `members` should be identical to the variable `memb`. There are a couple of ways to check that this is so. One way is to use the `assert` command:

```
assert members == memb
```

Since Stata doesn't give a statement telling us that the expression (`members == memb`) is false, we know it's true and that they are identical. A second way to go about checking the equivalence of `members` and `memb` is to create a categorical variable that takes on a value of 1 if they are not the same, and 0 if they are. Type the following commands:

```
generate error = 0
replace  error = 1 if members ~= memb
tab error
```

Note that replace is used here to make selective changes to `error`, by using the `if` statement, rather than sweeping changes as we made to `memb2` above. Now given the following output:

```
. generate error = 0

. replace error = 1 if members ~= memb
(0 real changes made)

. tab error

      error |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |       4800      100.00      100.00
------------+-----------------------------------
      Total |       4800      100.00
```

we can again verify that the two are identical. We could also have "generated" the `error` variable using the following command:

```
generate error = (members ~= memb)
```

Here Stata creates a variable, `error`, giving it a value of 1 if the expression in the parentheses is true, and 0 otherwise.

Let's now calculate the number of kids in the household (and call this new variable `children`) as we did with all members, and verify that this is identical to the variable in the dataset called `kids`. Enter the following commands:

```
gen children = boys + girls
assert children == kids
```

Ooops! The following output suggests that something is wrong:

```
. gen children = boys + girls
(2922 missing values generated)

. assert children == kids
2922 contradictions out of 4800
assertion is false
r(9);
```

In more than half of the cases, the variable `children` is not identical to `kids`. Let's see what is wrong by listing the relevant variables for the first 10 observations if they aren't the same:

```
list boys girls kids children in 1/10 if children~=kids
```

We see from the following output that there are missing values for boys and girls:

```
. list boys girls kids children in 1/10 if children~=kids

        boys     girls       kids   children
  2.       .         .          0          .
  5.       1         .          1          .
  6.       .         2          2          .
```

Based on this information, we can see that `boys` and `girls` are set to missing when there are no children in the household. (Verify this by tabulating these variables to see if there are any 0 values). So what happened is that the Stata set `children` to missing when one of the arguments was missing. *Be careful when using the* `generate` *command to sum up other variables if there are any missing values.*

We could recode these missing values to 0 before creating the variable `children` by using the following command,

```
recode boys  . = 0
recode girls . = 0
```

Or we could use `egen`, which stands for "extensions to `generate`"…

2. *`egen` and `replace`:*

As it's name suggests, `egen` is simply a set of extensions to `generate`, with the extension being that the new variable is defined as a function of "stuff" that you indicate ("stuff" depends on the particular function that you choose). If you can't figure out how to do something with `generate`, chances are you will be able to do it with `egen`. The basic syntax for `egen` is:

```
egen newvar = fcn(stuff) [if exp] [in range], [by(varlist)]
```

One of the functions (*`fcn`*) for `egen` is `rsum`, which creates the row sum of the variables indicated, treating missing values as zero. This is exactly what we need to calculate our `children` variable. Let's do so and verify that it's the same as `kids`:

```
drop children
egen children = rsum(boys girls)
assert children == kids
```

Now we get the results that we want…

```
. drop children

. egen children = rsum(boys girls)

. assert children == kids

.
```

`egen` has many other functions that are useful such as row means, medians, minima, maxima, standard deviations, and `sum`. For now, we'll only address one of these: `sum`. We do this in part because `generate` has a similarly named function which behaves differently.

Let's start by clearing this dataset, loading another (a household-level file), and creating a vector of ones:

```
clear
use c:\aerc\stata_review\data\memage.dta
gen ones = 1
```

Let's now see what the sum function produces when we use generate and egen:

```
sort hid
quietly by hid: gen gsum = sum(ones)
egen egsum = sum(ones), by(hid)
list hid gsum egsum in 1/16
```

There are a couple of things to note here. First, before you can use the by option in generate or egen, you must first sort the data by the variable of interest (here our household id). Second, we use the quietly option for generate because otherwise, generate provides information for each value of the by variable (try the same command, but without the quietly, to see what happens). The following output gives us an idea of how the two sum functions differ:

```
. gen ones = 1

. sort hid

. quietly by hid: gen gsum = sum(ones)

. egen egsum = sum(ones), by(hid)

. list hid gsum egsum in 1/16

            hid       gsum      egsum
  1.        101          1          6
  2.        101          2          6
  3.        101          3          6
  4.        101          4          6
  5.        101          5          6
  6.        101          6          6

  7.        103          1          3
  8.        103          2          3
  9.        103          3          3

 10.        104          1          7
 11.        104          2          7
 12.        104          3          7
 13.        104          4          7
 14.        104          5          7
 15.        104          6          7
 16.        104          7          7
```

Spaces were added between the households for purposes of emphasis. The combination of the by option with the sum function results in a running sum within the household when using generate, and a total sum within the household when using egen. The combination of these two commands is useful in creating distribution functions, and for creating other variables when we want to know the ranking of a record within a bygroup.

3. `collapse`*:*

A third very common means of creating new variables is the `collapse` command. This command allows you to collapse data to a more aggregate level of observation. For example, you may have an individual-level dataset (such as an excerpt from a household roster that we presently have loaded in Stata) and you want to create a household-level file with some information about the people in the household. This new file could have information such as: number of children in the household, average age of women in the household, sex of the household head, etc.. The `collapse` command can be used to create such a dataset.

The basic syntax of this command is

```
collapse (fnc1) varlist1 (fnc2) varlist2… , by(varlist)
```

where

- *`fnc1`* is a function generating a summary statistic (mean, max, min, sum)
- *`varlist1`* is a list of variables to which *`fnc1`* is applied
- *`fnc2`* is a function generating a summary statistic (mean, max, min, sum)
- *`varlist2`* is a list of variables to which *`fnc2`* is applied

For example, suppose that we want to create from the data presently loaded in Stata, a regional-level (defined as urban/rural areas in each region) dataset with the following information:

1. Share of women in the region
2. Age of the oldest women in the sample in the region
3. Age of the oldest man in the sample in the region
4. Number of children of age 1, 2, … , 15 in the sample in the region

This can be done with the following commands:

```
replace sex = sex – 1
gen oldwmn  = agey if sex == 1
gen oldmen  = agey if sex == 0
collapse (mean) sex (max) oldwmn oldmen (sum) age1-age15, by(region urban)
list region-age1
```

One thing to note here is the use of `age1-age15` and `region-age1`. The former tells Stata to include `age1` and `age15` and all of the variables in between as they are presently ordered in the loaded dataset. This save on typing!! But be sure that the data are ordered as you want them before you use this feature. You can check the order by viewing the variables in the **Variables Window**, or by using the `summarize` or `describe` commands.

Now for our output (you can look at the number of children age 2-15 on your own) …

```
. replace sex = sex - 1
(24068 real changes made)

. gen oldwmn  = agey if sex == 1
(11610 missing values generated)

. gen oldmen  = agey if sex == 0
(12458 missing values generated)

. collapse (mean) sex (max) oldwmn oldmen (sum) age1-age15, by(region urban)

. list region-age1

        region       urban         sex     oldwmn     oldmen       age1
  1.         1       Rural   .50566036         92         85         83
  2.         1       Urban   .53914589         84         75         13
  3.         2       Rural   .51951611         93         86         96
  4.         2       Urban    .5210191         90         84         10
  5.         3       Rural   .52530295         96         89         76
  6.         3       Urban   .51698112         79         81          3
  7.         4       Rural   .51153463         89         92         40
  8.         4       Urban   .51075876         88         84         14
  9.         5       Rural   .50130892         92         80         19
 10.         6       Rural   .50029641         88         90         27
 11.         6       Urban   .52235472         86         83         25
 12.         7       Rural   .52340233         91         90         98
 13.         7       Urban   .53948718         88         89         12
```

If we look at the first observation, we see that for rural areas in region 1, about 51% of the population is female, the oldest woman is 92, the oldest man is 85, and there are 83 one-year-olds in the sample. Note also that there are only 13 observations because there are no urban areas in region 5.

Exercise: Using the VNLSS93 household roster located in

`c:\aerc\stata_review\data\sect01a`

create a household-level dataset with the following information

1. Number of adult household members (15+ years)
2. Average age of household members
3. Gender of the household head

D. Merging Datasets

There are two commands in Stata that allow you to combine 2 datasets: `merge` and `append`. Since most household datasets involve many individual data files, we often use `merge` and `append` to combine files into one new dataset.

1. merge:

When we use `merge`, we generally link observations and add variables between two datasets. In order to do so, it is essential to have a variable (or several variables) that can be used to link observations together from the two datasets. Further, before we proceed with the merging, we want to verify that the identification variable(s) which we use in the `merge` command is unique in at least one of the datasets.

Let's work through a couple of examples to explain the process.

Example: One-to-One Merge: In this example, we will merge two household-level datasets using the household id as the linking variable.

(i) Done correctly

Let's start by (a) loading the VNLSS household roster; (b) keeping the household id, the individual id, relation to the head and gender; (c) keeping only the observation for the household head; (d) sorting by household id; (e) saving the dataset as a temporary file; and (f) listing the first 10 observations:

```
use c:\aerc\stata_review\data\sect01a.dta
keep hid-rel
keep if rel == 1
sort hid
save c:\aerc\stata_review\temp\temp, replace
list in 1/10
```

Our output is as follows:

```
. use c:\aerc\stata_review\data\sect01a.dta

. keep hid-rel

. keep if rel==1
(19268 observations deleted)

. sort hid

. save c:\aerc\stata_review\temp\temp, replace
file c:\aerc\stata_review\temp\temp.dta saved

. list in 1/10

           hid         pid         sex         rel
  1.       101           1           1           1
  2.       103           1           1           1
  3.       104           1           1           1
  4.       105           1           1           1
  5.       106           1           1           1
  6.       108           1           1           1
  7.       109           1           1           1
  8.       110           1           1           1
  9.       111           1           1           1
 10.       113           1           1           1
```

We will refer to this as our ***using*** dataset.

Now (a) load the household expenditure dataset; (b) keep the household id and the per capita household expenditure aggregate; (c) verify that the household id is unique; (d) sort by the household variable; (e) save the dataset at a temporary file for the next example; and (f) list the first 10 observations:

```
use c:\aerc\stata_review\data\hhexpend.dta
keep hid pcexpend
* verify that hid is unique
 sort hid
 quietly by hid: gen error = _n
 quietly by hid: assert error == 1
 drop error
save c:\aerc\stata_review\temp\temp2, replace
list in 1/10
```

Our output is as follows:

```
. use c:\aerc\stata_review\data\hhexpend.dta

. keep hid pcexpend

. * verify that hid is unique
. sort hid

. quietly by hid: gen error = _n

. quietly by hid: assert error == 1

. drop error

. save c:\aerc\stata_review\temp\temp2, replace
file c:\aerc\stata_review\temp\temp2.dta saved

. list in 1/10

            hid    pcexpend
  1.        101   252.22887
  2.        103   593.39282
  3.        104    317.8396
  4.        105    480.8645
  5.        106   822.06958
  6.        108   714.79736
  7.        109   180.93414
  8.        110   333.22534
  9.        111   499.15894
 10.        113   385.53247
```

We see that the "hid" uniquely identifies the households because there is no contradicting statement following the `assert` command. We refer to this dataset as our ***master*** dataset.

Let's now merge the ***using*** dataset into our ***master*** dataset using "hid" as our linking variable:

```
merge hid using c:\aerc\stata_review\temp\temp
```

When the `merge` command is used, Stata automatically creates a new variable called "_merge". This is a categorical variable that takes on the following three values:

1 if observations from the master dataset that did not match observations from the using dataset (i.e. in ***master*** not ***using***)

2 if observations from the using dataset that did not match observations from the master dataset (i.e. in ***using*** not ***master***)

3 if observations from both datasets that matched (i.e. in both ***master*** and ***using***).

This is *extremely* useful in evaluating the merge and making sure that we did exactly as we intended. Merging is often a source of errors and we should always carefully check that we are doing what we want. Let's do so here…

```
tab _merge
list in 1/10
```

As we can see from the output below, our merge worked as intended.

```
. merge hid using c:\aerc\stata_review\temp\temp

. tab _merge

    _merge |      Freq.     Percent        Cum.
-----------+-----------------------------------
         3 |       4800      100.00      100.00
-----------+-----------------------------------
     Total |       4800      100.00

. list in 1/10

           hid    pcexpend         pid         sex         rel    _merge
  1.       101   252.22887           1           1           1         3
  2.       103   593.39282           1           1           1         3
  3.       104    317.8396           1           1           1         3
  4.       105    480.8645           1           1           1         3
  5.       106   822.06958           1           1           1         3
  6.       108   714.79736           1           1           1         3
  7.       109   180.93414           1           1           1         3
  8.       110   333.22534           1           1           1         3
  9.       111   499.15894           1           1           1         3
 10.       113   385.53247           1           1           1         3
```

(ii) Correct outcome, <u>but</u> only by chance

Suppose that we forget to specify the linking variable (sometimes we will want to merge datasets without linking observations). What will happen? Let's see…

```
clear
use c:\aerc\stata_review\temp\temp2.dta
merge using c:\aerc\stata_review\temp\temp
tab _m
list in 1/10
```

produces the following output:

```
. clear

. use c:\aerc\stata_review\temp\temp2

. merge using c:\aerc\stata_review\temp\temp

. tab _m

    _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
         3 |       4800      100.00      100.00
------------+-----------------------------------
     Total |       4800      100.00

. list in 1/10

            hid    pcexpend         pid         sex         rel    _merge
  1.        101   252.22887           1           1           1         3
  2.        103   593.39282           1           1           1         3
  3.        104    317.8396           1           1           1         3
  4.        105    480.8645           1           1           1         3
  5.        106   822.06958           1           1           1         3
  6.        108   714.79736           1           1           1         3
  7.        109   180.93414           1           1           1         3
  8.        110   333.22534           1           1           1         3
  9.        111   499.15894           1           1           1         3
 10.        113   385.53247           1           1           1         3
```

We have the same results! Unfortunately, we got them just by chance because the two datasets were sorted by the same variable and there was a perfect match in the observations. This is less likely to occur when we do a *many-to-one* merge as we will see later.

(iii) Incorrect outcome

Let's see how forgetting to include the linking variable can get us into trouble. First we'll resort the ***master*** dataset by another variable, and then we'll repeat the steps of part *ii*.

```
clear
use c:\aerc\stata_review\temp\temp2.dta
sort pcexpend
merge using c:\aerc\stata_review\temp\temp
tab _m
sort hid
list in 1/10
```

Now let's examine the output…

```
. clear

. use c:\aerc\stata_review\temp\temp2

. sort pcexpend

. merge using c:\aerc\stata_review\temp\temp

. tab _m

     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          3 |       4800      100.00      100.00
------------+-----------------------------------
      Total |       4800      100.00

. sort hid

. list in 1/10

            hid    pcexpend         pid         sex         rel    _merge
  1.        101   252.22887           1           1           1         3
  2.        103   593.39282           1           1           1         3
  3.        104    317.8396           1           1           1         3
  4.        105    480.8645           1           2           1         3
  5.        106   822.06958           1           1           1         3
  6.        108   714.79736           1           1           1         3
  7.        109   180.93414           1           1           1         3
  8.        110   333.22534           1           1           1         3
  9.        111   499.15894           1           1           1         3
 10.        113   385.53247           1           1           1         3
```

OK, so the "_merge" variables all take on the value of 3, but the "pid", "sex" and "rel" variables are incorrect. The former follows because there are 4,800 records in each dataset, and as far as Stata can tell, it's a perfect match (because no link variable is defined). The latter follows because the household information (*"pid", "sex", "rel"*) were all associated with the household id (*"hid"*) in the order that it was found in the ***using*** dataset, not in the order of *"pcexpend"* that we had in the ***master*** dataset.

If we had included the linking variable, Stata would not have let us make this merge. Let's try it to see what happens…

```
clear
use c:\aerc\stata_review\temp\temp2.dta
sort pcexpend
merge hid using c:\aerc\stata_review\temp\temp
```

As we can see, Stata catches the error and refuses to proceed…

```
. clear

. use c:\aerc\stata_review\temp\temp2

. sort pcexpend

. merge hid using c:\aerc\stata_review\temp\temp
master data not sorted
r(5);
```

It's sorted, just not by the right variable (*hid*)!

Example: One-to-Many Merge: In this example, we will merge a household-level dataset to an individual level dataset using the household id as the linking variable.

Again we will use the VNLSS household roster, except now we will keep all of the observations:

```
clear
use c:\aerc\stata_review\data\sect01a.dta
keep hid-rel
sort hid
save c:\aerc\stata_review\temp\temp, replace
list in 1/10
```

This provides the following output…

```
. clear

. use c:\aerc\stata_review\data\sect01a.dta

. keep hid-rel

. sort hid

. save c:\aerc\stata_review\temp\temp, replace
file c:\aerc\stata_review\temp\temp.dta saved

. list in 1/10

            hid         pid         sex         rel
  1.        101           1           1           1
  2.        101           2           2           2
  3.        101           3           1           3
  4.        101           4           1           3
  5.        101           5           2           3
  6.        101           6           2           3
  7.        103           1           1           1
  8.        103           2           2           3
  9.        103           3           1           8
 10.        104           1           1           1
```

We will now refer to the individual-level data as the ***master*** dataset, and the household-level data as the ***using*** dataset. Merge the former with the latter by issuing the following commands:

```
merge hid using c:\aerc\stata_review\temp\temp2
tab _m
list in 1/10
```

Again we have exactly what we want…

```
. merge hid using c:\aerc\stata_review\temp\temp2

. tab _m

     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          3 |      24068      100.00      100.00
------------+-----------------------------------
      Total |      24068      100.00

. list in 1/10

            hid         pid         sex         rel    pcexpend    _merge
  1.        101           1           1           1   252.22887         3
  2.        101           2           2           2   252.22887         3
  3.        101           3           1           3   252.22887         3
  4.        101           4           1           3   252.22887         3
  5.        101           5           2           3   252.22887         3
  6.        101           6           2           3   252.22887         3
  7.        103           1           1           1   593.39282         3
  8.        103           2           2           3   593.39282         3
  9.        103           3           1           8   593.39282         3
 10.        104           1           1           1    317.8396         3
```

…to each member of household 101, we appended the level of per capita household expenditures.

Let's try one more experiment to get an understanding of what information the *"_merge"* variable provides for us. We will drop all of the observations for women in the household roster and then merge this truncated sample with the expenditure data.

Before doing so, what do we expect to happen? Provided that there are some households that consist of only male members, then we would expect to have observations in the ***using*** dataset that are not in the ***master*** dataset. Once we merge, we expect the *"_merge"* variable to take on a value of 2 for these households, and 3 for all of the others. Let's try it…

```
clear
use c:\aerc\stata_review\temp\temp
keep if sex==1
sort hid
merge hid using c:\aerc\stata_review\temp\temp2
tab _m
sort _m hid
list in 1/10
```

Indeed we find exactly what we expected…

```
. clear

. use c:\aerc\stata_review\temp\temp

. keep if sex==1
(12458 observations deleted)

. sort hid

. merge hid using c:\aerc\stata_review\temp\temp2

. tab _m

     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          2 |        240        2.03        2.03
          3 |      11610       97.97      100.00
------------+-----------------------------------
      Total |      11850      100.00

. sort _m hid

. list in 1/10

            hid         pid         sex         rel    pcexpend    _merge
  1.        310           .           .           .   596.30469         2
  2.       1302           .           .           .   766.91919         2
  3.       1516           .           .           .   830.67261         2
  4.       1711           .           .           .   781.28491         2
  5.       1808           .           .           .    1116.012         2
  6.       1906           .           .           .   1792.9006         2
  7.       2001           .           .           .   1037.5427         2
  8.       2006           .           .           .   549.62378         2
  9.       2008           .           .           .   1791.9211         2
 10.       2216           .           .           .   614.41724         2
```

…the *"_merge"* variable takes on values of 2 for the households not in the ***master*** dataset, but in the ***using*** dataset. You can verify on your own that the households listed above do not have any female members.

2. Append:

The append command is straightforward and involves simply adding rows (and sometimes columns if the two datasets have different variables) to the data loaded in memory. To illustrate how append works, let's create three datasets from the VNLSS household recode and append them.

In the first dataset we will keep only records for the household head. In the second, we will only keep records for the spouse of the household head. In the third, we will keep only the records for the oldest son of the household head. Obviously, not all households have members who are spouses or sons. We shall see how this affects the append.

```
use c:\aerc\stata_review\data\sect01a.dta
keep hid pid rel agey agem
save c:\aerc\stata_review\temp\temp, replace
```

First dataset:

```
use c:\aerc\stata_review\temp\temp
keep if rel==1
drop agey agem
gen head = 1
save c:\aerc\stata_review\temp\temp1, replace
```

Second dataset:

```
clear
use c:\aerc\stata_review\temp\temp
keep if rel==2
drop agey agem
gen spouse = 1
save c:\aerc\stata_review\temp\temp2, replace
```

Third dataset:

```
clear
use c:\aerc\stata_review\temp\temp
keep if rel==3
gsort hid –agey –agem
qui by hid: gen oldest = _n
drop agey agem
keep if oldest == 1
save c:\aerc\stata_review\temp\temp3, replace
```

A couple of comments need to be made briefly regarding the construction of the third dataset. First, the command `gsort` is very similar to `sort`, except that the former allows sorting by descending and ascending order, whereas the latter only permit ascending sorts. The way to read the command

```
gsort hid –agey –agem
```

is, "sort by *"hid"* from smallest to largest, then within *"hid"* sort *"agey"* from largest to smallest, and then within *"hid"* and *"agey"* sort *"agem"* from largest to smallest".

Second, Stata attaches internal numbers to each observation. One of these numbers is the variable number (*_n*), and the other is the total number of observations (*_N*). It is possible to modify these internal numbers to reflect the position within a *bygroup*. In this case, after we have sorted the data, we define a variable *"oldest"* that takes on the ranking of the observations within each household. This is exactly *_n*, as can be seen from a listing of the first 10 observations before the other observations were dropped…

```
          hid         pid         rel        agey        agem     oldest
 1.       101           3           3           8           2          1
 2.       101           4           3           5           9          2
 3.       101           5           3           3           3          3
 4.       101           6           3           0           7          4
 5.       103           2           3          20           .          1
 6.       104           3           3           2           5          1
 7.       104           4           3           1           6          2
 8.       104           5           3           0           4          3
 9.       105           3           3          11           .          1
10.       105           4           3          10           .          2
```

Let's proceed by appending the first and second datasets, and then the third:

```
clear
use c:\aerc\stata_review\temp\temp1
append using c:\aerc\stata_review\temp\temp2
append using c:\aerc\stata_review\temp\temp3
sort hid
```

and after viewing the data, the following household have results that are informative, so let's list them,

```
list hid rel-oldest if hid==101 | hid==103 | hid==113, noobs
```

```
. clear

. use c:\aerc\stata_review\temp\temp1

. append using c:\aerc\stata_review\temp\temp2

. append using c:\aerc\stata_review\temp\temp3

. sort hid rel

. list hid rel-oldest if hid==101 | hid==103 | hid==113, noobs

      hid         rel       head     spouse     oldest
      101           1          1          .          .
      101           2          .          1          .
      101           3          .          .          1
      103           1          1          .          .
      103           3          .          .          1
      113           1          1          .          .
      113           2          .          1          .
```

From the output above, we can see that household 101 has a head, a spouse and a son. Thus this household had an observation in each dataset, and consequently has three observations in the new dataset. Household 103 (113) doesn't have a spouse (son), and as such only has observations from the first and third (second) datasets. Note that Stata sets to missing the variables that are not common to the two datasets being appended. For example *"head"* is set to missing for all observations appended from the second and third dataset.

Exercise:

Using

```
c:\aerc\stata_review\data\sect01a.dta
```

create a variable for the most senior male in the household, where senior male is defined as

1) HH head if HH head is male;
2) spouse of HH head if HH head is female;
3) oldest male if neither 1) or 2).