Intro to Stata 3

1. **The Objectives of Stata Tutorial 3**

   - To learn *value labels* and *variable labels*
   - To learn how to *drop* and *rename* variables
   - To learn the commands *order* and *sort*
   - To learn how to *save* newly-created data
   - To learn how to *merge* a data set with another data set

2. **Preparing for Tutorial 3**

   Let's create a directory called "tutorial3" under "C:\temp\stata-tutorial\" (under which you should find the directories "tutorial1" and "tutorial2" unless you erased them). We use new data sets t89hh1.dta and t89inc.dta to demonstrate Stata commands in this tutorial. Copy both t89hh1.dta and t89inc.dta into the new directory "tutorial3" and set the directory as the working directory (If you are confused, you must go back to Tutorial 1).

   These real data were collected in China in 1989 as a part of a survey called the China Health and Nutrition Survey (CHNS). The CHNS is a large-scale socioeconomic survey that yields observational data on many aspects of Chinese people including health, education, and economic activities. t89hh1.dta includes demographic data for 15,924 individuals in 3,795 households. t89inc.dta contains household income (as well as other information) for the 3,795 households. Please load t89hh1.dta in Stata memory by the command *use*. Click on the "Variables" window. You see 22 variables in the data set as well as the definition of each variable. If you cannot see the definitions, expand the window horizontally by dragging the left end of the window toward the *Results* window first and then dragging the right end of the Label column toward the right. You can also browse the data to see how they look. *Clear* the data set, and load t89inc.dta. Repeat the same for t89inc.dta to look at what variables are available in the data set and how they look.

   In passing, in case you close the *Review*, *Variables*, *Properties*, or *Command* window for some reason, you can restore the window by selecting an appropriate window under the *Window* menu.

3. ***value labels* and *variable labels***

Clear t89inc.dta, and *use* t89hh1.dta. First, I want you to confirm that all variables (including id variables) are stored as *numeric.* To do so, either *describe* (often shortened as *des*) or *browse* the data (*numeric* variables are shown in black or blue fonts in the *Data Browser*).

Let me explain *value labels*, first. *Tabulate* (often shortened as *tab*) *a2* which indicates the sex of the sample individual. Although three individuals did not report their sexes (thus, missing), the vast majority of the respondents did so. In the table Stata returned, 1 represents Male and 2 represents Female. Sometimes, it is more convenient to see Male or Female directly rather than 1 or 2. Here is how to do this. Type the following two commands in the command line.

```
label define SEX 1 "Male" 2 "Female" (press return)
label values a2 SEX (press return)
```

Now the values of 1 and 2 in the variable *a2* have been labelled "Male" and "Female," respectively. "Male" and "Female" are called value labels in Stata language. *Tabulate a2* again. You see the difference. Also, browse the data. Now *a2* stores either Male or Female (or . (dot)) in each row instead of 1 or 2. Notice "Male" and "Female" are shown in blue fonts rather than in black fonts in the *Data Browser*, where blue fonts means that the corresponding variable is stored as *numeric* with *value labels. Describe* the data again and pay attention to *a2. a2* remains to be a *numeric* variable, but the name (SEX) of the *value labels* (Male and Female) is shown in the column under "value label."

Let me explain the two commands we used. The first command defines *value labels* "Male" and "Female," where 1 and 2 are the values each label is attached to. I used SEX as the name of the value labels ("Male" and "Female"). The names of value labels are not necessarily in upper case, but I like using upper-case letters for this purpose as my own rule. The second command attaches the value labels ("Male" and "Female") to the values (1 and 2) of the variable *a2.* Use *help* for further details.[1]

Value labels are convenient, but sometimes we want to see the original values to which value labels are attached. To do so, type

```
label list SEX
```

[1]In auto.dta we used for Stata Tutorials 1 and 2, the variable *foreign* is a *numeric* variable with value labels.

in the command line. Be careful that Stata is case-sensitive. If you just type “label list” without any argument, Stata returns all value labels stored in the data set.

Let me move to *variable labels* which is much simpler to explain/use. When you click on the “Variables” window, you see the names and definitions of the variables. These definitions are called *variable labels.* When you create a new variable, it has no *variable label.* Let me demonstrate this. Let’s create a new dummy variable called *adult* which indicates that the individual is 16 years old or older. To do so, type the following two commands in the command line.

```
gen adult=0 if a3<. (press return)
replace adult=1 if a3>=16 & a3<. (press return)
```

Here, the variable a3 shows the age of the sample individual. Now, *describe* the variables or click on the “Variables” window, to see there is no *variable label* for *adult.* To attach a *variable label* to *adult*, type (in one line)

```
label variable adult “adult indicator
where adults are those with aged 16 or older” (press return)
```

in the command line. Confirm the new *variable label* for *adult* by *describ*ing the data or clicking on the “Variables” window.

4. ***rename* and *drop***

Sometimes, you may want to change the name of a variable without altering the contents of the variable. The command *rename* is for this purpose. For example, you see in the data set *t6* which shows household size. To change the name of variable from *t6* to *hhsize*, type

```
rename t6 hhsize
```

in the command line.

As you can easily imagine from the name, the command *drop* drops a list of variables from the currently-used data set. For instance, type

```
drop a6 a7
```

in the command line. Then, you see that the variables *a6* and *a7* no longer exist in the currently-loaded data set. You can type a single variable or a list of variables after *drop.* You can use *drop* not only for dropping variables but for dropping observations. For instance, suppose you want to drop all female individuals in the data set t89hh1.dta.

To do so, type

```
drop if a2==2
```

in the command line.

The command *keep* is the antonym of *drop*. For example, type

```
keep commid hhid line
```

in the command line. Then, you have lost all variables except *commid*, *hhid*, and *line*. Like *drop*, you can type a single variable or a list of variables after *keep*. Further, you can use the command *keep* to keep a subset of observations. For example, suppose you want to keep only individuals who live in the community whose *commid* is equal to 21111. To do so, type

```
keep if commid==21111
```

in the command line. Before proceeding, *clear* the data set, and *use* t89hh1.dta again.

5. ***sort* and *order***

   Open the *Data Browser*. Looking at the first several observations (rows), you suspect that the observations are placed in the order of *hhid* and in the order of *line* within *hhid*. Suppose you want to arrange the observations in the order of *a2* (sex) and in the order of *a3* (age) within *a2*. Then, close the *Data Browser* first, and type

   ```
   sort a2 a3
   ```

   in the command line.
   Reopen the *Data Browser* and you see that the observations are in the order you specified. In case of ties (multiple individuals within the same sex and age), the current order of observations are preserved within groups specified by *sort*. The command *sort* arranges observations in ascending order. If you want to arrange observations in descending order, you must use the command *gsort* (Use *help* for details).

   In Tutorials 1 and 2, we used the prefix *bysort* with Stata commands such as *summarize*, *tabulate*, and *egen*. The prefix *bysort* is literally the same as the command *sort* followed by a *by*-able Stata command with the *by* option. Let me demonstrate what I mean. Suppose you want to know the mean age of the sample individuals separately for males and females. As we learned before, you can *summarize* (often shortened as *su*) *a3* by *a2*:

   ```
   bysort a2: su a3
   ```

   This is the same as the following two commands:

```
sort a2 (press return)
by a2: su a3 (press return)
```

The prefix *by* assumes that the observations are arranged in ascending order within variable(s) you specify after *by*. If this condition is not satisfied, Stata issues an error message. The prefix *bysort* can be used without this requirement. Both prefixes, *bysort* and *by*, can be used with multiple variables. For example, if you are interested in how many percentage of sample individuals is currently in school separately for sex and age, you can issue the following command using the variable *a13* where *a13* is equal to zero if the individual is currently not in school and one if the individual is currently in school.

```
bysort a2 a3: su a13
```

The same result will be obtained if you type the following two commands.

```
sort a2 a3 (press return)
by a2 a3: su a13 (press return)
```

Let me move to the command *order*. The command *sort* arranges the order of observations, while the command *order* changes the order of variables stored in Stata memory. The order of variables are shown in the *Variables* window (top to down) or in the *Data Browser* (left to right). Suppose you want to see birth date next to age in the *Data Browser*, to confirm that age is correctly reported. To do so, type

```
order a3 birth_year birth_month birth_day
```

in the command line.
Then, open the *Data Browser*. Now you see that the first four variables (columns) in the *Data Browser* are the variables you just specified, so it is much easier for you to compare age with birth year.

The command *order* could also be useful for another purpose. Suppose you want to *summarize a2*, *a3*, *a5*, *a6*, *a7*, *a8*, *a9*, *birth_year*, *birth_month*, and *birth_day*. Of course, you could type

```
su a2 a3 a5 a6 a7 a8 a9 birth_year birth_month birth_day
```

in the command line.
Another way is first to *order* variables of your choice and then use one of many shorthand conventions available in Stata. That is, type the following two commands in the command line.

```
order a2 a3 a5 a6 a7 a8 a9 birth_year birth_month birth_day (return)
su a2-birth_day (press return)
```

a2-birth_day in the second command means all variables between *a2* and *birth_day* (inclusive), where the order is shown in the *Variables* window (top to down). This technique is useful when you use the variable list *a2-birth_day* repeatedly. For another shorthand conventions available in Stata, *search varlist* under the *help* menu.

6. ***save* newly-created data**

   Among other things, we learned so far how to create new variables, rename existing variables, and attach value and variable labels to existing variables. If you want to keep new variables, renamed variables, and value and variable labels, you can use the command *save* which saves data in Stata memory to a data file with the extension dta. The syntax is easy.

   save *newfilename*, replace

   where *newfilename* is the name of the newly-created data file and you can choose whatever name you like. The suffix *, (comma) replace* is optional, and as we learned from the previous Tutorials, if a data file with the same name exists in the working directory, the option *replace* allows the newly-created data file to replace the old data file with the same name.

   You may find it convenient to use the following data-management technique. As explained before,

   save *newfilename*, replace

   creates a new data file in the working directory. You can create a data file in a directory of your choice by specifying the location of the directory in the command. For example,

   save C:\temp\stata-tutorial\data_test, replace

   creates a new data file called data_test in C:\temp\stata-tutorial\ instead of in the working directory. When the location of a directory includes spaces as in C:\my temp\stata-tutorial\, you must use double quotes after the command *save*. That is, you need to type

   save "C:\my temp\stata-tutorial\data_test", replace

   You can use this same technique to load data in Stata memory. Now your working directory is C:\temp\stata-tutorial\tutorial3\. If you want to *use* auto.dta in C:\temp\stata-tutorial\tutorial1\, you can type

   use C:\temp\stata-tutorial\tutorial1\auto

   in the command line or in a do file.

7. ***merge* a data set with another data set**[2]

The command *merge* merges two data sets.[3] Let me demonstrate how to merge data sets using t89hh1.dta and t89inc.dta. t89hh1.dta is an individual-level data set, containing individual information (age, sex, marital status, $\cdots$) of all household members in the sample households. t89inc.dta is a household-level data set, containing household information (household income, number of household members who contribute to household income, $\cdots$) of the sample households.

In t89hh1.dta, each row represents an individual, and in t89inc.dta, each row represents a household. Since there is at least one household member for each sample household, at least one row (in many cases, multiple rows) in t89hh1.dta should be merged with one row in t89inc.dta. To merge data sets, one id variable (or a set of id variables) must uniquely identify each observation in at least one of two merging data sets, and both data sets must contain the id variable (or the set of id variables). In Stata language, the id variable is called *match variable* if it serves this role in merging data sets. In our particular case, *hhid* uniquely identifies each observation in t89inc.dta, and *hhid* is included in both t89hh1.dta and t89inc.dta. So, *hhid* serves as the *match variable.*

Let's *merge* t89hh1.dta with t89inc.dta using *hhid* as the *match variable.* To start merging, one of the two merging data sets must be in Stata memory. Let's set t89hh1.dta in Stata memory by the command *use.* Then use the command *merge.*

```
clear (press return)
use t89hh1 (press return)
merge m:1 hhid using t89inc (press return)
```

Now, merged data are in Stata memory. In our particular case demonstrated above, t89inc.dta is called "*using data*," and t89hh1.dta is called "*master data*" in Stata language. In other words, *using data* are data

[2]How to use the command *merge* changed significantly before and after Stata 11. Here in this tutorial, I explain how to use the command *merge* in Stata 11 and 12 (and newer in the future). If you need to use the command *merge* in older versions of Stata (Stata 10 or older), please see the older manuals. For example, in older versions of Stata, the command *merge* requires you in advance to *sort* two merging data sets in ascending order of a *match variable* or *match variables.*

[3]If you have multiple data sets to merge, merge two data sets at one time and repeat *merge* multiple times as necessary.

you specify after *using* in the command *merge*, and *master data* are data in Stata memory when you issue the command *merge.* As you can see, the simplest syntax for *merge* is

merge #:# *matchvarlist* using *usingfilename*

where *matchvarlist* is a list of *match variables* or a single *match variable*,[4] and *usingfilename* is the name of a *using data* file. For #:#, you choose one of the following three alternatives: 1:1 or m:1 or 1:m. In the demonstrated command above, m:1 was used because multiple observations or rows in the *master data* were merged with a single observation or row in the *using data.* You should use 1:1 if a single observation or row in the *master data* is to be matched with a single observation or row in the *using data.* You should use 1:m if a single observation or row in the *master data* is to be matched with multiple observations or rows in the *using data.*[5] [6]

After merging data, Stata automatically creates the variable *_merge.* Let's *tabulate _merge.* Then, you see that 15,924 observations (or rows or individuals) in Stata memory have 3 under the variable *_merge. _merge* could take either 1 or 2 or 3. _merge=1 indicates that the observation comes only from the *master* data. _merge=2 indicates that the observation comes only from the *using* data. _merge=3 indicates that the observation comes from both *master* and *using* data. In our particular case, 15,924 individuals all have 3 under *_merge*, meaning that each observation in both t89hh1.dta and t89inc.dta has its counterpart(s) in the other data set.

When you merge data sets one after another, you must either *drop* or *rename _merge* before you proceed to the next *merge.* This is because Stata does not allow a newly-created variable to have the same name as an existing variable. When you *merge* data sets first time, *_merge* is created. When you try another *merge* after the first *merge*, Stata complains if *_merge* from the first *merge* is still in Stata memory.

One thing you should be careful when merging data sets is that

[4]Sometimes, you need to use multiple *match variables* to uniquely identify observations in at least one data file.

[5]If we happen to use t89inc.dta as the *master data* and t89hh1.dta as the *using data*, we should use 1:m.

[6]m:m is available as an option in Stata, but as the Stata manual clearly mentions, it is a very bad idea to use the m:m option. See the manual for details if you are interested.

if the *master* data and the *using* data contain variables with the same name, the merged data keep the variable of the same name from the *master* data and discard the variable of the same name from the *using* data. Before you get used to Stata, it is better to avoid the same variable name to be used for a variable in the *master* data on one hand and for a variable in the *using* data on the other hand.

Finally, let me briefly talk about the command *append*, which also combines two data sets. Intuitively, *merge* combines data horizontally where the number of observations remains the same but the number of variables increases, while *append* combines data vertically where the number of variables remains the same but the number of observations increases. In my experience, *merge* is much more frequently used than *append*. For details of *append*, *search append* under the *help* menu.

8. **Tutorial 3 Homework**

   (a) Data from Palau 2000 Census are attached to this Tutorial. One data set (palau_hh_2000.dta) contains household-level data, while the other data set (palau_individual_2000.dta) contains individual-level data. Palau is a Pacific island country with a small population. *tabulat*ing *sex* in palau_individual_2000.dta reveals that there are only 19,049 people in the data set. Remember that this is a Census data set, so this is the whole population of Palau as of 2000. Both palau_hh_2000.dta and palau_individual_2000.dta include the *string* variable *CASE_ID* which is household id. In palau_individual_2000.dta, the *numeric* variable *sequence_number* indicates individual id within each household. Create and run a *do* file that implements the following tasks. You must use asterisk (*) to indicate the question numbers i. through iv. in your *do* file. You will be penalized for not using asterisk (*) to indicate the question numbers. Please also put your name and ID in the top of your do file using asterisk (*).

      i. palau_hh_2000.dta contains the variable *h17* which records answers to the question "Do you have a flush toilet?" The respondents select one of five choices: 1. Yes, in this unit; 2. Yes, in this building; 3. Yes, outside this building; 4. No, outhouse or privy; or 5. No, other or none. First, *rename h17* to *toilet*. Then, attach appropriate *variable* and *value labels* to *toilet*.

ii. In palau_individual_2000.dta, the variable *religion* records answers to the question "What is your religion?" Create a category variable called *religious_group* that equals to one if the respondent has either Catholic or Protestant as his/her religion, two if the respondent has no religion or refuses to answer the question, and three otherwise. If the answer to the question is missing (*religion*=.), then *religious_group* should also be missing.

iii. Attach appropriate *variable* and *value labels* to *religious_group*.

iv. Calculate mean household income separately for three religious groups we have just created. Hint: Merge the data sets as appropriate, first. Using the merged data, simply calculate mean household income separately for the three religious groups you just created. The variable *fam_income* contains data on household income.

(b) If you have questions, stop by one of TA's. Stata Exercise 3 is due at the beginning of the class on April 25 (Wednesday). No late assignment will be accepted. You need to turn in a hard copy of your result file (convert your *smcl* file into a *txt* file). You MUST create your result file from a *do* file. (To create an output to be submitted as homework, you must complete a *do* file first and then use the command *do* to run the *do* file. Please do not work interactively with Stata to create an output to be submitted as homework.) If your result file is created interactively with Stata, you get zero credits from this assignment. Please do not include error commands in your *do* file.

End of Tutorial 3