Center for Effective Global Action
University of California, Berkeley

Module 1.2: STATA Basics

Contents

List of Figures

1. INTRODUCTION

The goal of this module is to introduce STATA, the statistical software that we will use for this course. STATA is a popular statistics package used across disciplines to (1) process large data sets, (2) visualize the data, and (3) perform a range of statistical analyses on the data. While we hope you appreciate the ease and possible rigor of analysis in STATA, you should know that it is not the only statistical software capable of conducting the analysis presented in this course; other examples of statistical packages include SAS and R. However, technical support will be provided only for STATA in this course. **Don't be scared of STATA; it is one of the easiest statistical packages available and we are there to support you!**

The objective of this module is to provide an overview of STATA and enable you to perform some basic operations in STATA. For more systematic basic- and intermediate-level training in STATA, you may access the following free online resources available from Princeton University:

- ✓ http://data.princeton.edu/STATA/
- ✓ http://dss.princeton.edu/training/
- ✓ http://www.princeton.edu/~otorres/STATA/

UCLA also offers free online instruction material for statistical analysis using STATA (http://www.ats.ucla.edu/stat/STATA/default.htm), and the D-lab at UC Berkeley frequently offers free courses on STATA (http://dlab.berkeley.edu/).

Also, while we will cover some basic tools of statistical analysis, this course assumes a prior level of competency in the statistical theories related to hypothesis testing and regression models. If you wish to review these subjects, we recommend taking one of the free online courses offered by UC Berkeley (http://webcast.berkeley.edu/), Princeton University (http://dss.princeton.edu/training/), MIT (http://ocw.mit.edu/index.htm), or your university.

Data processing, or preparing datasets for analysis, is an important precursor to data analysis. While we will use simplified datasets for pedagogical purposes, in real life "painful" datasets are prevalent. For this reason, some of the problem sets in this and other modules will expose you to "real-life datasets" and offer tips to deal with common problems such as missing data, non-response and attrition, an over-abudance of variables, and multiple datasets. We will cover some basic data processing and management tasks in this module, and then will offer tips and guidance as required in subsequent modules.

How to use this learning guide: We recommend first going through the learning guide quickly to assess how conversant you are with the concepts, tools and methods described. If you are not conversant with the presented material, then we recommend reading through the learning guide carefully and completing the short exercises. For Module 1, the video sessions—and quizzes therein—should take about 120 minutes to complete. The homework is expected to take about 60 minutes.

At the end of this session, you should be able to:

- ✓ Understand the setup and basic components of STATA
- ✓ Perform basic data processing and management operations in STATA

2. STATA SET UP

2.1 Obtaining STATA

Students at UC Berkeley can purchase a single-user one-year license for Small STATA through UC Berkeley's Graduate Plan for $49. However, this least-expensive version will not let you analyze datasets with more than 99 variables and 1200 observations. Therefore, we recommend STATA/IC version 13, which allows for up to 2047 variables and over 2 billion records and is available for $98 on an annual basis. Additionally, most department computer laboratories will have the larger STATA/SE version 13 installed for use by students and faculty.

2.2 STATA Interface

Once STATA is installed, it can be started from the Program Menu of your PC or by double-clicking the STATA shortcut that may be available on your desktop. The STATA interface has 5 windows, as shown in Figure 2 and described below:

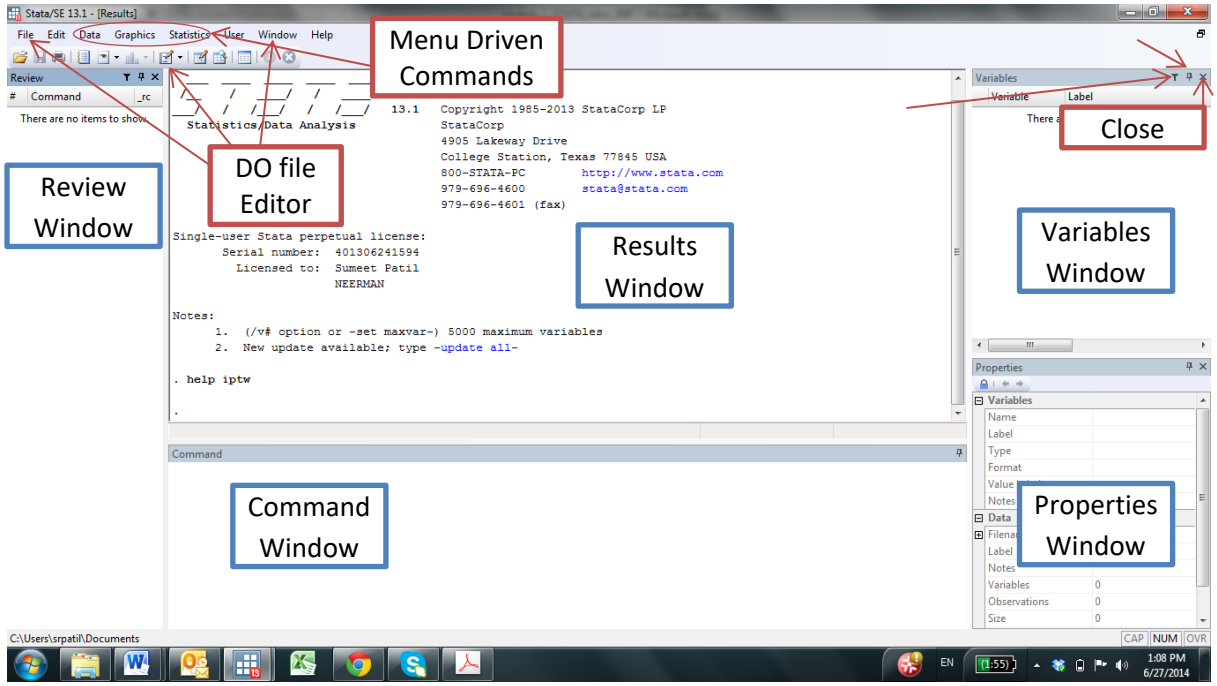


Figure 1. STATA main window

- ✓ Command window: where you tell STATA the operations that you would like to perform. You will enter these operations using STATA syntax or code.
- ✓ Results window: where output is displayed. This window allows you to observe the outcomes of any task that STATA performs, but it is not possible to save results from this window. We will explore alternatives to saving results later.
- ✓ Review window: where previous STATA commands are shown. It keeps track of all operations that have been performed during the current session. this window also provides a shortcut to access to previously run commands.
- ✓ Variables window: where all variables in the active data file are shown. Any time a variable is selected, a small arrow will show up, and the user can click on the arrow to make the name of the variable appear in the command window.
- ✓ Properties window: where details about the active data file (including information about each variable) are provided.

Each window typically has capabilities for:

- Close: to shut the window. To reopen the window, go to the windows menu on the menu bar at the top.
- Auto hide: to hide the window. To see window, click on the "hide" flag.
- Filter: to filter the content of the window. This is most useful in the variables and review windows to focus on relevant subsets.

You can use these buttons and drag the borders of each window to customize the look of the STATA user interface.

To exit STATA, type exit in the command window along with the option clear or shut the window directly.

2.3 STATA Help

Like most software, the best way to learn STATA is to practice and experiment with it. STATA also has help files accessible through the main menu. For STATA guidance for any command type "help [command name]" into the command window (Figure 1). Figure 2 is the screenshot of a help file from STATA for the "regress" command ("help regress"). These help files provide information about a specific command, guide you to general resources, and help you "search" for specific terms. The "search" is performed over STATA official documents and FAQs.

STATA list and other online resources are also available through Google and other search engines. Often writing a specific and detailed query in Google will yield specific guidance from the STATA community. As we will discuss later, you will also be able to find and use "user-written" STATA programs (usable as customized commands) to perform certain tasks.

The command *findit* is useful for broader searches and permits users to download user-written commands that are not part of the official STATA version.

Exercise: Google "export regression results from STATA to formatted table" and find at least one STATA command that you can use to create professional tables of regression results. Install these commands in STATA by following online instructions.

Answer Key: outreg, outreg2, esttab, estout. These programs can be installed by typing SSC install [PKG Name] in the command prompt.

Exercise: Type *ssc* in the Command Window. Explore the "hot" downloads and install them if you find them useful after reading the description.

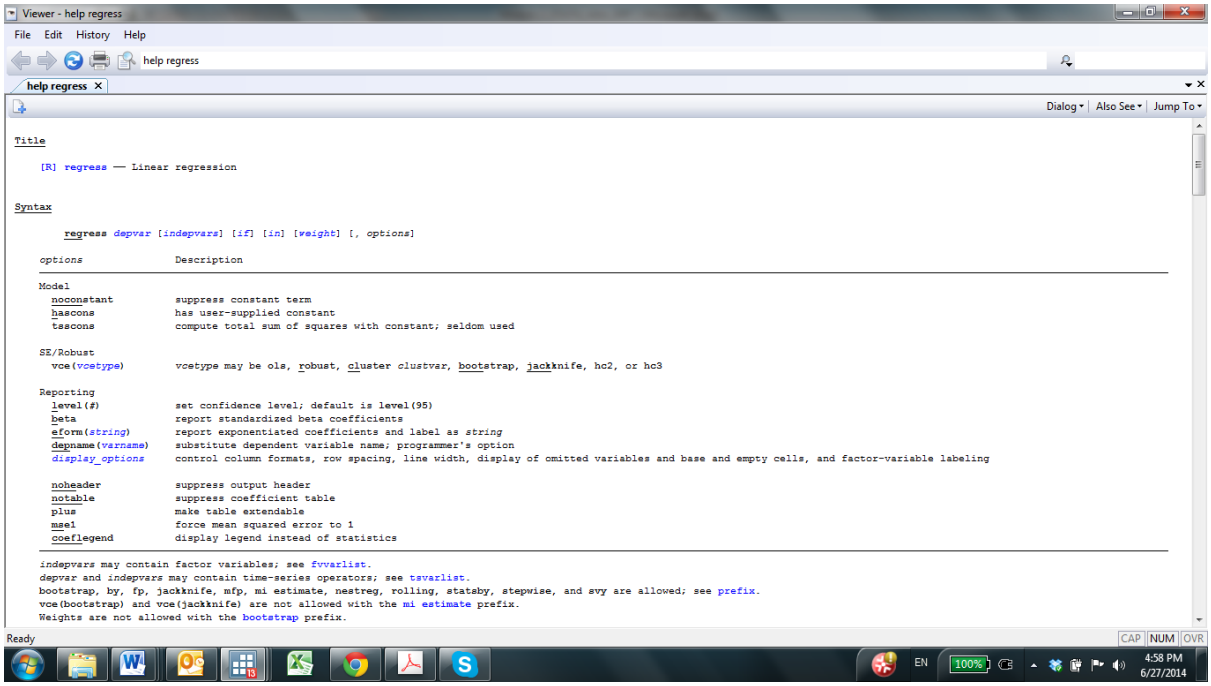


Figure 2. Help file for regression.

2.4 STATA Commands and the DO file

A key advantage of statistical software like STATA is clear documentation of its analysis steps and their reproducibility. Therefore, although STATA users can either issue commands directly or use menu-driven commands, we highly recommend that all commands be written and executed through a DO file. The DO file editor can be opened in three different ways, as denoted in Figure 1, and functions in a similar way to a Microsoft Word or WordPad interface.

One often runs commands by directly typing and executing though the command window or by using the menu-driven options for commands (see Figure 1). You may also refer to the command help files ; most help files include a menu-driven option to executive the command. Indeed, using these options will help you better understand STATA's capabilities, and will also ensure operational ease during the initial period in which you are becoming conversant with STATA syntax.

The DO file editor is shown in Figure 3. This editor opens as a separate window. No matter how you run a command, you should always copy the required commands from the review window (you can select multiple commands) and paste them into a DO file. It is also important to include comments and notes in the DO file to document any assumptions or decisions that went into your code or to explain the analysis steps.

For example, in Figure 3 the comments are in green font. To execute a command or a block of commands from a DO file, you can select those commands (for example, the grey highlighted text in Figure 3) and click the "Execute" option from the "Tools" ("View" in Mac) menu or click the Execute button on the menu bar.

Comments can be added to the DO file by:

- ✓ Starting a comment line with '*'. Please note that * can only be placed at the start of a line. For example:

```
* Take mean of the age of the respondent
mean age_respondent
```

- ✓ Starting a comment line with '//'. '//' can be placed at the end of a command but before starting a comment in the same line; a blank space after the command is essential. For example:

```
// Take mean of the age of the respondent
mean age_respondent  //  Note the blank space before the comment
```

- ✓ A block of multiple line comment can be placed between /* and */ delimiters. For example:

```
/* Sometimes we have multiple lines of comments or text.
Instead of using '*' or '//' at the start of each line, we can block out multiple lines as
    comment as shown here */
```

Additionally, long command line texts in the DO file can be split into multiple lines for better readability by placing '///' at the end of the text, separated from the text by blank space. For example, the command "regress income age gender education occupation family_size spouce_occupation number_children region" can be written in three lines as:

```
regress income  ///
age gender  education  occupation  ///
family_size  spouce_occupation  number_children  region
```

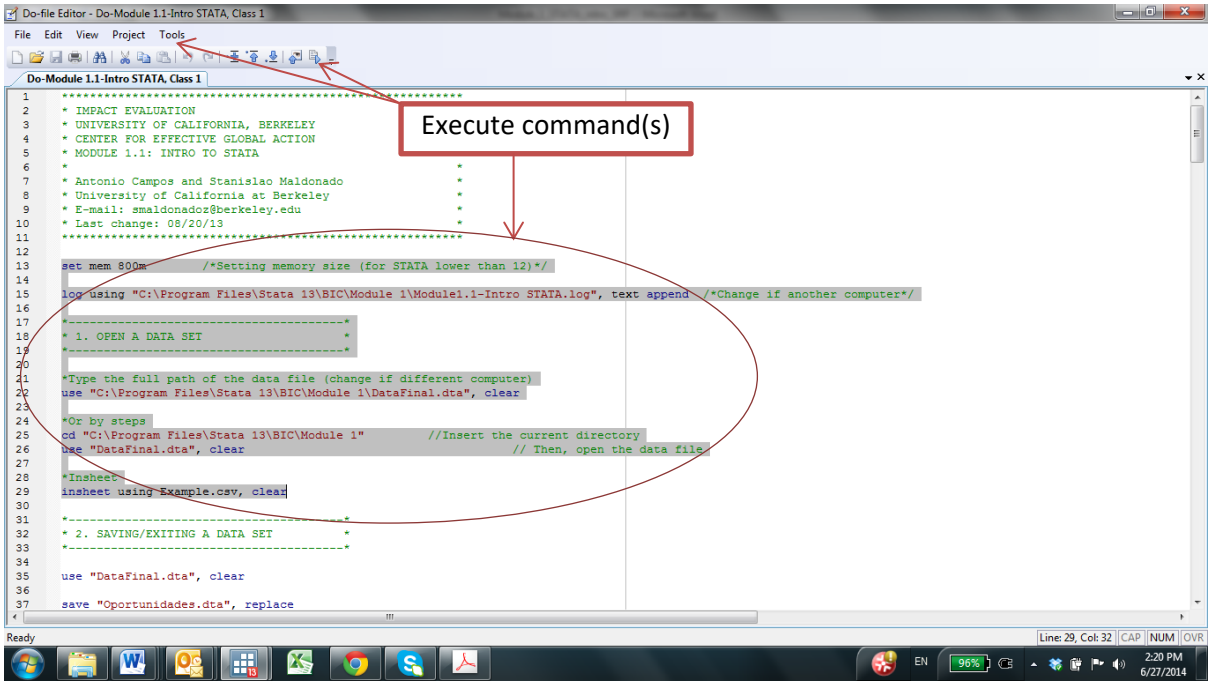


Figure 3. DO file editor to execute STATA command

Figure 4 is a screenshot of STATA after the user opens a menu-driven option. After selecting the appropriate command, the dialogue box guide the user in implementing the command. The help file for the selected command will provide technical guidance on using the various command options and settings as well as guide you to technical literature which provides more information about the analysis.

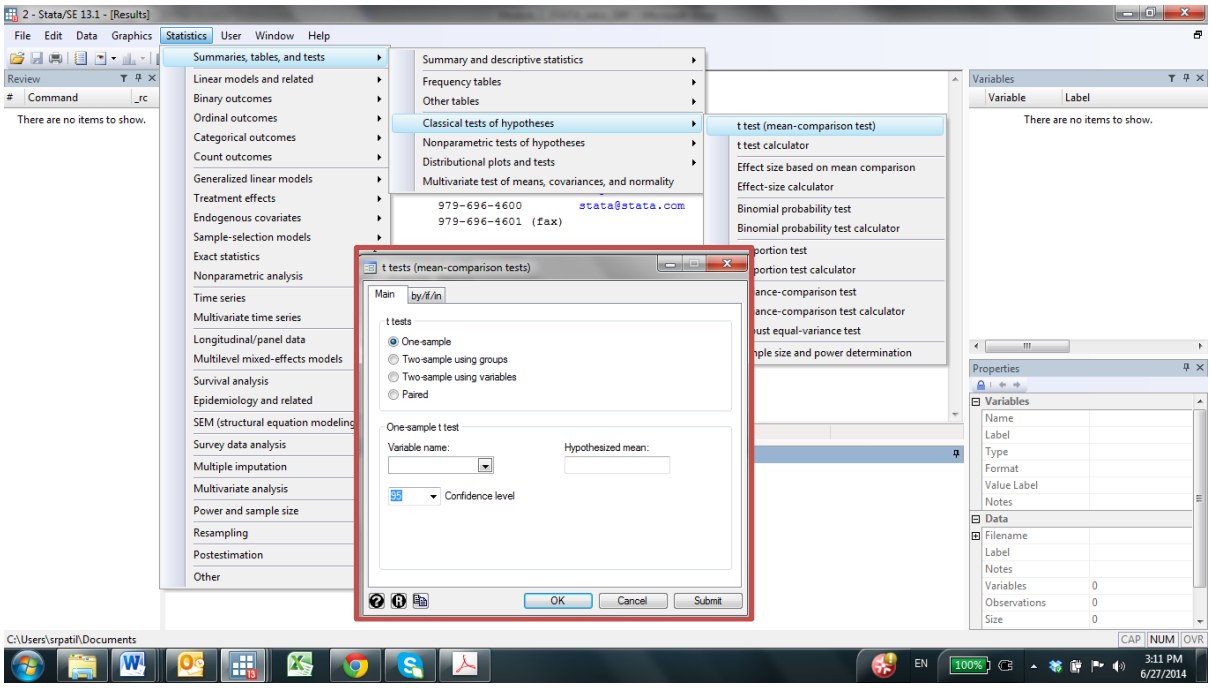


Figure 4. Example of menu-driven selection and the dialog box for a t-test

2.5 Using *.log Files

By default, STATA output is displayed in the results window. There are commands which can store or export the results of statistical commands to another STATA dataset or Excel file (see documentation for the STATA user-written programs *outreg2, estpost, estout,* and *tabout* if you have not already done so). However, it is often necessary, or merely good practice, to "log" or record the content of the Results Window. Logs include outcomes and results from STATA commands, warnings, and additional messages. STATA has a provision to save these contents in a "log" extension file which can be opened in any text editor (such as Microsoft Word, WordPad, or Notebook). These log files are useful in order to share the content of STATA analysis session with someone who does not have either STATA or your data. The following commands open and close log files. Users usually open a log file at the start of the DO file and close it at the end of the file.

- ✓ Open Log File: `log using "[FOLDER_PATH]/[FILE_NAME].log", text replace .` 'Text' indicates that is the log is formatted as a simple text file, while 'replace' tells STATA to overwrite anexisting log file with the same name (if such a file exists).
- ✓ Close log File: `log close .` This command is required at the end of the DO file (or section of DO file) in order to tell STATA to stop logging your work.
- ✓ Type "help log" into the Command Box for more details.

2.6 Opening and Saving Datasets

STATA data files are denoted by the *.dta extension. We can open an existing dataset, import data from other spreadsheets or ASCII data, and even manually enter the data.

- ✓ Importing the Data: The file menu (see Figure 1) helps us import data from Excel, XML or text form (as well as other data types). The commands that can be used in the Command Window or DO file include: 'infile', 'import' or 'insheet'.

  Exercise: Using the File>Import menu-driven commands, import the file "Example.csv", which is one of the files you downloaded at the beginning of this module. Import this file using the *insheet* command as an alternative method. (Hint: use the "help insheet" for STATA guidance)

- ✓ Directly Typing the Data: This option is rarely use in real life, but we can directly type in the data in STATA by using the command *input* and see the data using the command *list*. Note: for any datasets longer than a few rows and columns this method quickly becomes impractical.

- ✓ Opening a STATA dataset: An existing STATA dataset can be used through the main menu (FILE>OPEN) or through the command prompt. There are two common ways to open STATA data sets: write the full file path or change the main directory and then open the dta file. For example:

```
use "C:\Module 1\myFirstData.dta", clear
```

or

```
cd "C:\Module 1"
use "MyFirstData.dta", clear
```

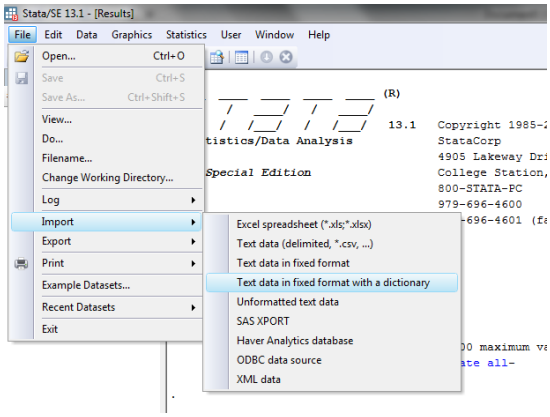


Figure 5. Importing data into STATA

To save a dataset we can either use the SAVE button or the file menu. The command for saving a file is

```
save "[FILE_PATH]/[FILE_NAME].dta", replace
```

where "replace" instructs STATA to overwrite the file in memory (if such a file exists). If you have transformed the dataset in some way, it is generally good practice to change the name and/or path of the dataset, in order to avoid losing or irrevocably changing data from the original dataset (see bullet point below).

Exercise: Open the DO file editor and find "First_DO_file.do" from the folder where you saved the STATA files for Module 1. Change the file path to the folder on your computer where you saved the Module 1 STATA files. Select the entire DO file content (ctrl + A) and click the "Execute" button. Next, type *describe* at the command prompt. What information does STATA provide about the dataset? Explore the *describe* command by using the different options as explained in its help files ("help describe"). Now, type *browse* in the command prompt to open the data matrix you have imported. Explore the new window. Note: you cannot make edits directly to the data matrix in "browse" mode, but "edit" mode allows you to make direct edits.

Tips

- ✓ A good indicator of your progress in STATA is that you don't or rarely use the menu and window driven commands and instead directly execute them from a DO file (or, in some cases, typing at the command prompt).
- ✓ Always maintain your raw data files in a separate folder from your 'derived', or cleaned, data files. Maintaining the sanctity of the original data is important in case you need to recreate your analysis from scratch.
- ✓ Instead of always writing the entire file path when you save or open data, especially if you frequently change the folder in which the data is stored, it is often more convenient to

define a global path at the beginning of the DO file and use it throughout as follows. We can do this through a global macro ("help macro").

```
global DataDir  "C:\STATA\cega\modue_1"
[various STATA commands]
use $DataDir\myFirstData.dta, clear
[various STATA commands]
save $DataDir\NameYouGive.dta, replace
```

- ✓ Include the following two lines at the start of your DO file. You can read the help files to know more about these commands.

```
clear all
set more off
capture log close
```

- ✓ In large datasets with thousands of variables, browsing for variables can be difficult. The command *lookfor* [SEARCH TERMS] searches for the search terms in the labels of active variables . For example, type *lookfor poverty* at the command prompt after running First_DO_file.do; STATA will identify the variable which has the word "poverty" in its label.

3. DATA PROCESSING COMMANDS

Load the dataset MyFirstData.dta in STATA as explained in Section 2.6. Also, open the DO file editor. It will automatically open a blank new DO file available for you to use. ***From this point forward, you should execute menu-driven or other commands (See Figure 4) from the STATA command window. Once a command executes without error and as you expect, copy and paste that command(s) from the review window and copy it into the DO file.***

Please review the help files in STATA for each of the commands below. Note that many commands have additional options and can be formulated in multiple ways. Here we merely describe the basic features of the commands you may use most often. The formatting below will help you interpret STATA help file nomenclature, and customize the STATA commands when you use them.

- ✓ Square brackets: Optional text in the command
- ✓ Italics: User should customize the information
- ✓ Underlined: Short-form instead of full command name that can be used to call the command

3.1 Typical Options and Terms Used in Command Syntax

- ✓ [varlist]: this means the list of variables in the dataset. The list can be specified the following ways:
  - myvar — just one variable (Note that STATA is case sensitive)
  - myvar thisvar thatvar — three variables (space in between each variable)
  - myvar* — all variables starting with 'myvar'
  - *var — all variables ending with 'var'

- my*var — all variables starting with 'my' and ending with 'var' with any number of other characters between
- my~var — only one variable starting with 'my' and ending with 'var' with any number of other characters between. Error given if multiple variables match this criteria
- my?var — variables starting with 'my' and ending with 'var' with only one other character between
- myvar1-myvar6 — all variables from myvar1 to myvar6 which are ordered sequentially (columns next to each other) in the dataset.
- _all *OR* * — all variables in the dataset

✓ [varname]: indicates that there is only one variable to be specified.

✓ [numlist]: this is a list of numbers separated by blanks or commas. There are a number of shorthand conventions available to reduce the amount of typing necessary. For example:

- 2 : just one number
- 23 2 11 : three numbers
- 5/8 : four numbers: 5, 6, 7, 8
- -1/2 : four numbers: -1, 0, 1, 2
- 10 15 to 30 : five numbers: 10, 15, 20, 25, 30
- 10 15:30 : five numbers: 10, 15, 20, 25, 30
- -1(.5)2.5 : the numbers: -1, -.5, 0, .5, 1, 1.5, 2, 2.5
- 1 2 3/7 10(2)16 : eleven numbers: 1, 2, 3, 4, 5, 6, 7, 10, 12, 14, 16

✓ [filename]: as explained in Section 2.6

✓ [if]: Commands are executed if the 'if' condition is met. The basic structure of using IF is `command if exp`, with the command only being applied to the rows of data in which the 'if' command is true. Below are some examples of how "if" is used

- `command if myvar1 == myvar2 // == indicates equality, while = indicates the assignment of a value.`
- `command if myvar1 != 234 // != or ~= indicates in-equality (or myvar1 "does not equal" 234)`
- `command if (myvar1 == myvar2) & (myvar1 != 234)`

✓ [in]: The commands are executed only for the observations or records within the specified 'in' range. The typical way of specifying a range is #/# where the # indicates the starting and the ending observation number. For example,

- 5/20 indicates the 5th to 20th observations
- –20/L indicates the 20th from the last to the last observations.
- F/L indicates all observations

3.2 Typical Operators

We often need various "expressions" in STATA commands based on arithmetic, logical, and relational operators as follows:

Arithmetic		Logical		Relational	
+	addition	&	and	>	greater than
-	subtraction	|	or	<	less than
*	multiplication	!	not	>=	> or equal
/	division	~	not	<=	< or equal
^	power			==	equal
-	negation			!=	not equal
+	string concatenation; e.g. "this" + "that" = "thisthat"			~=	not equal
*	string multiplication; e.g. 2*"this" = "thisthis"				

Missing values for a numeric variable are denoted by a dot (.), and string variables by two quotation marks ("").

Exercise: Type "`display (347 * 821) ^ 0.5`" at the command prompt. Did you get–533.74807? This command is for quick on-screen calculations. You are able to use results from regression analysis or summary statistics in such calculations, but that will be covered in a future module.

3.3 Commands to Understand the Data Structure

✓ **`describe`**

This command is used to describe the dataset. The basic format of the command is:

`describe [varlist] [, memory_options]`

Figure 6 is the output of a describe command.

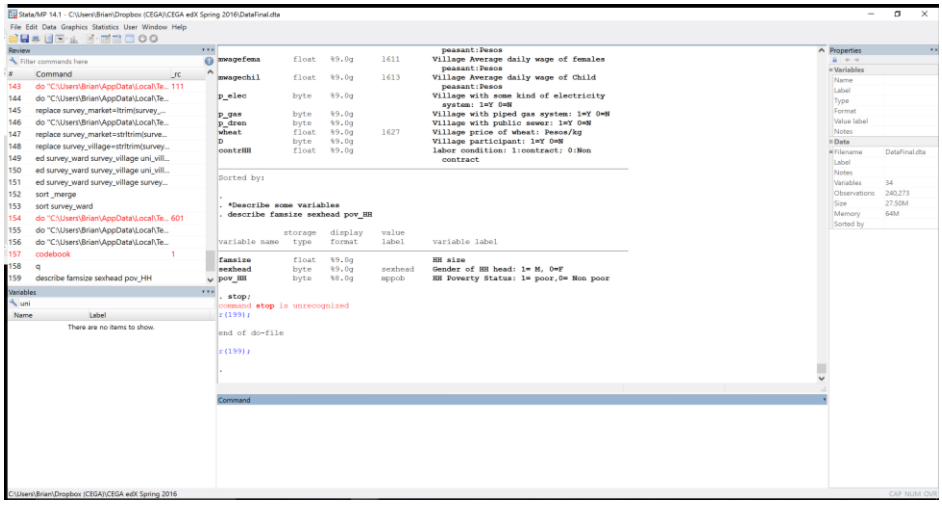


Figure 6. Output of DESCRIBE command

✓ **browse**

In the STATA interface there are two buttons for browsing data, with the second allowing us to manually edit the data as well. In the command prompt we can use the following command:

```
browse [varlist] [if] [in] [, nolabel]
```

where the option *nolabel* indicates that values of categorical variables are used instead of any labels assigned to those categories. Replacing 'browse' with 'edit' generates a command which allows the data to be manually edited in the browse window..

✓ **sort**

Often we need to sort data using one or multiple variables. This is accomplished by:

```
sort varlist [in] [, stable]
```

This command always sorts in ascending order in the sequence of variables specified in the varlist.

Exercise: in 'help', search for "ascending and descending order sort". Did you find a command `gsort`? Explore this command and sort myFirstData.dta in different ways.

✓ **list**

Often we wish to visualize partial data, such as a few records of a few variables. The commands `browse` and `sort` you are helpful, but `list` is a faster and sometimes more convenient way to visualize the data. The basic command syntax is,

```
list [varlist] [if] [in] [, options]
```

✓ **count**

This command counts the number of observations in the dataset that fulfil the given condition (specified by 'if'). The basic syntax is,

```
count [if] [in]
```

3.4 Creating New Variables

We often need to impute new variables that are useful in our analysis of the raw dataset. For example, we may wish to define and report income as a three-bin categorical variable (e.g. low, medium, and high levels) instead of income as a continuous variable. To consider a more involved example, household income is measured through a series of questions about the earning members of the family through primary, secondary and tertiary sources of income. These multiple questions or variables from the raw dataset can be converted to the same unit – say, US Dollars per month – and added to estimate the monthly household income. If we wish to estimate the per capita income then this value is divided by the number of family members. You will need to generate numerous such variables in the future; here is the code to do so:

✓ **generate**

You will typically use the following syntax to generate a new variable.

```
generate newvar =exp [if] [in] [, by(varname)]
```

The *exp* can be operations of any numbers or variables using the operators described in Section 3.2 such as,

- newvar = myvar1 * 234
- newvar = myvar1 & myvar2
- newvar = .
- newvar = “”

The *exp* can also use a large series of available functions in STATA for transform variables. We recommend you get conversant with the following basic functions for numeric and string variables using STATA help files.

- Mathematical functions
  - abs(x) : returns the absolute value of x
  - ceil(x) : returns the unique integer n such that n - 1 < x < n. Returns x (not ".") if x is missing
  - floor(x) : returns the unique integer n such that n < x < n + 1. Returns x (not ".") if x is missing
  - int(x) : returns the integer obtained by truncating the decimal expansion of x
  - ln(x) : returns the natural logarithm of x
  - log10(x) : returns the base 10 logarithm of x.
  - logit(x) : returns the log of the odds ratio of x, ln{x/(1-x)}.
  - invlogit(x) : returns the inverse of the logit function of x, exp(x)/{1 + exp(x)}.
  - sqrt(x) : returns the square root of x.
  - sum(x) : returns the running sum of x, treating missing values as zero. For example, y=sum(x) will generate values of y such that the j^{th} observation on y contains the sum of the first through j^{th} observations on x.

- String Functions
  - itrim(s) : returns s with multiple, consecutive internal blanks collapsed to one blank. itrim("hello there") = "hello there"
  - length(s) : returns the length of s. length("ab") = 2
  - ltrim(s) : returns s without leading blanks. ltrim(" this") = "this"
  - string(n) : converts n (entered in numerical format) to a string. string(1234567) = "1234567"
  - substr(s,n1,n2) : returns the substring of s, starting at character n1, for length n2. If n1 < 0, n1 is interpreted as distance from the end of the string; if n2 = . (missing), the remaining portion of the string is returned. substr("abcdef",2,3) = "bcd". substr("abcdef",-3,.) = "def"
  - mdy(M,D,Y) : returns the date format variable corresponding to M, D, Y

- destring(x) : An independent function (not an 'exp') that converts string variables to numeric variables. 'Destring' includes options to either replace the variable with a numeric value or generate a new variable. This command is often useful when, in importing data, a variable is imputed as a string by STATA when the variable is merely an integer with a single character entry (sometimes a text character is mistakenly entered during data entry).
- Tostring : An independent function that converts a number to a string variable. This is sometimes needed when you want to perform a string operations or use string functions for a numeric variable.

- Random Number Functions
  - runiform() : returns uniformly distributed random variates on the interval [0,1). runiform() takes no arguments, but the parentheses must be typed. runiform() can be seeded with the *set seed* command. To generate random variates over the interval [a,b), use a+(b-a)*runiform(). To generate random integers over [a,b], use a+int((b-a+1)*runiform()).
  - rbinomial(n, p) : returns binomial(n,p) random variates, where n is the number of trials and p is the success probability.
  - rnormal(m, s) : returns normal (Gaussian) random variates, where m is the mean and s is the standard deviation.

Exercise: create a new variable named poor_male which is 1 if the household is categorized as poor (pov_HH == 1) and the head of the household is male (sexhead == 1), and 0 otherwise.

✓ **replace**

The typical syntax to replace values of an existing variable is:

```
replace oldvar = exp [if] [in]
```

the *exp* are similar to those used for the `generate` command above and can use the oldvar. Here are two examples:

```
replace oldvar = oldvar * 5.
replace oldvar = oldvar * -1 if oldvar < 0
```

✓ **recode**

This command is useful to deal with missing values or special codes in the existing variables and to change the existing values of categorical variables. `recode` changes the values of numeric variables according to the specified rules. Values that do not meet any of the conditions of the rules are left unchanged. The examples of the rules are given in the table below. The basic syntax, in which we have an option to either create a new variable or change the values of an existing variable, is:

`recode varlist (rule) [(rule) ...] [, generate(newvar)]`

Rule	Example	Meaning
# = #	3 = 1	3 recoded to 1
# # = #	2 . = 9	2 and . recoded to 9
#/# = #	1/5 = 4	1 through 5 recoded to 4
nonmissing = #	nonmiss = 8	all other nonmissing to 8
missing = #	miss = 9	all other missings to 9

Exercise: change the values of the new variable created above (poor_male) using the 'recode' command such that the value 1 becomes 2, the value 0 becomes 1, and the missing values become 9. Save the recoded variable as a new variable named "poor_male12". Now execute command `tab poor_male poor_male12`.

✓ **egen**

This is one of the most useful commands in STATA and you will use it for several real life data transformation needs. Therefore, please carefully go through the help file for *egen* and understand various options available. The most typical syntax of egen is:

`egen newvar = function(arguments) [if] [in] [, options]`

The options in *egen* are specific to the functions you use and there are several functions available. A few key ones are as follows.

- `anycount(varlist), values(numlist)`: Generates the number of variables in varlist for which values are equal to any integer value in a supplied numlist. Values for any observations excluded by either if or in are set to 0 (not missing).
- `anymatch(varlist), values(numlist)`: Generates1 if any variable in varlist is equal to any integer value in a supplied numlist and 0 otherwise. Values for any observations excluded by either if or in are set to 0 (not missing).
- `anyvalue(varname), values(numlist)`: Generates the value of varname if varname is equal to any integer value in a supplied numlist and is missing otherwise.
- `cut(varname), {at(#,#,...,#)| group(#)} [icodes]`: Generates a new categorical variable coded with the left-hand ends of the grouping intervals specified in the at() option. The at(#,#,...,#) supplies the breaks for the groups at the numbers # in ascending order. Instead of at() option, the users can use the group() option. The option group(#) specifies the number of equal frequency grouping intervals to be used in the absence of breaks, a useful feature when you wish categorical variables for the quartile, pentile or decile of a continuous variable. The option *icodes* requests that the codes 0, 1, 2, etc., be used in place of the left-hand ends of the intervals. When you use option group(#) you don't need to specify *icodes* option because it is automatically invoked.
- `group(varlist) [, missing]`: Generates one variable taking on values 1, 2, ... for the unique groups formed by varlist. varlist may contain numeric variables, string variables, or a combination of the two. The order of the groups is that of the sort order of varlist. missing indicates that missing values in varlist (either . or "")

are to be treated like any other value when assigning groups, instead of as missing values being assigned to the group missing.

- `count(exp)[, by(varlist)]`: Generates a constant containing the number of non-missing observations of *exp* in unique groups found by *varlist*. Note, exp can be any *varname*.
- `mean(exp) [, by(varlist)]`: Generates a constant (within groups formed by *varlist*) containing the mean value of *exp*. Similarly, functions such as sd (standard deviation), max, and min are available.
- `rowmean(varlist)`: Generates the (row) means of the variables in varlist. However, *rowmean* ignores missing values; for example, if three variables are specified and, in some observations, one of the variables is missing, in those observations *newvar* will contain the mean of the two non-missing variables, while other observations will contain the mean of all three variables. Where all variables have a missing value, *newvar* is set to missing. Similarly, we have functions such as rowmax, rowmin, and rowtotal.
- `total(exp) [, missing] [by(varlist)]`: Generates a constant (within the groups formed by *varlist*) containing the sum of *exp* treating missing as 0. If missing is specified and all values in exp are missing, newvar is set to missing.

Exercise: Generate the following two variables: (1) a new variable named *family_category* which is 1 if family size is 2 or less, 2 if family size is between 3 and 5, 3 if family size is between 6 and 9, 4 if family size is between 10 and 14, and 5 if family size is 15 or larger; and (2) a new variable income_decile which reports deciles (10 equal frequency groups) for the variable IncomeLab. The categorical variable income_decile will have 10 values.

3.5 Labelling Variables and Values

Raw datasets, especially large ones, often contain variable names which are not intuitive. For example, don't be surprised to find variable named a001s01 or d23s02r34. For this reason, it is important to "label" variables so that we understand what exactly they mean, but variable labels cannot be so long that they appear verbose. To attach a label to a variable use the `label variable` command in the following way:

```
label variable sexhead "Sex of the head of the household"
```

The values of categorical variables have a meaning unlike those of continuous variables. We can clarify the values of the variable to data users . This is achieved by first defining a label of the values and then applying those value labels to a variable as shown below.

```
label define sexlbl 0 "Female" 1 "Male"

label values sexhead sexlbl
```

The above commands can also be used to change existing variable or value labels. Please note that the value label, once "defined", cannot be modified. If you want to use the same name for the value label, then you have to "drop" the existing value label and recreate it:

```
label drop sexlbl
label define sexlbl 0 "Female head of household" 1 "Male head of household"
label values sexhead sexlbl
```

Exercise: Assign an appropriate label to the variable name and variable values for the two variables you newly created previously – income_decile and family_category.

3.6 Deleting Variables and Observations

Over the course of your analysis you may find a need to delete variables. Sometimes you wish to drop the observations meeting or not meeting certain conditions from the dataset. The following commands can help you in deleting variables (columns) or observations (rows) from the dataset:

- Drop variables : `drop varlist`
- Drop observations : `drop if exp`
- Keep variables : `keep varlist`
- Keep observations that satisfy specified condition : `keep if exp`

3.7 Basic Summary Statistics

✓ **`summarize`**

summarize provides the summary statistics such as the count, mean, standard deviation, minimum, and maximum for the selected variables. The syntax is:

`summarize [varlist] [if] [in] [weight] [, options]`

The option `detail`, which provides more summary statistics, if often useful; observe the difference in output for the two commands in Figure 7.

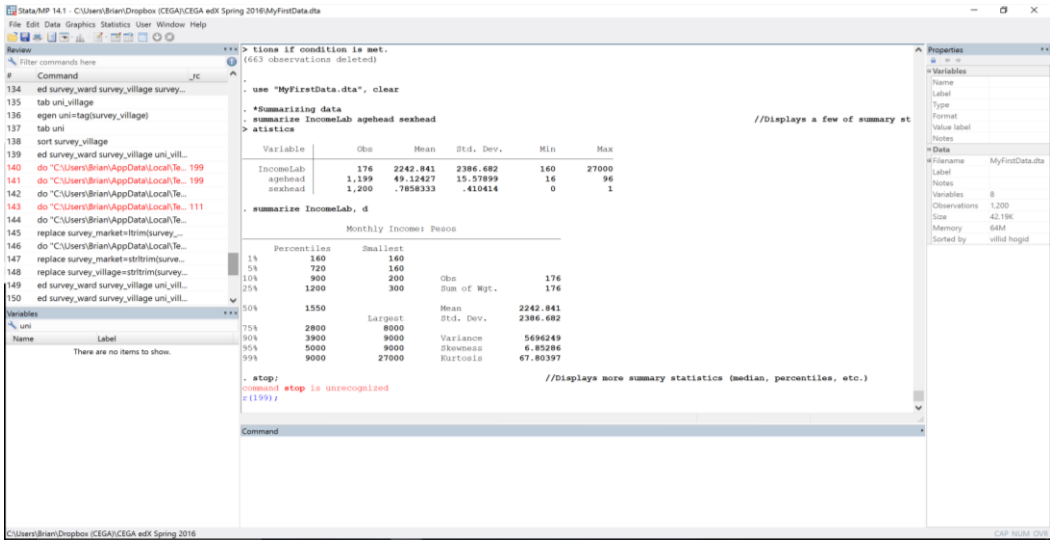


Figure 7. Output of SUMMARIZE Command

✓ **tabulate**

This command is particularly useful for categorical variables, while as summarize works best for continuous or binary variables. The oneway tabulate command provides the frequency table for the selected variables:

```
tab1 varlist [if] [in] [weight] [, tab1_options]
```

We can also create a cross tabulation of frequencies of two variables as:

```
tabulate varname1 varname2 [if] [in] [weight] [, options]
```

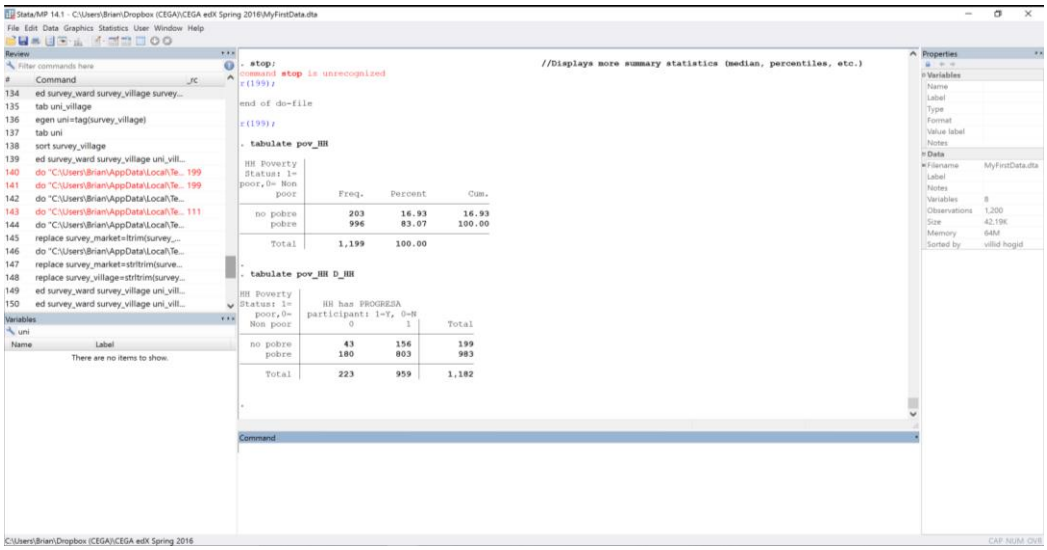


Figure 8. Output for TABULATE Command

Exercise: Execute the commands `tab pov_HH D_HH, cell` and `tab pov_HH D_HH, col`. How did the output change? What additional information does the second command provide? Now execute a command `tab pov_HH D_HH, sum(Inc)`. What additional information does this option provide? Is this useful to understand the dataset? Why do you think we could write only "Inc" instead of "IncomeLab"? (Answer: STATA can parse partial variable names as long as the partial name indicates a unique variable. Similarly, most commands can be specified with only few starting characters/letters)

✓ **tabstat**

Often we need more or different statistics than those provided by `summarize`. See Figure 9 for sample output.

```
tabstat varlist [if] [in] [weight] [, options]
```

The options that are most frequently used are:

- `by(varname)` group statistics by a categorical variable. For example, you can get descriptive statistics for several variables when sex of household head is female and then the same statistics for observations when sex of the household head is male.
- `statistics(… …)` report specified statistics. Some of the usual statistics that are available in tabstat command are: mean, count, sum, max, min, sd, semean, percentiles (p1, p5, …, p99)

- `columns(var)` this option formats the output such that variables are in the columns and statistics are in the rows; the default
- `columns(stat)` this is a formatting option such that the statistics are in the columns and the variables are in the rows.

Exercise: Execute tabstat command to get statistics (select 5-6 measures) of your choosing for the variables IncomeLab, pov_HH, agehead and sexhead and group them by variable D_HH. You can use the following menu navigation: Statistics > Summaries, tables, and tests > Other tables > Compact table of summary statistics. Read the help file for tabstat. Also, (optional) read help file for the command `table`, which provides a more advanced and flexible options to produce summary statistics.

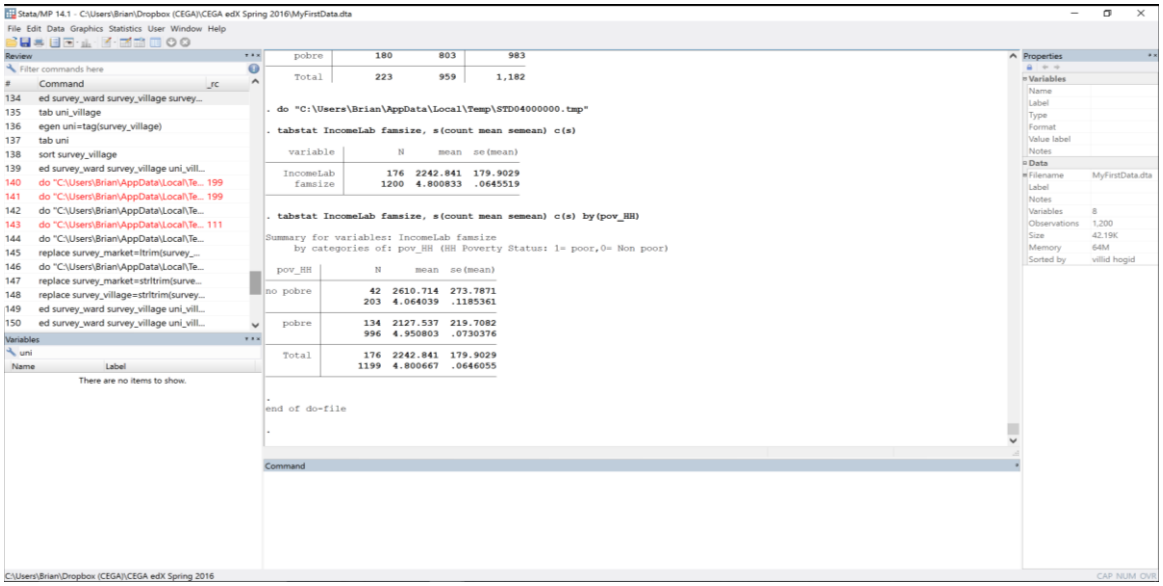


Figure 9. Example Output of TABSTAT Command

3.8 Combining Datasets

Data is often collected at multiple geographic levels–such as a city, a country, a city, a village, a household, a school, an individual, and so on—using different questionnaires. For this reason, researchers are often provided multiple datasets.

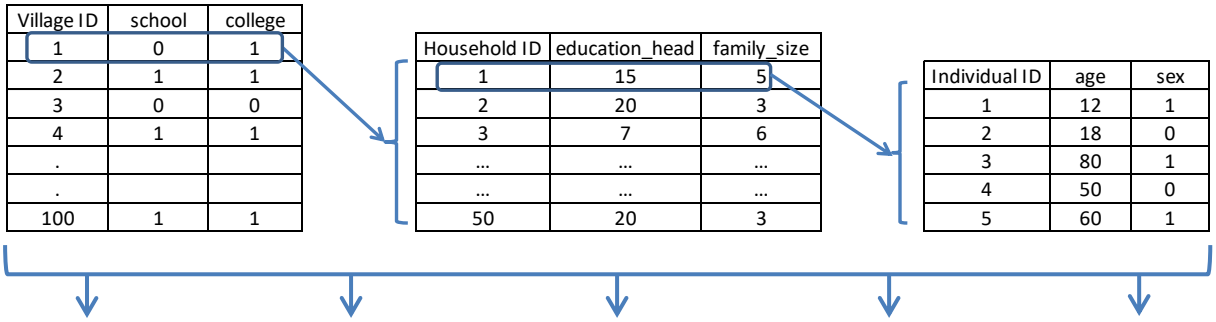
For example, in an impact evaluation project you may be provided with three datasets: (1) village-level information about the intervention (inputs, finances, infrastructure) which will be common to all households in the village; (2) household-level information such as family size, education of the head of household and number of rooms which is common to all individuals in the household; and (3) individual-level information such as sex, age and education. Suppose you want to estimate the following regression model,

$$education_{ijk} = \beta_0 + \beta_1 age_{ijk} + \beta_2 sex_{ijk} + \beta_3 education\_head_{ij} + \beta_4 family\_size_{ij} + \beta_5 school_i + \beta_6 college_i$$

Where *i* indicates the villageof individual *k* in household *j*. In order to estimate this model in STATA, you must merge the three datasets. Figure 10 presents a schematic of how merging three datasets is accomplished.

The STATA command for merging two datasets is *merge*; we have to merge two datasets at a time. *merge* joins corresponding observations from the dataset currently in memory (called the master dataset) with those from *filename.dta* (called the using dataset). *merge* can perform the following types of merges (or "joins" in the lingo of the relational database).

- ✓ One-to-one merge on specified key variables: One observation in the master dataset is matched with one observation in the using dataset. The command is specified as:
  `merge 1:1 varlist using filename [, options]`
- ✓ Many-to-one merge on specified key variables: One or more observation in the master dataset are matched with one observation in the using dataset. The command is specified as:
  `merge m:1 varlist using filename [, options]`
- ✓ One-to-many merge on specified key variables: One observation in the master dataset is matched with one or more observations in the using dataset. The command is specified as:
  `merge 1:m varlist using filename [, options]`
- ✓ Many-to-many merge on specified key variables: One or more observations in the master dataset are matched with one or more observations in the using dataset. The command is specified as:
  `merge m:m varlist using filename [, options]`



Village ID	school	college
1	0	1
2	1	1
3	0	0
4	1	1
.		
.		
100	1	1

Household ID	education_head	family_size
1	15	5
2	20	3
3	7	6
…	…	…
…	…	…
50	20	3

Individual ID	age	sex
1	12	1
2	18	0
3	80	1
4	50	0
5	60	1

Village ID	Household ID	Individual ID	age	sex	education_head	family_size	school	college
1	1	1	12	1	15	5	0	1
1	1	2	18	0	15	5	0	1
1	1	3	80	1	15	5	0	1
1	1	4	50	0	15	5	0	1
1	1	5	60	1	15	5	0	1
1	2	1	15	0	20	3	0	1
1	2	2	45	0	20	3	0	1
1	2	3	50	1	20	3	0	1
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
100	1	1	10	0	18	4	1	1
100	1	2	14	1	18	4	1	1
100	1	3	44	1	18	4	1	1
100	1	4	40	0	18	4	1	1
100	…	…	…	…	…	…	1	1
100	45	1	50	1	8	3	1	1
100	45	2	48	0	8	3	1	1
100	45	3	16	0	8	3	1	1

Figure 10. Schematic of Merging Datasets

In all above variants of *merge* command, *varlist* identifies the variables that "identify" the observations to be merged, like surname or village. In one-to-one merge, the *varlist* identifies one unique observation in the master and using datasets. In many-to-one merge, the *varlist* identified multiple observations in the master dataset but a single/unique observation in the using dataset. Many-to-many and one-to-many merges have similar interpretation for observations identified by *varlist*.

There are several useful options that user can specify in the *merge* command, and we encourage you to read the *merge* help file.

To understand how the merge command works, let us go through the steps in merging together:

- ✓ `use MyFirstData.dta`
- ✓ `sort villid`
- ✓ `merge m:1 villid using "[FOLDER_PATH]\VillageData.dta"`
- ✓ `browse`
- ✓ `drop if _merge == 2`

We sort the varlist on which we wish to merge the two dataset. We also know that there are many observations in the master data (which is a household-level dataset) which (obviously) share the same village ID (*villid*), but the village level data (using dataset) will have only one observation per village (villid) so that the merge will be many-to-one (m:1).

Figure 11 presents the output of the *merge* command, in which the results window summarizes the merge results. We see that all 1200 observations in the master dataset were merged with a village in the using dataset (_merge == 3). That is, there are no observation from the master dataset (household-level) that could not be merged with the using dataset (village-level) (_merge == 1).
On other hand, there are 674 villages in the using data which could not be merged with the household-level master data. This is because the village-level using dataset has 698 observations (villages) whereas the household-level master data contains data from only 24 villages.
When you browse the data you will find that several variables from the VillageData.dta are now merged with the master data, but there are several observations where the original variables in the master dataset are blank; why? The merge command has also appended the data from using dataset (VillageData.dta) which it could not merge, but flagged the new entries with the variable _merge. _merge is equal to 2 for all such unmatched records and we should delete them as shown in step 5 (type "help merge" for more details).

Exercise: Drop the created variable _merge (so that next merge command can create a variable _merge; else you will get an error message that _merge is already defined). Remember that the data in memory is still a household level dataset. Merge this dataset with the dataset IndividualData.dta. What type of merge is this: Many-to-one, one-to-many, or many-to-many? Read the help file and explore the option which can help you keep only those records where _merge == 3 or drop those records where _m ==2. Count the number of observations in your new dataset; has the number of observations increased? Is the dataset in memory now a household-level dataset or an individual-level dataset?

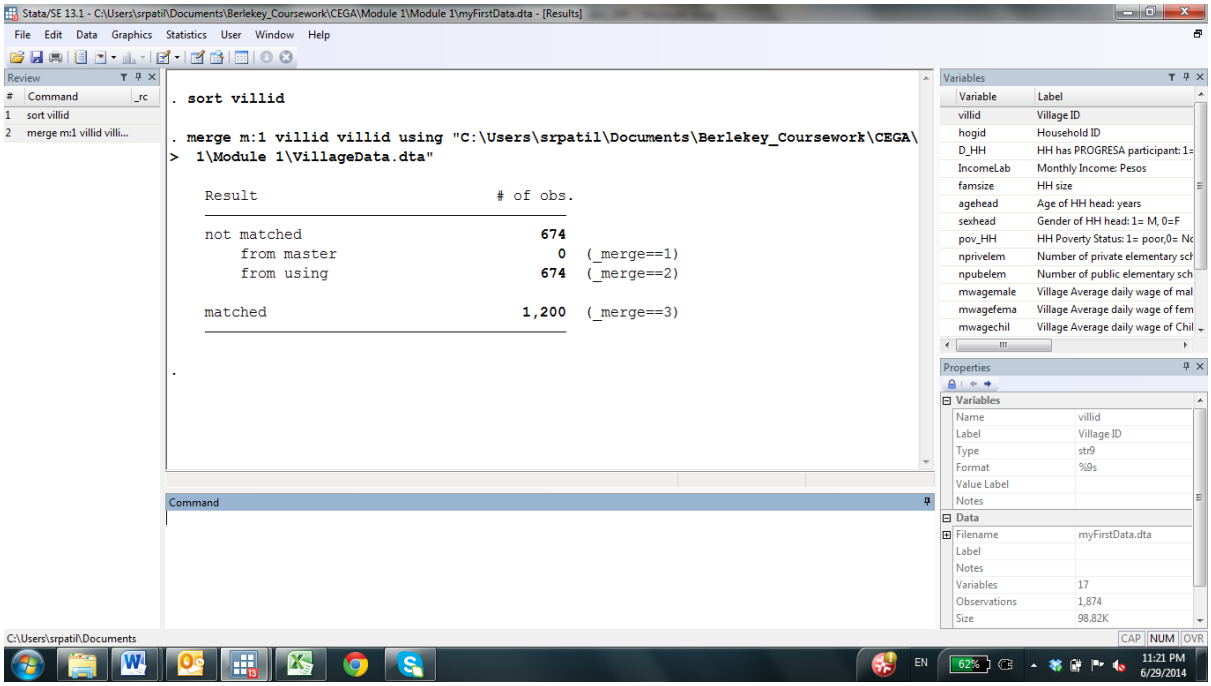


Figure 11. Output of MERGE Command

3.9 Append Datasets

Sometimes you need to append two or more dataset (while merge adds new variables, append adds new observations). The *append* command needs that both datasets have at least a subset of variables that are common.

Let's work through an example to understand the use of `append`:

- ✓ `Use MyFirstData.dta`
- ✓ `append using "[FOLDER_PATH]\AppendData.dta"`

You will find that 446 new observations are added to the end of MyFirstData.dta. If the using data had fewer variables – say, the variable *sexhead* was not present – then in the appended dataset *sexhead* variable is missing / blank for the newly added 446 observations.

Tips:

- ✓ STATA stores variables in different data types, like byte, integer, long, float and string. The size of the dataset (memory) depends on these variable types, so it is best to use the variable types that are most "efficient" for given variables. Therefore, for a large dataset or dataset where several new variables are created, using the command `compress` can be useful, but it can take several minutes to complete.
- ✓ If you need to break an ongoing command for any reason, press ctrl + pause/break on your key pad to break the execution or click the red 'Break' button at the top of the screen.
- ✓ STATA commands are always written in lower-case letters. All STATA variables and commands are letter case sensitive.

4. VISUALIZING THE DATA

Visual representations of data often help researcher form hypotheses, identify problems in the data distribution, select appropriate analysis methods, and present the results in easier and more appealing formats. STATA has excellent capabilities to produce figures, charts or tables, but it could take you significant time before you learn to apply them. Here, we only introduce the basics.

For this part of the lesson, please load MyFirstData.dta.

4.1 Histograms

A histogram is a graphical bar representation of a variable, in which the height of each bar is proportional to the frequency with which the values represented by that bar appear in the data. A typical syntax of histogram you may use is,

```
histogram varname [if] [in] [,] [bin(#)] [density
(default)|frequency | percent ] [kdensity] [normal]
[title("customized")] [xtitle("customized")] [ytitle("customized")]
```

The help file will provide you with many more options to customize the look of your graph. Figure 12 presents the output for a simple histogram command (a) and customized command (b) as follows. We find that family size does not exhibit a normal distribution, but instead has a long right tail:

- ✓ `histogram famsize`
- ✓ `histogram famsize, bin(10) percent normal title("Distribution of Family Size") xtitle("Number of Individuals") ytitle("percentage")`

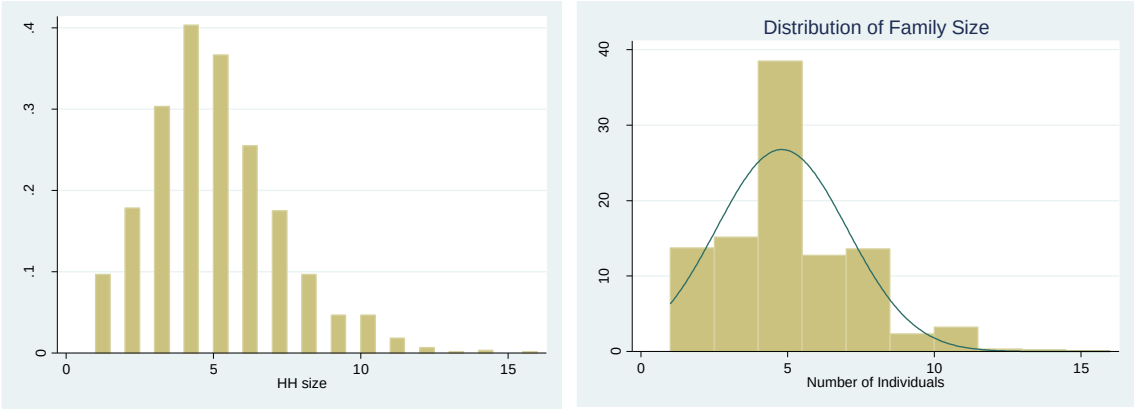


Figure 12. Output of HISTOGRAM Command

4.2 Kernel Densities

Kernel densities are an alternative and popular way of visualizing variables' distributions. The main advantage of kernel densities over histograms is that rather than giving equal weight to each bar, the kernel gives more weight to data that are close to the point of reference and gives you a smoother line plot. The usual syntax for kdensity is,

- ✓ `kdensity varname [if] [in] [,][bwidth(#)] [normal] [title("customized")] [xtitle("customized")] [ytitle("customized")]`

Figure 13 presents the output from the following *kdensity* command. The left hand figure is with the default bandwidth that STATA estimates, while the right hand figure is with customized bandwidth of 1000 as per the following command.

- ✓ `kdensity IncomeLab if pov_HH == 1, bwidth(1000) normal title("Distribution of Income for Poor Households") xtitle("Monthly Income") ytitle("Kernel Density")`

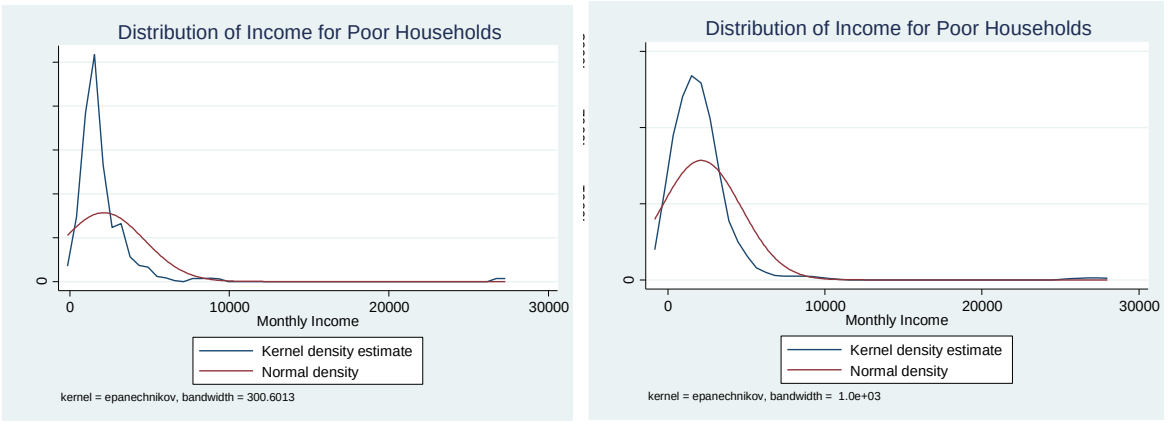


Figure 13. Output of KDENSITY command

The kernel density plot becomes smoother as the bandwidth increases, but the fit to the data is poorer compared to the smaller bandwidth.

4.3 Scatter Plots

Scatter plots depict the correlation between two variables. The basic syntax for scatter plot is:

```
scatter varname1 varname2 [if] [if] [,] [title("customized")]
[xtitle("customized")] [ytitle("customized")]
```

Scatter plotting, histogram, and all other graphs are part of a higher-level command called `twoway`. We will not attempt to fully explain the general 'twoway' command in this module, but note that it provides substantial flexibility in customizing the look of your graphs and in combining graphs of different types.

5. BASIC REGRESSION ANALYSIS

At its core, regression analysis is a tool that estimates the relationship between an outcome (or dependent) variable and a set of predictor variables (or independent variables, also referred to as covariates). Although the covariates can be transformed by log, power or other non-linear transformation, the regression model is itself linear and additive as follows.

$$Y_i = \beta_0 + \beta_1 X_i + \sum_j (\beta_j K_j)_i + \varepsilon_i$$

The coefficient β_1 indicates the associated change in output *Y* for a unit change in input *X* when all other variables *K* are held constants. To provide unbiased estimates of the β coefficients, called marginal effects in econometrics, several key regression model assumptions must hold true. When these assumptions are not initially met, more complex regression models and other methods may have to be used.

In these cases, applying regression analysis calls for considerable skills and theory-based knowledge. In this module, we only demonstrate how regressions models are specified in STATA and explain the output.

The basic command for regression models in STATA is as follows,

regress depvar [indepvars] [if] [in] [, options]

When we don't specify any *indepvars*, *regress* returns a linear regression with a constant term as the only covariate, and the coefficient can be interpreted as the sample mean of *depvar*. When we specify *indepvar* as a single binary variable, the regression analysis is similar to a t-test and β_1 is the difference in the mean of *depvar* between the two groups categorized by *indepvar;* the significance of the coefficient yields results similar to a t-test.

Figure 14 is a screenshot of the following regression command.

reg IncomeLab D_HH famsize agehead sexhead

Source	SS	df	MS		
Model	61568330.7	4	15392082.7	Number of obs =	174
Residual	934150145	169	5527515.65	F(4, 169) =	2.78
				Prob > F =	0.0283
Total	995718476	173	5755598.13	R-squared =	0.0618
				Adj R-squared =	0.0396
				Root MSE =	2351.1

IncomeLab	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
D_HH	-1190.092	414.8902	-2.87	0.005	-2009.127	-371.0569
famsize	130.87	80.31821	1.63	0.105	-27.68622	289.4262
agehead	-10.34949	12.19094	-0.85	0.397	-34.41563	13.71666
sexhead	-.7971899	432.048	-0.00	0.999	-853.7034	852.109
_cons	3018.288	774.3589	3.90	0.000	1489.626	4546.95

Figure 14. Output of REGRESSION command

The output of regression command uses standard notations found in any econometric or statistics text book. Although R^2 is important in regression models used for predictions, the most relevant output for our statistical inference is:

- ✓ As a rule of thumb, "Prob > f" should be 0.05 or lower. This result implies that an F-test that tests joint significance of all model coefficients is statistically significantly different from a zero vector at the five percent level.
- ✓ The three columns "Std Err", "t" and "P>|t|" denote the standard error of the coefficient, the t-value for testing the significance against the null hypothesis that the coefficient is 0, and p-value is the statistical significance level for a two sided/tailed t-test.

After each regression, it is possible to save the main results using the command estimate store and then assigning a name to the saved data for each regression.

We then can recover the saved results in a single table, but we will not expand on how to do this here. Interested readers can refer to STATA help.

The command regress also provides additional results that are not displayed in the regression results. One way to see and access them is by using the command ereturn list. Y can be predicted using the estimated regression model by using the command predict after running regress.

6. HYPOTHESIS TESTING

Hypothesis testing is a fundamental part of any statistical analysis. We specify a null hypothesis, which specifies a state of the world that we would like to see if our data provides evidence against, and an alternative hypothesis. For example, the null hypothesis in context of an impact evaluation could be:

H0: The intervention has no effect on household income levels
H1: The intervention changes the household income levels

The null hypothesis is either rejected or not rejected based on statistical significance tests. Please note, we never "accept" the alternative hypothesis in statistical parlance.

Student t-test is one of the oldest most powerful statistical tests to assess whether the difference in two means is statistically significant (different from 0). In fact, if you are able to design a randomized control trial and achieve both perfect covariate baseline balance and perfect compliance with your study design, then a t-test (or its variants) is all that you may need to estimate the impacts of the trial. In STATA, the t-test syntax is:

- ✓ One-sample t test: ttest *varname* == # [if] [in] [, level(#)]
- ✓ Two-sample t test using groups: ttest *varname* [if] [in], by(groupvar)
- ✓ Two-sample t test using variables: ttest varname1 == varname2 [if] [in], unpaired
- ✓ Paired t test: ttest varname1 == varname2 [if] [in] [, level(#)]

Figure 15 is the output of the following t-test command:

ttest IncomeLab, by(D_HH)

```
. ttest IncomeLab, by(D_HH)

Two-sample t test with equal variances
```

Group	Obs	Mean	Std. Err.	Std. Dev.	[95% Conf.	Interval]
0	47	3044.894	607.9791	4168.095	1821.095	4268.692
1	127	1945.906	98.45947	1109.582	1751.057	2140.754
combined	174	2242.759	181.874	2399.083	1883.781	2601.736
diff		1098.988	402.1601		305.1836	1892.793

```
    diff = mean(0) - mean(1)                                  t =   2.7327
Ho: diff = 0                     degrees of freedom =      172

    Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
 Pr(T < t) = 0.9965         Pr(|T| > |t|) = 0.0069          Pr(T > t) = 0.0035
```

Figure 15. Output of TTEST command

The row "diff" lists the difference in the mean of the two variables of interest and the standard error of this difference. Dividing the mean difference by its standard error, we obtain the provided t-statistic. STATA compares this t statistics with the standard student's t distribution and provides the results for three alternative hypothesis as listed, two one-sided tests and the two-sided test.

7. ORGANIZING EMPIRICAL WORK

There is no single way to organize an empirical project, so no general rules are available. However, some heuristics can be helpful when one is working on a complex project. We propose the following set of general rules based on our previous experience running impact evaluations:

- ✓ Organize folders on your computer in a clear manner. Take special care with folders containing databases, do-files and estimation results. Keep the original names of the original databases; store them by source and year for easy econometric handling. Keep survey documentation (survey manuals, questionnaires, and other guides) in the same folders for quick reference.
- ✓ Record all your command routines in DO files. Applied research is usually characterized by multiple DO files. Therefore, it is recommended to use a master DO file to record and document the general structure of the set of other DO files.
- ✓ Always protect the original data and perform any analysis or modification on a saved copy.
- ✓ Create folders with names like 'boneyard', 'dump', 'achieved' or similar names where you store earlier versions of DO file or customized data files. Only keep most recent/final

versions in the main folder. Version-control software like *git* is helpful to keep track of changes to your project over time.

- ✓ Protect your final database. It is recommended to use the command datasignature to do so. This command assigns a signature to the database, so one can be sure that the same dataset is used at a later date.

8. BIBLIOGRAPHY/FURTHER READING

1. http://www.STATA-press.com/
2. Baum, Cristopher (2006). An Introduction to Modern Econometrics using STATA. STATA Press.
3. Cameron, A. Colin and Pravin Trivedi (2009). Micro-econometrics using STATA. STATA Press.
4. Baum (2006) offers a standard introduction to most of the standard econometric methods. Cameron and Trivedi (2009) is more ambitious and covers more advanced micro-econometric techniques in a detailed manner. More advanced readers may also find useful the following book:
5. Baum, Cristopher (2009). An Introduction to Programing in STATA. STATA Press.
6. Mitchell, Michael (2012). A Visual Guide to STATA Graphics. Third Edition. STATA Press.
7. http://www.STATA-journal.com/