An Introduction to Stata:
A statistical package for managing, analyzing, & graphing data

Theresa A Scott, MS
Department of Biostatistics
Vanderbilt School of Medicine
theresa.scott@vanderbilt.edu
http://biostat.mc.vanderbilt.edu/TheresaScott

OBJECTIVE:

The objective of this lecture is to become familiar with the basics of Stata.

→ *NOTE:* We will not be "analyzing" data in any way except for generating some basic data descriptive statistics (eg, mean & SD for continuous variables; raw & relative frequencies for categorical variables).

INTERACTING WITH Stata:

Stata can be used either as a point-and-click application (ie, a graphical user interface, GUI) or as a command-driven package.

- GUI provides an easy interface for those new to Stata (ie, do not have to memorize command syntax).
- Command language provides a fast way to communicate with Stata & to communicate more complex ideas (ie, not limited by menu selections).
- Command syntax also has the advantage of reproducibility.
  - If you have an analysis you need to modify often & run repeatedly, you can submit commands in a "batch mode" via *do-files* (covered later).
- Many of the commands are short & easy to execute, & as proficiency grows, it is often faster to type commands rather than use the menu.

→ *NOTE:* We will be only covering the command syntax in these notes.

Stata LAYOUT:

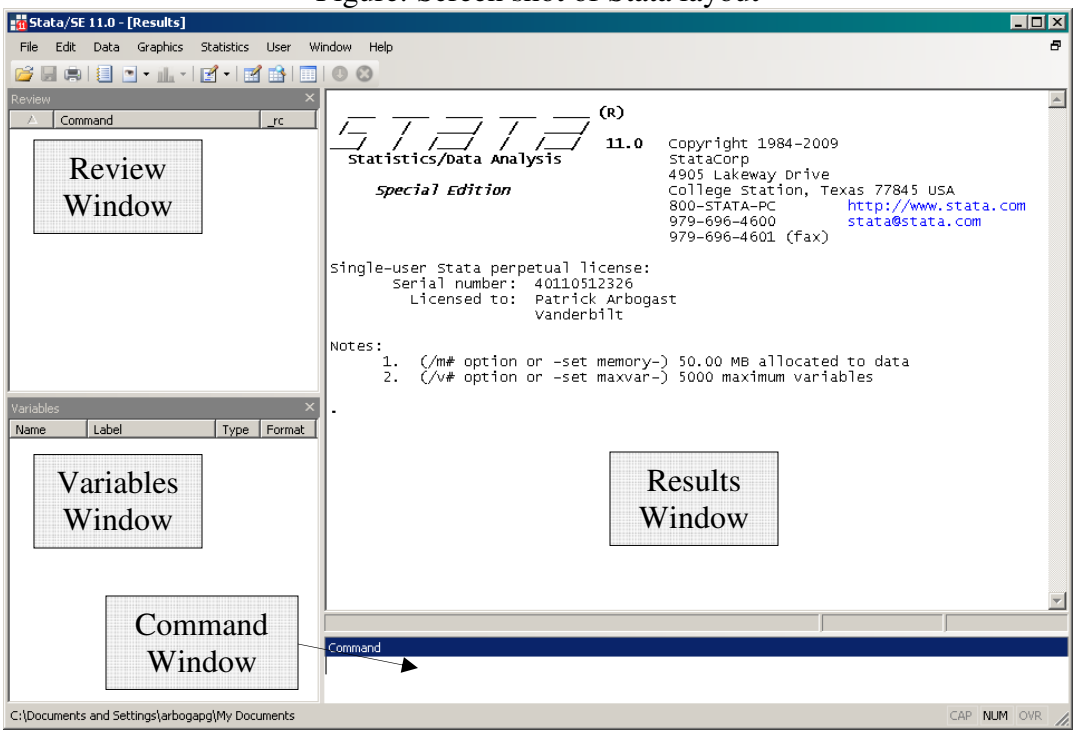
Stata has ***5*** windows (see image on next page):

- **Command** window: where commands are entered – press the *Enter* key on your keyboard to execute any command.
  → *NOTE:* All commands & variables are *case sensitive* (eg, "age" vs "Age").
- **Results** window: where results appear (ie, are printed).
- **Variables** window: lists the variables in the current dataset.
- **Review** window: lists past (ie, already executed) commands.
- **Graph** window: where graphs are displayed (appears when graphs are generated).

Clicking on a past command in the *Review* window brings it into the *Command* window where it can be modified & executed. Can also hit the *Page Up* button on your keyboard to bring up a previously executed command; can then be modified & executed.

→ *NOTE:* In these notes, all Stata syntax (eg, commands & variables) is typed as **`bold Courier New`** font.

Figure: Screen shot of Stata layout



THE "FABULOUS FIFTIES":

Stata has a lot of commands (around 500 depending on how you count them). Putting aside the statistical commands that might be of interest to you, the following table contains 50 some odd commands everyone should know.

→ *NOTE:* All of the listed Stata commands in following table will not be covered in these notes. However, we will cover a portion of them by way of a *sample Stata session* (see below).

Table: 50 some odd "essential" Stata commands.

Category	Stata Command	Functionality
Operating-system interface	**pwd**	Show the file-path of the current directory (stands for "*p*rint *w*orking *d*irectory")
	cd	Change directory
	sysdir	Query & set system directories
	mkdir	Create a (new) directory
	dir / ls	Show files in the current directory
	erase	Erase a disk file
	copy	Copy a file from disk or URL
	type	Display the contents of a file
Inputting data	**input**	Enter data from the keyboard
	infile	Read unformatted ASCII (text) data
	infix	Read ASCII (text) data in fixed format
	insheet	Read ASCII (text) data created by a spreadsheet
	use	Load a Stata-format dataset (ie, a Stata **.dta** file)
	clear	Clear the entire dataset & everything else

Table, *cont'd*

Data manipulation	**edit**	List & edit data using the *Data Editor*
	generate	Create a new variable (ie, column)
	replace	Replace one of a variable's values with another value
	egen	Extended generate – has special functions that can be used when creating a new variable
	recode	Recode the values of a categorical variable
	rename	Rename a variable
	drop	Drop variables and/or observations (ie, rows)
	keep	Keep variables and/or observations
	sort	Sort observations in a dataset
	encode	Create the numeric version of a string variable
	decode	Create the string version of a numeric variable
	order	Change the order of the variables in a dataset
	by	Repeat a Stata command on subsets of the dataset
	reshape	Convert a dataset from "wide" to "long" format & vice versa
	append	Append datasets
	merge	Merge datasets
	compress	Compress data in memory
	save	Save the dataset currently in memory as a Stata file (**.dta**)
Formatting	**format**	Specify the format to display variables
	label	Manipulate (define, assign, list, & drop) variable & value labels
Basic data reporting	**describe**	Describe the contents of the dataset in memory or on disk
	codebook	Show detailed contents of a dataset
	inspect	Display a simple summary of the dataset's attributes
	list	List values of variables
	browse	List data using the *Data Editor*
	count	Count observations satisfying specified condition(s)
	assert	Verify the truth of a claim (ie, that an expression is true)
	summarize	Calculate & display univariate summary statistics
	tabulate	Generate one- & two-way frequency tables
	tabstat	Calculate & display a table of summary statistics – summary statistics for a series of numeric variables, possibly broken down by another (categorical) variable; can specify the list of statistics to display
	table	Calculate & display a table of summary statistics – can break down each variable (numeric or categorical) by up to 3 categorical variables; can specify the list of statistics to display
Keeping track of your work	**log**	Create a *log file*
	notes	Apply notes to the dataset in memory
Getting on-line help	**help**	Display online help for a specific Stata command
	search	Search Stata documentation for a keyword
The Internet & updating Stata	**update**	Update Stata
	net	Install user-written additions from the net
	ado	Manage user-written additions from the net
	news	Report Stata news

SAMPLE Stata SESSION:

To demonstrate a portion of the Stata commands mentioned in the previous table, let's use an example dataset – a random subset of the well known *Primary Biliary Cirrhosis* data set. The full data set contains the data from the Mayo Clinic's trial on primary biliary cirrhosis (PBC) of the liver conducted between 1974 & 1984. Specifically, the trial was a randomized placebo controlled trial of the drug D-penicillamine. The random subset of the PBC data set that we will be using has been saved as a Stata data format file (**`pbc.dta`**), which is available on my website (given on the first page). The file contains N = 100 observations (rows) & the following variables (columns):

Table: Description of the motivating PBC dataset

Variable Name	Variable Description
`id`	Case number
`fudays`	Number of days between registration & the earlier of death, transplantion, or study analysis time in July, 1986
`status`	Status, where 0 = Censored, 1 = Censored due to liver treatment, & 2 = Death
`drug`	Treatment, where 1 = D-penicillamine & 2 = Placebo
`age`	Age in days
`sex`	Gender, where 1 = Female & 2 = Male
`ascites`	Presence of ascites, where 0 = No & 1 = Yes
`bili`	Serum bilirubin in mg/dl
`chol`	Serum cholesterol in mg/dl
`album`	Albumin in gm/dl
`stage`	Histological stage of disease

You'll notice that all of the categorical variables in our dataset (**`status`**, **`drug`**, **`sex`**, **`ascites`**, & **`stage`**) are coded with numeric values. Even though Stata can handle reading in character (string) values, such as "No" & "Yes", certain commands will have issues with such data. In the long run, it's safer (and saner) to code everything numerically & then define any categorical variables appropriately once the data has been read into Stata.

Before going further, let's discuss *some good programming practices*:

1. Create a directory (folder) where you will keep all your data, code, and output related to a particular project. This can be thought of as your `working directory' whenever you use Stata for that particular project.
   → Create a folder on your desktop named **`PBC_Analysis`** and save the **`pbc.dta`** data file to this folder.
2. Conduct your Stata session from within the relevant working directory. To do this, change to the directory *before* doing anything else in your Stata session (eg, before reading in a dataset). This will allow you to easily reference the names of your data files without having to include long path names (i.e., **`C:/MyDocuments/MyProjects/ProjectName/.../filename.dta`**). You can change directories using the **`cd`** command.
   → Use the **`cd`** command to move to the created **`PBC_Analysis`** folder. On a Windows machine, the command would look something like **`cd "C:\Desktop\PBC_Analysis"`**

Now, let's open a *log file*. Even though the contents of the *Results* window can be printed, only the most recent output is printed when the output becomes very long. Luckily, Stata can record your session into a *log file*, but it does not start a log automatically; you must tell Stata to record your session. To start a log, we execute **`log using filename`**. By default, the resulting log file contains what you type & what Stata produces in response, recorded in a format called *Stata Markup & Control Language* (*SMCL*; has a **`.smcl`** extension).

SMCL format is nice for viewing & printing within Stata. You can also create an *unformatted* log file, which is simpler to use if you plan to insert & edit the output in a Word Processor, by executing **set logtype text** before opening your log file. An unformatted log file is saved with a **.log** file extension. For our sample Stata session, let's open a (default) SMCL log file:

```
log using pbclog, replace
```

You'll notice that I also specified the **replace** option, which overwrites the log file if it already exists. If you do not specify **replace** & the log file already exists, you will receive an error message. You'll also notice that I *did not* specify any *file pathname* when I specified the file name. This is because I already changed to the directory where my data files are located & where I wish to save my log file.

Now, let's read in our PBC data set. The easiest data file to read in is a Stata data format **.dta** file:

```
use pbc.dta, clear
```

The **clear** option specifies that it is okay to replace any data in memory. NOTE, Stata only stores one dataset in memory at a time.

Often, our dataset is not saved as a Stata .dta file, but as a Microsft Excel file. To read in such a file, we first save the file as comma-delimited (**.csv**) or tab-delimited (**.txt**) ASCII (text) file using the *Save as type…* drop-box within the *Save As* dialog box of the *File* drop-menu. We can then read in the text file using the **insheet** command:

```
insheet using pbc.csv, comma clear
```

As with the **use** command, we can specify the **clear** option. We also specified the **comma** option, which tells Stata the columns are comma-separated.

Irregardless of how you read in your data, Stata works with a copy of the data that it loads into *memory*. By default, Stata/IMP & Stata/SE allocate 10 megabytes & Stata/IC allocates 1 megabyte to Stata's data areas. Most often, this is enough memory, but sometimes it is not. Luckily, you can easily increase the amount of memory allocated to Stata by your operating system while Stata is running. You can do this by executing **set memory #[B|K|M|G]**, where **#** is the amount of memory specified in bytes, kilobytes, megabytes, or gigabytes – **B**, **K**, **M**, or **G** is typed to distinguish the unit. For example, to set your memory to 1G execute **set memory 1G**.

With that said, let's get back to our motivating data set. It is always a good idea to explore your dataset after you read it into Stata to (1) make sure it was read in correctly, & (2) get a better understanding of each variable. To do so, we can first use the **describe** command, which shows us the basic information about our dataset – the number of observations, the number of variables, the names of the variables, & more. Next, the **codebook** command is a great tool for getting a quick overview of the variables in the dataset. Among other things, for each variable it prints the type (eg, numeric), range, number of unique values, & number of missing values.

```
describe
codebook
```

You probably noticed that when you executed the **codebook** command, you were given a **-more-** "prompt". Seeing **-more-** at the bottom of the Results window is Stata's way of telling you that it has something more to

show you, but showing you that something more will cause the information on the screen to scroll off. When you do see **-more-** at the bottom of the *Results* window,

Press …	and Stata …
letter *l* or Enter	displays the next line
letter *q*	Acts as if you pressed *Break*
space bar or any other key	displays the next screen

You can also press the "clear **-more-** condition button", the button labeled *Go* with a circle around it. Lastly, it is possible to "turn off" the printing of any additional output in your *do-file* – simplify specify **set more off.** This is especially useful when you will be generating a log file of the desired output & you are not interested in interactively watching the output.

Once a dataset is read into memory, we often wish to view (ie, list) all or a subset of the observations and variables. In Stata, we can use the **browse** and **list** commands to do so. Specifically, the **browse** command displays your dataset in a spreadsheet-style *Data Editor* window. In contrast, the **list** command uses the *Results* window to displays the data. Typed by itself, **list** lists all the observations and variables in the dataset. You can also execute **list *varlist***, which lists only those variables specified in ***varlist***. You can also specify one or both of **in *range*** and **if *exp***, which limit the observations listed – more to come regarding **if** statements. Some examples follow the **browse** command.

```
browse
list
list id drug sex
list in 1/10
list if sex == 1
list id drug if sex == 1
```

The **edit** command is also available, which not only lists the full dataset using the *Data Editor*, but also allows you to modify specific cell values and save the changes.

Let's now discuss how to create new variables for the dataset in memory using the **generate** command. To define a new *continuous* variable, we simply use a **generate *newvar = expression*** construct. For example, the **age** variable in our PBC dataset has been collected in *days*. It would be more convenient to use age in *years*. We also create a new variable representing follow-up in years.

```
generate ageyrs = age/365.25
generate fuyrs = fudays/365.25
```

The **generate** command can also incorporate **if** statements, which will conditionally define the new variable. This is a good place to discuss how Stata defines *missing values*. In general, a missing value in Stata is represented with a period, "**.**". However, missing values of *numeric* variables are represented by *large positive values*. Therefore, the expression **age > 60** is true if the variable age is greater than 60 or <u>*is missing*</u>! To exclude missing values from consideration, ask whether the value is less than "**.**". For example, going back to the list command: **list if age > 60 & age < .** . You can also use **!missing(*varname*)** – eg, **list if age > 60 & !missing(age)**.

We can also use the **generate** command to create a new categorical variable, but the steps are slightly different. Specifically, we must give the new categorical variable an initial value when we create it. We then use the **replace** command to overwrite the initial value for a new value in each sub-group of values. NOTE,

we use the **replace** command because it is used to modify the contents of an *existing variable*. Lastly, we often use **if** statements in our **replace** expression in order to properly break-down (ie, condition) the observations into appropriate groups. As an example, let's create a new **censored** variable, which will be the (collapsed) 2-level version of the (3-level) **status** variable:

```
generate censored = .
replace censored = 1 if status == 0 | status == 1
replace censored = 2 if status == 2
```

You'll notice that I specified the initial value of the newly created censored variable to be missing. This is quite convenient when you might have missing values in your dataset. You'll also notice that I used **==** rather than **=** to specify equality. There are a few other conditional operators to know about: **<**, **<=**, **>**, **>=**, **!=** (for "not equal to"), **&** (and), and **|** (or).

Now that we've added a few more variables to our dataset, let's look at the revised describe output.

```
describe
```

You'll notice that all of the original variables have a "*Label*", whereas the three new variables do not. Luckily, it is easy to add a label (up to 80 characters) to a variable using the **label** command. Specifically, we use a **label variable *varname "label"*** construct. As examples, let's add variable labels to the three newly created variables in our dataset:

```
label variable ageyrs "Age (yrs)"
label variable fuyrs "Follow-up (yrs)"
label variable censored "2-level survival status"
```

We'll see these new variable labels listed if we look at the revised describe output using the **describe** command.

We can also use the **label** command to define & apply (character string) labels to the (numeric) values of a (categorical) variable. For example, for the **sex** variable, we can label the values of 1 as "Female" and the values of 2 as "Male". These *value labels* will be then be displayed in subsequent output pertaining to the **sex** variable. This is much more useful than constantly having to remember what 1 and 2 represent. We first *define* the label, which associates each value label with each value. We then *assign* the defined label to the appropriate variable. As an example let's add the value labels to the **drug** variable. Recall, a value of 1 represents the "D-penicillamine" group, while a value of 2 represents the "Placebo" group.

```
label define druglabel 1 "D-penicillamine" 2 "Placebo"
label value drug druglabel
```

It should be noted that the name of each defined label must be unique – that is, you will get an error if you try to re-define a label of the same name. However, you can assign an already defined label to several variables.

It should also be noted that the commands given in this sample Stata session have been saved to a *do-file* – specifically, **pbc.do**, which is available on my website. The do-file contains additional expressions that define & apply value labels to the remaining categorical variables in the dataset (**status**, **censored**, **sex**, **ascites**, & **stage**). It is useful at this point in the lecture notes to save this do-file to your computer and to execute the additional **label define** and **label value** expressions. After executing the commands, let's once again look at (1) the codebook command & (2) the first 10 observations of our dataset.

```
codebook
list in 1/10
```

Now that we have our dataset revised and all of the variables appropriately defined, let's generate some summary (descriptive) statistics. Let's first use the **summarize** command to calculate & display a variety of (univariate) summary statistics. Executing the **summarize** command (with nothing else specified), causes Stata to calculate summary statistics on *all* variables in the dataset, which might be inappropriate (eg, calculating the mean on categorical variable). So, instead, we specify the (continuous) variables we would like to summarize. For example

```
summarize fudays fuyrs age ageyrs bili chol album
```

You'll notice from the output that, by default, the **summarize** command calculates & displays the number of non-missing values, the mean, the standard deviation (SD), minimum, and maximum value for each variable. Alternatively, we can specify the **detail** option (ie, **summarize fuyrs ageyrs, detail**), which calculates & displays a series of percentiles (1, 5, 10, 25, 50, 75, 90, 95, and 99), the 4 smallest and 4 largest values, the number of non-missing values, the mean, the SD, the variance, the skewness, and the kurtosis for each variable. The **summarize** command can also incorporate **if** statements.

We have a lot more options for calculating & displaying a table of summary statistics with the **tabstat** command. Specifically, there is a **by** option, which allows you to specify that the statistics be displayed separately for each unique value of a (categorical) grouping variable. There is also a **statistics** option, which allows you to specify the statistics to be displayed. The following are some examples.

```
tabstat ageyrs, by(sex) statistics(count mean sd p25 median p75) missing
tabstat ageyrs, by(drug) statistics(count mean sd p25 median p75) missing
```

Let's now use the tabulate command to generate one- and two-way frequency (ie, contingency) tables of categorical variables. Here are a few examples:

```
tabulate sex
tabulate drug, missing
tabulate sex drug, missing column
```

By default missing values are not counted in the generated table. However, specifying the **missing** option causes missing values to be counted in the table as their own category. There are also additional options that modify how the table is printed. For instance, specifying **column** reports the column relative frequencies (ie, column proportions) in addition to the raw frequencies (counts).

Stata can also create a 3-way contingency table using the **table** command. The first variable specified generates the rows of the table, the second the columns, and the third the "super" (ie, grouped) columns.

```
table stage sex drug, missing
```

Lastly, let's close our log file.

```
log close
```

MAKING Stata STOP WHAT IT IS DOING:

When you want to make Stata stop what it is doing & return you to the command prompt, press the *Break* button (the button with the big red X on the toolbar below the drop-menus).

DO-FILES:

Rather than typing commands at the keyboard, you can create a text file containing Stata commands & instruct Stata to execute the commands stored in that file. Such files are called *do-files* since the command that causes them to be executed is **do**. Do-files are very useful if you have to modify & rerun lengthy manipulation & analyses. Also they're useful if you have to revisit an analysis, say 12 months, after the study is completed. On a day-to-day basis, do-files allow you to "document" the commands you have executed so you don't have to remember them all.

To create a do-file: You can use any text editor (eg, WordPad) or the *Do-file Editor* within Stata (5th button in Toolbar from right).

- Open a new text file.
- Type commands. For example:

```
* generate an indicator for any previous preterm labors
generate anyptl=0
replace anyptl=1 if ptl>0
label variable anyptl "Previous preterm labors?"
label define yesnolab 0 "no" 1 "yes"
label value anyptl yesnolab
tab ptl anyptl
```

- Save the file with a **.do** extension (most often in the same directory as your data file(s)).

The Stata code demonstrated in the sample Stata session has been extracted to a do-file and is available on my website.

To execute a do-file: Type **do *filename*** at the command prompt at hit the *Enter* button on your keyboard. NOTE, if you haven't changed to the relevant working directory, the ***filename*** must specify the complete path name – eg, **"C:\analysis\project\data\analysis.do"**.

→ *NOTE:* There are several ways to include *comments* in a do-file:

- Begin the line with an asterisk (*****), which prevents a *single* line of code from executing.
- To add comments or block *multiple* lines of code from executing, start the block with **/*** & end with ***/**.

REFERENCES, RESOURCES & ONLINE HELP WITHIN Stata:

For these notes, I was lucky enough to have access to Patrick Arbogast's, PhD, "Introduction to Stata" lecture notes, which he covers in the MPH program's Biostat I class. I also heavily referenced an indispensable website for learning Stata: the UCLA Academic Technology Service's (ATS) Stata website (http://www.ats.ucla.edu/stat/stata/; specifically the class notes & learning modules covered in the *Stata Starter Kit*).

There is also a full library of Stata manuals for purchase. These include a *Getting Started* manual, the *User's Guide*, & the *Reference* manual, which are the primary tools for learning Stata. Unfortunately, most of us do not have the funds to purchase Stata manuals. Luckily, there are other sources of information, which include

- The Stata website (http://www.stata.com): Much of the site is dedicated to user support.
- The Stata Press website (http://www.stata-press.com): This site contains the datasets used throughout the Stata manuals.
- The Stata listserver: An active group of Stat users communicate over an Internet listserver, which you can join for free.

And **DON'T FORGET** Stata itself. Stata has a subject table of contents online with links to the help system & dialog boxes that make it easy to find & execute a Stata command – choose *Contents* from the *Help* drop-menu. As mentioned in the "Fabulous Fifties" table, there are also two Stata commands that are useful: **`help`** ***`command`*** and **`search`** ***`word`***. For example, you can open the help file for the **`summarize`** command by executing **`help summarize`**. NOTE, regardless of which command you use, results will be shown in a new window (by default). Also, blue text indicates a hypertext link, so you can click to go to related entries.