

## Stata Principles

Stata is [Stata Corp.](#)'s statistical software package. It accepts command-line input or input from a \*.do file. To invoke a do file without starting Stata beforehand, just double-click on it. Always use a PC with enough RAM for the dataset you want to use. To edit a do file within Stata, type *doedit* [optional file name]. To run a do file within Stata, type *do* [file name] (use *, nostop* option to keep do file running regardless of errors).

## Starting Stata

The following commands are useful for the top of a do file or an interactive session.

```
*****
* enter a short description of your program here *
* author and date of program *
*****
cd c:\data\          /* <- type in name of working data directory here */
set mem 25m          /* enter appropriately large value */
set more off        /* if 'more' is 'on' you may have to hit the space bar a lot */
set matsize 500     /* really only matters for large-scale operations */
log using filename.log, append /* use appropriate filename */
use dataset, clear /* "dataset"=name of dataset you want to use */
...
log close           /* ceases saving output to log file */
exit, STATA clear  /* shuts down operations and exits Stata */
```

## Stata Syntax

NOTE: Unlike most statistical software packages, Stata programs are case-sensitive.

### *Log Files*

- Turn a new log file on: `log using c:\working\yourlog, append`. Use the `, replace` option instead of `, append` if you want to write over anything that's currently in the named logfile.
- Temporarily turn a running logfile off: `log off`. Turn it on again: `log on`.
- Cease logging and close the logfile: `log close`.
- Here's a nice trick. Want to create a log file just of the commands you are going to enter in an interactive session, skipping all the output? You could use this to create a do-file after-the-fact for what you've done (replete with any mistakes in commands, of course). Start up a *command log* file, which works the exact same way as the `log` command, but with `cmdlog` as the necessary command.
- Did you forget to start a log file but want to see what commands you have entered? Start up a new log file and then type `#review x`, where *x* is the number of previously entered command lines you want to list.

### *Comments*

- Any line beginning with `*` is ignored by the Stata processor but will appear in the log file. You can take notes during class by inserting comments like this into your class log!
- Any section of an input line set off inside `/* */` delimiters is not executed – it's a comment, just like in the above do-file. Use comments to keep notes about what you're doing at this point in your program. But you can't use `/* */` comments in interactive mode, just in do-files.

### *Loading Up and Saving Datasets*

- Open a dataset using the `use` command, i.e., `use mydata, clear`. This wipes out whatever data you currently have in memory.

- Save a dataset like this: `save mydata, replace`. This wipes out whatever data was currently saved in the file of that name, if any. Just use a different name instead of `mydata` if you want to keep the old dataset in place!
- Sometimes, when dealing with huge datasets whose variables you already know the names of, you may want to open up only a subset of the dataset's variables. Do this as follows: use `varlist using mydata, clear`. Here `varlist` is a list of variables separated by spaces.

### *Simple Information & Memory Management*

- To find out what exactly is in a dataset you have loaded up, use the `describe` (or `des`) command.
- More extensive information can be obtained with the `codebook` or `inspect` commands.
- Use memory to find out how much memory is used by the dataset you have loaded up, and how much memory remains for use by Stata.
- If you are running into the upper bound of memory, you will need to save your work, type `clear all` to wipe the working memory clear, type `set mem 100m` (or some appropriately high number), and then load up your dataset again. You may need to assign even more memory than 100 megabytes for Stata's use, depending on the size of the dataset file you want to use. (You can find out how big it is by using the Windows Explorer and view-details in the appropriate folder window.) And you effectively won't be able to use a file bigger than the amount of RAM installed in the PC you are using. Here's your excuse for applying for a grant!
- While you await your grant money and your new PC with 10 times as much RAM, you can still do a few things to open up and work with your very large datasets. Use the `compress` command to squish each variable down to the lowest memory unit that still can hold all its significant digits. Use the `use varlist using mydata, clear` command (see above) to open up just those variables that you need from a larger dataset.

### *Viewing a Dataset in Detail*

- To see your dataset in Stata's spreadsheet window, type `browse` (and list specific variables after it if you want to see only a subset of the variables).
- To edit particular cells in the dataset by hand, type `edit`, and "preserve" your changes afterwards.
- To list the contents of a dataset into the logfile, type `list`, and list specific variables in the order you want them listed. You can use the `if` command with `list`. You may want to sort `varlist` before listing the contents, to see the values in ascending order of one or more variables. Check out `help gsort` for information on sorting in descending order.

### *Abbreviations & Help*

- Most Stata commands can be abbreviated, e.g., `summarize` can equally be inputted as `su`.
- To find out what the appropriate abbreviation is, take a guess, or look it up. Type `help commandname`, where `commandname` is the name of the command you want to abbreviate. The highlighted characters at the start of the command name at the top of the helpfile indicate what the abbreviation is.
- The helpfile on a particular command also spells out the exact syntax to be used, all the options available, a useful description of what the command actually *does*, and links to related commands. Use `help` frequently.
- You can also access help by clicking on the Help command on the drop-down menubar.

### *Conditions*

- In order to execute an operation on just a *subset* of the data, given that certain conditions are met, use the `if` option.
- `if variable=x /*` where `x` is another variable, expression, or particular number `*/`
- "Not equals" is written `~=`; "less than or equal to" is `<=`; "greater than or equal to" is `>=`.

- A numeric variable is missing when it equals a period, i.e., `==.`; a text variable is missing when it equals blank, i.e., `==" "`.
- You may want to combine conditions, e.g., the equivalent of saying, “do this if *a* equals a particular value *and* if *b* equals a particular value.” “And” is typed as `&` and “or” is `|` (perhaps found on your backslash key). Remember, “and” means “only if both conditions are true,” further restricting the size of the subset you are working with. “Or” means “if either condition is true,” which may expand the size of the subset you are dealing with.
- You can execute most commands on *multiple* subsets of the data in one command, with the `by` option. Simply preface your command line with `by var1: command var2. . .`. You have to sort by `var1`, the subset-defining variable, before doing this.

### Variable Types

- A variable can be stored in Stata memory in a variety of different ways: see `help datatypes`.
- Of the numeric types of variables, most will be either *ints* (integers, i.e., no decimals) or *floats* (decimals allowed). *floats* have only 7 significant digits; if you want more, the variable must be type *double*. A *double* uses more memory than a *float*, which in turn uses more memory than an *int*.
- If you are not sure what you will need for a given variable, use a larger data type and then compress.
- To see what a variable’s type is, `describe varname`.
- To set the type of the variables you are about to create, `set type vartype`, where `vartype` is `int`, `float`, or `double`, for example.
- To change an existing variable to a different type, you can (1) recast `newtype varlist`, where `newtype` is a datatype name and `varlist` is a list of variable names to be changed; use the `, force` option to force the change at the cost of possibly changing the variable(s)’s values accordingly; (2) `replace yourvar=value`, where `value` is an expression that would yield a non-integer result for at least one observation; or, (3) `set type appropriately` and then generate `newvar=oldvar`, where `newvar` is the new variable name, and `oldvar` is the name of the variable with the original datatype.
- “String” variables are text representations instead of simply numbers. Define the datatype as *string* like so: `set type strx`, where `x` is the maximum number of characters in the string.

### Creating a New Variable as a Function of Others

- To create a new variable, use the `generate` (or `gen`) command, such as `gen newvar=expression`. It can be used with `if`. If you want to change certain values of this variable, `replace newvar=newexpression if. . .`.
- Provide a description for your new variable by labeling it: `label var newvar "Description"`.
- Having defined variable type as a string, you can change a numeric variable to a string using `gen newstring=string(oldvar)`. Change a string to a number using `gen newvar=real(stringvar)`. Various functions exist to pick out selected pieces of a string variable (perhaps for later processing into a numeric variable): see `help functions`.
- If you want to create multiple dummy variables out of one categorical variable, one dummy for each value of the original variable, use `xi: command i.origvar`, for just about any command. This is useful for creating “fixed effects” on the fly, for example.

### Recoding Variables

- To recode particular values of a variable into new values in a single command (rather than use `replace` multiple times), use the `recode` command, following the syntax described in `help recode`.
- Some imported datasets use particular numeric values to represent observations of a variable that are actually supposed to be missing. Usually the codebook will tell you which values mean this. To convert such actual values into missing values, as appropriate, type `mvdecode varlist, mv(num)`, where `varlist` is the list of variables to be recoded, and `num` is the numeric value in the original

dataset which represents a true missing value. Use `_all` for `varlist` if you want to do this simultaneously to all variables.

### Labels

- In Stata, you can create or change three important text characteristics of any kind of variable, string or numeric: variable names, variable *description* labels, and variable *value* labels.
- Each variable (in Stata 6.0) can have up to an eight-character name (not starting with a number). You can change a variable's name with the *rename* command, e.g., `rename oldname newname`.
- Variable descriptions are the text in the second column in the variable window, or what shows up after the variable name when you type *describe*. Change this with the *label* command, e.g., `label var varname "Your description here, in quotes"`.
- Variable value labels are text descriptions that apply to each separate value a variable takes on, e.g., for a dummy variable, "Male" if it equals 0, "Female" if it equals 1. Assigning value labels is a two step process. First, create a bundle with the appropriate labels in it. Second, assign that bundle of labels to the variable in question. (You could create a bundle and assign the same one to many variables.) The steps are: `label define bundle value1 "label 1" value2 "label 2"`, etc. `bundle` here is the name for the label bundle you are making. `value1 "label 1"` is a pair containing the first value you want to assign a label to, e.g., 0, and its label, the latter in quotes. After making the bundle, assign it with the command `label value varname bundle`, where `varname` is the name of the variable whose values should be labeled, and `bundle` is the name of the label bundle you just made with the `label define` command.
- After labeling a variable's values, its labels will be displayed when you view the dataset or in any Stata output using that variable. But it retains its actual (numeric, if appropriate) values, which you can see with the *nolabel* option after the comma for most commands. E.g., `tab varname, nolabel`.