Stata Basics

1. Interacting with Stata – Command window and menu bar commands (*interactive* mode); do-file editor (*batch* mode); Review, Variables, Properties, and Results windows. Paging up/down in the Command window; double-clicking commands in the Review window.

2. Updating Stata – Stata will regularly check for updates, and you may be prompted to update to the latest version. To do so, type the following:

```
update all
```

3. Getting help: syntax

```
help
help keyword
help regress
help summarize
```

4. "Using Stata as a calculator" – in the command window (note: `sqrt` is one of Stata's many functions.)

```
display 2 + 2
display sqrt(49) + 10
```

   Abbreviations: Stata will accept abbreviated commands. You can use as few letters as you like, so long as the abbreviation remains unique to one command. If you use abbreviations, I recommend using at least 3 letters in the abbreviated command.

```
di 2 + 2
dis sqrt(49) + 10
```

   Common mathematical operators include `+, -, *, and /`. The exponent symbol is `^`, for example: `di 7^2`. For longer expressions, use parentheses to make the intended order of operations clear. For example: `di (7+2)*4`. Type `help functions` to see a complete list of functions and operations.

   The Expression Builder window is useful for constructing more complicated calculations that make use of operations, functions, saved results, and more. This is found under Data → Other Utilities → Hand Calculator → Create.

   You may also use `display` for text expressions, for example:

```
display "hello"
```

5. Opening a data file in Stata format (.dta file)

```
use C:\data\myfilename.dta
use http://www.stata-press.com/data/r9/census2
```

Note that file names with spaces must be in quotes. I generally use quotation marks whether they are required or not.

```
use "C:\data\my file name.dta"
```

The `clear` option tells Stata to remove any dataset already in memory and replace it with the specified one. Use this with caution, since unsaved changes will be lost. Most Stata commands have options that can be specified at the end of the line, following a comma:

```
use C:\data\myfilename.dta, clear
```

Stata has some built-in datasets you can use to learn Stata. Type `help dta_examples` to see a list. You can open the sample datasets directly using `sysuse`:

```
sysuse auto.dta
```

6. Saving a Stata data file. If the file already exists and you want to replace it with the version currently in memory, use the `replace` option. The same rule for quotes applies as in #5.

```
save "C:\data\my file name.dta"
save "C:\data\anotherfile.dta", replace
```

7. To close a dataset (remove it from memory) use the command `clear`. This takes it out of memory, but does *not* delete it from your disk. As with any other file, don't "clear" the data from memory if you have changes that need to be saved.

8. Reviewing the contents of a dataset – use the Variables and Properties windows, the Variables manager (Data → Variables Manager), and/or these commands:

```
browse              (allows you to view the data like a spreadsheet)
edit                (allows you to edit the data like a spreadsheet—not advised)
list                (lists entire dataset in the Results window—not advised)
list in 1/10        (lists observations 1-10 only)
browse var1 var2    (view only the variables var1 and var2 like a spreadsheet)
list var1 var2      (lists only the variables var1 and var2)

describe            (basic information about variables—like a table of contents)
describe varnames   (basic information about specific variables)
desc, short         (example of abbreviated command name and option)

codebook varname        (to get a little more information about a variable)
inspect varname         (to get a little more information about a variable)
count                   (to see the number of observations)
count if condition      (to see the number of obs. where a condition is true)
```

Note that Stata is case-sensitive! You must type variable names exactly as they appear in the Variables window.

Numeric variables are stored in different ways, depending on the size and precision of values contained in the variable. The integer types (in order of size) are: *byte, int,* and *long*. The real number types (in order of precision) are: *float* and *double*.

9. Simple descriptive statistics, tables, and frequency distributions

```
summarize
sum varnames
sum varnames, detail

tabulate varname          (this command has many options)
table varname             (this command has many options)
```

10. `if` and `in` clauses – The `if` clause tells Stata to execute a command only for cases where the `if` clause applies. The `in` clause tells Stata to execute the command only for cases with the specified range of observation numbers. Use the latter with caution, since the sort order can change. Examples:

```
summarize varname if varname2=="Y"
summarize varname if varname2~="N"
sum varname if varname2>=500
list varlist if _n < 40   (_n refers to the observation number)
```

Note equality conditions require two equals signs. An inequality condition ("not equal to") requires ~= or !=. Conditions can include > or < signs, as well as greater-than-or-equal-to (>=) and less-than-or-equal-to (<=) signs.

You can also include AND or OR conditions in a statement, where AND is designated by a "&" and OR is designated by a "|". Examples:

```
sum enroll if state=="NY" | state=="NJ" | state=="CT"
sum puptch if state=="NY" & enroll>250
```

Be very careful when specifying the logic of a complex "if" clause. Use parentheses when needed. For example, the following two statements will not yield the same result:

```
sum enroll if state=="NY" | state=="NJ" & enroll>1000
sum enroll if (state=="NY" | state=="NJ") & enroll>1000
```

Stata has special notation for missing values that depend on whether the variable is *numeric* or a *string*. If numeric, missing values are represented by a period. If string, missing values are represented by a pair of double quotes, as shown below.

```
count if numvar1==.
summ var2 if numvar1~=.
list var1 var2 if strvar1==""
```

11. Designating the order of variables listed in the Variables window – very useful when your dataset contains a large number of variables. You can also sort the viewing order of variables (temporarily) in the Variables window.

```
order varlist
aorder                          (alphabetizes list of variables)
```

The order of variables in the Variables window is important when you wish to use *variable lists* as shorthand. For example, if variables `var1` through `var5` are sorted in alphabetical order in the Variables window, then the following two commands are equivalent:

```
sum var1-var5
sum var1 var2 var3 var4 var5
```

Stata refers to `var1-var5` as a "varlist". The syntax indicates to Stata that it is a list of variables. Another type of varlist uses the *wildcard* symbol, e.g.: `sum var*` Here, you're telling Stata to execute the command for all variables starting with "var".

12. Dropping variables and cases from the dataset (or *keeping* specific variables/cases). E.g.:

```
drop varlist                    (drops specified variables from the dataset)
drop if condition               (drops cases that meet a condition)

keep varlist                    (keeps only specified variables)
keep if condition               (keeps only cases that meet a condition)
```

Dropping *variables* is like deleting columns in a spreadsheet. Dropping *observations/cases* is like deleting rows in a spreadsheet.

13. `.do`-files are programs—a set of instructions—that Stata will carry out when you Run it. These are created in the Do-File Editor. Some useful commands often found in .do files:

```
cd C:\mydirectory\              (set the working directory)
cd "C:\Data\Class"

* Comments in do file          (single line of commented text)
// Comments in do file         (single line of commented text)
/* Comments */                 (multiple lines of commented text)

version                         (displays the current version of Stata)
version 10                      (requires Stata be version 10+ to proceed)
clear all                       (removes everything from memory)
set more off                    (prevents Stata from pausing display)
```

```
set linesize 80          (sets display width – avoids awkward wrap)

# delimit ;              (set the line delimiter as a semicolon if your
                          commands will extend for multiple lines)
///                      (alternatively, use this at the end of the line if
                          your command extends to the next line)
# delimit cr             (reset the line delimiter to be a hard return)
```

It is possible to copy and paste commands from the Review window into the .do file. This allows you to "test" your command to see if it does what you want it to do, before including it. Do-files can be executed all at once, or in subsections (highlight the section you want to run, and click on the Execute (do) button).

14. Mac versus PC: the most notable difference between Stata on the Mac (vs PC) is how the working directories are set. PC uses backslashes: `cd "C:\Users\Sean"` while Mac uses forward slashes: `cd "/Users/Sean".`

    On a PC, the command `cd Sean` will return an error message. On a Mac, the command `cd Sean` will set the working directory to the subfolder titled "Sean" in your current working directory.

    Note the command `pwd` displays the current working directory (both PC and Mac). On the PC, the command `cd` (without arguments) also does this. However, on the Mac, `cd` (without arguments) when you are in a subfolder changes the working directory one level up.

    The following code in a .do file will help you move between Mac and PC:

```
if regexm(c(os),"Mac")==1 {
   local mypath= "/Users/Sean"
   }
else if regexm(c(os),"Windows")==1 local mypath=
"\Users\Sean "
cd "`mypath'"
```

15. Creating and viewing log files (in .txt or Stata markup language format .smcl)

```
log using C:\data\yourfilename.txt, text
log using C:\data\yourfilename.smcl, smcl
log close                          (stop logging)
log off                            (temporarily pause logging)
log on                             (resume logging)
view C:\data\yourfilename.txt      (view log in viewer)
capture log close                  (can be placed at the top of a do-file
                                    to ensure no log file is currently open)
cmdlog using filename              (logs only commands, not output)
cmdlog close                       (stop logging)
```

16. Creating new variables – note expressions can include arithmetic operations (+, -, *, /), mathematical functions, statistical functions, random number functions, string functions, date functions, and other (type `help functions` for a complete list, or use the Expression Builder mentioned earlier).

```
generate  newvar = expression
gen       aplusb = vara + varb
```

When creating complicated expressions, be sure to pay attention to order of operations, as mentioned earlier.

17. Replacing values of existing variables – you can replace values of existing variables using the `replace` command. For example, the first command below will divide all existing values of *var1* by 100. The second example will replace the value of *state* with `NY` whenever *state* is equal to `New York`.

```
replace var1 = var1/100
replace state = "NY" if state=="New York"
```

18. Labeling variables – can also be done in Variables Manager or Properties window.

```
label var varname "Descriptive label here"
```

19. Renaming variables and attaching notes

```
rename oldvarname newvarname
```

Some researchers prefer not to rename variables (which could risk losing information about the original variable name). Rather, a copy of the old variable can be made. It is also possible to attach notes to variables:

```
notes var1: Insert note here   (specify a note for var1)
notes var1                     (view the note for var1)
```

20. Stata treats missing values (".") as the *largest positive number*, so use caution when using the > operator when there are missing values. For example, the following would code `elderly` as "1" in cases where `age` is missing:

```
gen elderly = 0 if age<65
replace elderly = 1 if age>=65
```

21. Sorting data

```
sort varlist          (sorts in ascending order by varlist)
gsort -varname        (sorts in descending order by varname)
```

22. Using "by" groups

```
sort var1
by var1: sum var2
```

In the second statement above, the command `sum var2` will be executed separately for each unique value of *var1*. Therefore, this approach should only be used when there are a manageable number of values/categories for *var1*.

23. User-written commands (`.ado` files). "SSC" is Statistical Software Components, at Boston College. Examples:

```
help nmissing
ssc install nmissing        (installs .ado file called "nmissing")
```

24. The wildcard symbol (*) allows you to apply a command to numerous variables with the same prefix or suffix. For example:

```
summarize gdp19*
drop pop*
```

Use the wildcard with caution, to avoid doing things to variables you did not intend to do.

25. Changing memory allocated to Stata – important when using large-scale databases in older versions of Stata.

```
memory                      (get a report on memory settings)
set mem 500m                (set memory to 500 MB)
set matsize 400             (set maximum number of model vars)
```

Note: in recent versions of Stata it is usually not necessary to change the memory settings.

26. Debugging do-files – Stata provides (often uninformative) error numbers that can be looked up. With a little experience, it becomes easy to catch errors. The most common bugs are errors in syntax:

- Not using correct symbols in condition statement (e.g. double equals sign)
- Misspelling of command name and/or variable names
- Capitalizing letters in variable names that shouldn't be capitalized, or not capitalizing letters when they should be
- Spaces where none should be
- Incorrect options
- Options in the wrong place
- Variables are the wrong type for the desired command (e.g. string vs. numeric)

Some strategies for de-bugging:

- Try running the program in steps (sections of the do-file)
- Try running the program on different data
- Use tracing (use `set trace on`)
- Restart and/or update Stata (use `update all`)