Stata Files

There are four main types of Stata files:

.do files -> txt files with your commands saved, for future reference and editing
.log files -> txt files with the output from your .do files, for future reference and printing
.dta files -> data files in Stata format
.gph files -> graph files in Stata format

Other types of Stata files:

.ado files -> programs in Stata (every command you use has an associated .ado file on your computer)
.smcl files -> default format for log files unless you tell Stata to use .log extension

The Basics

While you can use Stata interactively, most work is done in .do files to aid in (i) editing mistakes and (ii) saving your work for future reference.

Typical structure of a .do file:

```
capture log close
clear
set more off
set mem 20m
log using filename.log, replace

.
.
.

log close
```

-> everything between the log using line and the log close line will be saved in the .log file
-> -capture- in front of a command line tells Stata to ignore the line if there is an error associated with it
-> -replace- in the log using line tells Stata to overwrite the log file if it exists… be sure this is what you want
-> -set mem XXXm- allocates memory to Stata. If you get an error message "insufficient memory," then more memory needs to be allocated until your program works, or your computer breaks

-> lines that begin with an "*" are ignored by Stata; these are useful for making comments to yourself
-> by default, Stata assumes that each line in the .do file is a separate command; alternatively, you can tell Stata to interpret a ";" as the end of each command line… this is done by typing:

```
#delimit ;        **turns on the ; option

.
.
.

#delimit cr       **turns the ; option off
```

.do files can be written using the do file editor within Stata, or using any other text editor of your liking

.do files can be executed within Stata by typing in the command window:

do *filename*

or, by opening the do file in the do file editor and clicking the "do current file" button

or, by clicking on "Do…" in the "File" menu within Stata and clicking on the .do file

-> .log files can be opened by clicking on the file in your directory (will use notepad by default) or in any text editor

Help Files

Stata has a built-in help window, which is a stripped-down version of the manual.

Relevant commands:

help *commandname* -> *commandname* must be the correct name of a Stata command
search *keyword* -> executes a keyword search for relevant commands in official Stata
findit *keyword* -> executes a keyword search for relevant commands in official Stata and in unofficial Stata online; if you find a command, you can then follow the relevant clicks to download it; it will placed automatically in a directory that Stata searches and it will act just like any "official" Stata command once installed. USER BEWARE!

Data Basics

Here is an example .do file to open a Stata data set in Stata, and do some basic manipulations

```
capture log close
clear
set more off
set mem 20m
log using filename.log, replace

use datafilename, clear

*list the data for viewing
list

*list certain variables
list varlist

*view the data in an excel-like environment
edit

*display a description of the data set
describe

*display summary statistics
summarize

*display detailed summary statistics of select variables
summarize varlist, detail

*display correlation matrix of select variables
corr varlist

*display covariance matrix of select variables
corr varlist, cov

*display frequency breakdown of a "sufficiently discrete" variable
tab varname

*display cross-tabulation two "sufficiently discrete" variables
tab varname1 varname2

*generate new variable in data set
generate newvarname = log(varname)

*delete a variable(s) from the data set
drop varlist

*instead of dropping all variables not needed, you can list the ones you want to
*keep, and the remainder are deleted
keep varlist

*rearrange the order of the observations (e.g., arrange a time series data set in
order by year)
sort varname

*save changes to the data set (be careful overwriting the existing data set!)
save datafilename, replace
```

```
      *or,
      save newdatafilename, replace

      log close

Notes:
   -> when generating new variables, Stata has many built-in functions (see -help
   functions-)… log(), ln(), abs(), sin(), cos(), exp(), int(), etc.
   -> see -help egen- for more complex commands for generating variables
```

Limiters

Many Stata commands may be executed on a sub-set of observations if desired. This is accomplishing using "if" statements.

Examples:

```
summarize varlist if varname==[expression]
summarize varlist if (varname1==[expression] & varname2==[expression])  *& "and"
summarize varlist if (varname1==[expression] | varname2==[expression])  *| "or"
summarize varlist if varname!=[expression]                              *!= "not equal"
summarize varlist if varname>[expression]
summarize varlist if varname>=[expression]

generate white=1 if race==1   *all obs with race!=1 have white coded as missing "."
replace white=0 if race!=1    *use replace to change the values of an existing var
generate black=1 if race==2
replace black=0 if race!=2
```

is equivalent to

```
generate white=race==1        *this is a shorthand way of generating dummy vars
generate black=race==2        *indicating if the expression after the "=" is true
```

The -by- command is also extremely useful and accomplishes some of the same things.

Example:

```
summarize varname if white==1
summarize varname if white==0
```

is equivalent to

```
sort white
by white: summarize varname
```

is equivalent to

```
bys white: summarize varname
```

Note: With -by- the data need to be sorted by the relevant variable first. -bys- is short for -bysort-, which sorts the data as needed, instead of having to do this yourself in an extra line of code.

Another useful basic command for exploring data used with -if- is -count-. This simply counts the number of observations meeting a certain criteria.

Example:

```
count if white==1
```

Stored Results

After many Stata commands, results are temporarily stored in "locals". These are very useful for reference to, but are over-written as soon as another command is executed. So, to keep them for reference later in your .do file, they should be stored in locals for which you give a name to.

To see what results are saved after a particular command, consult the actual manual (they are not listed in the help files within Stata). Or, type

```
return list
```

or,

```
ereturn list
```

after a command has been executed to see what is stored.

Using locals…

Example:

```
summarize varname
return list
local m=r(mean)
local sd=r(sd)

*stores the mean in the local m, which will exist until a new .do file is run, or
*Stata is exited, or it is over-written by yourself

generate newvarname = (varname - `m')/`sd'

*this generates a new var which is standardized to have zero mean, unit variance
*note the `.' single quotes used around locals when using them!
```

Example:

```
count if white==1
local n=r(N)
display "The number of whites in the sample is " `n'

*this counts the number and then uses the -display- command to write out a
*statement with the result
```

Locals can also be used to store repeatedly types phrases to save you time, and prevent typos. For example, suppose you are performing repeated summary statistics on the same four variables.

Example:

```
local mylocal "varname1 varname2 varname3 varname4"
summarize `mylocal' if white==1
summarize `mylocal' if black==1
summarize `mylocal' if otherrace==1
```

Basically, where you type `mylocal', Stata just sees what you defined `mylocal' to represent.

Loops

It will often save time to perform repeated operations within a loop. For example, in the above example, instead of using three different race variables (white, black, otherrace), you could do the following:

Example:

```
generate race=1 if white==1
replace race=2 if black==1
replace race=3 if otherrace==1
forvalues i=1/3 {
      summarize varname if race==`i'
      local m`i'=r(mean)
}
```

Note the use of `i' as a local as well. `m1', `m2', and `m3' will store the means for later use if desired.

Loops can be nested…

Example:

```
.
.
.
forvalues i=1/3 {
      forvalues j=0/1 {
            summarize varname if race==`i' & gender==`j'
            local m`i'`j'=r(mean)
      }
}
```

Loops can be performed over lists using –foreach-…

Example:

```
local mylocal "white black otherrace"
foreach var in `mylocal' {
      summarize varname if `var'==1
      local m`i'=r(mean)
}
```

Basic Econometrics

OLS…

Example:

```
regress depvar varlist, [options]
```

Coefficients are stored in a matrix, e(b), and the var-cov matrix of the coefficients in e(V). However, individual coefficients can be used or stored in locals as follows…

Example:

```
regress depvar x1 x2 x3
forvalues i=1/3 {
      local b`i'=_b[x`i']
}
local alpha=_b[_cons]
```

Commands for conducting tests of linear and non-linear hypotheses are
-test-
-testparm-
-testnl-

Estimation commands in Stata contain many post-estimation commands. Most relevant are commands for predicting fitted values of the dependent variable (i.e., y-hat) or the residuals (i.e., e-hat)

Example:

```
regress y x1 x2 x3
predict yhat, xb        *generates a new var, yhat, equal to x*beta-hat
predict e, res          *generates a new var, e, equal to y-(x*beta-hat)
```

Note: predict commands work not only on the estimation sample, but also out-of-sample observations (if there are any).

Example:

```
regress y x1 x2 x3 if white==1
predict yhat, xb
predict e, res
```

Here, only whites are used in the regression, but the values of beta-hat are used to generate predictions/forecasts for the entire data set. If you don't want this, use the following:

Example:

```
regress y x1 x2 x3 if white==1
predict yhat if e(sample), xb       *e(sample) is how Stata marks the sample used
predict e if e(sample), res
```

Other basic estimation commands frequently used by economists
-qreg-
-ivregress-
-probit-, -oprobit-, -mprobit-
-logit-, -ologit-, -mlogit-
-tobit-
-poisson-, -nbreg-
xt commands for panel data

Categorical Variables

Stata has a few options for creating dummy variables from categorical variables. The options differ in whether the new variables become part of the existing data set or are created only temporarily in Stata.

```
*Create new dummy variables in the data set
tab varname, gen(newvarname)
*Create new dummy variables in the data set as well as executes the command
xi: regress wages i.varname
*Stata 11 option: suppress the "xi" and new dummies are created only temporarily
regress wages i.varname
*Stata 11 option: suppress the "xi" and interactions only are created
regress wages varname1#varname2
*Stata 11 option: suppress the "xi" and new dummies and interactions are created
regress wages varname1##varname2
```

Making Tables

Stata has many time-saving commands that are useful for formatting results in a way that facilitates making tables of results or summary statistics in TeX or in Excel.

Regression tables…

Example:

```
loc x1 "varname1"
loc x2 " `x1' varname2"
loc x3 " `x2' varname3"

forval r=1/3 {
      qui reg y `x`r''
      estimates store m`r'
}
#delimit ;
estout m1 m2 m3, cells(b(star fmt(3)) se(nopar fmt(3))) style(fixed) stats(N,
   fmt(%9.0g)) starlevels(♯ 0.10 † 0.05 * 0.01) legend title("Titlename")
#delimit cr
```

Other relevant commands…

- -tabstat-
- -outtex-
- -est2tex-
- -sutex-
- -textab-
- -xml_tab-

Graphs

Stata has lots of graphics capabilities. Need to look at the manual for complex stuff. See help files within Stata for options.

Basic examples:

Histogram

```
hist varname, bin(number) frac title("Graph Title") saving(graphfilename, replace)
```

Scatter plot

```
scatter varname1 varname2, title("Graph Title") saving(graphfilename, replace)
```

Regression line

```
regress y x
predict yhat, xb
#delimit ;
tw connected yhat x, sort lp(solid) ms(x) title("Graph Title")
   saving(graphfilename, replace);
#delimit cr

*lp() is an option that stands for linepattern
*ms() is an option that stands for markerstyle
```

Combining two or more graphs into a single graph

```
regress y x
predict yhat, xb
#delimit ;
tw connected yhat x, sort lp(solid) ms(i) || scatter y x, title("Graph Title")
   saving(graphfilename, replace);
#delimit cr

*This will graph a scatter plot of the data and the regression line in one graph
```

Simulating Data

Stata has some random number generators that allow you to simulate your own data. For our purposes, we will focus on generating random draws from a uniform(0,1) distribution and a normal distribution.

Example:

```
set seed 12345                      *seed implies every time the .do file is run,
                                    *Stata will generate the same data set
set obs 1000                        *data set will have 1000 observations
generate u=runiform()               *u~U[0,1]
generate x=rnormal()                *x~N(0,1)
generate x=rnormal(2,3)             *x~N(2,9)
```

Let's put together some of the tools thus far. An example program that simulates 100 data sets of 1000 observations each, where the data has the following structure:

$y = 1 + 2x + e$
$x \sim N(0,1)$
$e \sim N(0,1)$

then regresses y on x using OLS for each data set, and finally computes the MAE (mean absolute deviation) of the 100 estimates of beta-hat relative to the 'true' value of 2

Example:

```
set seed 12345
loc nsets=100
loc obs=1000
loc mae=0                           *need to define this prior to the loop
forval i=1/`nsets' {
      clear
      set obs `obs'
      g x=rnormal()
      g y=1+2*x+rnormal()
      reg y x
      loc b=_b[x]
      loc mae=`mae' + (abs(`b'-2))/`nsets'
}
di in green "MAE (beta) = " in yellow %7.3g `mae'
```

Note: Try cutting and pasting the above code into the do file editor and running it, to verify you know how and why it works.

Suppressing Output

The above program writes out all 100 regressions on your screen/log file. We can suppress the output of commands we desire using the -quietly- option. You can do this individually, for each line, such as

```
set seed 1234567890
loc nsets=100
loc obs=1000
loc mae=0
forval i=1/`nsets' {
      clear
      set obs `obs'
      g x=rnormal()
      g y=1+2*x+rnormal()
      qui reg y x
      loc b=_b[x]
      loc mae=`mae' + (abs(`b'-2))/`sets'
}
di in green "MAE (beta) = " in yellow %7.3g `mae'
```

or you can put a -qui- "loop" around a series of commands, such as

```
set seed 1234567890
loc nsets=100
loc obs=1000
loc mae=0
qui {
      forval i=1/`nsets' {
            clear
            set obs `obs'
            g x=rnormal()
            g y=1+2*x+rnormal()
            reg y x
            loc b=_b[x]
            loc mae=`mae' + (abs(`b'-2))/`sets'
      }
}
di in green "MAE (beta) = " in yellow %7.3g `mae'
```

finally, you can force Stata to display the output from one line within the -qui- "loop" using the -noisily- option, such as

```
set seed 1234567890
loc nsets=100
loc obs=1000
loc mae=0
qui {
      forval i=1/`nsets' {
            noi di "Program is on data set #" `i'
            clear
            set obs `obs'
            g x=rnormal()
            g y=1+2*x+rnormal()
            reg y x
            loc b=_b[x]
            loc mae=`mae' + (abs(`b'-2))/`sets'
      }
}
di in green "MAE (beta) = " in yellow %7.3g `mae'
```

Note: Be careful suppressing output with -qui- since you may hide mistakes!

Creating a Data Set of Simulation Results

Let's say you want to simulate a bunch of data sets, estimate a model on each data set, and create a new data set where each observation corresponds to a data set and the variables correspond to parameter estimates from that data set. For instance, this would be useful to then analyze the empirical distribution of the sample estimates.

Example:

```
set seed 1234567890
loc nsets=100
loc obs=1000
************************************************
*create an empty data that will store the results*
************************************************
set obs `obs'
*_n corresponds to the observation #, so dataid will go from 1 to `obs'
generate dataid=_n
sort dataid
*save the data set
save outputfilename, replace
********************
*perform simulation*
********************
qui {
forval i=1/`nsets' {
      clear
      set obs `obs'
      g x=rnormal()
      g y=1+2*x+rnormal()
      reg y x
      g alpha=_b[_cons]
      g beta=_b[x]
      g dataid=`i'
      keep dataid alpha beta
      keep if _n==1
      sort dataid
      merge dataid using outputfilename
      drop _merge
      sort dataid
      save outputfilename, replace
}
}
use outputfilename, clear
su alpha beta, d
hist beta, bin(10) normal frac title("Graph Title") saving(graphfilename, replace)
```

Notes:

-> Try cutting and pasting the above code into the do file editor and running it, to verify you know how and why it works.

-> We introduced a new command –merge-. This merges two data sets together, as long as there is a common observation id variable(s) with which to link observations. In the above example, we create the variable *dataid* for merging purposes. Stata creates a new variable, called _merge, after merging data sets. It takes on 3 values (typically): 3 – for obs that merged perfectly (i.e., were in both data sets), 2 – for obs that were in the "using" data set, but not the data in memory at the time, and 1 – for obs that were in the data in memory at the time, but not in the "using" data set.

-> -append- is similar to –merge-, but instead of combining observations across two data sets, the new data set is stuck "at the bottom" of the original data set.

Bootstrap

Nonparametric bootstrap entails drawing a new sample *with replacement* from the data set in memory in Stata, estimating the model on the bootstrap sample and keeping track of the result, and repeating this *B* times. There are two ways of doing this in Stata: (i) use the –bootstrap- command; (ii) do it "manually" using the -bsample- command.

For example, suppose we simulate a data set as we did above (y=1+2x+e). We then want to generate 50 bootstrap samples, regress y on x in each sample, and create a new data set with the 50 estimates of alpha and beta.

Example #1:

```
set seed 1234567890
set obs 1000
g x=rnormal()
g y=1+2*x+rnormal()
bootstrap _b, reps(50) saving(outputfilename, replace): reg y x
use outputfilename, clear
su alpha beta, d
hist beta, bin(10) normal frac title("Graph Title") saving(graphfilename, replace)
```

Example #2:

```
set seed 1234567890
loc reps=50
set obs `reps'
g bootid=_n
sort bootid
save outputfilename, replace
clear
set obs 1000
g x=rnormal()
g y=1+2*x+rnormal()
forval b=1/`reps' {
      preserve
      bsample
      reg y x
      g alpha=_b[_cons]
      g beta=_b[x]
      g booted=`b'
      merge dataid using outputfilename
      drop _merge
      sort dataid
      save outputfilename, replace
      restore
}
use outputfilename, clear
su alpha beta, d
hist beta, bin(10) normal frac title("Graph Title") saving(graphfilename, replace)
```

Notes:

-> We introduce a new feature here in Stata: -preserve- and –restore-

-> Essentially, when you -preserve- the data, Stata stores in its memory the data set as it exists at that point. You can then make whatever changes you want to the data, but when you type –restore-, Stata reverts back to the data as it existed when the –preserve- command was entered.

-> This is needed here because we have the original data set we are basing everything off of. We then replace the original sample with a bootstrap sample, perform the estimation on the bootstrap sample, save the result, then restore the original data set and repeat the whole process again.

-> In the examples above, clearly #1 is preferable. Knowing how to do #2 is necessary in cases where the model being estimated is not a "canned" command in Stata, or it is something that requires several lines of code.

Referring to Other Observations

Stata has easy ways to generate variables that are functions of values across observations. For example, with a time-series data set, you may be interested in the change from the previous period. There are two ways of doing this.

Example #1:

```
sort year
g deltay = y - y[_n-1]
```

Here the number in brackets refers to the observation (line) number to be used; _n refers to the current observation. By default, "y" is equivalent to "y[_n]".

Note: For the initial time period in the data, y[_n-1] is missing, and thus deltay[1] will be missing (coded as "."; see Missing Values section).

Example #2:

```
tsset year
g deltay = D.y
g lagy = L.y
g lag2y = L2.y
g fy = F.y
g f2y = F2.y
```

Here, D.*varname* generates the first-differenced value of *varname*; L.*varname* refers to the first lag of *varname* (lagy[1] will be missing); L2.*varname* refers to the second lag of *varname* (lag2y[1] and lag2y[2] will be missing); F.*varname* refers to the first lead of *varname* (fy[_N] will be missing); F2.*varname* refers to the second lead of *varname* (f2y[_N] and f2y[_N-1] will be missing). -tsset- is used to tell Stata which variable should be used to measure "time".

Notes:
-> _N refers to the sample size of the data currently in memory in Stata.
-> There is an important difference between methods #1 and #2. In Example #1, deltay will be computed (i.e., non-missing) even if the preceding observation is, say, two periods prior (due to missing data for a particular period). In Example #2, deltay will be missing if there is a gap in the "year" variable (due to a missing year in the data set).

Missing Values

Stata uses a "." to fill in missing values. It is important to realize that Stata interprets a "." as the largest real number. For example, suppose there are 100 observations in a data set, a variable called x, and x is missing for 50 observations and negative for the other 50 values. If you type:

```
count if x>0
```

then Stata will return r(N)=50. This is because "." is interpreted as, essentially, infinity. To avoid this, type:

```
count if x>0 & x!=.
```

Changing the Shape of the Data

Without going into details, frequently used commands for changing the "shape" of data sets are:

-collapse-
-reshape, wide-
-reshape, long-

Consult the manual.

Other Commands

-update query- -> contacts Stata, and downloads updates if Stata is not up-to-date