

RUDIMENTS OF STATA

This handout covers the most often encountered Stata commands. It is not comprehensive, but the summary will allow you to do basic data management and basic econometrics. I will provide some information on more advanced estimation commands through class handouts of Stata output.

Reading Data Files

The command to read a Stata file is `use`. If the Stata file is called `WAGE1.DTA`, and the file is on the diskette in the A: drive in the directory `DATA`, the command is

```
use a:\data\wage1
```

After entering this command the data file `WAGE1.DTA` is loaded into memory.

There is also a command, called `infile`, that allows you to read an ASCII file. I will not cover that here since all of the files we use have already been converted to Stata format. You should read the Stata manual or, even better, a publication called *Getting Started with Stata for Windows*, which is published by the Stata Press (College Station, Texas).

If you have loaded a file, say `WAGE1.DTA`, completed your analysis, and then wish to use a different data set, you simply clear the existing data set from memory:

```
clear
```

In doing this, it is important to know that any changes you made to the previous data set will be lost. You must explicitly save any changes you made (see "Leaving Stata" below). If, for example, you created a bunch of new variables, and you would like to have these variables available the next time you use the data, you should save the data set before using the clear command.

Looking At and Summarizing Your Data

After reading in a data file, you can get a list of the available variables by typing

```
des
```

Often a short description has been given to each variable. To look at the observations on one or more variable, use the list command. Thus, to look at the variables `wage` and `educ` for all observations, I type

```
list wage educ
```

This will list, one screen at a time, the data on *wage* and *educ* for every person in the sample. (Missing values in Stata are denoted by a period.) If the data set is large, you may not wish to look at all observations. You can always stop the listing (which Stata provides a screenful at a time) by hitting Ctrl-Break. In fact, Ctrl-Break can be used to interrupt any Stata command.

Alternatively, there are various ways to restrict the range of the list and many other Stata commands. Suppose I want to look at the first 20 observations on *wage* and *educ*. Then I type

```
list wage educ in 1/20
```

Rather than specify a range of observations, I can use a logical command instead. For example, to look at the data on marital status and age for people with zero hours worked, I can type

```
list married age if hours == 0
```

Note how the double equals here is used by Stata to determine equivalence. The other relational operators in Stata are > (greater than), < (less than), >= (greater than or equal), <= (less than or equal), and, ~= (not equal). If I want to restrict attention to nonblacks, I can type

```
list married age if ~black
```

The variable *black* is a binary indicator equal to unity for black individuals, and zero otherwise. The "~" is the logical "not" operator. We can combine many different logical statements: the command

```
list married age if black & (hours >= 40)
```

restricts attention to black people who work at least 40 hours a week. (Logical and is "&" and logical or is "|" in Stata.)

Two useful commands for summarizing data are the **sum** and **tab** commands. The **sum** command computes the sample average, standard deviation, and the minimum and maximum values of all (nonmissing) observations. Since this command tells you how many observations were used for each variable in computing the summary statistics, you can easily find out how many missing data points there for any variable. Thus, the command

```
sum wage educ tenure married
```

computes the summary statistics for the four variables listed. Since *married* is a binary variable, its minimum and maximum value are not very interesting. The average value reported is simply the proportion of people in the sample who are married.

I can also obtain summary statistics for any subgroup of the sample by adding on a logical statement.

```
sum wage educ tenure married if black | hispanic
```

computes the summary statistics for all blacks and hispanics (I assume these are binary variables in my data set). If I have a pooled cross section or a panel data set, and I want to summarize for 1990, I type

```
sum wage educ tenure married if == 1990
```

I can restrict the sample to certain observation ranges using the command in `m/n`, just as illustrated in the `list` command.

For variables that take on a relatively small number of values -- such as number of children or number of times an individual was arrested during a year -- I can use the `tab` command to get a frequency tabulation. The command

```
tab arrests
```

will report the frequency associated with each value of arrests in the sample. I can also combine `tab` with logical statements or restrict the range of observations.

Sometimes, we want to restrict all subsequent analysis to a particular subset of the data. In such cases it is useful to drop the data that will not be used at the start. This can be done using the `drop` or `keep` commands.

For example, if we want to analyze only blacks in a wage equation, then we can type

```
drop if ~black
```

This drops everyone in the sample who is not black. Or, to analyze only the years between 1986 and 1990 (inclusive), we can type

```
keep if (year >= 1986) & (year <= 1990)
```

It is important to know that the data dropped are gone from the current Stata session. If you want to get them back, you must reread the original data file. Along these lines, do not make the mistake of saving the smaller data set over the original one, or you will lose a chunk of your data. (More on this below under "Leaving Stata.")

Defining New Variables

It is easy to create variables that are functions of existing variables. In Stata, this is accomplished using the `gen` command (short for generate). For example, to create the square of experience, I can type

```
gen expersq = exper^2
```

The new variable, *expersq*, can be used in regression or any place else Stata variables are used. (Stata does not allow us to put expressions into things like regression commands; we must create the variables first.) If an observation had a missing value for *exper* then, naturally, *expersq* will also be missing for that observation. In fact, Stata will tell you how many missing observations were created after every **gen** command. If it reports nothing, then no missing observations were generated.

In creating new variables, you should be aware of the fact that the names of variables must be no longer than eight characters. Stata will refuse to accept names longer than eight characters in the **gen** command (and all other commands).

If I have the variable *saving* and would like to compute the natural log of *saving*, I can type

```
gen lsaving = log(saving)
```

If *saving* is missing then *lsaving* will also be missing. For functions such as the natural log, there is an additional issue: $\log(\textit{saving})$ is not defined for $\textit{saving} \leq 0$. When a function is not defined for particular values of the variable, Stata sets the result to missing. If negative or zero saving is a legitimate outcome, you probably do not want to use its natural log.

Logical commands can be used to restrict observations used for generating new variables. For example,

```
gen lwage = log(wage) if hours > 0
```

creates $\log(\textit{wage})$ for people who work (and therefore whose wage can be observed). Using the **gen** command without the statement `if hours > 0` has the same effect in this example.

Creating interaction terms is easy:

```
gen blkeduc = black*educ
```

where "*" denotes multiplication; the division operator is "/". Addition is "+" while subtraction is "-".

The **gen** command can also be used to create binary variables. For example, if *fratio* is the funding ratio of a firm's pension plan, I can create an "overfunded" dummy variable which is unity when $\textit{fratio} > 1$ and zero otherwise:

```
gen overfund = fratio > 1
```

The way this works is that the logical statement on the right hand side is evaluated to be true or false; true is assigned the value unity, and false assigned the value zero. So *overfund* is unity if $\textit{fratio} > 1$ and *overfund* is zero if $\textit{fratio} \leq 1$. Because of the way Stata treats missing values, we must be somewhat careful. In particular, a missing value is treated as being greater than any number. Therefore, " $\textit{fratio} > 1$ " will be true whenever *fratio* is missing. We clearly

do not want to set *overfund* to unity when *fratio* is missing. So if there are missing data, we should add the command

```
replace overfund = . if fratio == .
```

The `replace` command is generally useful for redefining certain values of a variable.

As another example, we can create year dummies using a command such as

```
gen y85 = (year == 1985)
```

where *year* is assumed to be a variable defined in the data set. The variable *y85* is unity for observations corresponding to 1985, and zero otherwise. We can do this for each year in our sample to create a full set of year dummies.

The `gen` command can also be used to difference data across different years. Suppose that, for a sample of cities, we have two years of data for each city (say 1982 and 1987). The data are stored so that the two years for each city are adjacent, with 1982 preceding 1987. To eliminate observed effects, say in relating city crime rates to expenditures on crimes and other city characteristics, we can use changes over time. The changes will be stored alongside the 1987 data. It is important to remember that for 1982 there is no change from a previous time period because we do not have data on a previous time period. Thus, we should define the change data so that it is missing in 1982. This is easily done. For example,

```
gen ccrime = crime - crime[_n-1] if year == 1987
```

```
gen cexpend = expend - expend[_n-1] if year == 1987
```

The variable "*_n*" is the reserved Stata symbol for the current observation; thus, *_n-1* is the variable lagged once. The variable *ccrime* is the change in *crime* from 1982 to 1987; *cexpend* is the change in *expend* from 1982 to 1987. These are stored in 1987, and the corresponding variables for 1982 will be missing. We can then use these changes in a regression analysis, or some other analysis.

As we saw above, the `replace` command is useful for correcting mistakes in definitions and redefining variables after values of other variables have changed. Suppose, for example, that in creating the variable *expersq*, I mistakenly type `gen expersq = exper^3`. One possibility is to drop the variable *expersq* and try again:

```
drop expersq
```

```
gen expersq = exper^2
```

(The `drop` command can be used for two different purposes: to drop a variable entirely from the data set, as above, or to drop some certain observations from the data set.)

A faster route is to use the `replace` command:

```
replace expersq = exper^2
```

Stata explicitly requires the replace command to write over the contents in a previously defined variable.

Basic Estimation Commands

Stata makes estimating a variety of models by a variety of methods straightforward. These notes cover the basic ones. We will encounter additional commands for panel data estimation, probit, tobit, and some other models later on.

Ordinary Least Squares

For OLS regression, we use the command `reg`. Immediately following `reg` is the dependent variable, and after that, all of the independent variables (order of the independent variables is not, of course, important). An example is

```
reg lwage educ exper expersq married black
```

This produces OLS estimates, standard errors, t statistics, confidence intervals, and a variety of other statistics usually reported with OLS. Unless a specific range of observations or a logical statement is included, Stata uses all possible observations in obtaining estimates. It does not use observations for which data on the dependent variable or any of the independent variables is missing. Thus, you must be aware of the fact that adding another explanatory variable can result in fewer observations used in the regression if some observations do not contain a value for that variable. If I add to the above regression a variable called *motheduc* (mother's education), and this is missing for certain individuals, then the sample size will be decreased accordingly.

Sometimes we wish to restrict our regression based on the size of one or more of the explanatory variables. In the regression

```
reg contribs mtchrate size sizesq if size <= 5000
```

where *size* is number of employees of a firm, the analysis is restricted to firms with no more than 5,000 employees. I can also restrict the regression to a particular year using a similar "if" statement, or to a particular observation range using the command `in m/n`.

Predicted values are obtained using the `predict` command. Thus, if I have run a regression with *lwage* as the dependent variable, I get the fitted values by typing

```
predict lwagehat
```

The choice of the name *lwagehat* is mine, subject to its being no more than eight characters and its not already being used. Note that the `predict` command saves the fitted values for the most recently run regression.

The residuals can be obtained by

```
predict uhat, resid
```

where again the name *uhat* is my choice.

Tests of multiple linear restrictions can be obtained after an OLS regression using the `test` command. For exclusion restrictions, just list the variables hypothesize to have no effect:

```
test north south east
```

jointly tests whether the three regional indicators can be excluded from the previously estimated model. Along with the value of the F statistic you also get a p-value. As with the `predict` command, `test` is applied to the most recently estimated model. More general linear hypotheses can be tested, but I will not cover those here. (These can always be rewritten as exclusion restrictions, anyway.)

OLS with heteroskedasticity-robust standard errors and t statistics is obtained using the `reg` command but adding `robust` at the end of the command line, preceded by a comma. So

```
reg lwage educ exper expersq married black, robust
```

still obtains the OLS estimates but reports heteroskedasticity-robust standard errors. The `robust` option is useful in other setups, too, including cluster samples and panel data. Any command that can be used after `reg` can be used after `reg` with the `robust` option. For example, we can test multiple exclusion restrictions in a heteroskedasticity-robust fashion by using the `test` command.

Two Stage Least Squares

The `reg` command can also be used to estimate models by 2SLS. After specifying the dependent variable and the explanatory variables -- which presumably include at least one explanatory variable that is correlated with the error -- we then list all of the exogenous variables as instruments in parentheses. Naturally, the list of instruments does not contain any endogenous variables.

An example of a 2SLS command is

```
reg lwage educ exper expersq married (motheduc fatheduc exper expersq married)
```

This produces 2SLS estimates, standard errors, t statistics, and so on. By looking at this command, we see that *educ* is an endogenous explanatory variable in the $\log(\text{wage})$ equation while *exper*, *expersq*, and *married* are assumed to be exogenous explanatory variables. The variables *motheduc* and *fatheduc* are assumed to be additional exogenous variables that do not appear in the $\log(\text{wage})$ structural equation but should have some correlation with *educ*. These appear in the instrument list along with the exogenous explanatory variables. The order in which we order the instruments is

not important. The necessary condition for the model to be identified is that the number of terms in parentheses is at least as large as the total number of explanatory variables. In this example, the count is five to four, and so the order condition holds.

Allowing for more than one endogenous explanatory variable is also easy. Suppose caloric consumption (*calories*) and protein consumption (*protein*) are endogenous in a wage equation for people in developing countries. However, we have regional prices on five commodity groups, say *price1*, ..., *price5*, to use as instruments. The Stata command for 2SLS might look like

```
reg lwage educ exper male protein calories (educ exper male price1 price2 price3 price4 price5) if year == 1990
```

if the analysis is restricted to data for 1990. Note that *educ*, *exper*, and *male* are taken to be exogenous here. The order condition easily holds ($8 > 5$).

After 2SLS, we can test multiple restrictions using the test command, just as with OLS.

Editing the Command Line

Stata has several shortcuts for entering commands. Two useful keys are Page Up and Page Down. If at any point you hit Page Up, the previously executed command appears on the command line. This can save on a lot of typing because you can hit Page Up and edit the previous command. Among other things, this makes adding an independent variable to a regression, or expanding an instrument list, fairly easy. Hitting Page Up repeatedly allows you to traverse through previously executed commands until you find the one you want. Hitting Page Down takes you back down through all of the commands.

It is easy to edit the command line. Hitting Home takes the cursor to the beginning of the line; hitting End moves the cursor to the end of the line. The key Delete deletes a single character to the right of the cursor; holding it down will delete many characters. The Backspace key (a left arrow on many keyboards) deletes a character to the left of the cursor. Hitting the left arrow (J) moves you one character to the left, and the right arrow (L) takes you one character to the right. You can hold down J and L to move several characters.

The key Ins allows you to toggle between insert and overwrite model. Both of these modes are useful for editing commands.

Recording Your Work: Creating a Log File

For involved projects, it is a good idea to create a record of what you have done (data transformations, regressions, and so on). To do this you can create a log file. Suppose I have a diskette in the A: drive and I would like to create a log file on this diskette. Before doing any analysis (but maybe after reading in my data set), I can type

```
log using a:ps4
```

This will create the file PS4.LOG on the diskette in the A: drive. Note that, unless you specify suffix explicitly, Stata adds ".LOG" to the end of the file name. Of course, if I want the log file on the default drive (usually C:) I would omit a: in the command.

Often, you might wish to temporarily close your log file while you look at the data, or run regressions that you are not sure you wish to keep. Since log files can get big in a hurry, it is useful to know that they can be turned off and on at any time. The commands are

log off

log on

These commands require that a log file has been opened (note that the name of the log file is not needed in either case). If you use the log off command, remember to type log on before doing anything you wish to keep.

When you are finished, you can close the log file for good:

log close

After typing this command, log on will not open the log file. If I decided a want to add onto the end of an existing log file, say PS4.LOG on the A: drive, I must type

log using a:ps4, append

Any subsequent Stata commands and output will be added to the end of PS4.LOG. If I omit the append command, Stata will not allow me to open a file called PS4.LOG. If I decide that what is currently in PS4.LOG is no longer needed, but I want to use the same file name, I would type

log using a:ps4, replace

You should use this with caution, because you will lose the old contents of PS4.LOG.

Stata log files are just standard ASCII files, and can be looked at using standard DOS commands (such as type and more). They can also be directly sent to a printer.

Leaving Stata

If I have made no changes to my data file, then to leave Stata (after closing my log file, if I have opened one up), I simply type

exit

Stata will not let me exit if new variables have been created, or if I have dropped part of the sample. Generally, if the data set is at all different from the one I initially read in, Stata will tell you this and refuse to let you exit. If you do not wish to save any changes you have made then type

exit, clear

This is especially useful if, after reading in the initial file, you dropped some observations before undertaking your analysis. In most cases you do not want to save the smaller data set over the original one.

The Help Command

You can learn more about the previous commands, as well as many other Stata commands, using the on-line help feature. To get a listing of the topics available using the help command, type

help contents

Information on specific commands can be obtained by typing **help** followed by the command name. You must use the full command name, not the abbreviations. Below are some examples:

help regress

help generate

help test

Using Stata as a Calculator and Computing p-values

Stata can be used to compute a variety of expressions, including certain functions that are not available on a standard calculator. The command to compute an expression is **display**, or **di** for short. For example, the command

```
di .048/(2*.0016)
```

will return "15." We can use **di** to compute natural logs, exponentials, squares, and so on. For example,

```
di exp(3.5 + 4*.06)
```

returns the value 42.098 (approximately). These previous examples can be done on most calculators. More importantly, we can use **di** to compute p-values after computing a test statistic. The command

```
di normprob(1.58)
```

This gives the probability that a standard normal random variable is greater than the value 1.58 (about .943). Thus, if a standard normal test statistic takes on the value 1.58, the p-value is $1 - .943 = .057$. Other functions are defined to give the p-value directly.

```
di tprob(df,t)
```

returns the the p-value for a t test against a two-sided alternative (t is the absolute value of the t statistic and df is the degrees of freedom). For example, with $df = 31$ and $t = 1.32$, the command returns the value .196. To obtain the p-value for an F test, the command is

```
di fprob(df1,df2,F)
```

where $df1$ is the numerator degrees of freedom, $df2$ is the denominator df, and F is the value of the F statistic. As an example,

```
di fprob(3,142,2.18)
```

returns the p-value .093.