Useful Stata Commands (for Stata versions 13 & 14)

Kenneth L. Simons

– This document is updated continually. For the latest version, open it from the course disk space. –

This document briefly summarizes Stata commands useful in ECON-4570 Econometrics and ECON-6570 Advanced Econometrics.

This presumes a basic working knowledge of how to open Stata, use the menus, use the data editor, and use the do-file editor. We will cover these topics in early Stata sessions in class. If you miss the sessions, you might ask a fellow student to show you through basic usage of Stata, and get the recommended text about Stata for the course and use it to practice with Stata.

More replete information is available in Lawrence C. Hamilton's *Statistics with Stata*, Christopher F. Baum's *An Introduction to Modern Econometrics Using Stata*, and A. Colin Cameron and Pravin K. Trivedi's *Microeconometrics using Stata*. See: http://www.stata.com/bookstore/books-on-stata/ .

Readers on the Internet: I apologize but I cannot generally answer Stata questions. Useful places to direct Stata questions are: (1) built-in help and manuals (see Stata's Help menu), (2) your friends and colleagues, (3) Stata's technical support staff (you will need your serial number), (4) Statalist (http://www.stata.com/statalist/) (but check the Statalist archives before asking a question there).

Most commands work the same in Stata versions 12, 11, 10, and 9.

Throughout, estimation commands specify robust standard errors (Eicker-Huber-White heteroskedastic-consistent standard errors). This does not imply that robust rather than conventional estimates of Var[**b**|**X**] should always be used, nor that they are sufficient. Other estimators shown here include Davidson and MacKinnon's improved small-sample robust estimators for OLS, cluster-robust estimators useful when errors may be arbitrarily correlated within groups (one application is across time for an individual), and the Newey-West estimator to allow for time series correlation of errors. Selected GLS estimators are listed as well. Hopefully the constant presence of "vce(robust)" in estimation commands will make readers sensitive to the need to account for heteroskedasticity and other properties of errors typical in real data and models.

Contents

Useful Stata (Version 14) Commands

Preliminaries for RPI Dot.CIO Labs

RPI computer labs with Stata include, as of Fall 2016: Sage 4510, the VCC Lobby (all Windows PCs), and hopefully now all Dot CIO labs.

To access the Stata program, use the Q-drive. Look under My Computer and open the disk drive Q:, probably labeled as "Common Drive (Q:)", then double-click on the program icon that you see. You must start Stata this way – it does not work to double-click on a saved Stata file, because Windows in the labs is not set up to know Stata is installed or even which saved files are Stata files.

To access the course disk space, go to: \\hass11.win.rpi.edu\classes\ECON-4570-6560. If you are logged into the WIN domain you will go right to it. If you are logged in locally on your machine or into anther domain you will be prompted for credentials. Use:

username: win\"rcsid"
password: "rcspassword"

substituting your RCS username for "rcsid" and your RCS password for "rcspassword". Once entered correctly the folder should open up.

To access your personal RCS disk space from DotCIO computers, find the icon on the desktop labeled "RPI AFS Files," double-click on it, and enter your username and password. Your personal disk space will be attached probably as drive H. (Public RCS materials may be attached perhaps as drive P.) Save Stata do-files to your personal disk space or a memory stick. For handy use when logging in, you may put the web address to attach the course disk space in a file on your personal disk space (e.g., drive H:); that way at the start of a session you can attach the RCS disk space and then open the file with your saved command and run it.

A. Loading Data

edit — Opens the data editor, to type in or paste data. You must close the data editor before you can run any further commands.

use "filename.dta" — Reads in a Stata-format data file.

insheet delimited "filename.txt" — Reads in text data (allowing for various text encodings), in Stata 14 or newer.

insheet using "filename.txt" — Old way to read text data, faster for plain English-language text.

import excel "filename.xlsx", firstrow — Reads data from an Excel file's first worksheet, treating the first row as variable names.

import excel "filename.xlsx", sheet("price data") firstrow — Reads data from the worksheet named "price data" in an Excel file, treating the first row as variable names.

save "filename.dta" — Saves the data.

Before you load or save files, you may need to change to the right directory. Under the File menu, choose "Change Working Directory…", or use Stata's "cd" command.

A1. Memory in Stata Version 11 or Earlier

As of this writing, Stata is in version 14. If you are using Stata version 11 or earlier, and you will read in a big dataset, then before reading in your data you must tell Stata to make available enough computer memory for your data. For example:

set memory 100m — Sets memory available for data to 100 megabytes. Clear before setting.

If you get a message while using Stata 11 or earlier that there is not enough memory, then clear the existing data (with the "clear" command), set the memory to a large enough amount, and then re-do your analyses as necessary – you should be saving your work in a do file, as noted below in section M).

B. Variable Lists, If-Statements, and Options

Most commands in Stata allow (1) a list of variables, (2) an if-statement, and (3) options.

1. A list of variables consists of the names of the variables, separated with spaces. It goes immediately after the command. If you leave the list blank, Stata assumes where possible that you mean all variables. You can use an asterisk as a wildcard (see Stata's help for varlist). Examples:
   edit var1 var2 var3 — Opens the data editor, just with variables var1, var2, and var3.
   edit — Opens the data editor, with all variables.
   In later examples, *varlist* means a list of variables, and *varname* (or *yvar* etc.) means one variable.
2. An if-statement restricts the command to certain observations. You can also use an in-statement. If- and in-statements come after the list of variables. Examples:
   edit var1 if var2 > 3 — Opens the data editor, just with variable var1, only for observations in which var2 is greater than 3.
   edit if var2 == var3 — Opens the data editor, with all variables, only for observations in which var2 equals var3.
   edit var1 in 10 — Opens the data editor, just with var1, just in the 10th observation.
   edit var1 in 101/200 — Opens the data editor, just with var1, in observations 101-200.
   edit var1 if var2 > 3 in 101/200 — Opens the data editor, just with var1, in the subset of observations 101-200 that meet the requirement var2 > 3.
3. Options alter what the command does. There are many options, depending on the command – get help on the command to see a list of options. Options go after any variable list and if-statements, and must be preceded by a comma. Do not use an additional comma for additional options (the comma works like a toggle switch, so a second comma turns off the use of options!). Examples:
   use "filename.dta", clear — Reads in a Stata-format data file, clearing all data previously in memory! (Without the clear option, Stata refuses to let you load new data if you haven't saved the old data. Here the old data are forgotten and will be gone forever unless you saved some version of them.)
   save "filename.dta", replace — Saves the data, replacing a previously-existing file if any.
   You will see more examples of options below.

C. Lowercase and Uppercase Letters

Case matters: if you use an uppercase letter where a lowercase letter belongs, or vice versa, an error message will display.

D. Review Window, and Abbreviating Command Names

The Review window lists commands you typed previously. Click in the Review window to put a previous command in the Command window (then you can edit it as desired). Double-click to run a command. Another shortcut is that many commands can have their names abbreviated. For example below instead of typing "summarize", "su" will do, and instead of "regress", "reg" will do.

E. Viewing and Summarizing Data

Here, remember two points from above: (1) leave a *varlist* blank to mean all variables, and (2) you can use if-statements to restrict the observations used by each command.

E1. Just Looking

If you want to look at the data but not change them, it is bad practice to use Stata's data editor, as you could accidentally change the data! Instead, use the browser via the button at the top, or by using the following command. Or list the data in the main window.

browse *varlist* Opens the data viewer, to look at data without changing them.

list *varlist* Lists data. If there's more than 1 screenful, press space for the next screen, or q to quit listing.

E2. Mean, Variance, Number of Non-missing Observations, Minimum, Maximum, Etc.

summarize *varlist* See summary information for the variables listed.

summarize *varlist*, detail See detailed summary information for the variables listed.

by *byvars*: summarize *varlist* See summary information separately for each group of unique values of the variables in *byvars*. For example, "by gender: summarize wage".

inspect *varlist* See a mini-histogram, and numbers of positives / zeroes / negatives, integers / non-integers, and missing data values, for each variable.

codebook *varlist* Another view of information about variables.

E3. Tabulations, Histograms, Density Function Estimates

tabulate *varname* Creates a table listing the number of observations having each different value of the variable *varname*.

tabulate *var1 var2* Creates a two-way table listing the number of observations in each row and column.

tabulate *var1 var2*, exact Creates the same two-way table, and carries out a statistical test of the null hypothesis that *var1* and *var2* are independent. The test is exact, in that it does not rely on convergence to a distribution.

tabulate *var1 var2*, chi2 Same as above, except the statistical test relies on asymptotic convergence to a normal distribution. If you have lots of observations, exact tests can take a long time and can run out of available computer memory; if so, use this test instead.

histogram *varname* Plots a histogram of the specified variable.

histogram *varname*, bin(#) normal The bin(#) option specifies the number of bars. The normal option overlays a normal probability distribution with the same mean and variance.

kdensity *varname*, normal Creates a "kernel density plot", which is an estimate of the pdf that generated the data. The "normal" option lets you overlay a normal probability distribution with the same mean and variance.

E4. Scatter Plots and Other Plots

scatter *yvar xvar* Plots data, with *yvar* on the vertical axis and *xvar* on the horizontal axis.

scatter *yvar1 yvar2 … xvar* Plots multiple variables on the vertical axis and *xvar* on the horizontal axis.

Stata has lots of other possibilities for graphs, with an inch-and-a-half-thick manual. For a quick web-based introduction to some of Stata's graphics commands, try the "Graphics" section of this web page: http://www.ats.ucla.edu/stat/stata/modules/. Or go to Stata's pdf manuals and look at [G] Graph intro, viewing especially the section labeled "A quick tour." Or use Stata's Help menu and choose “Stata Command…”, type “graph_intro”, and press return. Scroll down past the table of contents and read the section labeled “A quick tour.”

E5. Correlations and Covariances

The following commands compute the correlations and covariances between any list of variables. Note that if any of the variables listed have missing values in some rows, those rows are ignored in all calculations.

correlate *var1 var2 ...* Computes the sample correlations between variables.

correlate *var1 var2 ...*, covariance Computes the sample covariances between variables.

Sometimes you have missing values in some rows, but want to use all available data wherever possible – i.e., for some correlations but not others. For example, if you have data on health, nutrition, and income, and income data are missing for 90% of your observations, then you could compute the correlation of health with nutrition using all of the observations, while computing the correlations of health with income and of nutrition with income for just the 10% of observations that have income data. These are called “pairwise” correlations and can be obtained as follows:

pwcorr *var1 var2 ...* Computes pairwise sample correlations between variables.

F. Generating and Changing Variables

A variable in Stata is a whole column of data. You can generate a new column of data using a formula, and you can replace existing values with new ones. Each time you do this, the calculation is done separately for every observation in the sample, using the same formula each time.

F1. Generating Variables

generate *newvar* = … Generate a new variable using the formula you enter in place of “…”. Examples follow.

gen f = m * a Remember, Stata allows abbreviations: “gen” means “generate”.

gen xsquared = x^2

gen logincome = log(income) Use log() or ln() for a log-base-e, or log10() for log-base-10.

gen q = exp(z) / (1 – exp(z))

gen a = abs(cos(x)) This uses functions for absolute value, abs(), and cosine, cos(). Many more functions are available – get help for “functions” for a list.

F2. Missing Data

Be aware of missing data in Stata. Missing data can result when you compute a number whose answer is not defined; for example, if you use “gen logincome = log(income)” then logincome will be missing for any observation in which income is zero or negative. Missing data can also result during data collection; for example, in data on publicly listed companies often R&D expenditures data are unavailable.

Missing data can be entered in Stata by using a period instead of a number. When you list data, a period likewise indicates a missing datum.

Missing data can be used in Stata calculations. For example, you can check whether logincome is missing, and only list the data for observations where this is true:

list if logincome==. List only observations in which logincome is missing.

A missing datum counts as infinity when making comparisons. For example, if logincome is not missing, then it is less than infinity, so you could create a variable that tells whether logincome is non-missing by checking whether logincome is less the missing value code:

gen notmiss = logincome<. If logincome is less than infinity, then notmiss equals true which is recorded as 1, but otherwise notmiss equals false which is recorded as 0.

Should you need to distinguish reasons why data are missing, you could use Stata's "extended missing value" codes. These codes are written .a, .b, .c, …, .z. They all count as infinity when compared versus normal numbers, but compared to each other they are ranked as . < .a < .b < .c < … < .z. For this reason, to check whether a number in variable *varname* is missing you should use not "*varname*==." but "*varname*>=."

F3. True-False Variables

Below are examples of how to create true-false variables in Stata. When you create these variables, true will be 1, and false will be 0. When you ask Stata to check whether a number means true or false, then 0 will mean false and anything else (including a missing value) will mean true.

The basic operators used when creating true-false values are == (check whether something is equal), <, <=, >, >=, ! ("not" which changes false to true and true to false), and != (check whether something is not equal). You can also use & and | to mean logical "and" and "or" respectively, and you can use parentheses as needed to group parts of your expressions or equations.

When creating true-false values, as noted above, missing values in Stata work like infinity. So if age is missing and you use "gen old = age >= 18", then old gets set to 1 when really you don't know whether or not someone is old. Instead you should "gen old = age >= 18 if age<.". This is discussed more in section K below.

When using true-false values, 0 is false and anything else – including missing values – counts as true. So !(0) = 1, !(1) = 0, !(3) = 0, and !(.) = 0. Again, use an if-statement to ensure you generate non-missing values only where appropriate.

With these fundamentals in mind, here are examples of how to create true-false data in Stata:

gen young = age < 18 if age<. If age is less than 18, then young is "true", represented in Stata as 1. If age is 18 or over, then young is "false", represented in Stata as 0. The calculation is done only if age is "less than missing," i.e. nonmissing (see above), so that no answer (a missing value) will be generated if the age is unknown.

gen old = age >= 18 if age<. If age is 18 or higher, this yields 1, otherwise this yields 0 (but missing-value ages result in missing values for old).

gen age18 = age == 18 if age<. Use a single equal sign to set a variable equal to something. Use a double equal sign to check whether the left hand side equals the right hand side. In this case, age18 is created and equals 1 if the observation has age 18 and 0 if it does not.

gen youngWoman = age < 18 & female==1 if age<. & female<. Here the ampersand, "&", means a logical and. The variable youngWoman is created and equals 1 if and only if age is less than 18 and also female equals one; otherwise it equals 0. Here, the "if" condition ensures that the answer will be missing if either age or female is missing.

gen youngOrWoman = age<18 | female==1 if age<. & female<. Here the vertical bar, "|", means a logical or. The variable youngOrWoman is created and equals 1 if age is less than 18 or if female equals one; otherwise it equals 0. Here, the "if" condition ensures that the answer will be missing if either age or female is missing. You could improve on this if-condition to make the answer non-missing if the person is known to be young but has a missing value for female, or if the person is known to be female but has a missing value for age. To do so you could use: “gen youngOrWoman = age<18 | female==1 if (age<. & female<.) | (age<18) | (female==1)”.

gen ageNot18 = age != 18 if age<. The “!=” symbol means “not equal to”.

gen notOld = !old if old<. The “!” symbol is pronounced “not” and switches true to false or false to true. The result is the same as the variable young above.

F4. Random Numbers

gen r1 = runiform() Random numbers, uniformly distributed between 0 and 1.

gen r2 = rnormal() Random numbers, with a standard normal distribution.

gen r3 = rnormal(5,2) Random numbers, with a normal distribution using mean 5 and standard deviation 2. Alternatively, you could use “gen r3 = 5 + 2 * rnormal()”, or “gen r3 = 5 + 2 * invnorm(runiform())”

gen r4 = rchi2(27) Random numbers, with a chi-squared distribution with 27 degrees of freedom.

gen r5 = rt(27) Random numbers, with a *t*-distribution with 27 degrees of freedom.

For other random number distributions use Stata’s menu to get help for “functions”. You can also set the “seed” for random number generation (e.g., “set seed 1234”), to ensure that a reproducible sequence of random numbers will result thereafter – that way if you rerun your analyses later you can get exactly the same results.

F5. Replacing Values of Variables

replace agesquared = age^2 Changes the value of the variable agesquared, to equal age squared. This would be useful if you had made a mistake when you first created the variable.

replace young = age < 16 if age<. Changes the value of the variable young, to equal 1 if and only if age is less than 16, and 0 otherwise. The “if age<.” Ensures that replacements are only made when values of age are nonmissing – see the comments about missing values in sections F2 and F3 above.

replace young = cond(age<., age < 16, .) Here is another way to ensure that the answer is missing if age is missing. To do this, we use Stata’s conditional function, cond(a,b,c), which checks whether a is true and then returns b if a is true or c if a is not true (see F7 below).

replace young = 0 if age>=16 & age<18 Changes the value of the variable young to 0, but only if age is at least 16 and less than 18. That is, no change is made if age is less than 16 or if age is at least 18.

F6. Getting Rid of Variables

drop *varlist* Gets rid of all variables in the list.

clear Gets rid of all variables, as well as labels which are discussed in section X5.

clear all Gets rid of not just variables and labels, but also all sorts of things that we haven’t discussed yet: matrices, scalars, constraints, clusters, postfile declarations, returned results, programs, mata contents, and timer settings, and closes all open files.

F7. If-then-else Formulas

gen val = cond(a, b, c)   Stata's cond(*if*, *then*, *else*) works much like Excel's IF(*if*, *then*, *else*). With the statement cond(a,b,c), Stata checks whether a is true and then returns b if a is true or c if a is not true.

gen realwage = cond(year==1992, wage*(188.9/140.3), wage)   Creates a variable that uses one formula for observations in which the year is 1992, or a different formula if the year is not 1992. This particular example would be useful if you have data from two years only, 1992 and 2004, and the consumer price index was 140.3 in 1992 and 188.9 in 2004; then the example given here would compute the real wage by rescaling 1992 wages while leaving 2004 wages the same.

F8. Quick Calculations

display …   Calculate the formula you type in, and display the result. Examples follow.

display (52.3-10.0)/12.7

display normal(1.96)   Compute the probability to the left of 1.96 using the cumulative standard normal distribution.

display F(10,9000,2.32)   Compute the probability that an F-distributed number, with 10 and 9000 degrees of freedom, is less than or equal to 2.32. Also, there is a function Ftail(*n1*,*n2*,*f*) = 1 – F(*n1*,*n2*,*f*). Similarly, you can use ttail(*n*,*t*) for the probability that *T*>*t*, for a t-distributed random variable *T* with *n* degrees of freedom.

F9. More

For functions available in equations in Stata, use Stata's Help menu, choose Stata Command…, and enter "functions". To generate variables separately for different groups of observations, see the commands in sections O and P4. For time-series and panel data, see section P, especially the notations for lags, leads, and differences in section P3. If you need to refer to a specific observation number, use a reference like x[3], meaning the valuable of the variable x in the 3rd observation. In Stata "_n" means the current observation (when using generate or replace), so that for example x[_n-1] means the value of x in the preceding observation, and "_N" means the number of observations, so that x[_N] means the value of x in the last observation.

G. Means: Hypothesis Tests and Confidence Intervals

G1. Confidence Intervals

In Stata version 13 or earlier, omit the word "means" below.

ci means *varname*   Confidence interval for the mean of *varname* (using asymptotic normal distribution).

ci means *varname*, level(#)   Confidence interval at #%. For example, use 99 for a 99% confidence interval.

by *varlist*: ci means *varname*   Compute confidence intervals separately for each unique set of values of the variables in *varlist*.

by female: ci means workhours   Compute confidence intervals for the mean of workhours, separately for people who are males versus females.

Other commands also report confidence intervals, and may be preferable because they do more, such as computing a confidence interval for the difference in means between "by" groups (e.g., between men and women). See section G2. (Also, Stata's "mean" command reports confidence intervals.)

G2. Hypothesis Tests

ttest *varname* == # Test the hypothesis that the mean of a variable is equal to some number, which you type instead of the number sign #.

ttest *varname1* == *varname2* Test the hypothesis that the mean of one variable equals the mean of another variable.

ttest *varname*, by(*groupvar*) Test the hypothesis that the mean of a single variable is the same for all groups. The *groupvar* must be a variable with a distinct value for each group. For example, *groupvar* might be year, to see if the mean of a variable is the same in every year of data.

H. OLS Regression (and WLS and GLS)

regress *yvar xvarlist* Regress the dependent variable *yvar* on the independent variables *xvarlist*. For example: "regress y x", or "regress y x1 x2 x3".

regress *yvar xvarlist*, vce(robust) Regress, but this time compute robust (Eicker-Huber-White) standard errors. We are always using the vce(robust) option in ECON-4570 Econometrics, because we want consistent (i.e,, asymptotically unbiased) results, but we do not want to have to assume homoskedasticity and normality of the random error terms. So if you are in ECON-4570 Econometrics, remember always to specify the vce(robust) option after estimation commands. The "vce" stands for variance-covariance estimates (of the estimated model parameters).

regress *yvar xvarlist*, vce(robust) level(#) Regress with robust standard errors, and this time change the confidence interval to #% (e.g. use 99 for a 99% confidence interval).

Occasionally you will need to regress *without* vce(robust), to allow post-regression tests that assume homoscedasticity. Notably, Stata displays adjusted R^2 values only under the assumption of homoscedasticity, since the usual interpretation of R^2 presumes homoscedasticity. However, another way to see the adjusted R^2 after using "regress, … vce(robust)" is to type "display e(r2_a)"; see section I5.

H1. Variable Lists with Automated Category Dummies and Interactions

Stata (beginning with Stata 11) allows you enter variable lists that automatically create dummies for categories as well as interaction variables. For example, suppose you have a variable named usstate numbered 1 through 50 for the fifty U.S. states, and you want to include forty-nine 0-1 dummy variables that allow for differences between the first state (Alabama, say) and other states. Then you could simply include i.usstate in the *xvarlist* for your regression. Similarly, suppose you want to create the interaction between two variables, named age (a continuous variable) and male (a 0-1 dummy variable). Then, including c.age#i.male includes the interaction (the multiple of the two variables) in the regression. The "c." in front of age indicates that it is a continuous variable, whereas the "i." in front of male indicates that it is a 0-1 dummy variable. Including c.age#i.usstate adds 49 variables to the model, age times each of the 49 state dummies. Use "##"

instead of "#" to add full interactions, for example c.age##i.male means age, male, *and* age×male. Similarly, c.age##i.usstate means age, 49 state dummies, and 49 state dummies multiplied by age.

You can use "#" to create polynomials. For example, "age age#age age#age#age" is a third-order polynomial, with variables age and age^2 and age^3. Having done this, you can use Stata's "margins" command to compute marginal effects: the average value of the derivatives d(y)/d(age) across all observations in the sample. This works even if your regression equation includes interactions of age with other variables.

Here are some examples using automated category dummies and interactions, termed "factor variables" in the Stata manuals (see the User's Guide U11.4 for more information):

reg *yvar x1 i.x2*, vce(robust)   Includes a 0-1 dummy variables for the groups indicated by unique values of variable x2.

reg wage c.age i.male c.age#i.male, vce(robust)   Regress wage on age, male, and age×male.

reg wage c.age##i.male, vce(robust)   Regress wage on age, male, and age×male.

reg wage c.age##i.male c.age#c.age, vce(robust)   Regress wage on age, male, age×male, and age^2.

reg wage c.age##i.male c.age#c.age c.age#c.age#i.male, vce(robust)   Regress wage on age, male, age×male, age^2, and age^2×male.

reg wage c.age##i.usstate c.age#c.age c.age#c.age#i.usstate, vce(robust)   Regress wage on age, 49 state dummies, 49 variable that are age×statedummy$_k$, age^2, and 49 variable that are age^2×statedummy$_k$ (k=1,…,49).

Speed Tip: Don't "generate" lots of dummy variables and interactions – instead use this "factor notation" to compute your dummy variables and interactions "on the fly" during statistical estimation. This usually is much faster and saves lots of memory, if you have a really big dataset.

H2. Improved Robust Standard Errors in Finite Samples

For robust standard errors, an apparent improvement is possible. Davidson and MacKinnon[*] report two variance-covariance estimation methods that seem, at least in their Monte Carlo simulations, to converge more quickly, as sample size n increases, to the correct variance-covariance estimates. Thus their methods seem better, although they require more computational time. Stata by default makes Davidson and MacKinnon's recommended simple degrees of freedom correction by multiplying the estimated variance matrix by n/(n-K). However, students in ECON-6570 Advanced Econometrics learn about an alternative in which the squared residuals are rescaled. To use this formula, specify "vce(hc2)" instead of "vce(robust)", to use the approach discussed in Hayashi p. 125 formula 2.5.5 using d=1 (or in Greene's text, 6th edition, on p. 164). An alternative is "vce(hc3)" instead of "vce(robust)" (Hayashi page 125 formula 2.5.5 using d=2 or Greene p. 164 footnote 15).

H3. Weighted Least Squares

Students in ECON-6570 Advanced Econometrics learn about (variance-)weighted least squares. If you know (to within a constant multiple) the variances of the error terms for all observations, this yields more efficient estimates (OLS with robust standard errors works properly using asymptotic methods but is not the most efficient estimator). Suppose you have, stored in a variable *sdvar*, a reasonable estimate of the standard deviation of the error term for each observation. Then weighted least squares can be performed as follows:

[*] R. Davidson and J. MacKinnon, *Estimation and Inference in Econometrics*, Oxford: Oxford University Press, 1993, section 16.3.

vwls *yvar xvarlist*, sd(*sdvar*)

H4. Feasible Generalized Least Squares

Students in ECON-6570 Advanced Econometrics learn about feasible generalized least squares (Greene pp. 156-158 and 169-175). The groupwise heteroskedasticity model can be estimated by computing the estimated standard deviation for each group using Greene's (6th edition) equation 8-36 (p. 173): do the OLS regression, get the residuals, and use "by *groupvars*: egen *estvar* = mean(*residual*^2)" with appropriate variable names in place of the italicized words, then "gen *estsd* = sqrt(*estvar*), then use this estimated standard deviation to carry out weighted least squares as shown above. (To get the residuals, see section I1 below). Or, if your independent variables are just the group variables (categorical variables that indicate which observation is in each group) you can use the command:
vwls *yvar xvarlist*

The multiplicative heteroskedasticity model is available via a free third-party add-on command for Stata. See section J2a of this document for how to use add-on commands. If you have your own copy of Stata, just use the help menu to search for "sg77" and click the appropriate link to install. A discussion of these commands was published in the Stata Technical Bulletin volume 42, available online at: http://www.stata.com/products/stb/journals/stb42.pdf. The command then can be estimated like this (see the help file and Stata Technical Bulletin for more information):
reghv *yvar xvarlist*, var(*zvarlist*) robust twostage

I. Post-Estimation Commands

Commands described here work after OLS regression. They sometimes work after other estimation commands, depending on the command.

I1. Fitted Values, Residuals, and Related Plots

predict *yhatvar* — After a regression, create a new variable, having the name you enter here, that contains for each observation its fitted value $\hat{y}_i$.

predict *rvar*, residuals — After a regression, create a new variable, having the name you enter here, that contains for each observation its residual $\hat{u}_i$ (in the notation of Hayashi and most books $\hat{u}_i$ is written $e_i = \hat{\varepsilon}_i$).

scatter y yhat x — Plot variables named y and yhat versus x.

scatter resids x — It is wise to plot your residuals versus each of your x-variables. Such "residual plots" may reveal a systematic relationship that your analysis has ignored. It is also wise to plot your residuals versus the fitted values of y, again to check for a possible nonlinearity that your analysis has ignored.

rvfplot — Plot the residuals versus the fitted values of y.

rvpplot — Plot the residuals versus a "predictor" (x-variable).

For more such commands, see the nice "[R] regress postestimation" section of the Stata manuals. This manual section is a great place to learn techniques to check the trustworthiness of regression results – always a good idea!

I2. Confidence Intervals and Hypothesis Tests

For a single coefficient in your statistical model, the confidence interval is already reported in the table of regression results, along with a 2-sided t-test for whether the true coefficient is zero.

However, you may need to carry out F-tests, as well as compute confidence intervals and t-tests for “linear combinations” of coefficients in the model. Here are example commands. Note that when a variable name is used in this subsection, it really refers to the coefficient (the β_k) in front of that variable in the model equation.

lincom logpl+logpk+logpf Compute the estimated sum of three model coefficients, which are the coefficients in front of the variables named logpl, logpk, and logpf. Along with this estimated sum, carry out a t-test with the null hypothesis being that the linear combination equals zero, and compute a confidence interval.

lincom 2*logpl+1*logpk-1*logpf Like the above, but now the formula is a different linear combination of regression coefficients.

lincom 2*logpl+1*logpk-1*logpf, level(#) As above, but this time change the confidence interval to #% (e.g. use 99 for a 99% confidence interval).

test logpl+logpk+logpf==1 Test the null hypothesis that the sum of the coefficients of variables logpl, logpk, and logpf, totals to 1. This only makes sense after a regression involving variables with these names. After OLS regression, this is an F-test. More generally, it is a Wald test.

test (logq2==logq1) (logq3==logq1) (logq4==logq1) (logq5==logq1) Test the null hypothesis that four equations are all true simultaneously: the coefficient of logq2 equals the coefficient of logq1, the coefficient of logq3 equals the coefficient of logq1, the coefficient of logq4 equals the coefficient of logq1, and the coefficient of logq5 equals the coefficient of logq1; i.e., they are all equal to each other. After OLS regression, this is an F-test. More generally, it is a Wald test.

test x3 x4 x5 Test the null hypothesis that the coefficient of x3 equals 0 and the coefficient of x4 equals 0 and the coefficient of x5 equals 0. After OLS regression, this is an F-test. More generally, it is a Wald test.

I3. Nonlinear Hypothesis Tests

Students in ECON-6570 Advanced Econometrics learn about nonlinear hypothesis tests. After estimating a model, you could do something like the following:

testnl _b[popdensity]*_b[landarea] = 3000 Test a nonlinear hypothesis. Note that coefficients *must* be specified using _b, whereas the linear “test” command lets you omit the _b[].

testnl (_b[mpg] = 1/_b[weight]) (_b[trunk] = 1/_b[length]) For multi-equation tests you can put parentheses around each equation (or use multiple equality signs in the same equation; see the Stata manual, [R] testnl, for examples).

I4. Computing Estimated Expected Values for the Dependent Variable

di _b[*xvarname*] Display the value of an estimated coefficient after a regression. Use the variable name “_cons” for the estimated constant term. Of course there’s no need just to display these numbers, but the good thing is that you can use them in formulae. See the next example.

di _b[_cons] + _b[age]*25 + _b[female]*1 After a regression of *y* on age and female (but no other independent variables), compute the estimated value of *y* for a 25-year-old female. See also the predict command mentioned above in section I1, and the margins command.

I5. Displaying Adjusted R^2 and Other Estimation Results

```
display e(r2_a)
```

After a regression, the adjusted R-squared, $\bar{R}^2$, can be looked up as "e(r2_a)". Or get $\bar{R}^2$ as in section J below. (Stata does not report the adjusted R^2 when you do regression with robust standard errors, because robust standard errors are used when the variance (conditional on your right-hand-side variables) is thought to differ between observations, and this would alter the standard interpretation of the adjusted R^2 statistic. Nonetheless, people often report the adjusted R^2 in this situation anyway. It may still be a useful indicator, and often the (conditional) variance is still reasonably close to constant across observations, so that it can be thought of as an approximation to the adjusted R^2 statistic that would occur if the (conditional) variance were constant.)

```
ereturn list
```

Display all results saved from the most recent model you estimated, including the adjusted R^2 and other items. Items that are matrices are not displayed; you can see them with the command "matrix list e(matrixname)".

Study Tip: Students are strongly advised to understand the meanings of the two main sets of estimates that come out of regression models, (a) the coefficient estimates, and (b) the estimated variances and covariances of those coefficient estimates:

```
matrix list e(b)
```

List the coefficient estimates of your recent regression.

```
matrix list e(V)
```

List the estimated variances and covariances of your coefficient estimates in your recent regression. This is a symmetric matrix, so the part above the diagonal is not shown. The diagonal entries are estimated variances of your coefficient estimates (take square roots to get the standard errors), and the off-diagonal entries are estimated covariances.

Once you understand what *both* of these are, you'll have a much better understanding of what regression does (and you'll probably never need these particular "matrix list" commands!).

I6. Plotting Any Mathematical Function

```
twoway function y=exp(-x/6)*sin(x), range(0 12.57)
```

Plot a function graphically, for any function of a single variable x. A command like this may be useful to examine how a polynomial in one regressor (x) affects the dependent variable in a regression, without specifying values for other variables. The variable name on the right hand side must be "x" – do not use the names of variables in your data, or some values of those variables may be plugged in instead! If you are getting funny looking results, you may have used a different variable name instead of x; the right-hand variable must be named x.

```
twoway function y=_b[_cons]+_b[age]*x +_b[age2]*x^2 +_b[female]*1+_b[black]*1, range(0 30)
```

Plot a fitted regression function graphically, showing the fitted role of age in determining the average value of the dependent variable for black females. This would make sense after a regression in which the independent variables were age, a variable named age2 equal to age squared, an indicator variable named female, and an indicator variable named black. The term _b[*varname*] gets the estimated coefficient of the variable named *varname* in the most recent regression, or the estimated constant term if *varname* is _cons.

twoway function y = 3*x^2, range(-10 10) xtitle("expansion rate") ytitle("cost") title("Growth Cost") Axis labels and an overall graph title are added using the xtitle, ytitle, and title options.

I7. Influence Statistics

Influence statistics give you a sense of how much your estimates are sensitive to particular observations in the data. This may be particularly important if there might be errors in the data. After running a regression, you can compute how much different the estimated coefficient of any given variable would be if any particular observation were dropped from the data. To do so for one variable, for all observations, use this command:

predict *newvarname*, dfbeta(*varname*) Computes the influence statistic ("DFBETA") for *varname*: how much the estimated coefficient of *varname* would change if each observation were excluded from the data. The change divided by the standard error of *varname*, for each observation i, is stored in the ith observation of the newly created variable *newvarname*. Then you might use "summarize *newvarname*, detail" to find out the largest values by which the estimates would change (relative to the standard error of the estimate). If these are large (say close to 1 or more), then you might be alarmed that one or more observations may completely change your results, so you had better make sure those results are valid or else use a more robust estimation technique (such as "robust regression," which is not related to robust standard errors, or "quantile regression," both available in Stata).

If you want to compute influence statistics for many or all regressors, Stata's "dfbeta" command lets you do so in one step.

I8. Functional Form Test

It is sometimes important to ensure that you have the right functional form for variables in your regression equation. Sometimes you don't want to be perfect, you just want to summarize roughly how some independent variables affect the dependent variable. But sometimes, e.g., if you want to control fully for the effects of an independent variable, it can be important to get the functional form right (e.g., by adding polynomials and interactions to the model). To check whether the functional form is reasonable and consider alternative forms, it helps to plot the residuals versus the fitted values and versus the predictors, as shown in section I1 above. Another approach is to formally test the null hypothesis that the patterns in the residuals cannot be explained by powers of the fitted values. One such formal test is the Ramsey RESET test:

estat ovtest Ramsey's (1969) regression equation specification error test.

I9. Heteroskedasticity Tests

Students in ECON-6570 Advanced Econometrics learn about heteroskedasticity tests. After running a regression, you can carry out White's test for heteroskedasticity using the command:

estat imtest, white Heteroskedasticity tests including White test.

You can also carry out the test by doing the auxiliary regression described in the textbook; indeed, this is a better way to understand how the test works. Note, however, that there are many other heteroskedasticity tests that may be more appropriate. Stata's imtest command also carries out other tests, and the commands hettest and szroeter carry out different tests for heteroskedasticity.

The Breusch-Pagan Lagrange multiplier test, which assumes normally distributed errors, can be carried out after running a regression, by using the command:

estat hettest, normal   Heteroskedasticity test - Breusch-Pagan Lagrange mulitplier.

Other tests that do not require normally distributed errors include:

estat hettest, iid   Heteroskedasticity test – Koenker's (1981)'s score test, assumes iid errors.
estat hettest, fstat   Heteroskedasticity test – Wooldridge's (2006) F-test, assumes iid errors.
estat szroeter, rhs mtest(bonf)   Heteroskedasticity test – Szroeter (1978) rank test for null hypothesis that variance of error term is unrelated to each variable.
estat imtest   Heteroskedasticity test – Cameron and Trivedi (1990), also includes tests for higher-order moments of residuals (skewness and kurtosis).

For further information see the Stata manuals.

See also the ivhettest command described in section T1 of this document. This makes available the Pagan-Hall test which has advantages over the results from "estat imtest".

I10. Serial Correlation Tests

Students in ECON-6570 Advanced Econometrics learn about tests for serial correlation. To carry out these tests in Stata, you must first "tsset" your data as described in section P of this document (see also section U). For a Breusch-Godfrey test where, say, p = 3, do your regression and then use Stata's "estat bgodfrey" command:

estat bgodfrey, lags(1 2 3)   Heteroskedasticity tests including White test.

Other tests for serial correlation are available. For example, the Durbin-Watson d-statistic is available using Stata's "estat dwatson" command. However, as Hayashi (p. 45) points out, the Durbin-Watson statistic assumes there is no endogeneity *even under the alternative hypothesis*, an assumption which is typically violated if there is serial correlation, so you really should use the Breusch-Godfrey test instead (or use Durbin's alternative test, "estat durbinalt"). For the Box-Pierce Q in Hayashi's 2.10.4 or the modified Box-Pierce Q in Hayashi's 2.10.20, you would need to compute them using matrices. The Ljung-Box test is available in Stata by using the command:

wntestq *varname*, lags(#)   Ljung-Box portmanteau (Q) test for white noise.

I11. Variance Inflation Factors

Students in ECON-6570 Advanced Econometrics may use variance inflation factors (VIFs), which show the multiple by which the estimated variance of each coefficient estimate is larger because of non-orthogonality with other variables in the model. To compute the VIFs, use:

estat vif   After a regression, display variance inflation factors.

I12. Marginal Effects

After using "regress" or almost any other estimation command, you can compute marginal effects using the "margins" command (available beginning in Stata 11). Marginal effects are d(y)/d(x_k) for continuous variables x_k, or delta-y/delta-x_k for discrete variables x_k. In particular, these are reported for the average individual in the sample. Use factor variables when writing the list of variables in the model, so that Stata knows the way in which each variable contributes to the model – see section H1 above. Here is a simple example, but you should read the Stata manual entry [R] margins if you plan to use the margins command much.

margins age After a regression where the x-variables involve age, compute d(y)/d(age) on average among individuals in the sample.

margins , at(age=(20 25 30)) After a regression where the x-variables involve age, compute the predicted value of the dependent variable, y, for the average individual in the sample, given three alternative counterfactual assumptions for age. That is, first replace each person's age with 20, and compute the fitted value of y for each individual in the sample, and report the average fitted value. Then replace age with 25 and report the average fitted value, and do the same for age 30. This tells you what is predicted to happen for the average person in the sample if they were of a particular age. Hence it lets you compare, for the average of the individuals actually in your sample, the estimated effects of age.

J. Tables of Regression Results

This section will make your work much easier!

You can store results of regressions, and use previously stored results to display a table. This makes it much easier to create tables of regression results in Word. By copying and pasting, most of the work of creating the table is trivial, without errors from typing wrong numbers. Stata has built-in commands for making tables, and you should try them to see how they work, as described in section J1. In practice it will be much easier to use add-on commands, that you install, discussed in section J2.

J0. Copying and Pasting from Stata to a Word Processor or Spreadsheet Program

To put results into Excel or Word, the following method is fiddly but sometimes helps. Select the table you want to copy, or part of it, but *do not select anything additional*. Then choose Copy Table from the Edit menu. Stata will copy information with tabs in the right places, to paste easily into a spreadsheet or word processing program. For this to work, the part of the table you select must be in a consistent format, i.e., it must have the same columns everywhere, and you must not select any extra blank lines. (Stata figures out where the tabs go based on the white space between columns.)

After pasting such tab-delimited text into Word, use Word's "Convert Text to Table…" command to turn it into a table. In Word 2007, from the Insert tab, in the Tables group, click Table and select Convert Text to Table... (see: http://www.uwec.edu/help/Word07/tb-txttotable.htm); choose Delimited data with Tab characters as delimiters. Or if in Stata you used Copy instead of Copy Table, you can Convert Text to Table... and choose Fixed Width data and indicate where the columns break – *but* this "fixed width" approach is dangerous because you can easily make mistakes, especially if some numbers span multiple columns. In either case, you can then adjust the font, borderlines, etc. appropriately.

In section J2, you will see how to save tables as files that you can open in Word, Excel, and other programs. These files are often easier to use than copying and pasting, and will help avoid mistakes.

J1. Tables of Regression Results Using Stata's Built-In Commands

Please use the more powerful commands in section J2 below. However, the commands shown here also work, and are a quick way to get the idea. Here is an example of how to store results of regressions, and then use previously stored results to display a table:

```
regress y x1, vce(robust)
```

```
estimates store model1
regress y x1 x2 x3 x4 x5 x6 x7, vce(robust)
estimates store model2
regress y x1 x2 x3 x4 x6 x8 x9, vce(robust)
estimates store model3
estimates table model1 model2 model3
```

The last line above creates a table of the coefficient estimates from three regressions. You can improve on the table in various ways. Here are some suggestions:

estimates table model1 model2 model3, se   Includes standard errors.

estimates table model1 model2 model3, star   Adds asterisks for significance levels. Unfortunately "estimates table" does not allow the star and se options to be combined, however (see section J2 for an alternative that lets you combine the two).

estimates table model1 model2 model3, star stats(N r2 r2_a rmse)   Also adds information on number of observations used, R^2, $\bar{R}^2$, and root mean squared error. (The latter is the estimated standard deviation of the error term.)

estimates table model1 model2 model3, b(%7.2f) se(%7.2f) stfmt(%7.4g) stats(N r2 r2_a rmse)
Similar to the above examples, but formats numbers to be closer to the appropriate format for papers or publications. The coefficients and standard errors in this case are displayed using the "%7.2f" format, and the statistics below the table are displayed using the "%7.4g" format. The "%7.2f" tells Stata to use a fixed width of (at least) 7 characters to display the number, with 2 digits after the decimal point. The "%7.4g" tells Stata to use a general format where it tries to choose the best way to display a number, trying to fit everything within at most 7 characters, with at most 4 characters after the decimal point. Stata has many options for how to specify number formats; for more information get help on the Stata command "format".

You can store estimates after any statistical command, not just regress. The estimates commands have lots more options; get help on "estimates table" or "estimates" for information. Also, for items you can include in the stats(…) option, type "ereturn list" after running a statistical command – you can use any of the scalar results (but not macros, matrices, or functions).

J2. Tables of Regression Results Using Add-On Commands

In practice you will find it much easier to go a step further. A free set of third-party add-on commands gives much needed flexibility and convenience when storing results and creating tables.

What is an add-on command? Stata allows people to write commands (called "ado files") which can easily be distributed to other users. If you ever need to find available add-on commands, use Stata's help menu and Search choosing to search resources on the internet, and also try using Stata's "ssc" command.

J2a. Installing or Accessing the Add-On Commands

On your own computer, the add-on commands used here can be permanently installed as follows:

ssc install estout, replace   Installs the estout suite of commands.

In RPI's Dot.CIO labs, use a different method (because in the installation folder for add-on files, you don't have file write permission). I have put the add-on commands in the course disk space in a folder named “stata extensions”. You merely need to tell Stata where to look (you could copy the relevant files anywhere, and just tell Stata where). Type the command listed below in Stata. You only need to run this command once after you start or restart Stata. Put the command at the beginning of your do-files (you also may need to include the command “eststo clear” to avoid any confusion with previous results – see section J2h).

```
adopath + folderToLookIn
```

Here, replace *folderToLookIn* with the name of the folder, by using one of the following two commands (the first for ECON-4570 or -6560, the second for ECON-6570):

```
adopath + "//hass11.win.rpi.edu/classes/ECON-4570-6560/stata extensions"
adopath + "//hass11.win.rpi.edu/classes/ECON-6570/stata extensions"
```

(Note the use of forward slashes above instead of the Windows standard of backslashes for file paths. If you use backslashes, you will probably need to use four backslashes instead of two at the front of the file path. Why? In certain settings, including in do-files, Stata converts two backslashes in a row into just one – for Stata \$ means $, \` means `, and \\ means \, in order to provide a way to tell Stata that a dollar sign is not the start of a global macro but is just a dollar sign, or a backquote is not the start of a local macro but is just a backquote. (A local macro is Stata’s name for a local variable in a program or do-file, and a global macro is Stata’s name for a global variable in a program or do-file.))

J2b. Storing Results and Making Tables

Once this is done, you can store results more simply, store additional results not saved by Stata’s built-in commands, and create tables that report information not allowed using Stata’s built-in commands.

```
eststo: reg y x1 x2, vce(robust)
```

Regress y on x1 and x2 (with robust standard errors) and store the results. Estimation results will be stored with names like “est1”, “est2”, etc. – the name will be printed out after each command.

```
eststo modelname: reg y x1 x2, vce(robust)
```

Same as above, but you choose the name to use when storing results, instead of just using “est1”, etc. The *modelname* could be for example myreg1 (begin your names with a letter, after which you can use letters, digits 0 through 9, or underscores _ up to 32 total characters).

```
eststo: quietly reg y x1 x2 x3, vce(robust)
```

Similar to above, but “quietly” tells Stata not to display any output.

J2c. Near-Publication-Quality Tables

Here is how to make a near-publication-quality table. In place of the “est1 est2” below, type the names of the stored estimates that you want in the table.

```
esttab est1 est2, b(a3) se(a3) star(+ 0.10 * 0.05 ** 0.01 *** 0.001) r2(3) ar2(3) scalars(F) nogaps
```

Make a near-publication-quality table. You will still need to make the variable names more meaningful, change the column headings, and set up the borders appropriately.

Here is how to save that table in a file that you can open in Word. Put “using *filename*” just before the comma in the above command, and add the “rtf” option after the comma. Make sure you change directory first, so the file will save in the right folder. To change directory, under the File menu, choose “Change Working Directory…”, or use Stata’s “cd” command.

esttab est1 est2 using mytable, rtf b(a3) se(a3) star(+ 0.10 * 0.05 ** 0.01 *** 0.001) r2(3) ar2(3) scalars(F) nogaps
Save a near-publication-quality table, putting it in a rich text file ("mytable.rtf") that can be opened by Word.

J2d. Understanding the Table Command's Options

The esttab commands for near-publication-quality had a lot in them, so it may help to look at simpler versions of the command to understand how esttab works:

esttab Display a table with all stored estimation results, with t-statistics (not standard errors). Numbers of observations used in estimation are at the bottom of each column.

esttab, se Display a table with standard errors instead of t-statistics.

esttab, se ar2 Display a table with standard errors and adjusted R-squared values.

esttab, se ar2 scalars(F) Like the previous table, but also display the F-statistic of each model (versus the null hypothesis that all coefficients except the constant term are zero).

esttab, b(a3) se(a3) ar2(2) Like "esttab, se ar2", but this controls the display format for numbers. The "(a3)" ensures at least 3 significant digits for each estimated regression coefficient and for each standard error. The "(2)" gives 2 decimal places for the adjusted R-squared values. You can also specify standard Stata number formats in the parentheses, e.g., "%9.0g" or "%8.2f" could go in the parentheses (use Stata's Help menu, choose Command, and get help on "format").

esttab, star(+ 0.10 * 0.05 ** 0.01 *** 0.001) Set the p-values at which different asterisks are used.

esttab, nogaps Get rid of blank spaces between rows. This aids copying of tables to paste into, e.g., Word.

Some options above – the R-squared, adjusted R-squared, and F statistics – pertain to OLS regression, but not to many other types of statistical analysis. After logit or probit regression, for example, these statistics are not defined. After a statistical analysis, type "ereturn list" to see a list of returned estimation results, like e(F), e(chi2), e(r2_p), and e(cmd). You can request these using esttab's scalars option, for example "scalars(F chi2 r2_p cmd)". The esttab command leaves blank cells wherever a statistic is not defined.

J2e. Saving Tables as Files

It can be helpful to save tables in files, which you can open later in Word, Excel, and other programs. Although they are not used here, you can use all the options discussed above (like in the near-publication-quality example that saved a rich text file for Word):

esttab est1 est2 using results.txt, tab Save the table, with columns for the stored estimates named "est1" and "est2", into a tab-delimited text file named "results.txt".

esttab est1 est2 using results, rtf Save a rich-text format file, good for opening in Word.

esttab est1 est2 using results, csv Save a comma-separated values text file, named "results.csv", with the table. This is good for opening in Excel. However, numbers will appear in Excel as text.

esttab est1 est2 using results, csv plain Save a file good for use in Excel. The "plain" option lets you use the numbers in calculations.

esttab est1 est2 using results, tex Save for LaTeX.

J2f. Wide Tables

If you try to display estimates from many models at once, they may not all fit on the screen. The solution is to drag the Results window to the right to allow longer lines. If you are using Stata 10 or earlier, you must also use the "set linesize #" command as in the example below to actually use longer lines:

```
set linesize 140            Tell Stata to allow 140 characters in each line of Results window output.
```

In any case, you can now make very wide tables with lots of columns. Another way to fit more in the Results window is to reduce the font size: right-click or control-click in the Results window and change your preference for the font size.

In Microsoft Word, wide tables may best fit on landscape pages: create a Section Break beginning on a new page, then format the new section of the document to turn the page sideways in landscape mode. You can create a new section break beginning on a new page to go back to vertical layout on later pages. Also, Microsoft Word has commands to auto-fit tables to their contents or to the "window" of available space, and to auto-format tables – though you will need to edit the automatic formatting appropriately.

J2g. Storing Additional Results

After estimating a statistical model, you can add additional results to the stored information. For example, you might want to do an F-test on a group of variables, or analyze a linear combination of coefficient estimates. Here is an example of how to compute a linear combination and add information from it to the stored results. You can display the added information at the bottom of tables of results by using the scalars() option:

```
eststo: reg y x1 x2, vce(robust)    Regress.
lincom x1 - x2                      Get estimated difference between the coefficients of x1 and x2.
estadd scalar xdiff = r(estimate)   Store the estimated difference along with the regression result.
                                        Here it is stored as a scalar named xdiff.
estadd scalar xdiffSE = r(se)       Store the standard error for the estimated difference too. Here it is
                                        stored as a scalar named xdiffSE.
esttab, scalars(xdiff xdiffSE)      Include xdiff and xdiffSE in a table of regression results.
```

J2h. Clearing Stored Results

Results stored using eststo stay around until you quit Stata. To remove previously stored results, do the following:

```
eststo clear                Clear out all previously stored results, to avoid confusion (or to free
                                some RAM memory).
```

J2i. More Options and Related Commands

For more examples of how to use this suite of commands, use Stata's on-line help after installing the commands, or better yet, use this website: http://fmwww.bc.edu/repec/bocode/e/estout/ . On the website, look under Examples at the left.

J3. Tabulations and General Tables Using Add-On Commands

To control the formatting of tabulations and other tables, try the "tabout" add-on command. A clear introduction is: http://www.ianwatson.com.au/stata/tabout_tutorial.pdf .

K. Data Types, When 3.3 ≠ 3.3, and Missing Values

This section is somewhat technical and may be skipped on a first reading. Computers can store numbers in more or less compact form, with more or fewer digits. If you need extra precision, you can use "double" precision variables instead of the default "float" variables (which are single-precision floating-point numbers). If you need compact storage of integers, to save memory (or to store precise values of big integers), Stata provides other data types, called "byte", "int", and "long". Also, a string data type, "str", is available.

gen *type varname* = … Generate a variable of the specified data-type, using the specified formula. Examples follow.

gen double bankHoldings = 1234567.89 Double-precision numbers have 16 digits of accuracy, instead of about 7 digits for regular float numbers.

gen byte young = age<16 Here since the result is a 0 or 1, using the "byte" number format accurately records the number in a small amount of memory.

gen str name = firstname + " " + lastname Generates a variable involving strings.

The following commands help deal with data types.

describe *varlist* Lists technical information about variables, including data types.

compress *varlist* Changes data to most compact form possible without losing information.

If you compare a floating-point number, accurate to about 7 digits, to a double-precision number, accurate to 16 digits, don't expect them to be equal. The actual calculations Stata carries out are in double-precision, even though variables are ordinarily "float" (single-precision) to save space. Suppose you generate a float-type variable named rating, equal to 3.3 in the first observation. Stata stores the number as 3.3 accurate to about 7 digits. Then typing "list if rating==3.3" will fail to list the first observation. Why? Stata looks up the value of rating, which in the first observation is 3.3 accurate to about 7 digits, and compares it to the number 3.3, which is immediately put into double-precision for the calculation and hence is accurate to 16 digits, and hence is different from the rating. Hence the first observation will not be listed. Instead you could do this:

list if rating == float(3.3) The float 3.3 converts to a number accurate to only about 7 digits, the same as the rating variable.

Missing values of numbers in Stata are written as a period. They occur if you enter missing values to begin with, or if they arise in a calculation that has for example 0/0 or a missing number plus another number. For comparison purposes, missing values are treated like infinity, and when you're not used to this you can get some weird results. For example, "replace z = 0 if y>3" causes z to be replaced with 0 not only if y has a known value greater than 3 but also if the value of y is missing. Instead use something like this: "replace z = 0 if y>3 & y<.". The same caution applies when generating variables, anytime you use an if-statement, etc. (see sections F2, F3, and F5).

L. Results Returned after Commands

Commands often return results that can be used by programs you might write. To see a list of the results from the most recent command that returned results, type:

return list Shows returned results from a general command, like summarize.

ereturn list Shows returned results from an estimation command, like regress.

M. Do-Files and Programs

You should become well used to the do-file editor, which is the sensible way to keep track of your commands. Using the do-file editor, you can save previously used lists of commands and reopen them whenever needed. If you are analyzing data (for class work, for a thesis, or for other reasons), keeping your work in do-files both provides a record of what you did, and lets you make corrections easily. This document mainly assumes you are used to the do-file editor, but below are two notes on using and writing do-files, plus an example of how to write a program.

At the top of the do-file editor are icons for various purposes. Move the mouse over each icon to display what it does. The set of icons varies across computer types and versions of Stata, but might include: new do-file, open do-file, save, print, find in this do-file, show white-space symbols, cut, copy, paste, undo, redo, preview in viewer, run, and do. The "preview in viewer" icon you won't need (it's useful when writing documents such as help files for Stata's viewer). The "do" icon, at the right, is most important. Click on it to "do" all of the commands in the do-file editor: the commands will be sent to Stata in the order listed. However, if you have selected some text in the do-file editor, then only the lines of text you selected will be done, instead of all of the text. (If you select part of a line, the whole line will still be done.) The "run" icon has the same effect, except that no output is printed in Stata's results window. Since you will want to see what is happening, you should use the "do" icon not the "run" icon.

You will want to include comments in the do-file editor, so you remember what your do-files were for. There are three ways to include comments: (1) put an asterisk at the beginning of a line (it is okay to have white space, i.e., spaces and tabs, before the asterisk) to make the line a comment; (2) put a double slash "//" anywhere in a line to make the rest of the line a comment; (3) put a "/*" at the beginning of a comment and end it with "*/" to make anything in between a comment, even if it spans multiple lines. For example, your do-file might look like this:

```
* My analysis of employee earnings data.
* Since the data are used in several weeks of the course, the do-file saves work for later use!
clear  // This gets rid of any pre-existing data!
adopath + "//hass11.win.rpi.edu/classes/ECON-4570/stata extensions"  // If you're in ECON-4570.
use "L:\myfolder\myfile.dta"
* I commented out the following three lines since I'm not using them now:
/* regress income age, vce(robust)
predict incomeHat
scatter incomeHat income age */
* Now do my polynomial age analyses:
gen age2 = age^2
gen age3 = age^3
eststo p3: regress income age age2 age3 bachelor, vce(robust)
eststo p2: regress income age age2 bachelor, vce(robust)
esttab p3 p2, b(a3) se(a3) star(+ 0.10 * 0.05 ** 0.01 *** 0.001) r2(3) ar2(3) scalars(F) nogaps
```

You can write programs in the do-file editor, and sometimes these are useful for repetitive tasks. Here is a program to create some random data and compute the mean.

```
capture program drop randomMean              Drops the program if it exists already.
program define randomMean, rclass            Begins the program, which is "rclass".
    drop _all                                Drops all variables.
    quietly set obs 30                       Use 30 observations, and don't say so.
    gen r = uniform()                        Generate random numbers.
    summarize r                              Compute mean.
    return scalar average = r(mean)          Return it in r(average).
end
```

Note above that "rclass" means the program can return a result. After doing this code in the do-file, you can use the program in Stata. Be careful, as it will drop all of your data! It will then generate 30 uniformly-distributed random numbers, summarize them, and return the average. (By the way, you can make the program work faster by using the "meanonly" option after the summarize command above, although then the program will not display any output.)

N. Monte-Carlo Simulations

It would be nice to know how well our statistical methods work in practice. Often the only way to know is to simulate what happens when we get some random data and apply our statistical methods. We do this many times and see how close our estimator is to being unbiased, normally distributed, etc. (Our OLS estimators will do better with larger sample sizes, when the x-variables are independent and have larger variance, and when the random error terms are closer to normally distributed and have smaller variance.) Here is a Stata command to call the above (at the end of section M) program 100,000 times and record the result from each time.

```
simulate "randomMean" avg=r(average), reps(100000)
```

The result will be a dataset containing one variable, named avg, with 100,000 observations. Then you can check the mean and distribution of the randomly generated sample averages, to see whether they seem to be nearly unbiased and nearly normally distributed.

```
summarize avg
kdensity avg , normal
```

"Unbiased" means right on average. Since the sample mean, of say 30 independent draws of a random variable, has been proven to give an unbiased estimate of the variable's true population mean, you had better find that the average (across all 100,000 experiments) result computed here is very close to the true population mean. And the central limit theorem tells you that as a sample size gets larger, in this case reaching the not-so-enormous size of 30 observations, the means you compute should have a probability distribution that is getting close to normally distributed. By plotting the results from the 100,000 experiments, you can see how close to normally-distributed the sample mean is. Of course, we would get slightly different results if we did another set of 100,000 random trials, and it is best to use as many trials as possible – to get exactly the right answer we would need to do an infinite number of such experiments.

Try similar simulations to check results of OLS regressions. You will need to change the program in section M and alter the "simulate" command above. One approach is to change the program in section M to return results named "b0", "b1", "b2", etc., by setting them equal to the coefficient estimates _b[*varname*], and then alter the "simulate" command above to use the regression coefficient estimates instead of the mean (you might say "b0=r(b0) b1=r(b1) b2=r(b2)" in place of "avg=r(average)"). An easier approach, though, is to get rid of the ", rclass" in the program at the end of section M, and just do the regression in the program – the regression command itself will return results that you can use; your simulate command might then be something like "simulate "randomReg" b0=_b[_cons] b1=_b[x1] b2=_b[x2], reps(1000)".

O. Doing Things Once for Each Group

Stata's "by" command lets you do something once for each of a number of groups. Data must be sorted first by the groups. For example:

```
sort year                          Sort the data by year.
by year: regress income age, vce(robust)    Regress separately for each year of data.
sort year state                    Sort the data by year, and within that by state.
by year state: regress income age, vce(robust)    Regress separately for each state and year
                                       combination.
```

Sometimes, when there are a lot of groups, you don't want Stata to display the output. The "quietly" command has Stata take action without showing the output:

quietly by year: generate xInFirstObservationOfYear = x[1] The "x[1]" means look at the first observation of x within each particular by-group.

quietly by year (dayofyear): generate xInFirstObservationOfYear = x[1] In the above command, a problem is that you might accidentally have the data sorted the wrong way within each year. Listing more variables in parentheses after the year requires that within each year, the data must be sorted correctly by the other variables. This doesn't do the sorting for you, but it ensures the sort order is correct. That way you know what you'll get when you refer to the first observation of the year.

quietly bysort year (dayofyear): generate xInFirstObservationOfYear = x[1] This is the same as above, but the "bysort" command sorts as requested before doing the command for each by-group.

qby year (dayofyear): generate xInFirstObservationOfYear = x[1] "qby" is shorthand for "quietly by".

qbys year (dayofyear): generate xInFirstObservationOfYear = x[1] "qbys" is shorthand for "quietly bysort".

See also section P4 for more ways to generate results, e.g., means or standard deviations, separately for each by-group.

Power User Tip: Master these commands for by-groups to help make yourself a data preparation whiz. Also master the "egen" command (see section P4).

P. Generating Variables for Time-Series and Panel Data

With panel and time series data, you may need to (1) create a time variable; (2) tell Stata what variable measures time (and for panel data what variable distinguishes individuals in the sample); (3) use lags, leads, and differences; and (4) generate values separately for each individual in the sample. Here are some commands to help you.

P1. Creating a Time Variable

You need a time variable that tells the year, quarter, month, day, second, or whatever unit of time corresponds to each observation. A common problem is to convert data from some other format, like a month-day-year string, or numeric values for quarter and year, into a single time variable. Stata has lots of tools to help, as documented in Stata's help for "datetime". Some common methods are listed below.

Your time variable should be an integer, and should not usually have gaps between numbers. For example, it is okay to have years in the data be 1970, 1971, …, 2006, but if your time variable is every other year, e.g., 1970, 1972, 1974, …, then you should create a new variable like time = (year-1970)/2. Stata has lots of options and commands to help with setting up quarterly data, etc. The following is (as always in this document) just a start.

P1a. Time Variable that Starts from a First Time and Increases by 1 at Each Observation

If you have not yet created a time variable, and your data are in order and do not have gaps, you might create a year, quarter, or day variable as follows:

generate year = 1900 + _n - 1 Create a new variable that specifies the year, beginning with 1900 in the first observation and increasing by 1 thereafter. Be sure your data are sorted in the right order first.

generate quarter = tq(1970q1) + _n - 1 Create a new variable that specifies the time, beginning with 1970 quarter 1 in the first observation, and increasing by 1 quarter in each observation. Be sure your data are sorted in the right order first. The result is an integer number increasing by 1 for each quarter (1960 quarter 2 is specified as 1, 1960 quarter 3 is specified as 2, etc.).

format quarter %tq Tell Stata to display values of quarter as quarters.

generate day = td(01jan1960) + _n - 1 Create a new variable that specifies the time, beginning with 1 Jan. 1960 in the first observation, and increasing by 1 day in each observation. Be sure your data are sorted in the right order first. The result is an integer number increasing by 1 for each day (01jan1960 is specified as 0, 02 jan1960 is specified as 2, etc.).

format day %td Tell Stata to display values of day as dates.

Like the td(…) and tq(…) functions used above, you may also use tw(…) for week, tm(…) for month, or th(…) for half-year. For more information, get help on “functions” and look under “time-series functions”.

P1b. Time Variable from a Date String

If you have a string variable that describes the date for each observation, and you want to convert it to a numeric date, you can probably use Stata’s very flexible date conversion functions. You will also want to format the new variable appropriately. Here are some examples:

gen t = daily(dstr, "mdy") Generate a variable t, starting from a variable “dstr” that contains dates like “Dec-1-2003”, “12-1-2003”, “12/1/2003”, “January 1, 2003, “jan1-2003”, etc. Note the "mdy", which tells Stata the ordering of the month, day, and year in the variable. If the order were year, month, day, you would use "ymd".

format t %td This tells Stata the variable is a date number that specifies a day.

Like the daily(…) function used above, The similar functions monthly(*strvar*, "ym") or monthly(*strvar*, "my"), and quarterly(*strvar*, "yq") or quarterly(*strvar*, "qy"), allow monthly or quarterly date formats. Use %tm or %tq, respectively, with the format command. These date functions require a way to separate the parts. Dates like “20050421” are not allowed. If d1 is a string variable with such dates, you could create dates with separators in a new variable d2 suitable for daily(…), like this:

gen str10 *d2* = substr(*d1*, 1, 4) +"-" + substr(*d1*, 5, 2) +"-" + substr(*d1*, 7, 2) This uses the substr(…) function, which returns a substring – the part of a string beginning at the first number’s character for a length given by the second number.

P1c. Time Variable from Multiple (e.g., Year and Month) Variables

What if you have a year variable and a month variable and need to create a single time variable? Or what if you have some other set of time-period numbers and need to create a single time variable? Stata has functions to build the time variable from its components:

gen t = ym(year, month) Create a single time variable t from separate year (the full 4-digit year) and month (1 through 12) variables.

format t %tm This tells Stata to display the variable’s values in a human-readable format like “2012m5” (meaning May 2012).

Other functions are available for other periods:

If your data are	Instead of ym() use	Instead of %tm use
Yearly	y(year)	%ty
Half-yearly	yh(year, halfyear)	%th
Quarterly	yq(year, quarter)	%tq
Monthly	ym(year, month)	%tm
Weekly	yw(year, week)	%tw
Daily	mdy(month, day, year)	%td
In Milliseconds *	mdyhms(month, day, year, hour, minute, second)	%tc

*For data in milliseconds, data *must* be stored in double-precision number format (see section K above), using "gen double t = mdyhms(month, day, year, hour, minute, second)". For any of the other periodicities above, you can use long or double data types to store a broader range of numbers than is possible using the default float data type. For data in milliseconds, a version accounting for leap seconds uses "Cmdyhms(month, day, year, hour, minute, second)" and "%tC".

If your data do not match one of these standard periodicities, you can create your own time variable as in section P1a, but without using the "format" command to specify a human-readable format (the time numbers will just display as the numbers they are).

P1d. Time Variable Representation in Stata

If you use one of these standard Stata date formats, formatting your numbers with a Stata format such as %tq or %td, then Stata is storing the information as ordinary numbers and has picked the number 0 (zero) to correspond to some particular time. For example, in daily format, 0 means the first day of January in 1960, 1 means the second day of January 1960, -1 means the last day of December 1959, and so on. In other formats except yearly (%ty), 0 means the first time period (quarter, day, second, etc.) at the beginning of the year 1960.

To figure out the number that corresponds to a given date, you can use functions like tq(1989q4) for a quarter, tm(2016apr) for a month, or td(16jan1770) for a day.

P2. Telling Stata You Have Time Series or Panel Data

You *must* declare your data as time series or panel data in order to use time-related commands:

tsset *timevar* — Tell Stata you have time series data, with the time listed in variable *timevar*.

tsset *idvar timevar* — Tell Stata you have panel data, with the *idvar* being a unique ID for each individual in the sample, and *timevar* being the measure of time.

P3. Lags, Forward Leads, and Differences

After using the tsset command (see above), it is easy to refer to past and future data. The value of *var* one unit of time ago is L.*var*, the value two units of time ago is L2.*var*, etc. (the Ls stand for "lag"). Future values, although you are unlikely to need them, are F.*var*, F2.*var*, etc. Below are some examples using them. Data must be sorted first, in order by time for time-series data, or in order by individual and within that by time for panel data.

sort *timevar* — Sort time-series data.

sort *idvar timevar* — Sort panel data.

gen changeInX = x - L.x — The variable changeInX created here equals x minus its value one year ago.

gen changeInX = D.x The same changeInX can be created via Stata's difference operator, D.*var*.

gen income2YearsAgo = L2.income

You can use these L. and F. notations in the list of variables for regression too:

regress gdp L.gdp L2.gdp L.unemployment L2.unemployment, vce(robust)

P4. Generating Means and Other Statistics by Individual, Year, or Group

The egen (extensions to generate) command can generate means, sums, counts, standard deviations, medians, and much more for each individual, year, or group:

qbys state: egen meanAge = mean(age) Mean of age, among all observations in each state.

qbys state year: egen meanIncome = mean(income) Mean of income, in each state and year.

qbys state year: egen totalChildren = total(children) Total number of children of people in the sample, separately in each state and year.

qbys state year: egen nPeople = count(personID) Number of nonmissing values of personID, separately in each state and year.

qbys state year: egen sdIncome = sd(income) Standard deviation, in each state and year.

qbys year: egen medianIncomeByYear = median(income) Median of income, in each year.

qbys year: egen p10IncomeByYear = pctile(income), p(12) 12th percentile of income, by year.

egen useIt = tag(state year) A variable equal to 1 in a single observation for each state-year combination, and 0 in all other observations

For many more uses of Stata's egen command, get help on "egen". One caution: When using egen, do not use "_n" or "_N", as these will cause egen to return meaningless results without any warning (Stata should really detect these and give an error message instead…).

The above methods generate values for every observation within each by-group (i.e., they create a variable with sensible values in every observation). If you just want to create a dataset of summary statistics, with one observation per by-group, try Stata's collapse command.

Q. Panel Data Statistical Methods

Q1. Fixed Effects – Using Dummy Variables

You can create dummy variables and include them as regressors. With n individuals, you should add (n-1) dummy variables. There is an easy way to do this, starting with a variable that has a unique number for each individual. In your list of variables, just put "i." in front of that variable's name, and the dummies will be made automatically during the regression (see section H1 earlier in this document). For example

regress y x1 x2 i.personid, vce(robust) Regress the dependent variable y on the independent variables x1 and x2 and on dummy that distinguish each separate person, as indicated by the person-identifier codes in the variable "personid".

This method can likewise be used to generate sets of dummy variables for any variable with identifier codes. For example:

regress y x1 x2 i.sex i.age i.city i.year, vce(robust) Regress the dependent variable y on the independent variables x1 and x2 and on dummy variables for sex, age, city, and year.

To create fixed effects and time effects, then:

regress *yvar xvars* i.*entity* i.*time*, vce(robust) Regress the dependent variable *yvar* on the independent variables listed in *xvars* and on dummy variables for the entity and for the time. There must be unique codes for each entity in the variable *entity*, and for each time in the variable *time*.

By the way, you can instead create dummy variables in the ordinary way and then list them as variables for your regression. If you need to make dummy variables for a lot of different values, a little text in the do-file editor will do the job quickly. Here is an example to emulate:

```
forvalues t = 1900/2010 {
    generate year`t' = year==`t'
}
```

This is a loop like in programming, where t goes from 1900 to 2010. Each time, the line between the curly brackets gets run. Wherever the `t' appears, Stata plugs in the value of t before running the line. (In programming lingo, t is a "local variable", called in Stata a "local macro" to avoid confusion with data variables.) To have values of t plugged in, the t needs to be encased between a left quote ` and a right quote '.

Q2. Fixed Effects – De-Meaning

Stata's "areg" command provides a simple way to include fixed effects in OLS regressions. More extensive commands are mentioned below, but the following will do for student coursework in ECON-4570/6560 Econometrics. Stata's areg command only lets you de-mean with respect to one identifier, e.g., person or year but not both – if you want fixed effects *and* time effects, you need to enter one of them using dummy variables (e.g., by including "i.year" in your *xvarlist*).

areg *yvar xvarlist*, absorb(*byvar*) vce(robust) Regress the dependent variable *yvar* on the independent variables *xvarlist* and on the dummy variables needed to distinguish each separate by-group indicated by the *byvar* variable in the absorb() option. For example the *byvar* might be the state, to include fixed effects for states. Coefficient estimates will not be reported for these fixed effect dummy variables.

Q3. Other Panel Data Estimators

Students in ECON-6570 Advanced Econometrics will need to use other panel data estimators. You will need to have declared your panel data first, as in section P2. Then:

xtreg *yvar xvarlist*, fe Fixed effects regression. The "fe" requests fixed effects estimates. This uses conventional (non-robust) standard errors.

xtreg *yvar xvarlist*, fe vce(cluster *clustervar*) Fixed effects regression again, but now with cluster-robust standard errors clustered by the specified variable. Typically the *clustervar* is the same as the *panelvar* used when tsset-ing your data (see section P2), in order to allow for arbitrary serial correlation of the error terms within each observation. *Actually* newer versions of Stata automatically compute cluster-robust standard errors (clustered by *panelvar*), for many panel data commands, if you merely specify vce(robust) – the regression output will indicate this clustering – and in the context of this specific command just specifying "robust" standard errors gives you standard errors that are the same as cluster-robust standard errors.

estimates store fixed Store estimates after running fixed effects model.

xtreg *yvar xvarlist*, re vce(robust) Random effects regression. The "re" requests random effects estimates. Here the "robust" option for variance-covariance estimation requests (Eicker-Huber-White) robust standard errors, but in the context of this specific command the resulting standard errors are in fact cluster-robust.

estimates store random   Store estimates after running fixed effects model.

hausman fixed random   Hausman test for whether random effects model is appropriate instead of fixed effects model. If the test is rejected, this suggests that the coefficient estimates are inconsistent when fixed effects are not used.

xtreg *yvar xvarlist*, mle vce(robust)   Random effects again, but now using the maximum-likelihood random-effects model.

A between-effects model ("be") is also available to estimate differences between the averages-over-time for each individual, and a population-averaged ("pa") model is also available.

See also the "newey" command in section U7, to account for serial correlation in error terms.

Stata has many other estimation commands for panel data, including dynamic panel data models such as Arellano-Bond estimation. A. Colin Cameron and Pravin K. Trivedi's book *Microeconometrics Using Stata* and Christopher Baum's book *An Introduction to Modern Econometrics Using Stata* show some of these commands. There are also panel equivalents of many other models, for example fixed and random effects versions of the logit model.

Q4. Time-Series Plots for Multiple Individuals

When making plots in Stata, the by(*varlist*) option lets you make a separate plot for each individual in the sample. For example, you could do:

```
sort companyid year
scatter employment year, by(companyid) connect(l)
```

This would make plots of each company's employment in each year, with a separate plot for each company, arranged in a grid. However, you might prefer to overlay these plots in a single graph. You could do this as follows:

```
tsset
xtline employment , overlay
```

The "xtline" command with the "overlay" option puts all companies' plots in a single graph, instead of having a separate plot for each company.

See also section E4, which talks briefly about graphing.

R. Probit and Logit Models

probit *yvar xvarlist*, vce(robust)   Probit regression.

logit *yvar xvarlist*, vce(robust)   Logit regression.

For probit and logit models, you have to be careful how you interpret the estimated coefficients:

R1. Interpreting Coefficients in Probit and Logit Models

When you use probit or logit models, or any other nonlinear models, you have to be careful about interpreting the estimated coefficients. Here let's consider how to carry out correct interpretation for probit and logit models. First, do not just look at a coefficient estimate and say, "When X gets larger by 1, the probability that Y equals 1 gets larger by (some amount)." To make statements like this you have to compute the fitted probabilities, for specific values of the regressors. Second, it can even be wrong to say that the probability increases or decreases with X according to the sign of X's coefficient estimate being positive or negative – this kind of statement may be wrong if the variable X has interaction terms in the model. Therefore it is important to have ways to compute the predicted probabilities for different possible types of individuals in the sample, and to compare how those predicted probabilities change when the value of a regressor changes. A first command that helps with this gives you the predicted probabilities for each separate individual in the sample, given that individual's regressors:

predict probOfOutcome, pr   Compute the predicted probability that the dependent variable is 1, separately for each observation.

However, you might want to compute the probability that Y equals 1 for a hypothetical individual, for which the values of the regressors are not in the data. How can you do this? One way is to write out the calculation of the probability using the display command. For example after the following probit and logit commands, the display command lets you enter a formula for the whole fitted regression equation (including the cumulative normal or logistic functions) to calculate the fitted probability, in this case when the variable pi_rat equals 0.3 and the variable black equals 0. Note the use of _b[*varname*] to mean the estimated coefficient of the variable *varname* in the most recent estimation command.

Here is an example for the probit model:

probit denyMortgage pi_rat black, vce(robust)   Estimate a probit model.

scalar z = _b[_cons]+_b[pi_rat]*0.3+_b[black]*0   Calculate $x_i'\hat{\beta}$ when the x-variables equal 0.3 and 0 respectively.

display normprob(scalar(z))   Calculate the fitted probit probability that the y-variable equals 1 when the x-variables equal 0.3 and 0 respectively.

Here is an example for the logit model:

logit denyMortgage pi_rat black, vce(robust)   Estimate a logit model.

scalar z = _b[_cons]+_b[pi_rat]*0.3+_b[black]*0   Calculate $x_i'\hat{\beta}$ when the x-variables equal 0.3 and 0 respectively.

display 1/(1+exp(-scalar(z)))   Calculate the fitted logit probability that the y-variable equals 1 when the x-variables equal 0.3 and 0 respectively.

In particular, one usually wants to estimate how much difference it makes if an x-variable is higher or lower by some amount. The difference made is a function of the values of any other regressors. For example, if a patient receives an anticancer drug versus does not receive it, the increase predicted in probability of patient survival depends on any other regressors like health and age of the patient. Therefore you have to compute two probabilities that y=1, in which you plug in (a) the value(s) of the variable(s) of interest in alternative cases (such as receiveDrug = 1 versus receiveDrug = 0), and (b) values of all other variables. For (b), there are two common approaches. The first approach is to use the mean values of all other regressors in the sample, the idea being that then your computed probabilities pertain to a hypothetical average individual who is, it is implicitly assumed, typical. This approach is *not* desirable because individuals in the sample could all be far from average and experience very different effects from what you compute. The second and better approach is to use the actual values of all other regressors and compute a separate estimated increase in the probability for each separate individual in the sample. Then you can give the ranges of the predicted increases in probability from lowest to highest among all individuals in the sample, you can graph how the increases differ depending on values of other variables, and you can report the average increase in probability across all individuals in the sample. Stata makes it easy to (1) use the means of other variables, or (2) compute the average increase across all individuals in the sample:

probit y x1 x2 x3 ..., vce(robust)   Estimate a probit model, or you could use a logit instead.

margins , at(x1=1) Compute the average probability (approach 2) that y=1 for *all* individuals in the sample, for the hypothetical possibility in which they all had x1=1.

margins , at(x1=(0 1)) Compute the average probability (approach 2) that y=1 for *all* individuals in the sample, for two different hypothetical cases: if they all had x1=0, and separately if they all had x1=1.

margins , at(x1=(0 1)) post Same as above, but allow the results to be used with post-estimation commands like "test" and "lincom".

margins, coeflegend After using the "post" option, check the variable names to use with commands like "test" and "lincom". The variable names can be hard to guess, e.g., "2._at#1.x1".

test _b[2._at#1.x1] = _b[1._at#1.x1] An example hypothesis test for the average (approach 2) increases in probabilities, in this case comparing the increases in probabilities if x1=1 versus if x1=0. Here the null hypothesis is that there is no difference.

margins if ..., at(x1=(0 1)) When you compute the average probabilities (approach 2), average across only a given type of individual in the sample, as specified by an if-statement such as "if x3==20" or "if female & old".

margins , at(x1=(0 1)) atmeans Compute probabilities using approach 1 instead of approach 2; that is, use the mean values of the other regressors instead of the actual values.

If you have interaction effects in your model, you will need to specify your regressors using a special notation so that Stata knows how to compute marginal effects. See section H1 of this document to see how to use notations like "i." to create dummy variables, "c." to specify continuous variables, and "#" and "##" to specify interaction effects. Then you can check the marginal effects of variables that are interacted or are involved in polynomials. For example:

probit y i.race i.female i.race#i.female c.age c.age#c.age i.female#c.age, vce(robust) Estimate a probit model with dummies for race and female, dummies for race-female interactions, age, age squared, and female times age.

margins , race Compute the average probability (approach 2) that y=1 for *all* individuals in the sample, for the hypothetical possibility in which each person were of the same race – doing so for every race that occurs in the data.

margins , female Do the same for the hypothetical possibility that each person were of the same sex.

margins , race female Do both of the above.

margins , race#female Do the same for the hypothetical possibility that each person were of the same race *and* the same sex.

margins , at(age=(20 30 40 50 60 70)) Compute the average probability (approach 2) that y=1 for *all* individuals in the sample, for the hypothetical possibility in which each person were of the same age – doing so for each age 20, 30, 40, 50, 60, and 70.

S. Other Models for Limited Dependent Variables

In Stata's help, you can easily find commands for models such as: Tobit and other censored regression models, truncated regression models, count data models (such as Poisson and negative binomial), ordered response models (such as ordered probit and ordered logit), multinomial response models (such as multinomial probit and multinomial logit), survival analysis models, and many other statistical models. Listed below are commands for a few of the most commonly used models.

With these models, familiarize yourself with Stata's margins command, and use margins after estimation to interpret the estimation results (section R1 shows how after probit and logit commands). This is important because otherwise it is all too common that analysts make incorrect interpretations.

S1. Censored and Truncated Regressions with Normally Distributed Errors

If the error terms are normally distributed, then the censored regression model (Tobit model) and truncated regression model can be estimated as follows.

tobit *yvar xvarlist*, vce(robust) ll(#) Estimate a censored regression (Tobit) model in which there is a lower limit to the values of the variables and it is specified by #. You can instead, or in addition, specify an upper limit using ul(#). If the censoring limits are different for different observations then use the "cnreg" command instead, or more generally if you also have data that are known only to fall in certain ranges then use the "intreg" command instead.

truncreg *yvar xvarlist*, vce(robust) ll(#) Estimate a truncated regression model in which there is a lower limit to the values of the variables and it is specified by #. You can instead, or in addition, specify an upper limit using ul(#).

Be careful that you really do think the error terms are close to normally distributed, as the results can be sensitive to the assumed distribution of the errors. There are also common models for truncated or censored data fitting particular distributions, such as zero-truncated count data for which no data are observed when the count is zero or right-censored survival times; you can find many such models in Stata.

S2. Count Data Models

The Poisson and negative binomial models are two of the most common count data models.

poisson *yvar xvarlist*, vce(robust) Estimate a model in which a count dependent variable *yvar* results from a Poisson arrival process, in which during a period of time the Poisson rate of "arrivals" (that each add 1 to the count in the y-variable) is proportional to exp($\mathbf{x}_i'\boldsymbol{\beta}$) where $\mathbf{x}_i$ includes the independent variables in *xvarlist*.

nbreg *yvar xvarlist*, vce(robust) Estimate a negative binomial count data model. (This allows the variance of y to exceed the mean, whereas the Poisson model assumes the two are equal.)

As always, see the Stata documentation and on-line help for lots more count data models and options to commands, and look for a book on the subject if you need to work with count data seriously.

S3. Survival Models (a.k.a. Hazard Models, Duration Models, Failure Time Models)

To fit survival models, or make plots or tables of survival or of the hazard of failure, you must first tell Stata about your data. There are a lot of options and variants to this, so look for a book on the subject if you really need to do this. A simple case is:

stset *survivalTime*, failure(*dummyEqualToOneIfFailedElseZero*) Tell Stata that you have survival data, with each individual having one observation. The variable *survivalTime* tells the elapsed time at which each individual either failed or ceased to be studied. It is the norm in survival data that some individuals are still surviving at the end of the study, and hence that the survival times are censored from above, i.e., "right-censored." The variable *dummyEqualToOneIfFailedElseZero* provides the relevant information on whether each option failed during the study (1) or was right-censored (0).

sts graph , survival yscale(log)   Plot a graph showing the fraction of individuals surviving as a function of elapsed time. The optional use of "yscale(log)" causes the vertical axis to be logarithmic, in which cases a line of constant (negative) slope on the graph corresponds to a hazard rate that remains constant over time. Another option is by(*groupvar*), in which case separate survival curves are drawn for the different groups each of which has a different value of *groupvar*. A hazard curve can be fitted by specifying "hazard" instead of "survival".

streg *xvarlist*, distribution(exponential) nohr vce(robust)   After using stset, estimate an exponential hazard model in which the hazard (Poisson arrival rate of the first failure) is proportional to exp($\mathbf{x}_i'\boldsymbol{\beta}$) where $\mathbf{x}_i$ includes the independent variables in *xvarlist*. Other common models make the hazard dependent on the elapsed time; such models can be specified instead by setting the distribution() option to weibull, gamma, gompertz, lognormal, loglogistic, or one of several other choices, and a strata(*groupvar*) option can be used to assume that the function of elapsed time differs between different groups.

stcox *xvarlist*, nohr vce(robust)   After using stset, estimate a Cox hazard model in which the hazard (Poisson arrival rate of the first failure) is proportional to f(elapsed time) × exp($\mathbf{x}_i'\boldsymbol{\beta}$) where $\mathbf{x}_i$ includes the independent variables in *xvarlist*. The function of elapsed time is implicitly estimated in a way that best fits the data, and a strata(*groupvar*) option can be used to assume that the function of elapsed time differs between different groups.

As always, see the Stata documentation and on-line help for lots more about survival analysis.

T. Instrumental Variables Regression

Note for Econometrics students using Stock and Watson's textbook: the term "instruments" in Stata output, and in the econometrics profession generally, means both excluded instruments *and* exogenous regressors. Thus, when Stata lists the instruments in 2SLS regression output, it will include both the Z's *and* the W's as listed in Stock and Watson's textbook.

Here is how to estimate two stage least squares (2SLS) regression models. Read the notes carefully for the first command below:

ivregress 2sls *yvar exogXVarlist* (*endogXVarlist* = *otherInstruments*), vce(robust)   Two-stage least squares regression of the dependent variable *yvar* on the independent variables *exogXVarlist* and *endogXVarlist*. The variables in *endogXVarlist* are assumed to be endogenous. The exogenous RHS variables are *exogXVarlist*, and the other exogenous instruments (not included in the RHS of the regression equation) are the variables listed in *otherInstruments*. For Econometrics students using Stock and Watson's textbook, *exogXVarlist* consists of the W's in the regression equation, *endogXVarlist* consists of the X's in the regression equation, and *otherInstruments* consists of the Z's. For Advanced Econometrics students using Hayashi's textbook, *exogXVarlist* consists of the exogenous variables in $\mathbf{z}_i$ (i.e. variables in $\mathbf{z}_i$ that are also in $\mathbf{x}_i$), *endogXVarlist* consists of the endogenous variables in $\mathbf{z}_i$ (i.e. variables in $\mathbf{z}_i$ that are not in $\mathbf{x}_i$), and *otherInstruments* consists of the excluded instruments (i.e. variables in $\mathbf{x}_i$ but not in $\mathbf{z}_i$).

ivregress 2sls *yvar exogXVarlist* (*endogXVarlist* = *otherInstruments*), vce(robust) first   Same, but also report the first-stage regression results.

ivregress 2sls *yvar exogXVarlist* (*endogXVarlist* = *otherInstruments*), vce(robust) first level(99)   Same, but use 99% confidence intervals.

predict *yhatvar*   After an ivreg, create a new variable, having the name you enter here, that contains for each observation its value of $\hat{y}_i$.

predict *rvar*, residuals   After an ivreg, create a new variable, having the name you enter here, that contains for each observation its residual $\hat{u}_i$. You can use this for residual plots as in OLS regression.

T1. GMM Instrumental Variables Regression

Students in ECON-6570 Advanced Econometrics learn about GMM instrumental variables regression. For single-equation (linear) GMM instrumental variables regression, type "gmm" instead of "2sls" in the above regression commands:

ivregress gmm *yvar exogXVarlist* (*endogXVarlist* = *otherInstruments*), vce(robust) first   GMM instrumental variables regression, showing first-stage results.

For single-equation LIML instrumental variables regression (Hayashi's section 8.6), type "liml" instead of "2sls" in the above regression commands:

ivregress liml *yvar exogXVarlist* (*endogXVarlist* = *otherInstruments*), vce(robust) first   LIML instrumental variables regression, showing first-stage results.

For more options to these commands, use the third-party "ivreg2" command described in section 8.7 of Baum's *An Introduction to Modern Econometrics using Stata* (use "ssc install ivreg2, replace" or "adopath + …" as in section J2a of this document). Multi-equation GMM instrumental variables regression is supported in Stata using the "gmm" command – see section V1 below (and see the manual entry [R] gmm, and read the Remarks section there, for an example of how to carry out multi-equation GMM IV regression).

After estimating a regression with instrumental variables, a J-test of overidentifying restrictions can be carried out as follows (for an example see section 8.6 of Baum's text). This requires installing the third-party "overid" command (use "ssc install ivreg2, replace" or "adopath + …" as in section J2a of this document):

overid   Carry out an overidentifying restrictions test after ivregress or ivreg2.

Also, a J-test is automatically carried out when using ivreg2.

To test a subset of the overidentifying restrictions, via a C-test (Hayashi p. 220), use the ivreg2 command with the list of variables to be tested in the orthog() option.

ivreg2 *yvar exogXVarlist* (*endogXVarlist* = *otherInstruments*), vce(robust) gmm orthog(*vars*)

After this GMM instrumental variables regression, an orthogonality

C-test is carried out only for the variables *vars* (if *vars* involves multiple variables then separate their names with spaces).

For a heteroskedasticity test after ivregress or ivreg2 (or also after regress), use the third-party ivhettest command (use "ssc install ivreg2, replace" or "adopath + …" as in section J2a of this document). The Pagan-Hall statistic reported is most robust to assumptions; see section 8.9 of Baum's text.

ivhettest Carry out a heteroskedasticity test.

Get help on ivhettest for options if you want to restrict the set of variables used in the auxiliary regression ($\boldsymbol{\psi}_i$ in Hayashi's section 2.7).

T2. Other Instrumental Variables Models

Some other models have been developed that accommodate instrumental variables methods. For probit models (0-1 dependent variables), see Stata's "ivprobit" command. For tobit models (with values above or below a threshold reported as the threshold value), see Stata's "ivtobit" command. Panel data estimators such as the Arellano-Bond model are available. Also, nonlinear GMM models in general can be estimated using Stata's "gmm" command.

For panel data instrumental variables methods, see A. Colin Cameron and Pravin K. Trivedi's book *Microeconometrics Using Stata* (chapter 9), or Christopher Baum's book *An Introduction to Modern Econometrics Using Stata*.

U. Time Series Models

First tsset your data as in section P above, and note how to use the lag (and lead) operators as described in section P.

U1. Autocorrelations

corrgram *varname* Create a table showing autocorrelations (among other statistics) for lagged values of the variable *varname*.

corrgram *varname*, lags(#) noplot You can specify the number of lags, and suppress the plot.

correlate x L.x L2.x L3.x L4.x L5.x L6.x L7.x L8.x Another way to compute autocorrelations, for x with its first eight lags.

correlate L(0/8).x This more compact notation also uses the 0th through 8th lags of x and computes the correlation.

correlate L(0/8).x, covariance This gives autocovariances instead of autocorrelations.

U2. Autoregressions (AR) and Autoregressive Distributed Lag (ADL) Models

regress y L.y, vce(robust) Regress y on its 1-period lag, with robust standard errors.

regress y L.y L2.y, vce(robust) Regress y on its first 2 lags, with robust standard errors.

regress y L(1/4).y, vce(robust) Regress y on its first 4 lags, with robust standard errors.

regress y L.y L.x1 L.x2, vce(robust) Regress y on the 1-period lags of y, x1, and x2, with robust standard errors.

regress y L(1/5).y L(1/4).x L.w, vce(robust) Regress y on its first 5 lags, plus the first 4 lags of x and the first lag of w, with robust standard errors.

test L2.x L3.x L4.x Hypothesis tests work as usual.

regress y L.y if tin(1962q1,1999q4), vce(robust) The "if tin(…)" used here restricts the sample to times in the specified range of dates, in this case from 1962 first quarter through 1999 fourth quarter.

U3. Information Criteria for Lag Length Selection

To get BIC (Bayes-Schwartz information criterion) and AIC (Akaike information criterion) values after doing a regression, use the "estat ic" command:

estat ic — Display the information criteria AIC and BIC after a regression.

To include BIC and AIC values in tables of regression results, you could use the "eststo" and "esttab" commands described in section J2 (if you have trouble with the eststo command below read section J above):

```
eststo m1: regress y L.y, vce(robust)
eststo m2: regress y L(1/2).y, vce(robust)
esttab m1 m2, scalars(bic aic)
```

After storing regression results, you can make a table of regression results reporting the BIC and AIC.

```
esttab m1 m2, b(a3) se(a3) star(+ 0.10 * 0.05 ** 0.01 *** 0.001) r2(3) ar2(3) scalars(F bic aic) nogaps
```

Here the BIC and AIC are displayed as part of a near-publication quality table as described in section J2c.

To speed up the process of comparing alternative numbers of lags, you could use a "forvalues" loop in your do-file editor. For example:

```
forvalues lags = 1/6 {
    eststo m`lags': regress y L(1/`lags').y, vce(robust)
}
esttab m1 m2 m3 m4 m5 m6, stats(bic aic)
```

U4. Augmented Dickey Fuller Tests for Unit Roots

dfuller y — Carry out a Dickey-Fuller test for nonstationarity, checking the null hypothesis (in a one-sided test) that y has a unit root.

dfuller y, regress — Show the associated regression when doing the Dickey-Fuller test.

dfuller y, lag(2) regress — Carry out an augmented Dickey-Fuller test for nonstationarity using two lags of y, checking the null hypothesis that y has a unit root, and show the associated regression.

dfuller y, lag(2) trend regress — As above, but now include a time trend term in the associated regression.

For panel data unit root tests, see Stata's "xtunitroot" command.

U5. Forecasting

regress y L.y L.x — After a regression…

tsappend, add(1) — Add an observation for one more time after the end of the sample. (Use add(#) to add # observations.) Use browse after this to check what happened. …

predict yhat, xb — Then compute the predicted or forecasted value for each observation.

predict rmsfe, stdf — And compute the standard error of the out-of-sample forecast.

If you want to compute multiple pseudo-out-of-sample forecasts, you could do something like this:

```
gen actual = y
gen forecast = .
gen rmsfe = .
forvalues p = 30/50 {
    regress y L.y if t<`p'
    predict yhatTemp, xb
    predict rmsfeTemp, stdf
    replace forecast = yhatTemp if t==`p'+1
    replace rmsfe = rmsfeTemp if t==`p'+1
    drop yhatTemp rmsfeTemp
}
gen fcastErr = actual - forecast
tsline actual forecast      Plot a graph of actual y versus forecasts made using prior data.
summarize fcastErr          Check the mean and standard deviation of the forecast errors.
gen fcastLow = forecast - 1.96*stdf    Low end of 95% forecast interval assuming there are
                               normally distributed and homoskedastic errors (otherwise the 1.96
                               would not be valid).
gen fcastHigh = forecast + 1.96*stdf    High end of 95% forecast interval assuming there are
                               normally distributed and homoskedastic errors (otherwise the 1.96
                               would not be valid).
tsline actual fcastLow forecast fcastHigh if forecast<.    Add forecast intervals to the graph of
                               actual versus forecast values of the y-variable.
```

U6. Break Tests

U6a. Breaks at Known Times

Testing for a break at a *known* time is simple. To do so, begin with a regression model in which there is no break. Then (1) create a dummy variable that equals 0 at times up to the break and 1 at all later times; (2) for any variables whose coefficients might change at the time of the break, including the constant term, interact them with the dummy variable and add the interactions to the model, so that the model includes the dummy variable (to allow the constant term to change) and the multiples of the original variables times the dummy (to allow the coefficients of the other variables to change); and (3) test the null hypothesis that all of the variables added in step 2 have coefficients of zero by using Stata's "test" command (section I2, page 14). This test is often called a Chow test.

Breaks at a known time are particularly easy to test for using Stata version 14. With Stata 14, you can first estimate a time series regression without a break, then use a command like this:

```
estat sbknown, break(tq(1973q3))        After a time series regression, test for a break in all
                               coefficients, with the break occurring immediately after quarter 3 of
                               1973.
estat sbknown, break(tq(1973q3) tq(1979q2))  After a time series regression, test for two breaks in
                               all coefficients, with the first break occurring immediately after
                               quarter 3 of 1973, and the second occurring immediately after
                               quarter 2 of 1979.  Additional breaks can be indicated with spaces in
                               between.
estat sbknown, break(tq(1973q3)) varlist(varlist, constant)      After a time series regression, test
                               for a break in selected coefficients, with the break occurring
                               immediately after quarter 3 of 1973.  Only allow the constant term
                               and the coefficients of the variables in varlist to have their
                               coefficients change at the time of the break.  Omit the ", constant" if
                               you do not want the constant term to change.
```

U6b. Breaks at Unknown Times

Testing for a break at an *unknown* time is more complex. (First read about Breaks at Known Times, above.) To test for a break at an unknown time, you must choose a time period when the break can be checked for, by excluding from consideration for the break ("trimming") a fraction of times at the beginning and end of the time series data sample. Then (1) a known-break test is estimated at each possible time in the range and the Chow test statistic is computed; (2) the time that yields the largest test statistic is selected as the candidate for when the break may have occurred; and (3) the test statistic is evaluated to get a *p*-value, using the appropriate statistical distribution which is *not* the same as the *p*-value from a known break test. In Stata 13 and earlier, you actually have to do all these steps yourself, as shown below. In Stata 14, there is a built-in command to do the steps for you. First you estimate the time series regression without a break, and then you use a Stata 14 command such as:

estat sbsingle — After a time series regression, test for a break at an unknown time, trimming 15% of observations at the start and 15% at the end of the sample.

estat sbsingle, trim(11) — Same, but trim 11% at the start and 11% at the end.

estat sbsingle, ltrim(5) rtrim(20) — Same, but trim 5% at the start and 20% at the end.

estat sbsingle, varlist(*varlist*, constant) — Same with 15% and 15%, but only allow the constant term and the coefficients of the variables in *varlist* to have their coefficients change at the time of the break. Omit the ", constant" if you do not want the constant term to change.

Using the largest test statistic obtained in step 2, the test statistic is called a "Quandt likelihood ratio statistic" or a "sup-Wald statistic."

If you are using Stata 13 or earlier, here is an example of what you could do. You will need to determine the first and last times that are candidates for the break after trimming. In the example below, the first and last times are arbitrarily given as -25 and 45, but you will need to figure out the correct values for your problem, by trimming say 15%. This Stata code should be used in the do-file editor:

```
forvalues k = -25/45 {  // Valid if break candidates after trimming are -25 to 45, change as needed!
  generate late = t>`k'
  generate lateL1y = late * L.y
  generate lateL2y = late * L2.y
  regress y L.y L2.y late lateL1y lateL2y
  test late lateL1y lateL2y
  display "`k'" r(F)
  drop late lateL1y lateL2y
}
```

You could improve the above code. Putting "quietly" in front of a command suppresses its output. You could also record the highest F-statistic so far from each F-test – to do, so (a) before the forvalues loop you could put these commands:

```
scalar highestFSoFar = 0
scalar kAtHighestF = -99
```

Then (b) inside the forvalues loop, after the test command, you could put:

```
if r(F)>scalar(highestFSoFar) {
  scalar highestFSoFar = r(F)
  scalar kAtHighestF = `k'
}
```

Then (c) at the end you could put:

```
display "Highest F: " scalar(highestFSoFar)
display "k with highest F: " scalar(kAtHighestF)
```

If you are using a Stata version preceding 14, this is a hassle, but it has the benefit of getting you used to valuable Stata programming techniques.

U7. Newey-West Heteroskedastic-and-Autocorrelation-Consistent Standard Errors

newey y x1 x2, lag(#)   Regress y on x1 and x2, using heteroskedastic-and-autocorrelation-consistent (Newey-West) standard errors assuming that the error term times each right-hand-side variable is autocorrelated for up to # periods of time. (If # is 0, this is the same as regression with robust standard errors.) A rule of thumb is to choose # = 0.75 * T^(1/3), rounded to an integer, where T is the number of observations used in the regression (see the text by Stock and Watson, page 607). If there is strong serial correlation, # might be made more than this rule of thumb suggests, while if there is little serial correlation, # might be made less than this rule of thumb suggests.

U8. Dynamic Multipliers and Cumulative Dynamic Multipliers

If you estimate the effect of multiple lags of X on Y, then the estimated effects on Y are effects that occur after different amounts of time. For example:

```
newey ipGrowth L(1/18).oilshock, lag(7)
```

Here, the growth rate of industrial production (ipGrowth) is related to the percentage oil price increase or 0 if there was no oil price increase (oilshock) in 18 previous months. This provides estimates of the effects of oil price shocks after 1 month, after 2 months, etc. The cumulative effect after 6 months then could be found by:

```
lincom L1.oilshock + L2.oilshock + L3.oilshock + L4.oilshock + L5.oilshock + L6.oilshock
```

Confidence intervals and *p*-values are reported along with these results.

You could draw by hand a graph of the estimated effects versus the time lag, along with 95% confidence intervals. You could also draw by hand a graph of the estimated cumulative effects versus the time lag, along with 95% confidence intervals. Making the same graphs in an automated fashion in Stata is a little more painstaking, but see my Stata do-file for Stock and Watson's exercise E15.1 for an example.

V. System Estimation Commands

Advanced Econometrics students work with estimators for systems of equations. Here is a brief introduction to some pertinent system estimation commands. Note that the 3SLS, SUR, and multivariate regression commands all assume conditionally homoskedastic errors, and have no "vce(robust)" option. To allow for heteroskedasticity in system estimation, you need to use Stata's "gmm" command, which is flexible and allows you to choose your methods. To refer to a coefficient in an equation after estimation, for the lincom, test, and testnl commands, see the example test command in section V2 below.

Remember, the advantage of this sort of system estimation is efficiency, but the disadvantage is risk of inconsistency. Inconsistency in (potentially) all equations occurs if assumptions are violated for any one of the equations (also if cross-equation orthogonality is wrong in SUR models). Contrary to a common misconception, single-equation estimates are consistent, as long as the requisite assumptions are satisfied. The efficiency advantage may be worth the risk in many cases, but do beware of the risk.

V1. GMM System Estimators

For the generalized method of moments system estimator, Stata's gmm command allows very flexible specification of models, instrumentation, and estimation methods. In fact, the command allows estimation of nonlinear as well as linear models (see section W). For details, see the Stata manuals for [R] gmm.

V2. Three-Stage Least Squares

Read the Stata manual's entry for the reg3 command to get a good sense of how it works. Here are some examples drawn from the Stata manual:

reg3 (consump wagepriv wagegovt) (wagepriv consump govt capital1) Estimate a two-equation 3SLS model in which the two dependent variables, consump and wagepriv, are assumed to be endogenous. (Dependent variables are assumed to be endogenous unless you list them in the exog() option.) The instruments consist of all other variables: wagegovt, govt, capital1, and the constant term. Note that the consumption equation estimates will be the same as in 2SLS, since that equation is just identified.

test [consump]wagegovt [wagepriv]capital1 The test, lincom, and testnl commands work fine after multi-equation estimations, but you have to specify each coefficient you are talking about by naming an equation as well as a variable in an equation. Thus, "[consump]wagegovt" refers to the coefficient of the variable wagegovt in the equation named consump. Stata by default names equations after their dependent variables. For nonlinear hypothesis tests, refer to coefficients for example using "_b[consump:wagegovt]".

reg3 (qDemand: quantity price pcompete income) (qSupply: quantity price praw) , endog(price) Estimate a two-equation model, naming the equations qDemand and qSupply since they have the same dependent variable, and treat price as endogenous. Treat the other three regressors and the constant as exogenous.

V3. Seemingly Unrelated Regression

Read the Stata manual's entry for the sureg command to get a good sense of how it works. Here is an example drawn from the Stata manual:

sureg (price foreign mpg displ) (weight foreign length), corr Estimate a two-equation SUR model. The "corr" option causes the cross-equation correlation matrix of the residuals to be displayed, along with a test of the null hypothesis that the error terms have zero covariance between equations.

V4. Multivariate Regression

mvreg headroom trunk turn = price mpg displ gear_ratio length weight, corr Estimate three regression equations, the first with headroom as the dependent variable, the second with trunk (space) as the dependent variable, the third with turn(ing circle) as the dependent variable. In each case, the six variables listed on the right-hand side of the equals sign are used as regressors. The "corr" option causes the cross-equation correlation matrix of the residuals to be displayed, along with a test of the null hypothesis that the error terms have zero covariance between equations. The same estimates could be obtained by running three separate regressions, but this also analyzes correlations of the error terms and makes it possible to carry out cross-equation tests afterward.

W. Flexible Nonlinear Estimation Methods

Advanced Econometrics students work with various other estimation methods discussed here.

W1. Nonlinear Least Squares

Read the Stata manual entry [R] nl to get a good sense of how the nl command works. There are several ways in which to use nonlinear regression commands. Here is an example showing how to estimate a nonlinear least squares model for the equation $y_i = \beta_1 + \beta_2 e^{\beta_3 x_i} + \varepsilon_i$:

nl (y = {b1}+{b2=1}*exp({b3}*x)) Estimate this simple nonlinear regression.

Look at the above line to understand its parts. The "nl" is the name of the nonlinear regression command. After that is an equation in parentheses. The left side of the equation is the dependent variable. The right side is the conditional expectation function, in this case $\beta_1 + \beta_2 e^{\beta_3 x_i}$. The terms in curly brackets are the parameters to be estimated, which we have called b1, b2, and b3. Stata will try to minimize the sum of squared errors by searching through the space of all possible values for the parameters. However, if we started by estimating β_2 as zero, we might not be able to search well – at that point, the estimate of β_3 would have no effect on the sum of squared errors. Instead, we start by estimating β_2 as one, using the "{b2=1}". The "=1" part tells Stata to start at 1 for this parameter.

Often you may have a linear combination of variables, as in the formula $y_i = \alpha_1 + \alpha_2 e^{\beta_1 x1_i + \beta_2 x2_i + \beta_3 x3_i + \beta_4 x4_i} + \varepsilon_i$. Stata has a shorthand notation, using "xb: *varlist*", to enter the linear combination:

nl (y = {a1}+{a2=1}*exp({xb: x1 x2 x3 x4})) Estimate this nonlinear regression.

After a nonlinear regression, you might want to use the "nlcom" command to estimate a nonlinear combination of the parameters.

W2. Generalized Method of Moments Estimation for Custom Models

Stata has powerful general-purpose GMM commands (including for nonlinear models). See the Stata manual entry [R] gmm.

W3. Maximum Likelihood Estimation for Custom Models

Stata has powerful general-purpose maximum likelihood estimation commands. See the Stata manual entry [R] ml.

X. Data Manipulation Tricks

In real statistical work, often the vast majority of time is spent preparing the data for analysis. Many of the commands given above are very useful for data preparation – see particularly sections F, M, O, and P above. This section describes several more Stata commands that are extremely useful for getting your data ready to use.

Make sure you organize all your work in a do-file (or multiple do-files). If you are using Stata 11 or earlier, your do-file should start with clear and set memory if needed. In any case, the do-file should next read in the data, then do anything else like generate variables, merge datasets, reshape the data, use tsset, et cetera. Finally, if desired the do-file should save the prepared data in a separate file. If running this do-file does not take long, just run it each time you want to do statistical analyses – so it reads in the data and does all your analyses in one click. If the do-file takes a long time to prepare the data, save a prepared data file at the end so you can just read in the data file when needed.

X1. Combining Datasets: Adding Rows

Suppose you have two datasets, typically with (at least some of) the same variables, and you want to combine them into a single dataset. To do so, use the append command:

append using *filename* Appends another dataset to the end of the data now in memory. You must have the other dataset saved as a Stata file. Variables with the same name will be placed in the same column; for example, if you have variables named "cusip" and "year" and the other dataset has variables with the same names, then all the "cusip" values in the appended data will be in the new rows of the cusip variable, while all of the "year" values in the appended data will be in the new rows of the year variable.

X2. Combining Datasets: Adding Columns

Suppose you have two datasets. The "master" dataset is the one now in use ("in memory"), and you want to add variables from a "using" dataset in another file. The goal is to use an identification code (one or more variables) to determine which rows match up across the two files, and bring in the extra columns of data. Stata's "merge" command does this.

To ensure a good match and add the right information, you have to get several issues right:

- *Identification code*. The identification code variable(s) specify what should (and should not) match. For example, you could have a variable named personID with a unique number for each person, and matching values of personID tell which row(s) in the using dataset correspond to each row in the master dataset. For another example, you could have two variables named country and year, which must *both* match for rows in the using dataset to correspond to a row in the master dataset. Any number of variables may be used in combination to create the identification code, and they must *all* match for row(s) in the using dataset to correspond to a row in the master dataset. *N.B.*: Missing values are values of the variables too, so beware: if they occur for multiple cases, they mess up your ability to get a proper match.
- *Uniqueness (or not) of identification code among master and using observations*. In the master dataset, *and* in the using dataset, there may be a different identification code in each row, or there may be multiple rows with the same identification code. For example, a dataset might contain observations for one million people, each in a separate row, each with a unique value of personID. In the same dataset, another variable named countryID may specify each person's country, but many people would share the same countryID.

Therefore each value of personID would be unique among the observations, but values of countryID would not be unique. You could match using personID as the identification code to bring more person-specific information into the dataset, or you could match using countryID as the identification code to bring more information about each person's country into the dataset.

If the identification code is unique in each dataset, then there is a one-to-one match, written "1:1". If the identification code is not unique in the master dataset, but is unique in the using dataset, then there is a many-to-one match, written "m:1". These two cases pertain to the personID and countryID examples above, if the using datasets have a unique identification code in each row. If a using dataset does not have unique identification code, then there would be a one-to-many or many-to-many match, "1:m" or "m:m".

A "1:m" match could arise if you reverse the order of the master and using datasets in the countryID match above: you start with a dataset of country information as your master dataset, and you match to the dataset of person information as your using dataset. Because the using dataset contains *multiple matching rows* for each row in the master dataset, *there will be more rows after the merge than there were in the using dataset*. The result will be the same regardless which dataset is the using dataset and which is the master dataset – subject to some caveats discussed below (if variables other than the identification code have the same name in the two datasets, or if not all observations are retained, there can be differences).

Mistakes in matching could arise if you meant to perform a 1:1 match but actually carried out a 1:m or m:1 match – then the resulting number of observations might be something other than what you intended – sometimes because of missing values in the identification code but commonly also because for some reason you have more than one entry for a code that you meant to be unique. Usually you know what kind of match *should* arise, and Stata requires you to specify this information. Indeed, this is really important to avoid horrible mistakes. *For m:m matches Stata's merge command does not do what you would expect – it does not create all the relevant pairs; instead you must use Stata's joinby command.*

- *Keeping matching, master-only, and using-only observations*. It may be that not all observations have matches across the master and using files. When no matches exist, certain observations are orphans for which it is not possible to add columns of data from the other dataset, so the added variables can only contain missing values. You may not want these orphans kept in the resulting data, particularly for orphans from the using dataset. Therefore Stata differentiates between matching (3), master-only (1), and using-only (2) observations. It creates a variable named _merge that contains the numbers 3, 1, and 2 respectively for these three cases. You can then drop observations that you do not want after the match.

  Even better, you can specify up-front to keep only observations of specific type(s). To do so, use the keep(…) option. Inside the option, specify one or more of "match", "master", and "using", separated by spaces, to cause matching, master-only, and using-only observations to be kept in the results. All other observations will be dropped in the results (though not in any data files on your hard disk).

  To ensure against mistakes, you can also use an "assert(…)" option to state that everything should match, or that there should never be orphan observations from one of the datasets. If your assertion that this is true turns out to be violated, then Stata will stop with an error message so you can check what happened and fix possible problems. Inside the assert(…) option, again specify one or more of "match", "master", and "using", separated by spaces, to assert that the results will contain only matching, master-only, and using-only observations.

- *Variables other than the identification code that are in both master and using datasets*. If a variable in the using dataset, other than the identification code, already exists in the master dataset, then it is not possible to bring that column of data into the results as an independent column (at least, not with the same variable name, and Stata just doesn't bring it in). The "update" option to the merge command causes missing values to be filled in using values in the using dataset, and the "update" and "replace" options together causes the reverse in which the using dataset's values are preferred except where they have missing values. For matching observations with variables with the same name in both datasets, if you use the update option, then _merge can take values not just of 1, 2, or 3, but also of 4 when a missing value was updated from the other dataset, or of 5 when there were conflicting non-missing values of a variable in the two datasets.
- *Reading in only selected variables*. If you only want to read in some of the variables from the using dataset, use the keepusing(*varlist*) option.

Before using the merge command, therefore, you need to go through each of the above issues and figure out what you want to do. Then you can use commands such as the following:

merge 1:1 personID using *filename* Match in observations from the using dataset, with personID as the identification code variable.

merge 1:1 country year month using *filename* Match in observations from the using dataset, with country, year, and month jointly as the identification code.

merge 1:1 personID using *filename*, keep(match master) Match in observations from the using dataset, with personID as the identification code variable, and only keep observations that match or are in the master dataset; ignore observations that are in the using dataset only.

merge 1:1 personID using *filename*, assert(match) Match in observations from the using dataset, with personID as the identification code variable, and assert that all observations in each dataset match – if they do not, stop with an error message.

merge 1:1 _n using *filename* This one-to-one merge assumes that each observation *i* in the master dataset matches to each observation *i* in the using dataset. This is *dangerous* because it's easy to mistakenly have a wrong sort order, so this is not recommended!

merge m:1 countryID using *filename* Match in observations from the using dataset, with personID as the identification code variable. This is specified as a many-to-one match, so the master dataset may contain multiple observations with the same countryID.

merge 1:m countryID using *filename* Match in observations from the using dataset, with personID as the identification code variable. This is specified as a one-to-many match, so the using dataset may contain multiple observations with the same countryID.

joinby familyID using *filename* Carry out a m:m match in which you want all pairs of matching observations. This is not what you get using "merge m:m", and you should not use “merge m:m” unless you really know what you are doing. See Stata’s help for this command for further information about options.

The merge command will display the number of resulting observations with _merge equal to 1, 2, and 3. Always check the values of _merge after merging two datasets, to avoid errors.

X3. Reshaping Data

Often, particularly with panel data, it is necessary to convert between “wide” and “long” forms of a dataset. Here is a trivially simple example:

Wide Form:

personid	income2005	income2006	income2007	birthyear
1	32437	33822	41079	1967
2	50061	23974	28553	1952

Long Form:

personid	year	income	birthyear
1	2005	32437	1967
1	2006	33822	1967
1	2007	41079	1967
2	2005	50061	1952
2	2006	23974	1952
2	2007	28553	1952

This is a trivially simple example because usually you would have many variables, not just income, that transpose between wide and long form, plus you would have many variables, not just birthyear, that are specific to the personid and don’t vary with the year.

Trivial or complex, all such cases can be converted from wide to long form or vice versa using Stata’s reshape command:

```
reshape long income, i(personid) j(year)    // Starting from wide form, convert to long form.
reshape wide income, i(personid) j(year)    // Starting from long form, convert to wide form.
```

If you have more variables that, like income, need to transpose between wide and long form, and regardless of how many variables there are that don’t vary with the year, just list the transposing variables after “reshape long” or “reshape wide”, e.g.:

```
reshape long income married yrseduc, i(personid) j(year)    Starting from wide form, convert to
                                long form.
reshape wide income married yrseduc, i(personid) j(year)    Starting from long form, convert to
                                wide form.
```

X4. Converting Between Strings and Numbers

Use the describe command to see which variables are strings versus numbers:

```
describe
```

If you have string variables that contain numbers, an easy way to convert them to numbers is to use the destring command. The tostring command works in the reverse direction. For example, if you have string variables named year, month, and day, and the strings really contain numbers, you could convert them to numbers as follows:

destring year month day, replace Convert string variables named year, month, and day, to numeric variables, assuming the strings really do contain numbers.

You could convert back again using tostring:

tostring year month day, replace Convert numeric variables named year, month, and day, to string variables.

When you convert from a string variable to a numeric variable, you are likely to get an error message because not all of the strings are numbers. For example, if a string is “2,345,678” then Stata will not recognize it to be a number because of the commas. Similar, values like “see note” or “>1000” cannot be converted to numbers. If this occurs, Stata will by default refuse to convert a string value into a number. This is good, because it points out that you need to look more closely to decide how to treat the data. If you want such non-numeric strings to be converted to missing values, instead of Stata stopping with an error message, then use the force option to the destring command:

destring year month day, replace force Convert string variables named year, month, and day, to numeric variables. If any string values do not seem to be numbers, convert them to missing values.

Like most Stata commands, these commands have a lot of options. Get help on the Stata command destring, or consult the Stata manuals, for more information.

X5. Labels

What if you have string variables that contain something other than numbers, like “male” versus “female”, or people’s names? It is sometimes useful to convert these values to categorical variables, with values 1,2,3,…, instead of strings. At the same time, you would like to record which numbers correspond to which strings. The association between numbers and strings is achieved using what are called “value labels”. Stata’s encode command creates a labeled numeric variable from a string variable. Stata’s decode command does the reverse. For example:

```
encode personName, generate(personNameN)
decode personName, generate(personNameS)
```

If you do a lot of encoding and decoding, try the add-on commands rencode, rdecode, rencodeall, and rdecodeall (outside RPI computer labs: from Stata’s Help menu, choose Search… and click the button for “Search net resources,” search for “rencode”, click on the link that results, and click “click here to install”; in my econometrics classes at RPI: follow the procedure in section J2a).

This example started with a string variable named personName, generated a new numeric variable named personNameN with corresponding labels, and then generated a new string variable personNameS that was once again a string variable just like the original. If you browse the data, personNameN will seem to be just like the string variable personName because Stata will automatically show the labels that correspond to each name. However, the numeric version may take up a lot less memory.

If you want to create your own value labels for a variable, that’s easy to do. For example, suppose a variable named female equals 1 for females or 0 for males. Then you might label it as follows:

label define femaleLab 0 "male" 1 "female" This defines a label named “femaleLab”.

label values female femaleLab This tells Stata that the values of the variable named female should be labeled using the label named “femaleLab”.

Once you have created a (labeled) numeric variable, it would be incorrect to compare the contents of a variable to a string:

summarize if country=="Canada" This causes an error if country is numeric!

However, Stata lets you look up the value corresponding to the label:

summarize if country=="Canada":countryLabel You can look up the values from a label this way. In this case, countryLabel is the name of a label, and "Canada":countryLabel is the number for which the label is “Canada” according to the label definition named countryLabel.

If you do not know the name of the label for a variable, use the describe command, and it will tell you the name of each variable’s label (if it has a label). You can list all the values of a label with the command:

label list *labelname* This lists all values and their labels for the label named *labelname*.

Stata also lets you label a whole dataset, so that when you get information about the data, the label appears. It also lets you label a variable, so that when you would display the name of the the variable, instead the label appears. For example:

label data "physical characteristics of butterfly species" This labels the data.

label variable income "real income in 1996 Australian dollars" This labels a variable.

X6. Notes

You may find it useful to add notes to your data. You record a note like this:

note: This dataset is proprietary; theft will be prosecuted to the full extent of the law.

However, notes are not by seen by users of the data unless the users make a point to read them. To see what notes there are, type:

notes

Notes are a way to keep track of information about the dataset or work you still need to do. You can also add notes about specific variables:

note income: Inflation-adjusted using Australian census data.

X7. More Useful Commands

For more useful commands, go to Stata’s Help menu, choose Contents, and click on Data management.