

Useful Stata Commands (for Stata version 10)

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, I suggest you ask fellow students 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 three very useful books: Lawrence C. Hamilton’s *Statistics with Stata updated for version 9* (recommended in past for ECON-4570 Econometrics), Christopher F. Baum’s *An Introduction to Modern Econometrics Using Stata* (recommended for ECON-6570 Advanced Econometrics), and A. Colin Cameron and Pravin K. Trivedi’s *Microeconometrics using Stata* (just released and is great and the most thorough of the three).

Throughout, estimation commands specify robust standard errors (Eicker-White heteroskedastic-consistent standard errors). This does not imply that robust rather than conventional estimates of $\text{Var}[\mathbf{b}|\mathbf{X}]$ 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

Preliminaries for RPI Dot.CIO Labs.....	4
A. Loading Data.....	4
B. Variable Lists, If-Statements, and Options	4
C. Lowercase and Uppercase Letters.....	5
D. Review Window, and Abbreviating Command Names.....	5
E. Viewing and Summarizing Data	5
E1. Just Looking	5
E2. Mean, Variance, Number of Non-missing Observations, Minimum, Maximum, Etc.	5
E3. Tabulations, Histograms, Density Function Estimates.....	5
E4. Scatter Plots and Other Plots	6
E5. Correlations and Covariances.....	6
F. Generating and Changing Variables.....	6
F1. Generating Variables	6

F2. True-False Variables.....	7
F3. Random Numbers.....	7
F4. Replacing Values of Variables.....	7
F5. Getting Rid of Variables.....	7
F6. If-then-else Formulas.....	8
F7. Quick Calculations.....	8
F8. More.....	8
G. Means: Hypothesis Tests and Confidence Intervals.....	8
G1. Confidence Intervals.....	8
G2. Hypothesis Tests.....	8
H. OLS Regression (and WLS and GLS).....	9
H1. Improved Robust Standard Errors in Finite Samples.....	9
H2. Weighted Least Squares.....	9
H3. Feasible Generalized Least Squares.....	10
I. Post-Estimation Commands.....	10
I1. Fitted Values, Residuals, and Related Plots.....	10
I2. Confidence Intervals and Hypothesis Tests.....	10
I3. Nonlinear Hypothesis Tests.....	11
I4. Computing Estimated Expected Values for the Dependent Variable.....	11
I5. Displaying Adjusted R ² and Other Estimation Results.....	12
I6. Plotting Any Mathematical Function.....	12
I7. Influence Statistics.....	12
I8. Functional Form Test.....	13
I9. Heteroskedasticity Tests.....	13
I10. Serial Correlation Tests.....	13
I11. Variance Inflation Factors.....	14
J. Tables of Regression Results.....	14
J1. Tables of Regression Results Using Stata's Built-In Commands.....	14
J2. Tables of Regression Results Using Add-On Commands.....	15
J2a. Installing or Accessing the Add-On Commands.....	15
J2b. Storing Results and Making Tables.....	16
J2c. Understanding the Table Command's Options.....	16
J2d. Wide Tables.....	16
J2e. Storing Additional Results.....	17
J2f. Saving Tables as Files.....	17
J2g. Clearing Stored Results.....	17
J2h. More Options and Related Commands.....	17
K. Data Types, When 3.3 ≠ 3.3, and Missing Values.....	17
L. Results Returned after Commands.....	18
M. Do-Files and Programs.....	18
N. Monte-Carlo Simulations.....	19
O. Doing Things Once for Each Group.....	20
P. Generating Variables for Time-Series and Panel Data.....	20
P1. Creating Time Variables that Start from a First Time and Increase by 1 at Each Observation.....	20
P2. Creating Time Variables Using a String Date.....	21
P3. Telling Stata You Have Time Series or Panel Data.....	21

P4. Lags, Forward Leads, and Differences	22
P5. Generating Means and Other Statistics by Individual, Year, or Group	22
Q. Panel Data Statistical Methods	22
Q1. Fixed Effects – Simplest Method	22
Q2. Other Panel Data Estimators	23
Q3. Time-Series Plots for Multiple Individuals	23
R. Probit and Logit Models	23
S. Other Models for Limited Dependent Variables	24
S1. Censored and Truncated Regressions with Normally Distributed Errors	24
S2. Count Data Models	24
S3. Survival Models (a.k.a. Hazard Models, Duration Models, Failure Time Models)	25
T. Instrumental Variables Regression	25
T1. GMM Instrumental Variables Regression	26
U. Time Series Models	27
U1. Autocorrelations	27
U2. Autoregressions (AR) and Autoregressive Distributed Lag (ADL) Models	27
U3. Information Criteria for Lag Length Selection	27
U4. Augmented Dickey Fuller Tests for Unit Roots	28
U5. Forecasting	28
U6. Newey-West Heteroskedastic-and-Autocorrelation-Consistent Standard Errors	28
U7. Dynamic Multipliers and Cumulative Dynamic Multipliers	29
V. System Estimation Commands	29
V1. Three-Stage Least Squares	29
V2. Seemingly Unrelated Regression	30
V3. Multivariate Regression	30
W. Other Estimation Methods	30
W1. Nonlinear Least Squares	30
X. Data Manipulation Tricks	31
X1. Combining Datasets: Adding Rows	31
X2. Combining Datasets: Adding Columns	31
X2a. Unmatched Merge	32
X2b. Matched One-to-One Merge	32
X2c. Matched Many-to-One Merge	33
X2d. Matched One-to-Many Merge	34
X2e. Matched Many-to-Many Merge	34
X3. Reshaping Data	34
X4. Converting Between Strings and Numbers	35
X5. Labels	35
X6. Notes	36
X7. More Useful Commands	36

Useful Stata (Version 10) Commands

Preliminaries for RPI Dot.CIO Labs

RPI computer labs with Stata include, as of Spring 2008: Sage 4510 and the VCC Lobby (all Windows PCs). Stata might also be available in Sage 3101 and Troy 2015 (try in the rooms to find out).

To access the Stata program, look under My Computer and open the disk drive X: (which in our classroom is named "Sage4510\$"), 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 has not been set up to know where to find Stata or even which saved files are Stata files.

To access the course disk space, go to: \\hass11.win.rpi.edu\classes\econ-6570. 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 another 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 "Connect to RCS," double-click on it, and enter your username and password. Your personal disk space will be attached as drive H. (Public RCS materials will be attached as drive P.) You will want to save Stata do-files to drive H. For handy use when logging in, you may also want to put the web address to attach the course disk space in a file on 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

set memory 100m	Sets memory available for data to 100 megabytes. Clear before using.
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 using "filename.txt"	Reads in text data.
save "filename.dta"	Saves the data.

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. Also, many of the commands below can have their names abbreviated. For example, 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, as noted below.

`browse varlist` Opens the data viewer, to look at data without changing them. Close the viewer before using other commands.

`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` Give summary information for the variables listed.

`summarize varlist, detail` Give detailed summary information for the variables listed.

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/>. To tour Stata’s graphics capabilities from inside Stata, go Stata’s Help menu, 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 the 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. True-False Variables

`gen young = age < 18` 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.
`gen old = age >= 18` If age is 18 or higher, this yields 1, otherwise this yields 0.
`gen age18 = age == 18` 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` 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.
`gen youngOrWoman = age<18 | female==1` 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.
`gen ageNot18 = age != 18` The “!=” symbol means “not equal to”.
`gen notOld = !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.

When creating true-false values, note that missing values in Stata work like infinity. So if age is missing and you “`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, note that 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.

F3. Random Numbers

`gen r = uniform()` Random numbers, uniformly distributed between 0 and 1.
`gen n = 5 + 2 * invnorm(uniform())` Normally-distributed random numbers with mean 5 and standard deviation 2. For other distributions use Stata’s menu to get help for “functions”.

F4. Replacing Values of Variables

`replace young = age < 16` Changes the value of the variable `young`, to equal 1 if and only if age is less than 16.
`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.

F5. Getting Rid of Variables

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

`clear` Gets rid of all variables, as well as other things not discussed yet (like global variables, programs, etc.).

F6. If-then-else Formulas

`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. Stata's `cond(if, then, else)` works much like Excel's `IF(if, then, else)`. In this case, suppose you have data from 1992 and 2004 only, and that 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.

F7. 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 in the cumulative 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.

F8. 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 P5. For time-series and panel data, see section P, especially the notations for lags, leads, and differences in section P4. If you need to refer to a specific observation number, use a reference like `x[3]`, meaning the value 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

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

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

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-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).

H1. 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 to be 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 Greene’s text on p. 164 (or in Hayashi p. 125 formula 2.5.5 using $d=1$). An alternative is “vce(hc3)” instead of “vce(robust)” (Greene p. 164 footnote 15 or Hayashi page 125 formula 2.5.5 using $d=2$).

H2. Weighted Least Squares

Students in ECON-6570 Advanced Econometrics learn about (variance-)weighted least squares (Greene pp. 167-169). 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:

`vwls yvar xvarlist, sd(sdvar)`

^{*} R. Davidson and J. MacKinnon, *Estimation and Inference in Econometrics*, Oxford: Oxford University Press, 1993, section 16.3.

H3. 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 heteroscedasticity model can be estimated by computing the estimated standard deviation for each group using Greene's 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 *eststd* = sqrt(*estvar*), then use this estimated standard deviation to carry out weighted least squares as shown above. 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 heteroscedasticity 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.

11. Fitted Values, Residuals, and Related Plots

predict <i>yhatvar</i>	After a regression, create a new variable, having the name you enter here, that contains for each observation its <u>fitted</u> value \hat{y}_i .
predict <i>rvar</i> , residuals	After a regression, create a new variable, having the name you enter here, that contains for each observation its <u>residual</u> \hat{u}_i .
scatter y yhat x	<u>Plot</u> variables named y and yhat versus x.
scatter resid x	It is wise to plot your residuals versus each of your x-variables. Such " <u>residual plots</u> " 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 "regression postestimation" section of the Stata manuals.

12. 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. This is an F-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. This is an F-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. This is an F-test.

13. 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 10 manual, Reference Q-Z, p. 469, for examples).

14. 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. Also Stata’s “`adjust`” command provides a powerful tool to display predicted values when the `x`-variables taken on various values (but for your own understanding, do the calculation by hand a few times before you try using `adjust`).

15. 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 r(matrixname)`”.

16. 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) that you specify. A command like this may be useful when you want to examine how a polynomial in one regressor (which here must be called x) affects the dependent variable in a regression, without specifying values for other variables.

17. 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 i th 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.

18. 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.

19. 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 multiplier.

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".

110. 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.

11. Variance Inflation Factors

Students in ECON-6570 Advanced Econometrics learn about 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.

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 done trivially, without the chance of typing wrong numbers. Stata has built-in commands for making tables, and you should try them first to see how they work, as described in section J1. In practice it will be easiest to use additional commands, which have to be installed, discussed in section J2.

To put results into Excel or Word, 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.

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>). You can then adjust the font, borderlines, etc. appropriately.

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

Here is an example of how to store results of regressions, and then use previously stored results to display a table:

```
regress y x1, robust
estimates store model1
regress y x1 x2 x3 x4 x5 x6 x7, robust
estimates store model2
regress y x1 x2 x3 x4 x6 x8 x9, 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.

`adopath + folderToLookIn`

Here, replace *folderToLookIn* with the name of the folder. It might be something like “`H:stata extensions\`”. If in doubt, under Stata’s File menu you can choose “File Name...” to look up the relevant folder name.

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. Below are some examples:

- `eststo: reg y x1 x2, robust` Regress y on x_1 and x_2 (with robust standard errors) and store the results. Estimation results will be stored with names like “est1”, “est2”, etc.
- `eststo: quietly reg y x1 x2 x3, robust` Similar to above, but “quietly” tells Stata not to display any output.
- `esttab est1 est2, se star(+ 0.10 * 0.05 ** 0.01 *** 0.001) r2 ar2 scalars(F) nogaps` Make a near-publication-quality table. You will still want to make the variable names more meaningful, change the column headings, and set up the borders appropriately.

J2c. Understanding the Table Command's Options

The `esttab` command above had a lot in it, so it may help to look at simpler versions of the command to understand how it works:

- `esttab` Displays 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` Displays 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` Gets rid of blank spaces between rows. This aids copying of tables to paste into, e.g., Word.

J2d. 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, then 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. Then you can make very wide tables with lots of columns. (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 pages on later pages.)

J2e. 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, 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.

J2f. 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:

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 Saves 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. If you want to be able to use the numbers in calculations, use the next command.

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

J2g. 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).

J2h. 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.

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 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.

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. From the left, they are: new do-file, open do-file, save, print, find in this do-file, 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 “run” and “do” icons, though, are really important. The “do” icon (the last icon) is the one you will usually use. 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!
set memory 100m // Allocate 100 mb for data.
use "L:\myfolder\myfile.dta"
* I commented out the following three lines since I'm not using them now:
/* regress income age, robust
predict incomeHat
scatter incomeHat income age */
* Now do my polynomial age analyses:
gen age2 = age^2
gen age3 = age^3
regress income age age2 age3 bachelor, robust
```

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 large variance, and when the random error terms are closer to normally distributed.) Here is a Stata command to call the above (at the end of section N) 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 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)`”, if your program returns results named “`b0`”, “`b1`”, and “`b2`”).

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, 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, 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.
```

```
qby year: generate xInFirstObservationOfYear = x[1]  “qby” is shorthand for “quietly by”.
```

```
qbys year: generate xInFirstObservationOfYear = x[1]  “qbys” sorts and then does “quietly by”.
```

See also section P5 for more ways to generate results, e.g., means or standard deviations, separately for each by-group.

P. Generating Variables for Time-Series and Panel Data

With panel and time series data, you may need to (a) generate values with reference to past or future times, and (b) generate values separately for each individual in the sample. Here are some commands to help you. Your time variable should be an integer, and should not always have gaps between numbers (e.g. years might be 1970, 1971, ..., 2006 – if they are every other year 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.).

P1. Creating Time Variables that Start from a First Time and Increase 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 = q(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 = d(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 `d(...)` and `q(...)` functions used above, you may also use `w(...)` for week, `m(...)` for month, `h(...)` for half-year, or `y(...)` for year. Inside the parentheses, you type a year followed (except for `y(...)`) by a separator (a comma, colon, dash, or word) followed by a second number. The second number specifies the day, week, month, quarter, or half-year (get help on “`tfcn`” for more information).

P2. Creating Time Variables Using a String Date

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.

P3. 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.

P4. 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, robust`

P5. 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 year: egen meanIncome = mean(income)` Mean of income, in each state and year.

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

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

For many more uses of Stata’s `egen` command, get help on “`egen`”.

Q. Panel Data Statistical Methods

Q1. Fixed Effects – Simplest Method

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 Econometrics.

`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 `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.

See also the “`newey`” command in section U6, to account for serial correlation in error terms.

Q2. 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 P3. 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 P3), in order to allow for arbitrary serial correlation of the error terms within each observation.

`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-White) robust standard errors, but again you could (and likely should) request cluster-robust standard errors instead.

`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.

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

Q3. 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, robust` Probit regression.

`logit yvar xvarlist, robust` Logit regression.

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

mfx Display “marginal effects,” i.e. dy/dx_j for each independent variable j when the variables are at their means. If you have any dummy variables as independent variables, “mfx” instead computes for each dummy variable $\Delta y/\Delta x_j$ for the change from $x_j=0$ to $x_j=1$, again while other variables are at their means.

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.

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. Actually you can tell Stata that `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 $f(\text{elapsed time}) \times \exp(\mathbf{x}_i'\boldsymbol{\beta})$ 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 GMM instrumental variables regression, add the "gmm" option at the end of 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), add the "liml" option at the end of 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 the recommended supplementary text, 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 not supported in Stata.

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 `ivhetttest` 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.

```
ivhetttest
```

Carry out a heteroskedasticity test.

Get help on `ivhetttest` for options if you want to restrict the set of variables used in the auxiliary regression (Ψ_i in Hayashi’s section 2.7).

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, robust
```

Regress *y* on its 1-period lag, with robust standard errors.

```
regress y L(1/4).y, robust
```

Regress *y* on its first 4 lags, with robust standard errors.

```
regress y L(1/4).y L(1/3).x L.w, robust
```

Regress *y* on its first 4 lags, plus the first 3 lags of *x* and the first lag of *w*, with robust standard errors.

```
regress y L.y L.x1 L.x2, robust
```

Regress *y* on the 1-period lags of *y*, *x1*, and *x2*, with robust standard errors.

```
test L2.x L3.x L4.x
```

Hypothesis tests work as usual.

```
regress y L.y if tin(1962q1,1999q4), 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 and AIC 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 “`estimates table`” command:

```
regress y L.y, robust
```

```
estimates store y1
```

```
regress y L(1/2).y, robust
estimates store y2
estimates table y1 y2, stats(bic aic)
```

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

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 {
    regress y L(1/\lags').y, robust
    estimates store y'\lags'
}
estimates table y1 y2 y3 y4 y5 y6, 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.

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, stdp
```

And compute the standard error of the out-of-sample prediction or forecast.

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

```
gen yhat = .
gen rmsfe = .
forvalues p = 30/50 {
    regress y L.y if t<`p', robust
    predict yhatTemp, xb
    predic rmsfeTemp, stdp
    replace yhat = yhatTemp if t==`p'+1
    replace rmsfe = rmsfeTemp if t==`p'+1
    drop yhatTemp rmsfeTemp
}
scatter yhatTemp t
```

U6. Newey-West Heteroskedastic-and-Autocorrelation-Consistent Standard Errors

```
newey y x1 x2, lag(#)
```

Regress y on x_1 and x_2 , 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.

U7. 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 these commands all assume conditionally homoskedastic errors. To refer to a coefficient in an equation after estimation, for the lincom, test, and testnl commands, see the example test command in section V1 below.

V1. 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=nt 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.

V2. 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.

V3. 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. Other Estimation Methods

Advanced Econometrics students work various other estimation methods discussed here.

W1. Nonlinear Least Squares

Read the Stata manual’s entry for the nl command to get a good sense of how it 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 least squares 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

$y_i = \alpha_1 + \alpha_2 e^{\beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \beta_4 x_{4i}} + \varepsilon_i$. 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.

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), starting with clear and set memory if needed, then reading in the data, then doing anything else like generating variables, merging datasets, reshaping the data, using `tsset`, et cetera, and then possibly saving the prepared data in a separate file. If running this do-file does not take too long, you can just run it each time you want to do statistical analyses. If it takes a long time, save a prepared data file at the end so that 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. You might carry out (i) a one-to-one merge, e.g., with two cross-sectional or panel datasets on the same 1,000 people, one with variables on age and education, the other with variables on employment and earnings; (ii) a many-to-one merge, e.g., with a master cross-sectional or panel dataset on 1,000 people along with a year or grocery-store-shopped-in variable for each observation, plus a using dataset with the US GDP in each year or with information about the location and size of each grocery store; or (iii) a one-to-many merge, e.g., with a dataset listing 50 US states along with characteristics of each state, plus a dataset listing many government agencies in each state. To add the columns in the latter dataset to the former dataset, use Stata’s `merge` command, as in the examples below.

When you merge two datasets, you will want to ensure that the rows of data match up properly. You should do so using a so-called “matched merge.” In (i), you would need a person id variable that is a unique value for each person in the dataset, with the same values used in the two different datasets. If the datasets involve panel data, you would also need a time variable (such as the year) that also can be matched when comparing the two datasets. In (ii), you would need a year or company identifier that can be matched when comparing the two datasets. In (iii), you would need the state identifier variable in each dataset (the state identifier might be a two-letter state abbreviation or an appropriately defined number for each state). If you have two datasets for which the *i*th row in one corresponds to the *i*th row in the other, you can do an “unmatched merge” in which corresponding rows are assumed to match up. Unmatched merges are very dangerous because too often you accidentally end up with your rows in a different order from what was intended in one or both of the files and hence end up matching information into the incorrect rows!

Here is an example unmatched merge, followed by the three example matched merges listed above.

X2a. Unmatched Merge

As mentioned above, unmatched merges are very dangerous and should be avoided unless absolutely necessary. Nonetheless, this is a simple starting point to understand the merge command:

merge using <i>filename</i>	This merges a saved Stata dataset with the master dataset, by adding extra variables at the right of the dataset. If the Stata file named <i>filename</i> has any variables with the same names as variables in the master dataset, these variables get ignored (although there are options to get information from these variables instead of ignoring them). A variable named <code>_merge</code> gets created that equals 1 if the observation was in the original dataset only, 2 if it was in the using dataset only, and 3 if it was in both datasets. If the same number of observations existed in both datasets before the unmatched merge, then <code>_merge</code> will always equal 3, but otherwise some rows will either be from only the original dataset (<code>_merge=1</code>) or only the using dataset (<code>_merge=3</code>). You could avoid extra observations being added from the using dataset (i.e., observations that would have <code>_merge=2</code>) by using the <code>nokeep</code> option.
tabulate <code>_merge</code>	Always check the values of <code>_merge</code> after merging two datasets, to avoid errors.

X2b. Matched One-to-One Merge

In a matched one-to-one merge, such as example (i) above, each one line in the master dataset corresponds to one line in the using dataset. The only exceptions are that some lines in either dataset may have zero matching lines in the other dataset. As emphasized above, you need to have the same identifier variables in each dataset in order to know which observation in the master dataset should be matched with which observation in the using dataset. Consider a match using data in which a variable named `personid` is a different value for each person, and each `personid` has one or more observations corresponding to different years or

sort <code>personid</code>	Matched merges require the data to be sorted first, in order by the matching variable. Even if the data are already sorted, Stata has to <i>know</i> that they are sorted, so you still have to use the <code>sort</code> command.
----------------------------	--

merge personid using *filename*, unique This merges a saved Stata dataset with the master dataset, by adding extra variables at the right of the master dataset. The variables listed immediately after the merge command – in this case “personid” – must match in order for an observation in one dataset to be declared the same as an observation in the other dataset. The using dataset must have been sorted by personid before saving it, or if it were not then you could add the sort option to this command; this would sort both the master dataset and the using dataset. The “unique” option tells Stata that in each dataset you have only one observation for each unique value of personid. If non-unique values are found, i.e., if two or more of the same person appear in either of the datasets, Stata will stop with an error message so that you can fix your apparent mistake (either something is wrong with your data or you do not really want a one-to-one merge). As with an unmatched merge, if the Stata file named *filename* has any variables with the same names as variables in the master dataset, these variables get ignored (although there are options to get information from these variables instead of ignoring them). A variable named `_merge` gets created that equals 1 if the observation was in the original dataset only, 2 if it was in the using dataset only, and 3 if it was in both datasets. You could avoid extra observations being added from the using dataset (i.e., observations that would have `_merge=2`) by using the `nokeep` option.

tabulate `_merge` Always check the values of `_merge` after merging two datasets, to avoid errors.

If you are merging two panel datasets, each unique observation in the dataset will be identified by unique combinations of two (or more) variables, such as a personid-year pair. This works the same as the above merge, except that “personid year” is used instead of just “personid”:

```
sort personid year
merge personid year using filename, unique
tabulate _merge
```

X2c. Matched Many-to-One Merge

In a matched many-to-one merge, such as example (ii) above, multiple lines in the master dataset can correspond to one line in the using dataset. As emphasized above, you need to have the same identifier variables in each dataset in order to match observations in the two datasets. Consider a match using cross-sectional data in which a variable named personid is a unique value for each person, and each person can be observed multiple times visiting grocery stores, with a variable groceryid used to identify the groceries (a person could have many observations in the same grocery store). (Alternatively, you might have panel data with people observed in different years, in which case year would take the place of groceryid.)

Before reading this, you should understand the commands in section X2b. Here there are two differences. First, the match variable is just “groceryid”. Second, because this is not a one-to-one merge, the “unique” option to the merge command is not appropriate: in the master data, each unique value of groceryid may occur many times. Instead, the `uniquing` option tells Stata that each different groceryid appears only once (at most) in the using dataset:

```
sort groceryid
```

merge groceryid using *filename*, unquising
tabulate _merge

X2d. Matched One-to-Many Merge

In a matched one-to-many merge, such as example (iii) above, one line in the master dataset can correspond to multiple lines in the using dataset. The master dataset's match variable values are unique, in the sense that at most one observation has any given value (or combination of values) of the match variable(s). Consider a match starting with master data having 50 observations corresponding to the 50 US states, with a variable named state being a unique value for each state. In the using data, each state is listed many times with each observation describing a government agency in one of the states.

Before reading this, you should understand the commands in section X2b. Here there are two differences. First, the match variable is just "state". Second, because this is not a one-to-one merge, the "unique" option to the merge command is not appropriate: in the using data, each unique value of state may occur many times. Instead, the `uniquemaster` option tells Stata that each different state appears only once (at most) in the master dataset:

```
sort state
merge state using filename, unquimaster
tabulate _merge
```

X2e. Matched Many-to-Many Merge

Many-to-many merges are also possible. For more information, get help on Stata's "joinby" command.

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 name the relevant 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)
```

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.