

MODULE ONE, PART THREE: READING DATA INTO STATA, CREATING AND RECODING VARIABLES, AND ESTIMATING AND TESTING MODELS IN STATA

This Part Three of Module One provides a cookbook-type demonstration of the steps required to read or import data into STATA. The reading of both small and large text and Excel files are shown through real data examples. The procedures to recode and create variables within STATA are demonstrated. Commands for least-squares regression estimation and maximum likelihood estimation of probit and logit models are provided. Consideration is given to analysis of variance and the testing of linear restrictions and structural differences, as outlined in Part One. (Parts Two and Four provide the LIMDEP and SAS commands for the same operations undertaken here in Part Three with STATA. For a review of STATA, version 7, see Kolenikov (2001).)

IMPORTING EXCEL FILES INTO STATA

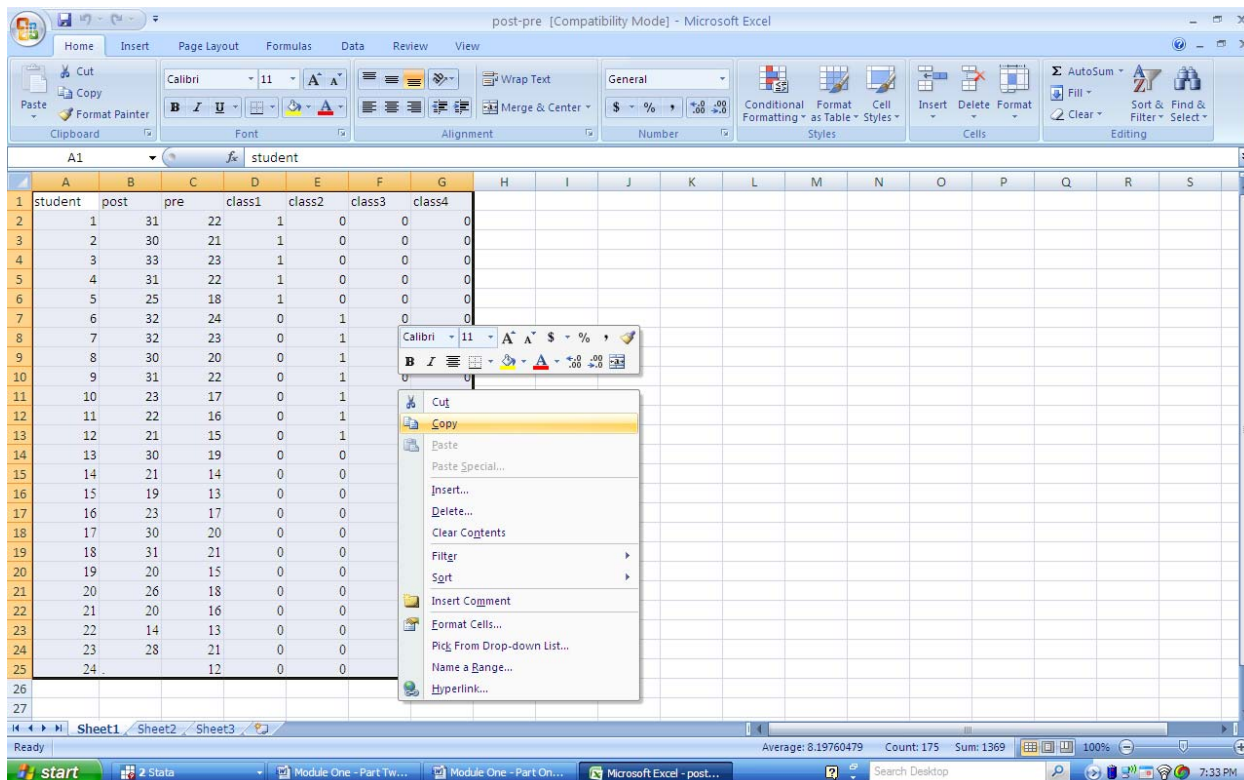
STATA can read data from many different formats. As an example of how to read data created in an Excel spreadsheet, consider the data from the Excel file “post-pre.xls,” which consists of test scores for 24 students in four classes. The column titled “Student” identifies the 24 students by number, “post” provides each student’s post-course test score, “pre” is each student’s pre-course test score, and “class” identifies to which one of the four classes the students was assigned, e.g., class4 = 1 if student was in the fourth class and class4 = 0 if not. The “.” in the post column for student 24 indicates that the student is missing a post-course test score.

To start, the file “post-pre.xls” must be downloaded and copied to your computer’s hard drive. Unfortunately, STATA does not work with “.xls” data by default (i.e., there is no default “import” function or command to get “.xls” data into STATA’s data editor); however, we can still transfer data from an Excel spreadsheet into STATA by copy and paste.* First, open the “post-pre.xls” file in Excel. The raw data are given below:

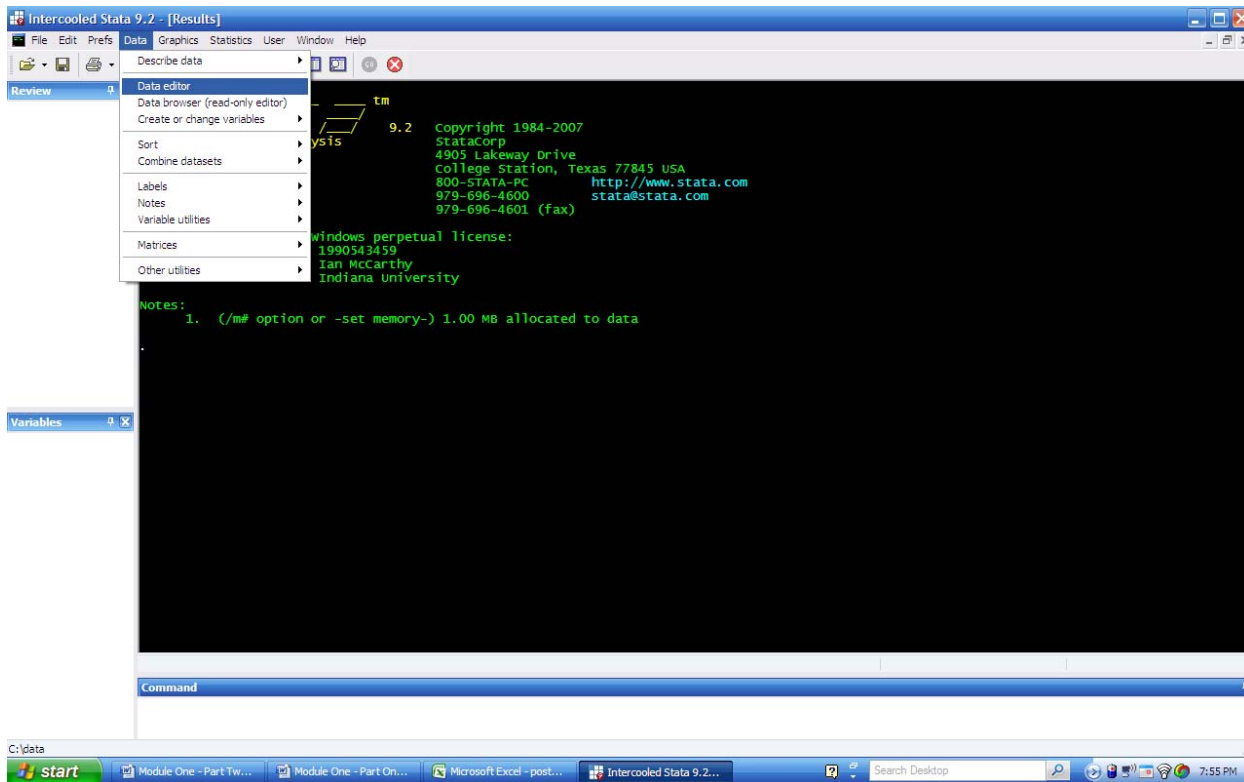
* See Appendix A for a description of Stat/Transfer, a program to convert data from one format to another.

student	post	pre	class1	class2	class3	class4
1	31	22	1	0	0	0
2	30	21	1	0	0	0
3	33	23	1	0	0	0
4	31	22	1	0	0	0
5	25	18	1	0	0	0
6	32	24	0	1	0	0
7	32	23	0	1	0	0
8	30	20	0	1	0	0
9	31	22	0	1	0	0
10	23	17	0	1	0	0
11	22	16	0	1	0	0
12	21	15	0	1	0	0
13	30	19	0	0	1	0
14	21	14	0	0	1	0
15	19	13	0	0	1	0
16	23	17	0	0	1	0
17	30	20	0	0	1	0
18	31	21	0	0	1	0
19	20	15	0	0	0	1
20	26	18	0	0	0	1
21	20	16	0	0	0	1
22	14	13	0	0	0	1
23	28	21	0	0	0	1
24	.	12	0	0	0	1

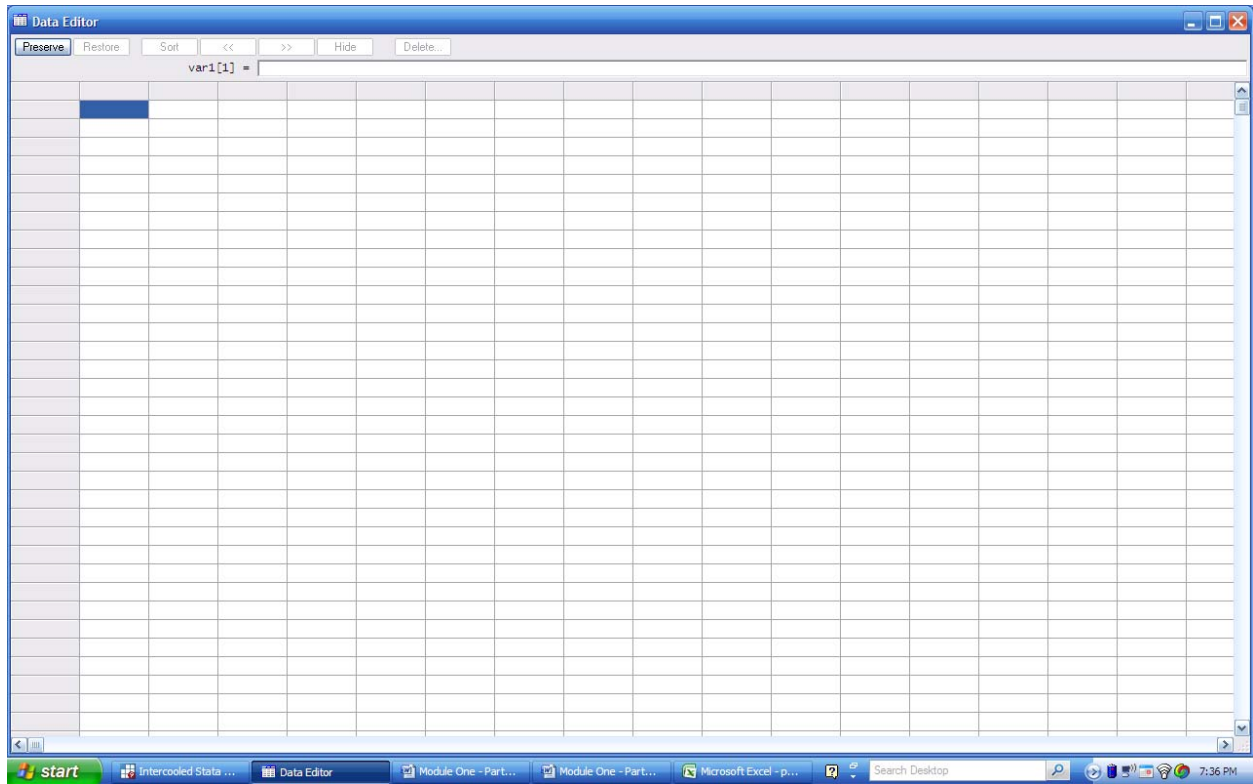
In Excel, highlight the appropriate cells, right-click on the highlighted area and click “copy”.
Your screen should look something like:



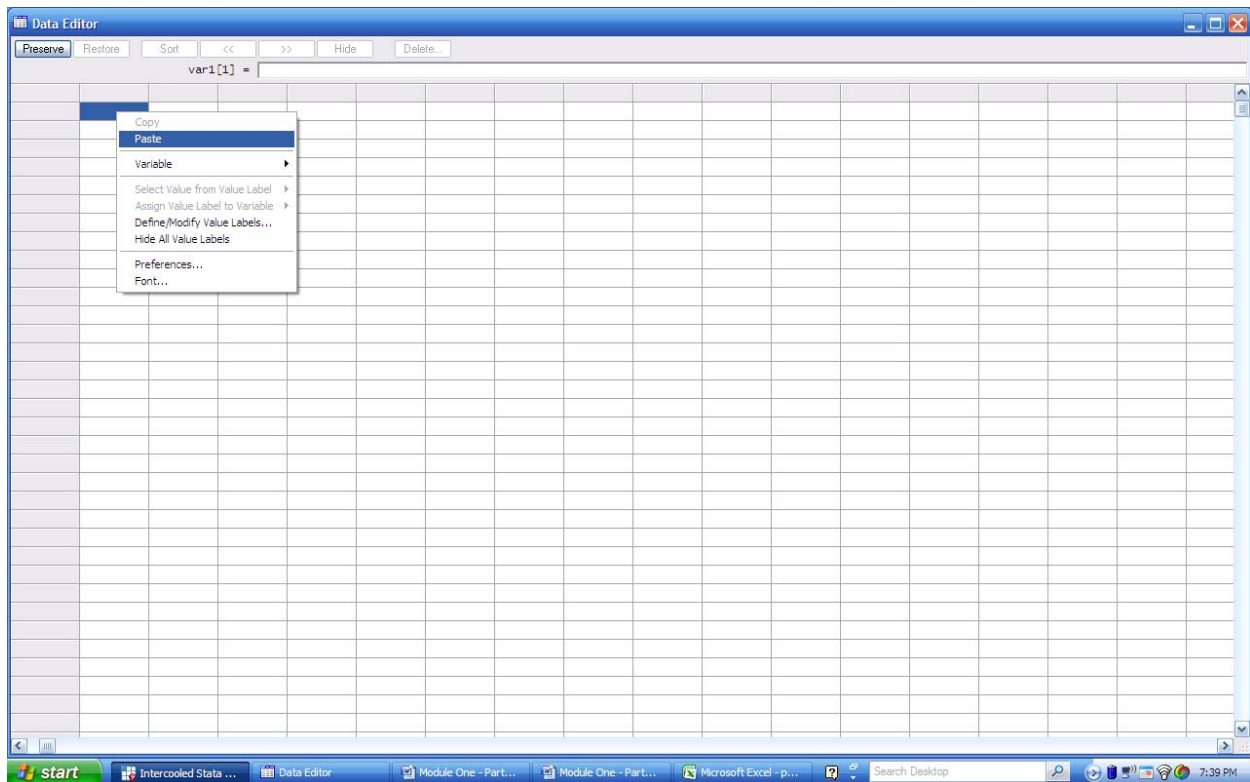
Then open STATA. Go to “Data”, and click on “Data Editor”:



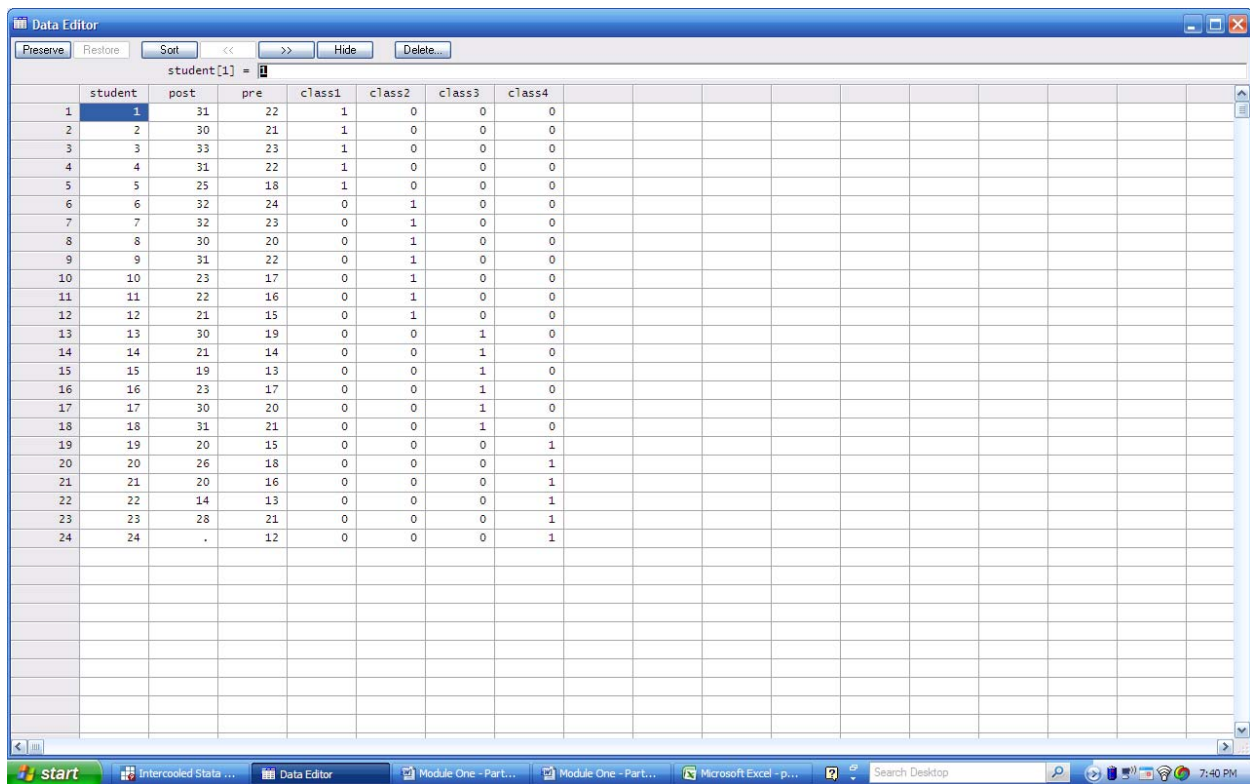
Clicking on “Data Editor” will yield the following screen:



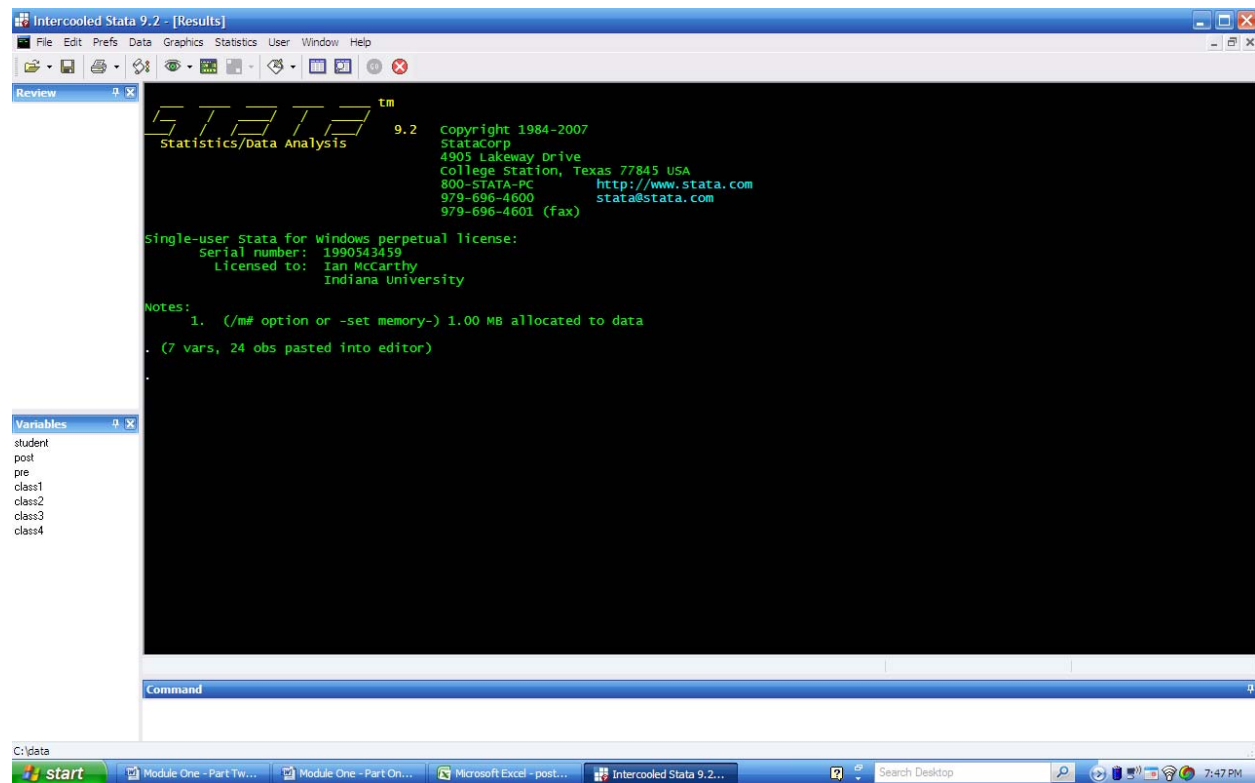
From here, right-click on the highlighted cell and click “paste”:



Your data is now in the STATA data editor, which yields the following screen:



Note that STATA, unlike LIMDEP, records missing observations with a period, rather than a blank space. Closing the data editor, we see that our variables are now added to the variable list in STATA, and we are correctly told that our data consist of 7 variables and 24 observations.



Any time you wish to see your current data, you can go back to the data editor. We can also view the data by typing in “browse” in the command window. As the terms suggest, “browse” only allows you to see the data, while you can manually alter data in the “data editor”.

READING SPACE, TAB, OR COMMA DELINEATED FILES INTO STATA

Next we consider externally created text files that are typically accompanied by the “.txt” or “.prn” extensions. As an example, we use the previous dataset with 24 observations on the 7 variables (“student,” “post,” “pre,” “class1,” “class2,” “class3,” and “class4”) and saved it as a space delineated text file “post-pre.txt.” To read the data into STATA, we need to utilize the “insheet” command. In the command window, type

```
insheet using "F:\NCEE (Becker)\post-pre.txt", delimiter(" ")
```

The “insheet” tells STATA to read in text data and “using” directs STATA to a particular file name. In this case, the file is saved in the location “F:\NCEE (Becker)\post-pre.txt”, but this will vary by user. Finally, the “delimiter(“ ”)” option tells STATA that the data points in this file are separated by a space. If your data were tab delimited, you could type

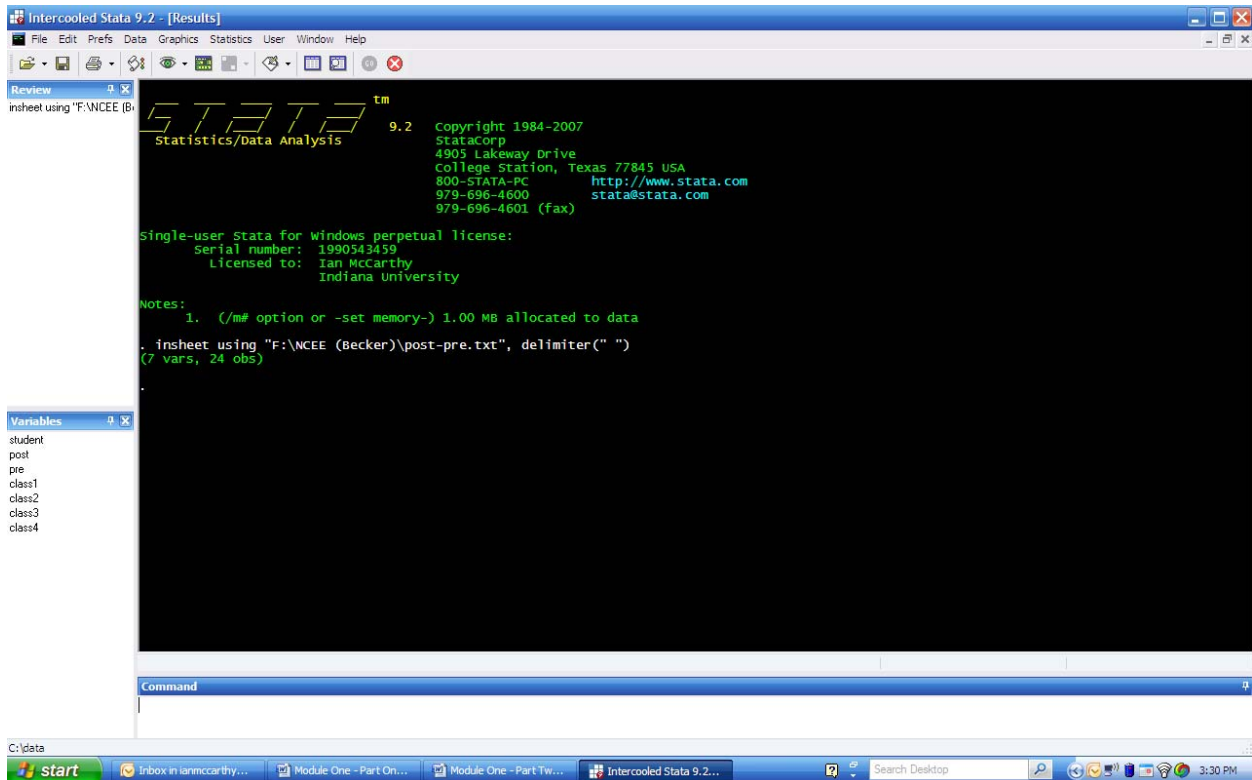
insheet using “F:\NCEE (Becker)\post-pre.txt”, tab

and if you were using a “.csv” file, you could type

insheet using “F:\NCEE (Becker)\post-pre.csv”, comma

In general, the “delimiter()” option is used when your data have a less standard delimiter (e.g., a colon, semicolon, etc.).

Once you’ve typed the appropriate command into the command window, press enter to run that line of text. This should yield the following screen:



Just as before, STATA tells us that it has read a data set consisting of 7 variables and 24 observations, and we can access our variable list in the lower-left window pane. We can also see previously written lines from the “review” window in the upper-left window pane. Again, we can view our data by typing “browse” in the command window and pressing enter.

READING LARGE DATA FILES INTO STATA

The default memory allocation is different depending on the version of STATA you are using. When STATA first opens, it will indicate how much memory is allocated by default. From the previous screenshot, for instance, STATA indicates that 1.00mb is set aside for STATA’s use.

This is shown in the note directly above the entered command, which appears every time we start STATA. The 1.00mb memory is the standard for Intercooled STATA, which is the version used for this module. For a slightly more detailed look at the current memory allocation, you can type into the command window, “memory” and press enter. This provides the following:

```

Intercooled Stata 9.2 - [Results]
File Edit Prefs Data Graphics Statistics User Window Help
Review memory
Variables

STATA 9.2 Copyright 1984-2007
Statistics/Data Analysis StataCorp
4905 Lakeway Drive
College Station, Texas 77845 USA
800-STATA-PC http://www.stata.com
979-696-4600 stata@stata.com
979-696-4601 (Fax)

Single-user Stata for windows perpetual license:
Serial number: 1990543459
Licensed to: Ian McCarthy
Indiana University

Notes:
1. (/m# option or -set memory-) 1.00 MB allocated to data

. memory

bytes

Details of set memory usage
overhead (pointers) 0 0.00%
data 0 0.00%
-----
data + overhead 0 0.00%
free 1,048,568 100.00%
-----
Total allocated 1,048,568 100.00%

Other memory usage
system overhead 745,154
set matsize usage 337,600
programs, saved results, etc. 105
-----
Total 1,082,859

Grand total 2,131,427

Command

```

A more useful (and detailed) description of STATA’s memory usage (among other things) can be obtained by typing “creturn list” into the command window. This provides:

The screenshot shows the STATA interface with the following output:

```

c(dirsep) = "/"

System limits

c(max_N_theory) = 2147483647
c(max_k_theory) = 2048
c(max_width_theory) = 24576

c(max_N_current) = 104854 (set memory)
c(max_k_current) = 2048 (set memory)
c(max_width_current) = 24576

c(max_matsize) = 800
c(min_matsize) = 10

c(max_macrolen) = 67784
c(max_cmlen) = 67800
c(max_cmdlen) = 67800
c(max_colname) = 67800
c(max_rname) = 32

Numerical and string limits

c(mindouble) = -8.9884656743e+307
c(maxdouble) = 8.9884656743e+307
c(epsdouble) = 2.22044604925e-16

c(minfloat) = -1.70141173319e+38
c(maxfloat) = 1.70141173319e+38
c(epsfloat) = 1.19209289551e-07

c(minlong) = -2147483647
c(maxlong) = 2147483620

c(minint) = -32767
c(maxint) = 32740

--more--

```

You may have to click on the “-more-“ link (bottom left of the STATA output window) to see this output. You can also press spacebar in the command window (or any key) to advance screens whenever you see “-more-“ at the bottom. Two things to notice from this screen are: (1) `c(max_N_theory)` tells us the maximum possible number of records our version of STATA will allow, while `c(max_N_current)` tells us the maximum possible number of records we have currently allocated to STATA based on our memory allocation, and (2) `c(max_k_theory)` tells us the maximum possible number of variables, while `c(max_k_current)` tells us the maximum number of variables based on our current memory allocation.

To work with large datasets (in this case, anything larger than 1mb), we can type “set memory 10m” into the command window and press enter. This increases the memory allocation to 10 mb, and you can increase by more or less to your preference. You can also increase STATA’s memory allocation permanently by typing, “set memory 10m, permanently” into the command line. To check that our memory has actually increased, again type “memory” into the command window and press enter. We get the following screen:

Intercooled Stata 9.2 - [Results]

```

File Edit Prefs Data Graphics Statistics User Window Help
[Icons]

Review
set mem 10m
memory

Variables

9.2 Copyright 1984-2007
StataCorp
4905 Lakeway Drive
College Station, Texas 77845 USA
800-STATA-PC http://www.stata.com
979-696-4600 stata@stata.com
979-696-4601 (Fax)

Single-user Stata for windows perpetual license:
Serial number: 1990543459
Licensed to: Ian McCarthy
Indiana University

NOTES:
1. (/m# option or -set memory-) 1.00 MB allocated to data

. set mem 10m
(10240k)

. memory

bytes

Details of set memory usage
overhead (pointers) 0 0.00%
data 0 0.00%
-----
data + overhead 0 0.00%
free 10,485,752 100.00%
-----
Total allocated 10,485,752 100.00%

Other memory usage
system overhead 745,154
set matsize usage 337,600
programs, saved results, etc. 105
-----
Total 1,082,859

Grand total 11,568,611

Command

```

C:\data

start | Inboxes in ianmccart... | Module One - Part... | Module One - Part... | Stata help for cret... | Intercooled Stata ... | Search Desktop | 3:54 PM

The maximum amount of memory you can allocate to STATA varies based on your computer's performance. If we try to allocate more memory than our RAM can allow, we get an error:

Intercooled Stata 9.2 - [Results]

```

File Edit Prefs Data Graphics Statistics User Window Help
[Icons]

Review
set mem 10m
memory
set mem 1000m

Variables

800-STATA-PC http://www.stata.com
979-696-4600 stata@stata.com
979-696-4601 (Fax)

Single-user Stata for windows perpetual license:
Serial number: 1990543459
Licensed to: Ian McCarthy
Indiana University

NOTES:
1. (/m# option or -set memory-) 1.00 MB allocated to data

. set mem 10m
(10240k)

. memory

bytes

Details of set memory usage
overhead (pointers) 0 0.00%
data 0 0.00%
-----
data + overhead 0 0.00%
free 10,485,752 100.00%
-----
Total allocated 10,485,752 100.00%

Other memory usage
system overhead 745,154
set matsize usage 337,600
programs, saved results, etc. 105
-----
Total 1,082,859

Grand total 11,568,611

. set mem 1000m
op. sys. refuses to provide memory
r(909);

Command

```

C:\data

start | Inboxes in ianmccart... | Module One - Part... | Module One - Part... | Stata help for cret... | Intercooled Stata ... | Search Desktop | 3:57 PM

Note that the total amount of memory allowed depends on the computer's performance; however, the total number of variables allowed may be restricted by your version of STATA. (If you're using Small STATA, then the memory allocation is also limited.) For Intercooled STATA, for instance, we cannot have more than 2048 variables in our data set.

For the Becker and Powers data set, the 1mb allocation is sufficient, so we need only follow the process to import a ".csv" file described above. Note, however, that this data set does not contain variable names in the top row. You can assign names yourself with a slight addition to the insheet command:

```
insheet var1 var2 var3 ... using "filename.csv", comma
```

Where, var1 var2 var3 ..., are the variable names for each of the 64 variables in the data set. Of course, manually adding all 64 variable names can be irritating. For more details on how to import data sets with data dictionaries (i.e., variable names and definitions in external files), try typing "help infile" into the command window. If you do not assign variable names, then STATA will provide default variable names of "v1, v2, v3, etc."

LEAST-SQUARES ESTIMATION AND LINEAR RESTRICTIONS IN STATA

As in the previous section using LIMDEP, we now demonstrate various regression tools in STATA using the "post-pre" data set. Recall the model being estimated is

$$post = \beta_1 + \beta_2 pre + f(classes) + \varepsilon.$$

STATA automatically drops any missing observations from our analysis, so we need not restrict the data in any of our commands. In general, the syntax for a basic OLS regression in STATA is

```
regress y-variable x-variables,
```

where *y-variable* is just the independent variable name and *x-variables* are the dependent variable names. Now is a good time to mention STATA's very useful help menu. Typing "help regress" into the command window and pressing enter will open a thorough description of the regress command and all of its options, and similarly with any command in STATA.

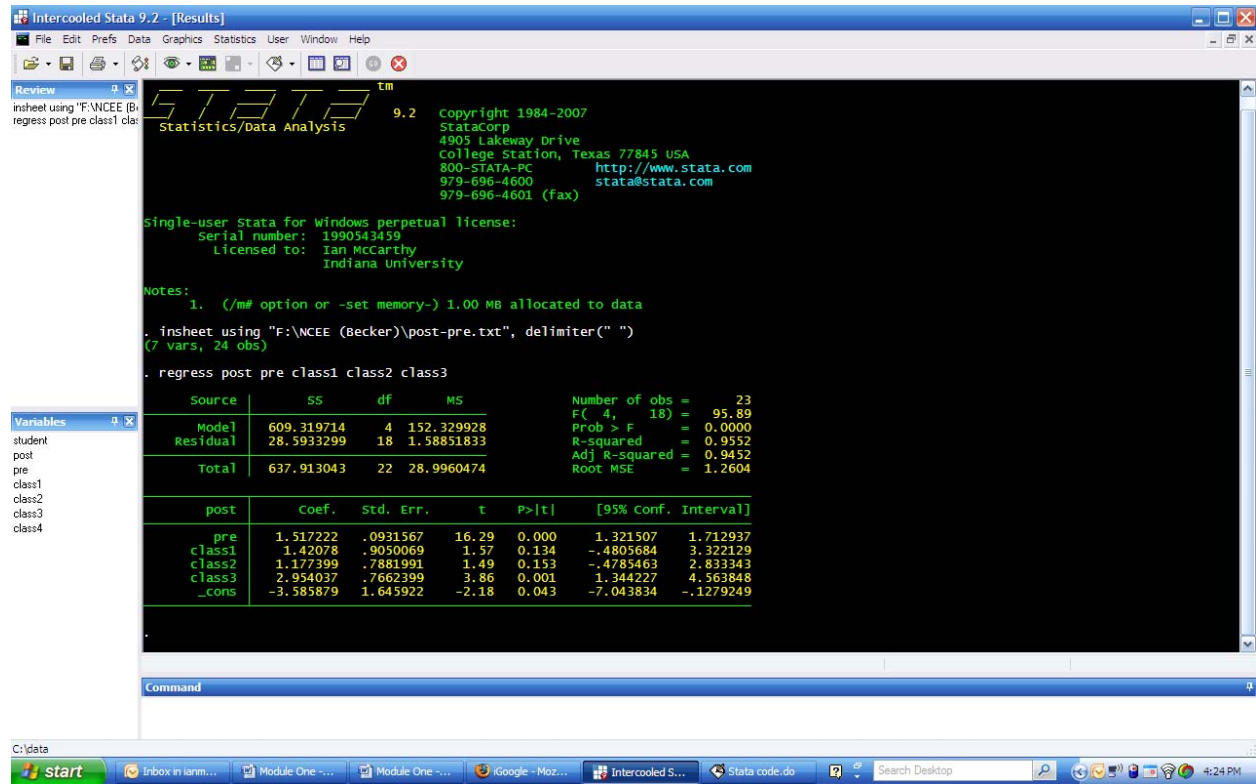
Once you have your data read into STATA, let's estimate the model

$$post = \beta_1 + \beta_2 pre + \beta_3 class1 + \beta_4 class2 + \beta_5 class3 + \varepsilon$$

by typing:

```
regress post pre class1 class2 class3
```

into the command window and pressing enter. We get the following output:



To see the predicted post-test scores (with confidence intervals) from our regression, we type:

```

predict posthat
predict se_f, stdf
generate upper_f = posthat + invttail(e(df_r),0.025)*se_f
generate lower_f = posthat - invttail(e(df_r),0.025)*se_f

```

You can either copy and paste these commands directly into the command window and press enter, or you can enter each one directly into the command window and press enter one at a time. Notice the use of the “predict” and “generate” keywords in the previous set of commands. After running a regression, STATA has lots of data stored away, some of which is shown in the output and some that is not. By typing “predict posthat”, STATA applies the estimated regression equation to the 24 observations in the sample to get predicted y -values. These predicted y -values are the default prediction for the “predict” command, and if we want the standard error of these predictions, we need to use “predict” again but this time specify the option “stdf”. This stands for the standard deviation of the forecast. Both posthat and se_f are new variables that STATA has created for us. Now, to get the upper and lower bounds of a 95% confidence interval, we apply the usual formula taking the predicted value plus/minus the margin of error. Typing “generate upper_f=...” and “generate lower_f=...” creates two new variables named “upper_f” and

“lower_f”, respectively. To see our predictions, we can type “browse” into the command window and press enter. This yields:

student	post	pre	class1	class2	class3	class4	posthat	se_f	upper_f	lower_f
1	31	22	1	0	0	0	31.21378	1.38267	34.11866	28.3089
2	30	21	1	0	0	0	29.69656	1.380786	32.59748	26.79563
3	33	23	1	0	0	0	32.731	1.390805	35.65297	29.80902
4	31	22	1	0	0	0	31.21378	1.38267	34.11866	28.3089
5	25	18	1	0	0	0	25.14489	1.412475	28.11239	22.17739
6	32	24	0	1	0	0	34.00484	1.40913	36.96531	31.04436
7	32	23	0	1	0	0	32.48762	1.384725	35.39682	29.57842
8	30	20	0	1	0	0	27.93595	1.347978	30.76795	25.10396
9	31	22	0	1	0	0	30.9704	1.366248	33.84077	28.10002
10	23	17	0	1	0	0	23.38429	1.368514	26.25943	20.50914
11	22	16	0	1	0	0	21.86707	1.387855	24.78284	18.95129
12	21	15	0	1	0	0	20.34984	1.413084	23.31862	17.38107
13	30	19	0	0	1	0	28.19537	1.370174	31.074	25.31674
14	21	14	0	0	1	0	20.60926	1.396315	23.54281	17.67571
15	19	13	0	0	1	0	19.09204	1.41994	22.07522	16.10886
16	23	17	0	0	1	0	25.16093	1.361703	28.02176	22.30009
17	30	20	0	0	1	0	29.71259	1.383829	32.61991	26.80527
18	31	21	0	0	1	0	31.22981	1.403547	34.17855	28.28107
19	20	15	0	0	0	1	19.17245	1.388682	22.08996	16.25493
20	26	18	0	0	0	1	23.72411	1.386806	26.63768	20.81054
21	20	16	0	0	0	1	20.68967	1.381791	23.5927	17.78663
22	14	13	0	0	0	1	16.138	1.420807	19.12301	13.153
23	28	21	0	0	0	1	28.27578	1.440219	31.30156	25.24999
24	.	12	0	0	0	1	14.62078	1.445632	17.65794	11.58362

Just as with LIMDEP, our 95% confidence interval for the 24th student’s predicted post-test score is [11.5836, 17.6579]. For more information on the “predict” command, try typing “help predict” into the command window.

To test the linear restriction of all class coefficients being zero, we type:

```
test class1 class2 class3
```

into the command window and press enter. STATA automatically forms the correct test statistic, and we see

```
F(3, 18) = 5.16
Prob > F = 0.0095
```

The second line gives us the p-value, where we see that we can reject the null that all class coefficients are zero at any probability of Type I error greater than 0.0095.

TEST FOR A STRUCTURAL BREAK (CHOW TEST)

The above test of the linear restriction $\beta_3 = \beta_4 = \beta_5 = 0$ (no difference among classes), assumed that the pretest slope coefficient was constant, fixed and unaffected by the class to which a student belonged. A full structural test can be performed in two possible ways. One, we can run each restricted regression and the unrestricted regression, take note of the residual sums of squares from each regression, and explicitly calculate the F-statistic. This requires the fitting of four separate regressions to obtain the four residual sum of squares that are added to obtain the unrestricted sum of squares. The restricted sum of squares is obtained from a regression of posttest on pretest with no dummies for the classes; that is, the class to which a student belongs is irrelevant in the manner in which pretests determine the posttest score.

For this, we can type:

```
regress post pre if class1==1
```

into the command window and press enter. The resulting output is as follows:

```
. regress post pre if class1==1
```

Source	SS	df	MS			
Model	35.7432432	1	35.7432432	Number of obs =	5	
Residual	.256756757	3	.085585586	F(1, 3) =	417.63	
Total	36	4	9	Prob > F =	0.0003	
				R-squared =	0.9929	
				Adj R-squared =	0.9905	
				Root MSE =	.29255	

post	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
pre	1.554054	.0760448	20.44	0.000	1.312046	1.796063
_cons	-2.945946	1.61745	-1.82	0.166	-8.093392	2.201501

We see in the upper-left portion of this output that the residual sum of squares from this restricted regression is 0.2568. We can similarly run a restricted regression for only students in class 2 by specifying the option “if class2==1”, and so forth for classes 3 and 4.

The second way to test for a structural break is to create several interaction terms and test whether the dummy and interaction terms are jointly significantly different from zero. To perform the Chow test this way, we first generate interaction terms between all dummy variables and independent variables. To do this in STATA, type the following into the command window and press enter:

```
generate pre_c1=pre*class1
generate pre_c2=pre*class2
generate pre_c3=pre*class3
```

With our new variables created, we now run a regression with all dummy and interaction terms included, as well as the original independent variable. In STATA, we need to type:

```
regress post pre class1 class2 class3 pre_c1 pre_c2 pre_c3
```

into the command window and press enter. The output for this regression is not meaningful, as it is only the test that we're interested in. To run the test, we can then type:

```
test class1 class2 class3 pre_c1 pre_c2 pre_c3
```

into the command window and press enter. The resulting output is:

```
. test class1 class2 class3 pre_c1 pre_c2 pre_c3
```

```
( 1) class1 = 0  
( 2) class2 = 0  
( 3) class3 = 0  
( 4) pre_c1 = 0  
( 5) pre_c2 = 0  
( 6) pre_c3 = 0
```

```
      F( 6, 15) = 2.93  
      Prob > F = 0.0427
```

Just as we saw in LIMDEP, our F -statistic is 2.93, with a p-value of 0.0427. We again reject the null (at a probability of Type I error=0.05) and conclude that class is important either through the slope or intercept coefficients. This type of test will always yield results identical to the restricted regression approach.

HETEROSCEDASTICITY

You can control for heteroscedasticity across observations or within specific groups (in this class, within a given class, but not across classes) by specifying the “robust” or “cluster” option, respectively, at the end of your regression command.

To account for a common error term within groups, but not across groups, we first create a class variable that identifies each student into one of the 4 classes. This is used to specify which group (or cluster) a student is in. To generate this variable, type:

```
generate class=class1 + 2*class2 + 3*class3 + 4*class4
```

into the command window and press enter. Then to allow for clustered error terms, our regression command is:

```
regress post pre class1 class2 class3, cluster(class)
```

This gives us the following output:

```
. regress post pre class1 class2 class3, cluster(class)
```

```
Linear regression                               Number of obs =      23
```

Number of clusters (class) = 4

F(0, 3) = .
Prob > F = .
R-squared = 0.9552
Root MSE = 1.2604

post	Coef.	Robust Std. Err.	t	P> t	[95% Conf. Interval]	
pre	1.517222	.1057293	14.35	0.001	1.180744	1.8537
class1	1.42078	.4863549	2.92	0.061	-.1270178	2.968579
class2	1.177399	.3141671	3.75	0.033	.1775785	2.177219
class3	2.954037	.0775348	38.10	0.000	2.707287	3.200788
_cons	-3.585879	1.755107	-2.04	0.134	-9.171412	1.999654

Similarly, to account for general heteroscedasticity across individual observations, our regression command is:

```
regress post pre class1 class2 class3, robust
```

and we get the following output:

```
. regress post pre class1 class2 class3, robust
```

Linear regression

Number of obs = 23
F(4, 18) = 165.74
Prob > F = 0.0000
R-squared = 0.9552
Root MSE = 1.2604

post	Coef.	Robust Std. Err.	t	P> t	[95% Conf. Interval]	
pre	1.517222	.0824978	18.39	0.000	1.3439	1.690543
class1	1.42078	.7658701	1.86	0.080	-.188253	3.029814
class2	1.177399	.8167026	1.44	0.167	-.53843	2.893227
class3	2.954037	.9108904	3.24	0.005	1.040328	4.867747
_cons	-3.585879	1.706498	-2.10	0.050	-7.171098	-.0006609

ESTIMATING PROBIT MODELS IN STATA

We now want to estimate a probit model using the Becker and Powers data set. First, read in the “.csv” file:¹

```
. insheet a1 a2 x3 c al am an ca cb cc ch ci cj ck cl cm cn co cs ct cu ///
> cv cw db dd di dj dk dl dm dn dq dr ds dy dz ea eb ee ef ///
> ei ej ep eq er et ey ez ff fn fx fy fz ge gh gm gn gq gr hb ///
> hc hd he hf using "F:\NCEE (Becker)\BECK8W02.csv", comma
(64 vars, 2849 obs)
```


Notice the “///” at the end of each line. Because STATA by default reads the end of the line as the end of a command, you have to tell it when the command actually goes on to the next line. The “///” tells STATA to continue reading this command through the next line.

As always, we should look at our data before we start doing any work. Typing “browse” into the command window and pressing enter, it looks as if several variables have been read as character strings rather than numeric values. We can see this by typing “describe” into the command window or simply by noting that string variables appear in red in the browsing window. This is a somewhat common problem when using STATA with Excel, usually because of variable names in the Excel files or because of spaces placed in front or after numeric values. If there are spaces in any cell that contains an otherwise numeric value, STATA will read the entire column as a character string. Since we know all variables should be numeric, we can fix this problem by typing:

```
destring, replace
```

into the command window and pressing enter. This automatically codes all variables as numeric variables.

Also note that the original Excel .csv file has several “extra” observations at the end of the data set. These are essentially extra rows that have been left blank but were somehow utilized in the original Excel file (for instance, just pressing enter at last cell will generate a new record with all missing variables). STATA correctly reads these 12 observations as missing values, but because we know these are not real observations, we can just drop these with the command “drop if a1==.”. This works because a1 is not missing for any of the other observations.

Now we recode the variable a2 as a categorical variable, where a2=1 for doctorate institutions (between 100 and 199), a2=2 for comprehensive master’s degree granting institutions (between 200 and 299), a2=3 for liberal arts colleges (between 300 and 399), and a2=4 for two-year colleges (between 400 and 499). To do this, type the following command into the command window:

```
recode a2 (100/199=1) (200/299=2) (300/399=3) (400/499=4)
```

Once we’ve recoded the variable, we can generate the 4 dummy variables as follows:ⁱⁱ

```
generate doc=(a2==1) if a2!=.  
generate comp=(a2==2) if a2!=.  
generate lib=(a2==3) if a2!=.  
generate twoyr=(a2==4) if a2!=.
```

The more lengthy way to generate these variables would be to first generate new variables equal to zero, and then replace each one if the relevant condition holds. But the above commands are a more concise way.

Next 1 - 0 bivariates are created to show whether the instructor had a PhD degree and where the student got a positive score on the postTUCE. We also create new variables, dmsq and hbsq, to allow for quadratic forms in teacher experiences and class size:

```
generate phd=(dj==3) if dj!=.
generate final=(cc>0) if cc!=.
generate dmsq=dm^2
generate hbsq=hb^2
```

In this data set, all missing values are coded -9. Thus, adding together some of the responses to the student evaluations provides information as to whether a student actually completed an evaluation. For example, if the sum of ge, gh, gm, and gq equals -36, we know that the student did not complete a student evaluation in a meaningful way. A dummy variable to reflect this fact is then created by:ⁱⁱⁱ

```
generate noeval=(ge + gh + gm + gq == -36)
```

Finally, from the TUCE developer it is known that student number 2216 was counted in term 2 but was in term 1 but no postTUCE was taken. This error is corrected with the following command:

```
recode hb (90=89)
```

We are now ready to estimate the probit model with final as our dependent variable. Because missing values are coded as -9 in this data set, we need to avoid these observations in our analysis. The quickest way to avoid this problem is just to recode all of the variables, setting every variable equal to "." if it equals "-9". Because there are 64 variables, we do not want to do this one at a time, so instead we type:

```
foreach x of varlist * {
  replace `x'=. if `x'=-9
}
```

You should type this command exactly as is for it to work correctly, including pressing enter after the first open bracket. Also note that the single quotes surrounding each x in the replace statement are two different characters. The first single quote is the key directly underneath the escape key (for most keyboards) while the closing single quote is the standard single quote keystroke by the enter key. For more help on this, type "help foreach" into the command window.

Finally, we drop all observations where an=. and where cs=0 and run the probit model by typing

```
drop if an==.
```

```
drop if cs==0
probit final an hb doc comp lib ci ck phd noeval
```

into the command window and pressing enter. We can then retrieve the marginal effects by typing “mfx” into the command window and pressing enter. This yields the following output:

```
. drop if cs==0
(1 observation deleted)

. drop if an==.
(249 observations deleted)

. probit final an hb doc comp lib ci ck phd noeval

Iteration 0: log likelihood = -1284.2161
Iteration 1: log likelihood = -840.66421
Iteration 2: log likelihood = -823.09278
Iteration 3: log likelihood = -822.74126
Iteration 4: log likelihood = -822.74107

Probit regression                               Number of obs   =       2587
                                                LR chi2(9)      =       922.95
                                                Prob > chi2     =       0.0000
Log likelihood = -822.74107                    Pseudo R2      =       0.3593
```

final	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]
an	.022039	.0094752	2.33	0.020	.003468 .04061
hb	-.0048826	.0019241	-2.54	0.011	-.0086537 -.0011114
doc	.9757148	.1463617	6.67	0.000	.6888511 1.262578
comp	.4064945	.1392651	2.92	0.004	.13354 .679449
lib	.5214436	.1766459	2.95	0.003	.175224 .8676632
ci	.1987315	.0916865	2.17	0.030	.0190293 .3784337
ck	.08779	.1342874	0.65	0.513	-.1754085 .3509885
phd	-.133505	.1030316	-1.30	0.195	-.3354433 .0684333
noeval	-1.930522	.0723911	-26.67	0.000	-2.072406 -1.788638
_cons	.9953498	.2432624	4.09	0.000	.5185642 1.472135

```
. mfx
Marginal effects after probit
y = Pr(final) (predict)
= .88118215
```

variable	dy/dx	Std. Err.	z	P> z	[95% C. I.]	X
an	.004378	.00188	2.33	0.020	.000699 .008057	10.5968
hb	-.0009699	.00038	-2.54	0.011	-.001719 -.00022	55.5589
doc*	.1595047	.02039	7.82	0.000	.119537 .199473	.317743
comp*	.0778334	.02588	3.01	0.003	.027107 .12856	.417859
lib*	.0820826	.02145	3.83	0.000	.040039 .124127	.135678
ci	.0394776	.01819	2.17	0.030	.003834 .075122	1.23116
ck*	.0182048	.02902	0.63	0.530	-.038667 .075077	.919985
phd*	-.0257543	.01933	-1.33	0.183	-.063632 .012123	.686123
noeval*	-.533985	.01959	-27.26	0.000	-.572373 -.495597	.290684

(*) dy/dx is for discrete change of dummy variable from 0 to 1

For the other probit model (using hc rather than hb), we get:

```
. probit final an hc doc comp lib ci ck phd noeval
```

```
Iteration 0: log likelihood = -1284.2161
Iteration 1: log likelihood = -843.39917
Iteration 2: log likelihood = -826.28953
Iteration 3: log likelihood = -825.94736
Iteration 4: log likelihood = -825.94717
```

```
Probit regression                               Number of obs   =       2587
                                                LR chi2(9)      =       916.54
                                                Prob > chi2     =       0.0000
Log likelihood = -825.94717                    Pseudo R2      =       0.3568
```

final	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]
an	.0225955	.0094553	2.39	0.017	.0040634 .0411276
hc	.0001586	.002104	0.08	0.940	-.0039651 .0042823
doc	.880404	.1486641	5.92	0.000	.5890278 1.17178
comp	.4596089	.1379817	3.33	0.001	.1891698 .730048
lib	.5585268	.1756814	3.18	0.001	.2141976 .902856
ci	.1797199	.090808	1.98	0.048	.0017394 .3577004
ck	.0141566	.1333267	0.11	0.915	-.2471589 .2754722
phd	-.2351326	.1010742	-2.33	0.020	-.4332344 -.0370308
noeval	-1.928216	.0723636	-26.65	0.000	-2.070046 -1.786386
_cons	.8712666	.2411741	3.61	0.000	.3985742 1.343959

```
. mfx
```

```
Marginal effects after probit
y = Pr(final) (predict)
= .88073351
```

variable	dy/dx	Std. Err.	z	P> z	[95% C.I.]	X
an	.0045005	.00188	2.40	0.017	.00082 .008181	10.5968
hc	.0000316	.00042	0.08	0.940	-.00079 .000853	49.9749
doc*	.1467544	.02132	6.88	0.000	.104969 .18854	.317743
comp*	.087859	.02554	3.44	0.001	.037809 .137909	.417859
lib*	.0867236	.02066	4.20	0.000	.046228 .12722	.135678
ci	.0357961	.01807	1.98	0.048	.000383 .071209	1.23116
ck*	.0028395	.02693	0.11	0.916	-.049938 .055617	.919985
phd*	-.0444863	.01819	-2.45	0.014	-.080145 -.008828	.686123
noeval*	-.5339711	.01957	-27.29	0.000	-.572326 -.495616	.290684

(*) dy/dx is for discrete change of dummy variable from 0 to 1

Results from each model are equivalent to those of LIMDEP, where we see that the estimated coefficient on hb is -0.005 with a p-value of 0.01, and the estimated coefficient on hc is 0.00007 with a p-value of 0.974. These results imply that initial class size is strongly significant while final class size is insignificant.

To assess model fit, we can form the predicted 0 or 1 values by first taking the predicted probabilities and then transforming these into 0 or 1 depending on whether the predicted probability is greater than .5. Then we can look at a tabulation to see how many correct 0s and 1s our probit model predicts. Because we have already run the models, we are not interested in the output, so to look only at these predictions, type the following into the command window:

```
quietly probit final an hb doc comp lib ci ck phd noeval
predict prob1
generate finalhat1=(prob1>.5)
```

this yields:

```
. quietly probit final an hb doc comp lib ci ck phd noeval
. predict prob1
(option p assumed; Pr(final))
. generate finalhat1=(prob1>.5)
. tab finalhat1 final
```

finalhat1	final		Total
	0	1	
0	342	197	539
1	168	1,880	2,048
Total	510	2,077	2,587

These results are exactly the same as with LIMDEP. For the second model, we get

```
. quietly probit final an hc doc comp lib ci ck phd noeval
. predict prob2
(option p assumed; Pr(final))
. generate finalhat2=(prob2>.5)
. tab finalhat2 final
```

finalhat2	final		Total
	0	1	
0	337	192	529
1	173	1,885	2,058
Total	510	2,077	2,587

Again, these results are identical to those of LIMDEP.

We can also use the built-in processes to do these calculations. To do so, type “estat class” after the model you’ve run. Part of the resulting output will be the tabulation of predicted versus actual values. Furthermore, to perform a Pearson goodness of fit test, type “estat gof” into the command window after you have run your model. This will provide a Chi-square value. All of these postestimation tools conclude that both models do a sufficient job of prediction.

REFERENCES

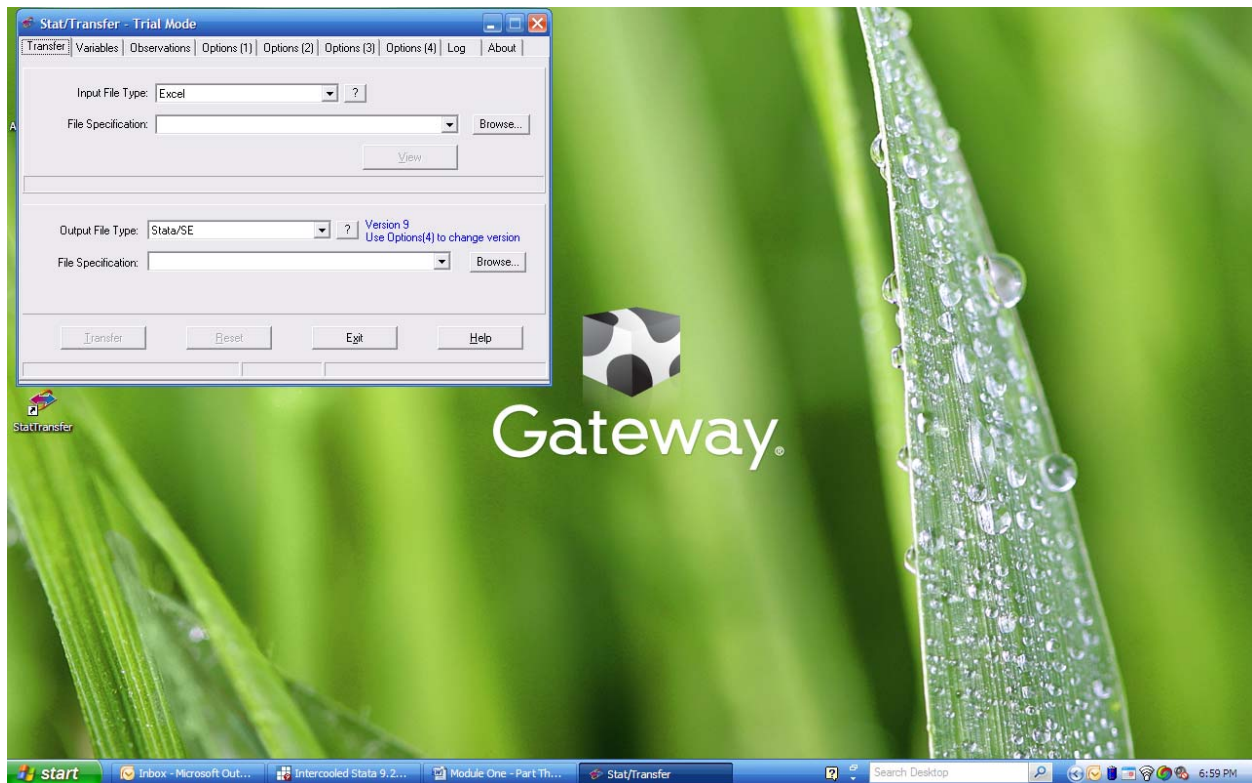
Kolenikov, Stanislav (2001). “Review of STATA 7” *The Journal of Applied Econometrics*, 16(5), October: 637-46.

Becker, William E. and John Powers (2001). “Student Performance, Attrition, and Class Size Given Missing Student Data,” *Economics of Education Review*, Vol. 20, August: 377-388.

APPENDIX A : Using Stat/Transfer

Stat/Transfer is a convenient program used to convert data from one format to another. Although this program is not free, there is a free trial version available at www.stattransfer.com. Note that the trial program will not convert the entire data set—it will drop one observation.

Nonetheless, Stat/Transfer is very user friendly. If you install and open the trial program, your screen should look something like:



We want to convert the “.xls” file into a STATA format (“.dta”). To do this, we need to first specify the original file type (e.g., Excel), then specify the location of the file. We then specify the format that we want (in this case, a STATA “.dta” file). Then click on Transfer, and Stat/Transfer automatically converts the data into the format you’ve asked.

To open this new “.dta” file in STATA, simply type

use “filename.dta”

into the command window and press enter.

ENDNOTES

ⁱ When using the “insheet” command, STATA automatically converts A1 to a1, A2 to a2 and so forth. STATA is, however, case sensitive. Therefore, whether users specify “insheet A1 A2 ...” or “insheet a1 a2 ...,” we must still call the variables in lower case. For instance, the following insheet command will work the exact same as that provided in the main text:

```
. insheet A1 A2 X3 C AL AM AN CA CB CC CH CI CJ CK CL CM CN CO CS CT CU ///
> CV CW DB DD DI DJ DK DL DM DN DQ DR DS DY DZ EA EB EE EF      ///
> EI EJ EP EQ ER ET EY EZ FF FN FX FY FZ GE GH GM GN GQ GR HB    ///
> HC HD HE HF using "F:\NCEE (Becker)\BECK8WO2.csv", comma
```

ⁱⁱ The conditions “if a2!=.” tell STATA to run the command only if a2 is not missing. Although this particular dataset does not contain any missing values, it is generally good practice to always use this type of condition when creating dummy variables the way we have done here. For example, if there were a missing observation, the command “gen doc=(a2==1)” would set doc=0 even if a2 is missing.

ⁱⁱⁱ An alternative procedure is to first set all variables to missing if they equal -9 and then generate the dummy variable using:

```
generate noeval=(ge==.&gh==.&gm==.&qg==.)
```