STATA FUNDAMENTALS
FOR MIDDLEBURY COLLEGE ECONOMICS STUDENTS

BY EMILY FORREST
AUGUST 2008

CONTENTS

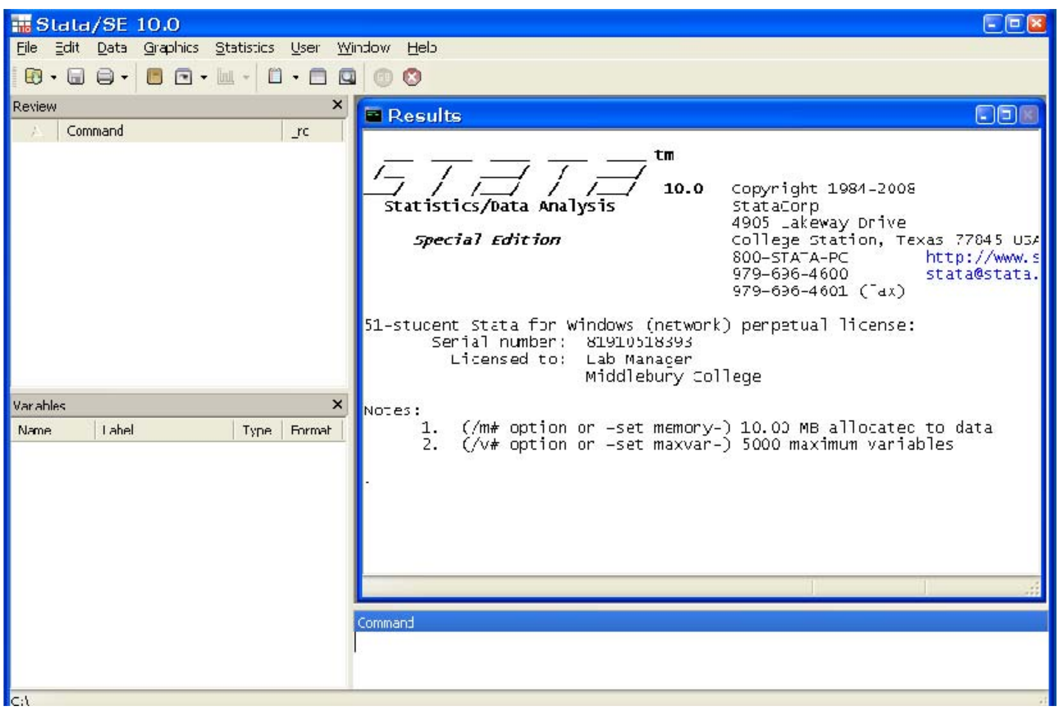
INTRODUCTION

Stata is a powerful statistical software package, used by students and researchers in many fields. Stata can manipulate data, calculate statistics, and run regressions. You will to need to use Stata to complete problem sets and write research papers for your economics classes. This manual is an introduction to Stata's basic features, covering the applications and functions that you are likely to use most often. It will explain the three types of files Stata uses: datasets (.dta), do-files (.do), and log files (.smcl).

The examples provided in this guide use the dataset **wages.dta**, which is available along with this manual and a sample do-file at the Economics Department website.

Stata is available on a number of computers on campus. To see a full list of computers that have Stata, go to the LIS website, select "Quick Links" and then click "Software on Library & Lab Computers." As of the writing of this manual, Stata was available in at least some labs in Sunderland, Munroe, the Library, and Bi-Hall.

When you first open Stata, you will see a screen that looks like this:



The large **Results** window shows output. Commands are typed into the **Command** window below it. The **Review** window displays recently executed commands – to repeat a command without typing it again, you can click on it in this window. The **Variables** window displays the names and labels of the variables in your dataset.

If you place the cursor over buttons in the toolbar, you will see that they are, from left to right, **Open...**, **Save**, **Print**, **Begin Log**, **View...**, **Graph**, **Do-File Editor**, **Data Editor**, **Data Browser**, **Go**, and **Break**.

STATA SYNTAX

You will need to input commands using Stata syntax. Stata is caps-sensitive; commands do not use capital letters and variable names should only be inputted with capital letters if that is how they appear in the variable window.

As we go through some Stata commands in this manual, you will notice that most follow a basic syntax:

> **command varlist if exp, options**
>
> The **command** tells Stata what action to take.
>
> The **varlist** tells Stata what variables to take this action on. This is optional in many cases.
>
> The **if exp** expression limits the command to a selected sample of the data set. This is always optional.
>
> The **, options** allows you to add options that are specific to that command.
>
> An example that uses all elements of this syntax appears in the "Summarizing Data" section.

Commands can generally be abbreviated to save typing. You don't need to type anything more than **su** in place of **summarize** or **ta** for **tabulate**, but you need to type out the whole word for **tabstat**.

There is a list of common commands to familiarize yourself with at the end of this guide.

DATASET FILES

OPENING A DATASET

Datasets in Stata format have the extension .dta. There are several ways to open a .dta file in Stata:

- Click on the .dta file. This will open Stata and load the data in a new window.

- Open Stata. From the File menu, choose "Open" and select the dataset.

- Open Stata. Type **use *filename***.

  Note: In order for this to work, you must be running Stata in the same directory (folder) as the file you need to open. To see what directory you are in, type **pwd**. To change the directory, type **cd**, followed by the name of the new directory. If your files are on Tigercat, type **cd U:**

  If you do not wish to change directories, you can type out the entire file path, that is, type **use "U:\wages.dta"** if the file is on Tigercat.

The third method seems the most complicated at first, but it is the best method to use when working with do-files since the command can be included in the do-file, allowing you to avoid having to open the file manually each time before running the do-file.

FROM EXCEL TO STATA

Often, data for a project will be in a different format, like an Excel spreadsheet. Usually, data in Excel requires some editing before it can be transferred to Stata. First, make sure that the first row of the Excel sheet is the variable names and that the observations begin in the next row. If there are no variable names, then the observations can start in the first row, but there cannot be any extra rows. Any missing values should be an empty cell, not a space or a dot. Make sure that the variable names do not contain any spaces. This is also a good time to make sure that the variable names make sense. Concise names that make sense to you are best. You can change the names in Stata if you wish to, but many people find it easier to do in Excel.

The easiest way to transfer a small Excel data set into Stata is to use copy and paste. Make sure that your excel spreadsheet is ready to copy. Open Stata and open the Data editor, which will look similar to a blank spreadsheet. Then, select all of the cells in the Excel file, copy them, and paste them into Stata's data editor.

An alternative is to save the Excel data as a tab-delimited text file. (**File>Save As…** In the "Save as type" dropdown menu, choose "Text (Tab delimited) (*.txt).) Then, in Stata, type **insheet using filename.txt.** (Again, you must be running Stata in the directory where you saved the .txt file. By now you may have noticed that it is best to change the directory to Tigercat when you open Stata.) This imports the dataset into Stata. You can type **browse** or click the **Data Browser** button in the toolbar (it has a magnifying glass) to look at the data. Look for obvious mistakes. Any observations in red are variables that Stata has interpreted as *string*, or nonnumeric variables. You won't be able to run regressions with these. If a variable shouldn't be a string variable, try to figure out why Stata interpreted it that way (e.g., one of the observations for that variable includes something like "na" for not available instead of a number), fix it in Excel, and input the data into Stata again.

WORKING WITH LARGE DATASETS

If a dataset is very large, you may receive an error message when you try to open it in Stata ("no room to add more variables"). In this case, you can increase the memory available to Stata by typing:

clear
set memory Xm

where X is the amount of memory you need. The default setting is 10m (MB). If your dataset is larger than 10MB, Stata will not be able to open it. To find out how much memory your dataset will need, right-click on the file and select **Properties...** Set the memory in Stata to be slightly larger than the size of the file. After you set the memory, try to open the dataset again.

Once you are able to load the data, the command **compress** may reduce the size of the dataset without changing the data. The variable may be formatted to use, for example, 8 digits, but if no observation uses more than two, Stata will eliminate the extra places to make space. This only works if there are extra places to eliminate.

SAVING DATA

Stata does not automatically save changes you make to the data. To save your data, use the **save** command. Typing **save *filename***, where you choose any *filename*, will save your data as filename.dta in the current directory. If you have already used the **cd** command to change the directory to Tigercat, then you can type **save *filename*** to save to Tigercat. Otherwise, you will have to type **save "U:\filename"** to save the file to Tigercat.

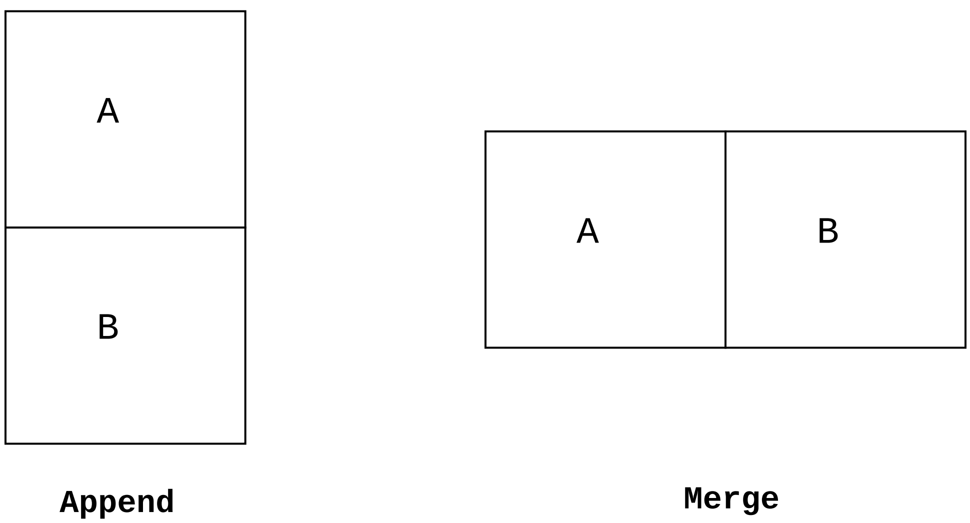
If you have already used the file name, then typing **save, replace** will replace the old file with the new version. Stata will not overwrite the old data unless you type out **replace**. Otherwise, if you try to use an existing file name, you will see an error message – this makes it difficult to accidentally overwrite data. If you would like to keep the old version of the data, choose a new file name.

It's a good idea to keep a clean copy of the original dataset so that you can start over if, for example, you accidentally drop a crucial variable.

COMBINING DATASETS

Instead of just one dataset, you might have data from several sources that you need to combine. If you want to stack data sets—that is, add more observations to the same variables—you want to combine data vertically. This situation might arise if, for instance, you had separate data sets for 2005 and 2006 and wanted to stack them into a single data set. If instead you want to add new information – new variables – about the same observations, you want to combine the data "horizontally."

Visual representation of combining data sets A and B



Combining data "vertically" is done using the **append** command. Both datasets need to be in Stata format (they need to be .dta files). Open the first dataset in Stata. Then, using the command **append using *seconddata*.dta,** where *seconddata* is the name of the second dataset, will combine the datasets.

To combine data "horizontally," use the command **merge**. Using **merge** is trickier than using **append**, because you have to ensure that the observations in each dataset correspond to each other. Usually you will want to merge data based on some kind of identification number (individual, household, etc). Both datasets need to be saved as .dta files. To merge the data, open the first dataset. Type **sort id** (where id is the name of your identification variable), then **save, replace**. Type **clear** and then open the second dataset. Type **sort id** again, and save the second dataset. While this second dataset is open, type **merge id using firstdata.dta** (where *firstdata* is the name of the first dataset) to combine the datasets. This performs what Stata calls a *match merge*. Note that this command will not work if the identification numbers are not unique (there must only be one observation per identification number).

EXPLORING THE DATA

Once you have your dataset open, you may wish to use Stata to familiarize yourself with it. You could begin by opening the data editor, which will show you a spreadsheet-like representation of your data. However, if your data sets has many variables and/or observations, this will be an unwieldy way to look at it. You may instead wish to use the **describe** command, which will return basic information about your data set and a list of variables. The command **lookfor x** is also helpful; it returns any variables that contains x in the name or description. For instance, if you were looking for a variable related to age, you might type **lookfor age** to see what variables have age in the name or description.

SUMMARIZING DATA

The first type of analysis that many people run when their data are ready is basic summary statistics. Remember that the Stata syntax of most commands is **command varlist if exp, options**. The following examples will take you through obtaining summary statistics using this syntax and examples of output from the sample dataset *wages.dta*.

summarize tells Stata to list the number of observations, mean, standard deviation, minimum, and maximum for each variable. This is the most basic syntax: it is only a command with no options.

Example

```
. summarize

    Variable |     Obs        Mean    Std. Dev.       Min        Max
-------------+-----------------------------------------------------
          id |       0
         age |     200       44.01    17.89023         16         85
         sex |     200       1.585    .4939585          1          2
   education |     200      39.875    3.030726         32         46
        race |     200        1.49    1.536785          1         18
-------------+-----------------------------------------------------
      ethnic |      22    1.954545    1.557693          1          5
  wkly_hours |     115    38.67826    8.047608         10         66
   wkly_earn |     123    678.1258    524.3861          0    2884.61
      region |     200        2.62    1.077686          1          4
```

The **varlist**, or variable list, tells Stata which variables to apply the command to. Type the names of the variables you want to use (or click on the names in the variables window to the left of the command window) without commas.

Example

summarize wkly_earn tells Stata to summarize weekly earnings only.

```
. summarize wkly_earn

    Variable |     Obs        Mean    Std. Dev.       Min        Max
-------------+-----------------------------------------------------
   wkly_earn |     123    678.1258    524.3861          0    2884.61
```

Using **if exp**, short for “if expression”, limits the command to certain values of a particular variable.

In these expressions:

== means “is equal to” Note that you must use *two* equal signs because this is a logical statement rather than a mathematical formula
!= means “does not equal”
~= also means "does not equal"
> means “greater than”
< means “less than”
>= means “greater than or equal to”
<= means “less than or equal to”
& means "and"
| means "or"

Note: Stata reads a missing value as infinity, so be careful when using these expressions. Stata might perform a command on missing observations that you intended to leave out.

<u>Examples</u>

summarize wkly_earn if sex==1 summarizes the weekly earnings of men as indicated by sex taking on a value of 1.

```
. summarize wkly_earn if sex==1

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
   wkly_earn |        54    746.4437    575.5715          0    2884.61
```

summarize wkly_earn if wkly_hours>=40 & sex==2 reports the number of observations that are women and work full time (40 or more hours per week).

```
. summarize wkly_earn if wkly_hours<40 & sex==2

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
   wkly_earn |        12    373.5792    292.7015        100       1080
```

summarize wkly_earn if sex==1|sex==2 reports the number of observations that are male or female. Of course, this is everyone in the sample.

```
. summarize wkly_earn if sex==1|sex==2

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
   wkly_earn |       123    678.1258    524.3861          0    2884.61
```

The modifier **bysort** can precede a command.

Example

bysort sex: summarize wkly_earn summarizes weekly earnings separately for each value of sex, that is, for men and for women.

```
. bysort sex: summarize wkly_earn

-------------------------------------------------------------------------------
-> sex = 1

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
   wkly_earn |        54    746.4437    575.5715          0    2884.61

-------------------------------------------------------------------------------
-> sex = 2

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
   wkly_earn |        69    624.6596    478.0536          0       2403
```

You can also include **options** with certain commands. Options come last in the command syntax and are preceded by a comma.

Example

Adding the option **detail** to the command **summarize** provides additional summary statistics.

```
. summarize wkly_earn, detail

                     Earnings per week
-------------------------------------------------------------
      Percentiles      Smallest
 1%            0              0
 5%          100              0
10%          220             51       Obs                 123
25%          346           82.4       Sum of Wgt.         123

50%          560                      Mean           678.1258
                        Largest       Std. Dev.      524.3861
75%          880        1923.07
90%      1403.84           2403       Variance       274980.8
95%         1680        2884.61       Skewness        1.89694
99%      2884.61        2884.61       Kurtosis       7.517251
```

The previous examples have all used the **summarize** command. Here are some other helpful commands for generating summary statistics.

tabulate makes a frequency table.

```
. tab sex

   Sex: 1 = |
  male, 2 = |
     female |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |         83       41.50       41.50
          2 |        117       58.50      100.00
------------+-----------------------------------
      Total |        200      100.00
```

codebook lists information about the values a variable takes. It shows the range, common percentiles, and how many unique values a variable has. It also shows how many observations are missing.

```
. codebook wage

--------------------------------------------------------------------------------------------------------------
wage                                                                                              (unlabeled)
--------------------------------------------------------------------------------------------------------------

                  type:  numeric (float)

                 range:  [0,72.11525]                 units:  1.000e-06
         unique values:  82                       missing .:  89/200

                  mean:   17.8266
              std. dev:   12.3417

           percentiles:        10%       25%       50%       75%       90%
                                 8        10     14.82   20.3846        30
```

tabstat allows you to customize a table of summary statistics.

Example

tabstat varlist, by(variable) stat(range sd p25 p50 p75) creates a table, sorted by *variable*, that includes the range, standard deviation, and 25th, 50th, and 75th percentiles of the specified variables. There are dozens of statistics options, see the online Stata help for tabstat for a complete list.

```
. tabstat wage, by(sex) stat(sd p25 p50 p75 range)

Summary for variables: wage
     by categories of: sex (Sex: 1 = male, 2 = female)

     sex |        sd       p25       p50       p75     range
---------+--------------------------------------------------
       1 |  13.80836     10.15  16.34324        24  72.11525
       2 |   10.9587       9.6        14        19    54.405
---------+--------------------------------------------------
   Total |  12.34169        10     14.82   20.3846  72.11525
------------------------------------------------------------
```

GENERATING VARIABLES

THE GENERATE COMMAND

In Stata, you can generate a new variable using the command **generate**. The general syntax is **generate newvarname =** ***expression***. You can include constants, variables, or both in the expression.

When using functions to generate a new variable,

* means multiply
/ means divide
+ means add
- means subtract
^ makes an exponent
ln(var) or log(var) takes the natural log of *var*

Example

Using the sample data, suppose you want to create an hourly wage variable by dividing weekly earnings by hours worked. Type **generate wage = wkly_earn/wkly_hours**.

```
. generate wage = wkly_earn/wkly_hours
(89 missing values generated)
```

Notice that Stata returns a message that 89 missing values have been generated. If Stata is unable to generate an observation, in this case because either **wkly_earn** or **wkly_hours** is missing, or wkly_hours is equal to zero, it records a missing value and notifies you.

If you wanted to include log wage in a regression, you could then generate a log wage variable by using the command **generate lwage = ln(wage)**. The missing values generated here result from a missing value for **wage** or a value of 0 for **wage** (since the natural log of 0 is undefined).

```
. generate lwage = ln(wage)
(90 missing values generated)
```

GENERATING DUMMY VARIABLES

You may often need to generate dummy variables to use in a regression. Dummy variables (also commonly called "indicator variables" or "binary variables") take on two values: 0 and 1.You can create a dummy variable using either a single command or two commands. Both options are shown in the examples below.

Example: Generating a dummy variable based on a continuous variable

Suppose you need to generate a dummy to indicate that an individual is of retirement age. a*ge* is a continuous variable, and you want your dummy variable to indicate when someone is over 65.

Option 1: Single command

generate retire = age>65

This creates a variable named **retire** that is equal to 1 if the expression is true (if the value of **age** is greater than 65) and equal to 0 if the expression is not true. By tabbing the new dummy variable, we see that 26 of the 200 observations (13%) are of retirement age.

```
. generate retire = age>65

. tab retire

     retire |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        174       87.00       87.00
          1 |         26       13.00      100.00
------------+-----------------------------------
      Total |        200      100.00
```

Option 2: Two commands

generate retire=1 if age>65

replace retire=0 if age<=65

This creates the same variable as in the first option, but you are being more explicit about what the conditions are. The first command generates retire, which is equal to 1 for people over 65 and is missing for everyone else. The second command replaces retire with 0 for people who are 65 or under, changing those missing values to zeros.

```
. generate retire=1 if age>65
(174 missing values generated)

. tab retire

     retire |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |         26      100.00      100.00
------------+-----------------------------------
      Total |         26      100.00

. replace retire=0 if age<=65
(174 real changes made)

. tab retire

     retire |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        174       87.00       87.00
          1 |         26       13.00      100.00
------------+-----------------------------------
      Total |        200      100.00
```

The above example works because the **age** variable in the sample dataset has no missing values. If any values were missing, Stata would code them as 1 in the new variable, because a missing value is read as infinity, which of course is greater than 65. I would be able to fix this by using the command **replace retire=. if age==.** after generating the variable.

The previous example involved creating a dummy variable based on a continuous variable, *age*. Data often includes categorical variables, which are variables that take on numeric values that are not inherently meaningful but instead correspond to different categories (region, race, and sex are examples variables that are often coded this way). A codebook usually accompanies the data to explain how the variables are coded, and in the case of a categorical variable, what each variable means. To include such a variable in a regression, you will need to create dummy variables for each category. The following examples cover several common situations that arise when creating dummy variables based on categorical variables.

Example: Generating a dummy variable based on a categorical variable that takes on two values

In the sample dataset, **sex** is set equal to 1 if an individual is male and 2 if she is female. So that the mean and coefficient for this variable can be easily interpreted, it should be converted to a dummy variable. The command **generate male= sex==1** creates a dummy variable equal to 1 for males and 0 for females.

```
. tab sex

   Sex: 1 =
  male, 2 =
     female |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |         83       41.50       41.50
          2 |        117       58.50      100.00
------------+-----------------------------------
      Total |        200      100.00

. generate male = sex==1

. tab male

       male |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        117       58.50       58.50
          1 |         83       41.50      100.00
------------+-----------------------------------
      Total |        200      100.00
```

Note: The same warning about missing values in the **retire** example above applies to the creation of dummy variables based on a categorical variable. If there are any missing observations in the original variable they will be coded as either 0s or 1s, depending on how you specify the command. If your variable does have missing observations, you will need to type **replace newvar = . if oldvar==.** after generating the variable to ensure that any observations that were missing remain missing in the new dummy variable. In this example, sex does not have any missing observations, so there is no problem to correct.

Example: Generating dummy variables based on a categorical variable that takes on more than two values

In the sample dataset, **region** is a categorical variable coded as 1 if a household is in the northeast, 2 if it is in the midwest, 3 if it is in the south, and 4 if it is in the west. If you wanted to control for region in a regression, you would need to create three dummy variables (because the fourth is omitted). This example does so using a different method for each variable.

```
. tab region

         1 = |
  northeast, |
         2 = |
  midwest, 3 |
  = south, 4 |
      = west |      Freq.     Percent        Cum.
-------------+-----------------------------------
           1 |         42       21.00       21.00
           2 |         42       21.00       42.00
           3 |         66       33.00       75.00
           4 |         50       25.00      100.00
-------------+-----------------------------------
       Total |        200      100.00
```

To create a dummy variable called **northeast**, equal to 1 if the individual lives in the northeast and 0 if he does not, I can type **generate northeast=1 if region==1**. This creates 1s for all the observations corresponding to individuals in the northeast, but observations corresponding to individuals outside of the northeast are created as missing values. I can replace the missing values with 0s by typing **replace northeast=0 if region!=1**.

```
. generate northeast=1 if region==1
(158 missing values generated)

. replace northeast=0 if region!=1
(158 real changes made)

. tab northeast

  northeast |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        158       79.00       79.00
          1 |         42       21.00      100.00
------------+-----------------------------------
      Total |        200      100.00
```

I could also simply type **generate midwest=0**, which creates a variable called **midwest** and sets every observation equal to 0, and then **replace midwest=1 if region==2** to change the 0s to 1s for those households in the midwest.

```
. generate midwest=0

. replace midwest=1 if region==2
(42 real changes made)

. tab midwest

    midwest |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        158       79.00       79.00
          1 |         42       21.00      100.00
------------+-----------------------------------
      Total |        200      100.00
```

An even easier way is to type **generate south= region==3**. This makes the 0s and 1s in one step.

```
. generate south= region==3

. tab south

      south |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        134       67.00       67.00
          1 |         66       33.00      100.00
------------+-----------------------------------
      Total |        200      100.00
```

<u>Example: Generating dummy variables that incorporate multiple values of a categorical variable</u>

A categorical variable can take on many values, but you might not need or want to create a separate dummy variable for each possible value of the categorical variable. The **education** variable in the sample dataset is an example. The values in **education** correspond to the highest grade an individual has completed. Instead, of making a separate dummy variable for each different grade, suppose you want to create only three dummy variables: one for individuals who did not finish high school, one for individuals who have a high school diploma but no college degree, and one for individuals who have some kind of college degree.

The education variable looks like this:

```
. tab education

    Highest |
      grade |
   attended |      Freq.     Percent        Cum.
------------+-----------------------------------
         32 |          3        1.50        1.50
         33 |          6        3.00        4.50
         34 |          4        2.00        6.50
         35 |          5        2.50        9.00
         36 |         11        5.50       14.50
         37 |          9        4.50       19.00
         38 |          3        1.50       20.50
         39 |         52       26.00       46.50
         40 |         35       17.50       64.00
         41 |          8        4.00       68.00
         42 |          8        4.00       72.00
         43 |         38       19.00       91.00
         44 |         11        5.50       96.50
         45 |          5        2.50       99.00
         46 |          2        1.00      100.00
------------+-----------------------------------
      Total |        200      100.00
```

The codebook that came with the data explains that the number 32 indicates less than a first-grade education, 33 indicates that an individual has completed at least the first grade but not beyond the fourth grade, 34 indicates that an individual has completed the fifth or sixth grade, and so on. Suppose that you wish to use this variable to create dummy variables for three mutually exclusive and collectively exhaustive educational categories: less than high school, high school diploma, and beyond high school. You note that you want to code these dummy variables using *education* as follows:

```
            . tab education

                Highest |
                  grade |
               attended |      Freq.     Percent        Cum.
            ------------+-----------------------------------
               {     32 |          3        1.50        1.50
               {     33 |          6        3.00        4.50
               {     34 |          4        2.00        6.50
no diploma    {      35 |          5        2.50        9.00
               {     36 |         11        5.50       14.50
               {     37 |          9        4.50       19.00
               {     38 |          3        1.50       20.50
high school   {      39 |         52       26.00       46.50
               {     40 |         35       17.50       64.00
               {     41 |          8        4.00       68.00
               {     42 |          8        4.00       72.00
college       {      43 |         38       19.00       91.00
               {     44 |         11        5.50       96.50
               {     45 |          5        2.50       99.00
               {     46 |          2        1.00      100.00
            ------------+-----------------------------------
                  Total |        200      100.00
```

To generate a dummy variable for no high school diploma, type **generate nodiploma=1 if education<=38** then **replace nodiploma=0 if education>38.**

```
. generate nodiploma=1 if education<=38
(159 missing values generated)

. replace nodiploma=0 if education>38
(159 real changes made)

. tab nodiploma

  nodiploma |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        159       79.50       79.50
          1 |         41       20.50      100.00
------------+-----------------------------------
      Total |        200      100.00
```

To generate a dummy variable for those who have a high school diploma but no college degree, we need to include categories 39 (high school diploma) and 40 (some college but no degree). Type **generate hsdiploma=0**, then **replace hsdiploma=1 if education>=39 & education<=40.**

```
. generate hsdiploma=0

. replace hsdiploma=1 if education>=39 & education<=40
(87 real changes made)

. tab hsdiploma

  hsdiploma |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        113       56.50       56.50
          1 |         87       43.50      100.00
------------+-----------------------------------
      Total |        200      100.00
```

The remaining individuals have some kind of college degree. Use the command **generate college= education>40** to generate a dummy variable for this category.

```
. generate college= education>40

. tab college

    college |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        128       64.00       64.00
          1 |         72       36.00      100.00
------------+-----------------------------------
      Total |        200      100.00
```

REGRESSIONS

Once your data are in Stata format and you have generated any necessary variables, you can begin to run your regressions.

LINEAR REGRESSION

The command for a ordinary least squares (OLS) regression is simply **regress** followed by the variables you are using, with the dependent variable (what you are predicting, the y variable) listed first, followed by any independent variables: **regress yvar xvar1 xvar2** …

Using the sample dataset, I could run a regression to look for a relationship between age and lwage, the log wage variable that I generated earlier.

```
. regress lwage age

      Source |       SS       df       MS              Number of obs =     110
-------------+------------------------------           F(  1,   108) =    2.60
       Model |  .791520315     1  .791520315           Prob > F      =  0.1096
    Residual |  32.8370879   108   .30404711           R-squared     =  0.0235
-------------+------------------------------           Adj R-squared =  0.0145
       Total |  33.6286082   109  .308519342           Root MSE      =   .5514

------------------------------------------------------------------------------
       lwage |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .0060651    .003759     1.61   0.110     -.001386    .0135161
       _cons |   2.473424   .1625819    15.21   0.000     2.151158    2.795689
------------------------------------------------------------------------------
```

I can add more x-variables to the regression:

```
. regress lwage male age hsdiploma college

      Source |       SS       df       MS              Number of obs =     110
-------------+------------------------------           F(  4,   105) =   13.80
       Model |  11.5855578     4  2.89638946           Prob > F      =  0.0000
    Residual |  22.0430504   105  .209933813           R-squared     =  0.3445
-------------+------------------------------           Adj R-squared =  0.3195
       Total |  33.6286082   109  .308519342           Root MSE      =  .45819

------------------------------------------------------------------------------
       lwage |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        male |   .0973496   .0905015     1.08   0.285    -.0820982    .2767974
         age |   .0063431   .0031265     2.03   0.045     .0001438    .0125423
   hsdiploma |    .111912   .1494281     0.75   0.456    -.1843764    .4082004
     college |   .7094032   .1498775     4.73   0.000     .4122237    1.006583
       _cons |   2.079982   .1976793    10.52   0.000     1.688021    2.471944
------------------------------------------------------------------------------
```

Using the option **robust** will display robust standard errors in the regression output.

```
. regress lwage male age hsdiploma college, robust

Linear regression                                      Number of obs =     110
                                                       F(  4,   105) =   15.26
                                                       Prob > F      =  0.0000
                                                       R-squared     =  0.3445
                                                       Root MSE      =  .45819

------------------------------------------------------------------------------
             |               Robust
       lwage |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        male |   .0973496   .0932892     1.04   0.299    -.0876256    .2823248
         age |   .0063431   .0028214     2.25   0.027     .0007486    .0119375
   hsdiploma |    .111912   .1311612     0.85   0.395    -.1481565    .3719805
     college |   .7094032   .1326931     5.35   0.000     .4462972    .9725091
       _cons |   2.079982   .1789866    11.62   0.000     1.725085    2.434879
------------------------------------------------------------------------------
```

PROBIT MODELS

To run another type of regression, like a probit, the syntax is the same: **probit yvar xvar1 xvar2…** Although the coefficients in a probit model do not indicate marginal effects, typing **mfx** directly after running the regression will display a table of the marginal effects.

```
. probit employed age male

Iteration 0:   log likelihood = -136.37092
Iteration 1:   log likelihood = -131.24364
Iteration 2:   log likelihood = -131.23857

Probit regression                                 Number of obs   =        200
                                                  LR chi2(2)      =      10.26
                                                  Prob > chi2     =     0.0059
Log likelihood = -131.23857                       Pseudo R2       =     0.0376

------------------------------------------------------------------------------
    employed |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |  -.0149433   .0051216    -2.92   0.004    -.0249814   -.0049052
        male |   .1755545   .1849013     0.95   0.342    -.1868452    .5379543
       _cons |   .7760095   .2601069     2.98   0.003     .2662093     1.28581
------------------------------------------------------------------------------

. mfx

Marginal effects after probit
      y  = Pr(employed) (predict)
         =  .57581905
------------------------------------------------------------------------------
variable |      dy/dx    Std. Err.     z    P>|z|  [    95% C.I.   ]      X
---------+--------------------------------------------------------------------
     age |  -.0058535      .00201   -2.91   0.004  -.009792 -.001915   44.01
   male* |   .0684797      .07172    0.95   0.340  -.072098  .209058    .415
------------------------------------------------------------------------------
(*) dy/dx is for discrete change of dummy variable from 0 to 1
```

You could also have obtained the marginal effects in a single command by entering **dprobit employed age male**.

DO-FILES

Do-files are programs that you write with all of the commands that you wish to run for your analysis. They have the extension .do.

When you type commands into the command window one at a time, you are working interactively. Creating a do-file is a very useful alternative to working interactively. Do-files allow you (or your professor or a Stata tutor) to easily reproduce your work. Saving your work in a do-file will let you pick up where you left off the last time you used Stata or perform the same set of operations on multiple datasets. Most importantly, if you realize that you made a mistake or forget a command at the beginning of a session, you can edit the do-file and run it again instead of manually re-entering each individual command into the command window. If you get stuck, a Stata tutor can look at your do-file to find out what went wrong.

You should not undertake a research project (or even a long homework) without writing a do-file.

To start a do-file, click on the "New Do-file Editor" button in the Stata toolbar (it looks like a notepad), or select This opens a blank window with a cursor. Type commands into the window, on separate lines, in the order you want Stata to perform them. You can make notes in your do-file by beginning a line with an asterisk (*).

Save your do-file by using the File menu or the Save button in the do-file editor.

There are two ways to carry out the commands in a do-file:

- **Run** performs all the commands in the do-file but does not show any output. If there are commands in the do-file that change the data, the changes will be made.
- Clicking **do** performs all the commands in the do-file and shows output in the results window for each command, just the same as if you had typed the commands into the command window. Typing **save** into the do-file saves your data, not the do-file itself.

You can run or do a do-file by clicking the buttons on the right of the toolbar in the do-file editor.

To open an existing do-file, right-click on the file and select **Edit**. This opens the do-file editor. Alternatively, open the do-file editor and select **Open…** from the **File** menu.

It is also possible to create and edit do files in any other text editor. To invoke them one simply types "do filename.do" on the command line.

RECORDING OUTPUT: LOG FILES

To avoid losing the results of analyses carried out each time you quit Stata, you will have to create a log file. Stata's log files have the extension .smcl. The command to begin a log file is **log using *logname***. This creates a new file, *logname*.smcl, in the current directory, in which everything that was seen in the results window during that session will be recorded. You could also select **Log > Begin…** from the **File** menu to open a log. When you are finished, you type **log close** or just exit Stata to close the log; your output will be saved automatically. The next time you open Stata, you can create a new log with a new name. If you would prefer to keep all your output in one log file, type **log using *logname*, append**, which will append the new log to the end of the existing file. To replace the old log with the new one, type **log using *logname*, replace.** As with any command, these can be incorporated into your do file so that it is automatically done each time you run the program.

To view a log file, select **Log > View…** from the **File** menu and choose the log file, or right-click on the file and select **Open**. This opens Stata's log viewer. You can print a log file from the log viewer to attach it to a problem set. Results can also be copied from the log viewer or from the results window in Stata and pasted into a problem set in Microsoft Word. Tables from Stata do not usually align properly in Times or Times New Roman, but changing the font to Courier or Courier New in size 9 or 10 will fix this. Since even this doesn't look very professional, for a formal research paper the best method is to create a neatly formatted results table using Excel.

MORE HELP

Stata official help website is extensive. You can get there by typing **search** followed by your problem into Stata, or through Google. If that doesn't solve your problem, consulting the Stata manuals can help. The manuals have more detail than the online help and include many useful examples. You can check out the manuals at the library. A set of manuals also is available in Sunderland ILC1; these manuals cannot be removed from the lab. Please be respectful to others and leave them there.

The Economics Department also has Stata tutors who are available to help you. These tutors will hold training sessions at the beginning of the Fall and Spring semesters and and can also be consulted during their office hours. Check the department website for a list of tutors and scheduled office hours.

GOOD COMMANDS TO KNOW

(This list is NOT a comprehensive list of commands that Stata uses, but these are the commands you may use most frequently.)

MANAGING DATA

pwd shows which directory Stata is in

cd U: changes the directory to Tigercat

use "***filename*.dta"** opens a dataset

set mem Xm sets the memory in Stata to X MB

compress may reduce the size of a dataset by storing the data more efficiently

insheet imports a tab-delimited (or comma-delimited) text file into Stata

browse opens the data browser

edit opens the data editor so you can manually edit the data

drop removes specified variables or observations from the dataset

keep command *keeps* the specified variables or observations and removes the rest

sort orders the data in a certain way. Typing **sort age** would order the observations from smallest age to largest age.

save saves your data, a common option is **replace**

append combines two datasets "vertically," adding observations to variables

merge combines datasets "horizontally"

rename *oldname newname* changes a variable name from oldname to newname

label adds a descriptive label to a variable. Type **label var *varname* "label"**

generate creates a new variable

The command **egen** also creates variables. **egen** is usually used for creating variables that have something to do with summary statistics

describe lists variables and their labels.

replace replaces specified observations with a value you can specify.

SUMMARIZING DATA

summarize provides basic summary statistics

tabulate makes a frequency table.

codebook lists information about the values a variable takes

tabstat allows you to customize a table of summary statistics.

REGRESSIONS

regress runs an ordinary linear regression

probit runs a probit model

mfx displays marginal effects for the most recent regression

xtreg performs fixed or random-effects linear regression (for panel data).