# Performing Basic Econometric Analysis in Stata

Todd R. Yarbrough[*]

**Abstract**

The basic tools and commands to perform basic econometric analysis in Stata. Particularly useful for undergraduates working on research projects.

**Keywords**:
**JEL Codes**:

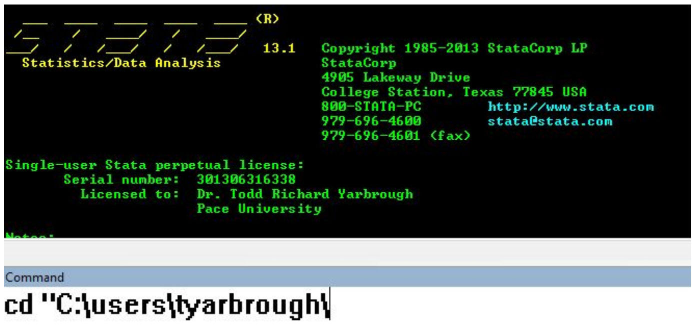
[*]Clinical Assistant Professor, Dyson College of Arts and Sciences at Pace University, 41 Park Row, New York City, NY 10007. Phone: 616-307-1767

## Contents

# 1 Getting Started with Stata

When you first open Stata you'll see a large window in the middle called the Results Window, and just below that is the Command Window. The command window is where you can type commands into Stata and execute them. To the right is a window that will display the variable names from whatever data set you may have loaded, and below that is information on data types (not very important most of the time). Windows users often see another window to the left of the results window which will list any command that have been executed

Figure 1: The Command Window Sits Just Below the Results Window



```
                                         (R)
                                         13.1   Copyright 1985-2013 StataCorp LP
  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 perpetual license:
       Serial number:  301306316338
         Licensed to:  Dr. Todd Richard Yarbrough
                       Pace University
```

```
cd "C:\users\tyarbrough\
```

While it is tempting to rely on Stata's convenient drop down menus, you really want to develop at least cursory Stata coding skills. Stata has it's own coding language, although the language itself is usually very straightforward and intuitive.

## 1.1 Setting Your Working Directory

Stata automatically saves work to a folder called the *Working Directory.* You have the power to name the *working directory* to any file folder you prefer. It is often a good idea to create working directory folders for each specific project you're working on that involve Stata work. Otherwise, Stata will save to the default working directory and it may be difficult to keep log files, do-files, and data sets easily accessible if they're all stuffed into the same folder. To set the working directory we type:

```
. cd "C:\users\tyarbrough\desktop\StataMetrics"
```

2. Once you set your working directory you can place data sets, Do-files, graphs, etc. into the directory and call them up within Stata, which we'll get to a bit later. You can also create new folders inside an existing working directory and then set the new folder to be the new working directory.

```
. mkdir folder_a, public
. cd "folder_a"
```

This creates a new folder (folder_a) within the current working directory. The *public* exists for those situations where many people may share the same parent folder. Otherwise, the new folder will retain whatever file permissions exist on the operating system.

## 1.2 Creating Practice Data Sets

A rare bit of advice when first getting into Stata is to learn how to create data sets, despite the many benefits of learning from very simple and straightforward Stata code. Later on you can use some of these tools in Monte Carlo Simulations, which become useful ways to better understand the concepts underlying Regression Analysis. And without further ado (this is a Stata pun), fire up a session of Stata and let's get started!

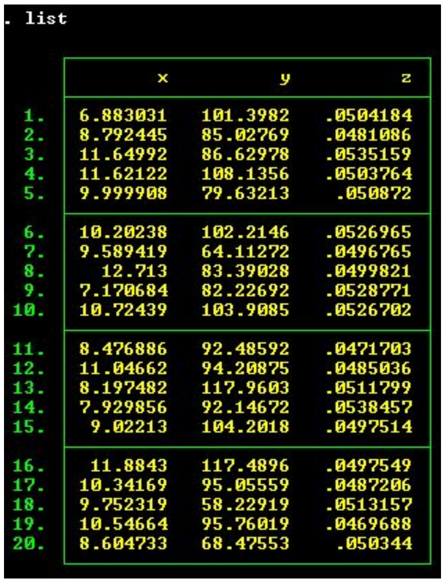
1. The first thing we'll do is create a few variables that will make up our practice data set. We'll come back to this type of practice data set routinely throughout these set of tips, creating specific types of practice data sets to fit specific needs and examples. Execute the following lines one at a time into your command window:

```
. set seed 001
. set obs 20
. gen x = rnormal(10,2)
. gen y = rnormal(100,20)
. gen z = rnormal(0.05, 0.002)
```

First we tell Stata to use a random number generator based on a *seed* number, here that is 001 (it can be anything). *set obs* tells Stata that we wish to create 20 observations of whatever variables we create.

The *generate* command (here shortened to *gen*) allows us to create a new variable. We can create a new variable in a blank data set or as you'll see later we can create a new variable in an existing data set. Here we generate 3 separate variables (x, y, and z), all of which are normally distributed (*rnormal*), with (mean, std. dev.). So, for x the mean is 10 and the standard deviation is 2; while for y the mean is 100 and the std. dev. is 20. And for each variable we should see 20 observations.

Figure 2: Our Practice Data Set!



```
. list

     +----------------------------------+
     |        x          y          z   |
     |----------------------------------|
  1. | 6.883031   101.3982   .0504184   |
  2. | 8.792445   85.02769   .0481086   |
  3. | 11.64992   86.62978   .0535159   |
  4. | 11.62122   108.1356   .0503764   |
  5. | 9.999908   79.63213    .050872   |
     |----------------------------------|
  6. | 10.20238   102.2146   .0526965   |
  7. | 9.589419   64.11272   .0496765   |
  8. |   12.713   83.39028   .0499821   |
  9. | 7.170684   82.22692   .0528771   |
 10. | 10.72439   103.9085   .0526702   |
     |----------------------------------|
 11. | 8.476886   92.48592   .0471703   |
 12. | 11.04662   94.20875   .0485036   |
 13. | 8.197482   117.9603   .0511799   |
 14. | 7.929856   92.14672   .0538457   |
 15. |  9.02213   104.2018   .0497514   |
     |----------------------------------|
 16. |  11.8843   117.4896   .0497549   |
 17. | 10.34169   95.05559   .0487206   |
 18. | 9.752319   58.22919   .0513157   |
 19. | 10.54664   95.76019   .0469688   |
 20. | 8.604733   68.47553    .050344   |
     +----------------------------------+
```

2. We can also create new variables which are transformation of existing variables. For example suppose we wanted to generate a variable w which is the ratio of x to y. We would type:

```
. gen w = x/y
```

3. Here are some other things we can do:

```
. gen b = x - y
. gen d = x*y
. gen l = x^2
. gen f = ln(x)
. gen m = (x+y)/z
. gen k = (2*x) + (0.5*y) + (10*z)
```

Common Variable Operators:

| | |
|---|---|
| + - * / | plus, minus, times, divded by |
| var^2 var^(1/2) | squared, square root |
| ln(var) | natural log |

4. Sometimes we want to create variables which take on particular values when certain conditions of existing variables are met or not met. For example, suppose we wanted to create a variable g which equaled 1 every time x was greater than 10 (its average). To do this we'd type:

```
. gen g = 0
. replace g = 1 if x > 10
```

We can get pretty complicated with the conditions:

```
. gen j = 0
. replace j = 1 if x > 10 & y < 100

. gen p = 0
. replace p = 1 if x > (y/10) & z > 0.05
```

5. We end this section by showing how we may come to understand our data set better. For example, suppose we want to know the mean, standard deviation, minimum value, and maximum value for each variable in our data set. We would use the *summarize or sum* command:

```
. sum
. sum, detail
. describe
. save data01, replace
```

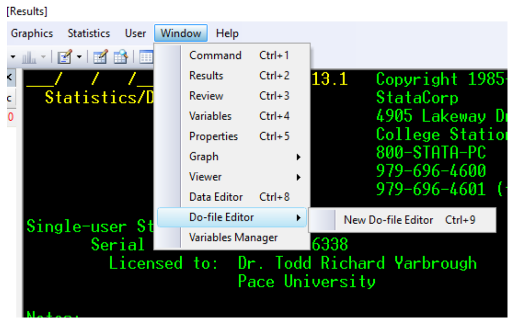
*sum, detail* provides us even more detailed information about our variables, while *describe* provides information on the types of data in the sample (e.g. byte or string). And finally, we can always save the current state of our data set with the *save* command.

******SPECIAL NOTE: there is no back button in Stata. Any change you make to a data set is permanent, so frequent saving of data sets is a good idea. It is also a good idea to keep some sort of consistent file naming system. Perhaps something like *data01.dta*; *data02.dta*; etc.******

## 1.3 Do-files and Logs

### 1.3.1 Do-Files

You'll spend a lot of time re-Googling Stata questions if you don't save what you learn along the way. *Do-files* are like any coding editor, wherein you add lines of code in a specific order so that the actions may be replicated. Do-files are especially useful if data require extensive manipulation, as you can create a Do-file of your commands. You then 'run' your Do-file and it executes each line of code, performing your desired tasks. To access the Do-file editor, navigate to the window drop down menu in the Stata client. Click 'Do-file Editor' and then 'New Do-file Editor.'



```
 ___/   /   /___          13.1   Copyright 1985-
  Statistics/D                   StataCorp
                                 4905 Lakeway Dr
                                 College Statio
                                 800-STATA-PC
                                 979-696-4600
                                 979-696-4601 (

Single-user St
      Serial                6338
        Licensed to:  Dr. Todd Richard Yarbrough
                      Pace University
```

Add the following four lines of code to your do-file:

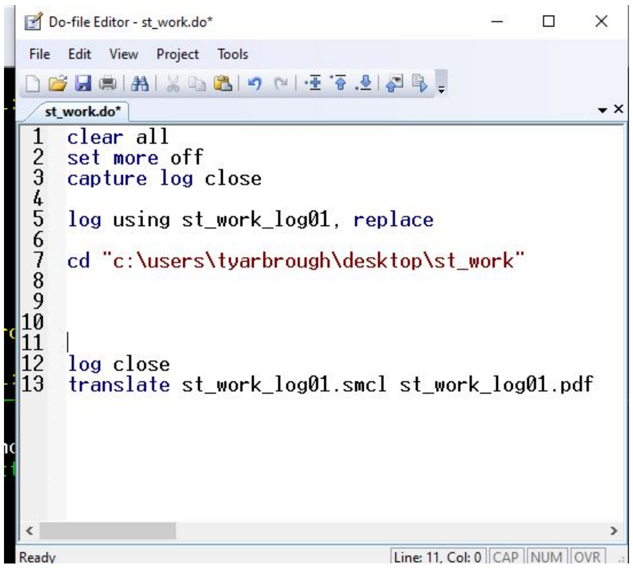
```
. log using log01, replace
. clear all
. set more off
.
.
.
. capture log close
```

These four lines of code are usually added to the beginning of all Do-files because they make it easy to quickly run the file. *log using <desired file name>* fires up a fresh log file.*clear all* clears any open data in the current Stata session. *set more off* turns off the annoying Stata feature which forces you to click or hit the space bar to advance through analysis. *capture log close* closes any currently open *Log Files.* Note: the *replace* option after the comma in the fourth line of code opens a log file named <desired file name> and replaces any existing log file named <desired file name>. This is so the code can be run multiple times without having to name the log files something new each time. The *replace* option will be used extensively while using Stata.

Finally, we'll add two final lines of code, giving ourselves some lines in between *cd...* and the following:

```
. log close
. translate log01.smcl log01.pdf

or

. translate log01.smcl log01.txt
```

*log close* is self-explanatory, while *translate...* converts the log file (.smcl) to a PDF. You can also convert to .txt file.


Do-file Editor - st_work.do*
File Edit View Project Tools
st_work.do*
Ready
Line: 11, Col: 0 CAP NUM OVR

```
clear all
set more off
capture log close

log using st_work_log01, replace

cd "c:\users\tyarbrough\desktop\st_work"



log close
translate st_work_log01.smcl st_work_log01.pdf
```

This will be useful later, but you can also run a do-file that is located in the working directory from the command prompt in Stata by typing:

```
. do dofilename
```

For more information on do-files, consult the Stata Documentation on Do-files

### 1.3.2 Using Log Files

1. To start a log, simply type the following code:

```
. log using log01, replace
```

The *replace* option will write over any existing file with the same name. It is useful when you will run the same code over and over many times but do not want to keep renaming your files, including log files.

2. Other important commands involving logs can pause them (*log off*), restart them (*log on*), and close them (*log close*).

```
. log off
. log on
. log close
```

3. We can also turn our log files, which by default are .smcl files, into other file types such as PDF or TXT.

```
. type log01.smcl
. translate log01.smcl log01.pdf
. translate log01.smcl log01.txt
```

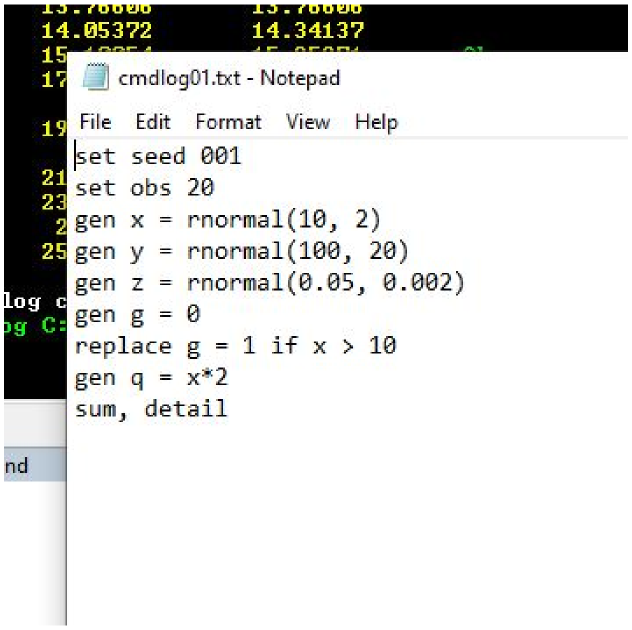
4. A nice go between for do-files and log files is the Command Log, produced by typing:

```
. cmdlog using cmdlog01, replace
```

The command log produces only the commands typed during your session (or the duration of the open log) and not the results. This makes command logs ideal for quickly taking your Stata work into a do-file. You'll want to make note of which commands produce errors, so you can remove them from your do-files. Command files are always TXT files.

All of your files will be located in whatever working directory you have set. For more information consult the manual sections on do files and log files.

Figure 3: An Example of a CMD Log (Command Log)



```
set seed 001
set obs 20
gen x = rnormal(10, 2)
gen y = rnormal(100, 20)
gen z = rnormal(0.05, 0.002)
gen g = 0
replace g = 1 if x > 10
gen q = x*2
sum, detail
```

You can see an example of a full Log File on the next page.

Statistics/Data Analysis

```
      name:  <unnamed>
       log:  C:\users\tyarbrough\desktop\StataMetrics\log_prac.smcl
  log type:  smcl
 opened on:  22 Nov 2021, 00:00:12

1 . cmdlog using cmdlog01, replace
  (cmdlog C:\users\tyarbrough\desktop\StataMetrics\cmdlog01.txt opened)

2 . cd "C:\users\tyarbrough\desktop\StataMetrics"
  C:\users\tyarbrough\desktop\StataMetrics

3 . set more off

4 . set seed 001

5 . set obs 20
  obs was 0, now 20

6 . gen x = rnormal(10,2)

7 . gen y = rnormal(100,20)

8 . gen z = rnormal(0.05, 0.002)

9 . gen b = x - y

10. gen d = x*y

11. gen l = x^2

12. gen f = ln(x)

13. gen m = (x+y)/z

14. gen k = (2*x) + (0.5*y) + (10*z)

15. gen g = 0

16. replace g = 1 if x > 10
  (9 real changes made)

17. gen j = 0

18. replace j = 1 if x > 10 & y < 100
  (5 real changes made)

19. gen p = 0

20. replace p = 1 if x > (y/10) & z > 0.05
  (6 real changes made)

21. sum

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
           x |         20    9.757453    1.611272   6.883031     12.713
           y |         20     91.6345    16.16517   58.22919   117.9603
           z |         20    .0504374    .0019894   .0469688   .0538457
           b |         20   -81.87705    16.06299  -109.7628  -48.47688
           d |         20    896.9168    232.1083   567.8697   1396.282
-------------+---------------------------------------------------------
           l |         20    97.67427    31.45111   47.37612   161.6205
           f |         20    2.264636    .1696034   1.929059   2.542625
           m |         20    2013.637    336.8971   1324.771   2600.223
           k |         20    65.83653    9.032266   49.13239   83.01097
           g |         20         .45    .5104178          0          1
-------------+---------------------------------------------------------
           j |         20         .25    .4442617          0          1
           p |         20          .3    .4701623          0          1
```

### 1.4 Calling Up Data

1. With a .dta file in the current working directory, you can call it up by typing:

```
. use data01, clear
. use "C:\users\tyarbrough\documents\data\data01, clear
```

2. If you instead have an Excel, TXT, or CSV file you can still import directly into Stata.

```
. import excel data01.xlsx, firstrow case(lower)
```

This imports the data with first row as variable names in lower case.

You can also import a portion of an excel file.

```
. import excel data01.xlsx, cellrange(A1-C21) firstrow case(lower)
. import excel data01.xlsx, cellrange(A1-C21) sheet("sheetname") firstrow case(lower)
```

3. You can also import TXT or CSV files.

```
. import delimited "data01.csv"
. import delimited "data01.csv", varnames(5)  <skips 4 rows, sets 5th row as var names>
```

4. There are some online sources of data that you can call up in Stata automatically.

## 2 Data Cleaning and Manipulation

Often times you will find a data set that has many more/less observations or variables than you need, or you want to combine data sets, or you need to transpose data. Manually cleaning your data in excel can be a real pain as you no doubt have experienced. Luckily Stata offers a broad suite of commands that can greatly reduce the stress of data cleaning.

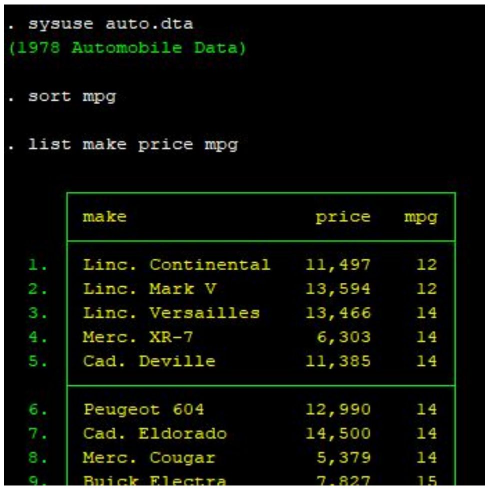
### 2.1 Sorting

1. Sometimes you want to have your data sorted by some particular variable and by some particular method (alphabetical or ascending/descending). Once you have your data called up into a Stata work frame, sorting on specific variables is quite easy and convenient.

Suppose we are using a data set called "auto.dta" which is available in each version of Stata. Let's sort by mileage (ascending) and then create a list with make, price, and mileage.

```
. sysuse auto.dta, clear
. sort mpg
. list name price mpg
```

Figure 4: List of Name, Price, and MPG: Sorted by MPG

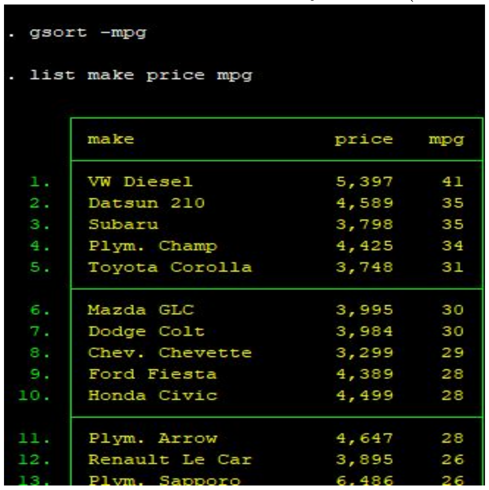


```
. sysuse auto.dta
(1978 Automobile Data)

. sort mpg

. list make price mpg

     +-----------------------------------+
     | make                  price   mpg |
     |-----------------------------------|
  1. | Linc. Continental    11,497    12 |
  2. | Linc. Mark V         13,594    12 |
  3. | Linc. Versailles     13,466    14 |
  4. | Merc. XR-7            6,303    14 |
  5. | Cad. Deville         11,385    14 |
     |-----------------------------------|
  6. | Peugeot 604          12,990    14 |
  7. | Cad. Eldorado        14,500    14 |
  8. | Merc. Cougar          5,379    14 |
  9. | Buick Electra         7,827    15 |
```

2. Suppose instead we want to sort on mileage in a descending fashion:

```
. gsort -mpg
. list name price mpg
```

Figure 5: Previous List Sorted by MPG (Descending)

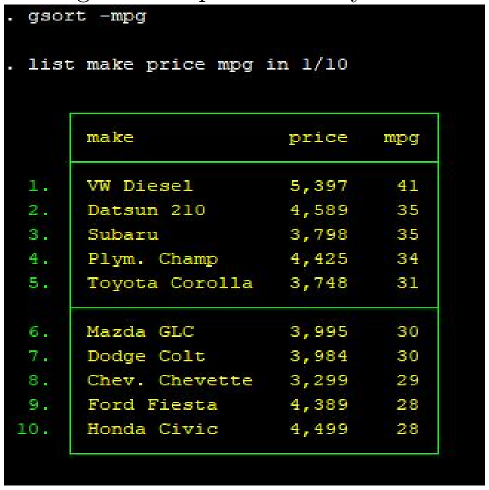


```
. gsort -mpg

. list make price mpg

     +-----------------------------------+
     | make                  price   mpg |
     |-----------------------------------|
  1. | VW Diesel             5,397    41 |
  2. | Datsun 210            4,589    35 |
  3. | Subaru                3,798    35 |
  4. | Plym. Champ           4,425    34 |
  5. | Toyota Corolla        3,748    31 |
     |-----------------------------------|
  6. | Mazda GLC             3,995    30 |
  7. | Dodge Colt            3,984    30 |
  8. | Chev. Chevette        3,299    29 |
  9. | Ford Fiesta           4,389    28 |
 10. | Honda Civic           4,499    28 |
     |-----------------------------------|
 11. | Plym. Arrow           4,647    28 |
 12. | Renault Le Car        3,895    26 |
 13. | Plym. Sapporo         6,486    26 |
```

3. Maybe we just want to list the top ten cars by mpg:

```
. list make price mpg in 1/10
```

Figure 6: Top Ten Cars by MPG



```
. gsort -mpg

. list make price mpg in 1/10

     +-----------------------------+
     | make              price  mpg |
     |-----------------------------|
  1. | VW Diesel         5,397   41 |
  2. | Datsun 210        4,589   35 |
  3. | Subaru            3,798   35 |
  4. | Plym. Champ       4,425   34 |
  5. | Toyota Corolla    3,748   31 |
     |-----------------------------|
  6. | Mazda GLC         3,995   30 |
  7. | Dodge Colt        3,984   30 |
  8. | Chev. Chevette    3,299   29 |
  9. | Ford Fiesta       4,389   28 |
 10. | Honda Civic       4,499   28 |
     +-----------------------------+
```

4. Now, sometimes you want to put your variables in a different order than the base data set. Suppose we wish to reorder our variables with price listed first then mileage then make, with other variables staying in the same order after these three.

```
. order price mpg make
```

## 2.2 Subsetting

One of the more useful tools is the ability to create subsets of data with your existing data.

1. First, we can drop variables we're not interested in and save the ones we are to a new data set. Here we keep price, make, and mpg:

```
. sysuse auto, clear
. save auto, replace
. order price mpg make
. keep price mpg make
. save newauto, replace
```

2. At this point we can go back to the base data set, secure in the idea that our newly organized and subsetted data is in our working directory and easily called up with a simple "use filename" command (you don't even have to include the .dta!).

We go back to the data set easily by the *use data, clear* command. The really nice benefit of this method is that it does not require the altering of the base data set whatsoever. The above commands allow you to subset your initial data by creating a new data set.[1]

```
. use auto, clear
. use newauto, clear
```

3. What if we only want data on 1) vehicles that cost less than $8,000; 2) the top ten cars ranked from highest to lower by mpg.

```
/*1*/
. keep if price < 8000
. save auto_highprice, replace
/*2*/
. gsort -mpg
. keep make mpg price in 1/10
. save auto_bestmpg, replace
```

## 2.3 Merging

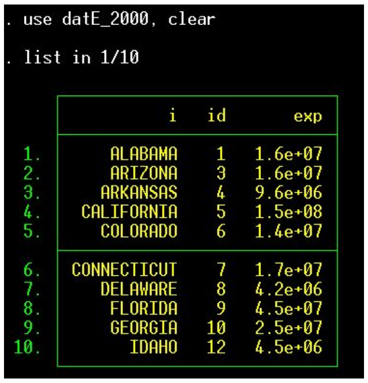
### 2.3.1 With 1 Unique Identifier

Sometimes we find ourselves with data that is split across two data sets. For example, you may have one data set that contains all expenditures for the U.S. states and another with all revenues for the U.S. states. We want to have both of these variables in the same data set. To do this we need to *Merge* the two data sets.

1. In order to be able to *merge* your data sets they need at least some observations where their unique identifiers are consistent. In our U.S. state example the unique identifiers would all be consistent. In this example we have two data sets datE.dta and datR.dta which contain expenditures for 5 U.S. states and revenues for those same states respectively. NOTE: to make this as easy as possible place both data sets in the same folder and set that folder as your working directory. Here are the two data sets.

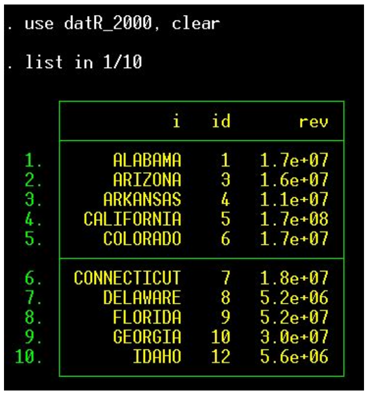
datE_2000.dta
datR_2000.dta

Data E



```
. use datE_2000, clear

. list in 1/10

              i   id        exp

  1.    ALABAMA    1    1.6e+07
  2.    ARIZONA    3    1.6e+07
  3.   ARKANSAS    4    9.6e+06
  4. CALIFORNIA    5    1.5e+08
  5.   COLORADO    6    1.4e+07

  6. CONNECTICUT   7    1.7e+07
  7.   DELAWARE    8    4.2e+06
  8.    FLORIDA    9    4.5e+07
  9.    GEORGIA   10    2.5e+07
 10.      IDAHO   12    4.5e+06
```

Data R



```
. use datR_2000, clear

. list in 1/10

              i   id        rev

  1.    ALABAMA    1    1.7e+07
  2.    ARIZONA    3    1.6e+07
  3.   ARKANSAS    4    1.1e+07
  4. CALIFORNIA    5    1.7e+08
  5.   COLORADO    6    1.7e+07

  6. CONNECTICUT   7    1.8e+07
  7.   DELAWARE    8    5.2e+06
  8.    FLORIDA    9    5.2e+07
  9.    GEORGIA   10    3.0e+07
 10.      IDAHO   12    5.6e+06
```

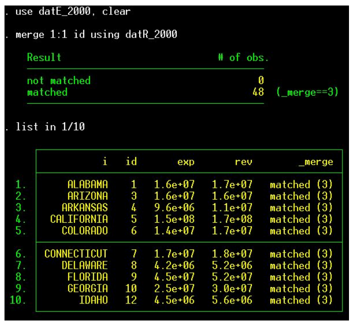
2. Here we can see that the states are unique identifiers across the two data sets. So we can perform a very simple *merge* exercise. First, we need to call up one of the data sets (datE.dta).

```
. use datE, clear
```

3. Then we merge the two data sets with:

```
. merge 1:1 id using datR.dta
```

Which should produce the following data set:



```
. use datE_2000, clear

. merge 1:1 id using datR_2000

    Result                           # of obs.
    -----------------------------------------
    not matched                             0
    matched                                48  (_merge==3)
    -----------------------------------------

. list in 1/10

     +-------------------------------------------------------+
     |           i   id       exp       rev          _merge |
     |-------------------------------------------------------|
  1. |     ALABAMA    1   1.6e+07   1.7e+07     matched (3) |
  2. |     ARIZONA    3   1.6e+07   1.6e+07     matched (3) |
  3. |    ARKANSAS    4   9.6e+06   1.1e+07     matched (3) |
  4. |  CALIFORNIA    5   1.5e+08   1.7e+08     matched (3) |
  5. |    COLORADO    6   1.4e+07   1.7e+07     matched (3) |
     |-------------------------------------------------------|
  6. | CONNECTICUT    7   1.7e+07   1.8e+07     matched (3) |
  7. |    DELAWARE    8   4.2e+06   5.2e+06     matched (3) |
  8. |     FLORIDA    9   4.5e+07   5.2e+07     matched (3) |
  9. |     GEORGIA   10   2.5e+07   3.0e+07     matched (3) |
 10. |       IDAHO   12   4.5e+06   5.6e+06     matched (3) |
     +-------------------------------------------------------+
```

4. Lastly, you typically want to get rid of the variable _merge that was created.

```
. drop _merge
```

### 2.3.2 *merge* with 2 unique identifiers

1. Often you're working with Panel Data and you have two unique identifiers (e.g. states and years). Merge is able to handle this easily with a minor change to our previous code. Here are the two data sets.
statepanel01.dta
statepanel02.dta

These data are US states over the years 1970-2015. statepanel01 has demographic and socioeconomic data along with some fiscal variables. statepanel02 has natural disaster crop damage and property damage. We start by loading both data sets and sorting them with respect to the panel data we have (states over time). You need to have your data sorted before you can merge.

```
. use statepanel01, clear
. xtset stateid year
. save statepanel01, replace
. use statepanel02, clear
. xtset stateid year
. save statepanel02, replace
. merge 1:1 stateid year using statepanel01
. drop _merge
. save statepanel0102, replace
```

2. Sometimes we're unlucky that our panels that we want to merge don't completely line up. For example, suppose in the previous example we were missing some states and some years randomly. We can handle this too by altering our previous code.

```
. merge m:m stateid year using statepanel01
```

Where *m:m* allows for the identifiers to not be completely balanced. Now, this will result in a merged data set with some unmatched entries, but it is the best we can do if our data truly is missing those entities and/or time periods.

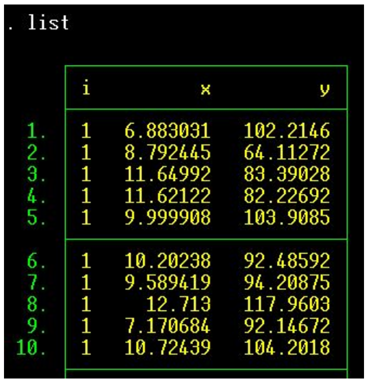
Here is the Do-file.
Here is the Do-file as .txt file.
For more on the *merge* command from Stata.

## 2.4 Appending

Suppose that you have two data sets which have all the same columns and you wish to add one data to another creating one large data set. We can use the *Append* command to do just that. See this simple example below.

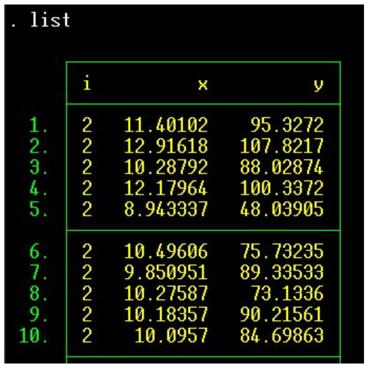
1. You have two data sets (datA.dta and datB.dta) that contain all the same columns (variables x and y) but have a different unique identifier i (1 or 2).

Data A



```
. list

       i          x          y
  1.   1   6.883031   102.2146
  2.   1   8.792445   64.11272
  3.   1   11.64992   83.39028
  4.   1   11.62122   82.22692
  5.   1   9.999908   103.9085

  6.   1   10.20238   92.48592
  7.   1   9.589419   94.20875
  8.   1     12.713   117.9603
  9.   1   7.170684   92.14672
 10.   1   10.72439   104.2018
```

Data B

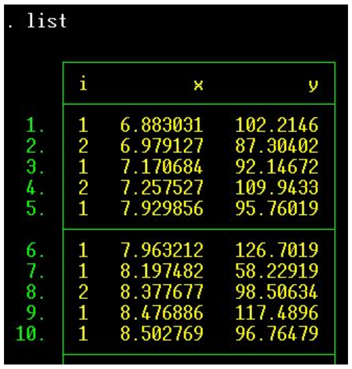


```
. list

       i          x          y
  1.   2   11.40102    95.3272
  2.   2   12.91618   107.8217
  3.   2   10.28792   88.02874
  4.   2   12.17964   100.3372
  5.   2   8.943337   48.03905

  6.   2   10.49606   75.73235
  7.   2   9.850951   89.33533
  8.   2   10.27587    73.1336
  9.   2   10.18357   90.21561
 10.   2    10.0957   84.69863
```

2. We want to make a single data set (A + B) which retains the same columns. This is quite easy. We just need to have data set A called up. Then we will *append* data set B to it:

```
. use datA, clear
. append using datB
. save datAB.dta
```

3. And now we have a single data set (datAB.dta) with observations for both i = 1 and i = 2 over variables x and y. Here we've sorted the data by x (. sort x) to show that both observations for i=1 and i=2 are now in the same data set.



```
. list

       i          x          y
  1.   1   6.883031   102.2146
  2.   2   6.979127   87.30402
  3.   1   7.170684   92.14672
  4.   2   7.257527   109.9433
  5.   1   7.929856   95.76019

  6.   1   7.963212   126.7019
  7.   1   8.197482   58.22919
  8.   2   8.377677   98.50634
  9.   1   8.476886   117.4896
 10.   1   8.502769   96.76479
```

Here is the Do-file.
Here is the Do-file as .txt file.
For more on the *append* command from Stata.

## 2.5 Reshaping

1. One of the more difficult things about data cleaning is that you may down load data that is in one format (e.g. "wide") when you need it in another format (e.g. "long"). In Stata, we need to work with exclusively long data once we want to run regressions. So, let's take a data set that is wide and transform it into long with the *reshape* command. Here is a data set on GDP for Brazil from 1960-present.
Excel File for Brazilian Data

2.
full code:

```
clear all
set more off
bcuse airfare, clear
drop if id > 2
keep year id fare
list
reshape wide fare, i(id) j(year)
list
save newairfare, replace
reshape long fare, i(id) j(year)
list
clear all
log close
```

## 2.6 Generating Variables

To create a variable, you use the "generate" command, which you can shorten to "gen".

**. gen x** <this would create a variable x, which would have no values>

We can also generate variables that take on values of mathematical functions of existing data in our sample. Suppose we have existing data with variables $j$ and $k$.

```
. gen x = j + k

. gen x = j - k

. gen x = j/k

. gen x = j*k
```

Some important transformations that you'll likely use at some point are taking natural logs, creating lagged variables, and percentage change:

```
. gen logx = ln(x)}

. gen lagx = x[_n-1]

. gen leadx = x[_n+1]

. gen chgx = (x - lx)/(lx)*100
```

In the above code examples, "lagx"/"leadx" is a one-period lag/lead of the variable x. For example, if your data is by year, then this would generate a variable that takes the value the variable took one year ago/one year from now. We can create lags and leads of any size.

### 2.6.1 Indicator Variables

Often you wan to create variables that are indicators, meaning they indicate a variable takes on a certain value. This comes up a lot.

For example, suppose we're using the the "CARD.dta". These data contain observations of individuals across several variables, including an "educ" variable which observes the number of years of education. Since we typically want to consider issues related to college graduates, we may wish to distinguish those individuals who have a college degree from those who do not.

**. gen col = 0** <this generates a variable named "col">
. replace col = 1 if educ >= 16

This replaces the 0s of "col" with 1 if the person has had 16 or greater years of schooling, which is the typical cut-off for a college degree. Note: it is possible to have greater than 15 years of schooling and not have a college degree.

We could also create indicator variables for those with some high school.

**. gen hs = 0** <this generates a variable named "col">
**. replace hs = 1 if educ > 8 & educ < 13**

We also often want to create interaction indicators. For example, suppose we wanted to created an indicator of those in the data that are both married and have had some college.

**. gen marcol = 0**
**. replace marcol = 1 if married = 1 & col = 1**
full code:

```
gen x = j + k
gen x = j - k
gen x = j/k
gen x = j*k

gen lx = ln(x)
gen lagx = x[_n]

gen chgx = (x - lx)/(lx)*100

gen x = 0
replace x = 1 if <some condition>

gen college = 0
replace college = 1 if educ >= 16

gen somecollege = 0
replace somecollege = 1 if educ > 12 & educ < 16

gen blcoll = 0
replace bcoll = 1 if college = 1 & black = 1
```

## 2.7 Dealing with Missing Data

You need to be careful with missing data, for both statistical and practical reasons. Typically, Stata will interpret a missing value in your data (an empty cell in the spread sheet of data) as just that, missing. Stata will give this data observation a ".". It is very important that missing values are not actually values of zero! Otherwise, Stata will interpret what is supposed to be a zero as actually missing, which drastically affects the resulting estimations.

So, really make sure you understand your data, and what a missing value really means.

If data is still missing in your data set after loading into Stata, then we can very easily convert those blank cells into the official missing value character for Stata: .

To do this, we can do a simple replace.

**. replace varname =. if missing(varname)**

We may sometimes want to drop missing data.

**. drop if missing(varname)**

# 3 Regressions and Diagnostics

## 3.1 Cross-Sectional Models

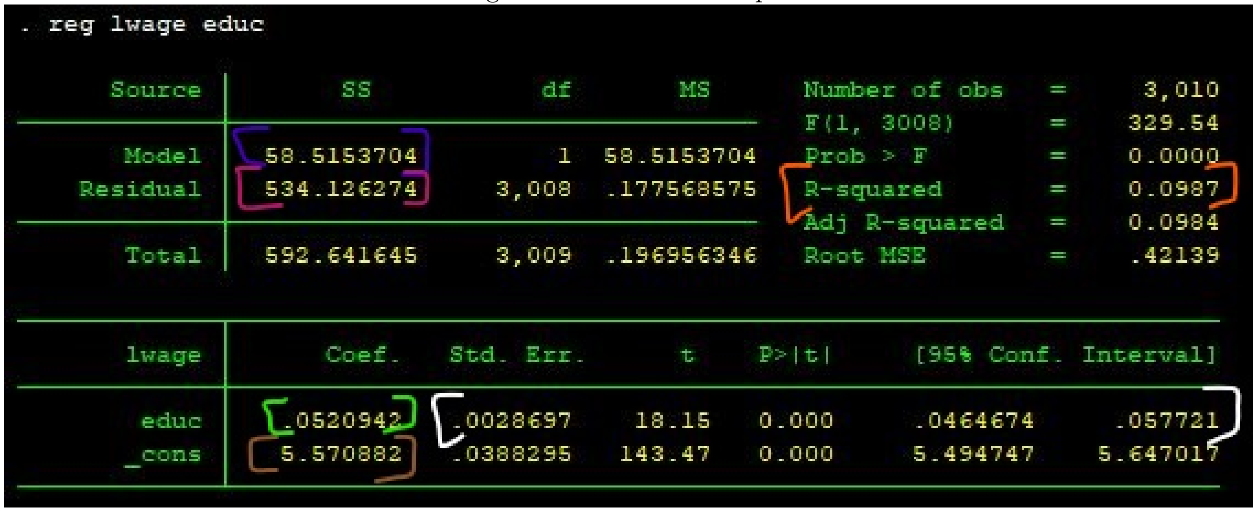
To regress a single independent variable on a dependent variable with cross-sectional data:

**. reg y x** <OLS regression of model: y = b0 + b1*x + u>

An example using the "CARD.dta" data. Regressing years of education on the log of hourly wages:

**. reg lwage educ** <OLS regression of model: lwage = b0 + b1*educ + u>

Figure 7: OLS Stata Output



```
. reg lwage educ

      Source |       SS           df       MS      Number of obs   =     3,010
-------------+----------------------------------   F(1, 3008)      =    329.54
       Model |  58.5153704         1  58.5153704   Prob > F        =    0.0000
    Residual |  534.126274     3,008  .177568575   R-squared       =    0.0987
-------------+----------------------------------   Adj R-squared   =    0.0984
       Total |  592.641645     3,009  .196956346   Root MSE        =    .42139

------------------------------------------------------------------------------
       lwage |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        educ |   .0520942   .0028697    18.15   0.000     .0464674     .057721
       _cons |   5.570882   .0388295   143.47   0.000     5.494747    5.647017
------------------------------------------------------------------------------
```

The Green brackets: this is the coefficient estimation, this is your $\beta_1$

The White brackets: these are your coefficient statistics, we use these to determine statistical significance

The Orange brackets: this is your $R^2$, is the percentage of the variation in $y$ explained by your model. And this is computed using the numbers in the blue and purple brackets. The blue bracket is your SST, while the purple bracket is your SSR.

The Brown brackets: this is the constant, this is your $\beta_0$

Full Code

```
clear all
capture log close
set more off
log using regression001, replace
cd "your working directory"
use CARD.dta, clear
describe
sum
reg lwage educ
```

**Full Code**
**using CARD.dta**

```
clear all
capture log close
set more off
log using regression001, replace
cd "your working directory"
use CARD.dta, clear
graph twoway (lfitci lwage educ) (scatter lwage educ), ///
title(Wages and Education) xtitle("Education in Years") ytitle("Log of Wages")
graph save scatter001, replace
```

## 3.2 Heteroskedasticity and Weighted Least Squares

We often want to check for heteroskedasticity in our model. This is easy to do in Stata. First, let's run a regression:

**. quietly: reg lwage educ** <the quietly command runs the regression without the resulting output>

we then want to calculate the predictions of our independent variable, here this is *educ*:

**. predict exp_educ** <this computes the expected value of *educ*>

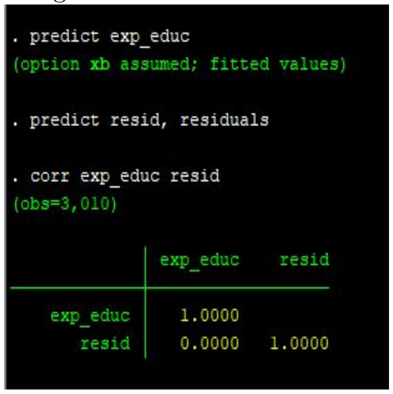
then we need to predict the residuals:

**. predict resid, residuals** <"resid" names the variable, while "residuals" tells Stata to compute residuals>

Then we can just run a simple covariance matrix to see if $u$ and $x$ are related:

**. corr exp_educ resid**

Figure 8: Correlation Matrix

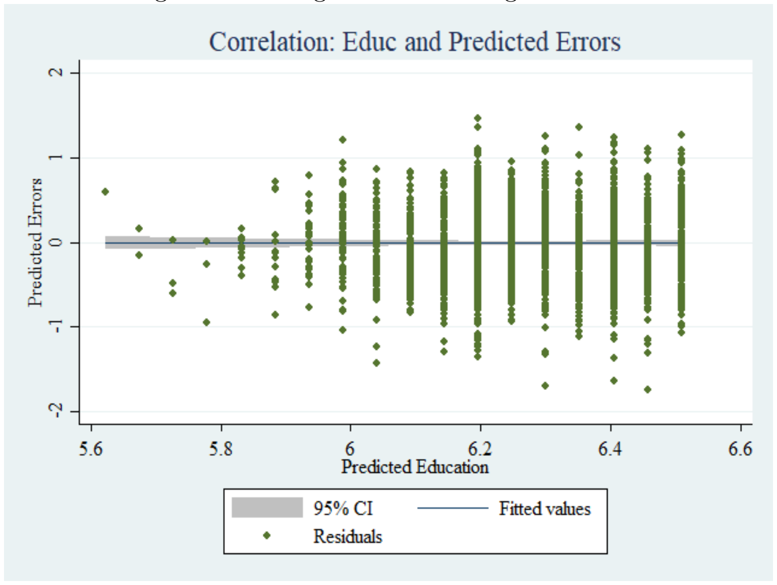


```
. predict exp_educ
(option xb assumed; fitted values)

. predict resid, residuals

. corr exp_educ resid
(obs=3,010)

             | exp_educ    resid
-------------+------------------
    exp_educ |   1.0000
       resid |   0.0000   1.0000
```

Graphically:

**. graph twoway (lfitci exp_educ resid) (scatter exp_educ resid)**

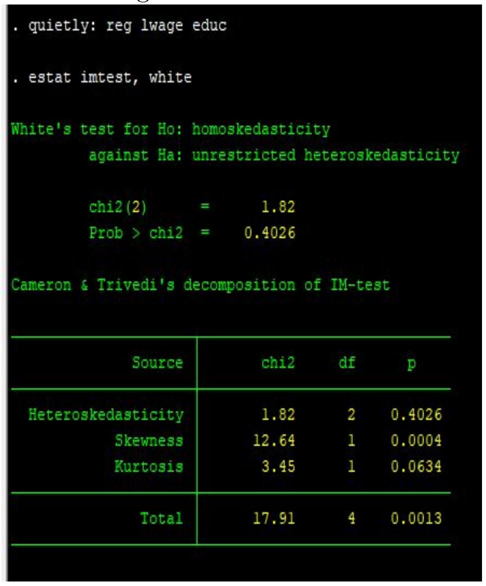
Figure 9: Fitting a Line to u Regressed on x



Or, we can just run a White Test within Stata to check for Heteroskedasticity:

**. quietly: reg lwage educ**
**. estat imtest, white**

Figure 10: White Test



```
. quietly: reg lwage educ

. estat imtest, white

White's test for Ho: homoskedasticity
         against Ha: unrestricted heteroskedasticity

         chi2(2)      =      1.82
         Prob > chi2  =    0.4026

Cameron & Trivedi's decomposition of IM-test

---------------------------------------------------
              Source |       chi2     df      p
---------------------+-----------------------------
  Heteroskedasticity |       1.82      2    0.4026
            Skewness |      12.64      1    0.0004
            Kurtosis |       3.45      1    0.0634
---------------------+-----------------------------
               Total |      17.91      4    0.0013
---------------------------------------------------
```

The rule of thumb is that a Het Score below 8 suggests Homoskedasticity.

**The Full Code**

**using CARD.dta**

```
clear all
capture log close
set more off
log using regression001, replace
cd "your working directory"
use CARD.dta, clear
describe
sum
reg lwage educ
graph twoway (lfitci lwage educ) (scatter lwage educ), ///
title(Wages and Education) xtitle("Education in Years") ytitle("Log of Wages")
graph save scatter001, replace
quietly: reg lwage educ
predict exp_educ
predict resid, residuals
corr exp_educ resid
graph twoway (lfitci resid exp_educ) (scatter resid exp_educ), ///
title(Correlation: Educ and Predicted Errors) xtitle("Predicted Education") ///
ytitle("Predicted Errors")
graph save resid001, replace
quietly: reg lwage educ
estat imtest, white
clear all
log close
```

## 3.3 Time-Series

```
clear all
set seed 001
set obs 100
seq j, f(1) t(100)
gen t = j+1889
gen x = rnormal(10000, 2000)
gen u = rnormal(5000)
gen y = (10000*ln(t)) + (25*ln(x)) + (5*u)
tsset t
tsline y
reg y x
estat dwatson
reg y L.y x
reg y L.y L2.y x
newey y x, lag(2)
gen lny = ln(y)
reg lny x
```

```
tsset t
tsline x
```

### 3.4 Panel Data

## 4 Making Nice Tables

### 4.1 Summary Statistics Tables

```
ssc install estout
sysuse auto
eststo clear
quietly eststo summstats: estpost summarize price length trunk headroom
esttab using sumstats.rtf, cell("mean sd min max") mtitle("Summary Statistics")
```

Table 1: Summary Statistics Table

| | mean | sd | min | max |
|---|---|---|---|---|
| price | 6165.257 | 2949.496 | 3291 | 15906 |
| length | 187.9324 | 22.26634 | 142 | 233 |
| trunk | 13.75676 | 4.277404 | 5 | 23 |
| headroom | 2.993243 | .8459948 | 1.5 | 5 |
| *N* | 74 | | | |

### 4.2 Regression Tables

```
sysuse auto, clear
eststo clear
/*3 OLS regressions*/
eststo: quietly reg price length
eststo: quietly reg price length, vce(cluster make)
eststo: quietly reg price length weight foreign
/*1 WLS regression*/
eststo: quietly wls0 price length weight foreign, wvar(length) type(abse) noconst
/*output table (this is one line of code)*/
esttab using results_cs.rtf, replace n se r2 ar2 star(* 0.10 ** 0.05 *** 0.01) ///
mlabel("OLS" "OLS w/ RSE" "OLS" "WLS")
```

Table 2: OLS and WLS Regressions

| | (1) | (2) | (3) | (4) |
|---|---|---|---|---|
| | OLS | OLS w/ RSE | OLS | WLS |
| length | 57.20$^{***}$ | 57.20$^{***}$ | -91.37$^{***}$ | -62.24$^{*}$ |
| | (14.08) | (12.06) | (32.83) | (31.84) |
| weight | | | 5.775$^{***}$ | 4.725$^{***}$ |
| | | | (0.959) | (0.948) |
| foreign | | | 3573.1$^{***}$ | 3088.8$^{***}$ |
| | | | (639.3) | (588.8) |
| _cons | -4584.9$^{*}$ | -4584.9$^{**}$ | 4838.0 | 2677.2 |
| | (2664.4) | (2074.9) | (3742.0) | (3549.6) |
| $N$ | 74 | 74 | 74 | 74 |
| $R^2$ | 0.186 | 0.186 | 0.549 | 0.501 |
| adj. $R^2$ | 0.175 | 0.175 | 0.530 | 0.480 |

Standard errors in parentheses

$^{*}$ $p < 0.10$, $^{**}$ $p < 0.05$, $^{***}$ $p < 0.01$

### 4.2.1 Panel Regressions

```
bcuse fatality, clear
xtset state year
eststo clear
/*randome effects*/
eststo: quietly xtreg mralln beertax jaild spircons
quietly estadd local fixedstate "No", replace
quietly estadd local fixedyear "No", replace
/*fixed effects*/
eststo: quietly xtreg mralln beertax jaild spircons, fe
quietly estadd local fixedstate "Yes", replace
quietly estadd local fixedyear "No", replace
/*two-way fixed effects*/
eststo: quietly xtreg mralln beertax jaild spircons i.year, fe
quietly estadd local fixedstate "Yes", replace
quietly estadd local fixedyear "Yes", replace
/*output table (this is one line of code)*/
esttab using reults01.rtf, replace se star(* 0.10 ** 0.05 *** 0.01) ///
drop(*year _cons) s(fixedstate fixedyear r2_ wr2_b r2_o N, ///
label("Fixed Effects" "Time Effects" "Within r2" "Between r2" ///
"Overall r2" "Obs"))
```

Table 3: Random and Fixed Effects Regression

| | (1) | (2) | (3) |
|---|---|---|---|
| | ln_mralln | ln_mralln | ln_mralln |
| beertax | 0.0660 | -0.247 | -0.271$^{*}$ |
| | (0.0642) | (0.156) | (0.156) |
| jaild | -0.0335 | -0.0958$^{*}$ | -0.0385 |
| | (0.0473) | (0.0571) | (0.0563) |
| spircons | 0.111$^{***}$ | 0.343$^{***}$ | 0.270$^{**}$ |
| | (0.0385) | (0.0593) | (0.108) |
| Fixed Effects | No | Yes | Yes |
| Time Effects | No | No | Yes |
| Within r2 | 0.103 | 0.126 | 0.211 |
| Between r2 | 0.00892 | 0.0520 | 0.0486 |
| Overall r2 | 0.00124 | 0.0217 | 0.0104 |
| Obs | 335 | 335 | 335 |

Standard errors in parentheses
$^{*}$ $p < 0.10$, $^{**}$ $p < 0.05$, $^{***}$ $p < 0.01$

## 4.3 Additional Info

1. For Macs, the default program to open .rtf (rich text format) files may not be Microsoft Word. You can set Word as your default .rtf program by navigating to "options" from the "file" command with Word open. Then select "general" and then hit the "default programs" button towards the bottom. Or for newer versions of Word, go to your computer "settings", then select "apps" and then hit the "default apps" button. Then,

scroll down and click "choose default apps by file type". Scroll all the way down to ".rtf" and change the default to Word.

2. There are many options, you can read more about those here: http://repec.org/bocode/e/estout/esttab.html

## 5 Data Visualizations

### 5.1 Histograms and Bar Charts

1. We should always want to know how our data are distributed. Now, in these practice data sets we provide the distributional information, but it can be good to visualize this nonetheless. To produce a Histogram or our data we can use the *histogram* command:

```
. histogram y
. hist y,  frequency normal
```

### 5.2 Time Plots

Here is the data file I use for the examples below:

stfiscal_7015.dta <1970-2015 state fiscal data>

In particular, we often want to express data for a specific or a set of specific entities. In this example, U.S. states. Perhaps we want to see the behavior of education spending per capita over time for some states. Let's take a look how to do this.

Step 1. Download the data set linked above to your working directory

Step 2: Let's drop everything but 1990-2015, and from Tennessee, North Carolina, New York, Michigan, and Texas.

```
. use stfiscal_7015, clear

. drop if year < 1990

. drop if states != 32 & states != 22 & states != 43 & states != 42 & states != 33
```

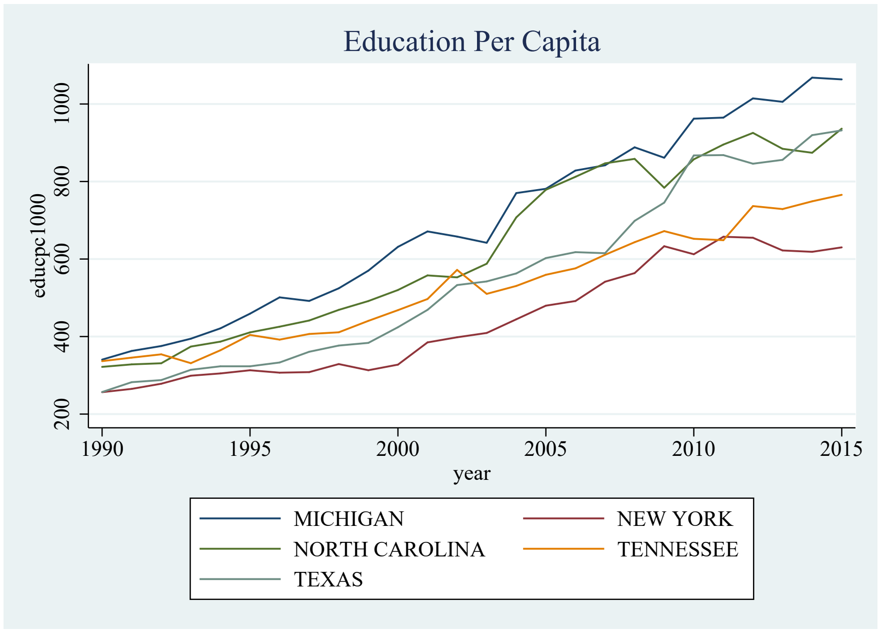
Step 4: Next, we tell Stata that we have panel data:

```
. xtset states year
```

Step 5: We create a graph with 5 lines, each representing the level of per capita education spending for the 5 states we've selected.

```
. gen educpc = educ/population

. xtline educpc, overlay title("Education Per Capita")
```

And that should look like this:



Here is the full code:

```
. clear all

. set more off

. cd "C:\your\desired\file\path"

. use stfiscal_7015, clear

. xtset name year

. drop if year < 1990

. drop if states != 32 & states != 22 & states != 43 & states != 42 & states != 33

. gen educpc = educ/population

. xtline educpc, overlay title("Education Per Capita")

. graph save educpc, replace

. graph export educpc.pdf, replace
```

## 5.3 Visualizing the Regression

**. graph twoway scatter y x,** ***options*** <scatters y against x>

**. graph twoway (lfit y x) (scatter y x),** ***options*** <fits an OLS line to the scatter>

**. graph twoway (lfitci y x) (scatter y x),** ***options*** <puts a 95% confidence interval on the line>

Common Options:
**title(<title of graph>)**
**xtitle("x")**
**ytitle("y")**

And we can save our graphs to our working directory like this:

**. graph save <your graph name>, replace**

then,

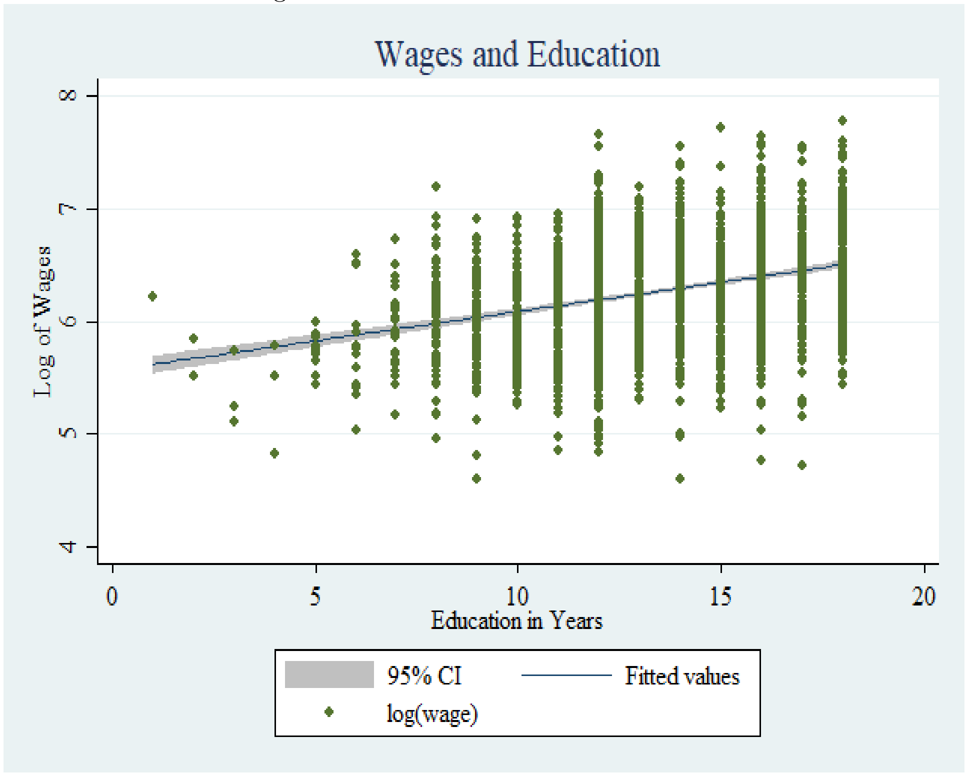
**. graph export <your graph name.png>, replace** <saves as .png file>

Here is the twoway graph from the regression in the previous section:

**. graph twoway (lfitci lwage educ) (scatter lwage educ), title(Wages and Education) /// xtitle("Education in Years") ytitle("Log of Wages")**

Which produces this graph:

Figure 11: Scatter Plot with Fitted Line



## 5.4 Geographically Detailed Heat Maps

We need to download some files first inside of Stata. You'll only need to do this once.

```
. ssc install spmap

. ssc install shp2dta

. ssc install mif2dta
```

In order to use the "spmap" command to create maps, we need to have certain map files that use coordinates. So, depending on the type of map you're looking for, country, state, county level, you'll need to gather those particular shape map files. This can be an annoying process. There are plenty of how-tos on the the internet, so I suggest you supplement this guide with a thorough search of available resources that cover your specific need. In particular, this guide is a great introduction into the process of creating the coordinate data sets.

Step 1. Create a folder and set it as your working directory:

```
. cd "C:\ file\ path\ of \ desired \ working \ directory \ folder"
```

Step 2: Download the following data set into this working directory:

1. xy_coor.dta <the coordinate file>

2. geo2xy_us_data.dta <the map data file>

3. stfiscal_15.dta <2015 state fiscal data>

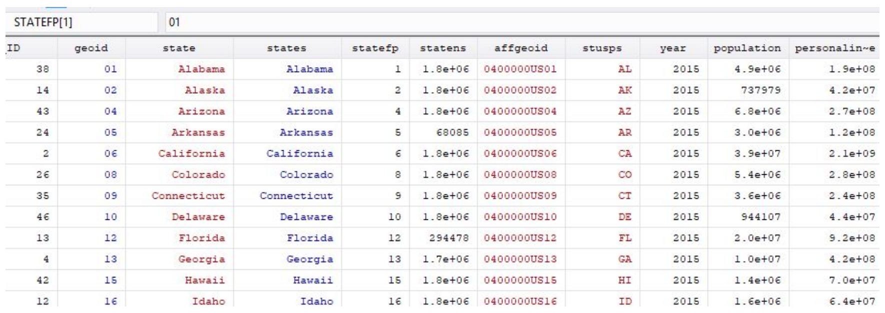
These data sets will allow you to create maps of the 50 U.S. states, and have Alaska and Hawaii situated to create a nice map.

Step 3: If you're not using Puerto Rico or Washington D.C., drop them from the map data file:

```
. use geo2xy_us_data, clear

. drop if GEOID == "72"

. drop if GEOID == "11"

. save geo2xy_us_data, replace
```

Step 4: Next, You'll need to merge the data sets of your desired data (the data you want to express in a map) and map data file from above (the second file). Before doing this however, you need to have an identifying column that is consistent across both data sets. Luckily, the map data file above contains several unique state identifiers that can usually be copy and pasted into your existing data set. Just be sure that when you paste the column into your data set that it properly identifies each state.

What I did was take the GEOID column in the map file and copy it to my data set such that it accurately aligns with the correct state. In fact, I took all the unique identifiers in the map file and copied them to my data.



STATEFP[1] 01

| ID | geoid | state | states | statefp | statens | affgeoid | stusps | year | population | personalin~e |
|---|---|---|---|---|---|---|---|---|---|---|
| 38 | 01 | Alabama | Alabama | 1 | 1.8e+06 | 0400000US01 | AL | 2015 | 4.9e+06 | 1.9e+08 |
| 14 | 02 | Alaska | Alaska | 2 | 1.8e+06 | 0400000US02 | AK | 2015 | 737979 | 4.2e+07 |
| 43 | 04 | Arizona | Arizona | 4 | 1.8e+06 | 0400000US04 | AZ | 2015 | 6.8e+06 | 2.7e+08 |
| 24 | 05 | Arkansas | Arkansas | 5 | 68085 | 0400000US05 | AR | 2015 | 3.0e+06 | 1.2e+08 |
| 2 | 06 | California | California | 6 | 1.8e+06 | 0400000US06 | CA | 2015 | 3.9e+07 | 2.1e+09 |
| 26 | 08 | Colorado | Colorado | 8 | 1.8e+06 | 0400000US08 | CO | 2015 | 5.4e+06 | 2.8e+08 |
| 35 | 09 | Connecticut | Connecticut | 9 | 1.8e+06 | 0400000US09 | CT | 2015 | 3.6e+06 | 2.4e+08 |
| 46 | 10 | Delaware | Delaware | 10 | 1.8e+06 | 0400000US10 | DE | 2015 | 944107 | 4.4e+07 |
| 13 | 12 | Florida | Florida | 12 | 294478 | 0400000US12 | FL | 2015 | 2.0e+07 | 9.2e+08 |
| 4 | 13 | Georgia | Georgia | 13 | 1.7e+06 | 0400000US13 | GA | 2015 | 1.0e+07 | 4.2e+08 |
| 42 | 15 | Hawaii | Hawaii | 15 | 1.8e+06 | 0400000US15 | HI | 2015 | 1.4e+06 | 7.0e+07 |
| 12 | 16 | Idaho | Idaho | 16 | 1.8e+06 | 0400000US16 | ID | 2015 | 1.6e+06 | 6.4e+07 |

Notice that it has the same unique identifiers for each state as the map data file has.

Now, we can merge the map data file and our data set:

```
. use geo2xy_us_data, clear

. merge 1:1 GEOID using stfiscal_15

. drop if _merge != 3

. drop _merge
```

Step 5: We can now create our preliminary map:

I want to create a map of debt per capita levels. I first create the debt per capita variable:

```
. gen debtpc = debttotal/population
```

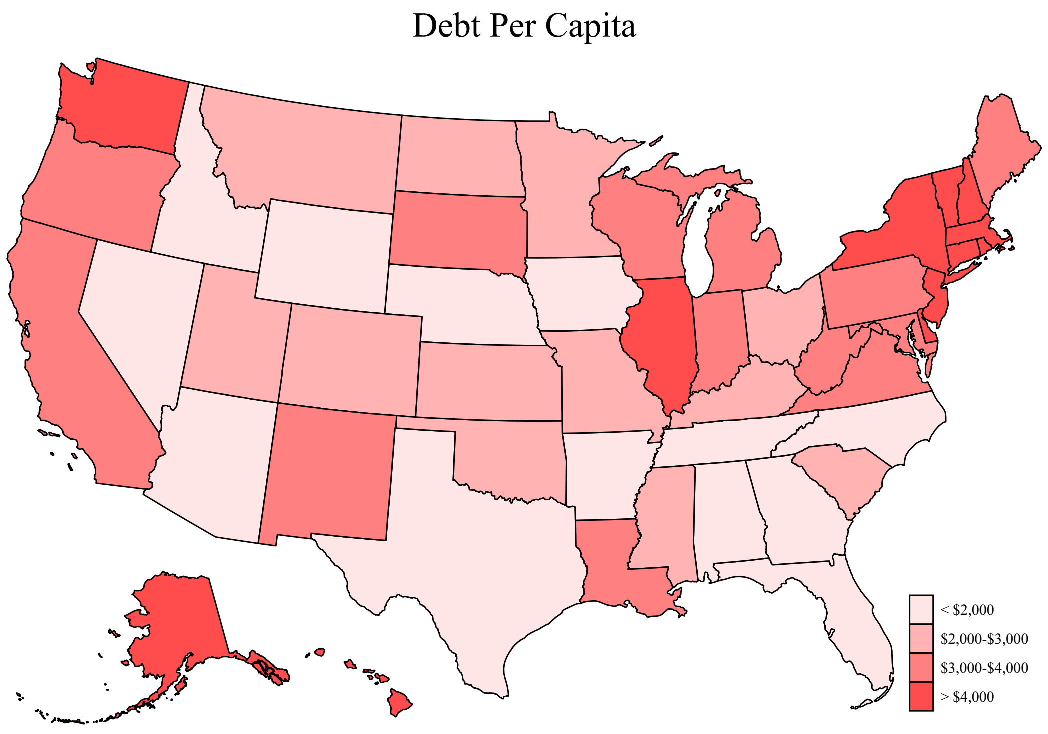
Now, I create the map:

```
. spmap debtpc using xy_coor.dta, title("Debt Per Capita") id(_ID)
```

You'll want to mess around with the graph editor once you create the map, but here's some code I use to make the map look nicer:

```
. spmap debtpc using xy_coor.dta, title("Debt Per Capita") id(_ID) legend(on position(4) symysize(4)

symxsize() label(2 "< $2,000") label(3 "$2,000-$3,000") label(4 "$3,000-$4,000")

label(5 "> $4,000") order(2 3 4 5)) fcolor(red*0.1 red*0.3 red*0.5 red*0.7)


. graph save debtpc, replace

. graph export debtpc.png, replace
```

Which looks like this!



NOTE: You will likely experience headaches trying to get this to work, but once you do it feels great!

Here is the full code:

```
. clear all
. set more off

. cd "C:\your\desired\file\path"

. clear all
. use "geo2xy_us_data.dta",clear
. drop if GEOID == "72"
. drop if GEOID == "11"
. merge 1:1 GEOID using bbrtypes
. drop if _merge != 3
. drop _merge

. gen debtpc = debttotal/population

. spmap debtpc using xy_coor.dta, title("Debt Per Capita") id(_ID)

. spmap debtpc using xy_coor.dta, title("Debt Per Capita") id(_ID) legend(on position(4) symysize(4)

symxsize() label(2 "< $2,000") label(3 "$2,000-$3,000") label(4 "$3,000-$4,000")
label(5 "> $4,000") order(2 3 4 5)) fcolor(red*0.1 red*0.3 red*0.5 red*0.7)

. graph save debtpc, replace
. graph export debtpc.pdf, replace
```