The Do's and Don'ts of a Do-file*

Emmanuel Milet†

Before starting: Is your Stata updated? Type **update query** in the command window and update it.
Do-files and datasets should be saved in the same folder. Graphs and logs can eventually be stored in a subfolder. The folder name should not include any blank space. It might not change anything, but programming softwares usually don't like blank space, so avoid it as much as possible.
To get started on the right foot:

1. Give explicit titles to your do files

2. Enumerate the do-files

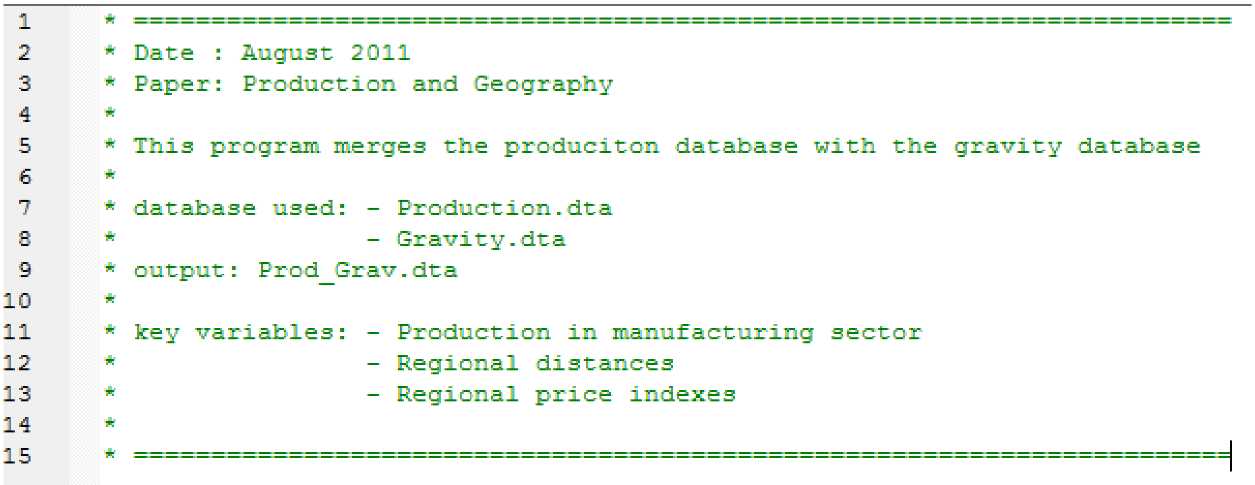
Example: 0-master.do, 1-merge.do, 2-stylized-facts.do, etc...
You can then organize them in sub-do-files: if you have different set of stylized facts, you could have: 2.1-stylized-facts-geography.do, 2.2-stylized-facts-count.do etc.... Now you can start writing your do-file. One thing you have to keep in mind: your do-file will certainly, at some point in your research, be read by someone else. Imagine having to understand a code with no comments, no indications whatsoever...a nightmare. Then, imagine you reading your own code a year after last modifying it: no comments, no indications.... You've just wasted two days reading and understanding your own code! To avoid that, try to apply, as much as possible, the following recommendations:

1 Comments

The first thing you should do when starting a new do-file, is write a small paragraph at the beginning of the do-file that explicitly describes what the program does. Mention all the database used (especially for the 1-merge.do), the stylized facts produced, the database output etc... All along your program, add as many comments as necessary: explain why you scale a variable, indicate the dataset you use in a merge before doing it (”we merge with dataset 1 to get these variables:...”).

*I thank Remi Bazillier, Mathieu Couttenier, Matthieu Crozet, Julien Vauday and Anna Zagrajczuk-Ray for helpful comments on previous versions of this document.

†Paris School of Economics (Université Paris 1)



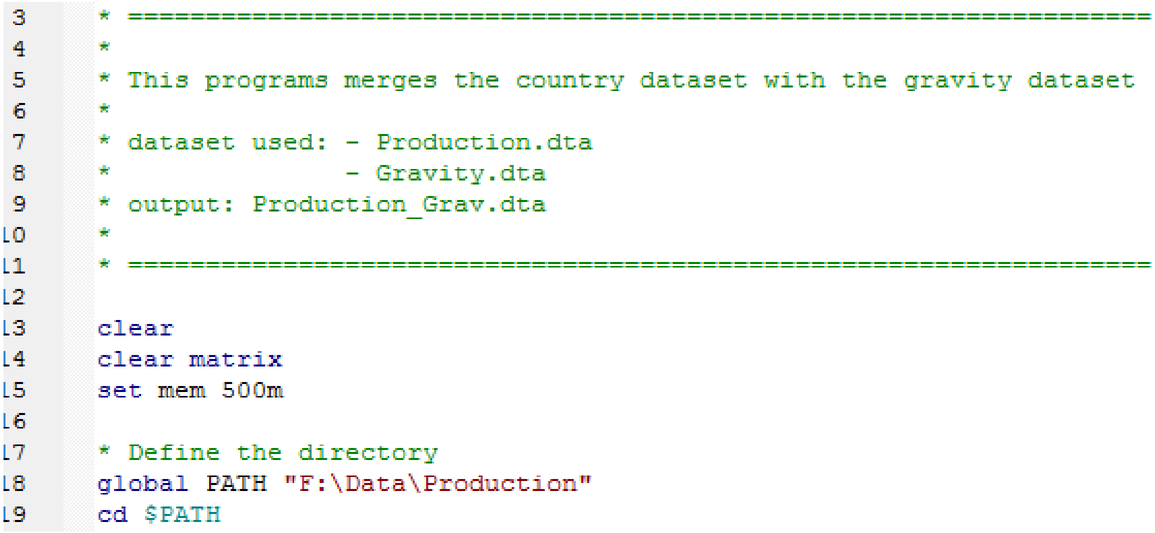
```
* =======================================================================
* Date : August 2011
* Paper: Production and Geography
*
* This program merges the produciton database with the gravity database
*
* database used: - Production.dta
*                - Gravity.dta
* output: Prod_Grav.dta
*
* key variables: - Production in manufacturing sector
*                - Regional distances
*                - Regional price indexes
*
* =======================================================================
```

2 Memory

You need to clear your Stata from all previous macros or matrix defined: **capture clear matrix**, **clear all** etc... The capture command allows the program to run even if it encounters an error. Be cautious with this command as you might want your program to actually stop if something goes wrong. This can however be useful if you are running many regressions that take time (say several hours per regression). You don't want your program to stop just because there is a tiny error in one command line (you mispelled a variable name for instance). You want it to get to the next regression, and you'll come back to the problematic one later. To do so, write **capture** at the beginning of you line

3 Directory and Global variables

Define the directory in which you have saved you dataset and do-files. A useful trick is to define a global variable and affect the directory path to your global variable. A global variable is a variable which is not directly created by Stata (you don't see it on the left side of the Stata window). To call a global variable, use the sign $ before the variable. For example:

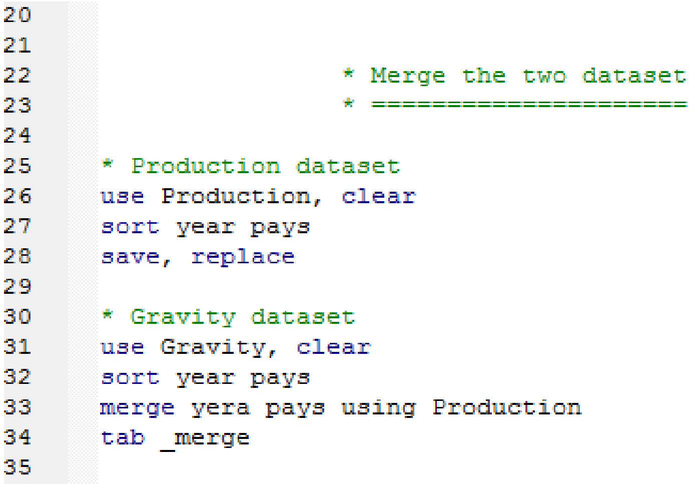


```
* ===================================================================
*
* This programs merges the country dataset with the gravity dataset
*
* dataset used: - Production.dta
*               - Gravity.dta
* output: Production_Grav.dta
*
* ===================================================================

clear
clear matrix
set mem 500m

* Define the directory
global PATH "F:\Data\Production"
cd $PATH
```

Now, to use any do-file or dataset present in this directory, you just need to type: **use dataset1, clear** instead of **use c:/research/data/dataset1, clear**. Same for saving a dataset.

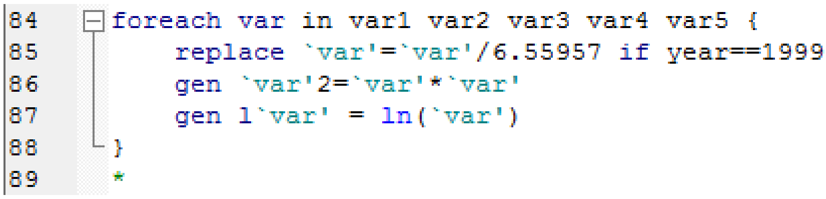
4 General Presentation

In the new Stata11, do-files are now available in color. People familiar with other programming languages still wonder why it took so long for Stata to implement this minor change, but here it is. Another important thing that makes your code tractable is the overall layout. Give explicit titles to important parts of you code, such as: Merging, General Statistics, Labels, Regression in Cross-section, etc... and maybe more important, write these titles in the following way:



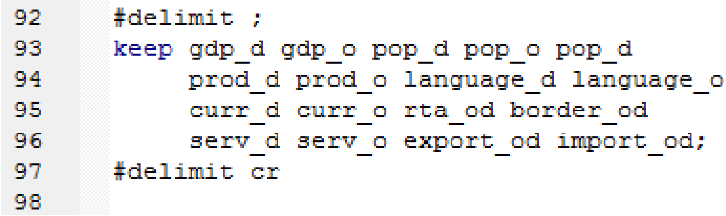
```


                * Merge the two dataset
                * =====================

* Production dataset
use Production, clear
sort year pays
save, replace

* Gravity dataset
use Gravity, clear
sort year pays
merge yera pays using Production
tab _merge

```

Another nice way of organizing your code deals with loops. Whether you use a "foreach" loop, or a usual loop, you should offset the instructions inside the loop.



```
foreach var in var1 var2 var3 var4 var5 {
    replace `var'=`var'/6.55957 if year==1999
    gen `var'2=`var'*`var'
    gen l`var' = ln(`var')
}
*
```

A useful command to make your code nicer and easier to read is the **#delimit** command. It allows you to write a command on several lines. It works like this: **#delimit ;** means that the following command will be finished when Stata encounters the sign ”;”. To go back to the usual way Stata works, type: **#delimit cr**. It is especially useful for graphs. Here is an example of how to use this command:
Example:
Note that in the example the code is offset to the right, to ease the reading.



```
#delimit ;
keep gdp_d gdp_o pop_d pop_o pop_d
     prod_d prod_o language_d language_o
     curr_d curr_o rta_od border_od
     serv_d serv_o export_od import_od;
#delimit cr
```

5 Variable Names and Labels

One of the first thing you're going to do while playing with your new dataset is create variables. You will probably create tons of variables, sometimes just for the fun of it, sometimes beccause you really need them. In any case, follow this simple rule: `make up your own rule and stick to it !`. What does it mean ? It means you should :

1. Give explicit names
2. Always follow the same pattern

By pattern I mean the way you write the name of your newly created variables. For instance, if you want to add numbers at the end of the name, say export99, export00, export01, then stick to the rule : ”‘name of the variable”’ & ”‘two digits”’, and try to apply that rule for further variables. Or, you might prefer to insert an underscore between the name and the numbers: ”‘name of the variable”’ & _ & ”‘two digits”’. That's fine, but please, don't mix both. You'll end up with variables like : ”‘export99”’ and ”‘export_99”’, and you will not remember why they are so different if their names are so similar.
Regarding the **labels**, well nothing new here: make labels ! My advice is that you make a do-file that contains only labels. It avoids your code from being jammed up by tons of label declarations, and makes any modification really straightforward: open your do-file, change the label, save it, and then run the do-file.

6 Regression Tables

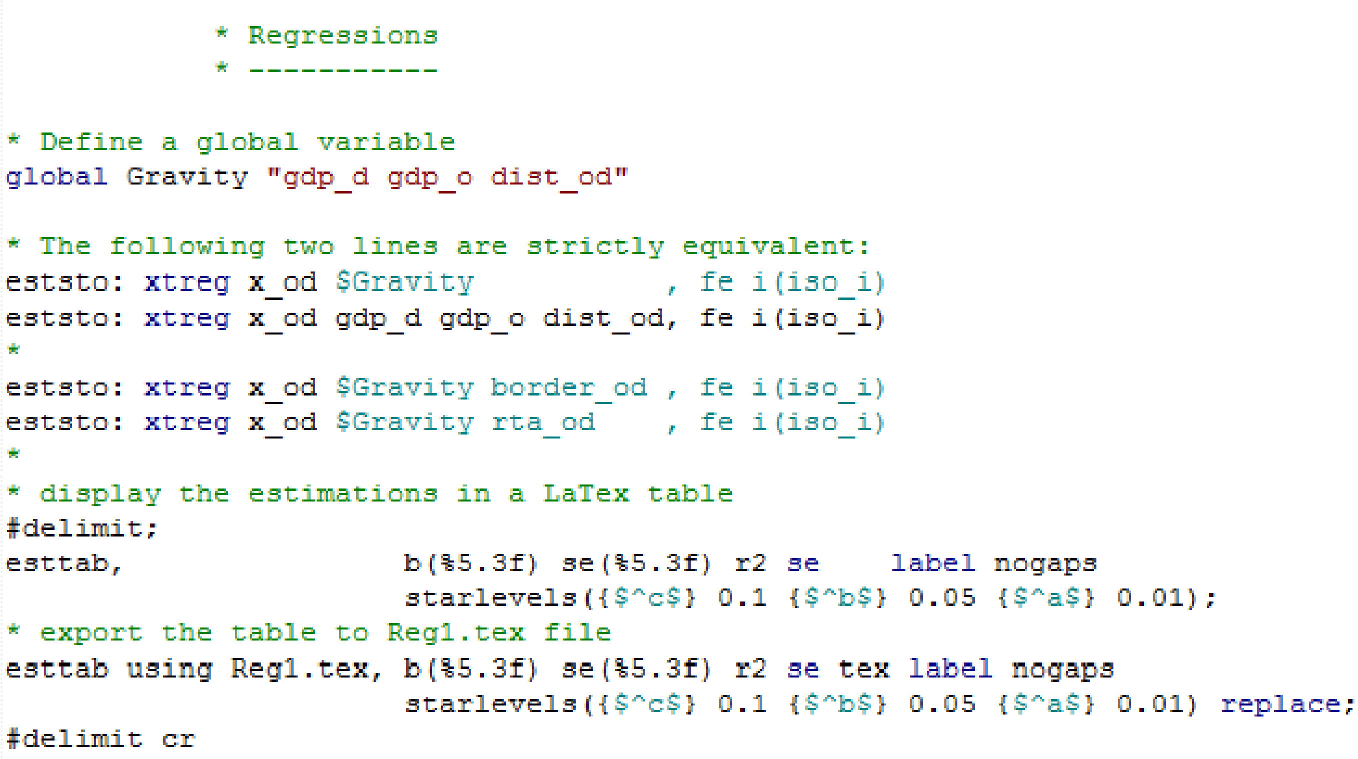
Regression tables are one of the most important output of your do-file, so don't mess with them: use labels, don't use stars to indicate the significancy levels (looks like you're rating restaurants for a tourist guide). Stars take a lot of place in the columns, and the less you put into those columns, the easier it is going to be to read them. Your footnote can (should) be quite large. Apart from the significant levels, include indications on the use of fixed effects or dummy variables, the number of years, countries, the econometric strategy employed in each column, details about some of the variables etc... These information are not crucial to the understanding of your table, so instead of writing them down in the core of your article, put them in the footnote: you will save time for people reading your article.

A quick way to get nice tables is to use the **eststo** and **esttab**. **Esttab** is in fact a simplified version of the **estout** command[1].

These commands display your regression tables the way you want them. You can specify the symbols for the significant levels, whether you use labels or not (Yes!), if you want it in the LaTex, cvs, html format etc... Many options are available, don't hesitate to have a look at the help file. **Esttab** produces estimation tables on the result window of Stata, with only the information you asked for, and can export your regression tables in the chosen format. I recommand you to do both: use **esttab** to display the regression table on your result window (this way it is saved in your log), then export it. You can export them in a different format: unless you're a LaTex fanatic, you don't want to see a table displayed in the LaTex format on your Stata result window...

Global variable can also be very useful in your econometric specification when a set of variable is recurrent. To avoid repeating the variables over and over again, define a global variable that will account for these variables. Then add the variable you are mainly interested in. Here is a rather complete code to get nice tables using a global variable:

[1] For more informations on the esttab command, see the paper by Benn Jann (`http://www.soz.unibe.ch/content/about_us/jann/publikationen/e19610/e20677/files20678/esttab.pdf`), and for more details on the estout command, see his other paper (`http://www.soz.unibe.ch/content/about_us/jann/publikationen/e19610/e20677/files20680/estout.pdf`)



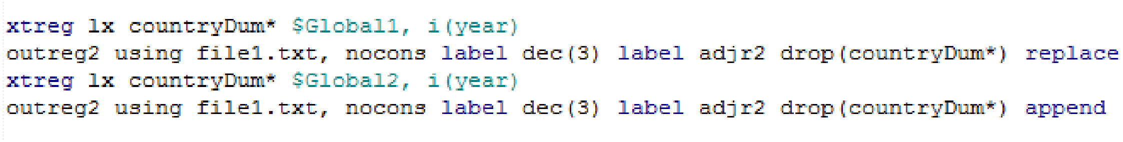
```
            * Regressions
            * -----------

* Define a global variable
global Gravity "gdp_d gdp_o dist_od"

* The following two lines are strictly equivalent:
eststo: xtreg x_od $Gravity           , fe i(iso_i)
eststo: xtreg x_od gdp_d gdp_o dist_od, fe i(iso_i)
*
eststo: xtreg x_od $Gravity border_od , fe i(iso_i)
eststo: xtreg x_od $Gravity rta_od    , fe i(iso_i)
*
* display the estimations in a LaTex table
#delimit;
esttab,                b(%5.3f) se(%5.3f) r2 se    label nogaps
                       starlevels({$^c$} 0.1 {$^b$} 0.05 {$^a$} 0.01);
* export the table to Reg1.tex file
esttab using Reg1.tex, b(%5.3f) se(%5.3f) r2 se tex label nogaps
                       starlevels({$^c$} 0.1 {$^b$} 0.05 {$^a$} 0.01) replace;
#delimit cr
```

One important thing: all the code that follows the **$** sign will be displayed in light blue even if it does not involve only global variable. This is a slight "bug" and should be corrected in the coming version of Stata.

Another popular way of computing regression table is to use the **outreg2** command. It does basically the same job, but has different features compare to **estout**, some good and some bad:

- + : It works with all regression commands and all versions of Stata. For obscure reasons, eststo/estadd does not work well on StataMP.
- − : You have to write the **outreg2** command after each regression command!
- − : If you want to export stored estimates, you have to use the **estout** command at the end of your **outreg2** command... (makes you wonder what's the point of having two procedures to do the exact same thing if they interact with each other...)
- − : outreg2 is only for exporting tables, it does not display them on the result screen, thus does not save them on your log

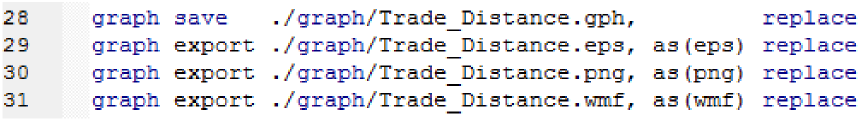
The outreg2 command is as follows (where file1.txt is text file containing the result of the previous regression) :



```
xtreg lx countryDum* $Global1, i(year)
outreg2 using file1.txt, nocons label dec(3) label adjr2 drop(countryDum*) replace
xtreg lx countryDum* $Global2, i(year)
outreg2 using file1.txt, nocons label dec(3) label adjr2 drop(countryDum*) append
```

7 Graphics

Like regression tables , figures are a very important output of your do-file. It is crucial that your graphs look nice, and attract the eye of the reader. You can save the same graph in different formats. Note the use of the command **graph export** instead of **graph save** when we save the graph in a different format.



```
graph save   ./graph/Trade_Distance.gph,          replace
graph export ./graph/Trade_Distance.eps, as(eps) replace
graph export ./graph/Trade_Distance.png, as(png) replace
graph export ./graph/Trade_Distance.wmf, as(wmf) replace
```

A few recommandations:
First, Stata uses a colorized background for its figures: light blue. It is ugly. You have a nice paper written on white pages, and your graph shows up with its bluish background color! Here is the option command to get rid of the background color and the frame color (that's right, Stata also applies a color for the frame of the graph region): **bgcolor(white) graphregion(color(white))**

Second, something I find useful is not to give titles to your graphs, you can include them directly when you write your paper. One thing Stata is not very good at is dealing with long titles Just like the footnote of the regression tables allowed you to include any not-so-interesting information, the footnote of the graphs can do the same: explain how to read the graph. This is particularly relevant if your graph is not a simple ”‘twoway plot”’ of two variables[2]. You will want to use the #delimit command introduced earlier in your graph commands[3].

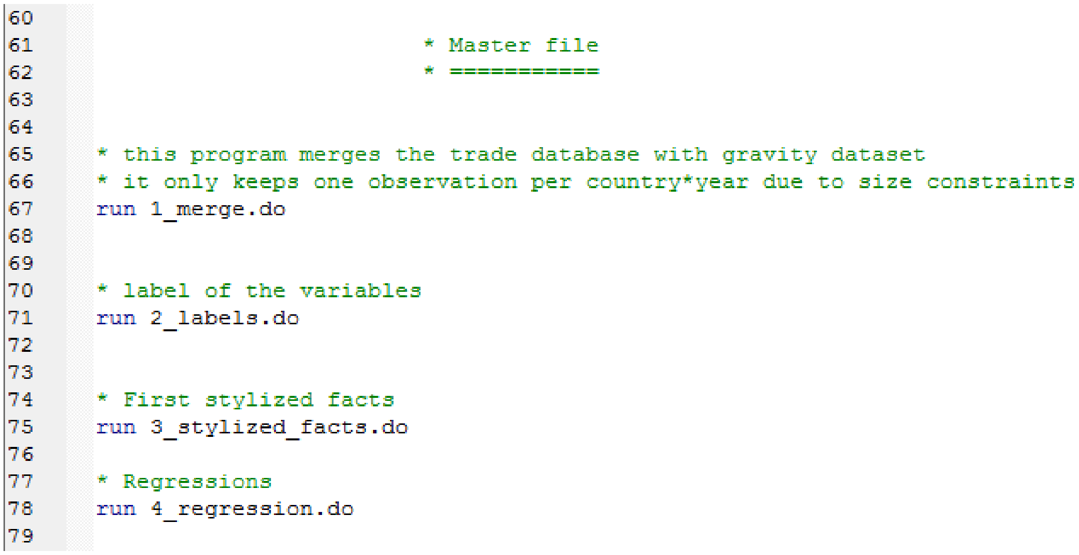
8 Master file

The purpose of the master do-file is to organize the sequence in which you want to execute your do-files. It is especially useful when you have many do-files. A usual master file looks like this:

One thing to notice here is the use of the command **run** instead of **do**. Both commands

[2] To see many example of interesting graphs, check the following website: `http://www.ats.ucla.edu/stat/stata/library/GraphExamples/default.htm` Click on a graph, and the stata code will appear.

[3] More on how to make graphs and organize regression tables can be found at John Ries' website (`http://strategy.sauder.ubc.ca/head/`). Check also his advices on how to write an introduction.



```

                                * Master file
                                * ===========


* this program merges the trade database with gravity dataset
* it only keeps one observation per country*year due to size constraints
run 1_merge.do


* label of the variables
run 2_labels.do


* First stylized facts
run 3_stylized_facts.do

* Regressions
run 4_regression.do

```

execute the do-file, but the **run** command does not display any result on the screen, while the **do** command does.

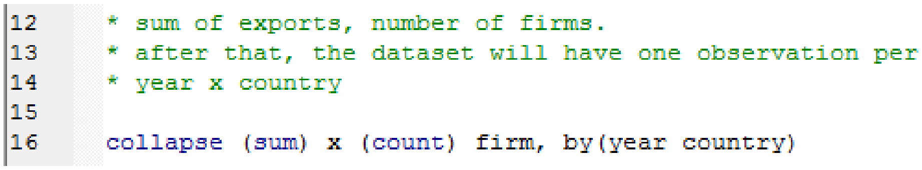
9 Miscellaneous tricks

This last section deals less with the do's and dont's of a do-file. Rather, it is a short description of useful command you should know and will make your life easier.

Set More Off Stata is a software that wants to make sure you are working when it is. When the command you execute produces a lot of output on the Result window, Stata will - at some point - stop and wait for you to press the Enter key or the Space bar before proceeding forward. To avoid this, just write **set more off** at the beginning of your code.

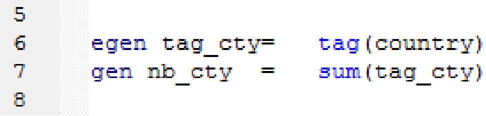
Quietly The command **quietly** simply tells Stata to not show the execution of the command on the Result window. This is particularly useful when you want to create a lot of variables from a loop, or from a **tabulate** procedure (to get dummy variables).

Collapse This command does exactly what it says: it collapses your database to the size you want. It reduces the number of dimensions and can calculate many statistics based on your original variables. It works like this:



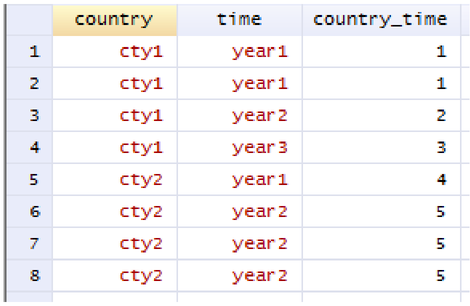
```
* sum of exports, number of firms.
* after that, the dataset will have one observation per
* year x country

collapse (sum) x (count) firm, by(year country)
```

Tag This command creates dummy variables that take the value 1 every time it encounters a different value of the specified variable, or of a couple of variables. This is particularly interesting when you deal with multiple dimensions datasets. For example, if you have a dataset with the following dimensions: Year × Country × Product. you want to count the number of countries you have. the tag command will create a variable that takes the value 1 each time it encouters a different country. It prevents you from collapsing your database.

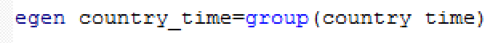


```

egen tag_cty=    tag(country)
gen nb_cty   =   sum(tag_cty)

```

Group The group **command** works with the same idea as the **tag** command. It creates a unique variable (as opposed to a serie of dummy variables) that takes a different value everytime it encounters an occurence of a variable or of a group of variables. In panel regressions, string variables cannot be used a fixed effects, one way to get around this is to create a group variable for instance. Here is the (very simple) code and an example of what your variable looks like:



	country	time	country_time
1	cty1	year1	1
2	cty1	year1	1
3	cty1	year2	2
4	cty1	year3	3
5	cty2	year1	4
6	cty2	year2	5
7	cty2	year2	5
8	cty2	year2	5



```
egen country_time=group(country time)
```