Stata How-to:
Get Started

Patrick Blanchenay

2021-12-27

Contents

1. Why use Stata?

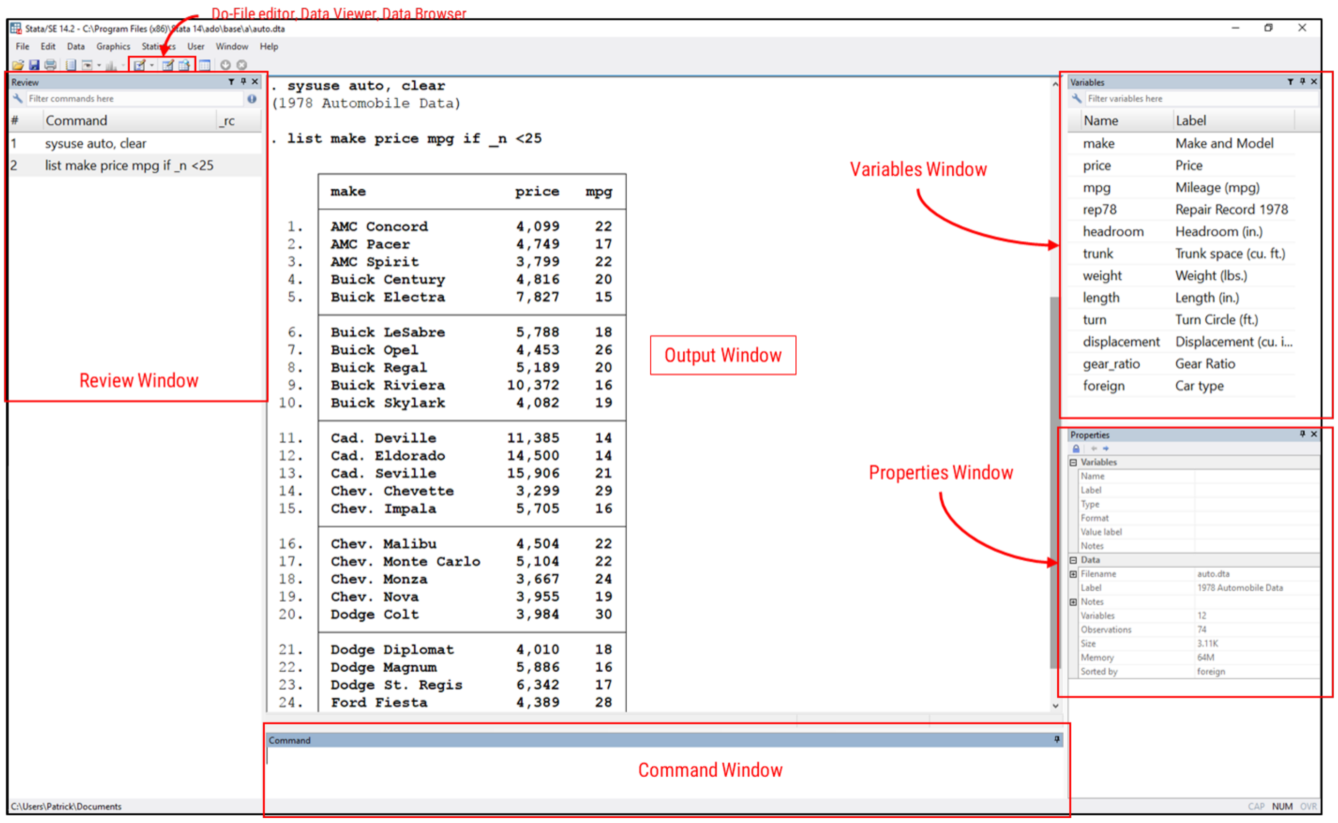
Stata is a powerful software that can perform data management tasks, conduct statistical analyses, and produce graphics for large datasets. The user can write scripts containing step-by-step instructions for preparing and analyzing a data file; this makes it easy to repeat the steps (replicability), and check/modify individuals steps (quality control). Stata's use of scripts, the wide range of available data analysis tools, and its capacity for working with large datasets makes it more suitable for statistical analysis non-specialized programs like Excel. It's also easier to get started in statistical programming using Stata than it is in the R or Python programming languages, although Stata sacrifices some flexibility in turn. Stata is used by many economists, both in academia and industry. Many of the skills you will learn working with Stata are transferable to other statistical programming environments and languages.

2. Getting Stata

Please refer to the syllabus for information on purchasing Stata. Just a reminder: you should NOT purchase the "Small Stata" edition; all other flavours (IC, SE and MP) are fine for this course, and you should feel free to purchase the cheapest option (6-months license is also fine for this course).

Stata is also accessible in the Map and Data Library, on Robarts Library 5th floor. See `https://mdl.library.utoronto.ca/technology/reference-area-computers` for more details.

3. Stata environment


Do-File editor, Data Viewer, Data Browser
Stata/SE 14.2 - C:\Program Files (x86)\Stata 14\ado\base\a\auto.dta
File Edit Data Graphics Statistics User Window Help
Review
Filter commands here
Review Window
Output Window
Variables
Filter variables here
Variables Window
Properties
Properties Window
Command
Command Window
C:\Users\Patrick\Documents
CAP NUM OVR

#	Command	_rc
1	sysuse auto, clear	
2	list make price mpg if _n <25	

```
. sysuse auto, clear
(1978 Automobile Data)

. list make price mpg if _n <25

     +-----------------------------+
     | make               price  mpg |
     |-----------------------------|
  1. | AMC Concord        4,099   22 |
  2. | AMC Pacer          4,749   17 |
  3. | AMC Spirit         3,799   22 |
  4. | Buick Century      4,816   20 |
  5. | Buick Electra      7,827   15 |
     |-----------------------------|
  6. | Buick LeSabre      5,788   18 |
  7. | Buick Opel         4,453   26 |
  8. | Buick Regal        5,189   20 |
  9. | Buick Riviera     10,372   16 |
 10. | Buick Skylark      4,082   19 |
     |-----------------------------|
 11. | Cad. Deville      11,385   14 |
 12. | Cad. Eldorado     14,500   14 |
 13. | Cad. Seville      15,906   21 |
 14. | Chev. Chevette     3,299   29 |
 15. | Chev. Impala       5,705   16 |
     |-----------------------------|
 16. | Chev. Malibu       4,504   22 |
 17. | Chev. Monte Carlo  5,104   22 |
 18. | Chev. Monza        3,667   24 |
 19. | Chev. Nova         3,955   19 |
 20. | Dodge Colt         3,984   30 |
     |-----------------------------|
 21. | Dodge Diplomat     4,010   18 |
 22. | Dodge Magnum       5,886   16 |
 23. | Dodge St. Regis    6,342   17 |
 24. | Ford Fiesta        4,389   28 |
```

Name	Label
make	Make and Model
price	Price
mpg	Mileage (mpg)
rep78	Repair Record 1978
headroom	Headroom (in.)
trunk	Trunk space (cu. ft.)
weight	Weight (lbs.)
length	Length (in.)
turn	Turn Circle (ft.)
displacement	Displacement (cu. i...
gear_ratio	Gear Ratio
foreign	Car type

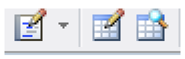
Variables	
Name	
Label	
Type	
Format	
Value label	
Notes	
Data	
Filename	auto.dta
Label	1978 Automobile Data
Notes	
Variables	12
Observations	74
Size	3.11K
Memory	64M
Sorted by	foreign

Stata is at the same time a statistical software, an interface to that software and an Integrated Development Environment (IDE). That means that you can do everything within Stata.

The main window is separated into different components:

Variables Window lists the variables from the current datasets and their labels;
Properties Window displays properties of the dataset in memory, or the variable selected;
Command Window is where you can type individuals commands; you should very rarely use it once you learn how to use scripts;
Review Window lists commands previously entered (in red if they generated an error);
Output Window is where the output of all commands is displayed.

Three important icons are also available:



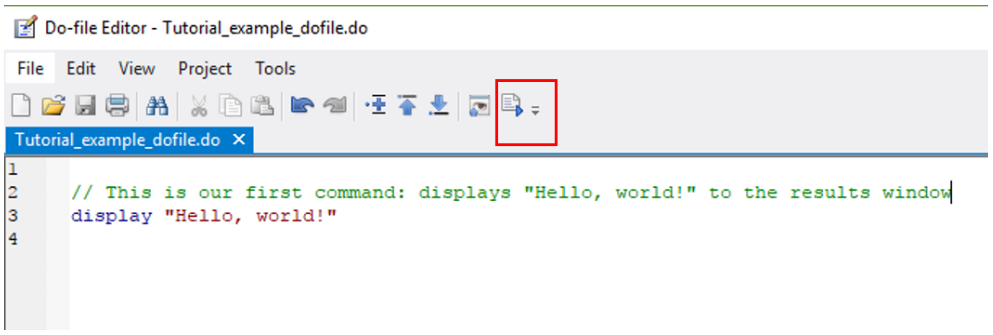
The left one is the do-file editor (see below). The other two are respectively the data editor and the data browser. You should never edit data manually so do not use the data editor; but it can be useful to browse the data using the data browser.

4. Commands, Do-files and the Do-file editor

Stata has a graphical interface that lets the user perform a number of tasks using icons and menus. However Stata's real power lies in its use of command lines, which gives a lot more flexbility and control over what the software does. Instead of clicking and pointing, the users types the command line, either in the Command Window (not recommended for serious analysis), or in programs, called do-files.

In general, Stata users refrain from typing command lines individually in the Command Window, but instead groups those commands in a script that is executed at once. Such scripts are called do-files and have a *.do file extension. Running a do-file means Stata will read the do-file and execute sequentially all the commands it contains.

Using do-files has two advantages. First: *reproducibility*. It is much easier to keep track of the steps that were done to perform a particular analysis, which means that it is easier to perform the same analysis again, even a few months or years later. Second: *quality control*. It is much easier to check each steps one by one, add comments (see below), check that the correct commands were used to perform what was intended.


Do-file Editor - Tutorial_example_dofile.do
File
Edit
View
Project
Tools
Tutorial_example_dofile.do

```

// This is our first command: displays "Hello, world!" to the results window
display "Hello, world!"

```

The do-file editor is a text editor embedded in Stata and specifically designed for Stata do-files. It highlights the Stata syntax to make it easier to read code. And it can execute do-file using the button circled highlighted above. (There is also a keyboard shortcut for this: Ctrl+D on Windows, Shift+Cmd+D on Mac.)

Warning

Executing a do-file does not save it. Make sure to regularly save your work. And as with anything important, you should always keep backups. Dropbox, Google Drive, OneDrive, etc. are all free services to ensure your files are always safely backed-up; in addition, most of them offer the ability to "rewind time" and revert back to previous versions of your do-files, should something catastrophic happen.

5. Stata syntax

Stata has its own language. In a do-file, Stata ignores multiple white spaces, and ignores blank lines. However, end of lines signals the end of a command, so each command should be on its own line.

```
command_name arguments, options
```

The command is usually the first word that starts a line, and is followed by arguments that can be required or optional. It can be followed by options after a comma; options affect the way the command behaves. Just as an example:

```
summarize mpg    , detail
```

asks Stata to summarize the variable mpg, and offer a detailed output (because of the **detail** option). Notice that I purposefully inserted unnecessary spaces before the comma: Stata will ignore those.

If you want to split long commands on multiple lines for legibility, you can actually use **///** (three slashes) and Stata will carry on reading on the following line(s). This is useful for commands that take a lot of options, such as graphing commands. The example below would ask Stata to run a regression with y as a dependent variable and variables x_1, x_2, x_3 and x_4 as regressors.

```
regress y x1 x2 ///
        x3 x4
```

Stata is case-sensitive: uppercase and lowercase are different things and this might generate errors if you are not careful. For instance, each of the following commands would generate an error:

```
Summarize mpg, detail
summarize MPG, detail
```

The first will fail because the command **Summarize** does not exist (only **summarize** does); the second command will fail because if the variable is called mpg, Stata will not find the variable called MPG.

5.1. Command syntax and documentation

All commands in Stata have their own documentation or help page. Typing help followed by the name of the command (e.g. **help display**) will bring the corresponding help page in Stata.

The top of any help screen provides the syntax of the command (in what orders its arguments and its options should go), followed by a detailed description of what the command actually does, and the effects of the various options of that particular command.

For instance, from the **regress** help page (type **help regress**):

```
regress depvar [indepvars] [if] [in] [weight] [, options]
```

The parts in square brackets [] are optional. So the syntax aid tells us that to run a regression we should type **regress** followed by the dependent variable, followed by the independent variables,[1] followed by an optional **if** condition,[2] followed by optional **in** condition (not used in this course), optionally followed by weights (for weighted survey data), optionally followed by options after a comma.

Good Practice

Stata knows how to recognize abbreviated commands, for instance **dis** instead of **display**. However, it is good practice to use full command instead of their abbreviation: it makes code easier to re-read, and minimizes chances of mistakes. Only keep this trick for very common commands.

[1] From the syntax, notice that independent variables are optional; if you do not include them, you are just regressing the dependent variable on a constant (not very useful).

[2] More on **if** conditions in Conditions, Subsetting Data for more details.

6. Code Errors and Finding Help

If your do-file generates an error, it will yield a red error message in the Output Window, and Stata stops executing the rest of commands. Check regularly your Output Window to find out if your code has executed properly. Error messages may or may not be sufficiently clear to explain what is wrong.

The first step is to find what line of your do-file generated the error, and to try to understand the error message. If a variable cannot be found, chances are its name is mispelled; check the command help page to ensure you used the correct syntax; check the dataset path is correctly specified, and so on. The devil is in the details: it is easy to get error because of small typos.

In addition to Stata's excellent documentation, there are numerous resources available online, including the Statalist mailing list; you will rarely be the first one facing a given problem. And as with many computer problems, googling is always a good idea.

7. Preliminaries: setting the project working directory

7.1. Working directory

Before loading a dataset in Stata or performing any command, we need to specify the working directory. The working directory acts as the default folder for Stata. Unless you explicitly specify full file paths, this is where Stata will 1/ look for datasets, and 2/ save what you tell it to save (an output, a graph, a log file, etc.). To keep things organized, I recommend that you create a new folder for each new project/assignment, and specify that folder as your working directory in Stata.

The folder can be located anywhere on your computer; however, since this is where all your analysis will take place, it is a good idea to set your folder in a place that is automatically backed up, e.g. as a folder in Dropbox, Google Drive, or OneDrive (free for all UofT students).

Once you have created the folder, you must tell Stata where it is. To specify a working directory, use the command **cd** followed by the path to the directory between quotes (" "):

```
cd "C:\Users\Patrick\MyStataProject"
```

All files relevant to your project should go into that folder. When you get more proficient you can refine by organizing files into subfolders (e.g. one for the original dataset, one for figures, one for regression outputs, etc.).

7.2. Organizing your files

Within your working directory, it is a good practice to organize your files into subfolders, for instance according to their nature. A suggested folder structure is:

```
[Working directory]
myprojectdofile.do
|- graphs
|- original_data
|- results
|- temp
```

The do-file for this project is at the root of the working directory. The subfolders suggested use are:

`graphs`	for any graphs created in your analysis;
`original_data`	for the original datasets that you will never-ever-ever modify or overwrite; this is the "input" for your analysis;
`results`	for any output that your code produces: regression tables, summary statistics tables, etc. ; this is the "output" for your analysis;
`temp`	for any kind of temporary file created by your code, such as modified datasets (anything in there should be automatically recreated by running your do-file, and therefore could be safely erased).

You should of course feel free to adapt the structure to your needs/habits.

Good Practice

Because modifying or over-writing your original data is a big *no-no*, you should store your original datasets into a dedicated subfolder and vouch never to save any output in there (even with different file names). You should also never modify this files manually; all data manipulation should be done with a do-file.

8. Good programming habits

I suggest the following good habits for any of your Stata project, in this course and beyond:

- Create a folder for each Stata project, preferably in a location backed up by Dropbox / Google Drive / OneDrive. Set this folder as your working directory.
- If datasets are provided to you (or you download them), save them in that directory.
- **Create a do-file for your project; any command to manipulate or analyze data should be in the do-file.**[3]
- **Frequently re-run your do-file from the beginning**, and debug any command that generates an error message. Do not let errors accumulate. (You will lose points on your assignments if your code generates any error.)
- **Never overwrite the original datasets**; if you modify data and must save it, use a different file name. That being said: if you include all data modification commands in your do-file (as you should), you can always reproduce these by re-running the do-file: unless your project is particularly complex, you should never need to save modified data.

8.1. Log file

A log file records all of the commands you run, and their results, to a text file which can be reviewed at a later date. [Note: you will have to do this for every assignment.] The contents of a Stata log file mirror the output in the results window; it will contain both the commands that you asked Stata to perform and their results (including error messages).

You can start and stop a log file with the following commands:

```
log using "mylogfile.log", replace text
log close
```

[3] For large projects, put all data manipulations command first (creating new variables, merging datasets, etc.), then all analysis commands.

The first command creates a log file and should therefore be at the beginning of your do-file, just after setting up your working directory.

The file `mylogfile.log` will be created in your working directory. The **`replace`** option tells Stata to overwrite the previous log file: the log file is overwritten every time you re-run your do-file.

The second command stops recording and saves the log file.

Warning

If there is a mistake in your code, Stata stops running the do-file at the point that generated the error. This means the **`log close`** command is not reached: you must manually close the log file by typing **`log close`** in the command window.
Alternatively, you can put the command **`capture log close`** at the beginning of your do-file to close any opened log file from a previous run.

8.2. Comment your code

To keep track of the code in your do file (and to make it legible by others) you should always include comments. Comments are blocks of text in your do-file that Stata ignores when running your do-file. You can therefore use these to type in "human" language notes to yourself or other people reading your code. Typing is quick, but trying to understand old code is not: comments will save you time.

Two types of comments exist: end-of-line comments and block comments. End-of-line comments can be entered using two slash symbols: **`//`** . Anything that follows **`//`** will be ignored. For longer comments, an entire block can be commented out if it is enclosed by **`/*`** and **`*/`** as in the following example:

```
/*      This is a block of comment
        that can run on several lines and
        will be ignored by Stata */

// End of line comment: Nothing to right of // is read by Stata
cd "C:/Temp" // Changing the working directory
```

8.3. Typical do-file structure

1. A block comment to explain what the do-file does.
2. Clear memory (**`clear`**), and allow display to run uninterrupted (**`set more off`**)
3. Set up working directory (**`cd`**).
4. Close any previously opened log file (**`log close`**), and start the log file from scratch (**`log using`**).
5. Your commands, with comments.
6. Close the log (**`log close`**).

```
/*      ============================
        Example of a typical do-file
        ============================ */
clear                                   // clear memory
set more off                            // allows uninterrupted display
cd "/path/to/my/folder/"                // working directory
capture log close                                   // close any previously opened log file
log using "mylogfile.txt", replace text // log all output

// Loading dataset
```

```
use "xyz.dta", clear

/* === Data cleaning and manipulation === */
generate lnincwage = ...   // creating a new variable

/* === Analysis === */
reg lnincwage age           // some regressions

// === Final commands
log close
// ==== End of do-file
```

9. Practice

1. ☐ Choose a folder at a convenient location on your computer, and find out its path.[4]
2. ☐ Open Stata, open the do-file editor, and save the empty do-file you just created, choosing a good file name, into that folder.
3. ☐ In your do-file, create the structure as suggested in section 8.3 (remove the **`generate`** and **`reg`** commands from the example). Do not forget to correctly specify as working directory the path to the folder you created in the first step.
4. ☐ Insert the following command to load the example dataset provided with Stata: **`sysuse auto, clear`**.
5. ☐ Run the do-file and check the output in Stata Output window. If it generates any error, solve for that error in your do-file and make sure it runs without error.
6. ☐ Use the Variables window and Properties window to find:
   a. ☐ how many variables are provided in this dataset; (12)
   b. ☐ how many observations there are. (74)
7. ☐ On a separate line before the end of your do-file, insert the command **`count`** to find the number of observations in your dataset. Save and re-run your do-file.
8. ☐ Add a comment "counting observations" to the command you just added. Save and re-run your do-file.
9. ☐ Open the log file, and check that it contains both the commands from the do-file (with their comments), and their output (e.g. the count of observations).
10. ☐ Open the help page of the **`correlate`** command: What does it do? What option displays covariances instead of correlations?

[4] You can usually find this at the top of the file explorer, or by right-clicking on the folder.