JEM174 Stata for Economists
Institute of Economic Studies
Charles University in Prague

Radek Janhuba
janhuba@gmail.com
Spring 2018

Contents

1 Introduction to Stata

1.1 What is Stata?

Stata is a complex statistical package widely used in various fields of research as well as a data processing software in companies. It is capable of performing various data management tasks as well as running complex statistical and econometric models, including but not limited to estimation itself, forecasting. It also includes a very powerful interface for creating graphics.

In these lecture notes, we will cover the basic functionalities of Stata. Notice, however, that it will be impossible to cover anything in big detail, as Stata is too complex for that (the current version's help files span over 14 thousand pages of pdf documentation).

Note that the correct way to spell Stata is "Stata". See the following point from the "Advice on posting to Statalist":[1]

> The correct spelling is "Stata", please, not "STATA". Several of the most active experts on the list can get a little irritated if you get that wrong, although you are free to regard them as pedantic. More importantly, if you write "STATA" you are making it obvious that you didn't read this guide carefully and to the end.

[1] https://www.statalist.org/forums/help#spelling

1.1.1 How Does Stata Look Like?

The default layout of Stata is shown on the figure below. On the top, you can find the command menu, one of the ways how to tell Stata what to do. A second way is the command line at the center bottom. The center of the screen is filled with the results window where you can see the output of commands Stata performed recently. On the left is the review window, where you can access history of commands. On the right side is a list of variables and their properties.

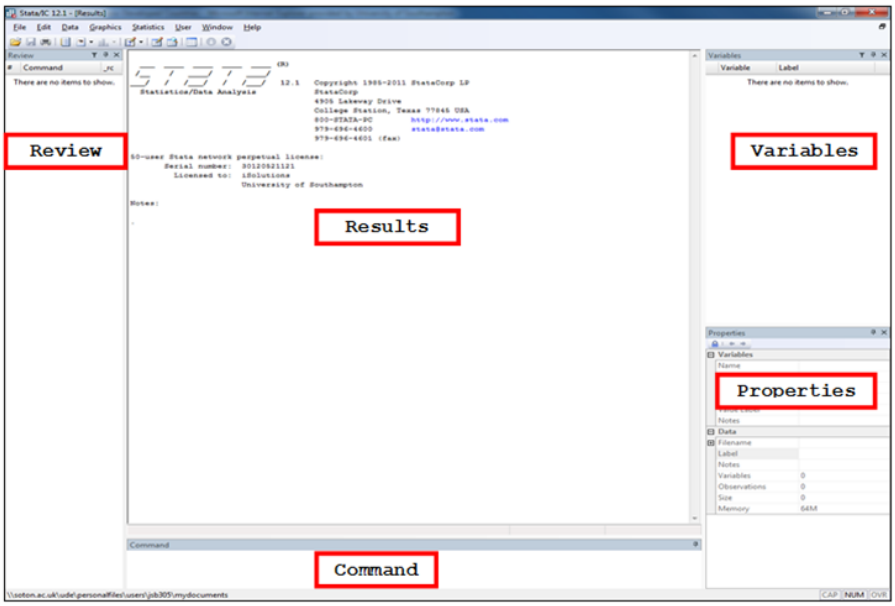

Figure 1: Basic Layout of Stata

1.1.2 Three Ways How We Can Use Stata

There are three main ways how you can tell Stata what to do:

1. GUI (graphical user interface)
2. Command line
3. Do files

The first way, "clicking interface", is the easiest for beginners, but it is slower than the others and it does not allow you to easily reproduce the steps. Therefore, although there will be some exceptions, we will mostly use the combination of second and third type in this class. You can find demonstrations of the first type of operating Stata e.g. at the Stata Youtube channel.[2]

The second way works through typing the commands into the "command" line located at the bottom of Stata underneath the results window. This way is recommended for experimenting with data and trying to figure out what is the best way of achieving the desired result.

The third way allows processing of batch blocks of code. It is recommended in order to achieve reproducible steps - once you figure out what the comment needs to be, you can copy it into your do file.

1.1.3 Selected Types of Stata Files

- `*.dta`: Data files

[2] `https://www.youtube.com/channel/UCVk4G4nEtBS4tLOyHqustDA`

- `*.do`: Do files (execute batch commands)
- `*.smcl`: Log files (record what happens)
- `*.gph`: Graph files (store graphs)

1.1.4 The First Command: Displaying Something on the Screen

Our first command is `display`. This can be used to display messages on the results screen. You can display e.g. text or use the command as a hand calculator:

```
.  display 10+10
20

. display 10^3
1000

. display "Hello World"
Hello World
```

Before we continue, note now that Stata is case sensitive, therefore variable `price` is not the same as variable `Price`. Similarly, you can try writing `Display 1` and you will see that Stata will return an error:

```
.  Display 1
command Display is unrecognized
r(199);
```

1.1.5 Use of Preexisting Datasets

Stata carries some preexisting datasets within the program, so users can try its functionalities without the need to obtain a dataset. These datasets are accessed using the command `sysuse`. Let's now load probably the most used out of these datasets, *auto*.

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

Note that in this case the option `clear` is not necessary as we have started with a clear Stata (no data were loaded in memory). However, if you do not have a specific reason to do so (namely a loss of data in memory), I suggest you use the option "by default" in order to avoid the following error:

```
.  sysuse auto
no; data in memory would be lost
r(4);
```

1.1.6 Browsing the Data

If we want to look at what data is loaded in Stata's memory, we can click the *Data Editor (Browse)*[3] button at the top or use the "browse" command:

[3]Note that there is also a *Data Editor (Edit)* button - however, unless you have a very good reason to do so, you should not be using that in order to avoid incidental changing of the data.

```
. browse
```

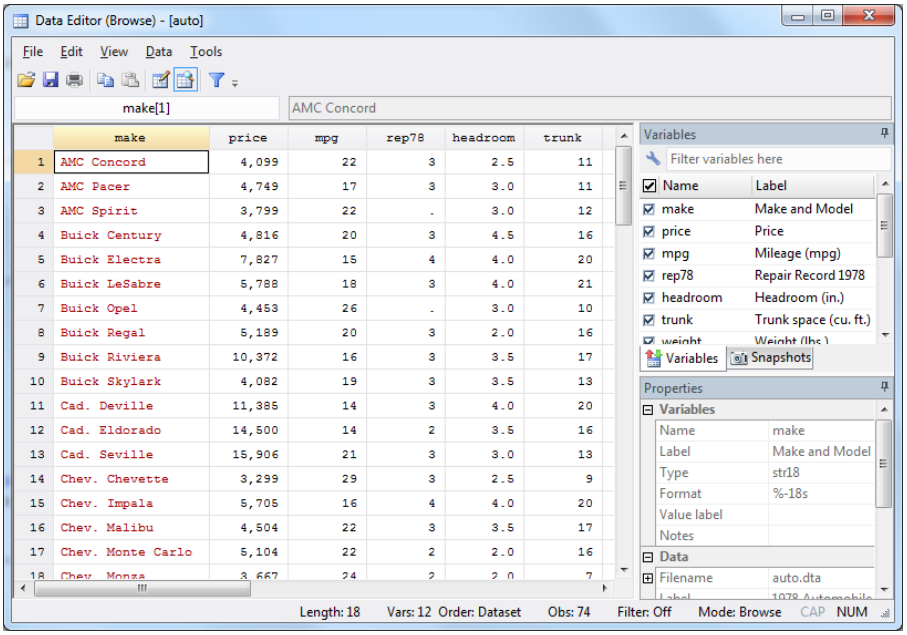

Figure 2: Data Browser

The window above will open the data browser where we can look at specific values, filter them based on variables, or look at what type each variable is (notice that value labeled and string variables are shown with a different color - we will cover these later in the course).

1.2 How to Get Help

The excellent help system is one of the most powerful features of Stata. If you need help with specific command, simply write `help` *command* and a help file will open:

```
. help summarize
```

Firstly, a brief description of the command is given, followed by the syntax (see the following) and description where to find the command in menu.

Usual command syntax goes something like the following: `command varlist [if] [in] [, options]` The general rule of thumb is that words **in bold** must be written exactly like they appear in help, words *in italics* must be replaced by variable name(s) and words in [] are optional. Command options must come at the end of the line and the first option has to be preceded by a comma. Note also that often some part of a word (e. g. `command`) will be underlined in help - This is the minimum you need to write in order for Stata to understand you. This can save you a lot of time, as Stata understands abbreviations quite often - you can, e.g., execute command `di` or `disp` instead of `display`, `su` instead of `summarize` or `reg` instead of `regress`.

varlist is a list of one or more variable names. Sometimes it is optional, sometimes it is mandatory. You can also use so called wildcards to specify certain variables matching a pattern - e.g. "*day" would match all variables ending with "day" (monday, tuesday, etc.). See `help varlist` for more information.

[, options]: Most of the commands have one or more options that allow you to get differing outcomes from executing them. We will see this a lot throughout the course.

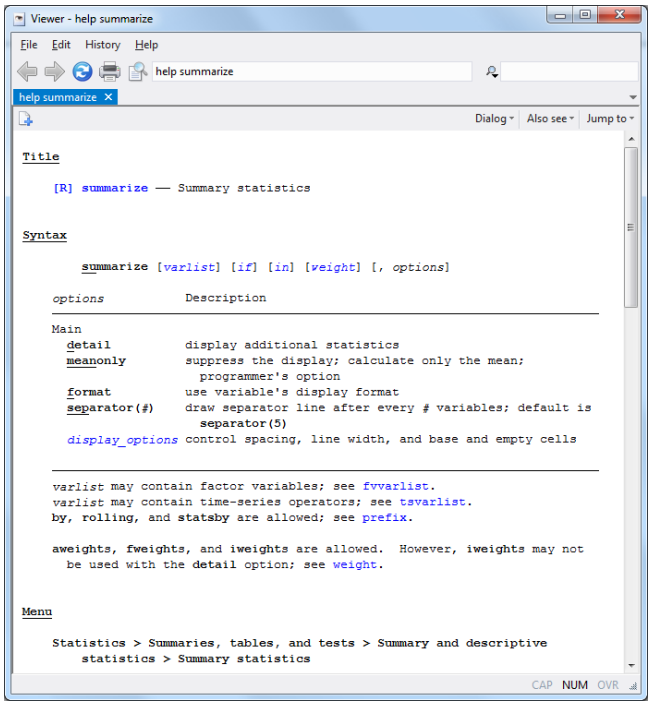

Figure 3: Help window layout

[if]: Many times you will want to do some command for a subset of the data, this is where if comes in - it allows you to run the command specifically for a subset specifying some logical condition that you will select. Similarly, [in] will select the subset of data based on the current observation number. In order to show the functionality of [if] and [in], take the following command:

```
. sum price if mpg>20 in 1/5

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |          2        3949     212.132        3799       4099
```

The command tells Stata to summarize price of cars that run more than 20 miles per gallon, moreover, only in the first five observations of the dataset. Notice the difference from the following commands:

```
. sum price if mpg>20

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         36    5350.306    2358.612       3299      15906

. sum price in 1/5

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |          5        5058    1606.718       3799       7827

. sum price
```

```
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906
```

The list of Stata's possible logical operators follows:

Arithmetical:

```
+ Addition
- Subtraction
* Multiplication
/ Division
^ Power
```

Relational

```
>  greater than
<  less than
>= greater or equal
<= less or equal
== equal
!= not equal
```

Logical

```
! not
| or
& and
```

Finally, in case the information presented in the window is not enough, you can try to click on the blue heading and a pdf file with more detailed explanations will open. And if even that is not enough, Google should help you in almost every case.

1.3 Using do Files and Commenting Your Work

1.3.1 Opening an Empty do File Editor

To start a new do editor, you can click on the particular button on the toolbar or you can simply type `doedit` into the command line - in both cases a new do file editor will open. You can then start writing your commands.

To open an existing do file, use the command `doedit` *`filename`*, where *filename* is the name of the file including a directory reference. Note that it is strongly recommended not to use absolute file paths. Instead of using

```
c:\Users\rjanhuba\Dropbox\CERGE\Teaching\_IES\Part1\P01.do
```

it is much easier to work with relative file paths, such as `Part1/P01.do` after setting a working directory

```
c:\Users\rjanhuba\Dropbox\CERGE\Teaching\_IES\
```

(see below on setting a working directory).[4]

[4]Note that you can use backward as well as forward slashes, even though forward ones are recommended due to compatibility of operating systems.

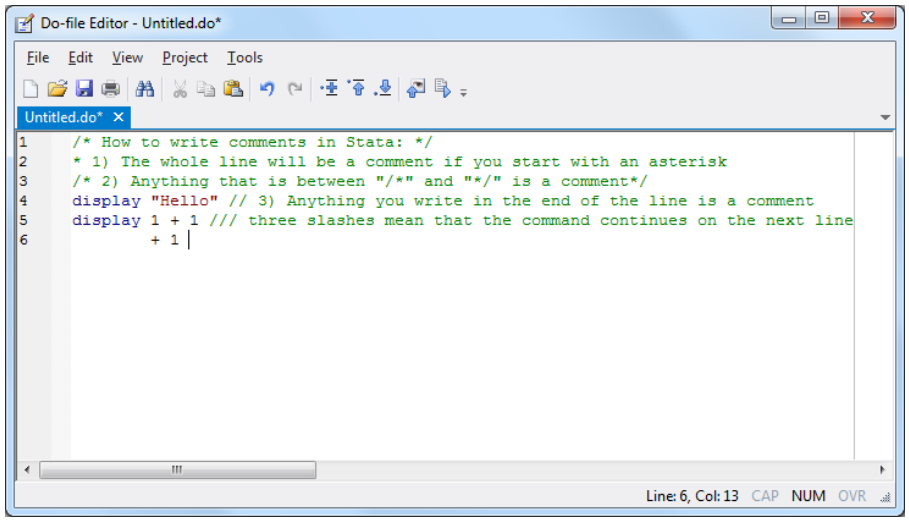

Figure 4: Do file editor

1.3.2 Running Commands from the do File

To execute part of the code, select it and click the rightmost icon on the toolbar. Alternatively, you can use the shortcut CTRL+D.

To run the whole do file, open it and click the same icon without selecting anything. Alternatively, you can use the command `do` *`filename`*.

1.3.3 How to Write Comments in Stata

Figure 4 shows you the several ways how you can comment your do files. Each of these has some advantages and disadvantages.

The figure also shows that comments are by default shown with a green color. Commands are shown by dark blue and text embedded in quotation marks is highlighted by "brownish" color.

Remember: Writing good comments saves you a lot of time when you come back to your work after some time. It also helps other people tremendously when they try to get oriented in your code.

1.4 Some Useful Functionalities

The list of functionalities contained here is not absolutely necessary to be able to work with Stata, but it can save you a lot of time and typing (and even errors) while performing repetitive tasks.

1.4.1 Setting a Working Directory

Stata needs a working directory in order to know where to save files by default etc. You can check the current directory by a command `cd`:

```
. cd
C:\Dropbox\CERGE\Teaching_IES_SFE\LN
```

When you launch Stata using a do file or a data set, it will set a working directory into a directory where the file is located. If you launch it by a desktop icon, it will set it to a default location depending on its settings. You can change this directory by typing `cd` *`directorypath`*

```
. cd "C:\Dropbox\CERGE\Teaching_IES_SFE\LN\"
C:\Dropbox\CERGE\Teaching_IES_SFE\LN
```

Some remarks:

- You only need to use quotation marks if the directory has a space in itself
- Unless you have a good reason to do so, it is advisable to use relative directory addresses as well (although it may be necessary to use the absolute ones from time to time in case of working directories - e.g. when using a network version of Stata).

1.4.2 Logging Your Work

When you run Stata commands, the results appear in the results window. However, sometimes (most of the time) you need a way to report the results and perhaps save them for later analysis. This is where logs come in. In order to start a log, you can write:

```
.  capture log close

. log using MyFirstLog, replace
-------------------------------------------------------------------------------
      name:  <unnamed>
       log:  C:\Dropbox\CERGE\Teaching_IES_SFE\LN\MyFirstLog.smcl
  log type:  smcl
 opened on:  10 Feb 2018, 19:27:50
```

Actually, the first line is not needed, however, it assures that a log will be open at this time. `log close` will close any previously running log and `capture` will assure that Stata will ignore an error and continue working when it discovers that no log file is open (hence it cannot close it).

Note: use capture **WITH CAUTION!** If you use it too much, it can happen that you would get erroneous results and/or obtain errors which disguise true nature of the error (just try to write the following two lines into the command line):

```
.  capture a = 1

. display a
a not found
r(111);
```

Even though the true error is a bad syntax in `a = 1`, Stata says "a not found".

Log files track commands which you sent into Stata and their results. They are a special class of files (`*.smcl`). You can either see them in Stata viewer or you can translate them into other formats.[5]

```
.  log close
      name:  <unnamed>
       log:  C:\Dropbox\CERGE\Teaching_IES_SFE\LN\MyFirstLog.smcl
  log type:  smcl
 closed on:  10 Feb 2018, 19:27:50
-------------------------------------------------------------------------------

. translate MyFirstLog.smcl MyFirstLog.txt, replace
(file MyFirstLog.txt written in .txt format)
```

[5]Note that even though you do not need to, it is recommended to close the log before translating it unless you are trying to get some "progress" information.

```
. translate MyFirstLog.smcl MyFirstLog.pdf, replace
(file MyFirstLog.pdf written in PDF format)
```

1.4.3 Suppressing the Output on Multiple Pages

If you open fresh Stata, load the `auto` example dataset, and type

```
.  list make

     +-------------------+
     | make              |
     |-------------------|
  1. | AMC Concord       |
  2. | AMC Pacer         |
  3. | AMC Spirit        |
  4. | Buick Century     |
  5. | Buick Electra     |
     |-------------------|
  6. | Buick LeSabre     |
  7. | Buick Opel        |
  8. | Buick Regal       |
  9. | Buick Riviera     |
 10. | Buick Skylark     |
     |-------------------|
 11. | Cad. Deville      |
 12. | Cad. Eldorado     |
 13. | Cad. Seville      |
 14. | Chev. Chevette    |
 15. | Chev. Impala      |
     |-------------------|
 16. | Chev. Malibu      |
 17. | Chev. Monte Carlo |
 18. | Chev. Monza       |
 19. | Chev. Nova        |
 20. | Dodge Colt        |
     |-------------------|
 21. | Dodge Diplomat    |
 22. | Dodge Magnum      |
 23. | Dodge St. Regis   |
 24. | Ford Fiesta       |
 25. | Ford Mustang      |
     |-------------------|
 26. | Linc. Continental |
 27. | Linc. Mark V      |
 28. | Linc. Versailles  |
 29. | Merc. Bobcat      |
 30. | Merc. Cougar      |
     |-------------------|
 31. | Merc. Marquis     |
 32. | Merc. Monarch     |
```

```
33. | Merc. XR-7        |
34. | Merc. Zephyr      |
35. | Olds 98           |
    |-------------------|
36. | Olds Cutl Supr    |
37. | Olds Cutlass      |
38. | Olds Delta 88     |
39. | Olds Omega        |
40. | Olds Starfire     |
    |-------------------|
41. | Olds Toronado     |
42. | Plym. Arrow       |
43. | Plym. Champ       |
44. | Plym. Horizon     |
45. | Plym. Sapporo     |
    |-------------------|
46. | Plym. Volare      |
47. | Pont. Catalina    |
48. | Pont. Firebird    |
49. | Pont. Grand Prix  |
50. | Pont. Le Mans     |
    |-------------------|
51. | Pont. Phoenix     |
52. | Pont. Sunbird     |
53. | Audi 5000         |
54. | Audi Fox          |
55. | BMW 320i          |
    |-------------------|
56. | Datsun 200        |
57. | Datsun 210        |
58. | Datsun 510        |
59. | Datsun 810        |
60. | Fiat Strada       |
    |-------------------|
61. | Honda Accord      |
62. | Honda Civic       |
63. | Mazda GLC         |
64. | Peugeot 604       |
65. | Renault Le Car    |
    |-------------------|
66. | Subaru            |
67. | Toyota Celica     |
68. | Toyota Corolla    |
69. | Toyota Corona     |
70. | VW Dasher         |
    |-------------------|
71. | VW Diesel         |
72. | VW Rabbit         |
73. | VW Scirocco       |
74. | Volvo 260         |
    +-------------------+
```

You can see that the listed output has "three pages" and you have to click on "more" or press a key in order for the display to continue to the next page. By running the next line, you can turn this functionality off:

```
. set more off

. list make // output omitted now - notice that it ran without a stop.
```

Now everything will appear together. In order for your do files to run smoothly, I recommend that you write `set more off` into the beginning of every do file that you write.

1.4.4 Resetting Stata

In case you need to "reset" Stata (e.g. in between two independent parts of a project), you may use `clear` or `clear all` commands. The difference between the two commands is that while `clear` drops all variables.[6], `clear all` drops not only the data, but also all other information that is stored in Stata (it actually refreshes it back to the state it was when you opened it.)

```
. clear all
```

1.4.5 Making Sure Your do Files Will Work in Future

Stata is a dynamic program that adds new functionalities and updates commands in each version. In order for it to produce exactly the same output if you run your code several years from now, you can use the "version" command.

```
. version // Without the number, this shows you which version Stata is using
version 14.2

. version 13.1 // With the number, you tell Stata which version it should use
```

1.4.6 Installing Additional Packages

Sometimes Stata is not able to perform all the tasks you want. Luckily, it has a great functionality that you can install personal packages and then use them in the same way as built-in commands. Suppose now that we want to install a package of additional commands associated with Baum (2006). We know from the book that the package is called *itmeus*. If we type

```
. help itmeus // Stata does not know the package
```

we can see that Stata does not know it (unless you already have it installed) To install it, we can write

```
. ssc install itmeus
checking itmeus consistency and verifying not already installed...
installing into c:\ado\plus\...
installation complete.
```

[6] Actually, variables plus labels.

Now, after installing it, we can see that the help file works:

```
. help itmeus
```

1.4.7 Checking Something is Correct

You can use the `assert` command to control your code. This command checks an expression you feed to it and returns an error in case it is not true.

```
. assert 10+1==11 // The expression is true, there is no error

. assert 10+1==12 // Now Stata returns an error
assertion is false
r(9);
```

If you find the above example rather stupid, do not worry about it - we will come back to `assert` when there is an example that makes sense.

1.4.8 Changing the Delimiter

By default, Stata treats an end of each line as an end of a command. Therefore, the following two lines will produce an error:

```
. display 1 + 1 +
invalid syntax
r(198);

.                 1
1 is not a valid command name
r(199);
```

There are two ways how we can avoid it. In the first case, as we have already seen in the comments section, we can put three forward slashes at the end of the line. In such case, Stata will take the next line as a continuation of the preceding command:

```
. display 1 + 1 + ///
                  1

3
```

Alternatively, in cases we have a very long command spanning over multiple lines, we can set the delimiter to a semi-colon (;). This way, Stata will treat everything as one command up to the point where it finds this delimiter:

```
. #delimit ;
delimiter now ;
. display 1 + 1 +
             1;

3
```

```
. #delimit cr
delimiter now cr
```

Note that the last line clears the delimiter again - Stata will resume treating the end of the line as the end of a command after that.

1.4.9 Running Commands Quietly

If you want to suppress output of a command, you can use the command `quietly` in front of it. This will cause Stata to run the command, but it will not report the results in the results window (such a behavior is often useful if you need to calculate something with a very extensive text output).

```
.  display "Hello" // displays
Hello

. quietly display "Hello" // does not display
```

Again, do not worry about this command now, we will show a demonstrative example when we will get to it.

References

Baum, C. (2006). *An Introduction to Modern Econometrics Using Stata*. Stata Press publication. Taylor & Francis.