Tutorial 1
Introduction to STATA*

Empirical Banking and Finance
Konrad Adler

Summer, 2020

We start with a small example in order to provide a first look at the Stata environment and the way it works. In order to follow the example, you first need to download the dataset caschool.dta from eCampus. The ending .dta shows that we are dealing with a Stata dataset.

Data description The California Standardized Testing and Reporting (STAR) dataset contains data on test performance, school characteristics and student demographic backgrounds. The data used here are from all 420 K-6 and K-8 districts in California with data available for 1998 and 1999.

Test scores are the average of the reading and math scores on the Stanford 9 standardized test administered to 5th grade students. School characteristics (averaged across the district) include enrollment, number of teachers (measured as fulltime- equivalents"), number of computers per classroom, and expenditures per student. The student-teacher ratio used here is the number of full-time equivalent teachers in the district, divided by the number of students. Demographic variables for the students also are averaged across the district. The demographic variables include the percentage of students in the public assistance program CalWorks (formerly AFDC), the percentage of students that qualify for a reduced price lunch, and the percentage of students that are English Learners (that is, students for whom English is a second language). All of these data were obtained from the California Department of Education (www.cde.ca.gov).

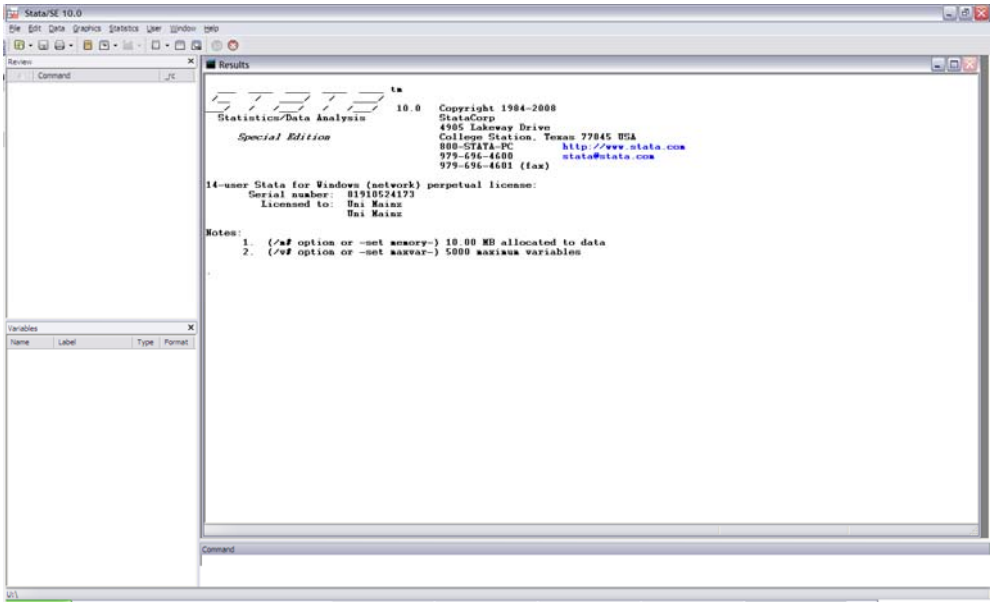
*Thanks to Ulrich Schüwer for providing the Tutorial

Series in Data Set:

DIST_CODE: DISTRICT CODE;
READ_SCR: AVG READING SCORE;
MATH_SCR: AVG MATH SCORE;
COUNTY : COUNTY;
DISTRICT: DISTRICT;
GR_SPAN: GRADE SPAN OF DISTRICT;
ENRL_TOT : TOTAL ENROLLMENT;
TEACHERS: NUMBER OF TEACHERS;
COMPUTER: NUMBER OF COMPUTERS;
TESTSCR: AVG TEST SCORE (= (READ SCR+MATH SCR)/2);
COMP_STU: COMPUTERS PER STUDENT (= COMPUTER/ENRL TOT);
EXPN_STU: EXPENTITURES PER STUDENT ($'S);
STR: STUDENT TEACHER RATIO (ENRL TOT/TEACHERS);
EL_PCT: PERCENT OF ENGLISH LEARNERS;
MEAL_PCT: PERCENT QUALIFYING FOR REDUCED-PRICE LUNCH;
CALW_PCT: PERCENT QUALIFYING FOR CALWORKS;
AVGINC: DISTRICT AVERAGE INCOME (IN $1000'S);

Getting started

After opening Stata, we obtain the following window:



The Stata screen has four major parts:

1. **Command Window:** window to write the commands that are to be executed (commands can be quickly repeated with the buttons “page up” or “page down”)
2. **Results Window:** shows the results of the executed commands
3. **Review Window:** lists the commands used in the current session
4. **Variables Window:** lists the variables in the currently open data file

In order to open caschool.dta, use the menu File→Open… or type the following command in the command window and press Enter (assuming that you saved the file “caschool.dta” in the folder “U.\empBF”):

use U:\empBF\caschool.dta

Alternatively, you can change your working directory using

cd U:\empBF

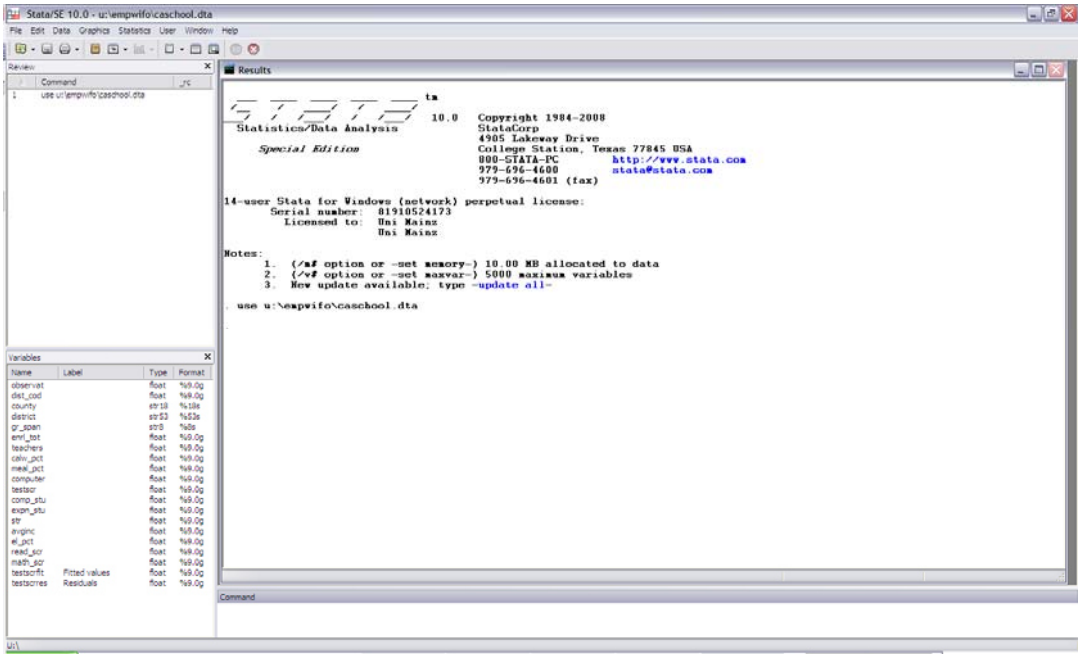
and then just type

use caschool.dta.

Generally, if the path to the file contains empty spaces, one should use quotation marks:

use "U:\empBF\caschool.dta"

After executing the command, your Stata environment looks approximately like this:



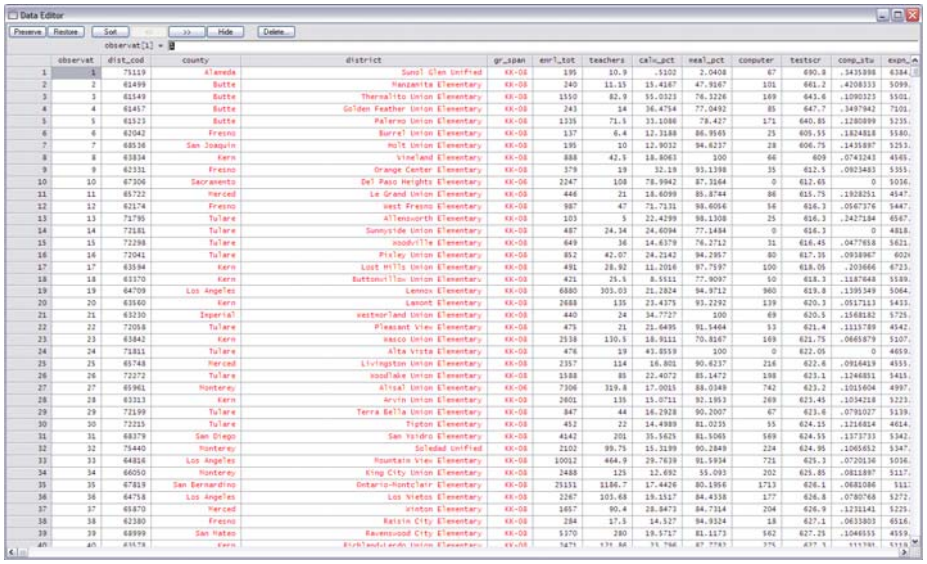
In this case, the Results window shows only the use command, however we can now see the list of loaded variables in the Variables window.

Data Editor

In order to have a look at the loaded data itself, you can type the following command:

edit

It opens the data editor:



The first column (in gray) contains the number of observations, 420 in our case. The dataset contains information for a set of US school districts (their names are in column 4).

Sorting your data

Sometimes it is useful (or required) to sort your data by a certain criterion. The simplest sorting command for both numerical and string variables, is sort:

sort testscr

The above command sorts the observations in ascending order of average district test score. Since sort supports only ascending order, a more flexible command is gsort, as it provides both ascending and descending ordering. The equivalent of the previous command is then

gsort testscr

If we want to sort *testscr* from the highest to the lowest value, we use the following syntax:

gsort -testscr

We can sort by several different criteria, as well. The following command sorts alphabetically the counties (*county*) in ascending order, while within counties, the observations are ordered in descending order of average district tests cores:

gsort county -testscr

In addition to sorting, Stata allows a reordering of the variables in the dataset. For example,

order str

puts the student-to-teacher ratio (*str*) on the top of the variable list in the variables window, while

order _all, alphabetic

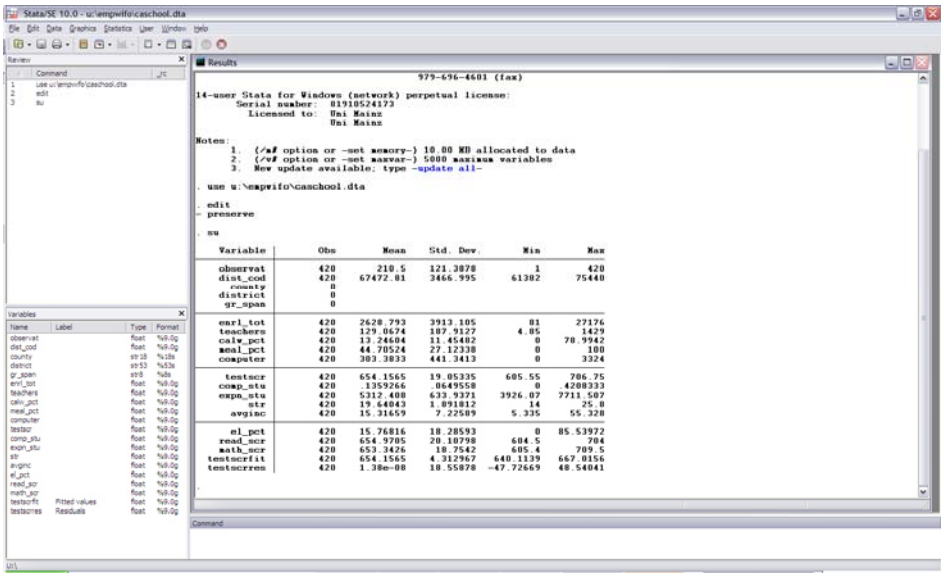
arranges all our variables in ascending alphabetic order, depending on their names.

Summary Statistics

Close the data editor and type the following command:

su

su stands for "**Su**mmary Statistics," that is, the most important descriptive statistics for the **numerical** variables in the dataset: number of observations (Obs), mean, standard deviation (Std. Dev.), as well as the smallest (Min) and largest (Max) values. No descriptive statistics can be calculated for non-numerical variables.



```
. su

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    observat |       420       210.5    121.3878         1        420
    dist_cod |       420    67472.81    3466.995     61382      75440
      county |         0
    district |         0
     gr_span |         0
-------------+--------------------------------------------------------
     enrl_tot |       420    2628.793    3913.105        81      27176
     teachers |       420     129.0674    187.9127     4.05       1429
     calw_pct |       420     13.24604    11.45402        0      78.9942
     meal_pct |       420     44.70524    27.12338        0        100
     computer |       420     303.3833    441.3413        0       3324
-------------+--------------------------------------------------------
      testscr |       420     654.1565    19.05335    605.55     706.75
     comp_stu |       420     .1359266    .0649558        0   .4208333
     expn_stu |       420     5312.408    633.9371  3926.07   7711.507
          str |       420     19.64043    1.891812       14       25.8
       avginc |       420     15.31659     7.22589    5.335      55.328
-------------+--------------------------------------------------------
        el_pct |       420     15.76816    18.28593        0   85.53972
     read_scr |       420     654.9705    20.10798    604.5        704
     math_scr |       420     653.3426     18.7542    605.4      709.5
   testscrfit |       420     654.1565    4.312967   640.1139   667.0156
   testscrres |       420     1.38e-08    18.55878  -47.72669   48.54041
```

Do-files

What you did so far can be referred to as *interactive* work: typing a command in the command window and directly executing it. Alternatively, one could write a *program* (a *do-file*), which runs several commands at once. The current course makes extensive use of such programs, since this is the usual practice in scientific research. Such an approach has two major advantages: first, it allows you to keep a complete record of your work in Stata, and second, it allows you to reproduce your results as often as you like. The latter is helpful, e.g., if you detect errors in your commands.

The do-files are text files that have the extension .do and contain multiple Stata commands. When you run the do-file, all commands that are included will be automatically executed. An important feature of a well-written program is not only the correctness and quality of the codes it contains, but also how understandable and well-documented it is. Therefore, one should try to include spaces and tabs to make the program easily readable, as well as comments to facilitate the understandability of the code itself. The comments are parts of the program that are not interpreted as commands, and they are simply ignored by Stata. They start with a star.

Example:

**This is a comment*

If you want to include a comment on multiple lines, the notation is as follows:

*/*This is a*

*comment */*

Alternatively, you can start both lines with a star. However, one has to use the second notation when one wants to designate only part of the content of a line as a comment.

Creation of a do-file: A do-file can be created in a simple text editor and later saved with the extension .do. An easier way is to use the do-file editor included in Stata: *Window → Do-file Editor → New Do-file*. You can then save the program with a name of your choice.

Example: Let us go back to the commands from our introduction example. Open the do-file editor and write:

```
*** Program caschool.do (introduction example)***
clear /*deletes the current dataset from the memory*/

** Loading the dataset
use U:\empBF\caschool.dta

** Descriptive statistics
su
```

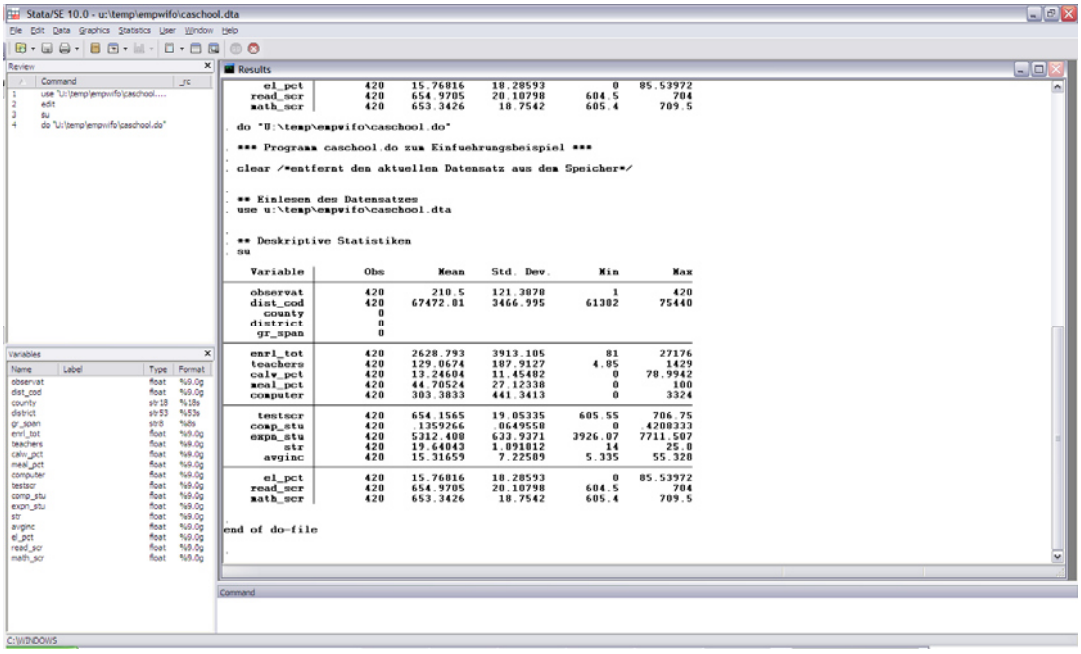
Save the document as caschool.do. Your first do-file is ready.

We use the command clear to delete the currently opened dataset. This is only relevant when we have previously loaded a dataset in Stata's memory. Stata can work only with one dataset at a time.

Running a program: There are various ways to run a do-file. For example, you can type in the command window:

do U:\empBF\caschool.do

Another possibility is to choose the do-file from the menu *File → Do…*. When we start the do-file, we get:



Log files

We would often like to get access to the results of a program without running the program again. For this purpose, one typically uses a *log-file*. The log-file depicts as text document all results that show up in the results window. If you copy the output in a Microsoft Word document, use *Courier New* to get aligned results.
Example: Open the do-file editor using the menu *Window → Do-file Editor → New Do-file*. Then, open the do-file caschool.do using *File → Open*.

Now you can extend it with the commands that will generate the log-file:

```
*** Program caschool.do (introduction example)***
clear /*deletes the current dataset from the memory*/


** Creating a Log-File
log using U:\empBF\caschool, replace

** Loading the dataset
use U:\empBF\caschool.dta
** Descriptive statistics
su

** Closing the log-file
log close
```

We save the file again under caschool.do. Running the do-file creates the file caschool.smcl (.smcl is the extension of Stata's log-files) in the directory U:\empBF\. Now, you can view the file using *File → Log → View…* and entering the path to caschool.smcl, hence U:\empBF\caschool.smcl.

Replace is the command used to replace the log-file after rerunning the do-file. Without it, you will receive a warning message.

General Syntax

Stata differentiates between small and capital letters. For example, district, District and DISTRICT are three different variables in Stata. The variables can contain up to 32 signs. Commands are always written with small caps.

Stata accepts shortcuts for commands as long as they are unambiguous, e.g.

- *su* for *summarize*
- *ge* for *generate*
- *reg* for *regress*

The same holds for variables. You can also use * to call several variables. E.g. The command

*su comp**

calls the descriptive statistics of all variables that start with *comp*, that is *computer* and *comp_stu*.

Creating new variables

You can create new variables using the command generate:

ge newvariable = function(variable)

The variable *newvariable* must not exist in the dataset already, otherwise you will receive an error message.

Example (taking the natural logarithm of the variable *testscr*):

ge logtestscr = log(testscr)

Other functions and operators, implemented in Stata are, for example:

exp(*variable*)	exponential function
abs(*variable*)	absolute value
sqrt(*variable*)	square root
variable^c	power of c
*variable**c	multiplication with c
variable/c	division by c
*var1*var2*	interaction term of two variables

If you want to substitute an already generated variable, you use the command replace instead of generate: e.g.

replace newvariable = log(variable).

To rename variables, one uses

rename variable,

and to delete variables

drop variable.

In many econometric applications, we use binary (dummy) variables – variables that take the value 1 if a certain condition is fulfilled, and zero otherwise. For example:

ge testscr_dummy = (testscr > 650)

Here, the new variable *testscr_dummy* has a value of one if the test score in the district is above 650 point, and zero in the opposite case. For most purposes, this command

suffices to correctly generate the required dummy variable. However, if our dataset is incomplete and *testscr* has missing values, the command above will erroneously attribute 1 to *testscr_dummy* for the missing data, while it should have attributed a missing value for the dummy variable as well. The reason is that Stata codes missing values as the largest possible number. The problem is circumvented by the following command (make sure that you have deleted *testscr_dummy*, using the drop *variable* command before you apply this alternative command):

ge testscr_dummy = (testscr > 650) if testscr !=.

where "if" introduces the condition ("!=" or "~=" is equivalent to the logical operation "not equal to)[1], and "." is the usual syntax in Stata to designate missing values. In our case, there are no missing test score values.

[1] Note that "equal to" has to be written with two equality signs: ==, larger (smaller) or equal is written as >= (<=).

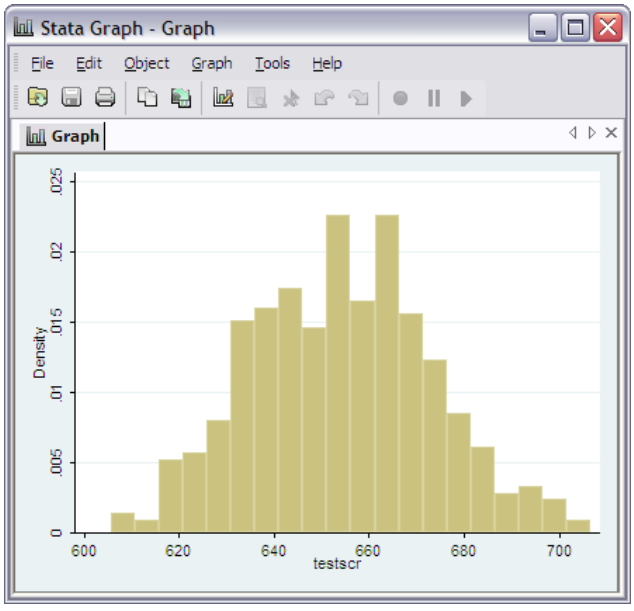
Data analysis with Stata

Assume that we are interested in the relationship between school class size and the students' results in standardized tests. The dataset caschool.dta contains the needed data on a school district level for California in 1998: *testscr* (*test score*) is the average performance on the *Stanford 9 Achievement Test* of the fifth-graders in a district, while *str* (*student-teacher ratio*) is the average ratio of students to teachers in a district. Given the descriptive statistics (see above), we can see that the average test score across all districts is 654.16 and that the average student-teacher-ratio is 19.64. It is useful to illustrate the data graphically.

Graphs

At first, we can have a look at the distribution of the variables of interest in a histogram. E.g., for the test score, we have:

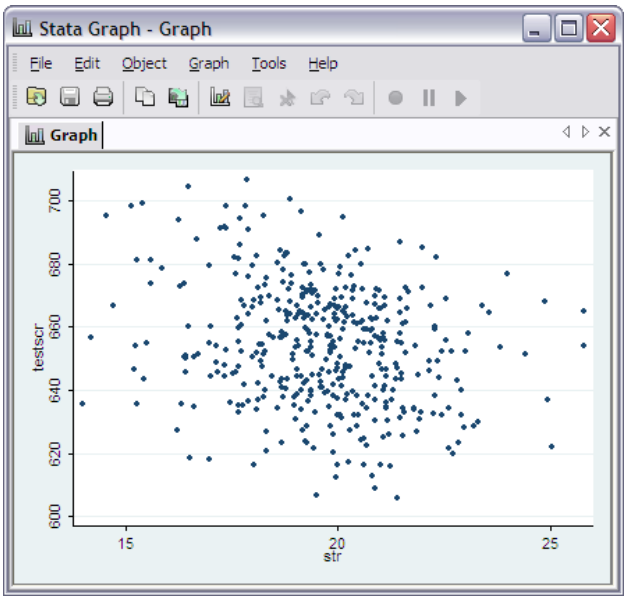
hist testscr



The graphical output appears in a separate window („Stata Graph“) and not in the results window as with the previous commands. Graphs are also not saved in log-files, so when needed they should be saved or printed separately.

Stata can easily create two-dimensional graphs. For example, to create a scatter plot, type:

twoway scatter testscr str

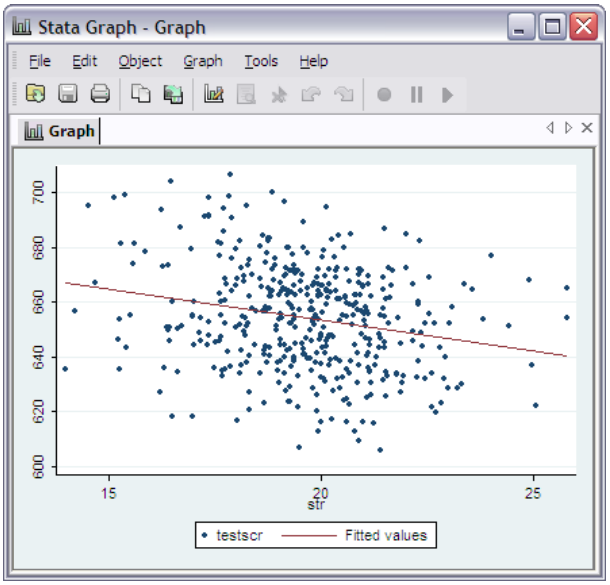


The command to create a twoway plot is

twoway plottype y x

The plot type could be, e.g.: scatter, line, connected (points connected with a line), or lfit (fit of a simple linear regression with a constant). You can also combine two plots of different types:

twoway (scatter testscr str) (lfit testscr str)



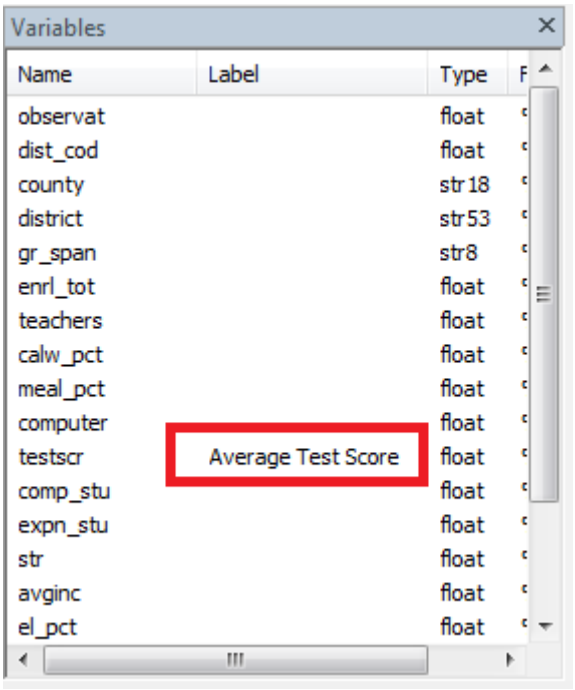
This graph shows that there appears to be a negative relationship between both variables.

Documenting the dataset

In order to make the dataset understandable to others, it is advisable to provide brief descriptions of your variables. The command label is an easy way to do so. For example, let us label the variable testscr:

label var testscr "Average Test Score"

If we carefully examine our variables window, we will notice that our new description shows up in the column Label:

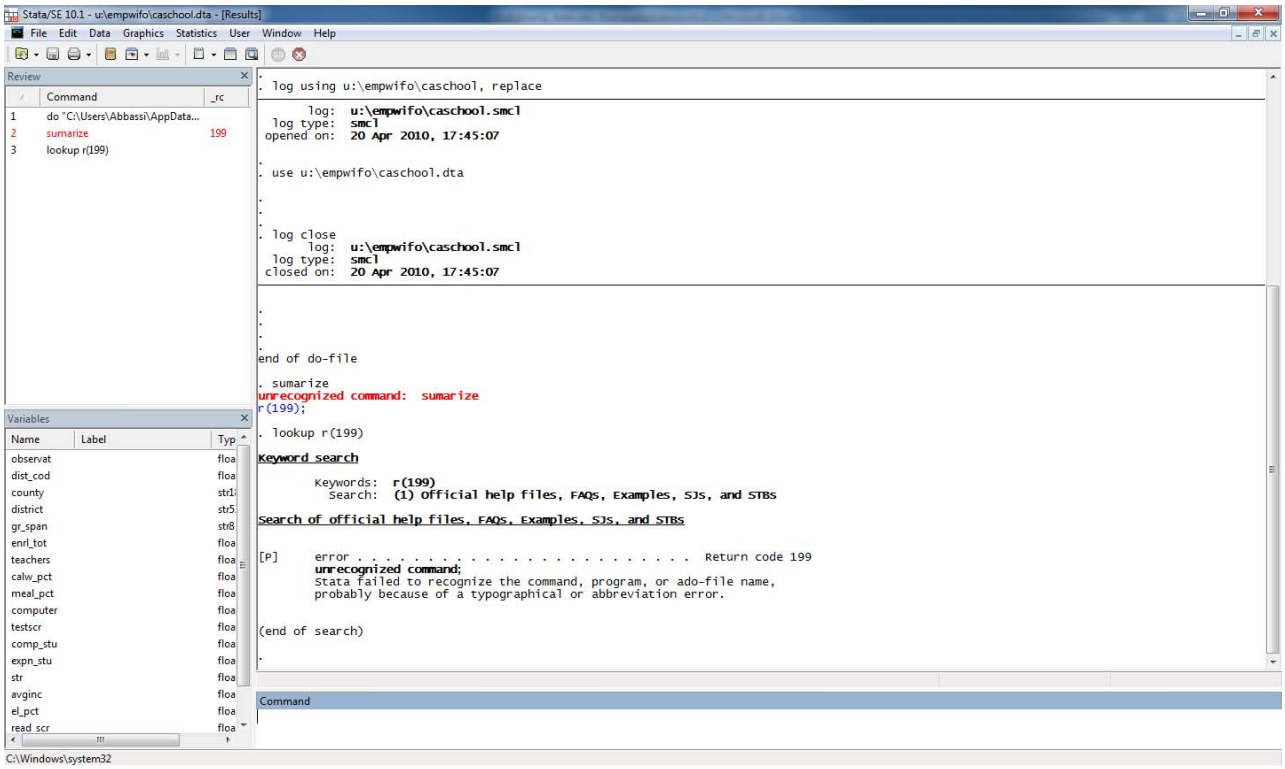


You can undo the labeling by repeating the previous command without the label string itself:

label var testscr

Stata Help

In the menu *Help*, you can find extensive explanations for the different functions in Stata. One can search by key words (*Search…*) or directly for a particular Stata command (*Stata command…*). The latter helps you only if you already know the name of the command that you need help for. When you receive an error message, you will see an error code:



Here, we have misspelled the command summarize. The command lookup r(199) (here we should write the specific error code) provides an additional suggestions for the possible causes of the error.

Basic Command Structure

To use Stata commands efficiently, you have to know their structure and additional functionality. Type the help command to analyze a command's syntax and available options. The basic command structure looks like this:

[prefix:] command [varlist] [=exp] [if] [in] [, options],

where

prefix: is a Stata command, referring to command
command: is the main Stata command
varlist: is a list of variables
exp: is a mathematical expression
if: is an algebraic expression
in: is an observation range
options: a list of options for command

The information that appears in square brackets is optional. Note that only the command options appear after the comma.

Example:

quietly: reg testscr str, robust

Here, apart from the option robust, we use the prefix quietly for our previous regression. This prefix suppresses the output of a command (in the case reg). This prefix is useful when we do not need to view the results of a command (e.g., if it is being repeated in a long loop), but we do want to use them for further analysis. For example, let us calculate the fit of the regression above:

predict testscrfit1

We can now compare the predictions of both regressions using the browse command to open the Data Editor (Browse)[2]:

browse testscrfit testscrfit1

[2] The `browse` command is an alternative to `edit` and displays the results using the browse mode of the data editor. The difference between the Data Editor (Browse) and the Data Editor (Edit) is that in the former you can only view the data and not edit it. This mode has certain advantages, e.g. you can avoid accidentally changing a data point of a variable.

We see that, as expected, the values are the same.

Let us now exemplify the if-expression:

list if testscrfit1 >= 650

This command displays in the results window all observations, for which the fit is equal to or above 650 points. To have an idea how many observations fulfill this criterion, we could type

su testscrfit1 if testscrfit1 >= 650

We find that our regression results fit average test scores above 650 points for 358 out of the total of 420 school districts.

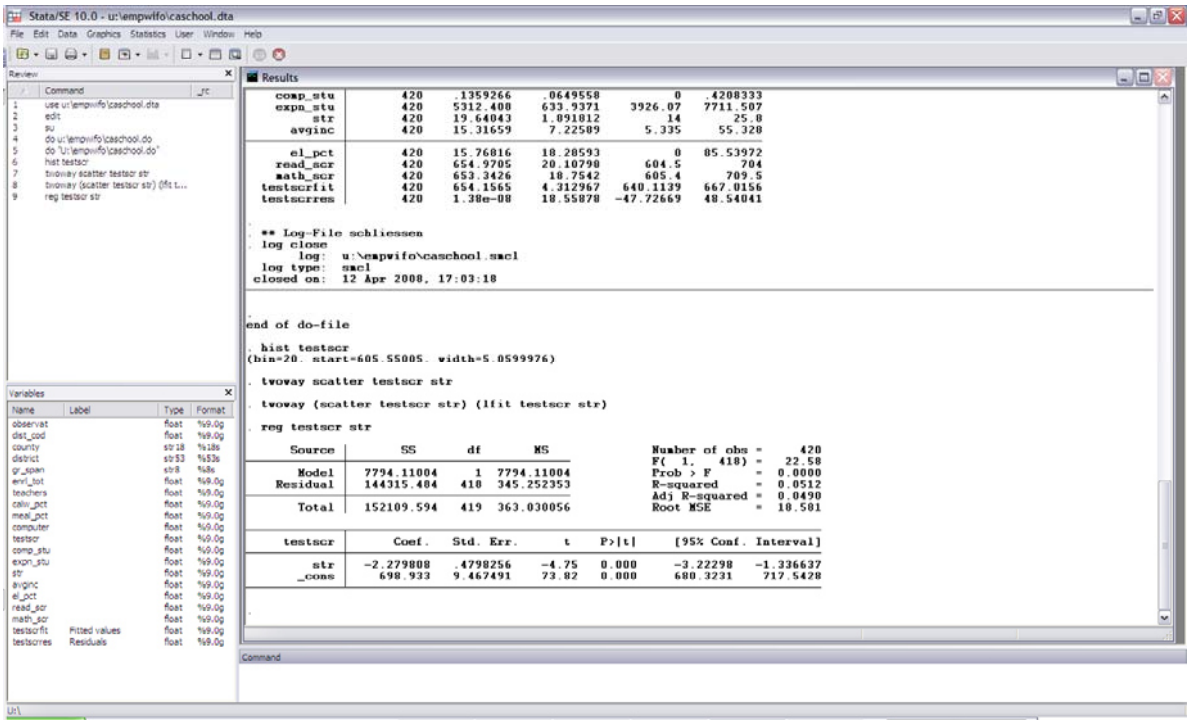
Linear regression

The line above is a regression line. It is the result of an Ordinary Least Squares (OLS) estimation, implemented by using the command regress.

Example:

reg testscr str

With this command, we estimate a linear relationship between *testscr* as dependent (or endogenous) variable and *str* as independent (exogenous, explanatory) variable using the OLS method. You can add further exogenous variables, but the first variable in the list is always the dependent variable. Below is the output from our regression.



Stata automatically includes a constant in the estimation. If we want to run a regression without a constant, we use:

reg testscr str, noconstant

This is used only in rare cases, though. To calculate heteroscedasticity-robust standard errors (which we normally do), we extend the command in the following way:

reg testscr str, robust

To generate the fit of a regression (that is, the predicted values of the dependent variable), we use:

predict testscrfit

Stata creates a new variable for the predicted values with the name (defined by the user) *testscrfit.* To extract the residuals of the previous regression, use the command:

predict testscrres, resid

Please note that the command predict can be used only in combination with a regression, namely with the regression preceding the command predict.

We now complete our program caschool.do with the additional commands that we just learned. Open caschool.do in the do-file editor and write:

```
*** Program caschool.do (introduction example)***
clear /*deletes the current dataset from the memory*/

** Creating a log-file
log using U:\empBF\caschool, replace

** Loading the dataset
use U:\empBF\caschool.dta

** Descriptive statistics
su

** Creating a scatterplot with a regression line
twoway (scatter testscr str) (lfit testscr str)

** Regression with robust standard errors
reg testscr str, robust

** Deriving the fit and the residuals
predict testscrfit
predict testscrres, resid

**Saving the extended file
save U:\empBF\caschool2.dta, replace

** Closing the Log-File
log close
```

The command replace in the save-command means that the existing dataset will be overwritten when we run the code again. It is important, therefore, to save the changed dataset with a new name (in our case caschool2.dta). We must always keep the original data file unchanged. Ideally, you should have a separate folder with a copy of the original dataset. We save the program under caschool.do again, and we can now run it.

If there is a typo in the do-file, Stata will issue an error message, and the program will not run until the end. Please note that in case of an error, you have to close the log file manually with log close before you run the file again. Otherwise, you will get the error message log file already open.