

5) Close Data Editor Window.



6) To save your data set click on , you will receive a standard Windows dialog-box. STATA suggests you to save data in a .dta file that could be read by STATA. Note, that there are many versions of STATA. STATA 8 could read .dta files created by the previous versions, but not vice versa!!! To send your data set to a person who uses older version of STATA than you, you should save your data in a .dta format compatible with the previous version of STATA.

- **How to open already existing data set?**

1) If you want to use a Stata-format (dta) data set previously saved on your disk, select File – Open from the top menu bar, or click on  to receive a standard Windows dialog- box.

2) You may want to use one of data sets provided by STATA Corporation for demonstration purposes. The whole list of these data sets you could find on <http://www.stata-press.com/data/> . I will use data set auto.dta. To download it from the web print command

- **webuse auto**

It happens that this particular data set is also provided with standard STATA package and is saved on your local disk. If you want to use this copy print command

- **sysuse auto**

- **How can I see the content of a data set?**

From now let's start using data set auto.dta, which contains an information on reselling price and characteristics of 74 used cars.

1) Print in the command line **describe** you will receive the following screen:

```

Stata Results

. describe

Contains data from C:\Stata8\ado\base/a/auto.dta
  obs:          74          1978 Automobile Data
  vars:         12          3 Oct 2002 13:53
  size:        3,478 <99.9% of memory free>

-----
variable name   storage   display   value   variable label
                type     format    label
-----
make            str18    %-18s
price           int      %8.0gc   Price
mpg             int      %8.0g    Mileage <mpg>
rep78           int      %8.0g    Repair Record 1978
headroom        float    %6.1f    Headroom <in.>
trunk           int      %8.0g    Trunk space <cu. ft.>
weight          int      %8.0gc   Weight <lbs.>
length          int      %8.0g    Length <in.>
turn            int      %8.0g    Turn Circle <ft.>
displacement    int      %8.0g    Displacement <cu. in.>
gear_ratio      float    %6.2f    Gear Ratio
foreign         byte     %8.0g    origin    Car type

-----more-----

```

It contains the names, descriptions and format of all variables. You could also receive the same result selecting Data – Describe from the top menu bar.

2) To see data print a command list. You will find that for a large data set it's not convenient to list the whole data set. Put variables name after the command if you want to see only these variables. In order to print on screen just several observations use in. For example, command **list price weight in 1/7** gives you values of price and weight only for observations from the first to the seventh.

```

. list price weight in 1/7

   +-----+
   | price | weight |
   +-----+
  1.  4,099 |  2,930 |
  2.  4,749 |  3,350 |
  3.  3,799 |  2,640 |
  4.  4,816 |  3,250 |
  5.  7,827 |  4,080 |
  6.  5,788 |  3,670 |
  7.  4,453 |  2,230 |

```

3) You could also see summary statistics (mean, std, max, min) for your data by printing **summarize**. If you instead use a command **summarize price mpg**, you will receive summary statistics only for these two variables.

4) You could want to calculate correlations between different variables. Use for this purpose a command `correlate`. Note, that this command could be used only numerical variables. For example, a command

```
·correlate price mpg weight length gear_ratio
```

gives you the following output:

```
. correlate price mpg weight length gear_ratio
(obs=74)
```

	price	mpg	weight	length	gear_ratio
price	1.0000				
mpg	-0.4686	1.0000			
weight	0.5386	-0.8072	1.0000		
length	0.4318	-0.7958	0.9460	1.0000	
gear_ratio	-0.3137	0.6162	-0.7593	-0.6964	1.0000

How can I run a simple regression?

1) You can use the command `regress`. To see all options of this command print a command help `regress`. The most simple binary regression of price on mileage could be done by a command

```
regress price mpg
```

```
. regress price mpg
```

Source	SS	df	MS			
Model	139449474	1	139449474	Number of obs =	74	
Residual	495615923	72	6883554.48	F(1, 72) =	20.26	
Total	635065396	73	8699525.97	Prob > F =	0.0000	
				R-squared =	0.2196	
				Adj R-squared =	0.2087	
				Root MSE =	2623.7	

	price	Coef.	Std. Err.	t	P> t	[95% Conf. Intvall]
mpg		-238.8943	53.07669	-4.50	0.000	-344.7008 -133.0879
_cons		11253.06	1170.813	9.61	0.000	8919.088 13587.03

2) To perform a multiple regression of price on different characteristics of a car use `regress price [list of variables]`.

3) A command `predict pr` creates a new variable `pr`, that equals to the fitted value of the mostly currently performed regression.

4) To receive residuals you can exploit two options:

- `predict e, resid`
- `generate e=price-pr`

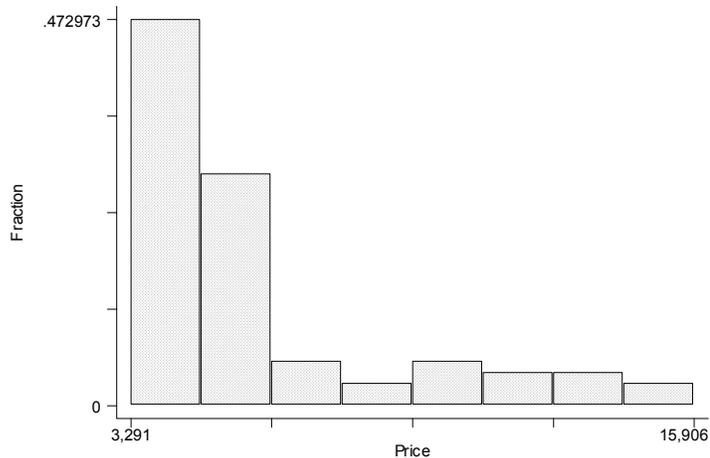
They both produce a new variable `e` that contains residual from the regression.

How could I visualize my results?

1) STATA 8 has different graphic options than STATA 7. It happens that new graphics works slower than the previous version. We recommend you to use in this class STATA 7 graphics. In fact, you are just supposed to add 7 after a name of a command.

2) Usage of a command `graph7` with a name of one variable produces a histogram.

`graph7 price, bin(8)`

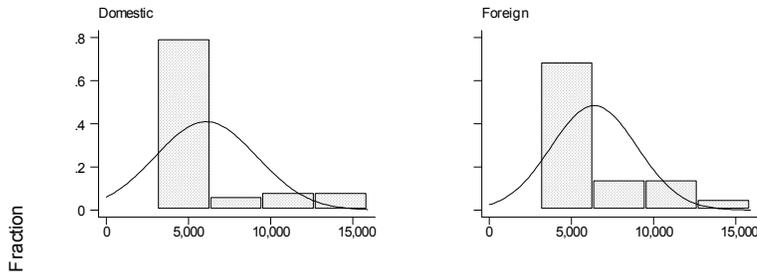


3) This command has a lot of options that might be used after a comma. Options could be:

norm	Draws a normal curve over the histogram, using sample mean and std. You could also specify parameters by using <code>norm(,)</code>
title("text")	Adds a title of the graph in large letters at the bottom of the graph.
t1(), t2(), b1(), b2(), l1(), l2(), r1(), r2(),	Allow to add titles to every possible side of the graph
xlabel	Label x axis
ylabel	Label y axis
saving(file name)	Saves the graph in a file with an extension <code>.gph</code> (if it isn't specified explicitly)
by(varname)	Draws several graphs for groups defined by varname

For example, below is a product of a command

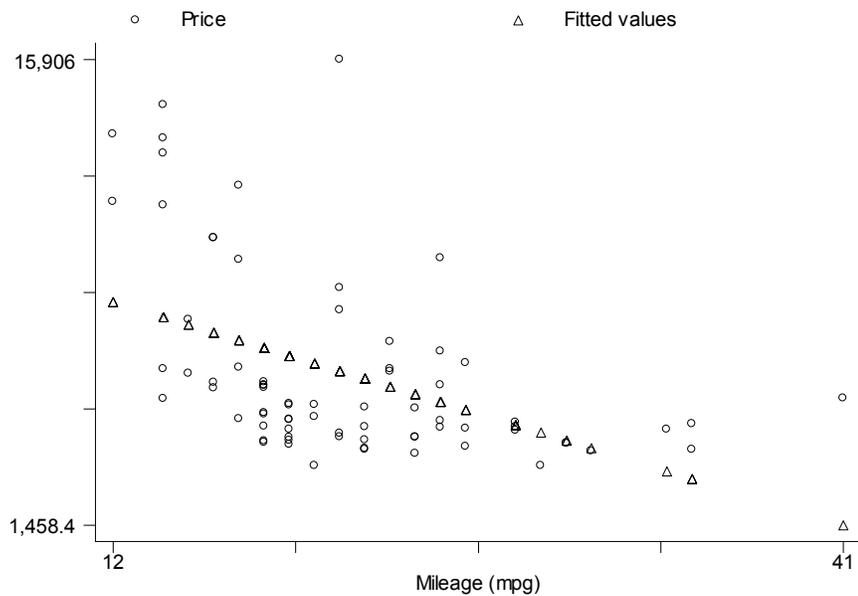
```
graph7 price, by( foreign) title("Price by foreign  
origin ") xlabel ylabel norm
```



Price
Price by foreign origin

4) Command `graph7` with at least two variables name will draw a scatterplot.

`graph7 price pr mpg`



5) You could specify plotting symbols by adding **`symbol(s..s)`**. Specify **s** as

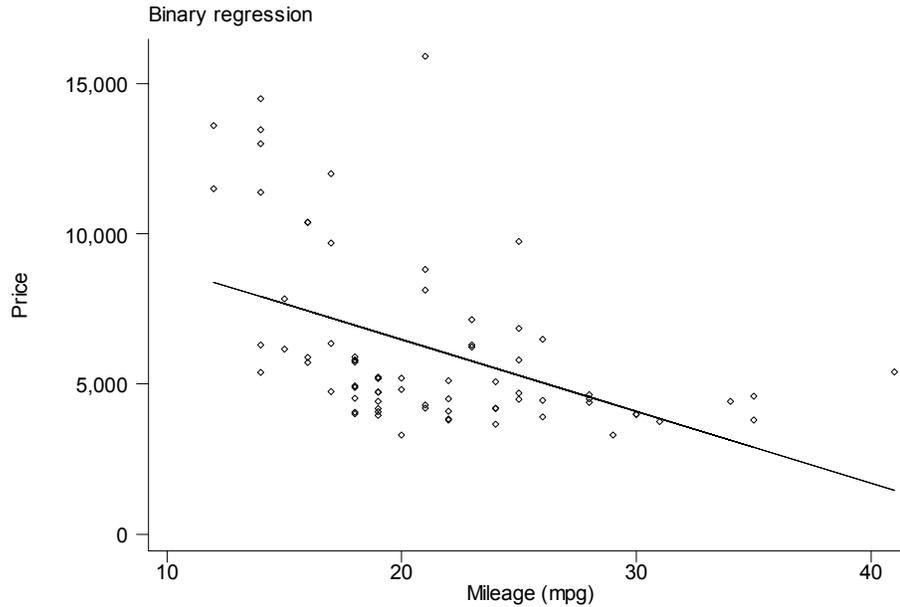
O	large circle
T	triangle
o	small circle
d	diamond
i	invisible
[varname]	Variable to be used as text

6) You also can connect points by adding **`connect(c..c)`**. Specify **c** as

.	do not connect (default)
l	draw straight lines between points

Below is an output of the command

```
graph7 price pr mpg, symbol(di) connect(.1) xlabel
ylabel t1("Binary regression")
```



- **How do I perform t-tests?**

ttest command performs one-sample, two-sample, and paired t tests on the equality of means.

One sample t-test:

To test the hypothesis that price of car has a mean of 100 use the following command:

```
ttest price=100
```

Output:

One-sample t test

Variable	Obs	Mean	Std. Err.	Std. Dev.	[95% Conf. Interval]
price	74	6165.257	342.8719	2949.496	5481.914 6848.6

Degrees of freedom: 73

Ho: mean(price) = 100

Ha: mean < 100
t = 17.6896
P < t = 1.0000

Ha: mean != 100
t = 17.6896
P > |t| = 0.0000

Ha: mean > 100
t = 17.6896
P > t = 0.0000

Two sample t-test:

To test the hypothesis that mean price of foreign and domestic cars is the same, use the command:

ttest price, by(foreign) unequal

Output:

Two-sample t test with unequal variances

Group	Obs	Mean	Std. Err.	Std. Dev.	[95% Conf. Interval]	
Domestic	52	6072.423	429.4911	3097.104	5210.184	6934.662
Foreign	22	6384.682	558.9942	2621.915	5222.19	7547.174
combined	74	6165.257	342.8719	2949.496	5481.914	6848.6
diff		-312.2587	704.9376		-1730.856	1106.339

Satterthwaite's degrees of freedom: 46.4471

Ho: mean(Domestic) - mean(Foreign) = diff = 0

Ha: diff < 0	Ha: diff != 0	Ha: diff > 0
t = -0.4430	t = -0.4430	t = -0.4430
P < t = 0.3299	P > t = 0.6599	P > t = 0.6701

• How and why do I create do files?

A do file is a program file where I list all the commands I want to execute. We need do files to perform complicated and sometimes repetitive data manipulations. For instance, sometimes economists need to run 1000 regressions. It would be extremely tedious to run all those regressions by hand. Instead an economist can create a program in the do file and execute it, prompting Stata to run all 1000 regressions by itself and return final results.

To open a do file, click on Window and scroll down to Do-file-Editor or click on . Open the editor. You will be able to type commands directly into the editor and then execute those commands.

Example of a do file:

```
/*This program regresses headroom on miles per gallon, predicts mpg*/  
/* and graphs headroom vs mpg and regression line*/  
reg mpg headroom  
predict miles  
graph7 mpg miles headroom, c(.1)
```

Please note that anything within /* */ is regarded by Stata as comments and Stata does not do anything with what is inside those symbols. We advise you to comment your code so that you remember at a later time what the code means.

To execute the do file hit the  button and the program will be executed in your main window. If there is something wrong with the program, the main (black) window will tell

you where the mistake is. To run a part of the code, highlight the part you want to run, and Stata will run just that part in the main window.

To save the do file, click on the little disk icon in the do file editor and save. Make sure to save the file with a .do extension.

How to create a list of dummies?

If you have a variable, say *State*, such that for every different value of it you want to create a separate dummy variable you should use the following command:

```
tabulate State, generate (st)
```

This command will create variables st1, st2, and so on. The number of variables is equal to the number of different values of State.

Why Do I need to use global variables?

If you have a list of variables that you always use together, and it's time consuming to print their names again and again, you could create one global variable (group of variables). If I have variables st1, st2, st3, st4, I could do the following

```
global stvar " st1 st2 st3 st4"
```

After that you could use the group as a whole. For example, the command:

```
reg vio $stvar
```

regresses vio on st1, st2, st3, st4; that is, it's equivalent to

```
reg vio st1 st2 st3 st4
```

The command

```
test $stvar
```

tests 4 restrictions that all four coefficients on the regressors st1, st2, st3, st4 are equal to zero simultaneously.

How do I run a regression with a very large number of regressors?

STATA uses the following convention: printing *st** refer to all variables starting with *st*. Assume that you want to regress variable *vio* on 140 dummy-regressors: *st1, st2, ..., st140*. Then print the following command

```
reg vio st*, robust
```

If after that you want to test the hypothesis that all coefficients before "*st*" variables are equal to zero simultaneously, use the command **testparm**:

```
testparm st*
```

Does STATA have a special command to run a fixed effects regression?

Yes, it does. If you want regress a variable *vio* on a variable *shall* using fixed effects for different states, you could print

```
areg vio shall, absorb(state) robust
```

The only problem with this – you cannot perform an F-test on the hypothesis that all fixed effects are simultaneously equal to zero. For that you need to create a list of dummies as described above and then run a regression with many regressors.

How can I run a probit regression?

There is a special command **probit** (that could be reduced to **prob**) that has the same syntax as **regress**. For example, if you have a binary variable **employed** and want to regress it on education, age and experience, using probit model, you should type:

```
probit employed educ age exper, r
```

If you want to test the hypothesis that both coefficients before education and age are equal to zero simultaneously, then you could use the command **test** (the same as in OLS regression), STATA will perform chi-squared test:

```
test educ age
```

How can I do arithmetic calculations using STATA?

For this you can use scalar expressions. By printing

```
scalar scalar_name=expression
```

You receive a new scalar. For example, the command

```
scalar a=25*34
```

Calculates the product of 25*34 and store it in scalar *a*. You can also use different functions with scalars (such as log or exp). For the normal cdf (in the lectures we called it Φ) use the command **normprob**. So, if you type

```
scalar b=normprob(a)
```

STATA calculates the values of normal cdf at the argument stored in scalar *a* (in our case it 25*34=850), and then stores it in *b*. To see the value of scalar use **display** command (could be reduced to **dis**)

```
display b
```

I want to use values of coefficients from the last estimated regression. What should I do?

Imagine that you last regression was

```
probit employed educ age exper, r
```

You want to calculate the sum of coefficient on *age* and twice the coefficient on *education*. To refer to the coefficients you could use **_b[name of variable]**

```
scalar c=_b[age]+2*_b[educ]
```

How can I run an IV regression?

You can do it in two different ways: perform two stages by hand or use standard STATA command.

For example, assume that you want to regress wage on education and experience, but you are afraid of endogeneity of education and choose as instruments mother's education and father's education (put aside the discussion of whether these are valid instruments).

If you are doing the calculation "by hand" you need to perform two stages:

Stage 1.

```
reg educ exper motheduc fatheduc,r  
predict educhat
```

Stage 2.

```
reg wage educhat exper, r
```

The regression in the second stage gives the right estimates of coefficients, but **the standard errors are wrong** (as well as t-stats and p-values). That's why we need to use a special command, **ivreg**.

```
ivreg wage exper (educ = motheduc fatheduc),r
```

This command gives the same estimates as the two stage procedure but with correct standard errors.

Why then do we bother to run two stage regressions "by hand"?

Because the first stage regression can help us determine whether instruments are weak or strong. To understand whether instruments are strong we need to do the following: run the first stage regression

```
reg educ exper motheduc fatheduc,r
```

Then perform F-test testing that coefficients on both instruments are zero:

```
test motheduc fatheduc
```

The rule of thumb: if F-statistic is bigger than 10 the instruments are strong.

How can I perform a J-test?

STATA does not have a standard command for that, so, you have to do it by hand. In our example above, when we have a dependent variable *wage*, one exogenous variable *exper*, one endogenous variable *educ* and two instruments *motheduc* and *fatheduc*, we are dealing with an overidentified case, and can perform J-test.

In order to do this first run IV regression using **ivreg**:

```
ivreg wage exper (educ = motheduc fatheduc),r
```

Then construct the residuals. It's very important that you use **ivreg** command rather than two stage procedure "by hand" since the residuals will be different.

```
predict e, resid
```

Now you need to regress residuals on instruments and exogenous variables:

```
reg e exper motheduc fatheduc,r
```

and perform a test that coefficients before both instruments are zeros;

```
test motheduc fatheduc
```

STATA will calculate the F-statistic for you. Unfortunately F-test itself is not valid (p-value will be wrong), so you need to correct it: J-statistic is equal to F-statistic calculated by STATA on the last step multiplied by the number of instruments, in our case $J=2 \cdot F$ (you can get the F-statistic calculated by STATA on the last step, it's kept in a special scalar $\mathbf{r}(\mathbf{F})$). J-statistic is distributed with chi-squared distribution with degrees of freedom equal to the number of instruments minus the number of endogenous variables (in our case $2-1=1$). You can calculate the p-value of the test in STATA

```
sca J=r(F)*2
dis "J-stat=" J
sca pval= chiprob(1,J)
dis "p value=" pval
```

If p value is smaller than significance level (often 0.05) then we reject the hypothesis of instrument exogeneity (meaning that we have a problem with our instruments; some of the instruments are correlated with the error term). If p-value is bigger than significance level, then we think that our instruments are exogenous.

Making Stata recognize your data as a time series:

In order for Stata to treat your data as a time series, you need to create date variables in Stata format.

If your data is on a monthly basis and your starting date is January 1971, use the following commands to generate data from date January 1971 to the end date of the sample and to make Stata treat your data as a time series:

```
gen date = m(1971m1) + _n-1
format date %tm
sort date
tsset date
```

If your data is on a quarterly basis use the commands below to generate data from first quarter of 1971 to the present:

```
gen date = q(1971q1) + _n-1
format date %tq
sort date
tsset date
```

If you need to create a difference variable that represents a change in some variable from last period, call the variable Δy , use the following command:

```
gen delta_y = y[_n] - y[_n-1]
```

Autoregressive Processes in Stata

To regress variable y on its first lag, like in the model $y_t = \beta_0 + \beta_1 y_{t-1} + u_t$ use the command:

```
reg y L(1/1).y, r
```

In general to refer to lags of a variable use the syntax $L(x/y).var_name$ where x = number of periods from t that the starting lag of variable var_name lies and y = number of periods from t that the last lag of variable var_name lies. Using this command creates $y-x+1$ lags of variable var_name .

To regress variable y on its third and fourth lag, like in the regression equation $y_t = \beta_0 + \beta_1 y_{t-3} + \beta_2 y_{t-4} + \beta_3 y_{t-5} + u_t$ use the command:

```
reg y L(3/5).y, r
```

to test whether lags 4 and 5 from the model above are significant, use F-test command:

```
test L4.y L5.y
```

to get adjusted R squared for the regression, type in command below after running the regression:

```
dis "Adjusted Rsquared =" _result(8)
```

AIC and BIC in Stata

To choose the number of lags in a forecasting model, one can implement the BIC (Bayesian Information Criterion) or AIC (Aikaike Information Criterion). Remember that only under BIC estimates are consistent. You can calculate BIC and AIC by hand using the formulas from lecture 23 or you can do them in Stata.

To get the optimal number of lags in an AR model with y being regressor, type in the command:

```
varsoc y, maxlag(8)
```

note that here I specified that I want the maximum number of lags from which to choose to be 8. You can make this number as big or small as you want.

AR model with HAC standard errors:

If you want to run an AR(1) regression regressing y on its first lag ($y_t = \beta_0 + \beta_1 y_{t-1} + u_t$) with HAC standard errors type in the following command:

```
newey y L(1/1).y, lag(6)
```

Here you can vary the number of lags in parentheses. Remember that this number is determined by the formula $\frac{3}{4} * T^{1/3}$