Introduction to Stata – Session 2

Tarjei Havnes

[1]ESOP and Department of Economics
University of Oslo

[2]Research department
Statistics Norway

ECON 3150/4150, UiO, 2012

Before we start

1. Download the files country1.dta and country2.dta from the course homepage
   - http://www.uio.no/studier/emner/sv/oekonomi/ECON4150/v12/
2. Save the file to the folder statacourse
3. Go to kiosk.uio.no (Internet Explorer!) and log on using your UIO user name
4. Navigate to Analyse (english: Analysis)
5. Open StataIC 11

Data types

Revisit the auto dataset and notice the numeric and string data types

```
. use auto
(1978 Automobile Data)

. describe

Contains data from auto.dta
  obs:             74                          1978 Automobile Data
 vars:             12                          13 Apr 2009 17:45
 size:          3,774 (99.9% of memory free)    (_dta has notes)
------------------------------------------------------------------------------
              storage   display     value
variable name   type    format      label      variable label
------------------------------------------------------------------------------
make            str18   %-18s                  Make and Model
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
------------------------------------------------------------------------------
Sorted by:  foreign
```

We will discuss each in turn

Numeric data types

Stata distinguishes the following numeric data types:

Type	min (max)	dist to 0	bytes
`byte`	-127 (100)	1	1
`int`	-32,767 (32,740)	1	2
`long`	-2,147,483,647 (...620)	2	4
`float`	$-1.70141173319*10^{38}$	10^{-38}	4
`double`	$-8.9884656743*10^{307}$	10^{-323}	8

About floats:

- default data type
- about 7 digits of accuracy (123456789 is rounded to 123456792)

When storing identification numbers, rounding could matter:

- If id's are integers and take 9 digits or less, store them as longs; otherwise as doubles

Saving space

Don't need a float to store an indicator variable, or an int to store a byte:

```
. ta rep78

     Repair |
Record 1978 |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2        2.90        2.90
          2 |          8       11.59       14.49
          3 |         30       43.48       57.97
          4 |         18       26.09       84.06
          5 |         11       15.94      100.00
------------+-----------------------------------
      Total |         69      100.00

. compress
mpg was int now byte
rep78 was int now byte
trunk was int now byte
turn was int now byte
make was str18 now str17

. d

Contains data from auto.dta
  obs:            74                          1978 Automobile Data
 vars:            12                          3 Apr 2011 13:02
 size:         3,404 (99.9% of memory free)   (_dta has notes)
[snip]

. di 3404/3774
.90196078
```

Adding and changing variables

```
. sum weight

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      weight |         73    3005.205    772.7712        1760       4840

. sum weight, detail

                        Weight (lbs.)
-------------------------------------------------------------
      Percentiles      Smallest
 1%          1760            1760
 5%          1830            1800
10%          2020            1800       Obs                  73
25%          2240            1830       Sum of Wgt.          73

50%          3180                       Mean           3005.205
                        Largest         Std. Dev.      772.7712
75%          3600            4290
90%          4030            4330       Variance       597175.3
95%          4290            4720       Skewness       .1743642
99%          4840            4840       Kurtosis       2.162421

. g heavy = 0 if weight<=4000
(9 missing values generated)

. replace heavy = 1 if weight>4000
(9 real changes made)

// what is the data type?
// do you need to worry about missings?
// try: g byte heavy2 = weight > 4000 if weight<.
```

Functions

When generating variables you can use functions and expressions

- `gen lninc = ln(income + 1)`

Functions are available -`help functions`-

- mathematical functions: `abs()`, `int()`, `round()`, `sqrt()`
- random numbers: `runiform()`, `rnormal()`
- prob distributions: `normal()`, `ttail()`, `invttail()`

and many more...

-egen- is a smart generate

- `by foreign : egen maxprice = max(price)`
- `egen meany = rowmean(y*)`

Variable naming

The command line is a powerful tool, smart naming of your variables helps

Choose you variable names such that you

1. minimize typing
   - no uppercase (Female)
   - no underscore (aar_forste_reg_uh_t, aar_andere_reg_uh_t)
2. can effectively use wildcards
   regyr1, regyr2

Use variable labels to document

Documenting - Labels

Document your data by attaching labels to variables:

```
. desc price

              storage   display     value
variable name   type    format      label      variable label
------------------------------------------------------------
price           int     %8.2f                  Price

. label var price "Price (USD)"

. desc price

              storage   display     value
variable name   type    format      label      variable label
------------------------------------------------------------
price           int     %8.2f                  Price (USD)

. label var price ""

. desc price

              storage   display     value
variable name   type    format      label      variable label
------------------------------------------------------------
price           int     %8.2f

// what about: label var heavy "Car is heavier than 4000 lbs"
```

Documenting - Labels

```
. l foreign in 52/54

     +----------+
     |  foreign |
     |----------|
 52. | Domestic |
 53. |  Foreign |
 54. |  Foreign |
     +----------+

. d foreign

              storage   display     value
variable name   type    format      label      variable label
------------------------------------------------------------------------------
foreign         byte    %8.0g       origin     Car type
```

Documenting - Labels

```
. tab foreign

   Car type |      Freq.     Percent        Cum.
------------+-----------------------------------
   Domestic |         52       70.27       70.27
    Foreign |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00

. sum price if foreign==Foreign
Foreign not found
r(111);

. sum price if foreign=="Foreign"
type mismatch
r(109);
```

Documenting - Labels

```
. tab foreign, nolabel

   Car type |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |         52       70.27       70.27
          1 |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00

. sum price if foreign==1

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         22    6384.682    2621.915      3748      12990

//What about: sum price if foreign
```

Documenting - Labels

Adding a label consists of two steps

1. define a mapping from values to labels: a value label
2. associate the value label (mapping) to the variable

```
// label define lblname # "label" [# "label" ...]
. label define record 1 "Poor" 2 "Fair" 3 "Average" 4 "Good" 5 "Excellent"

// label value varlist [labelname]
. label value rep78 record

. tab rep

     Repair |
Record 1978 |      Freq.     Percent        Cum.
------------+-----------------------------------
       Poor |          2        2.90        2.90
       Fair |          8       11.59       14.49
    Average |         30       43.48       57.97
       Good |         18       26.09       84.06
  Excellent |         11       15.94      100.00
------------+-----------------------------------
      Total |         69      100.00
```

value labels can be recycled!

Formats

Stata relies on formats when displaying data:

```
. l price in 1/2

     +-------+
     | price |
     |-------|
  1. | 4,099 |
  2. | 4,749 |
     +-------+

. format price %8.2f

. l price in 1/2
     +----------+
     |    price |
     |----------|
  1. |  4099.00 |
  2. |  4749.00 |
     +----------+
```

This is especially useful in combination with -display-

```
. di "Pi equals" %8.2f _pi ", or more exactly " %14.0g _pi
Pi equals     3.14, or more exactly  3.14159265359
```

Strings

Strings are good for ID's, but most of the time we do not want our data as strings:

- recurrent strings can take a lot of space
- we cannot do our calculations with string vars

When numbers are stored as strings we can easily convert them

- `gen numvar = real(stringvar)`
  fast, single var, non numbers ("1,233") are converted to missing
- `destring stringvar, replace`
  smart, handles many vars at a time, not so fast

With categorical string data we use

- `encode catvar, gen(newcatvar)`

this preserves the information in the data as value labels

Strings

```
// do you understand this? (try it step by step)
. u largeauto
. d
. g make2 = word(make,1)
. d
. cl make2
. tab make2
. encode make2, gen(manuf)
. tab manuf
. drop make make2
. rename manuf make
. d
. order make
. compress
. drop if rep78>=.
. save myauto
```

Keeping track of dates

Stata can store points in time as numbers

- year, month, date, time...
- all references are relative to Jan 1, 1960
- convenient for sorting and extracting

Keeping track of dates

Stata can store points in time as numbers

- year, month, date, time...
- all references are relative to Jan 1, 1960
- convenient for sorting and extracting

Type	Format	-1	0	1
clock	%tc	31dec1959 23:59:59.999	01jan1960 00:00:00.000	01jan1960 00:00:00.001
days	%td	31dec1959	01jan1960	02jan1960
weeks	%tw	1959w52	1960w1	1960w2
months	%tm	1959m12	1960m1	1960m2
quarters	%tq	1959q4	1960q1	1960q2
half-years	%th	1959h2	1960h1	1960h2

Inputting dates

Type	Store as	Str input	Num input
clock	double	clock(string, mask)	--see help--
days	float/long	date(string, mask)	mdy(M, D, Y)
weeks	float/int	weekly(string, mask)	yw(Y, W)
months	float/int	monthly(string, mask)	ym(Y, M)
quarters	float/int	quarterly(string, mask)	yq(Y, Q)
half-years	float/int	halfyearly(string, mask)	yh(Y, H)

Using masks in the date() function

Code	Meaning
M	month
D	day within month
Y	4-digit year
19Y	2-digit year to be interpreted
20Y	2-digit year to be interpreted
h	hour of day
m	minutes within hour
s	seconds within minute
#	ignore one element

For example:

```
. display %d date("20060125", "YMD")
25jan2006
. display %td date("060125", "20YMD")
25jan2006
```

Extracting dates

Function	Returns	Result if d = td(05jul1972) (i.e. d = 4,569)
year(d)	calendar year	1972
month(d)	calendar month	7
day(d)	day within month	5
doy(d)	day of year	187
halfyear(d)	half of year	2
quarter(d)	quarter	3
week(d)	week within year	27
dow(d)	day of week (0 = Sun)	3 (=Wed)

What you should have learned...

- Add and change variables (generate, replace)
- Be aware of the type of your variables
- Label your variables (label ...)
- String <-> Numeric (destring, real(), encode)
- Date formats and functions
- Commands: compress, rename, order, drop (keep)

Do files

Until now we have used the command line:

- great to develop but not to reproduce your analysis
- ALWAYS organize your work in Stata scripts

Stata scripts are called do-files after their extension (.do)

Use do-files (with informative names) to organize your work:

- create dataset
  crincome.do makes data file income.dta
- analysis
  andescr.do calculates my descriptive statistics
  anreg.do performs my regression analysis
- making graphs
  grwageplot.do makes the graph wageplot.eps

Note: do-files can call do-files.

- You can create a master do-file which calls the do-files which reproduce your complete preparation and analysis trail

Indexing

We have seen how to index observations:

- `x[i]` = the value of the i-th observation of x
- `x[_n]` = the value of the current observation of x
- `x[_N]` = the value of the last observation of x

We can for example easily

- take 1st lag of x
  ```
  gen l1x = x[_n - 1]
  ```
- reverse the order of x
  ```
  gen xreverse = x[_N - _n + 1]
  ```

Indexing

```
. clear

. set obs 5
obs was 0, now 5

. g x = _n

. l, clean

             x
  1.         1
  2.         2
  3.         3
  4.         4
  5.         5

. g xr = x[_N - _n + 1]

. g l2x = x[_n - 2]
(2 missing values generated)

. l, clean

             x          xr         l2x
  1.         1           5           .
  2.         2           4           .
  3.         3           3           1
  4.         4           2           2
  5.         5           1           3
```

By & indexing

```
. use country1

. by country year : g x = _n
not sorted
r(5);

. by country year, sort : g x = _n

. cl

        country        year           x
  1.        NOR        2001           1
  2.        NOR        2002           1
  3.        NOR        2003           1
  4.        NOR        2004           1
  5.        NLD        2001           1
  6.        NLD        2002           1
  7.        NLD        2002           2
  8.        NLD        2003           1
  9.        NLD        2004           1

. bysort country year : assert _N==1
1 contradiction in 8 by-groups
assertion is false
r(9);

// EXCERCISE: REMOVE THE SUPERFLUOUS OBSERVATION
// What about: duplicates tag country year, gen(dup)
```

Counting

Indexing is also useful for counting

```
. use auto, clear
. g manuf = word(make, 1)
. count if manuf=="Toyota"
    3

// but, how many manufacturers in data?

. codebook manuf
. sort manuf
. g x = manuf!=manuf[_n-1]
. cl manuf x
. g sx = sum(x)
. cl manuf x sx
. di sx[_N]

// one-liner: g sx = sum(manuf != manuf[_n - 1])


// data: id komnr income popsize
// average popsize per kommune?
```

Style

Space around operators

- `gen x = y + z`

Space after comma

- `gen fx = normalden(x, 0, 1)`

Indent (1 tab) after '{' and close at the level of the opening command

- ```
  if (_rc == 0) {
      di "Warning"
      exit
  }
  ```
  ```

Documenting - Comments

Use comments in your do-files when the code needs explaining or is better readable with a comment

- Single line comments:
  ```
  // comment here
  ```
- Multi line comments:
  ```
  /*
  [commented out]
  */
  ```
- Break lines:
  ```
  list pop19??  /// the rest of the line is commented out
      if country=="NOR"
  ```

Making tables

Estimation commands such as -regress- store results like coefficients and covariance matrices

- These can be used to make tables using Stata's -estimates-
- Make the following do-file

```
clear
set more off
u auto

// Main reg
   reg price mpg headroom trunk weight length
   estimates store price
// Domestic reg
   reg price mpg headroom trunk weight length if foreign==0
   estimates store price_dom
// Foreign reg
   reg price mpg headroom trunk weight length if foreign==1
   estimates store price_for
// Tabulate results
   estimates table *, b(%9.1f) se
```

Making tables (estout/esttab)

`estout` is a user contributed add-on with many options

- you should install such add-ons in a dedicated directory (named e.g. ado or stata)
- net set ado "PATH" (e.g. "M:\pc\Dokumenter\stata")
  - new add-ons (ado-files) are installed in this folder
- adopath + "PATH"
  - Stata will search for add-ons in this folder

Now type `findit estout` , scroll down and click through to install

```
. esttab *, se
```

What you should have learned...

- How to use do-files
- Save your results in a log file
- That you can use Stata's indexing to solve many data challenges
- Changing and combining datasets
- Importing ASCII data
- Make tables from estimation output