Stata – Session 2

Tarjei Havnes

[1]ESOP and Department of Economics
University of Oslo

[2]Research department
Statistics Norway

ECON 4136, UiO, 2012

Preparation

Before we start:

1. Go to kiosk.uio.no (Internet Explorer!) and log on using your UIO user name
2. Navigate to Analyse (english: Analysis)
3. Open StataIC 11
4. Follow the instructions on installing add-ons available on the course homepage
5. Command: - findit estout -
6. Install the estout-package (st0085_1)
7. Make sure you have downloaded the Wooldridge data

Learning goals

You should learn how to

1. use scalars, matrices and local macros
   - sca, sca list,
   - mat, mat list, mat colnames, [mat score, mat colstripe]
   - svmat, mkmat
   - local, macrolists
2. run regressions and access results
   - regress / ivregress, predict
   - predict
   - return(), ereturn(), _b(), _se(), [creturn()]
3. repeat the same command multiple times
   - forvalues, foreach, while
   - by, bysort
   - prog def(?)

Scalars and matrices

In addition to your data set, Stata can store

- scalars
- matrices
- macros (i.e. string scalars)

Scalars and matrices

In addition to your data set, Stata can store

- scalars
- matrices
- macros (i.e. string scalars)

You may create these yourself, they may be built-in, or a previous Stata command has created them for you.

Scalars and matrices

In addition to your data set, Stata can store

- scalars
- matrices
- macros (i.e. string scalars)

You may create these yourself, they may be built-in, or a previous Stata command has created them for you.

```
. use auto
(1978 Automobile Data)

. sca turn = turn[1]

. sca diameter = turn / _pi

. sca area = (diameter/2)^2 * _pi

. sca list
       area =  127.32395
   diameter =  12.732395
       turn =         40

// try using matrices to do this for all cars
```

Scalars and matrices

Matrices can also be collected from the data set

Scalars and matrices

Matrices can also be collected from the data set

```
. mkmat price mpg weight

. mat cons = J(rowsof(price),1,1) // (n X 1)-vector of ones

. mat X = cons , mpg , weight

. mat b = inv(X' * X) * X' * price

. mat list b

b[3,1]
              price
    c1    1946.0687
   mpg   -49.512221
weight    1.7465592

// Compare to - regress price mpg weight
```

Macros (local)

Stata refers to string scalars as macros.

- -local- sets string scalars that are valid until the end of the current run / session
- -local- is very powerful:
  - used for lists
  - used in loops
  - used to store all the arguments you call in a command

Again, we can input by hand, from the data set, use built-in locals, or a Stata-command may set locals for us.

Macros (local)

```
. local y  price mpg  weight

. su `y'

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906
         mpg |         74     21.2973    5.785503         12         41
      weight |         74    3019.459    777.1936       1760       4840

. local k : list sizeof y

. local dof = 74 - (`k' + 1)

. di "dof = " `dof'
dof = 70

. local date : di date("`c(current_date)'","DMY")

. di "`c(current_date)' is stored as `date'"
 6 Sep 2012 is stored as 19242
```

Looping

Looping is what programming is all about:

- Note: -gen lny = ln(y)- essentially loops over all observations to create lny for every observation in your data set
- You could do this manually, but it would be much slower than the built-in code

Looping

Looping is what programming is all about:

- Note: -gen lny = ln(y)- essentially loops over all observations to create lny for every observation in your data set
- You could do this manually, but it would be much slower than the built-in code

In Stata, local macros are used to specify what the loop replaces on separate runs.

- Recall that locals are called by adding apostrophes around the local-name

```
. sysuse auto, clear

. loc i = 1

. di "`i'. " _col(5) make[`i'] _col(20) turn[`i']
1.  AMC Concord    40
```

Looping

Stata allows you to loop

- over numbers

```
. forvalues i = 1/3 {
  2.          if `i' == 1 di _col(5) "make" _col(20) "turn"
  3.          di "`i'. " _col(5) make[`i'] _col(20) turn[`i']
  4. }
    make            turn
1.  AMC Concord     40
2.  AMC Pacer       40
3.  AMC Spirit      35

// Compare to -list make turn in 1/3-
// Note that -i = 1/3- is equivalent to the more general -i = 1(1)3-
```

Looping

Stata allows you to loop

- over lists of strings, numbers or ...

```
. cap mat drop b

. foreach x in mpg weight turn {
  2.          qui reg price `x'
  3.          qui su `x'
  4.          mat b = nullmat(b) , _b[`x']*r(Var)
  5. }

. mat colnames b = mpg weight turn

. mat list b
b[1,3]
            mpg       weight         turn
r1   -7996.2829    1234674.8    4017.5572

// Compare to -corr price mpg weight turn, cov-
```

Looping

Stata allows you to loop

- by the values of a variable (list)

```
. bysort foreign: su price
-------------------------------------------------------------------------------
-> foreign = Domestic

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       price |        52    6072.423    3097.104        3291      15906

-------------------------------------------------------------------------------
-> foreign = Foreign

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

// Note that -by varlist, sort:- is equivalent to -bysort varlist:-
// Look up -bysort varlist (varlist):-
```

Looping

Stata allows you to loop

- over numbers `-forvalues i=numlist {-`
- over lists `-foreach a in list {-`
- by the values of a variable (list) `-bysort varlist:-`
- over specialized lists:
  - `-foreach x of varlist p* {-`
  - `-foreach i of numlist 1(1)3 {-`
  - `-foreach c of local varlist {-`
- while a statement is true `-while statement {-`

```
. loc i = 1

. while `i' < 4 {
  2.          if `i' == 1 di _col(5) "make" _col(20) "turn"
  3.          di "`i'. " _col(5) make[`i'] _col(20) turn[`i']
  4.          loc i = `i' + 1
  5. }
     make            turn
1.   AMC Concord     40
2.   AMC Pacer       40
3.   AMC Spirit      35
```

Accessing results

When we run estimation commands, Stata saves scalars, matrices and locals in -ereturn()-

```
. regress price weight mpg
[... Output omitted]

. ereturn list

scalars:
                  e(N) =  74
               e(df_m) =  2
               e(df_r) =  71
                  e(F) =  14.7398153853841
                 e(r2) =  .2933891231947529
               e(rmse) =  2514.028573297152
                e(mss) =  186321279.739451
                e(rss) =  448744116.3821706
               e(r2_a) =  .27348459145376
                 e(ll) =  -682.8636883111165
               e(ll_0) =  -695.7128688987767
               e(rank) =  3
macros:
            e(cmdline) : "regress price weight mpg"
              e(title) : "Linear regression"
          e(marginsok) : "XB default"
                e(vce) : "ols"
            e(depvar) : "price"
                e(cmd) : "regress"
         e(properties) : "b V"
            e(predict) : "regres_p"
              e(model) : "ols"
          e(estat_cmd) : "regress_estat"
```

Accessing results

Other commands often save in -return()-

```
. su price weight mpg
[... Output omitted]

. return list

scalars:
                  r(N) =  74
              r(sum_w) =  74
               r(mean) =  21.2972972972973
                r(Var) =  33.47204738985561
                 r(sd) =  5.785503209735141
                r(min) =  12
                r(max) =  41
                r(sum) =  1576
```

Accessing results

We can work with these results by calling them as scalars or locals

```
. su price
[... Output omitted]

. di "Mean = " _col(8) %14.2f `r(mean)' _n "SD = " _col(8) %14.2f sqrt( r(Var) )
Mean =         6165.26
SD =           2949.50
```

Accessing results

We can work with these results by calling them as scalars or locals

```
. su price
[... Output omitted]

. di "Mean = " _col(8) %14.2f `r(mean)' _n "SD = " _col(8) %14.2f sqrt( r(Var) )
Mean =         6165.26
SD =           2949.50
```

When we have estimation results in memory, there is a dedicated syntax for accessing coefficients and standard errors

```
. regress price weight mpg
[... Output omitted]

. di "t(weight) = " _b[weight] / _se[weight]
t(weight) = 2.7232382
```

Basic regressions

In the dataset -bwght.dta-, consider the model

$$\ln bwght = \beta_0 + \beta_1 male + \beta_2 parity + \beta_3 \ln faminc + \beta_4 packs + u$$

- estimate the model using the OLS-command - regress -
- tabulate the results using - estimates table -, then using - esttab -
- use the stored coefficients and SE's to test $H_0 : \beta_1 = 0$

Basic regressions

In the dataset -bwght.dta-, consider the model

$$\ln bwght = \beta_0 + \beta_1 male + \beta_2 parity + \beta_3 \ln faminc + \beta_4 packs + u$$

- estimate the model using the OLS-command - regress -
- tabulate the results using - estimates table -, then using - esttab -
- use the stored coefficients and SE's to test $H_0 : \beta_1 = 0$

```
. regress lbwght male parity lfaminc packs
[... Output omitted]

. est table, b(%14.4f) se stats(r2 N)

-------------------------------
    Variable |      active
-------------+-----------------
        male |          0.0262
             |          0.0101
      parity |          0.0147
             |          0.0057
     lfaminc |          0.0180
             |          0.0056
       packs |         -0.0837
             |          0.0171
       _cons |          4.6756
             |          0.0219
-------------+-----------------
           N |            1388
-------------------------------
                  legend: b/se
```

Basic regressions

```
. esttab, se wide

--------------------------------------------
                      (1)
                   lbwght
--------------------------------------------
male                0.0262**       (0.0101)
parity              0.0147**      (0.00566)
lfaminc             0.0180**      (0.00558)
packs              -0.0837***      (0.0171)
_cons               4.676***       (0.0219)
--------------------------------------------
N                      1388
--------------------------------------------
Standard errors in parentheses
* p<0.05, ** p<0.01, *** p<0.001

. di "p-value (b1=0) = " %14.2f ttail(e(df_m),_b[male]/_se[male])
p-value (b1=0) =           0.03
```

Basic regressions

We now use -cigprice- as an instrument for -packs-.

1. Estimate the first stage and the reduced form of this model, i.e.

$$\ln bwght = \pi_0^r + \pi_1^r male + \pi_2^r parity + \pi_3^r \ln faminc + \pi_4^r cigprice + u$$
$$packs = \pi_0 + \pi_1 male + \pi_2 parity + \pi_3 \ln faminc + \pi_z cigprice + v$$

2. Calculate the IV-estimate of β_4 by using the fact that $\beta_4^{IV} = \pi_4^r / \pi_z$.
3. Estimate the IV-model by predicting -packs- from the FS, and then running the original model with this variable in place of -packs- .
4. Estimate the IV-model using -ivreg-
5. Tabulate the models (OLS, IV, RF, FS) using -esttab-
   - Note that you can store regression results by using -estimates store-
   - (Alternatively, use -eststo- from the estout-package)

Basic regressions

```
. eststo ols

. qui regress lbwght male parity lfaminc cigprice

. sca rf = _b[cigprice]

. eststo rf

. qui regress packs male parity lfaminc cigprice

. di "b4(IV) = " %14.3f rf/_b[cigprice]
b4(IV) =          0.797

. predict pr_packs
(option xb assumed; fitted values)

. eststo fs

. qui regress lbwght male parity lfaminc pr_packs

. eststo IV1
```

Basic regressions

```
. ivreg lbwght (packs=cigprice) male parity lfaminc

Instrumental variables (2SLS) regression
      Source |       SS       df       MS              Number of obs =    1388
-------------+------------------------------           F(  4,  1383) =    2.39
       Model |  -91.350027     4 -22.8375067           Prob > F      =  0.0490
    Residual |  141.770361  1383  .102509299           R-squared     =       .
-------------+------------------------------           Adj R-squared =       .
       Total |  50.4203336  1387  .036352079           Root MSE      =  .32017
------------------------------------------------------------------------------
      lbwght |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       packs |   .7971063   1.086275     0.73   0.463    -1.333819    2.928031
        male |   .0298205    .017779     1.68   0.094    -.0050562    .0646972
      parity |  -.0012391   .0219322    -0.06   0.955     -.044263    .0417848
     lfaminc |    .063646   .0570128     1.12   0.264    -.0481949    .1754869
       _cons |   4.467861   .2588289    17.26   0.000     3.960122    4.975601
------------------------------------------------------------------------------
Instrumented:  packs
Instruments:   male parity lfaminc cigprice
------------------------------------------------------------------------------

. eststo iv
```

Basic regressions

```
. esttab ols rf fs IV1 IV2, mtit
--------------------------------------------------------------------------------
                      (1)             (2)             (3)             (4)             (5)
                      ols              rf              fs             IV1             IV2
--------------------------------------------------------------------------------
male                0.0262**        0.0261*       -0.00473         0.0298**          0.0298
                    (2.60)          (2.56)         (-0.30)          (2.84)          (1.68)
parity              0.0147**        0.0132*         0.0181*       -0.00124        -0.00124
                    (2.60)          (2.32)          (2.04)         (-0.10)         (-0.06)
lfaminc             0.0180**        0.0217***     -0.0526***         0.0636          0.0636
                    (3.23)          (3.88)         (-6.05)          (1.89)          (1.12)
packs              -0.0837***                                                       0.797
                   (-4.89)                                                         (0.73)
cigprice                            0.000619        0.000777
                                      (1.24)          (1.00)
pr_packs                                                              0.797
                                                                     (1.24)
_cons                4.676***        4.577***          0.137         4.468***        4.468***
                   (213.68)         (68.55)         (1.32)         (29.23)         (17.26)
--------------------------------------------------------------------------------
N                      1388            1388            1388            1388            1388
--------------------------------------------------------------------------------
t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001
```