STATA COMMANDS USEFUL FOR DATA CLEANING IN SHARE

Dimitris Christelis
SHARE and CSEF, University of Naples Federico II

SHARE Berlin Meeting, June 8, 2009
MENTALITY

- Assume that everything that can go wrong will go wrong (instrument problems, interviewer screw-ups, missing id’s, duplicate observations, wrong interviews, wrong panel identifiers, wrong coding of answers (e.g. height equal to 6)). Check your happy personality out of the door!

- 99% accuracy is unfortunately not good enough for our purposes. It’s OK for writing a paper, but not for present the data to the users. Users will complain even about 2-3 cases, they will start distrusting the data, they will feel compelled to make additional checks that they originally did not intend to do → SHARE will start getting a bad reputation

- Do not make unnecessary assumptions about accuracy of variables. Check, check, check!

- Document your work (e.g. put the sequence of programs in a document, so that someone can go from the raw input data to your output)

- Have a second person check as much as possible. Mistakes are very likely
**ASSERT command**

- Most useful STATA command for data cleaning
- Confirms that things are the way you think they are
- Unforgiving; any contradiction (even for 1 obs) stops the program
- Therefore: frustrating, but ultimately saves a lot of time in the long run
**ASSERT examples**

- Check that no obs is missing

  ```
  assert var1 != .
  assert var1 != ""
  ```

- Check that the values of the variable are lower than a minimum and higher than a maximum

  ```
  sca tolr = 1e-6
  assert var1 > varmin - tolr & var1 < varmax + tolr
  assert var1 >= varmin & var1 <= varmax
  ```

- Check that a variable takes only some values (e.g. self-reported health)

  ```
  assert var1 == 1 | var1 == 2 | var1 == 3 | var1 == 4 | var1 == 5
  ```

- Other examples where `assert` can prove useful:

  - check that every person who reports to be married/in a partnership has a partner who also appears at the least in the coverscreen
EGEN command, variable TYPES

• Can be very useful to transfer information from one obs (i.e. line in your file) to others

• Example: try to find the id of the partner of the person with cvid==1, and put it in that person’s line

```
destring person2,g(numperson)       /* Make person2 numerical */
gen d1=cvidp if cvid==1             /* Create a variable equal to cvidp for obs with cvid==1, missing for all others */
egen sd1=max(d1),by(implicat sampid2)  /* put the cvidp info in the line of every person in the household */
gen double d2=numperson if cvid==sd1 /* create a variable equal to the id, for the obs whose id matches the cvidp */
egen double sd2=max(d2),by(implicat sampid2)  /* put the id info in the line of all household members */
gen double idpartn = sd2 if cvid==1  /* create a variable equal to the id of the partner for the obs for which cvid==1 */
tostring idpartn,g(idpartn_str)     /* Make the new id variable a string one */
drop d1 sd1 d2 sd2
```
• Importance of variable types: without the **double** option the d2, sd2, idpartn variables would get wrong values

• Variable types

  o **byte**, with range: -127 to 100
  o **int**, with range: -32,767 to 32,740
  o **long**, with range: -2,147,483,647 to 2,147,483,620
  o **float**, with range: -1.70141173319*10^38 to 1.70141173319*10^36
  o **double**, with range: -8.9884656743*10^307 to 8.9884656743*10^307

• When creating a 0-1 dummy, one can use the **byte** option, but be ware of 998, 999 in SHARE datasets

• In general, one can say nothing about dummies, use **double** for large numbers (numerical id’s, financial amounts). When saving at then end, command **compress**
SET TRACE ON/OFF statement

- Shows you in detail how your commands are executed

```stata
forvalues k=1/2 {
    if `k'==1 {
        gen var`k' =0
    }
    if `k'!=1 {
        gen var`k' = 1
    }
}

set trace on
forvalues k=1/2 {
    assert var`k'==0
}

- forvalues k=1/2 {
    - assert var`k'==0
    = assert var1==0
    - }
```
- assert var`k'==0
  = assert var2==0
  16605 contradictions in 16605 observations
  assertion is false
  }
  r(9);

- Turn trace off before complicated commands

  set trace on
  data work commands (gen , replace )
  set trace off

  probit var1 var2 var3

  set trace on
  continue with data work
  set trace off
MERGE command

• When using the **update** option, **missing** values in the master datasets are replaced by non-missing values in the using datasets.

• When using the **replace** option, **any** values in the master datasets are replaced by non-missing values in the using datasets.

• If the update option is specified, the variable _merge takes the value 4 if the value in the master data is missing, and the value 5 if the value is not missing.

• Remember to drop/rename the _merge variable, otherwise the next merge statement will not be executed.

• When merging, try to put in the using files only variable that do not exist in the master file, or that you want to be updated from the using file. Use keep option

merge var1 var2 using temp1.dta, sort keep(var3new var4new var5updated)
**BYSORT command**

- Useful when defining units (the variables in the bysort option), and then isolating obs within the unit

```
bysort implicat sampid2: gen d1=(_n==1)
```

```
ta d1
```
```
  d1  |  Freq.  Percent  Cum.  
  ------------------------+---------------------
     0 |  7,125  42.91    42.91
     1 |  9,480  57.09   100.00
  ------------------------+
        Total | 16,605  100.00
```
```
ta d1 if implicat==1

<table>
<thead>
<tr>
<th>d1</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1,425</td>
<td>42.91</td>
<td>42.91</td>
</tr>
<tr>
<td>1</td>
<td>1,896</td>
<td>57.09</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Total | 3,321 | 100.00

.bysort sampid2: gen d1=_n==1

.ta d1

<table>
<thead>
<tr>
<th>d1</th>
<th>Freq.</th>
<th>Percent</th>
<th>Cum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14,709</td>
<td>88.58</td>
<td>88.58</td>
</tr>
<tr>
<td>1</td>
<td>1,896</td>
<td>11.42</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Total | 16,605 | 100.00
```
IF statement

- Syntax:

```plaintext
if .......... {
    execute command
}

if r(mean)>10  {
    gen d1=r(mean)^10
}
```

- Loop examples:

```plaintext
foreach k in var1 var2 var3 {
    foreach m in var4 var2 var6 {
        if "\k"=="\m" {
            gen d1=\k^2
        }
    }
}
```
forvalues k=1/5 {
    if `k'==1 {
        replace var`k' = var`k'/10
    }
    if `k'!=1 {
        replace var`k' = var`k'/20
    }
}

• Ordered probit example:

    oprobit var1 var2 var3 var4 var5
    eret list

    if e(converged)==0 {
        oprobit var1 var2 var3
    }

• Careful when using with variables: if will look only at the first line of the file and execute depending only on what happens with that line.
Multiple imputation commands

- Use package `mim`

  ```
  gen _mj = implicat /* Create indicator for implicates */
  destring person2,g(numperson)
  gen double _mi=numperson /* Create indicator for persons */
  mim: probit var1 var2 var3 /* Probit regression */
  mim: testparm var2 /* Test for the parameter of var2 being equal to 0 */
  ```
Stata versions

- Not everybody has Stata 10!

- In Stata 10, use `saveold` command to save in Stata 8, Stata 9 formats

- One can also use the option `version X.X` before a command:

  `version 9.2: graph`