Introduction to Stata- A. Chevalier

Lecture 2: Advanced data manipulation

Content of Lecture 2:

-creating data
-using dates
-merging and appending datasets
- wide and long
-collapse

A] Creating data

As mentioned in Lecture 1, the most useful commands to create variables are the following:
generate, replace, egen, rename, drop, keep, sort, encode, reshape

You should already have had a look at the help file for them, and I will only stress some specific features here.

* generate

You want to generate the ratio of weight per length, and a dummy when this ratio is greater than 16:

```
. generate lbperin =weight/length
. generate dense=0
. replace dense =1 if lbperin>16
(38 real changes made)
```

Alternatively, I could have typed instead of the generate and replace
```
. gen dense2=lbperin>16
```

This feature of generate to create dummy variables is quite useful, the logic of it can be summarised as, if the condition is true, new variable equals 1, otherwise, new variable equals 0.

Attention: be careful when missing observations are present in your dataset.
Missing variables are interpreted as a very large positive number, hence depending on your inequality, the condition will always be true (or untrue) for missing observations. It is always wise to use a recode command, to make sure that no missing observations has been coded in your dummy variable.
```
recode dense2 *=. If lbperin==.
```

We can check that dense and dense2 are the same.
```
. compare dense dense2
```

```
                             ---------- difference ----------
                        count       minimum      average     maximum
-----------------------------------------------------------------------
dense=dense2               74
jointly defined            74             0            0           0
total                      74
```

The best solution is to do all this in one step only:

```
gen dense3 = lbperin>16 if lbperin~=.
```

If you want to generate string variables or create a variable not in float format, you have to specify the format.

```
drop dense
generate byte dense = lbperin>16

generate str18 newmake=make
```

* Dealing with String variables

There are quite a few commands that deal specifically with string variables.

First we are looking at ways of extracting part of a string variable: substr.
Let's say, we want to separate the manufacturer and the model from newmake. The manufacturer is the first part of the string and the model is the second one, after the space.
Substr allows us to do just that, in fact as this is quite complicated, as the various strings that we want to extract are of different length. Let's start with a simpler example, to check how substr works.

Say that we have a string variable (sentence) that contains the following sentence "hello there" and that we want to split the two components.

```
gen hello=substr (sentence, 1,5)
```

substr needs to be provided with a set of 3 information:

- which string you want to split
- what is the position of the first character of the splitted string
- how many characters does the splitted string include.

Here, we created a new string hello, which is part of sentence, it starts at the first character and is 5 characters long. Be careful, blank characters at the beginning of the string counts as characters. It is possible to eliminate them (using trim).

To go back to our example, the difficulty comes from the fact that we do not know the length of the first string, it varies with the name of the constructor….

```
List newmake in 1/5
                newmake
  1.        AMC Concord
  2.          AMC Pacer
  3.         AMC Spirit
  4.      Buick Century
  5.      Buick Electra
```

To isolate the make, we are going to rely on another stata command (as you can see, they can sometimes be combined): index

From the help file, we know that:
index(s1,s2) returns the position in s1 in which s2 is first found or 0 if s1 does not contain s2.

Comment: Have a look at the string functions to find out the best way to manipulate strings.

So basically, what we want to do it to find the position of the blank that separates the make from the model in newmake, and then cut the make string at the previous character, this is done by:

```
gen str18 manuf=substr(newmake, 1,index(newmake," ")-1)
(1 missing value generated)
```

You have to let stata know that manuf is going to be a string variable, length maxi 18 characters….
Comments: Do not despair of manipulating strings, just break the problem into its basic components.

However, something went wrong somewhere, since one missing value was generated. To find out the culprit:

```
. list make if manuf==""

     make
 66. Subaru
```

Our syntax was based on finding the blank character between the make and model, obviously in the case of Subaru, where the model is absent, it did not work. However, it is easy to correct this;

```
. replace manuf = make in 66
(1 real change made)
```

We have now created the manuf variable, but this is a string and it cannot be used easily in most of the analysis, we want to make. What about creating a new variable which will contain the same information as manuf but in a real format?

There are three ways of doing this; we are going to review them now, from the most tedious to the least. However, the most tedious one, has the interesting feature of exploiting some new and particularly useful commands (by and _n)….

Basically, we want to group the data by manuf, and create a unique identifier for each manuf:

Method 1:

```
. sort manuf

. by manuf: gen manuf1=1 if _n==1
(51 missing values generated)
```

by is a useful command, it allows you to repeat the command for each group of observations for which the values of the variables in varlist are the same.

_n and _N are also very useful for data manipulation, especially when combined with by.

_N is the total number of observations (when used with by, it is therefore the total number of observations within a group of observations for which the value of the varlist is the same). _N is the total number within each by group.

_n is a unique internal identifier, starting at 1 and going up to _N, when combined with by, the identifier is unique within the group.
_n is a running counter within each by group.

We should have another example on the use of *_n and* _N later.

Hence, our last command, created a flag variable taking the value 1 for the first observation of the group, as defined by manuf.

```
. list manuf manuf1 in 1/8

                  manuf     manuf1
  1.                AMC          1
  2.                AMC          .
  3.                AMC          .
  4.               Audi          1
  5.               Audi          .
  6.                BMW          1
  7.              Buick          1
  8.              Buick          .
```

We now have a flag for the first observation of each group, to get to what we want, it is now time to have a look at the sum function;

sum(x) returns the running sum of x, treating missing values as zero.

```
. replace manuf1=sum(manuf1)
(73 real changes made)

. list manuf manuf1 in 1/8

                  manuf     manuf1
  1.                AMC          1
  2.                AMC          1
  3.                AMC          1
  4.               Audi          2
  5.               Audi          2
  6.                BMW          3
  7.              Buick          4
  8.              Buick          4
```

Method 2:

egen is a generic command, that allows you to generate variables according to some specific functions. Have a look at the help file for egen. Egen tends to make life easier in a multitude of problems.

In our case of creating a group identifier, we could have simply typed:

```
. egen manuf2=group(manuf)

. list manuf manuf2 in 1/8

                manuf     manuf2
  1.              AMC          1
  2.              AMC          1
  3.              AMC          1
  4.             Audi          2
  5.             Audi          2
  6.              BMW          3
  7.            Buick          4
  8.            Buick          4
```

This gives us the same results as our previous laborious work, this is not a surprise since egen with the group function does exactly what the three lines of instruction that we previously typed. This has also the advantaged of skipping a group for which manuf would be missing.
These two approaches are OK, but they both suffer from a drawback that the label corresponding to the created groups have not been reported. It will be possible, to insert them ourselves (as seen in Lecture 1). However, there is a more efficient way of doing it.

* Method 3:

This is the simplest approach, as this command is a program that just does what we have previously been typing, and make sure that the labels are created. Encode reads string variables and with the generate option, create a numeric equivalent to the original string variable.

```
. encode manuf, gen(manuf3)

. tab manuf3

     manuf3 |      Freq.     Percent        Cum.
------------+-----------------------------------
        AMC |          3        4.05        4.05
       Audi |          2        2.70        6.76
        BMW |          1        1.35        8.11
      Buick |          7        9.46       17.57
       Cad. |          3        4.05       21.62
      Chev. |          6        8.11       29.73
     Datsun |          4        5.41       35.14
```

```
      Dodge |          4        5.41       40.54
       Fiat |          1        1.35       41.89
       Ford |          2        2.70       44.59
      Honda |          2        2.70       47.30
      Linc. |          3        4.05       51.35
      Mazda |          1        1.35       52.70
      Merc. |          6        8.11       60.81
       Olds |          7        9.46       70.27
    Peugeot |          1        1.35       71.62
      Plym. |          5        6.76       78.38
      Pont. |          6        8.11       86.49
    Renault |          1        1.35       87.84
     Subaru |          1        1.35       89.19
     Toyota |          3        4.05       93.24
         VW |          4        5.41       98.65
      Volvo |          1        1.35      100.00
------------+-----------------------------------
      Total |         74      100.00
```

There is also a decode command that does exactly the opposite operation (from a labelled real variable to a string).

* remark on memory
Say that we are happy now with our auto datasets, however, before saving it, we could delete some of the variables we created today that are not of much use now.

```
. drop  dense2 newmake manuf manuf1 manuf2
```

that leaves us with an extra three variables compared to when we started. These variables have been created in formats that are memory hungry. A useful command to prevent the size of your dataset to get our of control (and slow stata a great deal) is:

```
. compress
mpg was int now byte
rep78 was int now byte
trunk was int now byte
turn was int now byte
manuf3 was long now byte
make was str18 now str17
```

Compress changes the format of your variables to the most effective format.
You can now save your dataset.

When opening large datasets, you will end up with a message like:

```
. use "D:\Data\ECPH2\data\w2pers.dta", clear
no room to add more observations
r(901);
```

Do not despair, it is that stata default memory set up is not adequate for your data. This can be changed by:

set mem xxxm

where xxx is the amount of memory you are allocating to stata. This is not limited by stata but the amount of RAM and your OS on your computer. Do not allocate more memory than 90% of your RAM to stata, this will slooow things down enormously, as the hard drive will be used to provide some memory. As a rule, set your memory to 130% of your dataset, unless you are planning to create a large number of variables. This should be OK. Also remember to delete unneeded variables and to compress your file.

* Back to the use of _n and _N
As said previously, _n is a running counter within the by group and _N is the total number of observations in your by group.

Let say that we have a dataset containing family id and sibling's age and that we want to create a variable on sibling order for each family.

The dataset looks like: (clear your data, and paste this data in your editor)

famid	age
23	4
23	7
23	11
27	8
27	15
48	2
48	3
48	5
48	7

So we want to generate the number of sibling within a family and then the birth order as a function of age.

```
. sort famid

. by famid: gen sibs= _N

. sort famid age

. by famid: gen b_order=sibs -_n + 1
```

Here is what the outcome looks like:

		by famid:			
famid	**age**	**_n**	**_N**	**sibs**	**B_order**
23	4	1	3	3	3
23	7	2	3	3	2
23	11	3	3	3	1
27	8	1	2	2	2
27	15	2	2	2	1
48	2	1	4	4	4
48	3	2	4	4	3
48	5	3	4	4	2
48	7	4	4	4	1

Finally, _n can be used to create lag values:

Gen y=y[_n-1]

B] using dates

Dates can be written in a variety of format in real life: 18 September 2001, Sept 18, 2001, 18/09/01 and plenty of others.
Stata stores dates in a single format so there exist some function to transform dates into stata dates. (help date). In order to be able to compute calculation on dates, stata stores them as a real variable. The first of January 1960 is set as 1, which means that dates can be negative. The advantage of this format, is that it makes it easy to calculate elapsed time between two dates, but is confusing to anyone but a computer. It is nevertheless possible to change the display format of dates so that you can get a grip on what is happening.

Copy the following variable in your editor:

date
12869
12876
-2757
13044
14610

you would like to know which dates these numbers correspond to;

```
format date %d

. list

         date
  1. 27mar1995
  2. 03apr1995
  3. 14jun1952
  4. 18sep1995
  5. 01jan2000

. format date %dM_d,_CY

. list

                  date
  1.     March 27, 1995
  2.      April 3, 1995
  3.      June 14, 1952
  4. September 18, 1995
  5.    January 1, 2000
```

Most of the time, date variables will come in a standard format in your dataset. The following dataset contains an identifier and a date of birth, open it using the infile command.

Begin hosp1.raw

05721 01/21/1952
10322 07/11/1948
51331 11/15/1968
end hosp1.raw

First, copy the data in an editor (notepad) and save as a raw file in your current stata directory (see lecture 1).

```
. clear
. infile patid str20 bdate using hosp1
(3 observations read)

. list

        patid                 bdate
  1.     5721            01/21/1952
  2.    10322            07/11/1948
  3.    51331            11/15/1968

. des

Contains data
  obs:             3
 vars:             2
 size:            84 (100.0% of memory free)
-------------------------------------------------------------
----------------------
              storage  display     value
variable name   type   format      label      variable
label
-------------------------------------------------------------
----------------------
patid           float  %9.0g
bdate           str20  %20s
-------------------------------------------------------------
----------------------
Sorted by:
     Note:  dataset has changed since last saved
```

Although bdate looks Ok, it is a string variable and you cannot do any calculation using it. It needs to be transformed into a stata dates using the date function.

```
. gen bdate2 =date(bdate,"md19y")

. list

        patid                 bdate     bdate2
  1.     5721            01/21/1952      -2902
  2.    10322            07/11/1948      -4191
  3.    51331            11/15/1968       3241
```

Bdate2 is now in stata date format which is incomprehensible, so we can transform the display format.

```
. format bdate2 %d

. list

        patid                 bdate     bdate2
  1.     5721            01/21/1952  21jan1952
  2.    10322            07/11/1948  11jul1948
  3.    51331            11/15/1968  15nov1968
```

C] Merging and appending datasets

These two commands allow you to combine datasets. Append is the easiest one and allows you to add observations (rows) while merge adds variables (columns). Append does not require that the "master" and "using" datasets contain the same number of variables, even so this is typically the case.
see help append

For merging datasets, the requirements are a bit more stringent. The datasets need to have an identifier (or a set of variables) that allow you to uniquely define observations (in case of a one to one match) or group of observations (group match).

Let's say that you have two datasets;

one.dta

id	a	b
1	10	11
2	12	13
3	14	15
5	16	17

two.dta

id	c
1	18
3	19
4	20

That you want to merge by id

Let say that the two files have been sorted by id and have been saved on your directory. One is your "master" file,

use one
merge id using two

Merge.dta

id	a	b	c	_merge
1	10	11	18	3
2	12	13		1
3	14	15	19	3
4			20	2
5	16	17		1

So what exactly happens here? The data have been merged by id, that is, the new variable c has been merged into the "master" dataset for each observation. A variable _merge has also been created which allows you to check the result of the merge for each observation.
3 means everything went OK, the id was observed in both datasets hence for this observation you now have 3 observed variables.
1 means that the id from the master dataset was not available in the second dataset, so for these observations no change has been made compared to the master file. The new variable is missing for these observations.
2 means that the id from the "using" file was not found in the master file. For these observations, the "master" variables are missing and the "using" variables remain what they originally were.

Comment: ***It is of crucial importance to check _merge after merge****, as not everything always goes to plan. In case of unsolvable problems, check that the identifiers are unique.*

This can be done easily

```
use one,clear
sort id
quietly by id: assert _N=1
```

Assert check the statement made and return nothing if statement correct and an error message if statement incorrect. Here the statement is that each group defined by id should contain only 1 member (remember what _N and by do for you).

```
. assert 2+2==5
assertion is false
r(9);
```

I also like using;

```
gen duplicate=id[_n]==id[_n-1]
```

which flag observations for which the statement is right (remember how to create a dummy)

Now you can copy the two original files in the stata editor, and practice with the merge command. Check the quality of your merge, by looking at the editor, list _merge and tab _merge

D] Wide versus long

I'm only referring to the way your dataset is organised. So first I will explain what I mean by these expressions:

Wide format

id	sex	Inc90	Inc91
1	1	5000	5500
2	0	2000	4000
3	0	3000	3200

Long Format:

id	year	sex	Inc
1	90	1	5000
1	91	1	5500
2	90	0	2000
2	91	0	4000
3	90	0	3000
3	91	0	3200

There is no specific advantage of having the data in one format rather than the other (even so long usually provides you with more commands), it all depends on what you want to do.

Say that you want to calculate the income growth, then it is easier to have the data in wide format:

```
gen growth=(inc91-inc90)/inc90
```

Say you want to study income by sex, then you need to have the data in long format

```
xtreg inc sex, i(id) be
```

As there is no specific information that is in one dataset and not in the other, it is easy to go from one form to the other. In most cases you will want to reshape when dealing with time varying variable, so I will keep my explanation with this set-up. However, reshape is not limited to this case. You can use it, with household data, when it is sometime easier to have all household members on one observation and sometime each one of them on one line.

It is important to differentiate between the different types of variables that we have here.
id is your identification variable, it is needed to make sure that income from different years stay associated to the right person.
Year is needed to find out, when time varying variables occurred.
Fixed effect variables: Variables that do not change over time, in our example sex
Time specific variables: Variables changing over time.

To go from wide to long:

```
reshape long inc, i(id) j(year)
```

The variable year does not exist nor does the variable inc. Inc is only the common root of the time varying variable, and year is the name you want to give to the variable measuring time. Note that sex, the fixed effect variable is not specified, but all time varying variables must be specified. If a time varying variable is not specified, then the command will return an error message. If this happen, you can get a report on where the error occur by typing `reshape error`.
i identifies a logical observation
j identifies the name of the grouping variable

To go from long to wide:
```
reshape wide inc, i(id) j(year)
```

Comment: when reshaping from long to wide, make sure that the root of the time varying variable is small enough so that the year identifier can be added without the variable name getting larger than 8 characters.

If you copy the wide dataset on your editor, you can practice with the reshape function.
Now, when the data is in long format, change the value of sex from 0 to 1 for the second occurrence of observation 3. Try reshaping the data to wide format. Use `reshape error`. to find out where the problem is.

E] Collapse

collapse converts the data in memory into a dataset of means, sums, medians, etc.
This is extremely useful when you want to provide summary information at a higher level of aggregation that what is available in your data. For example, you have data at the individual level and what to aggregate it at the firm level.

Be careful, collapse changes your data drastically. All individual information disappear and only the specified variables remain at the group level.
For example, in the auto dataset we had 74 cars. We previously created manuf and found that we had data on 23 manufacturers. If we are interested in the average price of cars for each manufacturer:

```
collapse price, by(manuf)
```

We are now left with 23 observations (manufacturers) and one observation: the mean range price.