Useful Stata Commands for Longitudinal Data Analysis

Josef Brüderl

Volker Ludwig

University of Munich

May 2012

Nuts and Bolts I

First some „Nuts and Bolts“ about data preparation with Stata.

```
Mathematical and Logical Expressions
+    add          ~ [!]  not    <      less than             ln()   natural log
-    subtract     &      and    <=     less than or equal    exp()  exponential
/    divide       |      or     >      greater than          sqrt() square root
*    multiply                   ==      equal                abs()  absolute
^    power                      ~= [!=] not equal
```

```
RECODE
recode varname 1 3/5=7                //1 and 3 through 5 changed to 7

recode varname 2=1 .=. *=0            //2 changed to 1, all else is 0, . stays .
recode varname (2=1 yes) (nonmiss=0 no) //the same including labels, () needed

recode varname 5/max=max              //5 through maximum changed to maximum (. stays .)
recode varname 1/2=1.5 2/3=2.5        //2 is changed to 1.5
recode varlist (2=1)(nonmiss=0)       //you need () if recoding a varlist
```

```
Creating a Dummy
recode   varname (2=1 yes) (nonmiss=0 no), into(dummy) //elegant solution I
generate dummy = varname==2 if varname<.               //elegant solution II
tab varname, gen(dummy)                                //most simple but boring
```

Nuts and Bolts II

Comments

```
* ignore the complete line                 //  ignore the rest excluding line break
/* ignore the text in between */           /// ignore the rest including line break
```

Be careful with missing values:

. == +∞, this might produce unwanted results. For instance, if you want to group a variable X, this is what you get

```
gen Xgrouped = X>2

     | X    Xgrouped |
     |---------------|
  1. | 3           1 |
  2. | 2           0 |
  3. | .           1 |
  4. | 1           0 |
  5. | 4           1 |
     +---------------+
* better:
gen Xgrouped = X>2  if X<.

N.B.: . < .a < .b < ...
N.B.: X==. is true only if .
      missing(X) is true for
        all missing values
```

Data in wide-format: counting values in varlists

```
. egen numb1   = anycount(var1-var3), v(1)
. egen numbmis = rowmiss(var1-var3)
. list var1 var2 var3  numb1 numbmis

     +---------------------------------------+
     | var1   var2   var3   numb1   numbmis |
     |---------------------------------------|
  1. |    1      0      .       1         1 |
  2. |    1      0      0       1         0 |
  3. |    1      1      0       2         0 |
  4. |    1      1      1       3         0 |
     +---------------------------------------+
```

Further example: number of valid episodes

```
egen nepi = rownonmiss(ts*)
```

Further example: max in “time finish”

```
egen maxage = rowmax(tf*)
```


Nuts and Bolts III

Missing Values

```
misstable summarize          //gives overview of MV in the data

misstable patterns           //MV patterns in the data

mvdecode _all, mv(-1)        //-1 is set to . in all variables

mark     nomiss              //generates markervariable “nomiss”
markout  nomiss Y X1 X2 X3   //0=somewhere missing, 1=nowhere missing
drop if  nomiss == 0         //listwise deletion
```

Value-Label

```
label define geschlbl 1 "Man" 2 “Woman"
label value  sex  geschlbl
```

Display a Scalar

```
display  5*8
```

Regression Coefficients

```
regress, coeflegend     //shows names of coefficients
display _b[bild]        //displays a coefficient
```

Formating Output (permanent!)

```
set cformat %9.4f, permanently        //format of coeff, S.E, C.I.
set pformat %5.3f, permanently        //format of p-value
set showbaselevels on, permanently    //display reference category
```

Nuts and Bolts IV

IF-Command

```
if expression {
  commands            //commands are executed if expression is true
}
```

GLOBAl Macros

```
* Directory where the data are stored
global pfad1 `""I:\Daten\SOEP Analysen\Zufriedenheit\Fullsample\""'
* Load data
cd $pfad1              //$pfad1 is expanded to "I:\Daten\..."
use Happiness, clear
```

Working with date functions

```
* Date information is transformed in "elapsed months since Jan. 1960"
gen    birth = ym(birthy,birthm)   //mdy(M,D,Y) if you have also days
gen    birthc=birth
format birthc %tm                  //%td if you have elapsed days

     | id   birthy   birthm   birth    birthc |
     |----------------------------------------|
  1. |  1     1961        4      15    1961m4 |
  2. |  2     1963       11      46   1963m11 |
     +----------------------------------------+
```

Note that Jan.1960 is month 0 here!!

Matching datasets: `append` and `merge`

A common task is to match information from different datasets

- `append`: Observations with information on the same variables are stored separately
- `merge`: Different variables are defined for the same observations, but stored separately

Consider the following SOEP example:

- We have the first two SOEP person data sets **ap.dta** and **bp.dta**
- The same 5 persons in each data set
- Variables: person id, year of wave, happiness (11-point scale 0-10, 10=very happy)

ap.dta

	id	year	happy
1.	901	84	8
2.	1001	84	9
3.	1101	84	6
4.	1201	84	8
5.	1202	84	8

bp.dta

	id	year	happy
1.	901	85	8
2.	1001	85	6
3.	1101	85	7
4.	1201	85	8
5.	1202	85	8

Matching datasets: append

```
         +-----------------------
           id     year    happy
         ------------------------
1.       901       84       8
2.       1001      84       9
3.       1101      84       6
4.       1201      84       8
5.       1202      84       8
6.       901       85       8
7.       1001      85       6
8.       1101      85       7
9.       1201      85       8
10.      1202      85       8
         +-----------------------
```

append the rows of the second file beyond the last row of the first:

```
use ap.dta
append using bp.dta
```

ap.dta is the master-file
bp.dta is the using-file

```
           id     year    happy
         ------------------------
1.       901       84       8
2.       901       85       8
3.       1001      84       9
4.       1001      85       6
5.       1101      84       6
6.       1101      85       7
7.       1201      84       8
8.       1201      85       8
9.       1202      84       8
10.      1202      85       8
```

```
sort id year
```

Grouping observations of persons together and ordering them by year results in a
→ **panel dataset in long-format.**
Each row is called a
→ **“person-year”.**

Matching datasets: merge

Suppose that, for the persons in ap.dta, you need additional information on variable `hhinc` which is stored in apequiv.dta. To match variables on identical observations we can use `merge`.

```
              ap.dta
         +---------------------+
          id   year   happy
         ---------------------
1.       901    84       8
2.       1001   84       9
3.       1101   84       6
4.       1201   84       8
5.       1202   84       8
         +---------------------+
```

```
            apequiv.dta
         +-----------------------
          id   year    hhinc
         ------------------------
1.       901    84    9136.79
2.       1001   84    5773.51
3.       1101   84   10199.25
4.       1201   84   19776.77
5.       1202   84   19776.77
         +-----------------------
```

```
use ap.dta
merge 1:1 id using apequiv.dta
```

```
    +----------------------------------
      id  year   happy    hhinc   _merge
    -----------------------------------
1.   901   84      8    9136.79      3
2.  1001   84      9    5773.51      3
3.  1101   84      6   10199.25      3
4.  1201   84      8   19776.77      3
5.  1202   84      8   19776.77      3
    +----------------------------------
```

STATA added a variable _merge which equals 3 for all observations. This indicates that all observations are part of both files. If there were observations which occur only in ap.dta (the master-file), these would get value 1. Obs. which occur only in apequiv.dta (the using-file), would have _merge==2. (Naturally, obs. of the first type would have missings on hhinc, and obs. of the second type would have missings on happy.)

Reshaping datasets from wide- to long-format

id	ts1	tf1	st1	fail1	ts2	tf2	st2	fail2	ts3	tf3	st3	fail3	educ
1	19	22	1	1	22	26	2	1	26	29	1	0	9
2	23	28	1	1	28	30	2	0	.	.	.	.	13

```
reshape long ts tf st fail, i(id) j(episode)
```

id	episode	ts	tf	st	fail	educ
1	1	19	22	1	1	9
1	2	22	26	2	1	9
1	3	26	29	1	0	9
2	1	23	28	1	1	13
2	2	28	30	2	0	13
2	3	.	.	.	.	13

Here we have two persons, with 3 episodes each. In wide format all variables from the same episode need a common suffix. Here we simply numbered the episodes. The command for transforming in long format is `reshape long`. Then we list all episode-specific variables (without suffix). `i()` gives the person identifier variable and `j()` the new episode identifier variable created by Stata. All constant variables are copied to each episode.

How to repeat yourself without going mad: Loops

An extremely helpful technique to do tasks over and over again are loops. In Stata, there are (among others) `foreach`-loops and `forvalues`-loops. Both work in a similar way: They set a user-defined local macro to each element of a list of strings or list of numbers, and then execute the commands within the loop repeatedly, assuming that one element is true after the other.

```
foreach lname in list {
  commands referring to `lname'        //best for looping over strings
}                                      //or variable lists

forvalues lname = numlist {
  commands referring to `lname'        //best for looping over numbers
}
```

"lname" is the name of the local macro, "list" is any kind of list, "numlist" is a list of numbers (Examples: 1/10 or 0(10)100).

The local can then be addressed by `` `lname' `` in the commands.

Loops

To append files ap.dta, bp.dta,..., wp.dta, one could type many appends. However, the following does the same much more efficiently:

```
use ap.dta
foreach wave in b c d e f g h i j k l m n o p q r s t u v w {
    append using `wave'p.dta
}
```

`foreach` also recognizes varlists:

```
foreach var of varlist ts1-ts10 {
  replace `var'=. if `var'==-3
}
```

`forvalues` loops over numlists:

```
forvalues k=1/10 {
  replace ts`k'=. if ts`k'==-3
}
```

Second counter:
"k" is the counter. Sometimes we need a second counter, derived from the first:

```
forvalues k=1/100 {
   local l=`k'+1
   ...
}
```

Finding the month from a date variable:
Imagine the month an event has happened is measured in months since January 1983 (months83). From this we want to create a new variable (month) telling us, in which month (January, …, December) the event happened:

```
gen month = 0
forvalues j=1/12 {
 forvalues k=`j'(12)280 {
  quietly replace month = `j' if months83==`k'
 }
}    //note that Jan.83 is 1 here!!
```


Loops Example: Converting EH Data to Panel Data

Note: Data are in process time (i.e. age). Therefore, we produce also panel data on an age scale (sequence data). Normally, panel data are in calendar time (i.e. years).

id	ts1	tf1	st1	fail1	ts2	tf2	st2	fail2	ts3	tf3	st3	fail3	educ
1	19	22	1	1	22	26	2	1	26	29	1	0	9
2	23	28	1	1	28	30	2	0	.	.	.	.	13

```
egen maxage = rowmax(tf*)  //generate the max value for the looping

forvalues j = 15/30  {                  //panels from age 15 to age 30
   generate s`j' = 0 if `j' < maxage    //initializing with 0
   forvalues k = 1/3   {
      replace s`j' = st`k' if (`j' >= ts`k' & `j' < tf`k')
   }
}
```

id	s15	s16	s17	s18	s19	s20	s21	s22	s23	s24	s25	s26	s27	s28	s29	s30
1	0	0	0	0	1	1	1	2	2	2	2	1	1	1	.	.
2	0	0	0	0	0	0	0	0	1	1	1	1	1	2	2	.

Computations within panels (long-format)

- With panel data one often has to do computations within panels (groups)
- This is an example of a panel data set in long-format
  - Each record reports the observations on a person (id) in a specific year
  - This is termed “person-year”
  - A “panel” is defined as all person-years of a person

```
     +----------------+
     | id   year   X  |
     |----------------|
  1. |  1     84   2  |
  2. |  1     85   4  |
  3. |  1     86   1  |
  4. |  1     87   6  |
  5. |  1     88   4  |
     |----------------|
  6. |  2     84   3  |
  7. |  2     85   4  |
     +----------------+
```


The basic idea

It is essential that one knows the following:

```
bysort bylist1 (bylist2): command
     the by prefix; data are sorted according to bylist1 and bylist2
     and computations are done for the groups defined by bylist1
_n   system variable, contains the running number of the observation
_N   system variable, contains the maximum number of observations
```

- This does the computations separately for each panel:

```
sort id
by id: command
```

  - `bysort id:` is a shortcut

- If the time ordering within the panels is important for the computations then use

```
sort id year
by id: command
```

  - `bysort id (year):` is a shortcut

Numbering person-years

Example: Numbering the person-years

```
gen                 recnr   = _n     //assigns a record ID
bysort id (year): gen pynr    = _n     //person-year ID (within person)
bysort id:        gen pycount = _N     //# of person-years (within person)

     | id   year   X   recnr   pynr   pycount |
     |----------------------------------------|
  1. |  1     84   2       1      1         5 |
  2. |  1     85   4       2      2         5 |
  3. |  1     86   1       3      3         5 |
  4. |  1     87   6       4      4         5 |
  5. |  1     88   4       5      5         5 |
     |----------------------------------------|
  6. |  2     84   3       6      1         2 |
  7. |  2     85   4       7      2         2 |
     +----------------------------------------+
N.B.: If you now drop person-years, due to missing values (casewise
deletion) pycount is no longer correct! Compute it anew.
```

Example: Statistics over persons

```
tabulate pycount  if pynr==1    //distribution of person-years
```

Example: Identifying specific person-years

```
bysort id (year): gen first = 1   if _n==1     //first person-year
bysort id (year): gen last  = 1   if _n==_N    //last person-year
```


Using information from the year before

Explicit subscripting

It is possible to address specific values of a variable X (within a group) by using subscripts:

```
   X[1]        //X value of first person-year
   X[_N]       //X value of last person-year
   X[_n-1]     //X value of person-year before (X[0] is .)

   bysort id (year): gen firstx = X[1]   //firstx contains the first X-value
```

Example: Computing growth

```
   bysort id (year): gen grx = (X – X[_n-1]) / X[_n-1]

     | id   year   X          grx |
     |----------------------------|
  1. |  1     84   2            . |   Note:
  2. |  1     85   4            1 |   Always think about
  3. |  1     86   1         -.75 |   how your solution
  4. |  1     87   6            5 |   behaves at the
  5. |  1     88   4    -.3333333 |   first person-year!
     |----------------------------|
  6. |  2     84   3            . |
  7. |  2     85   4     .3333333 |
     +----------------------------+
```

Using the lag-operator

The Lag-Operator "L." uses the observation in t-1. If this observation does not exist (due to a gap in the data) L.X returns a missing. X[_n-1] returns the value of the observation before, irrespective of any gaps.

```
bysort id (year): gen xn_1 = X[_n-1]
xtset id year
gen lx = L.X

     +---------------------------+
     | id   year   X   xn_1   lx |
     |---------------------------|
  1. |  1     84   2      .    . |
  2. |  1     85   5      2    2 |
  3. |  1     87   3      5    . |
  4. |  1     88   7      3    3 |
     +---------------------------+
```


Finding statistics of X within persons

The egen command is very helpful (many more functions are available, see `help egen`):

```
bysort id (year): gen  cumx   = sum(X)      //Summing up X
bysort id:        egen maxx   = max(X)      //Maximum
bysort id:        egen totx   = total(X)    //Sum
bysort id:        egen meanx  = mean(X)     //Mean
bysort id:        egen validx = count(X)    //Number of nonmiss
bysort id:        gen  missx  = _N-validx   //Number of missings

     +--------------------------------------------------------------+
     | id   year   X   cumx   maxx   totx   meanx   validx   missx |
     |--------------------------------------------------------------|
  1. |  1     84   2      2      7     17    4.25        4       0 |
  2. |  1     85   5      7      7     17    4.25        4       0 |
  3. |  1     86   3     10      7     17    4.25        4       0 |
  4. |  1     87   7     17      7     17    4.25        4       0 |
     |--------------------------------------------------------------|
  5. |  2     84   4      4      6     10       5        2       1 |
  6. |  2     85   .      4      6     10       5        2       1 |
  7. |  2     86   6     10      6     10       5        2       1 |
     +--------------------------------------------------------------+
```

```
Variant: Finding statistics within person-episodes (spells)
Assume that we have a spell-indicator variable "spell"
bysort id:       gen minX1 = min(X) if spell==1  //minimum within spelltype 1
bysort id spell: gen minX  = min(X)              //minimum within each spelltype
```

Deriving time-varying covariates I

In this context the **function sum(exp)** is very important (exp is a logical expression)

- exp can be 1 (true), 0 (false), or .
- sum(exp) returns a 0 in the first person-year also if exp==.

```
* marr is an indicator variable for the person-year of marriage
bysort id (year): gen married = sum(marr==1)          //married=1 after marriage
bysort id (year): gen ybefore = married[_n+1]-married //the year before marriage

* lf gives the activity status (0=out of lf, 1=employed, 2=unemployed)
bysort id (year): gen lfchg = sum(lf~=lf[_n-1] & _n~=1)     //# of changes in lf

     +---------------------------------------------------+
     | id   year   marr   lf   married   ybefore   lfchg |
     |---------------------------------------------------|
  1. |  1     84     -1    0         0         0       0 |
  2. |  1     85     -1    0         0         1       0 |
  3. |  1     86      1    1         1         0       1 |
  4. |  1     87     -1    1         1         0       1 |
  5. |  1     88     -1    1         1         .       1 |
     |---------------------------------------------------|
  6. |  2     84     -1    0         0         0       0 |
  7. |  2     85     -1    0         0         1       0 |
  8. |  2     86      1    1         1         0       1 |
  9. |  2     87     -1    2         1         1       2 |
 10. |  2     88      1    1         2         .       3 |
     +---------------------------------------------------+
```


Deriving time-varying covariates II

Identifying first and last occurrences of specific states. Here unemployment (lf==2)

```
* Identifying the first occurrence
bysort id (year): gen first = sum(lf==2)==1 & sum(lf[_n-1]==2)== 0

* Identifying the last occurrence
gsort id  -year                                //sorting in reverse time order
by id: gen last = sum(lf==2)==1 & sum(lf[_n-1]==2)==0    //do not sort again
sort id year

     +--------------------------------+
     | id   year   lf   first    last |
     |--------------------------------|
  1. |  1     84    0       0       0 |
  2. |  1     85    2       1       0 |
  3. |  1     86    1       0       0 |
  4. |  1     87    1       0       0 |
  5. |  1     88    2       0       1 |
     |--------------------------------|
  6. |  2     84    0       0       0 |
  7. |  2     85    0       0       0 |
  8. |  2     86    1       0       0 |
  9. |  2     87    2       1       1 |
 10. |  2     88    1       0       0 |
     +--------------------------------+
```

Copying time of first occurrence:

```
bysort id (first):       ///
  gen yfirst = year[_N]
```

Missings / gaps in panels

When programming always be aware that there are certainly missings or even gaps (a whole person-year is missing) in the panels. These have the potential to wreck your analysis. Consider an example. We want to analyze the effect of being married on Y. We have a variable on civil status "fam" (0=single, 1=married, 2=divorce):

```
     +------------------+
     | id   year    fam |
     |------------------|
  1. |  1     84      0 |
  2. |  1     85      1 |
  3. |  1     86      1 |
  4. |  1     87      1 |
  5. |  1     88      2 |
     |------------------|
  6. |  2     84      0 |
  7. |  2     85      1 |
  8. |  2     86      . |
  9. |  2     87      1 |
 10. |  2     88      1 |
 11. |  2     89      1 |
     +------------------+
```

How to deal with the missing? In this case it might make sense to impute 1 (see the example below, on how this could be done). Normally, however, one would drop the whole person-year (`drop if fam==.`) and create thereby a gap. This has to be taken into regard, when constructing time-varying covariates (see next slide).

Missings / gaps in panels

Example: Years since marriage

```
* This is the correct solution taking gaps into account
recode fam 2/max=. , into(marr)                     //marriage indicator (spell)
bysort id: egen ymarr = min(year) if marr==1        //finding marriage year
gen yrsmarr = year – ymarr                          //years since marriage

* This produces a wrong result
bysort id (year): gen yrsmarr1 = sum(marr[_n-1]) if marr==1

     +------------------------------------------------------+
     | id   year   fam   marr   ymarr   yrsmarr   yrsmarr1 |
     |------------------------------------------------------|
  1. |  1     84     0      0       .         .          . |
  2. |  1     85     1      1      85         0          0 |
  3. |  1     86     1      1      85         1          1 |
  4. |  1     87     1      1      85         2          2 |
  5. |  1     88     2      .       .         .          . |
     |------------------------------------------------------|
  6. |  2     84     0      0       .         .          . |
  7. |  2     85     1      1      85         0          0 |
  8. |  2     87     1      1      85         2          1 |
  9. |  2     88     1      1      85         3          2 |
 10. |  2     89     1      1      85         4          3 |
     +------------------------------------------------------+
```

Lessons for panel data preparation

- Make yourself comfortable with
  - `merge` and `append`
  - `reshape`
  - `foreach` and `forvalues`
  - by-Prefix
  - `egen`-functions
  - Explicit subscripting
- Always think about what happens with your solution
  - In the first person-year
  - If there are missings in the panel
  - If there are gaps in the panel
- List, list, and list
  - After each programming step try to understand what is going on by listing a few persons (complicated persons with missings, gaps, …)
  - `list id year … if id<4, sepby(id)`

Complex Examples

The following slides contain more complex examples

Filling up missings with the value from before, but only in between valid observations

```
* This is only an exercise, this kind of imputation usually makes no sense
gen X = inc
bysort id (year): gen first = sum(X<.)==1 & sum(X[_n-1]<.)== 0  //first valid inc
gsort id  -year
by id: gen last = sum(X<.)==1 & sum(X[_n-1]<.)==0         //last valid inc
bysort id (year): gen spell = sum(first)-sum(last[_n-1])  //spell "being in panel"
* Filling in the value from before (this produces a cascade effect)
bysort id (year): replace X=X[_n-1] if X==. & spell==1
* Running # of missings encountered (this is only a little add on)
bysort id (year): gen nmiss=sum(missing(inc))
     +---------------------------------------------------------+
     | id   year    inc      X   first   last   spell   nmiss |
     |---------------------------------------------------------|
  1. |  1     84   1000   1000       1      0       1       0 |
  2. |  1     85   1100   1100       0      0       1       0 |
  3. |  1     86      .   1100       0      0       1       1 |
  4. |  1     87      .   1100       0      0       1       2 |
  5. |  1     88   1400   1400       0      1       1       2 |
     |---------------------------------------------------------|
  6. |  2     84      .      .       0      0       0       1 |
  7. |  2     85   2300   2300       1      0       1       1 |
  8. |  2     86      .   2300       0      0       1       2 |
  9. |  2     87   2400   2400       0      1       1       2 |
 10. |  2     88      .      .       0      0       0       3 |
     +---------------------------------------------------------+
```


Imputation of missings by linear interpolation

```
gen X = inc
bysort id (year): gen first = sum(X<.)==1 & sum(X[_n-1]<.)== 0  //first valid inc
gsort id  -year
by id: gen last = sum(X<.)==1 & sum(X[_n-1]<.)==0        //last valid inc
bysort id (year): gen spell = sum(first)-sum(last[_n-1]) //indicator for spell "being in panel"

gen spellm = 1 if spell==1 & X==.                        //indicator for missing spell (MS)
bysort id spellm:        gen lspellm=_N   if spellm==1   //length of missing spell
bysort id spellm (year): gen nrspellm=_n  if spellm==1   //numbering the person-years of MS

bysort id (year): gen incb = X[_n-1] if spellm==1 & spellm[_n-1]==.   //last inc before MS
gsort id -year
by id:            gen inca = X[_n-1] if spellm==1 & spellm[_n-1]==.   //first inc after MS
bysort id (incb): replace incb = incb[1] if spellm==1                 //filling up incb
bysort id (inca): replace inca = inca[1] if spellm==1                 //filling up inca
sort id year
replace X = incb + nrspellm * ((inca-incb)/(lspellm+1)) if spellm==1   //imputing missing inc
     +-----------------------------------------------------------------+
     | id   year    inc      X   spellm   lspellm   nrspellm   incb   inca |
     |-----------------------------------------------------------------|
  1. |  1     84   1000   1000        .         .          .      .      . |
  2. |  1     85   1100   1100        .         .          .      .      . | Note:
  3. |  1     86      .   1200        1         2          1   1100   1400 | Works only if
  4. |  1     87      .   1300        1         2          2   1100   1400 | there is only
  5. |  1     88   1400   1400        .         .          .      .      . | one MS per id
     |-----------------------------------------------------------------|
  6. |  2     84      .      .        .         .          .      .      . |
  7. |  2     85   2300   2300        .         .          .      .      . |
  8. |  2     86      .   2350        1         1          1   2300   2400 |
  9. |  2     87   2400   2400        .         .          .      .      . |
 10. |  2     88      .      .        .         .          .      .      . |
     +-----------------------------------------------------------------+
```

Creating a balanced panel

Sometimes one would like to “blow up” the dataset to a balanced one. In the following example the max person-years is 3. We create a new dataset, where every id has 3 observations.

```
* Starting with the real data (data.dta)
* Creating a list of the ids (idlist.dta)
bysort id: keep if _n==1
keep id
save idlist.dta

clear
set obs 3                           //number of observations in balanced panel
gen time = _n
cross using idlist.dta              //all pair wise combinations of time and id
merge 1:1 id time using data.dta    //merge the real data
```

DATA.DTA

	id	time	X
1.	1	1	4
2.	1	2	7
3.	1	3	2
4.	2	1	3
5.	2	3	5
6.	3	2	8

	id	time	X	_merge
1.	1	1	4	3
2.	1	2	7	3
3.	1	3	2	3
4.	2	1	3	3
5.	2	2	.	1
6.	2	3	5	3
7.	3	1	.	1
8.	3	2	8	3
9.	3	3	.	1

Converting EH Data to Panel Data (EH data.do)

EH Data (Marriage Episodes), Calendar Axis

id	birthy	ts1	tf1	end1	ts2	tf2	end2	inty
1	1971	1990	1993	1	1997	.	.	2000
2	1970	1993	1998	2	.	.	.	2000

EH Data, Age Axis

id	birthy	ts1	tf1	end1	ts2	tf2	end2	ts3	inty
1	1971	19	22	1	26	29	3	30	2000
2	1970	23	28	2	31	.	.	.	2000

end*:
1=divorce
2=death spouse
3=censoring

Panel Data, Age 17-30 (0=single, 1=married, 2=divorced, 3=widowed)
These data could be used as an input for a sequence analysis!

id	s17	s18	s19	s20	s21	s22	s23	s24	s25	s26	s27	s28	s29	s30
1	0	0	1	1	1	2	2	2	2	1	1	1	1	.
2	0	0	0	0	0	0	1	1	1	1	1	3	3	3

Converting EH Data to Panel Data (EH data.do)

```
gen  ageint  = inty - birthy       //age at interview
egen nepi    = rownonmiss(ts*)     //number of valid marriage episodes

* Preset state 0 (single) over the whole sequence
forvalues j = 17/30  {
   generate s`j' = 0 if `j' <= ageint
}

if nepi>0 {                   //The rest is only for those who married at least once

  forvalues k=1/2 {
    replace tf`k'  = inty  if `k'==nepi & tf`k'==.    //imputing inty for censored episodes
    replace end`k' = 3     if `k'==nepi & end`k'==.   //flaging censored episodes with end==3
  }

  forvalues k=1/2 {                                   //converting years to age
    replace ts`k' = ts`k' - birthy
    replace tf`k' = tf`k' - birthy
  }

  forvalues k=1/2 {                                   //setting the endpoint of the sequence
        replace ts`k'=ageint+1 if `k'==nepi+1
  }
  gen  ts3 = ageint+1 if nepi==2

  forvalues j=17/30 {
    forvalues k=1/2 {
          local l=`k'+1
          quietly replace s`j' = 1  if `j'>=ts`k' & `j'<=tf`k'               //married
          quietly replace s`j' = 2  if `j'>=tf`k' & `j'< ts`l' & end`k'==1   //divorced
          quietly replace s`j' = 3  if `j'>=tf`k' & `j'< ts`l' & end`k'==2   //widowed
```