Using remote access to big datasets efficiently with Stata

2009 German Stata Users Group meeting, Bonn

Volker Lang, M.A. Dipl.-Vw.
Eberhard-Karls-University Tübingen
Institut of Sociology
v.lang@uni-tuebingen.de

26-06-2009

1 Data used with remote access

LIAB: Linked-Employer-Employee-Data of the Institut for Employment Research (IAB) in Nuremberg (cf. Jacobebbinghaus, 2008; Alda et al., 2005)

- Longitudal data of German firms and their employees covering the timespan between 1993 and 2006.
- Consists of waves of the IAB firm panel ("IAB-Betriebspanel") and waves of the IAB employee sample ("IAB-Beschäftigtenstichprobe").

1 Anonymisation of the output

- Typically remote access is implemented on survey or process generated data for privacy reasons. Therefore parts of the output are typically anonymised.
- In case of the LIAB every suppopulation of the data smaller then 20 observational units is blanked in the output submitted to the users. Since the LIAB data set contains a huge number of cases (see later) one typically only runs into problems with that when analysing very rare strata.
- For the same reason graphs are only submitted to the users if their are saved including the option **asis**.

2 Using ado-files

- For security reasons the servers hosting the data are typically not directly connected to the internet. In case of the LIAB one sends his do-files to the FDZ of the IAB and they pass it on to the server.
- This implies that you cannot directly call and install ado-files from the internet. However on request it is possible to get them installed.
- I ran into problems with that when I tried to use the scheme **lean** with graphs and the package **parmest**.
- These problems can be solved through communication.

3 Size of the data

- Typically remote access data sets are huge. F.e. the LIAB contains information on about 2 million employees. Using them for event history analysis as in my case this can ad up to more than 7 million job spells and data sets of 8-10 GB size when using episode splitting on this spells.
- Therefore running do-files on the whole data set can be very slow and sometimes even cause convergence problems with models.
- I ran into problems with that often in the beginning using **stcox** and **streg**.
- The obvious solution to that problem is using **sample** or **bsample** before estimating the models. That is fine but would not exploid one big advantage of huge data sets ..

3 Size of the data

- Huge data sets typically contain large case numbers even in rare strata.
- Drawing a random sample of such a dataset would typically reproduce the distribution of the original data but with relativly smaller case numbers.
- In the subsample absolute case numbers in rare strata can become so small that one runs into technical difficulties estimating models on them. (A problem similar to smaller data sets.)
- But with a huge data set there is an alternative: **Sampling equal sized strata**.

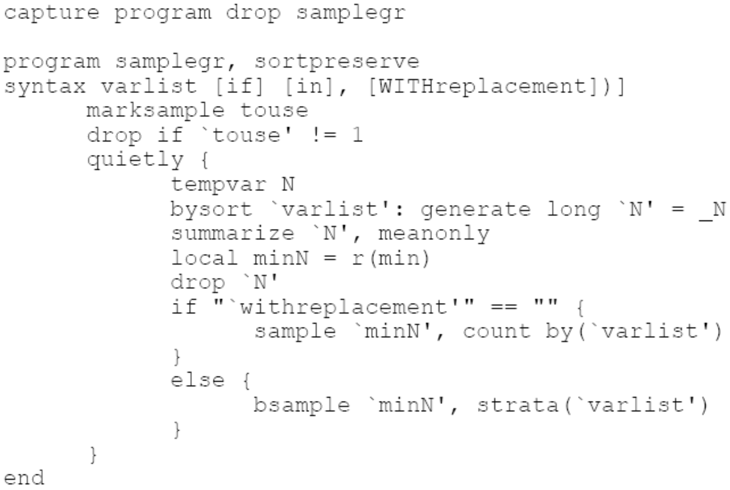
1 Sampling equal sized strata

Basically this means ..

1. Using the case number information which would be produced by a n-way cross-tabulation of the variables used for stratification.
2. Use a function to find the minimum case number in that matrix.
3. Plug that information into the sampler used.

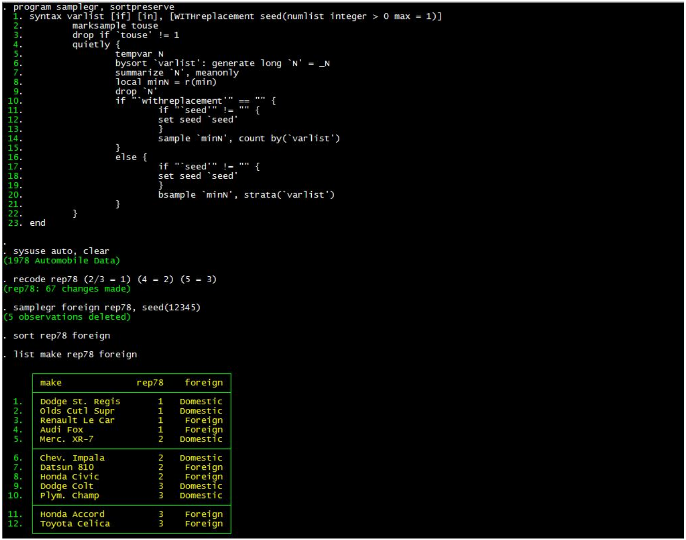
I wrote a program called **samplegr** preforming these 3 steps. The user only has to specify the variables he wants to use for stratification.

2 samplegr



```
capture program drop samplegr

program samplegr, sortpreserve
syntax varlist [if] [in], [WITHreplacement])]
      marksample touse
      drop if `touse' != 1
      quietly {
            tempvar N
            bysort `varlist': generate long `N' = _N
            summarize `N', meanonly
            local minN = r(min)
            drop `N'
            if "`withreplacement'" == "" {
                  sample `minN', count by(`varlist')
            }
            else {
                  bsample `minN', strata(`varlist')
            }
      }
end
```

3 Used on car data



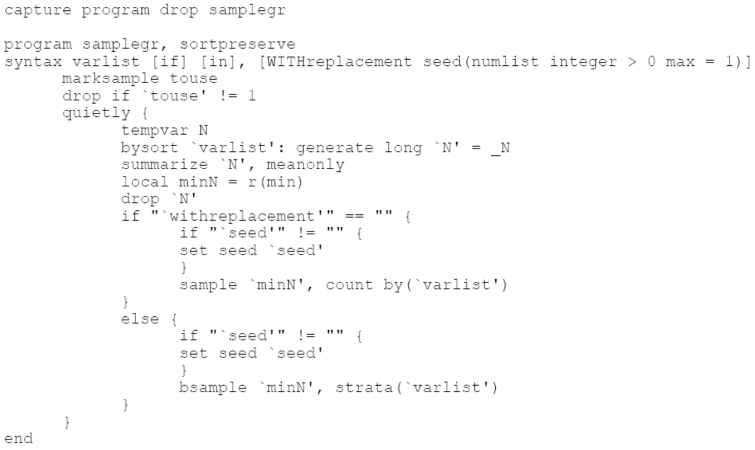
```
. program samplegr, sortpreserve
  syntax varlist [if] [in], [WITHreplacement seed(numlist integer > 0 max = 1)]
          marksample touse
          drop if `touse' != 1
          quietly {
                  tempvar N
                  bysort `varlist': generate long `N' = _N
                  summarize `N', meanonly
                  local minN = r(min)
                  drop `N'
                  if "`withreplacement'" == "" {
                          if "`seed'" != "" {
                          set seed `seed'
                          }
                          sample `minN', count by(`varlist')
                  }
                  else {
                          if "`seed'" != "" {
                          set seed `seed'
                          }
                          bsample `minN', strata(`varlist')
                  }
          }
  end

.
. sysuse auto, clear
(1978 Automobile Data)

. recode rep78 (2/3 = 1) (4 = 2) (5 = 3)
(rep78: 67 changes made)

. samplegr foreign rep78, seed(12345)
(5 observations deleted)

. sort rep78 foreign

. list make rep78 foreign

     +-------------------------------------+
     | make                rep78   foreign |
     |-------------------------------------|
  1. | Dodge St. Regis         1  Domestic |
  2. | Olds Cutl Supr          1  Domestic |
  3. | Renault Le Car          1   Foreign |
  4. | Audi Fox                1   Foreign |
  5. | Merc. XR-7              2  Domestic |
     |-------------------------------------|
  6. | Chev. Impala            2  Domestic |
  7. | Datsun 810              2   Foreign |
  8. | Honda Civic             2   Foreign |
  9. | Dodge Colt              3  Domestic |
 10. | Plym. Champ             3  Domestic |
     |-------------------------------------|
 11. | Honda Accord            3   Foreign |
 12. | Toyota Celica           3   Foreign |
     +-------------------------------------+
```

4 Setting the seed

Problem: For reasons I haven't figured out yet **samplegr** doesn't get the information of the global stored if a **set seed** command is used in the code.

To solve the problem I tried to plug the **set seed** command into the program using an option to be specified in the syntax by the user (see next slide). Unfortunately that doesn't work out!

5 samplegr with set seed



```
capture program drop samplegr

program samplegr, sortpreserve
syntax varlist [if] [in], [WITHreplacement seed(numlist integer > 0 max = 1)]
      marksample touse
      drop if `touse' != 1
      quietly {
            tempvar N
            bysort `varlist': generate long `N' = _N
            summarize `N', meanonly
            local minN = r(min)
            drop `N'
            if "`withreplacement'" == "" {
                  if "`seed'" != "" {
                  set seed `seed'
                  }
                  sample `minN', count by(`varlist')
            }
            else {
                  if "`seed'" != "" {
                  set seed `seed'
                  }
                  bsample `minN', strata(`varlist')
            }
      }
end
```

4 Questions for discussion

1. Do you have any suggestions on how to fix the problem combining **samplegr** with **set seed**?
2. Do you have any suggestions on how to improve or extend the program in other ways?
3. Do you think that **samplegr** can be useful for other people?