Transform a Hierarchy Long Format Data to Wide Format with Large Number of Variables that Have Heterogeneous Names

Liang Xie   Department of Economics, SUNY Binghamton, Binghamton, NY

ABSTRACT

In one nursing research project, each record from the original data set contains information either about the household or about the individuals under the household. The research fellow requires that the two-layer structure data be transposed into a WIDE format in that all records from the same household and associated individuals form just one record. The difficulties with this manipulation include:

1. Finish the manipulation on a million-record database in a reasonable time;
2. Process large number of heterogeneous variables' names in a neat and automatic way;
3. Allow flexibility in handling different data sets with similar structure.

One approach utilizes the matrix manipulation power of SAS/IML®. This approach is natural and easy to implement, but it does not have adequate efficiency. The other approach takes advantage of the efficiency of DATA STEP and flexibility of SAS/MACRO®. The gain in efficiency is enormous while maintains flexibility. In this paper I am going to explore this latter method.

INTRODUCTION

I have been involved in a nursing project that the research fellow requires a transformation of 53 data sets that share the same structure but different data from different states of US.

The original format of the data is a multi-layer structure, i.e. for each record in a row, it has either observations from household level data or observations from individual level data such that the individual is associated with the household record above him. The structure is like this:

ORIGINAL DATA

HOUSEHOLD VAR 1	….	HOUSEHOLD VAR H	PERSONAL VAR 1	…	PERSONAL VAR P
….	….	….	….	….	….
H obs n	….	H obs n	missing	…	missing
H obs n retained	….	H obs n retained	1st P obs of the n_th household	…	4th P obs of the n_th household
H obs n+1	….	H obs n+4	missing	…	missing
H obs n+2	….	H obs n+2	missing	…	missing
H obs n+3	….	H obs n+3	missing	…	missing
H obs n+3 retained	….	H obs n+3 retained	1st P obs of n+3 household	…	1st P obs of n+3 household
H obs n+3 retained	….	H obs n+3 retained	2nd P obs of n+3 household	…	2nd P obs of n+3 household
H obs n+3 retained	….	H obs n+3 retained	3rd P obs of n+3 household	…	3rd P obs of n+3 household
H obs n+3 retained	….	H obs n+3 retained	4th P obs of n+3 household	…	4th P obs of n+3 household
H obs n+3 retained	….	H obs n+3 retained	5th P obs of n+3 household	…	5th P obs of n+3 household
H obs n+4	….	H obs n+7	missing	…	missing
…	…	…	…	…	…

By examining the structure of the original data, we find some characteristics:

1. This is a 2-layer structure data. The first layer is household record; after each household record, there are associated sub-layer individuals' records;
2. The number of individuals under a single household is heterogeneous, ranging from 0 individuals to several ones. While the specific number could only be obtained after go over the whole data set;
3. Each record for a person under a household retained the data from household record while corresponding personal variables are set to missing for household record;
4. The original data set has 260 variables in total, while 106 are household variables and the rest 154 are personal variables; And the names are heterogeneous share no common pattern;
5. The number of observations ranges from just under 10k to over 300k for a single data set in the 1% sample, while for the 5% sample, those numbers will be 5 times bigger.

The research fellow requires us to transform the data into a wide format in that all records from the same household should be grouped into one single row. The desired structure is like this:

DESIRED DATA

H VAR 1	**....**	**H VAR h**	**P_1's VAR 1**	**...**	**P_1 VAR p**	**....**	**P_max's VAR 1**	**...**	**P_max VAR p**
....	....	....	....	....	....	**....**	....	....	....
H obs n	....	H obs n	missing	...	missing	**....**	missing	....	missing
H obs n+1	....	H obs n+1	1st P obs of H n	...	4th P obs of H n	**....**	missing	....	missing
H obs n+2	....	H obs n+2	missing	...	missing	**....**	missing	...	missing
H obs n+3	....	H obs n+3	Max P obs of H n+3	...	max P obs of H n+3	**....**	max P obs of H n+3	...	max P obs of H n+3
...	...	...	...	...	...	**....**	...	...	...

We can find out some features from above table:

1. Each row contains all observations associated with one household;
2. Different rows corresponds to different households;
3. The data set is sparse over all the personal data fields;
4. Many new variables will be generated;

Some papers have already addressed the issue of transforming multi-record data set into single-record data set. These applications share some common requirements that make them unsuitable to process this database. For example, all these applications require the original data be sorted and indexed. Besides, most of them require the same number of observations for each group. In addition, most of them handle the variables in a way that is not appropriate for hundreds.

Therefore, the major challenges include:
1. Deal with unknown maximum number of individuals in a single household;
2. Transform the data without sorting and indexing but with enough efficiency;

3. Handle the large number of heterogeneous variables in a neat and traceable way: once the data set is spanned, the total number of variables will be about 2,000 and the number of observations ranges from 3,000 to 460,000!

A METHOD UTILIZES DATA STEP & SAS/MACRO

DATA STEP is a very powerful tool in SAS® to handle almost any size and any kind of data. I also utilize SAS/Macro to deal with large number of heterogeneous variables' names and the number of data sets to be processed.

The first task is to determine the maximum number of individuals in a single household so that we can span the data set appropriately. The basic idea is just to go through the whole data set and do some simple count. The following Macro implements this idea:

```
%macro count_P(ds, var, value) /                            ①
              store source
              des="count the max number of individuals in a household ";
%global pid_hi;
  data _null_;
    set &ds;
      if ctspbuff=. then do;                                ②
      ctspbuff=0;
        ctsp=0;
      end;
      retain ctsp ctspbuff;                                 ③
      if &var="&value" then ctsp=0;
    else do;
      ctsp=ctsp+1;
      if ctsp>ctspbuff then ctspbuff=ctsp;
    end;
    call symput('pid_hi', ctspbuff) ;                       ④
    %put &pid_hi;
  run;
%mend count_P;
```

① pid_hi is set to be global for further use;

② initialize counters. ctspbuff stores the maximum number of individuals in a household;

③ retain counters' values to make count correct;

④ transfer buffered max number to macro &pid_hi;

The second challenge is to deal with the heterogeneity of variables' names. I tried to solve this by specifying the desired variables in a separate data sets and transferring their values to two series of 'well-organized' macro names: the macro variables are named as hname&i or pname&i so that we can use loop to get desired variables' names and associated values. The data set for variables makes users conveniently input desired variables. These tasks are done via following two steps.

First, the variables' names are stored in two data sets via simple line-stream data step:

```
libname xdata "Path";
data xdata.varh;
  input hname $ length;

datalines;
name1  length1
name2  length2
name3  length3
name4  length4
name5  length5
…
;
```

It is similar for variables of individuals.

To transfer the names, using simple call routine of symput in next data step:

```
%macro getname /
       store source
       des="Get household and personal vars' name as macro vars";
data _null_;
  set xdata.varh;
  call symput(compress('hname'||_n_),hname);                ①
  call symput(compress('hlength'||_n_),length);
  call symput('numh',_n_);                                  ②
run;
data _null_;
  set xdata.varp;
  call symput(compress('pname'||_n_),pname);
  call symput(compress('plength'||_n_),length);
  call symput('nump',_n_);
run;
%mend getname;
```

① Ordering the variables and their associated length;

② Macro variable numh counts up how many variables for Household observations;

Once we stored our variables in a series of ordered macros, we can resolve the variables' names and underlying values using multiple ampersand &&hname&n and &&&hname&n. Resolving more than 2 ampersands in Macro facility is not common and it needs some explanation. The Macro facility first resolve it to [&&][&hname][&n], in the next resolving process, it becomes [&][var name][order number], say &hname5, which actually resolves the underlying value of macro variable hname5. In the program, I first resolve the names and then resolve the underlying values by referring to the resolved name. This is a clearer way.

Now we are ready to transform the LONG format into WIDE format.

First, we need to define an array for group-level variables, say Household variables, and the length of this array is the desired number of group-level variables. Second, we need to define one array for each sub-group level variable and its length is the maximum number of individuals in a household. The following segment completes this task:

```
array H_array(&hvar_low:&hvar_hi) $ _H&hvar_low - _H&hvar_hi;     ①
      retain _H&hvar_low - _H&hvar_hi;                            ②
      %do i=&hvar_low %to &hvar_hi;                               ③
        rename _H&i=&&hname&i;
      %end;

%do i=&pvar_low %to &pvar_hi;                                     ④
    %let firstpart=&&pname&i;
    %let nametemp=&firstpart;
    %let dash=_;
    %let macro_name=&nametemp&dash;                               ⑤
    array P_&i(&pid_low:&pid_hi) $ &&plength&i                    ⑥
               &macro_name&pid_low - &macro_name&pid_hi;
    retain &macro_name&pid_low - &macro_name&pid_hi;
%end;
```

① All household variables are defined in an array, and given name _H&i so that it is easy to manipulate them;
② The values are retained;
③ Change the ordered names of household variables to their original names;
④ Individual variables' array is defined for each one of variables;
⑤Get the newly defined individual variables' names for easy exposure and manipulation;
⑥ The length parameters are included;

We still need a counter that will give us the right value of individual sequential number so that the data can be put in the right cells. This is done by including a counter:

```
if pid=. then pid=1;
else pid=pid+1;
```

The first command line is to initialize the value of the counter.

Given the structure of the data set, observations for a group always start from household records. Therefore, a household observation is a good indicator for output. That is when a household observation is read, the program outputs all stored values before process this household record. Obviously, output operation is only doable from the second household record on. Once output is done, we transfer all values from macro variable to appropriate cells in corresponding arrays. Note that since all arrays' values are retains at the start of DATA STEP iteration, we need to initialize the values for individual record at the household record step to truncate redundant individual records for next group. The counter for individuals in a single household is initialized, too, at this point. The macro variable &IDvar has been referred to variable 'Rectype' in the original data.

```
if (&IDvar = "&IDvalue") then do;
   if _n_ >1 then output;
   %do i=&hvar_low %to &hvar_hi;
```

```
     %let tempvarname=&&hname&i;
     H_array[&i]=%sysfunc(left(&&tempvarname));
   %end;
   %do n=&pid_low %to &pid_hi;
      %do i=&pvar_low %to &pvar_hi;
         %let tempvarname=&&pname&i;
         %let dash=_;
         %let macro_name=&tempvarname&dash;
         P_&i[&n]=' ';
      %end;
   %end;
   pid=0;
end;

if (&IDvar ne "&IDvalue") then do;
  pidhi="&pid_hi";
  if (pid LE pidhi) then do;
    %do i=&pvar_low %to &pvar_hi;
       %let tempvarname=&&pname&i;
       %let dash=_;
       %let macro_name=&tempvarname&dash;
       P_&i[pid]=%sysfunc(left(&tempvarname));
    %end;
  end;
end;
```

Further, if we substitute the naïve ‘if-then-else’ flow control statement with a ‘select-when’ statement, together with a sequence of numbered macro reference of ‘IDvalues’, we can extend the ability of the program from dealing with two-layer structure to dealing with arbitrary multi-layer structure. A sample code is the following:

```
%macro action1;
   statement1;
%mend;

%macro action2;
   statement2;
%mend;


……
%macro ExceptionHandeling;
   Exception handling statement;
%mend;

%macro generator;
      %nrbquote(when (&val) %action&val;)
%mend;

%macro Processor(InitialCaseNum,TotalCases);
data _null_;
   select (&IDvar);
   %do val=&InitialCaseNum %to &TotalCases;
     %generator
   %end;
   otherwise %ExceptionHandeling;
   end;
```

```
run;
%mend;
```

As a reminder, note that there is no semicolons after the '*when*' generating macro reference. Quoting function is used to prevent SAS from immediately resolving the generated statement. In my program, however, I did not use this flexible method for simplicity purpose only.

At the final step, we keep desired variables only. This is done by the 'keep' statement. For a neat presentation, I used a macro named %m_keep to generate the keep statement for newly generated individual-level variables.

```
%macro m_keep(name, low, hi) /
       store source
       des="Generate Keep Statement in DATA step";
  %let dash=_;
  %let name=&name&dash;
  keep
    %do k=&low %to &hi;
        &name&k
    %end;
%mend;
```

```
 ….
Keep                                                      ①
  %do i=&hvar_low %to &hvar_hi;
    &&hname&i
  %end;;                                                  ②

  %do i=&pvar_low %to &pvar_hi;
    %m_keep(&&pname&i, &pid_low, &pid_hi);                ③
  %end;
retain pid;
drop pid pidhi;
```

① generate the keep statement for household variables only, no semicolon in the looped statement;

② note the double semicolon at the end;

③ invoke the %m_keep macro to generate keep statement for new individual variables;

CONCLUSION

Macro facility is a very powerful tool for data processing, especially when it interacts with DATA STEP. Note that all techniques used in this paper have already been explored elsewhere separately, and when they are bind together, they get difficult job done.

The above macro is efficient in transforming the real data. By using a time() function at the beginning and at the end of the processing, we find that, on a regular PC, with P4 1.8GHz, 256MB RAM, and a 5400rpm 2MB cache HDD, it can de-normalize a 469,066-observation dataset with 130,340 household records on 1,646 variables in just 64 seconds, while the whole process time is less than 75 seconds. Following are some NOTES from SAS Log

```
NOTE: There were 469066 observations read from the data set XDATA.D0006.
NOTE: The data set XDATA.D0006_WIDE has 130340 observations and 1646 variables.
NOTE: DATA statement used:
      real time           1:03.87
      cpu time            45.12 seconds


5069   %let time1=%sysfunc(time());
5070   %let diff_time=%sysevalf(&time1-&time0);
5071   %put Total Processing Time of &longds is &diff_time;
Total Processing Time of xdata.d0006 is 73.1210000514984
```

Reference:

SAS Online Document for version 8, 1999, SAS Institute Inc., Cary, NC, USA.

Contact Information

Please forward comments and questions to:

Liang Xie
Email: lxie0@binghamton.edu