Paper 102-2010

Transposing Data Using PROC SUMMARY'S IDGROUP Option

John King, Ouachita Clinical Data Services, Mount Ida, AR
Mike Zdeb, University@Albany School of Public Health, Rensselaer, NY

ABSTRACT

When clinical data are stored with multiple observations per subject, a common task is to rearrange the data so there is only one observation per subject. That single observation contains all or part of the information previously spread over multiple observations. Such rearranging of data is commonly done with either PROC TRANSPOSE or with a data step that often contains one or more arrays. There is a little used alternative to these methods, the IDGROUP option in PROC SUMMARY. The method is little used since the task is not made explicit in the documentation for the procedure and it has not been described in any papers that the authors could find at SUGI, SAS Global Forum, or various regional user group meetings. This paper describes several situations and shows the SAS® code needed for using the IDGROUP option in PROC SUMMARY as an alternative to the more common methods for rearranging data.

INTRODUCTION

The "id-group-specification" was added to PROC SUMMARY in version 8; this excerpt from the online users guide describes it as follows ...

"id-group-specification combines the features and extends the ID statement, the IDMIN option in the PROC statement, and the MAXID and MINID options in the OUTPUT statement to create an OUT= data set that identifies multiple extreme values. The form of the id-group-specification is

IDGROUP (<MIN|MAX (variable-list-1) <...MIN|MAX (variable-list-n)>> <<MISSING> <OBS> <LAST>> OUT <[n]> (id-variable-list)=<name(s)>)"

This statement implies that IDGROUP supersedes all the other ID forms and that we should consider those forms depreciated. The IDGROUP keyword can be used multiple times in an OUTPUT statement, and the OUTPUT statement can be use multiple times in a single call to PROC SUMMARY.

The MIN|MAX keywords can appear zero or more times within an IDGROUP specification and the effect is somewhat analogous to a BY statement where MAX is similar to DESCENDING and MIN is similar to ASCENDING. For example, a BY statement in PROC SORT "BY descending A B" would order observations first by A, largest to smallest, and within each level of A, B would be ordered from smallest to largest. This would be written in IDGROUP as MAX (A) MIN (B). Also note that while the options MIN and MAX when used in the context of a summary procedure like PROC SUMMARY may imply that the target variables be numeric, that is not the case for IDGROUP variables. They may be either character or numeric. This makes it possible to find the MAX and MIN of character variables. In summary, the MIN|MAX keywords and the associated variables function similar to the BY statement to order the observations within each level of the class variables.

The MISSING keyword option directs PROC SUMMARY to treat missing values as valid levels for the ID variables. The default as with most procedures is to ignore observations with missing values. The OBS keyword option directs PROC SUMMARY to include variable(s) in the output data set to identify the observation number of the extreme value(s). The keyword option LAST directs PROC SUMMARY to change the criterion for selecting observations when there are multiple observations that fall into an ID group.

FINDING EXTREME VALUES AND TRANSPOSING DATA USING IDGROUP

One common example of finding extreme values is finding CMAX and TMAX from concentration data. Using the following small sample data, the example computes the extreme values CMAX, TMAX, CLAST. TMAX and CLAST could be obtained sorting by and merging but using PROC SUMMARY the values are obtained in one step without sorting. While finding interesting extremes we can also compute common summary statistics for concentration.

```
data conc;
   input subjid:$3. time:time5. conc @@;
   format time time5.;
datalines;
001 08:00 0    001 08:30 .1   001 09:00 .25
001 09:35 .2   001 10:05 .12  001 10:30 .05
002 08:01 0    002 08:31 .2   002 09:01 .2
002 09:36 .33  002 10:06 .22  002 10:31 .10
;
run;

proc summary data=conc nway;                                    1
   class subjid;                                                2
   var conc;                                                    3
   output out=concStats(drop=_:)                                4
      idgroup(max(conc) out(conc time)=cmax tmax)               5
      idgroup(max(time) out(conc)=clast)                        6
      n=n mean=mean std=std median=median                       7
      ;
run;
```

Code details ...

1. Call PROC SUMMARY using NWAY option to get class groups by SUBJID only.
2. Specify the CLASS or grouping variable SUBJID.
3. Tell PROC SUMMARY to use CONC as an analysis variable.
4. Begin the OUTPUT statement. Name the output data set and drop variables beginning with the underscore character; we don't need them for this application. No output is produce without at least one OUTPUT statement.
5. This IDGROUP option finds maximum concentration for each subject and outputs OUT (CONC TIME)= the maximum concentration and the TIME associated with the concentration naming them CMAX and TMAX.
6. This second IDGROUP options finds the maximum time for each subject and outputs OUT (CONC) the concentration associated with that time naming it CLAST.
7. These statistic options request summary statistics for the variable named in the VAR statement. This syntax is acceptable when there is one variable in the VAR list.

PROC SUMMARY produces the data set shown below, the maximum concentration (CMAX), the time (TMAX) associated with that extreme value and the concentration from the last observed time (CLAST).

Obs	subjid	cmax	tmax	clast	n	mean	std	median
1	001	0.25	9:00	0.05	6	0.120	0.09274	0.11
2	002	0.33	9:36	0.10	6	0.175	0.11274	0.20

Another example of finding multiple extreme values where all values from all observations are returned is very similar to a transpose. Transposing multiple variables to wide format usually requires at least two calls to PROC TRANSPOSE or a somewhat complicated data step using multiple arrays. Consider the following data and the PROC TRANSPOSE steps required to convert it to wide format. The data have FORMAT/INFORMAT and LABEL attributes associated with each variable. This will help show how the attributes are inherited by the different transpose methods demonstrated.

```
data have;
   attrib memberID length=$1 informat=$1. format=$1. label='Member ID';
   attrib IC       length=$3 informat=$3. format=$upcase3. label='IC';
   attrib Charge   length=8  informat=F8. format=dollar10.
                   label='Charge';
   input (_all_)(:) @@;
datalines;
1 301  5207  1 301  .     1   .  6082
2 473  5207  2 473  5207  2 301  6082
3 325  6082  3 473  5207
4 473    94  4 352    94
;
run;
```

PROC CONTENTS and PROC PRINT of data set HAVE look as follows ...

```
                        Variables in Creation Order
 #   Variable   Type   Len   Format      Informat   Label
 1   memberID   Char     1   $1.         $1.        Member ID
 2   IC         Char     3   $UPCASE3.   $3.        IC
 3   Charge     Num      8   DOLLAR10.   F8.        Charge

                    member
              Obs      ID      IC         Charge
                1      1       301        $5,207
                2      1       301             .
                3      1                  $6,082
                4      2       473        $5,207
                5      2       473        $5,207
                6      2       301        $6,082
                7      3       325        $6,082
                8      3       473        $5,207
                9      4       473           $94
               10      4       352           $94
```

Rearranging the data set with PROC TRANSPOSE entails using the following ...

```
proc transpose data=have out=IC(drop=_:) prefix=IC_;                1
   by memberID;                                                     2
   var IC;                                                          3
run;

proc transpose data=have out=charge(drop=_:) prefix=Charge_;        1
   by memberID;                                                     2
   var charge;                                                      3
run;

data wideFromTranspose;                                             4
   merge IC charge;                                                 5
   by memberID;                                                     6
run;
```

Code details:

1. Call PROC TRANSPOSE
   a. Specify input and output data sets names.
   b. Output name is transpose variable name
   c. PREFIX= specifies the root name of the SAS enumerated variable list created by PROC TRANSPOSE
2. BY statement to group the observations for each MEMBERID, data must be sorted.
3. Specify the name of the variable to transpose.
4. Create a new wide data set.
5. Merge statement lists the names of the output data sets from the PROC TRANSPOSE steps.
6. Match the observations in the merge BY the variable(s) listed.

PROC CONTENTS and PROC PRINT of data set WIDEFROMTRANSPOS look as follows ...

```
Variables in Creation Order
  #   Variable   Type   Len   Format      Informat   Label
  1   memberID   Char     1   $1.         $1.        Member ID
  2   IC_1       Char     3   $UPCASE3.
  3   IC_2       Char     3   $UPCASE3.
  4   IC_3       Char     3   $UPCASE3.
  5   Charge_1   Num      8   DOLLAR10.
  6   Charge_2   Num      8   DOLLAR10.
  7   Charge_3   Num      8   DOLLAR10.

      member
   Obs   ID   IC_1 IC_2 IC_3   Charge_1   Charge_2   Charge_3
    1    1    301  301           $5,207          .     $6,082
    2    2    473  473  301      $5,207     $5,207     $6,082
    3    3    325  473           $6,082     $5,207          .
    4    4    473  352              $94        $94          .
```

Advantages to this technique are inheritance of the FORMAT attribute and you do not need to know the dimension of the SAS enumerated variable lists that are created. Disadvantages include the amount of code needed; you need one call to PROC TRANSPOSE for each variable transposed plus the data step to combine the data.

We could use a data set transpose but we consider that method the least desirable option due to complexity and increased possibility for error.

```
data wideFromDataStep;                                  1
   array IC_[3] $3;                                     2
   array Charge_[3];                                    2
   do _n_ = 1 by 1 until(last.memberID);                3
      set have;                                         4
      by memberID;                                      4
      IC_[_n_] = IC;                                    5
      Charge_[_n_] = Charge;                            5
      end;                                              3
   drop IC Charge;                                      6
run;                                                    7
```

Code details ...

1. Start a data step and name the output.
2. Define ARRAY of variables to receive the values of the transposed variables.
3. Use a DO loop UNTIL (LAST. to read each BY group of the data. This avoids the need to RETAIN the array elements and takes advantage of the default initialize to missing feature of the data step.
4. SET statement names input data set and BY statement sets up FIRST and LAST variables.
5. Using index variable _N_ assign the transpose variables to the associated ARRAYS.
6. Drop the transposed variables.
7. End the data step.

This code produces exactly the same data but not the same data set as above because none of the attributes for the transposed variables are inherited. Each transposed variable must be specified in an array, of the proper type and length, contributing to the possibility of error.

PROC CONTENTS output for data set WIDEFROMDATASTEP looks as follows ...

```
Variables in Creation Order
#    Variable    Type    Len    Format    Informat    Label
1    IC_1        Char      3
2    IC_2        Char      3
3    IC_3        Char      3
4    Charge_1    Num       8
5    Charge_2    Num       8
6    Charge_3    Num       8
7    memberID    Char      1    $1.       $1.         Member ID
```

Contrast the above with a similar transpose produced using PROC SUMMARY, with a little help from PROC SQL.

```
proc sql noprint;                                                    1
   select max(obs) into :obs                                         2
from                                                                 3
      (                                                              3
   select count(*) as obs                                            3
         from have                                                   3
         group by memberid                                           3
       )                                                             3
      ;
  quit;
run;

proc summary nway data=have missing;                                 4
   class memberID;                                                   5
   output                                                            6
      out = work.wide(drop=_type_ _freq_)                            7
      idgroup(out[&obs](ic charge)=)                                 8
     ;
run;
```

Code details ...

1. Call PROC SQL with NOPRINT option we are creating a macro variable no printed output is required.
2. Create a macro variable OBS as MAX (OBS), we need a macro variable because the value needs to become part of the code for PROC SUMMARY.
3. Sub query is count of records for each MEMBERID.
4. Call PROC SUMMARY
   a. NWAY causes CLASS to output the highest WAY of the class levels. Similar to using BY MEMBERID but no sorting needed.
   b. MISSING specifies that MISSING values of the CLASS variable be treated as a valid class level.
5. Specify MEMBERID as the CLASS variable to group the observations.
6. OUTPUT statement needed to get PROC SUMMARY to create output data set.
7. OUT= specifies the name of the output data set. We don't need the two variables (_TYPE_ and _FREQ_) created by PROC SUMMARY so drop them.
8. This IDGROUP specification with no explicit ID variable(s) outputs each variable named in the OUT parameter in the order the values are encountered in the data set.
   a. The variables names mentioned in the parenthesized list following OUT[] are the variables for whom we want extreme values.
   b. The = following tells PROC SUMMARY to use the same names as the names in the parenthesized list.
   c. The number contained in the macro variable &OBS is the number of extreme observations.

This code produced exactly the same output as with a PROC TRANSPOSE for each variable and a MERGE step to combine the wide data sets, or the one step data step transpose. The advantage being that any number of character or numeric variables can be transposed in one step, and all variable attributes are inherited. Since we did not use the MIN or MAX options the extreme (transposed) values are returned in observation order. The disadvantage being the dimension of the SAS enumerated variable lists created by PROC SUMMARY needs to be determined before calling PROC SUMMARY. Here, done using PROC SQL, and the maximum number of observations for any CLASS group cannot exceed 100.

CONCLUSION

The IDGROUP option in PROC SUMMARY can be used to rearrange a data set from narrow (multiple observations per subject) to wide (single observation per subject with multiple variables). There are some limitations, one being the number of observations within CLASS groups. However, the reduced amount of SAS code required makes the IDGROUP a useful alternative to other conventional methods for transposing many variables in a data set to wide format

TRADEMARK CITATION

CONTACT INFORMATION

Comments and suggestions are welcome and can be sent to the authors via e-mail ...

John King ouachitaclinicaldataservices@gmail.com

Mike Zdeb msz03@albany.edu