Paper 517-2013

Increase Your Productivity by Doing Less

Arthur S. Tabachneck, Ph.D., myQNA, Inc., Thornhill, Ontario Canada
Xia Ke Shan, Chinese Financial Electrical Company, Beijing, China
Robert Virgile, Robert Virgile Associates, Inc., Lexington, MA
Joe Whitehurst, High Impact Technologies, Atlanta GA

ABSTRACT

Using a keep dataset option when declaring a data option has mixed results with various SAS® procedures. It might have no observable effect when running PROC MEANS or PROC FREQ but, if your datasets have many variables, it could drastically reduce the time required to run some procs like PROC SORT and PROC TRANSPOSE. This paper describes a fairly simple macro that could easily be modified to use with any proc that defines which variables should be kept and, as a result, make your programs run 12 to 15 times faster.

THE PROBLEM

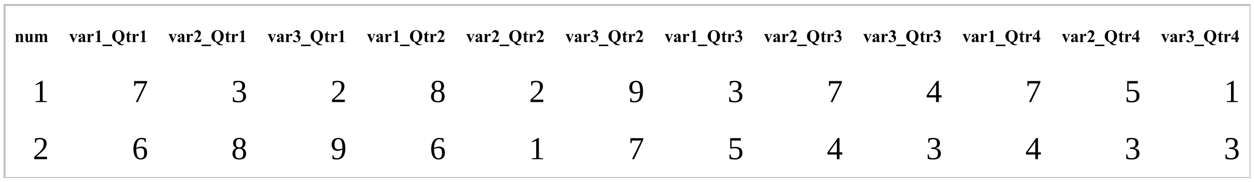
In a tutorial presented at the 1996 SUGI meeting, Bruce Gilsen presented a number of things SAS programmers can do to improve the efficiency of their programs. One of those methods was to only keep the variables which will be necessary for a particular analysis. Unfortunately, the efficiency was only mentioned as the sixth out of nine methods, and was only mentioned without showing how effective the efficiency could be.

Have you ever considered how much time you could save by excluding irrelevant data from your tasks and analyses? Even with the present paper's authors' over 100 years combined experience as SAS users, and the fact that we are all quite familiar with the technique, most of us seldomly use it. Additionally, we have all wasted time running a procedure that required a sorted dataset, only to discover either that we had failed to sort the incoming data, or that the data weren't sorted because the dataset's sort flag had contained an incorrect value.

How much time is saved by dropping irrelevant variables? Presume that you had to run a PROC TRANSPOSE on a dataset like the one that would be created by the following code, and that you had to produce a file like the one shown in Example 1 (we didn't think you needed to see the results for all 10,000 idnums thus only showed the first two):

```
data have (drop=months i);
  array var(*) var1-var1000;
  do idnum=1 to 10000;
    date="01dec2010"d;
    do months=3 to 12 by 3;
      date=intnx('month',date,3);
      do i=1 to 1000;
        var(i) = ceil( 9*ranuni(123) );
      end;
      output;
    end;
  end;
run;
```

Example 1



num	var1_Qtr1	var2_Qtr1	var3_Qtr1	var1_Qtr2	var2_Qtr2	var3_Qtr2	var1_Qtr3	var2_Qtr3	var3_Qtr3	var1_Qtr4	var2_Qtr4	var3_Qtr4
1	7	3	2	8	2	9	3	7	4	7	5	1
2	6	8	9	6	1	7	5	4	3	4	3	3

Many SAS users would use code like the following in order to accomplish the task:

```
proc sort data=have out=need;
  by idnum date;
run;

PROC TRANSPOSE data=need out=tall;
  by idnum date;
  var var1-var3;
run;

PROC TRANSPOSE data=tall out=want (drop=_:) delimiter=_Qtr;
  by idnum;
  var col1;
  id _name_ date;
  format date Qtr1.;
run;
```

If you aren't familiar with PROC TRANSPOSE, you may be wondering why the procedure needed to be run twice in order to accomplish the task. The procedure had to be run first in order to convert the wide file into a tall file that would have one record for every *by, id* and *var* variable combination. Then, the procedure had to be run a second time in, on the tall file, in order to produce the desired result.

However, significant time was lost in the first step, when we ran PROC SORT. The following code would have run almost 15 times faster:

```
proc sort data=have (keep=idnum date var1-var3) out=need noequals;
  by idnum date;
run;
```

And, similarly, the first PROC TRANSPOSE run would have taken less than one-sixth the time to run if only the necessary variables had been kept during the call to PROC SORT.

THE SOLUTION

Since procedures like PROC TRANSPOSE inherently define all of the variables that will be required, we wrote a fairly simple SAS macro that lets you indicate whether the data has to be sorted and, regardless, ensures that only the relevant variables are used throughout the process. The following two calls of the macro would produce the desired result, capitalize on the above mentioned time savings, and require less code:

```
%transpose( data=have, out=tall, by=idnum date,
            var=var1-var3, sort=yes)

%transpose( data=tall, out=want (drop=_:),
            delimiter=_Qtr, by=idnum, var=col1,
            id=_name_  date,  format=date Qtr1.)
```

The two calls would have caused the macro to write and submit the two SAS programs shown as Program 1 and Program 2 on the following page:

Program 1

```
proc sort data=have(keep=idnum date var1-var3) out=_temp noequals;
run;

proc transpose data=_temp  out=tall;
  by idnum date;
  var var1-var3;
run;
```

Program 2

```
Proc transpose data=tall (keep=idnum date _name_ col1)
    delimiter=_Qtr out=want (drop=_:);
  by idnum;
  id _name_  date;
  var col1;
  format date Qtr1.;
run;
```

We included a *noequals* option in the PROC SORT call because of a suggestion that Philip Mason made in a 2001 SUGI tutorial. Philip suggested that the order of records within by groups seldom has to be maintained, that the order will be maintained unless the option is specified, and that he has typically saved approximately five perccent of his processing time by including the option. Since order within by groups is totally irrelevant when one is using PROC TRANSPOSE, we evaluated the option's effect on file sizes of 250,000 and 2.5 million records, when there were three variables and only one *by* variable. The savings were 12.5 percent on the smaller file and over 14 percent on the larger file.

Philip also suggested that using the *tagsort* option could save additional processing time, as could using views and compressed datasets, but that such use could also be costly dependent upon the size of one's data and the number of keys specified in a by statement. As such, we decided to only include the *noequals* option, but allow users to specify additional options in a *sort_options* parameter.

You could achieve the same processing time savings by writing the above code yourself, but the point of the macro is that you necessarily already provide the needed information whenever you run PROC TRANSPOSE. Since such macros are easy to write, and only have to be written once, they result in your having to type less code, lower the chance of your making typographical errors or omissions, and have the added benefit of not requiring you to remember to include the various options. Further, since the macro uses *named parameters,* any settings that you typically use can be set as default values, thus require even less typing.

With the macro, you always get to benefit from the above mentioned processing efficiencies and get to do it by typing less code and having less to remember.

THE %TRANSPOSE MACRO

The %transpose macro was designed to provide the above mentioned solution. Basically, the program simply creates and runs the code that actually accomplishes the task. The macro's named parameters are the various options and statements that one might use when actually running the procedure, along with two additional parameters that define whether the dataset should be sorted and the options that should be applied. Whether the parameters are actually included in the procedure call is controlled by a series of %if statements.

Named Parameters. Named parameters were used so that (1) default values could be assigned and (2) the various parameters would only have to be specified when values other than the default values are required. When calling the macro, the default values will be used unless you specify the desired value. Thus, if you wanted the macro to typically sort your data and result in a file called "work.transposed_file", you would modify the parameters by specifying them in the macro definition. Example:

```
%macro transpose(data=, out=transposed_file,  by=,  prefix=, suffix=, label=,   let=,  name=,
      var=, id=,     idlabel=,  format=, delimiter=, copy=, sort=yes,     sort_options=  );
```

All of the parameters, except for the *sort* and *sort_options* parameters, reflect the various options and statements that can be used with PROC TRANSPOSE, and their descriptions can be found in the PROC TRANSPOSE documentation. *Sort* is the parameter you would use to indicate whether the input dataset must first be sorted before the data is transposed. Possible values are YES or NO. If left null, the macro code will set this parameter to equal NO. The *sort_options* parameter is the parameter you can use to incorporate any additional sort options which you think will reduce your overall processing time (e.g., tagsort, presorted and/or force).

CONCLUSION

The purpose of the present paper was to describe and share a SAS macro that could be used to accomplish data transpositions faster and easier than can be accomplished with PROC TRANSPOSE. The principal benefits include: (1) the automatic inclusion of a *keep* dataset option; (2) the ability to sort and transpose in one step; (3) the automatic inclusion of a *noequals* option for all sorts; and (4) the use of named parameters so that one doesn't have to distinguish between options and statements and frequently used values can be saved in the macro definition.

In addition to satisfying the purpose for which it was designed, the macro can be used as a template for achieving similar benefits with other SAS procedures.

DISCLAIMER

The contents of this paper are the work of the authors and do not necessarily represent the opinions, practices or recommendations of the authors' organizations.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Arthur Tabachneck, Ph.D., President
myQNA, Inc.
Thornhill, ON Canada
E-mail: atabachneck@gmail.com

Xia Ke Shan
Chinese Financial Electrical Company
Beijing, China
E-mail: xiakeshan@yahoo.com.cn

Robert Virgile
Robert Virgile Associates, Inc.
Lexington, MA
E-mail: rvirgile@verizon.net

Joe Whitehurst
High Impact Technologies
Atlanta GA
E-mail: joewhitehurst@gmail.com

REFERENCES

SAS PROGRAM EFFICIENCY FOR BEGINNERS, Gilsen, Bruce, SUGI 1996 Proceedings,
http://www.sascommunity.org/sugi/SUGI96/Sugi-96-53%20Gilsen.pdf

SAS TIPS I LEARNT WHILE AT OXFORD, Mason, Philip, SUGI 2001 Proceedings,
http://www2.sas.com/proceedings/sugi26/p020-26.pdf

SAS Institute Inc. 2009. *Base SAS® 9.2 Help and Documentation*. Cary, NC, USA.

APPENDIX I

THE %TRANSPOSE MACRO

```
/**This program runs PROC TRANSPOSE and includes a keep statement to limit the data
   that needs to be read and lets you indicate whether the data should be sorted
  *
  * AUTHORS: Arthur Tabachneck, Xia Ke Shan, Robert Virgile and Joe Whitehurst
  * DATE: February 11, 2013

  Parameter Descriptions: All of the parameters, except for two, reflect the various
  options and statements that can be used with PROC TRANSPOSE and their descriptions
  can be found in the PROC TRANSPOSE documentation.  The two exceptions are:

  *sort* the parameter you would use to indicate whether the input dataset must be
  sorted before the data is transposed.  Possible values are: YES or NO

  *sort_options* The noequals option will be used for all sorts, but this parameter
  can be used to specify any other options you want used (e.g., presorted or tagsort)
*/

%macro transpose(data=, out=, by=, prefix=, suffix=, label=, let=,  name=, var=,
                 id=,  idlabel=, format=,  delimiter=, copy=,  sort=, sort_options= );

/*Populate var parameter in the event it has a null value*/
  data _temp;
    set &data. (obs=1 drop=&by. &id. &copy.);
  run;

  %if %length(&var.) eq 0 %then %do;
    proc sql noprint;
      select name into :var separated by " "
        from dictionary.columns
          where libname="WORK" and memname="_TEMP" and type="num" ;
    quit;
  %end;

/*If sort parameter has a value of YES, create a sorted temporary data file*/
  %if %sysfunc(upcase("&sort.")) eq "YES" %then %do;
    proc sort data=&data (keep=&by. &id. &var. &copy.) out=_temp &sort_options. noequals;
      by &by;
    run;
    %let data=_temp;
  %end;

/*Run the procedure*/
  proc transpose data=&data. (keep=&by. &id. &var. &copy.)
    %if %length(&delimiter.) gt 0 %then delimiter=&delimiter.;
    %if %length(&label.) gt 0 %then label=&label.;
    %if %length(&let.) gt 0 %then let;
    %if %length(&name.) gt 0 %then name=&name.;
    %if %length(&out.) gt 0 %then out=&out.;
    %if %length(&prefix) gt 0 %then prefix=&prefix.;
    %if %length(&suffix) gt 0 %then suffix=&suffix.;;
    %if %length(&by.) gt 0 %then by &by.;;
    %if %length(&copy.) gt 0 %then copy &copy.;;
    %if %length(&id.) gt 0 %then id &id.;;
    %if %length(&idlabel.) gt 0 %then idlabel &idlabel.;;
    %if %length(&var.) gt 0 %then var &var.;;
    %if %length(&format.) gt 0 %then format &format.;;
  run;

/*Delete temporary file*/
  proc delete data=work._temp;
  run;
%mend transpose;
```