Paper 081-29

Using Templates Created by the SAS/STAT® Procedures

Yanhong Huang, Ph.D. UMDNJ, Newark, NJ
Jianming He, Solucient, LLC. , Berkeley Heights, NJ

ABSTRACT

SAS procedures provide a large quantity of templates for users to choose. This function has been enhanced by ODS. If programmers are familiar with these templates, they can save a lot of energy and time. By contrasting the method of constructing a template with the method of borrowing an existing template, this paper will focus on how to utilize the templates produced by SAS procedures, especially SAS/STAT procedures, to achieve programming efficiency.

INTRODUCTION

Most institutes with SAS/BASE also have licenses for SAS/STAT or other SAS products. However, not all SAS programmers can get the benefits from the templates produced by these products. The template in SAS software determines how the data are formatted and shown. Since SAS version 7.0, Output Delivery System (ODS) provides tools to present, extract and modify those templates created by SAS procedures. For instance, PROC TEMPLATE can define the fonts, colors, headers, footer etc. If a programmer is familiar enough with both the procedures and the templates created by ODS, tremendous time and energy can often be saved.

This paper focuses on how to use the templates produced by SAS procedures, primarily on the contents. We are going to walk through two scenarios to show it is more convenient to borrow an existing template than to construct a new one.

SCENARIO 1: USING PROC FREQ TO CREATE A TEMPLATE FOR PROC TRANSPOSE

Suppose we have a dataset named “smoker” with three fields; country, year and counts, and we want to transpose it to put each year as a column and list them in ascending order. Generally we sort the dataset and transpose it by setting the “year” variable as id.

```
data smoker;
       input  county $7. year  counts;
       cards;
       Essex   2001  2000
       Essex   2002  2009
       Essex   2004  2004
       Dover   2000  4179
       Dover   2003  4194
;

proc sort data=smoker; by county year;
proc transpose data =smoker out=a(drop=_name_) prefix=year;
        by county;  id year; var counts;

proc print data =a noobs;
run;
```

The output is as follows.

```
county    year2000    year2003    year2001    year2002    year2004
Dover       4179        4194           .           .           .
Essex          .           .        2000        2009        2004
```

Before transposing, we try to sort the dataset to make sure that the “year” column is in ascending order. To our surprise, the above codes do not produce such a sequence. SAS chooses the first non-missing cell as the first column, and so on. In the above example, while the variable county equals “Dover”, the years of 2001, 2002 and 2004 are all missing. Therefore, the sequence appears to be year2000, year2003, etc after transposing. Notably, year2003 jumps in front of year2001.

There are two ways to handle this problem — “before the PROC TRANSPOSE” approach or “after PROC TRANSPOSE” approach. For the second approach, we may use ‘retain’ in the data step, or using PROC SQL statement to control output column sequence. The shortcoming of this approach is that you need to know how many columns there are and in what sequence you are going to output them. For the approach of “before PROC

TRANSPOSE", we need to create a template in which all counties have non-missing year values. Whether the variable "counts" is missing does not matter. There are two ways to do this. One is to use a data step, i.e., we need to write codes to insert some records into the smoker dataset. For instance, if we have added the following records into the dataset smoker, the problem is solved.

```
Essex 2000
Essex 2003
Dover 2001
Dover 2002
Dover 2004
```

Using a data step to implement this requires the following steps:

1. To know the content of all counties and years and to get all possible permutations of counties and years.
2. To drop those columns that exist the original dataset.
3. To append this new complimentary dataset to the original one.

Basically, it takes a while to code using the above approach. To implement the first step, the code is as follows.

```
proc sql;
       select count(distinct county) into:county from smoker;
       select distinct county into:clist separated by ' ' from smoker;
       select count(distinct year) into: year from smoker;
       select distinct year into: ylist separated by ' ' from smoker;
quit;

data a;
      do a=1 to &year by 1;
             do b=1 to &county by 1;
                    output;
             end;
       end;
run;

%macro first;
data out1;
       set a;
        %do i=1 %to &year ;
                   %let a&i=%scan(&ylist, &i);
                   if a=&i then year="&&a&i";
        %end;
        %do j=1 %to &county ;
                   %let b&j=%scan(&clist, &j);
                   if b=&j then county="&&b&j";
        %end;
drop a b ;
run;
%mend;

%first;
```

For the above so-called data step approach, we need to control the number of possible different years. Also we need to create some macro variables. Since we construct the template step by step, we name it as the "constructing" method.

There is neat solution for this. If we are familiar with PROC FREQ, we may use the SPARSE option to create the template in a more efficient way.

```
proc freq data=smoker noprint;
       tables county*year / out=out1 (keep=county year) sparse;
run;
```

Those codes are doing the same job as using the data step approach.

The rest of the story is simple.

```
data new;
  merge smoker out1;
  by county year;

proc transpose data =new out=b(drop=_name_) prefix=year;
       by county;  id year; var counts;

proc print data =b noobs;
run;
```

The result is as follows:

```
county    year2000    year2001    year2002    year2003    year2004
Dover       4179           .           .        4194           .
Essex          .        2000        2009           .        2004
```

The PROC FREQ procedure is primarily a procedure that delivers one-way to n-way frequency, cross tabulations tables and other statistical testing. In this example we did not apply any primary functionality of PROC FREQ. However, the byproduct of PROC FREQ helps out. By borrowing the template created by PROC FREQ with SPARSE option, we can more efficiently program than with the "constructing method".

Without ODS, the tables output by SAS procedures are limited for direct use. Since SAS version 7.0, ODS has been a powerful tool for us to explore the SAS/STAT procedures. It outputs most of the statistical tables of SAS /STAT procedures depending on the specification. It liberates people of the painful work of retyping statistical results.

The syntax for checking what ODS delivers is quite simple. Before the procedure, we need to specify "ODS trace on". Then we check the .log file and see what results we got from there. After the procedure is finished, we set " ODS trace off". The next step is to extract the wanted dataset. The syntax is "ODS OUTPUT data-set-definition(s);" Please note, we need to quit all previous procedures with either "run;" or "quit;", depending on the context. Otherwise, the expected dataset will not be delivered.

Here we are going to combine the function of ODS and some SAS/STAT procedures to tackle a seemingly difficult programming problem.

Scenario 2: Create interactive terms for linear stepwise regression

PROC LOGISTIC provides a convenient solution for the stepwise regression considering interactive terms. People can chose the level of interactive terms by choosing the number after sign '@' and choosing variable by sign '|'.

For linear regression, it is a little difficult because PROC REG does not allow us to create interactive terms in the model directly though it can do stepwise model selection. PROC GLM can do the linear regression for interactive terms. However, it does not provide stepwise solution. We need a good strategy to find a way to automate the creation of interactive terms.

In order to construct those interactive terms, we need to know how many of them we are going to build. If we have *n* regressors, for two regressors interaction, the number is C_n^2 (the number of ways of picking 2 unordered outcomes from n possibilities), for three repressors, the number is C_n^3 …Since generally it is difficult to interpret if the interactive variables are more than three, here we are going to create a two interactive terms by ODS and PROC PLAN as an example. Different level combination follows the same approach. PROC PLAN is a procedure for design experiment. Here we use it to make combination numbers. Since version 9.0, we may also check ALLPERM, RANPERM, RANPERK etc call routines to display difference combinations.

For example we have a dataset called "smoker" which has 10 independent variables, including gender, age, region, etc. For 10 variables, we have 45 different two variables interactive terms. We created as macro variable with all the independent variables separated by spaces.

```
ods output Plan=combinations;

proc plan;
       factors Block=45 ordered
              Treat=2 of 10 comb;
run;

%macro a;
```

```
        data new;
                set combinations;
                j=_n_;
                select (treat1);
                        %do i=1 %to 11;
                                %let v&i=%scan(&list, &i);
                                when (&i) var1="&&v&i";
                        %end;
                otherwise;
                end;
                select (treat2);
                        %do i=1 %to 11;
                                %let v=%scan(&list, &i);
                                when (&i)  var2="&&v&i";
                        %end;
                otherwise;
                end;
        newvar=compress(var1||var2);
        comp=compress(var1||var2||'='||var1||'*'||var2);
        run;

        proc sql;
                select distinct master into: comp separated by '; '  from new  ;
        quit ;

        data smokernew;
                set smoker;
                &comp
        run;

%mend;
%a;

proc sql;
        select distinct newvar into : list2 separated by ' ' from new;

proc reg data=smokenew;
        model  dependentvar=  &list2  /selection=backward ;
quit ;
```

The above macro produces a column called “NEWVAR” containing new variables two interactive terms, we also have a column called “comp” with the formula to compute those variables. Still, the above is a “constructing” method because we construct both the name of the variables and formula for the two interactive terms.

Where can we find an existing template for this problem? We have seen two interactive terms from either PROC LOGISTIC or PROC GLM, we can borrow the template created by those models although we are not going to run the logistic model. The strategy is to select one categorical variable (for instance, we select “region”) as a dependent variable and use the pool of variables, which we want, to create the two factor interactive terms as the regressors. This model by itself does not make any sense since the dependent variable also appears as a regressor. For our purpose, we are just interested in the template which PROC LOGISTIC /GLM created. We create a macro variable “secl” with all independent variables separated by ‘|’ . The code is as follows.

```
/***** Second solution ****/
%Macro second;

ods output parameterestimates=parameterestimates;

proc logistic data=smoker;
        model  region  = &sec1 @2;
        run;

proc sql ;
        select distinct variable into : varlist separated by ' ' from
        parameterestimates
        where variable~='Intercept' and index(variable, '*')>0;

        select count(distinct variable) into: ct from parameterestimates
        where variable~='Intercept' and index(variable, '*')>0;

%put &varlist;
```

```
data smokenew;
       set smoker;
       %do i= 1 %to &ct;
       var&i  =%scan(&varlist, &i, ' ');
       label var&i="%unquote(%scan(&varlist, &i, ' ' ))";
       %end;
run ;
proc reg data=smokenew ;
       model dependentvar = &list
              %do i=1 %to &ct;
                     var&i
              %end ;
       /selection=backward;
quit ;
%mend;

%second;
```

The statistical result from the PROC LOGISTIC is not our focus. We are just interested in the template it created. Clearly, it takes more time constructing a new template than using an existing one in the above two cases. The second solution is preferable because it follows a method not to construct but to borrow a template somewhere else.

CONCLUSION

In summary, it is good practice for SAS programmers to be familiar with all SAS procedures, even when not doing statistical analysis. The evolution and development of SAS gives more convenient tools for both research and programming, it is really worthwhile to dig around every procedure to know what exactly is available. Those procedures are not only applicable for the specific statistical purposes, but also a gold mine for efficient SAS programming.

REFERENCES

SAS Institute Inc. (1990), SAS/STAT user's Guide, Version 6, Fourth Edition, Cary, NC: SAS institute Inc.

ACKNOWLEDGMENTS

Thanks Barry Green and Michael Mace for proofreading.

CONTACT INFORMATION

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

Jianming He
Solucient, LLC
Connell Corporate Center
One Connell Drive, Suite 1000
Berkeley Heights, NJ 07922
(734) 669-7835 (O)
(908) 665-1764 (F)
jhe@solucient.com

Yanhong Huang, Ph.D.
Cardiovascular Research Institute
University of Medicine and Dentistry of New Jersey
South orange 185,
Newark, NJ 07101
(973) 972-5530 (O)
huangya@umdnj.edu