A SAS® Macro for Transposing Large Data sets
Chen Li, Educational Testing Service, Princeton, NJ

ABSTRACT
When facing the task of transposing a large vertical data set with multiple records per person into the corresponding horizontal data set with a single record per person, we commonly use the SAS® methods of DATA Step and PROC TRANSPOSE. For each particular data set, it is important to consider the unique requirements and specifications for this step of the data cleaning process. There are often many variables to be transposed and a differing amount of records for each person. Other details that need to be considered are column positions and headings, variable types and various design specifications. These obstacles can make this work tedious, repetitive, time-consuming and prone to error. Thus, a question naturally arises from this process, that is, can we generalize the above procedure by creating a SAS® macro to accomplish this objective? This paper presents a SAS® macro that aims to provide a solution to this problem. One application in which simulated test data be organized by student will be used to demonstrate this macro. In addition, other possible applications of this macro will be discussed.

INTRODUCTION
Transposing or rotating a data set is one of the most frequent operations within the data cleaning process. The rotation has to be accomplished separately for each group of observations for the same individual based on one or more key variables. There are a few approaches to accomplish this. The most common approach to rotate the data set is using arrays in a DATA step. An alternate approach is to rotate the data set using the TRANSPOSE procedure. The PROC TRANSPOSE code can be much simpler than the corresponding code in the DATA step, however it also has an important disadvantage. That is only one variable can be transposed at a time. If we want to transpose multiple variables, we need to merge the transposed results after multiple PROC TRANSPOSE steps. The procedure also creates additional variables that may be considered extraneous.

This paper will present a short piece of code using a macro to transpose a large data set that has many observations into a corresponding "wide" data set with many variables. This code can transpose multiple variables at a time and eliminates the need to merge any data sets. The code also avoids creating extraneous additional variables in the new data set.

This code is useful for working with testing and coursework data received from various institutions. The data in this paper are for demonstration purposes only and do not represent actual data. Typically, a data set may have more than thirty variables but the sample data used here are smaller and only include a few variables: a student identifier, test year, test date and three scores. The code can easily be expanded to larger data sets. The image below shows a sample of the data used to demonstrate the utility of the program. The data are illustrative and not actual.

Fig 1. Sample Input Data set
A typical input data set for this program includes the unique identifier (Student_ID), background information like test year, test date and grade, and scores within many different content areas. Sometimes duplicated records are wrongly input into the data set. To ensure validity and reliability is properly assessed, it is important to remove duplicate cases before transposing the data set. Please see Figure 2.

![Fig 2. Code to Check for Duplicates](image)

```
proc sort data=NS.NESUG_demo_data noduprecs;
  by Student_ID testyear testdate;
run;
```

THE PRE-MACRO VERSION OF THE SOURCE CODE FOR TRANSPOSING

Since the number of records for each person is not constant, a variable named “count” is created for students’ ID numbers. After sorting by student IDs in the last step, values can now be assigned to the count variable. When the program reads a new student ID, the count is set to “1”, and for any repeated ID, the count is increased by 1. Thus, we can record for keep track of which rows are associated with each student and then we can easily use PROC MEANS to find out the maximum number of records for each student. For the example here, the maximum number of records of all students is 11. The code for this process can be found in Figure 3.

![Fig 3. Code to Count the Number of Records per Student ID](image)

```
data NS.NESUG_demo_data;
  set NS.NESUG_demo_data;
  count + 1;
  if first.Student_ID then count = 1;
  by Student_ID;
run;
proc means data=NS.NESUG_demo_data MAX;
  var count;
run;
```

USING THE MACRO TO TRANSPOSE THE DATA SET

When transposing the data, we must be careful with the data formats. Data will be lost if they are read into incorrect formats. Hence, before assigning values to each variable, we have to define the format types and make sure these are consistent with the format types of the original variables.

Since the maximum number of records for each student is known now, which is denoted by N, we can create \(N \times \text{(initial number of variables} - 1\text{)}\) new variables to store the N sets of information for each student. We start with reading in the first set of information for each student. Then, we add a new record which will append the second set of information. The third line will then append the third set of information. This process is continued iteratively until the final line for the student contains all N sets of information.

After completing this process for one student, it is important to remember to make sure that all variables are set to missing before beginning with the records for the next student. If not, there is the risk of carrying over the previous student’s information to the next student, because every time we write a new row, we retain the former data first. This is especially crucial in cases where the previous student may have more sets of data than the following student. For example, imagine that a student has nine sets of test records and the following student only has four set of records. In this case, the 5th to 9th set of values for the previous student will be written incorrectly to the current student’s records if this data had not been correctly cleaned.

The syntax in Figure 4 shows the main steps for transposing the data in macro form:
The following image in Figure 5 shows a sample of data after the macro is run. We can see that the shape of the data set is triangular for each student.

Fig 5. Sample Output Data set after the Macro TRANSPOSE

Now we have the complete information for each student in the last row that corresponds with each student ID. The only thing left is to delete the extraneous records from the data set and the count variable, and check for remaining duplicates in the data set. The below syntax in Figure 6 takes care of this final step.

```sas
%let maxcount=11; /*The maximum number of records in this example is 11.*/
%macro Transpose();
data NS.NESUG_demo_data;
set NS.NESUG_demo_data;
   %do i = 1 %to &maxcount.;
      format testyear_&i. $9.;
      format testdate_&i. MMDDYY10.;
      format score1_&i. 3.;
      format score2_&i. 3.;
      format score3_&i. 1.;
   %end;
   %do i=1 %to &maxcount.;
      if count=&i. then do;
         retain testyear_1-testyear_&i. testdate_1-testdate_&i. score1_1-score1_&i. score2_1-score2_&i. score3_1-score3_&i.;
         testyear_&i.=testyear;
         testdate_&i.=testdate;
         score1_&i.=score1;
         score2_&i.=score2;
         score3_&i.=score3;
      %do j=&i.+1 %to &maxcount.;
         testyear_&j.=.;
         testdate_&j.=.;
         score1_&j.=.;
         score2_&j.=.;
         score3_&j.=.;
      %end;
      end;
   %end;
   drop testyear testdate score1 score2 score3;
run;
%mend Transpose;
```
CONCLUSIONS
This paper introduces how to use a macro to transpose a large data set. A simple example of what this macro can accomplish has been shown. The code and principle used in this paper can easily be expanded and applied to large data sets. The methods introduced here have numerous advantages compared to the traditional methods discussed in the introduction of this paper. With this proposed method, multiple variables can be transposed at the same time and variable names can be easily customized. Therefore, this syntax can be used to help make the data cleaning process more efficient for those working with data sets of all types across industries.

REFERENCES:
SAS Programming II: Data Manipulation Techniques

ACKNOWLEDGMENTS
The author would like to thank Ted Blew, Steven Holtzman, Bruce Kaplan, Ed Kulick, Jonathan Steinberg and Sailesh Vezzu for their support and assistance in proofreading this paper.

CONTACT INFORMATION
Your comments and questions are appreciated and encouraged. Please contact the author at:
Chen Li
Assistant Research Data Analyst
Center for Data Analysis Research
Educational Testing Service
Rosedale Road, MS 20T
Princeton, NJ 08541
(609) 734-5993
cli@ets.org

TRADEMARKS
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are registered trademarks or trademarks of their respective companies.