Paper 039-2012

One Flip, Two Flip, Fat File, Flat File

Ethan Miller, SRI International, Menlo Park, Ca

ABSTRACT

Tired of getting data sets with multiple records per ID when you need a flat file for analysis? Tired of struggling with one-too-many merges? With just a small investment of time you too can easily create a flat file with one record per ID. Through the use of PROC SORT, PROC TRANSPOSE, SAS® MACRO, and the DATA step, this paper examines how to create a flat file with customized variable names and labels for multiple time intervals. Through the use of SAS MACRO, the code shown in this paper creates a flat file, and your friends will be amazed at how easy it is. This paper was written for those with beginner-to-intermediate skills; the code was written using SAS® 9.2 on a Windows operating system.

INTRODUCTION

Many times, it is desirable to have a flat file for the purpose of doing an analysis or to avoid a many-to-many merge. The macro discussed in this paper creates a flat file from a data set that contains multiple observations per entity, where each observation is for a different interval. The flat file that it creates contains one record per entity where the variables associated with different intervals are renamed to correspond to those intervals. The interval could be time, treatment number, or any other event that happens more than once. It could also be distinct categories, such as type of service.

A SIMPLE EXAMPLE

The goal is to take the data in Table 1 and produce the data structure of Table 2. (Note: The code to generate the data in Table 1 is included in Appendix A.) Notice that Table 1 has multiple observations for a given entity (in this case, student ID), with each record having an interval variable and two score variables. Table 2 has one record per entity, with two score variables for every interval.

Table 1: Data with Multiple Records per Entity

Student ID	Test Interval	Test Score 1	Test Score 2
1	1	0.822	0.43738
1	2	0.93793	0.00613
1	3	0.8349	0.53064
1	4	0.59342	0.91467
1	5	0.36768	0.99229
2	1	0.43057	0.97496
2	2	0.00666	0.1605

Table 2: Data with One Record per Entity

Student ID	Interval 1 Test Score 1	Interval 1 Test Score 2	Interval 2 Test Score 1	Interval 2 Test Score 2	Interval 3 Test Score 1	…
1	0.822	0.43738	0.93793	0.00613	0.8349	…
2	0.43057	0.97496	0.00666	0.1605	.	…

To begin the process, a sort is performed. This sort ***must*** include a combination of BY variables that will make each observation in the data set unique. In our example, we use ID and INTERVAL as the BY variables.

```
proc sort data = d;
  by id interval;
run;
```

Next, the first PROC TRANSPOSE is run to transform the data. The BY variables will be the same as the sort. The variables in the VAR statement will be the variables that you want to analyze as the interval changes. In this example, it is the two test score variables.

```
proc transpose data = d out = dt1_1;
  by id interval;
  var s1 s2;
run;
```

The result is a data set with one observation per BY variable combination (i.e., ID and INTERVAL) for every variable included in the VAR statement (i.e., test scores contained in S1 and S2).

In Table 3 below, the first observation from Table1 became the first two observations in Table 3. The variables _NAME_ ,COL1, and _LABEL_ are created by PROC TRANSPOSE:

- _NAME_ contains the names of the variables included in the VAR statement.
- COL1 contains the value of the variable specified in the _NAME_ column.
- _LABEL_ contains the labels of the variables included in the VAR statement.

A COL variable is created for every BY variable combination. In order for the code presented here to work, you must have a BY variable combination that is unique so that only one column is produced.

Table 3: After One Flip

ID	INTERVAL	_NAME_	COL1	_LABEL_
1	1	s1	0.822	Test Score 1
1	1	s2	0.43738	Test Score 2
1	2	s1	0.93793	Test Score 1
1	2	s2	0.00613	Test Score 2
1	3	s1	0.8349	Test Score 1
1	3	s2	0.53064	Test Score 2
1	4	s1	0.59342	Test Score 1
1	4	s2	0.91467	Test Score 2
1	5	s1	0.36768	Test Score 1
1	5	s2	0.99229	Test Score 2
2	1	s1	0.43057	Test Score 1
2	1	s2	0.97496	Test Score 2
2	2	s1	0.00666	Test Score 1
2	2	s2	0.1605	Test Score 2

The next step is to use the _NAME_ and _LABEL_ variables to create new variable names and labels for our flat file. This is accomplished using a DATA step:

```
data dt2_1 ;
  length nvarname $32
         nvarlabel $200;
  set dt1_1;

  nvarname = trim(left(_name_)) || "_" || trim(left(interval));
  nvarlabel = "Interval " || trim(left(interval)) || " " || trim(left(_label_));
run;
```

This DATA step creates two variables: NVARLABEL (which will become the label in our final data set) and NVARNAME (which will become the variable name in our final data set).

NVARNAME is a concatenation of the old variable name with an underscore and an interval number added as a suffix. For example, in the final data set, S2 for interval 3 will have a variable name of “S2_3.”

The same technique is used for the label, which becomes a concatenation of the word “Interval” followed by the appropriate interval number with the original label placed at the end. For example, in the final data set, S2 for interval 3 will have a label of “Interval 3 Test Score 2.”

Table 4: New Variable Names and Labels Added to Data Set

NVARNAME	NVARLABEL	Student ID	Test Interval	Name of Original Variable	Label of Original Variable	COL1
s1_1	Interval 1 Test Score 1	1	1	s1	Test Score 1	0.822
s2_1	Interval 1 Test Score 2	1	1	s2	Test Score 2	0.4374
s1_2	Interval 2 Test Score 1	1	2	s1	Test Score 1	0.9379
…						

Now, the final transpose is performed to produce our flat file, and following code is used:

```
proc transpose data = dt2_1 out = done;
  by id;
  idlabel nvarlabel;
  id nvarname;
  var col1;
run;
```

Notice that this time we have only one BY variable (ID) on the TRANSPOSE. This will give us one record per ID. The IDLABEL option is used to assign the value of NVARLABEL as the label to its corresponding variable. Likewise, the ID option assigns the value of the variable NVARNAME to be the variable name. Now the flat file is complete.

Table 5: Two Flip, Fat File, Flat File (or Data with One Record per Entity)

Student ID	Interval 1 Test Score 1	Interval 1 Test Score 2	Interval 2 Test Score 1	Interval 2 Test Score 2	Interval 3 Test Score 1	…
1	0.822	0.43738	0.93793	0.00613	0.8349	…
2	0.43057	0.97496	0.00666	0.1605	.	…

A NOT-QUITE-SO-SIMPLE EXAMPLE— TRANSPOSING A DATA SET WITH CHARACTER AND NUMERIC VARIABLES

Sometimes it is desirable to create a flat file that contains both character and numeric data. This process is a little more complicated. The reason is that after the first PROC TRANSPOSE, the COL1 variable will contain strings making COL1 a character variable. To get around this, the use of a MACRO and MACRO variables are employed.

The MACRO FLIPTHATSTUFF loops through and performs the double transpose twice, once for the numeric variables and once for the character variables.

```
   %macro flipthatstuff;
```

❶The first step in this process is to identify which variables are character and which are numeric. This information is attained by using a PROC CONTENTS using the OUT option.

```
❶  proc contents data = d
     out = dcont noprint;
   run;
```

❷PROC SQL is used to create two MACRO variables from the contents data set: TYPE1 and TYPE2. TYPE1 and TYPE2 are space-delimited strings containing all of the numeric and character variables, respectively. At this point, we want to create a MACRO containing the code discussed above.

```
❷  proc sql noprint;
     select name into: type1
       separated by ' '
       from dcont where type =1;
     select name into: type2
       separated by ' '
       from dcont where type =2;
   quit;
   proc sort data  = d;
     by id interval;
   run;
```

❸To loop through the MACRO, a DO UNTIL is used.

```
❸  %let i = 1;
   %do %until (&i = 3);
       proc transpose data = d
         out = dt1_&i;
         by id interval;
         var ❹&&type&i;
       run;
       data dt2_&i;
         length nvarname $32
                nvarlabel $200;
         set dt1_&i ❺
         (where=(_name_ not in
                ("id","interval")));

         nvarname = trim(left(_name_)) ||
           "_" || trim(left(interval));

         nvarlabel =   "Interval " ||
           trim(left(interval)) || " " ||
           trim(left(_label_));
       run;
       proc sort data = dt2_&i;
         by id;
       run;
       proc transpose data = dt2_&i
         out = wide&i;
         by id;
         idlabel nvarlabel;
         id nvarname;
         var col1;
       run;
       proc sort data = wide&i;
         by id;
       run;
       %let i = %eval(&i + 1);
   %end;
```

❹The VAR statement in the first PROC TRANSPOSE now contains the MACRO variables containing the strings of our variable names.

❺The BY variables used here are both numeric; therefore, they will be included when the MACRO variable TYPE1 resolves. As a result of their inclusion in the VAR statement on the first PROC TRANSPOSE, there will be a row of data created for both of them for any BY variable combination. Carrying them through is meaningless to the final data set since ID will not change over time and interval will always be the same within a given interval. The easiest place to drop them seemed to be after the first transpose.

❻Finally, the flat numeric data set is merged with the flat character data set.

```
❻  data flat;
     merge wide1 wide2;
     by id;
   run;
   %mend flipthatstuff;
```

Variables that do not change across your unique BY variables should be dropped from the data set that is passed to the MACRO; otherwise, there will be redundancy. One solution is to drop them when you run the PROC CONTENTS and then merge them back on to the final flat file.

CONCLUSION

One downside of creating the flat file is that it is tougher to report across variables than over observations. For example, you can group over observations to get a mean, but your only option for getting the mean across variables is explicit coding or use of a colon after the root (e.g. mean(of S1_:) - to calculate the mean of all variables with a prefix S1_). A major upside to the syntax to create the flat file with PROC TRANSPOSE is that is data driven. The number of intervals in the data dictates the naming and creation of new variables.

ACKNOWLEDGMENTS

A huge thank you to the Sultan of SAS, Patrick “P-Ditty” Thornton for his never-ending patience, time, and encouragement. Thank you to Cyndi C-Dubs Williamson, Doris Perkins, and Mary McCracken for being Awesome. And to Sue Douglas and Erik Tilanus for accepting my paper.

CONTACT INFORMATION

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

Ethan Miller
SRI International
333 Ravenswood Ave BS159
Menlo Park, Ca 94025
Work Phone: (650) 859-5726
E-mail: ethan.miller@sri.com

APPENDIX A

The code to generate the data used in this paper:

```
data d (drop = k);
  do id = 1 to 10;
     k = round(ranuni(12309)*7);
     do interval = 1 to k;
        s1 = ranuni(1343987);
        s2 = ranuni(8581235);
        output;
     end;
  end;

  label
    s1 = "Test Score 1"
    s2 = "Test Score 2"
    id    = "Student ID"
    Interval = "test interval"
  ;
run;
```