Paper 60

Changing the Shape of Your Data: PROC TRANSPOSE vs. Arrays

Bob Virgile
Robert Virgile Associates, Inc.

Overview

To transpose your data (turning variables into observations or turning observations into variables), you can use either PROC TRANSPOSE or array processing within a DATA step. This tutorial examines basic variations on both methods, comparing features and advantages of each.

A Simple Transposition

Let's begin the analysis by examining a simple situation, where the program should transpose observations into variables. In the original data, each person has 3 observations. In the final version, each person should have just one observation. In the "before" picture, the data are already sorted BY NAME DATE:

NAME	DATE
Amy	Date #A1
Amy	Date #A2
Amy	Date #A3
Bob	Date #B1
Bob	Date #B2
Bob	Date #B3

In the "after" picture, the data will still be sorted by NAME:

NAME	DATE1	DATE2	DATE3
Amy	Date #A1	Date #A2	Date #A3
Bob	Date #B1	Date #B2	Date #B3

The PROC TRANSPOSE program is short and sweet:

```
PROC TRANSPOSE DATA=OLD OUT=NEW
     PREFIX=DATE;
     VAR DATE;
     BY NAME;
```

The PREFIX= option controls the names for the transposed variables (DATE1, DATE2, etc.) Without it, the names of the new variables would be COL1, COL2, etc.
Actually, PROC TRANSPOSE creates an extra variable, _NAME_, indicating the name of the transposed variable. _NAME_ has a value of DATE on both observations. To eliminate the extra variable, modify a portion of the PROC statement:

```
OUT=NEW (DROP=_NAME_)
```

The equivalent DATA step code using arrays could be:

```
DATA NEW (KEEP=NAME DATE1-DATE3);
SET OLD;
BY NAME;
ARRAY DATES {3} DATE1-DATE3;
RETAIN DATE1-DATE3;
IF FIRST.NAME THEN I=1;
ELSE I + 1;
DATES{I} = DATE;
IF LAST.NAME;
```

This program assumes that each NAME has exactly three observations. If a NAME had more, the program would generate an error message when hitting the fourth observation for that NAME. When I=4, this statement encounters an array subscript out of range:

```
DATES{I} = DATE;
```

On the other hand, if a NAME had only two incoming observations, the transposed observation for that NAME would also include the retained value of DATE3 from the previous NAME. The following section, **How Many Variables Are Needed?**, will examine this issue in more detail.

Certainly the DATA step code is longer and more complex than PROC TRANSPOSE. A slightly shorter version would be:

```
DATA NEW (KEEP=NAME DATE1-DATE3);
ARRAY DATES {3} DATE1-DATE3;
DO I=1 TO 3;
   SET OLD;
   DATES{I} = DATE;
END;
```

While unusual, it is perfectly legal to place the SET statement inside a DO loop. In that way, the program eliminates the need to retain variables, since the program never reinitializes DATE1 through DATE3 to missing until it returns to the DATA statement. For each NAME, the DO loop reads all incoming observations, assigning values to DATE1 through DATE3. Then the DATA step outputs the transposed observation, returns to the DATA statement, and resets DATE1 through DATE3 to missing.

The advantages of each method at this point:

- PROC TRANSPOSE uses a simpler program.
- The DATA step has more flexibility. For example, it can perform calculations at the same time it transposes the data. If the incoming data set contains additional variables, the DATA step can calculate the lowest, highest, or mean value of those variables for each NAME, and include them in the transposed data set.

How Many Variables Are Needed?

So far, the DATA step has relied on two major assumptions:

- Each NAME has the same number of observations, and
- We know what that number is.

Often, neither assumption is true. For example, consider a slight change to the incoming data:

NAME	DATE
Amy	Date #A1
Amy	Date #A2
Bob	Date #B1
Bob	Date #B2
Bob	Date #B3

Since Amy now has just two observations, the "after" picture will contain a missing value for DATE3:

NAME	DATE1	DATE2	DATE3
Amy	Date #A1	Date #A2	.
Bob	Date #B1	Date #B2	Date #B3

In that case, PROC TRANSPOSE has a major advantage: the program does not have to change! Automatically, PROC TRANSPOSE creates as many variables as are needed.

The DATA step, on the other hand, requires significant changes. First, an initial step must calculate and capture as a macro variable the largest number of observations for any NAME. Here is one approach:

```
PROC FREQ DATA=OLD ORDER=FREQ;
TABLES NAME / NOPRINT OUT=TEMP;

DATA _NULL_;
SET TEMP;
CALL SYMPUT('N',
     COMPRESS(PUT(COUNT,3.)));
STOP;
```

The output data set TEMP contains one observation per NAME, with the variable COUNT holding the number of observations for that NAME. Because of the ORDER=FREQ option on the PROC statement, the first observation contains the largest value of COUNT. So the DATA step can read that observation, capture COUNT as a macro variable, and then stop executing.

In some cases, the programmer will know this largest number, so the program won't need to calculate it. Most of the time, however, the program will need the extra steps; either the programmer doesn't know the largest number of observations per NAME, or "knowing" is unreliable and the program should double-check. Offsetting this disadvantage of the DATA step is the fact that PROC TRANSPOSE doesn't reveal how many variables it created (except for notes in the SAS log about the number of variables in the output data set). The programmer may need to run a PROC CONTENTS later to discover the structure of the output data set.

Once the program has calculated the largest number of observations per NAME, the subsequent DATA step must still utilize the information. The DATA step's complexity varies depending on which programming style it uses. By continuing to place the SET statement inside a DO loop, the DATA step remains short:

```
DATA NEW (KEEP=NAME DATE1-DATE&N);
ARRAY DATES {&N} DATE1-DATE&N;
DO I=1 TO &N UNTIL (LAST.NAME);
   SET OLD;
   BY NAME;
   DATES{I} = DATE;
END;
```

Because of the DO UNTIL condition, the loop ends after two iterations when AMY has only two incoming observations.

Instead of this DATA step, the program could try to stick with the original, longer DATA step. In that case, allowing for different numbers of observations per NAME leads to more involved programming:

```
DATA NEW (KEEP=NAME DATE1-DATE&N);
SET OLD;
BY NAME;
ARRAY DATES {&N} DATE1-DATE&N;
RETAIN DATE1-DATE&N;
IF FIRST.NAME THEN I=1;
ELSE I + 1;
DATES{I} = DATE;
IF LAST.NAME;
IF I < &N THEN DO I=I+1 TO &N;
   DATES{I}=.;
END;
```

The DO group in the last three lines handles NAMEs which don't need all &N variables. It resets to missing retained date variables which were needed by the previous NAME.

What's In a Name?

Sometimes, the number of incoming observations varies dramatically from one NAME to the next. For example, one NAME might contain 100 observations, while the other NAMEs contain just three. In that case, transposing the data would create 100 variables (DATE1 through DATE100), most of which are missing on most observations. In these types of cases, compressing the transposed data will help save storage space. When Version 7 of the software becomes available, it will supply better compression algorithms for compressing numeric data. (In practice, the vast majority of transpositions involve numeric variables.)

PROC TRANSPOSE cannot change its methods: the result will always be 100 new variables. A DATA step, however, can produce many different forms to the transposed data. One possibility is to create just 20 new variables (DATE1 through DATE20), but create multiple observations per NAME as needed. In the transposed data, then, NAMEs with up to 20 incoming observations would need just 1 transposed observation, NAMEs with 21 to 40 incoming observations would need 2 transposed observations, etc. The variables in the transposed data set would be:

NAME
DATE1 - DATE20
OBSNO

OBSNO numbers the observations (1, 2, 3, 4, 5) for the current value of NAME.

To create this new form to the transposed data, the DATA step requires very little change:

```
DATA NEW (KEEP=NAME DATE1-DATE20);
ARRAY DATES {20} DATE1-DATE20;
OBSNO + 1;
DO I=1 TO 20 UNTIL (LAST.NAME);
   SET OLD;
   BY NAME;
   IF FIRST.NAME THEN OBSNO=1;
   DATES{I} = DATE;
END;
```

This DATA step has two ways it can exit the DO loop and output an observation. Either it fills in values for all 20 array elements, or it encounters the last observation for a NAME.

Transposing Two Variables

As the situations become more complex, a necessary skill for the programmer is the ability to visualize the data before and after transposing. Consider the case where the program transposes multiple variables. Here is the incoming data set:

NAME	DATE	RESULT
Amy	Date #A1	Res #A1
Amy	Date #A2	Res #A2
Amy	Date #A3	Res #A3
Bob	Date #B1	Res #B1
Bob	Date #B2	Res #B2
Bob	Date #B3	Res #B3

PROC TRANSPOSE requires little change. This would be the program:

```
PROC TRANSPOSE DATA=OLD OUT=NEW;
     VAR DATE RESULT;
     BY NAME;
```

The output data set would contain two observations per NAME, one for each transposed variable:

NAME	_NAME_	COL1	COL2	COL3
Amy	DATE	Date#A1	Date#A2	Date#A3
Amy	RESULT	Res#A1	Res#A2	Res#A3
Bob	DATE	Date#B1	Date#B2	Date#B3
Bob	RESULT	Res#B1	Res#B2	Res#B3

For practical purposes, this structure is a drawback. Most analyses require one observation per NAME, with all seven variables:

```
NAME
DATE1-DATE3
RESULT1-RESULT3
```

The DATA step can easily create the more desirable structure:

```
DATA ALL7VARS
       (DROP=I DATE RESULT);
ARRAY DATES   {3} DATE1-DATE3;
ARRAY RESULTS {3} RESULT1-RESULT3;
DO I=1 TO 3 UNTIL (LAST.NAME);
   SET OLD;
   BY NAME;
   DATES  {I} = DATE;
   RESULTS{I} = RESULT;
END;
```

While PROC TRANSPOSE can also create the more desirable form, it cannot do so in one step. The possibilities include:

- Run PROC TRANSPOSE just once (as above), and use a more complex DATA step to recombine the results, or
- Run PROC TRANSPOSE twice, and merge the results.

The first method might continue (ater PROC TRANSPOSE) with:

```
DATA ALL7VARS (DROP=_NAME_);
MERGE NEW (WHERE=(_NAME_='DATE')
           RENAME=(COL1=DATE1
                   COL2=DATE2
                   COL3=DATE3))
      NEW (WHERE=(_NAME_='RESULT')
           RENAME=(COL1=RESULT1
                   COL2=RESULT2
                   COL3=RESULT3));
BY NAME;
```

There is no shortcut for renaming variables individually. The software does not support renaming a list of variables in this fashion:

```
RENAME=(COL1-COL3=DATE1-DATE3)
```

However, the original PROC TRANSPOSE could have reduced the renaming load by adding PREFIX=DATE. If the number of variables being renamed is sufficiently large, it would pay to write a macro to generate sets of renames, based on a prefix coming in (such as COL), a prefix on the way out (such as DATE), and a numeric range (such as from 1 to 3).

The alternate approach using PROC TRANSPOSE would transpose the data twice, transposing a different variable each time. Afterwards, MERGE the results. Here is an example:

```
PROC TRANSPOSE DATA=OLD OUT=TEMP1
     PREFIX=DATE;
     VAR DATE;
     BY NAME;

PROC TRANSPOSE DATA=OLD OUT=TEMP2
     PREFIX=RESULT;
     VAR RESULT;
     BY NAME;

DATA ALL7VARS (DROP=_NAME_);
MERGE TEMP1 TEMP2;
BY NAME;
```

This program may be simpler; that judgment is in the eye of the beholder. However, processing large data sets three times instead of once can take a while.

Transposing Back Again

Occasionally a program must transpose data in the other direction, creating multiple observations from each existing observation. Here is the "before" picture:

NAME	DATE1	DATE2	DATE3
Amy	Date #A1	Date #A2	.
Bob	Date #B1	.	Date #B3

The "after" picture should look like this:

NAME	DATE
Amy	Date #A1
Amy	Date #A2
Bob	Date #B1
Bob	.
Bob	Date #B3

Notice how the program must handle missing values. Missing values at the end of the stream of dates (as in Amy's data) should be ignored. However, missing values with a valid DATE following (as in Bob's data) should be included in the transposed data.

Under these conditions, PROC TRANSPOSE cannot do the job! It can come close:

```
PROC TRANSPOSE DATA=BEFORE
     OUT=AFTER (DROP=_NAME_
         RENAME=(COL1=DATE));
     VAR DATE1-DATE3;
     BY NAME;
```

However, this program creates three observations for Amy, not two. Without the DROP= and RENAME= data set options, the output data set would have contained:

NAME	COL1	_NAME_
Amy	Date #A1	DATE1
Amy	Date #A2	DATE2
Amy	.	DATE3
Bob	Date #B1	DATE1
Bob	.	DATE2
Bob	Date #B3	DATE3

A subsequent DATA step could process this output, eliminating the third observation for Amy. However, if the program is going to introduce a DATA step, it can use the same DATA step to transpose the data. For the simplified case where no missing values belong in the output data set, this program would suffice:

```
DATA AFTER (KEEP=NAME DATE);
SET BEFORE;
ARRAY DATES {3} DATE1-DATE3;
DO I=1 TO 3;
   IF DATES{I} > . THEN DO;
      DATE = DATES{I};
      OUTPUT;
   END;
END;
```

In this situation, however, Bob requires three observations, not two. The embedded missing value (DATE2, falling between nonmissing values in DATE1 and DATE3) must also appear in the output data set. In that case, a slightly more complex DATA step would do the trick:

```
DATA AFTER (KEEP=NAME DATE);
SET BEFORE;
NONMISS = N(OF DATE1-DATE3);
NFOUND=0;
ARRAY DATES {3} DATE1-DATE3;
IF NONMISS > 0 THEN DO I=1 TO 3;
   IF DATES{I} > . THEN NFOUND + 1;
   OUTPUT;
   IF NFOUND=NONMISS THEN LEAVE;
END;
```

The N function counts how many of the variables (DATE1 through DATE3) have nonmissing values. As the DO loop outputs observations, it counts the nonmissing values that it outputs. Once the DO loop has output all the nonmissing values (as well as any missing values found before that point), the LEAVE statement exits the loop.

Transpose Abuse

Some programs transpose data because the programmer finds it easier to work with variables than with observations. In many cases, this constitutes an abuse of PROC TRANSPOSE, and indicates that the programmer should become more familiar with processing groups of observations in a DATA step.

Let's take an example of a program that transposes unnecessarily. Using our NAME and DATE data set, the program should calculate for each NAME the average number of days between DATE values. In every case, the program has prepared the data first using:

```
PROC SORT DATA=OLD;
BY NAME DATE;
```

With the observations in the proper order, a DATA step could easily compute the differences between DATEs:

```
DATA DIFFER;
SET OLD;
BY NAME;
DIFFER = DIF(DATE);
IF FIRST.NAME=0;
```

Computing the average of the DIFFER values is simple. PROC MEANS could do it:

```
PROC MEANS DATA=DIFFER;
VAR DIFFER;
BY NAME;
```

Or, the same DATA step could continue with additional calculations:

```
IF DIFFER > . THEN DO;
   N + 1;
   TOTAL + DIFFER;
END;
IF LAST.NAME;
IF N > 0 THEN MEAN = TOTAL / N;
OUTPUT;
N = 0;
TOTAL = 0;
```

However, if you really wanted to (or didn't have the programming skills to choose one of the methods above), you could transpose the data to work with variables instead of observations:

```
PROC TRANSPOSE DATA=OLD OUT=NEW
     PREFIX=DATE;
VAR DATE;
BY NAME;
```

Assuming you know the maximum number of transposed variables per NAME is 20, calculate the differences in a DATA step:

```
DATA DIFFER;
SET NEW;
ARRAY DATES {20} DATE1-DATE20;
ARRAY DIFFS {19} DIFF1-DIFF19;
DO I=1 TO 19;
   DIFFS{I} = DATES{I+1} - DATES{I};
END;
```

At this point, computing the means is easy. Just add to the same DATA step:

```
MEAN = MEAN(OF DIFF1-DIFF19);
```

However, if you enjoy a really long program, you could always transpose these differences back the other way:

```
PROC TRANSPOSE DATA=DIFFER OUT=FINAL
     (KEEP=NAME COL1
      RENAME=(COL1=DIFFER));
VAR DIFF1-DIFF19;
BY NAME;
```

Finally, the data are ready for the same PROC MEANS as in the original example:

```
PROC MEANS DATA=FINAL;
VAR DIFFER;
BY NAME;
```

Of course, you wouldn't write such an involved program. On the other hand, I didn't make up this example out of thin air! For some individuals, the lesson is clear: learn to process groups of observations in a DATA step and you won't have to transpose your data so much.

The author welcomes questions, comments, and bright ideas on interesting programming techniques. Feel free to call or write:

Bob Virgile
Robert Virgile Associates, Inc.
3 Rock Street
Woburn, MA 01801
(781) 938-0307