PharmaSUG2012 - Paper TF10

Getting Your Data in Shape With PROC TRANSPOSE

Nancy Brucken, PharmaNet/i3, Ann Arbor, MI

ABSTRACT

How many times have you been able to transpose your data sets successfully on the first try with PROC TRANSPOSE? With the increasing use of CDISC SDTM and ADaM data sets, we frequently need to go from “narrow” to “wide” data sets, as when transposing lab data for a listing, or from “wide” to “narrow” data sets, as when transposing vital signs data from its original source to an SDTM VS domain. This paper explores some of the more commonly-used features of PROC TRANSPOSE, and also covers several situations where PROC TRANSPOSE cannot easily handle the required data set restructuring.

DEFINITIONS

“Narrow” data set: A data set with a single analysis variable per row, also known as a “vertical” data set. A common example of a narrow data set is the ADaM Basic Dataset Structure (BDS).

“Wide” data set: A data set with multiple analysis variables per row, also known as a “horizontal” data set. A common example of a wide data set is the ADaM Subject-Level Dataset (ADSL).

WIDE TO NARROW DATA SETS

Let’s start by transposing a simple wide data set to a narrow data set, using PROC TRANSPOSE default settings. Suppose we have the following wide data set:

X1	X2	X3	X4
1	3	4	8

We’d like to convert it to the following narrow data set:

X1	1
X2	3
X3	4
X4	8

PROC TRANSPOSE in its simplest form transposes all numeric variables in the input data set that do not appear in another statement in the PROC TRANSPOSE step. Thus, all we need to do is code the following:

```
proc transpose data=a out=b;
run;
```

All numeric variables will be transposed and stacked. The resulting data set actually becomes:

NAME	COL1
x1	1
x2	3
x3	4
x4	8

Note that SAS automatically generates the variable names _NAME_ and COL1.

RENAMING OUTPUT DATA SET VARIABLES

Since all variables in a SAS data set must have names, SAS makes an attempt to provide meaningful variable names in the output data set. However, the names it chooses by default are probably not the ones you would prefer to assign. PROC TRANSPOSE gives you the ability to override the default variable names with something more descriptive of your data. The NAME= option allows you to specify the name of the _NAME_ variable, while the PREFIX= option specifies the prefix to be used when naming the transposed variables.

If we take the data set shown in the preceding section and add the following options to our PROC TRANSPOSE step:

```
proc transpose data=a out=b name=test prefix=var;
run;
```

we obtain the following data set:

TEST	VAR1
x1	1
x2	3
x3	4
x4	8

BY-GROUP PROCESSING

It's unusual that we would be transposing an entire data set. More commonly, we need to transpose records within a subject or visit. Suppose we have the following data set:

Region	x1	x2	x3	x4
North	1	3	4	8
South	2	6	9	7

In this example, we can transpose the records by region with the addition of a BY statement to our PROC TRANSPOSE code:

```
proc transpose data=a out=b (drop=_label_) name=test prefix=var;
  by region;
run;
```

In this example, the variable names and labels are identical, so the resulting TEST and _LABEL_ variables in the transposed data set will be redundant. As such, we can drop _LABEL_ when the transposed data set is created. Variables will be transposed within BY-groups, and the resulting data set looks like:

Region	test	var1
North	x1	1
North	x2	3
North	x3	4
North	x4	8
South	x1	2
South	x2	6
South	x3	9
South	x4	7

MORE COMPLEX TRANSPOSES

Suppose we have the following data set:

Region	x1	x2	x3	x4
North	1	3	4	8
South	2	6	9	7

We would like to transpose pairs of variables on each record, so it looks something like the following, where each pair of color-coded values ends up on a separate record:

Region	test	var1	var2
North	x1	1	3
North	x2	4	8
South	x1	2	6
South	x2	9	7

This is something that PROC TRANSPOSE simply can't handle in a single step. Several calls to PROC TRANSPOSE would be required to create this data set, which might take a considerable amount of processing time for a large data set.

Instead, we can write a DATA step transpose that can achieve the desired result in a single pass through the data set:

```
data b (keep=region var1-var2);
  set a;
  by region;
  array xs(*) x1-x4;
  array vars(*) var1-var2;
  do i=1 to dim(xs);
    vars(2 - mod(i, 2)) = xs(i);
    if mod(i, 2) = 0 then output;
  end;
run;
```

In this step, we set up an array containing X1-X4 to store the variables from the input data set, and a separate array containing VAR1-VAR2 to hold the transposed pairs of variables for the output data set. Using the MOD function to control the array index, we move the pairs of variables from the input data set to the output data set, writing a record after each pair has been completed.

NARROW TO WIDE DATA SETS

Let's look now at transposing a narrow data set to a wide data set, using the default settings for PROC TRANSPOSE. Suppose we have this narrow data set:

x
1
3
4
8

We'd like to convert it to the following wide data set:

col1	col2	col3	col4
1	3	4	8

Remember that PROC TRANSPOSE by default transposes all numeric variables in the data set that do not appear in any other statements in the PROC TRANSPOSE step. Thus, if we run the following simple PROC TRANSPOSE step:

```
proc transpose data=a out=b;
run;
```

we'll end up with the following output data set:

NAME	COL1	COL2	COL3	COL4
x	1	3	4	8

That PROC TRANSPOSE code should look familiar- it's identical to the syntax used earlier to transpose a wide data set to a narrow data set. PROC TRANSPOSE merely swaps the rows and columns in the data set; that is, the rows become columns and the columns become rows.

RENAMING OUTPUT DATA SET VARIABLES

Again, we can use the NAME= and PREFIX= options to control the names of the variables in out output data set. By adding these options:

```
proc transpose data=a out=b name=test prefix=var;
run;
```

we obtain the following data set:

TEST	VAR1	VAR2	VAR3	VAR4
x	1	3	4	8

DATA-DRIVEN VARIABLE NAMES

Suppose we have the following data set:

Group	x
1	1
3	3
2	4
4	8

We would like to use the value of the GROUP variable to determine the names of the output data set variables, so that the resulting data set looks like:

test	var1	var2	var3	var4
x	1	**4**	**3**	8

Note the variable values in color. If we did not use the values of the GROUP variable to determine the names of the transposed data set variables, SAS would assign them in the order in which the records appear in the input data set. In that case, VAR2 would be 3, and VAR3 would be 4. The ID statement in PROC TRANSPOSE assigns the name of the output variable based on the value of the ID variable. If the ID variable is numeric, the names of the output variables begin with an underscore (_). If the PREFIX option is specified as well, the resulting variable names are a concatenation of the prefix and ID variable values.

The following code:

```
proc transpose data=a out=b name=test prefix=var;
  id group;
run;
```

produces our desired data set:

test	var1	var2	var3	var4
x	1	**4**	**3**	8

BY-GROUP PROCESSING

BY-group processing works the same way for a narrow-to-wide data set transpose as it does for a wide-to-narrow data set transpose. Suppose we have:

Region	Group	x
North	1	1
North	2	3
North	3	5
South	1	4
South	2	8
South	3	6

We'd like to transpose records within region, while using the value of GROUP to determine the transposed variable names, so we end up with a data set like:

Region	test	var1	var2	var3
North	x	1	3	5
South	x	4	8	6

The following code will produce that data set:

```
proc transpose data=a out=b (drop=_label_) name=test prefix=var;
  by region;
  id group;
run;
```

Let's look at a more realistic example. Suppose we have a narrow data set containing clinical lab results, such as an ADaM BDS data set:

Subject	AVISIT	PARAM	PARAMN	AVAL
1	Base	Hemoglobin	1	50
1	Base	Hematocrit	2	40
1	Base	WBC	3	9.2
1	Wk 1	Hemoglobin	1	55
1	Wk 1	Hematocrit	2	38
1	Wk 1	WBC	3	10.1

We need to generate a listing showing all lab parameters for a single visit on the same row, where the parameter name in the input data set becomes the label of the corresponding variable in the output data set:

Subject	AVISIT	Hemoglobin	Hematocrit	WBC
1	Base	50	40	9.2
1	Wk 1	55	38	10.1

The IDLABEL statement can be used for just that purpose- the value of the variable specified in an IDLABEL statement becomes the label of the output data set variable. There is one limitation on the use of an IDLABEL statement- it must be specified in conjunction with an ID statement.

The following code produces the desired data set:

```
proc transpose data=a out=b (drop=_name_) prefix=var;
  id paramn;
  idlabel param;
  var aval;
run;
```

The transposed variables will be named VAR1-VAR3, but the variable labels become as shown above.

HANDLING FORMATTED VARIABLES

Suppose your data set contains a variable X, with an associated format YN (unformatted values are shown in parentheses):

x
No (0)
Yes (1)
Yes (1)
No (0)

Transposing the data set using PROC TRANSPOSE with its default settings yields the following data set, with the YN format applied to all of the transposed variables:

col1	col2	col3	col4
No (0)	Yes (1)	Yes (1)	No (0)

TRANSPOSING MULTIPLE VARIABLES

Given the following input data set:

Group	x	y
1	1	2
3	3	5
2	4	7
4	8	9

What happens if we run the following code, specifying both X and Y in the VAR statement:

```
proc transpose data=a out=b name=test prefix=var;
  id group;
  var x y;
run;
```

SAS creates a row for each column, and a column for each row, in the output data set:

test	var1	var3	var2	var4
x	1	3	4	8
y	2	5	7	9

All of the values of variable X are stored on the same record, with TEST='x'; the same thing happens to the values of the variable Y.

Suppose we need a more complicated transpose, though, with the following input data set:

ANALYTEC	LISTTYPE	LISTNUM	RXGRP	PATCT	DENOM
CALCIUM	Subjects With One or More Abnormal Screen/Baseline Values	1	1	6	260
CALCIUM	Subjects With One or More Abnormal Screen/Baseline Values	1	2	4	227
CALCIUM	Subjects With Normal Screen/Baseline Values and One Abnormal Value After Initiation of Treatment	2	1	5	260
CALCIUM	Subjects With Normal Screen/Baseline Values and One Abnormal Value After Initiation of Treatment	2	2	6	227

In this example, we need to put the counts and denominators for both treatment groups on the same record, so they can be displayed on a summary table:

ANALYTEC	LISTTYPE	LISTNUM	PATCT1	PATCT2	DENOM1	DENOM2
CALCIUM	Subjects With One or More Abnormal Screen/Baseline Values	1	6	4	260	227
CALCIUM	Subjects With Normal Screen/Baseline Values and One Abnormal Value After Initiation of Treatment	2	5	6	260	227

PATCT1 and DENOM1 should contain the count and denominator for treatment group 1, while PATCT2 and DENOM2 should contain the count and denominator for treatment group 2. If we call PROC TRANSPOSE and specify PATCT and DENOM as the variables to be transposed, we'll end up with a data set where they are stacked onto separate records- remember that PROC TRANSPOSE changes columns into rows. Multiple variables specified in a VAR statement result in 1 record per transposed variable in the output data set. In this case, that is not what we're looking for.

One solution would be to run a separate PROC TRANSPOSE for each variable, and then merge the resulting data sets. That will work. However, it requires three passes through the data set- one pass for each PROC TRANSPOSE call, and then a final pass through each of the transposed data sets to perform the merge. For a small data set, this may not take very long. For a large data set, the run time may be significant.

Instead we can employ a technique known as a DOW-Loop to perform the transpose in a single DATA step. The distinguishing feature of this method is that it moves the DATA step SET statement inside of an explicit DO-loop, so the DATA statement is only executed at the beginning of a BY-group. Thus, values of variables created in that DATA step are retained in the Program Data Vector (PDV) throughout all of the program statements executed for that BY-group.

The following code accomplishes the desired transpose:

```
data out_ct (drop=patct denom rxgrp);
  array patcts (*) patct1-patct2;
  array denoms (*) denom1-denom2;
  do until (last.listnum);
    set out_ct1;
    by analytec listnum;

    patcts(rxgrp) = patct;
    denoms(rxgrp) = denom;
  end;
  output;
run;
```

Since PATCT1-PATCT2 and DENOM1-DENOM2 are created in the DATA step, rather than being read in from the input data set, their values are retained over all records in the BY-group. This allows us to populate all values in the arrays before generating the output (transposed) record at the end of the BY-group. Note also that the treatment group value (RXGRP) is used as the array index, so that the treatment group counts and denominators are stored in the correct array elements.

ADDITIONAL NOTES

If the VAR statement in a PROC TRANSPOSE contains both numeric and character variables, the resulting transposed variables are all character fields.

The ID statement uses the value of the ID variable in the name of the transposed variable. The IDLABEL statement uses the label of the IDLABEL variable as the label of the transposed variable. The IDLABEL statement cannot be used without a corresponding ID statement.

CONCLUSION

PROC TRANSPOSE can be used to easily convert a wide data set to a narrow data set, or a narrow data set to a wide data set. In either case, the PROC TRANSPOSE code is identical, because all the procedure does is create a row in the output data set for every value of the variables specified on the VAR statement. Various PROC TRANSPOSE procedure options can be used to further control the structure of the transposed data set. Finally, for the cases where PROC TRANSPOSE just won't quite do, there's always the possibility of performing the required transpose using a DATA step.

REFERENCES

PROC TRANSPOSE

See the SAS documentation for other procedure options.

Go to http://www.lexjansen.com and search for TRANSPOSE

DOW-LOOP REFERENCES

Dorfman, Paul. (2008) "The DOW-Loop Unrolled". PharmaSUG 2008 Conference Proceedings.

Chakravarthy, Venky, (2005). "RETAIN or NOT? Is LAG Far Behind?". PharmaSUG 2005 Conference Proceedings

Brucken, Nancy. (2009) "One-Step Change from Baseline Calculations". PharmaSUG 2009 Conference Proceedings

Search the SAS-L listserv archives at http://www.listserv.uga.edu/archives/sas-l.html for postings by Ian Whitlock and Paul Dorfman, among others.

OTHER SAS REFERENCES

SAS-L listserv archives (http://www.listserv.uga.edu/archives/sas-l.html- also available from Google Groups as comp.soft-sys.sas)

sasCommunity.org (http://www.sascommunity.org)

SAS Discussion Forums (http://support.sas.com/forums/index.jspa)

SAS Technical Support (http://support.sas.com)

CONTACT INFORMATION

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

Nancy Brucken
PharmaNet/i3
5430 Data Court, Suite 200
Ann Arbor, MI 48108
734-757-9045
nbrucken@pharmanet-i3.com