A Row is a Row is a Row, or is it? Get Comfortable with Transposing your Data

Christianna S. Williams, Abt Associates Inc, Durham, NC

ABSTRACT

Sometimes life would be easier for the busy SAS programmer if information stored in multiple rows (observations) were all accessible in one observation, using additional columns (variables) to hold that data. Sometimes it makes more sense to turn a short, wide data set into a long, skinny one (i.e. convert columns into rows). Base SAS® provides us with two primary methods for converting rows (observations) into columns (variables) or vice versa – PROC TRANSPOSE and our trusty old friend the DATA step. How do these two methods work? Which is best suited to different transposition problems, such as situations requiring multi-row arithmetic or transposing multiple variables? The purpose of this example-packed tutorial is to demonstrate various types of transpositions using the DATA step and to demystify the TRANSPOSE procedure (e.g. nail the BY statement and the NAME= option once and for all). Afterwards, you should be the office go-to gal/guy for reshaping data. Sweet!

INTRODUCTION

I probably shouldn't admit this, but for me, the best way to really learn something in SAS is trial and error, or, perhaps more accurately, to see what happens when I try different things…and to gradually narrow down to what I'm trying to accomplish. The hope is that next time around, I'll get there a little more quickly. I'll also confess that PROC TRANSPOSE sort of confused me for a long time (die-hard DATA Step-per that I am), until I played with all the different features in TRANSPOSE enough to really (almost) commit them to memory. My intention in this paper is to have you climb that learning curve with me, through a series of examples showing what TRANSPOSE can do. For many of the examples, I also provide DATA step code that will accomplish the same thing, and muse a bit, about the pros and cons of each.

THE DATA

All the examples in this paper are based on some made-up data for 30 participants in a longitudinal study of blood pressure. These patients may have had up to 3 visits over the course of the study. At each visit, their blood pressure (systolic and diastolic, both in mm/Hg – SBP and DBP, respectively), and their waist-hip ratio (WHR) were recorded. A listing of the entire data set is at the end of this paper, in case you want to try out some of the examples – or make up your own! – with a few more observations.

EXAMPLE 1 – PLAIN VANILLA PROC TRANSPOSE

For this first example, I'm starting with a small subset of the larger data set, just the first 8 observations and a subset of three variables – patient ID, visit number, and systolic blood pressure (SBP). What it looks like to start is shown in ***Exhibit 1.1.***

Exhibit 1.1. Plain vanilla TRANSPOSE
Before Transposition (Data set = LONG1)

Obs	ptid	visit	sbp
1	01	1	142
2	01	2	141
3	01	3	131
4	02	1	107
5	02	2	.
6	02	3	111
7	03	1	135
8	03	3	128

Currently this data set is normalized – there is a separate row in the data set for each visit for a given patient. A few features to note as we proceed through the examples: First, for PTID=02, VISIT=2, there is a missing value for SBP. Also, PTID 03 has no observation at all for VISIT=2.

Let's see what happens if we use PROC TRANSPOSE without any additional statements or options. This is the code:

```
PROC TRANSPOSE DATA=long1 OUT=wide1;
RUN;
```

What it looks like after this transposition is shown in ***Exhibit 1.2***.

Exhibit 1.2. Plain vanilla TRANSPOSE
After Transposition (Data set = WIDE1)

Obs	_NAME_	_LABEL_	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8
1	ptid	Patient ID	1	1	1	2	2	2	3	3
2	visit	Visit #	1	2	3	1	2	3	1	3
3	sbp	Systolic BP	142	141	131	107	.	111	135	128

So what happened? The three columns in the input data set (PTID, VISIT and SBP) became three rows in the output. And the data values that were in the eight rows are now held in eight columns (variables), and these variables were given the names COL1 – COL8. Additionally, there are two more columns, containing the name (_NAME_) and label (_LABEL_) of the transposed variables. Incidentally, if none of the transposed variables had labels on the input data set (LONG1), the transposed (wide) data set would not have the _LABEL_ variable. While I can imagine some scenarios where this wide data set might be the desired result (e.g. in preparations for some type of reporting), it does strike me as a rather odd because each COL variable contains three different kinds of information, which is unusual for a SAS data set…and, fiddler that I am, made me want to know what would happen if one of the variables in the original (long) data set was a *character* variable.

A note about Character Variables and PROC TRANSPOSE

So, let's make a slight tweak to the input data set, by including SEX, which is a character variable. This data set, LONG2 is shown in exhibit 1.3. We run the same code, just changing the data set names:

```
PROC TRANSPOSE DATA=long2 OUT=wide2;
RUN;
```

Exhibit 1.3. Plain vanilla TRANSPOSE with a character variable
Before Transposition (Data set = LONG2)

Obs	ptid	sex	visit	sbp
1	01	F	1	142
2	01	F	2	141
3	01	F	3	131
4	02	F	1	107
5	02	F	2	.
6	02	F	3	111
7	03	M	1	135
8	03	M	3	128

Somewhat surprisingly, the resulting data set, WIDE2, is *identical* to WIDE1. SAS ignored the character variable. By default, only numeric variables are transposed by PROC TRANSPOSE. If we want to have a character variable transposed, then we must include a VAR statement as follows:

```
PROC TRANSPOSE DATA=long2 OUT=wide3;
VAR ptid sex visit sbp;
RUN;
```

The resulting data set is shown in ***Exhibit 1.4.***

Exhibit 1.4. TRANSPOSE with VAR statement to include a character variable
After Transposition (Data set = WIDE3)

Obs	_NAME_	_LABEL_	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8
1	ptid	Patient ID	01	01	01	02	02	02	03	03
2	sbp	Systolic BP	142	141	131	107	.	111	135	128
3	sex	Gender (M/F)	F	F	F	F	F	F	M	M
4	visit	Visit #	1	2	3	1	2	3	1	3

So...now COL1-COL8 are storing numeric and character data…how is this possible?? It's not…check the log, and the following message is there:

```
NOTE: Numeric variables in the input data set will be converted to character in the
output data set.
```

So, even my data that is truly quantitative (e.g. the systolic blood pressure) has gotten converted to character values. This is a little deceptive, since SAS even goes so far as to show a period (.) for the missing blood pressure value, which might even fool you into thinking it was a numeric variable. I'll note also that funky things can happen even for numeric variables, because SAS has to figure out what to do about different formats. We actually see this above a bit. The PTID variable has a Z2. format on the LONG data set, and this was lost in the first transposition (***Exhibit 1.2***). Also, if one of the variables contained non-integer values, all the variables would show the number of decimal points being displayed for the non-integer, and all are given a numeric length corresponding to the longest length of any of the variables being transposed. …So, at least for this data set, this kind of TRANSPOSE is probably not very useful, but we've learned something about the PROC.

The NAME= and LABEL= Options in PROC TRANSPOSE

Before moving onto the next main example, I'll demonstrate the use of two options in PROC TRANSPOSE that are very useful if you are transposing multiple variables in one step. Instead of the default variable names of _NAME_ and _LABEL_, we can use the NAME= and LABEL=options to give our choice of names to those variables in the output data set. The code is shown at right and the result below (***Exhibit 1.5***). Good to keep in mind if you wanted to use a TRANSPOSEd data set as a basis for some type of report.

```
PROC TRANSPOSE DATA=long2 OUT=wide3a
   NAME=varname LABEL=varlabel;
VAR ptid sex visit sbp;
RUN;
```

Exhibit 1.5. TRANSPOSE with VAR statement to include a character variable and Using NAME= and LABEL= Options
After Transposition (Data set = WIDE3a)

Obs	varname	varlabel	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8
1	ptid	Patient ID	01	01	01	02	02	02	03	03
2	sbp	Systolic BP	142	141	131	107	.	111	135	128
3	sex	Gender (M/F)	F	F	F	F	F	F	M	M
4	visit	Visit #	1	2	3	1	2	3	1	3

EXAMPLE 2 – PROC TRANSPOSE WITH BY STATEMENT

It is more likely, given the structure of our long data set (i.e. multiple rows per person), that we would want to re-shape it so that it has one observation for each person (PTID) and variables corresponding to the different measurements on each person at each time point. In terms of PROC TRANSPOSE that means using a BY statement – that is, TRANSPOSing BY PTID. Again, we start with the LONG1 data set from ***Exhibit 1.1***, and use this code:

```
PROC TRANSPOSE DATA=long1 OUT=wide4;
BY ptid ;
RUN;
```

And the output is shown below in ***Exhibit 2.1***.

Exhibit 2.1. TRANSPOSE with BY statement
After Transposition (Data set = WIDE4)

Obs	ptid	_NAME_	_LABEL_	COL1	COL2	COL3
1	01	visit	Visit #	1	2	3
2	01	sbp	Systolic BP (mm/Hg)	142	141	131
3	02	visit	Visit #	1	2	3
4	02	sbp	Systolic BP (mm/Hg)	107	.	111
5	03	visit	Visit #	1	3	.
6	03	sbp	Systolic BP (mm/Hg)	135	128	.

So, in comparing this to the output without the BY statement (***Exhibit 1.1***), we see that instead of one observation per variable (or one row in the output for each column of the input) we now have one set of rows for each value of the BY variable, and each set contains a row for each of the variables (other than the BY variable).

Let's make a few enhancements. First, so that our variables are not storing different kinds of information, let's just TRANSPOSE the SBP variable; we do this by using the VAR statement. (We'll deal with the VISIT number in the next example). Second, it would sure be nice for the columns in the output data set to have meaningful names; for that we use the PREFIX option on the PROC statement, so that the output variables will be SYSTOLIC1 – SYSTOLICn where N is the largest number of observations for any BY group in the input data set (here, 3). And, third, since we won't really need the _NAME_ and _LABEL_ variables anymore (it is redundant with the info in the new variable/column names), we'll drop those. So, the new code is shown here, followed by the output in ***Exhibit 2.2*** below:

```
PROC TRANSPOSE DATA=long1
        OUT=wide5 (DROP=_NAME_ _LABEL_)
         PREFIX=systolic;
BY ptid;
VAR sbp;
RUN;
```

Exhibit 2.2. TRANSPOSE with BY statement and PREFIX option
After Transposition (Data set = WIDE5)

Obs	ptid	systolic1	systolic2	systolic3
1	01	142	141	131
2	02	107	.	111
3	03	135	128	.

Now, this is all great *EXCEPT* we have lost some critical information. In the input data set (***Exhbit 1.1***), PTID 2 has an observation for VISIT=2 but the SBP value is missing. This is reflected in the output data set shown in ***Exhibit 2.2*** (i.e. SYSTOLIC2 is missing). In contrast, in the input data, PTID 3 has no observation for VISIT 2, but because we are not using the VISIT information in this transposition, SAS doesn't "know" that the second observation for PTID 3 corresponds to VISIT 3. Hence, the SBP value for the second observation is stored in SYSTOLIC2, resulting in the situation where the variable SYSTOLIC2 is storing data that we may not consider to be comparable across observations (i.e. for different visits). There might be some applications where that is ok – but for this data, we have lost an important feature of the study design. Read on…

EXAMPLE 3 – PROC TRANSPOSE WITH *BY* STATEMENT AND *ID* STATEMENT

Fortunately, If we want the suffixes for the systolic variables to correspond to the visit number, we can use the ID statement in PROC TRANSPOSE to achieve just that. That is all that has changed in the code between the last example and this one.

```
PROC TRANSPOSE DATA=long1
                OUT=wide6 (DROP=_NAME_ _LABEL_)
              PREFIX=systolic;
BY ptid;
VAR sbp;
ID visit;
RUN;
```

The new and improved output data set is shown below ***(Exhibit 3.1)***.

Exhibit 3.1 TRANSPOSE with BY statement, ID statement and PREFIX option
After Transposition (Data set = WIDE6)

Obs	ptid	systolic1	systolic2	systolic3
1	01	142	141	131
2	02	107	.	104
3	03	135	.	128

This is more like it! Consistently, across PTID's, SYSTOLIC1 holds the SBP value for VISIT = 1, SYSTOLIC2 corresponds to VISIT=2, and SYSTOLIC3 corresponds to VISIT=3. What we can't tell from the above is that PTID 2 was just missing the systolic value for VISIT 2, while PTID 3 was missing the entire visit; that may or may not be important. If it is important, we would need to also transpose the VISIT variable, but as we've seen above, when multiple variables are included in the VAR statement for PROC TRANSPOSE, multiple rows (per BY value) are generated, rather than additional columns for each additional transposed variable. Examples showing how to get around that are included later in this paper. BUT I want to drive home a point here about TRANSPOSE syntax (that [despite reading the documentation] took me a lot of trial and error to commit to memory!), and that is…

> Variables in the **BY** statement affect the ***structure*** of the output data set; that is, what generates a new observation (or set of observations). In contrast,
>
> ***Values*** of variables in the **ID** statement affect the ***names*** of the variables in the output data set, and can provide additional information about the data structure.

Another note here – if there are other variables that are at the level of the BY group that you want to keep associated with each BY group value, you can add them into the BY statement and they will be carried along. More concretely, in this example data set, the variable SEX is constant within each PTID, and I would like to keep it on my transposed data set…basically it needs to come along for the ride in the TRANSPOSE. Simply add SEX to the BY statement, as shown here. The output is shown in ***Exhibit 3.2.***

```
PROC TRANSPOSE DATA=long2
         OUT=wide6a (DROP=_NAME_ _LABEL_)
          PREFIX=systolic;
BY ptid sex;
VAR sbp;
ID visit;
RUN;
```

Exhibit 3.2 TRANSPOSE with BY statement, ID statement and PREFIX option
After Transposition (Data set = WIDE6A)

Obs	ptid	sex	systolic1	systolic2	systolic3
1	01	F	142	141	131
2	02	F	107	.	111
3	03	M	135	.	128

Using the IDLABEL statement in PROC TRANSPOSE

There's one other statement in PROC TRANSPOSE that I've found to be handy – especially when there are LOTS of observations per BY group in the "long" data set and thus, lots of new variables in the transposed "wide" data set. And that is the IDLABEL statement. The variable that is named on the IDLABEL statement provides LABELS to the transposed variables. In my experience, it often requires some pre-processing so that an appropriate IDLABEL variable exists on the long (pre-transposition) data set. As shown at left, let's add a variable to the LONG1 data set that will work in this way…and then do the PROC TRANSPOSE. The variable SBPLABEL concatenates some informative text with the value of visit for the current observation. The resulting LONG1a data set is shown in ***Exhibit 3.3*** (though it's looking a little wide) ☺. This new variable is specified in the IDLABEL statement in our TRANSPOSE. If we use

```
DATA long1a ;
SET long1 ;
sbplabel = 'Systolic BP visit '||PUT(visit,1.);
RUN;

PROC TRANSPOSE DATA=long2a
               OUT=wide6b (DROP=_NAME_ _LABEL_)
            PREFIX=systolic;
BY ptid;
VAR sbp;
ID visit;
IDLABEL sbplabel ;
RUN;
```

the LABEL option in PROC PRINT of dataset WIDE6B, we see the effects of the IDLABEL statement ***(Exhibit 3.4)***. We'll use this statement again when we are transposing multiple variables…

Exhibit 3.3. Adding a variable that will work as IDLABEL
Before Transposition (Data set = LONG1a)

Obs	ptid	visit	sbp	sbplabel
1	01	1	142	Systolic BP visit 1
2	01	2	141	Systolic BP visit 2
3	01	3	131	Systolic BP visit 3
4	02	1	107	Systolic BP visit 1
5	02	2	.	Systolic BP visit 2
6	02	3	111	Systolic BP visit 3
7	03	1	135	Systolic BP visit 1
8	03	3	128	Systolic BP visit 3

Exhibit 3.4. Transposition using the IDLABEL statement

Obs	Patient ID	Systolic BP visit 1	Systolic BP visit 2	Systolic BP visit 3
1	01	142	141	131
2	02	107	.	111
3	03	135	.	128

EXAMPLE 4 – USING THE DATA STEP TO TRANSPOSE A SINGLE VARIABLE

If you need to transpose a single variable – as we've been doing in most of the above examples, then PROC TRANSPOSE may be the way to go. However, when you need to transpose (or as we sometimes say at work, "horizontalize") multiple variables, DATA step methods may be preferable. To build up to that, I first show DATA step code for transposing a single variable. One method is shown to the right.

```
DATA wide7 (KEEP = ptid systolic1-systolic3);
 SET long1 ;
BY PTID ;

ARRAY sys{3} systolic1 - systolic3;
RETAIN  systolic1 - systolic3;
IF FIRST.ptid THEN DO i = 1 TO 3;
   sys{i} = . ;
END;
sys{visit} = sbp ;

IF LAST.ptid;

RUN;
```

The result, shown in ***Exhibit 4.1***, is identical to what we saw in the last example ***(Exhibit 3.1)***. A few notes about this strategy:

1) VISIT is used as an index in the array, to place the SBP values in the right places, which is handy.
2) We needed to know what the max value of the VISIT variable is in order to set this up, which might require some pre-processing.
3) The RETAIN is needed so that values assigned to each element of the SYSTOLIC array are maintained across observations for a given BY value.
4) It is necessary to initialize the array elements to missing values at the beginning of each BY group so that values are not carried over from previous PTID by the RETAIN statement. This would not be necessary if there were no missing data and all PTID's had the same number of visits.

Exhibit 4.1 Using the DATA step to transpose a single variable
After Transposition (Data set = WIDE7)

Obs	ptid	systolic1	systolic2	systolic3
1	01	142	141	131
2	02	107	.	104
3	03	135	.	128

EXAMPLE 5 – AN ALTERNATIVE DATA STEP METHOD

A slight twist on the above is to put the SET statement within the DO loop. This may be somewhat unconventional, but it eliminates the need for RETAIN, because the DATA step doesn't reinitialize SYSTOLIC1-SYSTOLIC3 to missing until it returns to the DATA statement, which will be the last observation for the BY group.

```
DATA wide8 (KEEP = ptid systolic1-systolic3);

ARRAY sys{3} systolic1 - systolic3;
DO i = 1 TO 3 UNTIL (LAST.ptid);
   SET long1;
       BY ptid ;
   sys{visit} = sbp ;
END;

RUN;
```

The resulting data set, not repeated again, is identical to that shown in ***Exhibit 3.1*** (& ***4.1***).

In summary, for transposing a single variable, the advantages of the TRANSPOSE method are that the code is a little shorter and that there is no requirement to know the maximum number of observations in a BY group. Of course, one could use a little bit of additional code to determine that maximum number and store it in a macro variable so that it wouldn't have to be hard-coded (e.g. Virgile, 1998 or Williams, 2005). On the other hand, the DATA step has some additional flexibility, if, for example we wanted to do some cross-row arithmetic, as in the next example.

Also, if we want to keep other variables at the level of the BY variable – that is, are constant across observations for a PTID – such as SEX, all that is required in either DATA step method is to KEEP those variables.

EXAMPLE 6 – TRANSPOSING WITH CROSS-ROW ARITHMETIC

Let's say that we want to compute the average of the systolic BP values for each PTID, and determine at which visit, the value was the lowest. If we wanted to use PROC TRANSPOSE, we'd also have to add a DATA step. In my mind, it's simpler to just use the DATA step. The code shown here will work. It starts with the same program as Example 4, and elaborates on it a bit. When the DATA step is at the first observation within a BY group (i.e. FIRST.ptid is true), we initialize not only the elments of the systolic array, but also our summary variables, setting these (MIN_SYS, which will store the lowest systolic value for each patient, and MIN_SYS_VIS, which will store the VISIT at which that minimum value was recorded) to missing, so that the RETAIN statement only maintains these values within records for a given PTID, not across different patients.

```
DATA wide9 (KEEP = ptid systolic1-systolic3
                   min_sys avg_sys min_sys_vis);
SET long1 ;
BY PTID ;

ARRAY sys{3} systolic1 - systolic3;
RETAIN systolic1-systolic3 min_sys min_sys_vis;
IF FIRST.ptid THEN DO ;
   DO i = 1 TO 3;
     sys{i} = . ;
   END;
   min_sys = . ;
   min_sys_vis = . ;
END;

sys{visit} = sbp ;
min_sys = MIN(min_sys,sbp) ;
IF min_sys = sbp THEN min_sys_vis = visit ;

IF LAST.ptid THEN DO;
   avg_sys = MEAN(OF systolic1-systolic3) ;
   OUTPUT ;
END;
RUN;
```

As in Example 4, we assign the current SBP value to its correct spot in the array. We then check to see which value is lower – the current SBP value or the minimum (so far) SBP value for the patient, and assign that to MIN_SYS. If the current value is the lowest value (so far), then the current visit number is assigned to the variable MIN_SYS_VIS.

Finally, when the DATA step gets to the last observation for the BY group (i.e. LAST.ptid is true), we obtain the average value of the array elements and assign it to AVG_SYS, and OUTPUT an observation. The resulting data set is shown in ***Exhibit 6.1***.

TIP!! Don't forget that 'OF" in the argument to the MEAN function; without it the value assigned to AVG_SYS will be the value of SYSTOLIC1 *minus* the value of SYSTOLIC3 – likely to be a negative value – a lesson I've learned the hard way!

Exhibit 6.1 Transposing with Cross-Row Arithmetic (Data set = WIDE9)

Obs	ptid	systolic1	systolic2	systolic3	min_sys	min_sys_vis	avg_sys
1	01	142	141	131	131	3	138.0
2	02	107	.	104	104	3	105.5
3	03	135	.	128	128	3	131.5

EXAMPLE 7 – TRANSPOSING MULTIPLE VARIABLES FOR A BY GROUP – DATA STEP

As we saw in Example 2 (***Exhibit 2.1***), if multple variables are in the VAR statement for PROC TRANSPOSE, then the resulting data set has multiple rows per BY group – one per transposed variables. So, if what we need to do is string all the variables out within one row for each BY group, we need a different strategy. Using the DATA step, we can very simply modify either technique – from Example 4 or 5 – to accommodate multiple variables.

First, I create another subset of the full data set (LONG3; ***Exhibit 7.1***), including the variables PTID, SEX, VISIT, SBP, DBP, and WHR, and all observations for the first 5 PTID's. Below I show the two different DATA step methods, which are obvious extensions of the one-variable methods, and the results – identical for both methods – are shown in ***Exhibit 7.2***. The program on the right that has the SET and BY

Exhibit 7.1. Transposing multiple variables for a BY Groups
Before Transposition (Data set = LONG3)

Obs	ptid	sex	visit	sbp	dbp	whr
1	01	F	1	142	92	0.88
2	01	F	2	141	91	0.87
3	01	F	3	131	83	0.83
4	02	F	1	107	58	0.75
5	02	F	2	.	58	0.75
6	02	F	3	111	55	0.71
7	03	M	1	135	80	0.97
8	03	M	3	128	74	0.94
9	04	F	1	145	84	.
10	04	F	2	145	84	0.71
11	04	F	3	139	79	0.68
12	05	M	1	136	86	1.00
13	05	M	2	132	83	0.99
14	05	M	3	126	.	0.96

```
DATA wide10 (KEEP = ptid sex
    sbp1-sbp3 dbp1-dbp3 whr1-whr3 );

SET long3 ;
BY PTID ;

ARRAY sys{3} sbp1 - sbp3;
ARRAY dia{3} dbp1-dbp3;
ARRAY wst{3} whr1-whr3;
RETAIN  sbp1-sbp3 dbp1-dbp3
        waisthip1-waisthip3;

IF FIRST.ptid THEN DO i = 1 TO 3;
   sys{i} = . ;
   dia{i} = . ;
   wst{i} = . ;
END;

sys{visit} = sbp ;
dia{visit} = dbp ;
wst{visit} = whr ;

IF LAST.ptid;
RUN;
```

```
DATA wide10a (KEEP = ptid sex
       sbp1-sbp3 dbp1-dbp3 whr1-whr3);

ARRAY sys{3} sbp1 - sbp3;
ARRAY dia{3} dbp1-dbp3;
ARRAY wst{3} whr1-whr3;

DO i = 1 TO 3 UNTIL (LAST.ptid);
   SET long3;
     BY ptid ;
   sys{visit} = sbp ;
   dia{visit} = dbp ;
   wst{visit} = whr ;
END;

RUN;
```

statements inside the DO loop is notably shorter, as it doesn't require the "intialization" step or the RETAIN, but the two accomplish exactly the same result.

Exhibit 7.1 Transposing Multiple Variables for a BY Group (Data set = WIDE10 & 10a)

ptid	sex	sbp1	sbp2	sbp3	dbp1	dbp2	dbp3	whr1	whr2	whr3
01	F	142	141	131	92	91	83	0.88	0.87	0.83
02	F	107	.	111	58	58	55	0.75	0.75	0.71
03	M	135	.	128	80	.	74	0.97	.	0.94
04	F	145	145	139	84	84	79	.	0.71	0.68
05	M	136	132	126	86	83	.	1.00	0.99	0.96

EXAMPLE 8 – TRANSPOSING MULTIPLE VARIABLES FOR A BY GROUP – PROC TRANSPOSE

To accomplish the same task with PROC TRANSPOSE requires a separate step for each variable. The three data sets can then be combined with a MERGE step. I'm also assuming that I have a data set LONG3a, which contains variables SBPLABEL, DBPLABEL and WHRLABEL that are analogous to the SBPLABEL variable from ***Exhibit 3.3***. Aside from that enhancement (i.e. the transposed variables have labels), the result is identical to the data set shown in ***Exhibit 7.1***.

```
PROC TRANSPOSE DATA=long3a OUT=wide11a  (DROP= _NAME_ _LABEL_) PREFIX=sbp;
BY ptid sex;
VAR sbp;
ID visit ;
IDLABEL sbplabel ;
RUN;

PROC TRANSPOSE DATA=long3a OUT=wide11b  (DROP= _NAME_ _LABEL_) PREFIX=dbp;
BY ptid sex;
VAR dbp;
ID visit ;
IDLABEL dbplabel ;
RUN;

PROC TRANSPOSE DATA=long3a OUT=wide11c  (DROP= _NAME_ _LABEL_) PREFIX=whr;
BY ptid sex;
VAR whr;
ID visit ;
IDLABEL whrlabel ;
RUN;

DATA wide11 ;
MERGE wide11a wide11b wide11c ;
BY ptid ;
RUN;
```

Of course, we can write a simple MACRO for the TRANSPOSE step, setting the variable to be transposed as a macro parameter, and calling the macro once for each variable to TRANSPOSE; it might look something like the code to the right, which will again reproduce the data set shown in ***Exhibit 7.1***. This might well be the way to go if your application requires some flexibility with regard to what variables needed to be transposed.

```
%MACRO x1var(inds=long3a,byvar=ptid sex,
             idvar=visit,xvar=);
PROC TRANSPOSE DATA=&inds
     OUT=wide_&xvar (DROP= _NAME_ _LABEL_) PREFIX=&xvar;
BY ptid sex;
VAR &xvar;
ID &idvar ;
IDLABEL &xvar.label ;
RUN;
%MEND x1var ;

%x1var(xvar=sbp);  %x1var(xvar=dbp);  %x1var(xvar=whr);

DATA wide11x ;
 MERGE wide_sbp wide_dbp wide_whr ;
 BY ptid ;
RUN;
```

Further, as noted before, the

TRANSPOSE method does not require that you "know" what the maximum number of observations per BY group. Otherwise, whether you choose TRANSPOSE or the DATA step might be just a matter of personal preference.

EXAMPLE 9 – GOING FROM WIDE TO LONG…

So far all the examples have been different variations on going from a long, skinny (normalized) data set to a shorter, wider one. What if you need to go in the other direction – if for example you are starting with the data set shown in ***Exhibit 7.1***, and need to get to a data set that has one observation per visit? This might be the case if you are doing some type of repeated measures or other longitudinal analyses.

```
DATA long3b (KEEP = ptid visit systolic diastolic
                    waisthip);
 SET wide10 ;
 BY ptid ;

ARRAY sys{3} sbp1-sbp3 ;
ARRAY dia{3} dbp1-dbp3;
ARRAY wst{3} whr1-whr3;

DO visit = 1 TO 3;
   systolic = sys{visit} ;
   diastolic = dia{visit} ;
   waisthip = wst{visit} ;
   OUTPUT ;
END;
RUN;
```

For this task, I'd go straight to the DATA step toolbox. Instead of taking 3 observations per PTID and putting out just one, we are putting out one observation for each VISIT, and assigning the values from the correct positions in the ARRAYS to the measurement variables. The resulting data set is shown in ***Exhibit 9.1***.

Exhibit 9.1 Going from Wide to Long (Data set = LONG3b)

Obs	ptid	visit	systolic	diastolic	waisthip
1	01	1	142	92	0.88
2	01	2	141	91	0.87
3	01	3	131	83	0.83
4	02	1	107	58	0.75
5	02	2	.	58	0.75
6	02	3	111	55	0.71
7	03	1	135	80	0.97
8	03	2	.	.	.
9	03	3	128	74	0.94
10	04	1	145	84	.
11	04	2	145	84	0.71
12	04	3	139	79	0.68
13	05	1	136	86	1.00
14	05	2	132	83	0.99
15	05	3	126	.	0.96

If you look carefully, you'll see that this data set is not identical to LONG3, shown in Exhibit 7.1. For one thing, it has 15 observations, rather than 14. This is because we put out an observation for each visit for each PTID even if they had no data from that VISIT. Now, we might want to do this, or we might only want to put out an observation if there was data for at least one of the measurements – or some other decision rule. Probably in a real study, you'd have some variable telling you whether the individual had a visit or not. A simple way to generate the data set identical to LONG3 is to make the OUTPUT statement in the code above conditional. For example, it could say…

```
IF N(systolic, diastolic, waisthip) > 0 THEN OUTPUT;
```

The problem with using PROC TRANSPOSE for this task of wide to long transposition is that TRANSPOSE doesn't know anything about the structure of your data set…it can't reverse the ID statement so to speak, and create a VISIT variable from the suffixes (1, 2, 3) of the measurement variables. This is not to say that you might not want to TRANSPOSE to go wide to long, but the result will be different. You can experiment…see what happens if you use PTID as an ID variable, or as the BY variable…You might see results that could be useful in some contexts.

CONCLUSIONS

My intention in this paper has been to take some of the guesswork out of using PROC TRANSPOSE, demonstrating some of its different features, as well as providing DATA step syntax that, sometimes, accomplishes the same tasks as TRANSPOSE more simply. As with most data manipulation jobs in SAS, when it comes to reshaping your data from long to wide or vice versa, there are multiple means to the same end. Each may have its advantages or disadvantages in terms of clarity, efficiency (processing or programming), flexibility – or programmer preference, but it's good to be aware of different paths – some may be more adaptable to particular variations than others. And, while I am a firm believer in reading the manual – and reading papers written by other users – sometimes there is no substitute for experimentation. Take a small data set, such as the one at the end of this paper – and *PLAY*…Not only are you likely to really get how TRANSPOSE works after doing this, you are likely to see a way that it might generate something that could be useful to you in the future!

Speaking of reading papers by other users, there have been MANY papers written on the subject of data transposition, both with and without PROC TRANSPOSE. A selection of these that I have found useful and ones that I've referenced in this paper are listed below. Enjoy!

REFERENCES

1. Leighton, Ralph W. *Some Uses (and Handy Abuses) of PROC TRANSPOSE. SUGI 29.*
2. Virgile, Bob. *Changing the Shape of Your Data – PROC TRANSPOSE vs. ARRAYS*. NESUG 1998.
3. Williams, Christianna. *SYMPLIFY your Data Set Transposition with SYMPUT, and Make it Data-Driven Too!.* NESUG 2005.

ACKNOWLEDGMENTS

CONTACT INFORMATION

I welcome comments, suggestions and questions at:

Christianna S. Williams, PhD
Christianna_Williams@abtassoc.com

Appendix – Complete listing of data set … in case you want to play!!

ptid	group	sex	visit	sbp	dbp	whr
01	3	F	1	142	92	0.88
01	3	F	2	141	91	0.87
01	3	F	3	131	83	0.83
02	1	F	1	107	58	0.75
02	1	F	2	.	58	0.75
02	1	F	3	111	55	0.71
03	2	M	1	135	80	0.97
03	2	M	3	128	74	0.94
04	2	F	1	145	84	.
04	2	F	2	145	84	0.71
04	2	F	3	139	79	0.68
05	3	M	1	136	86	1.00
05	3	M	2	132	83	0.99
05	3	M	3	126	.	0.96
06	2	M	1	178	83	1.05
06	2	M	2	176	81	1.02
06	2	M	3	176	81	1.02
07	1	M	1	113	74	0.82
07	1	M	2	113	74	0.82
07	1	M	3	114	75	.
08	2	M	1	169	74	0.96
08	2	M	2	168	74	0.96
08	2	M	3	161	68	0.93
09	1	M	1	120	77	1.04
09	1	M	2	119	77	1.04
09	1	M	3	103	63	0.97
10	1	M	1	125	50	0.87
10	1	M	2	118	45	0.84
10	1	M	3	120	46	0.85
11	3	F	1	150	74	0.91
11	3	F	2	149	73	0.90
11	3	F	3	149	73	0.89
12	2	F	1	119	66	0.83
12	2	F	2	117	64	0.83
12	2	F	3	101	51	0.76
13	3	F	1	.	.	0.78
13	3	F	2	138	82	0.80
13	3	F	3	136	80	0.76
14	2	M	1	130	80	0.88
14	2	M	2	129	79	0.88
14	2	M	3	122	73	0.85
15	3	M	1	169	93	1.12
15	3	M	2	160	86	1.08
15	3	M	3	156	83	1.07
16	3	M	2	194	98	1.19
16	3	M	3	194	98	1.19
17	2	F	1	125	81	0.81
17	2	F	2	116	74	0.77
18	3	F	1	148	86	1.03
18	3	F	2	140	79	1.00
18	3	F	3	145	83	1.02
19	1	F	1	110	78	0.56
19	1	F	2	113	81	0.58
19	1	F	3	114	81	0.58
20	3	F	1	141	100	0.84
20	3	F	2	142	101	0.85
20	3	F	3	147	105	0.87
21	1	F	1	.	61	0.66
21	1	F	2	113	61	0.66
21	1	F	3	120	66	0.69
22	1	F	1	99	58	0.56
22	1	F	2	98	57	0.55
22	1	F	3	100	59	0.59
23	1	F	1	98	63	0.68
23	1	F	2	91	57	0.65
23	1	F	3	90	56	0.65
24	2	M	1	146	101	0.92
24	2	M	2	140	96	0.90
24	2	M	3	144	99	0.91
25	3	M	1	164	.	1.17
25	3	M	2	168	117	1.19
25	3	M	3	166	115	1.18
26	3	M	1	173	80	1.26
26	3	M	2	174	81	1.29
27	1	M	1	106	48	1.00
27	1	M	2	104	46	0.99
27	1	M	3	93	38	0.95
28	2	M	1	157	87	1.07
28	2	M	2	157	87	1.06
29	2	F	1	139	77	0.79
29	2	F	2	139	77	0.79
29	2	F	3	143	80	0.81
30	1	M	1	134	45	1.10
30	1	M	2	139	49	1.12
30	1	M	3	147	55	1.16