Paper BB08-2014

Five Ways to Flip-Flop Your Data

Joshua M. Horstman, Nested Loop Consulting, Indianapolis, IN

ABSTRACT

Data is often stored in highly normalized ("tall and skinny") structures that are not convenient for analysis. The SAS® programmer frequently needs to transform the data to arrange relevant variables together in a single row. Sometimes this is a simple matter of using the TRANSPOSE procedure to flip the values of a single variable into separate variables.

However, when there are multiple variables to be transposed to a single row, it might require multiple transpositions to obtain the desired result. This paper describes five different ways to achieve this flip-flop, explains how each method works, and compares the usefulness of each method in various situations. Emphasis is given to achieving a data-driven solution that minimizes hard-coding based on prior knowledge of the possible values each variable can have and that improves maintainability and reusability of the code.

The intended audience is novice and intermediate SAS programmers who have a basic understanding of the DATA step and the TRANSPOSE procedure.

INTRODUCTION

The primary objective of a SAS programmer is often to perform a statistical analysis of some data. However, the data frequently needs to be transformed, manipulated, and rearranged to produce something that is ready for an analysis. In many cases, it is necessary to transpose certain data elements so that information stored in rows is moved into columns, or vice-versa. The principal tool used by the SAS programmer to carry out this operation is the TRANSPOSE procedure.

Sometimes a single PROC TRANSPOSE will do the trick. Other times, the situation is more complex. In this paper, we will examine a case in which there are multiple variables to be transposed to a single row. We'll look at five different ways to accomplish the task and discuss some of the advantages and disadvantages of each.

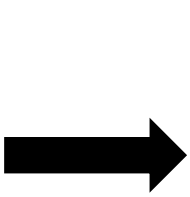
PROLOGUE: THE TRIVIAL CASE

Before we dig into a discussion of multiple transpositions, let us first consider the trivial case in which we need to transpose only one variable. In such cases, the methods described later in this paper are unnecessary. We can simply invoke the TRANSPOSE procedure and obtain the desired result in a single step.

For example, consider the data shown below in Figure 1. There is only one response variable to be transposed, namely PH. The values of SAMPLE are used to name the variables in the output data set, but this can be accomplished using the ID statement in PROC TRANSPOSE as follows.

SOILTEST_IN

LOCATION	SAMPLE	PH
A	1	6.8
A	2	6.6
A	3	6.5
B	1	5.2
B	2	5.2
B	3	5.3



SOILTEST_OUT

LOCATION	PH_1	PH_2	PH_3
A	6.8	6.6	6.5
B	5.2	5.2	5.3

Figure 1. SOILTEST Example

```
proc transpose data=soiltest_in out=soiltest_out(drop=_NAME_) prefix=ph_;
   by location;
   var ph;
   id sample;
run;
```

This example demonstrates the most basic usage of PROC TRANSPOSE. The purpose of this paper is to consider situations in which this functionality is insufficient to the task at hand.

THE PLOT THICKENS: A MORE COMPLEX EXAMPLE

Consider the data shown below in Figure 2. In this example, we have collected both LDL and HDL cholesterol for three subjects at each of two visits. We would like to obtain a data set having only one row per subject with all four cholesterol observations in the same row.

There are many reasons one might wish to perform such a transformation. For example, there could be a need to derive an additional variable using a calculation involving all four values. Alternatively, this arrangement may facilitate the creation of a particular report or graph.

In any case, this result cannot be accomplished with a single PROC TRANSPOSE. While the TRANSPOSE procedure will accept multiple variables on the VAR statement, it will not produce the output shown in Figure 2. The actual result will depend on which variables we include on the BY statement.

If we use SUBJECT as the only BY variable, the output data set will have six rows, one for each combination of subject and cholesterol type (LDL or HDL), and two columns, one for each visit. If we include both SUBJECT and VISIT as BY variables, we will get an output data set having 12 rows with all 12 of the cholesterol observations in a single column. If we use no BY variables at all, the result is a data set having two rows, one containing all the LDL values and another with all the HDL values.

However, all hope is not lost. To the contrary, there are many ways to generate the desired output data set. In this paper, we will walk through five ways to get the job done and see how each method applies to our cholesterol example.

CHOLESTEROL_IN

SUBJECT	VISIT	LDL	HDL
1	1	115	33
1	2	112	43
2	1	136	51
2	2	121	50
3	1	99	57
3	2	100	59

CHOLESTEROL_OUT

SUBJECT	LDL_1	LDL_2	HDL_1	HDL_2
1	115	112	33	43
2	136	121	51	50
3	99	100	57	59

Figure 2. CHOLESTEROL Example

GENERAL CONSIDERATIONS

Before presenting the methods, it is worthwhile to consider how we will evaluate each method. All of the methods presented will produce the correct result, but there are other considerations. Our purpose is not to single out one method as being the best, but to simply compare the advantages and disadvantages of each. Different situations may favor one method or another. The savvy programmer keeps a variety of tools in his or her toolbox and understands when to use each one.

In some environments, it might be important to consider the performance characteristics of each method, especially if dealing with very large amounts of data and limited system resources. While this is a worthy topic, it is outside the scope of this paper.

Our focus will instead be on the ease of maintaining and reusing the code. All else being equal, it is preferable to produce code that can more easily be used in a wide variety of situations without extensive changes. As much as possible, it is helpful to make the code data-driven so that it has some capacity to adapt to different types of input.

Obviously, there is a trade-off between writing code that adapts to the data and writing code that is simple and doesn't require much development time. Nonetheless, a little thought and planning upfront can often pay off handsomely in time and effort saved in the future.

Since we will be applying each method to the cholesterol example shown above, we will consider each method in light of the following questions:

- What changes are required if we add an additional value of the BY variable (SUBJECT)?
- What changes are required if we add an additional value of the ID variable (VISIT)?
- What changes are required if we add an additional response variable to be transposed? For example, in addition to LDL and HDL, what if we also need to transpose TRIG (triglycerides)?
- What changes are required if we add an additional ID variable? For example, what if two samples were taken at each VISIT, and we wish the output record to contain a separate column for each combination of VISIT and SAMPLE?

METHOD 1: ONE TRANSPOSE PER VARIABLE

The first method we will consider is to transpose each response variable separately and then merge the results back together. This process is shown below in Figure 3 using the cholesterol example. Each variable is placed in turn on the VAR statement of its own PROC TRANSPOSE. The respective output data sets are then merged together by subject in a final DATA step as shown in the following code. Note that each PROC TRANSPOSE uses the PREFIX= option with a unique prefix to ensure that the resulting variable names are different for each transposed data set. Unique variable names are essential for the merge to operate correctly. Alternatively, this could have been achieved by renaming the variables using the RENAME= data set option on the MERGE statement.

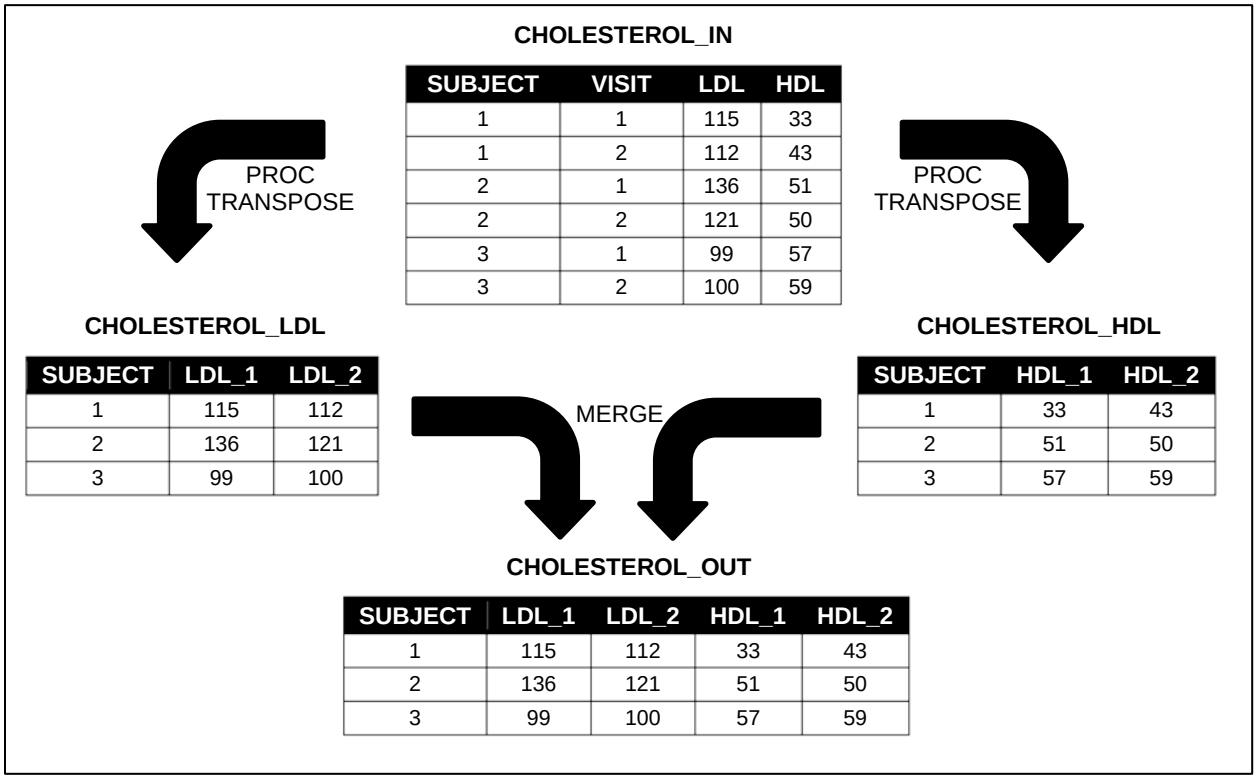


Figure 3. Logical flow of Method #1 using CHOLESTEROL data

```
proc transpose data=cholesterol_in out=cholesterol_ldl(drop=_NAME_) prefix=ldl_;
   by subject;
   var ldl;
   id visit;
run;
```

```
proc transpose data=cholesterol_in out=cholesterol_hdl(drop=_NAME_) prefix=hdl_;
   by subject;
   var hdl;
   id visit;
run;

data cholesterol_out;
   merge cholesterol_ldl cholesterol_hdl;
   by subject;
run;
```

This method is easy to understand and produces the correct result. It is also robust with respect to the levels of the BY variable (SUBJECT) and ID variable (VISIT). We could add more subjects and/or visits arbitrarily without requiring a single change to the code. On the other hand, this method does not scale well when there are more response variables since it requires a separate PROC TRANSPOSE for each variable to be transposed. This limitation can be overcome by using the macro language to generate the PROC TRANSPOSE code in a %DO loop, but this requires that the code be placed inside a macro definition.

The impact of adding another ID variable depends on the version of SAS being used. Starting with version 9.2, PROC TRANSPOSE will accept multiple variables on the ID statement, so it would be a simple matter of adding the new variable to each one. Prior to version 9.2 only one ID variable is allowed, so it would be necessary to construct an artificial ID variable by concatenating the values of the individual ID variables in some manner. This would require the addition of another DATA step at the beginning of the code.

METHOD 2: SINGLE TRANSPOSE

The second method is a variation on the first method that eliminates the need to call PROC TRANSPOSE separately for each variable to be transposed. Instead, all response variables are included on the VAR statement of a single PROC TRANSPOSE. For our cholesterol example, this results in an intermediate data set with a row for each subject and cholesterol type (LDL and HDL) and a column for each value of VISIT, our ID variable. Next, this data set is split into disjoint pieces which are merged together. The splitting is accomplished using the WHERE= data set option on the MERGE statement of a DATA step. It is also necessary to use the RENAME= data set option to assign a unique name to each variable being merged. The code is shown below and the logical flow is shown in Figure 4 below.

```
proc transpose data=cholesterol_in out=cholesterol_trans;
   by subject;
   var ldl hdl;
   id visit;
run;

data cholesterol_out(drop=_:);
   merge cholesterol_trans(where=(_NAME_="ldl") rename=(_1=ldl_1 _2=ldl_2))
         cholesterol_trans(where=(_NAME_="hdl") rename=(_1=hdl_1 _2=hdl_2));
   by subject;
run;
```

Like the first method, this one can handle any values of SUBJECT without changes to the code. However, unlike the first method, the same is not true with respect to the values of VISIT. Each unique value of visit results in a RENAME= data set option on each data set in the MERGE statement. If we add or remove visits, we have to be sure to adjust the code accordingly. Fortunately, this method makes it slightly easier to add a new response variable than the first method. Since there is only one PROC TRANSPOSE, we only need to add the new variable to the VAR statement. We also have to add a corresponding data set to the MERGE statement with the appropriate WHERE= and RENAME= data set options.

Adding a new ID variable under this method is the same as with the first method. With SAS 9.2 or later, it's a simple matter of using multiple variables on the ID statement. Prior to SAS 9.2, we must add an initial DATA step to create an artificial identifier as we did with the first method.

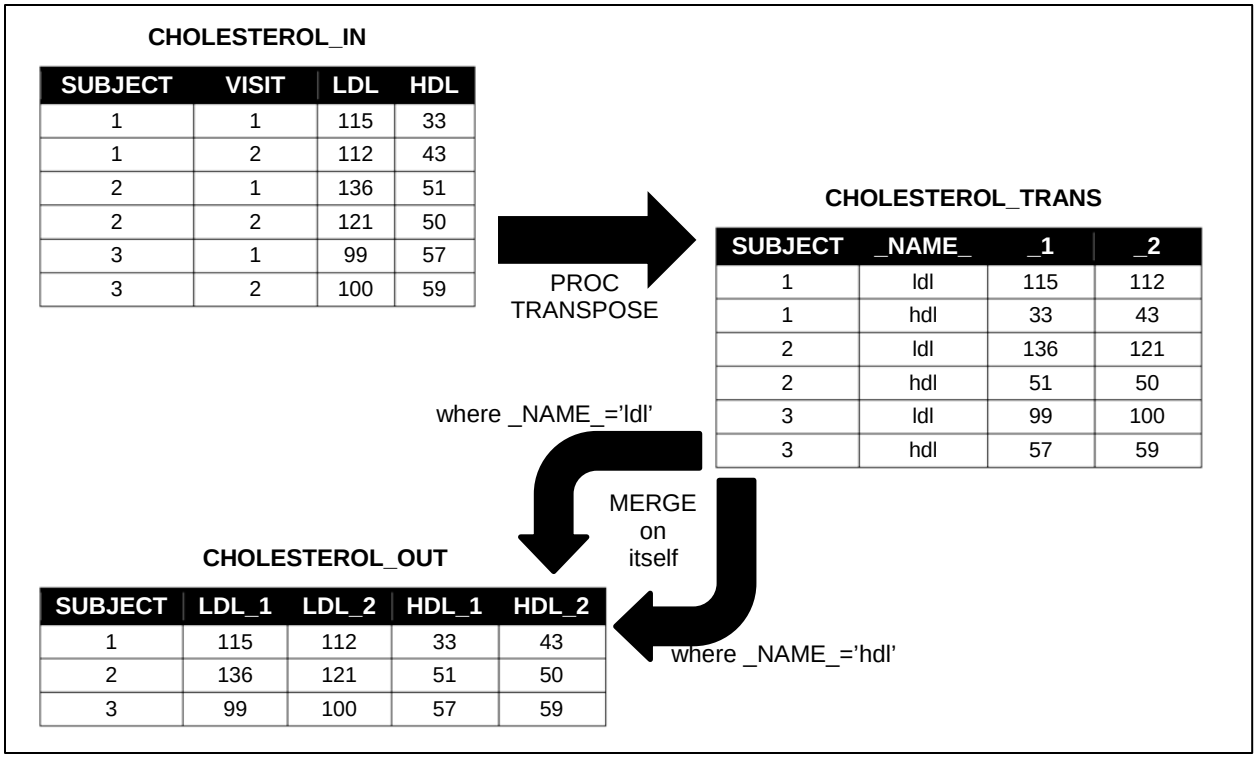


Figure 4. Logical flow of Method #2 using CHOLESTEROL data

METHOD 3: DOUBLE TRANSPOSE

Our third method will call PROC TRANSPOSE exactly twice, regardless of the number of variables being transposed. The first PROC TRANSPOSE completely normalizes the data so that all of the values being transposed are in a single column. The second PROC TRANSPOSE then transforms this into the desired output. In between the two transpositions, we use a DATA step to create an artificial identifier which will be used as the ID variable on the second PROC TRANSPOSE. In our cholesterol example, the identifier is the concatenation of the cholesterol type and the visit number with an underscore in between. The values of this identifier variable will become the variable names in the output data set. The code is shown below and the logical flow is displayed in Figure 5.

```
proc transpose data=cholesterol_in out=cholesterol_trans;
   by subject visit;
   var ldl hdl;
run;

data cholesterol_trans2;
   set cholesterol_trans;
   idvar = catx('_',_NAME_,visit);
run;

proc transpose data=cholesterol_trans2 out=cholesterol_out(drop=_NAME_);
   by subject;
   var col1;
   id idvar;
run;
```

In version 9.2 of the SAS software, PROC TRANSPOSE was enhanced to allow for multiple variables on the ID statement. As a result, it is no longer necessary to include the intermediate DATA step to construct a single ID variable (SAS Institute, "Sample 44637"). The code below has been modified accordingly. Note that PROC TRANSPOSE will not insert the underscore character between the cholesterol type and visit number. If we wish to adhere to our naming convention, we can use the RENAME= data set option on the OUT= option of PROC

TRANSPOSE to rename the variables as desired.

```
proc transpose data=cholesterol_in out=cholesterol_trans;
   by subject visit;
   var ldl hdl;
run;

proc transpose data=cholesterol_trans out=cholesterol_out(drop=_NAME_);
   by subject;
   var col1;
   id _NAME_ visit;
run;
```

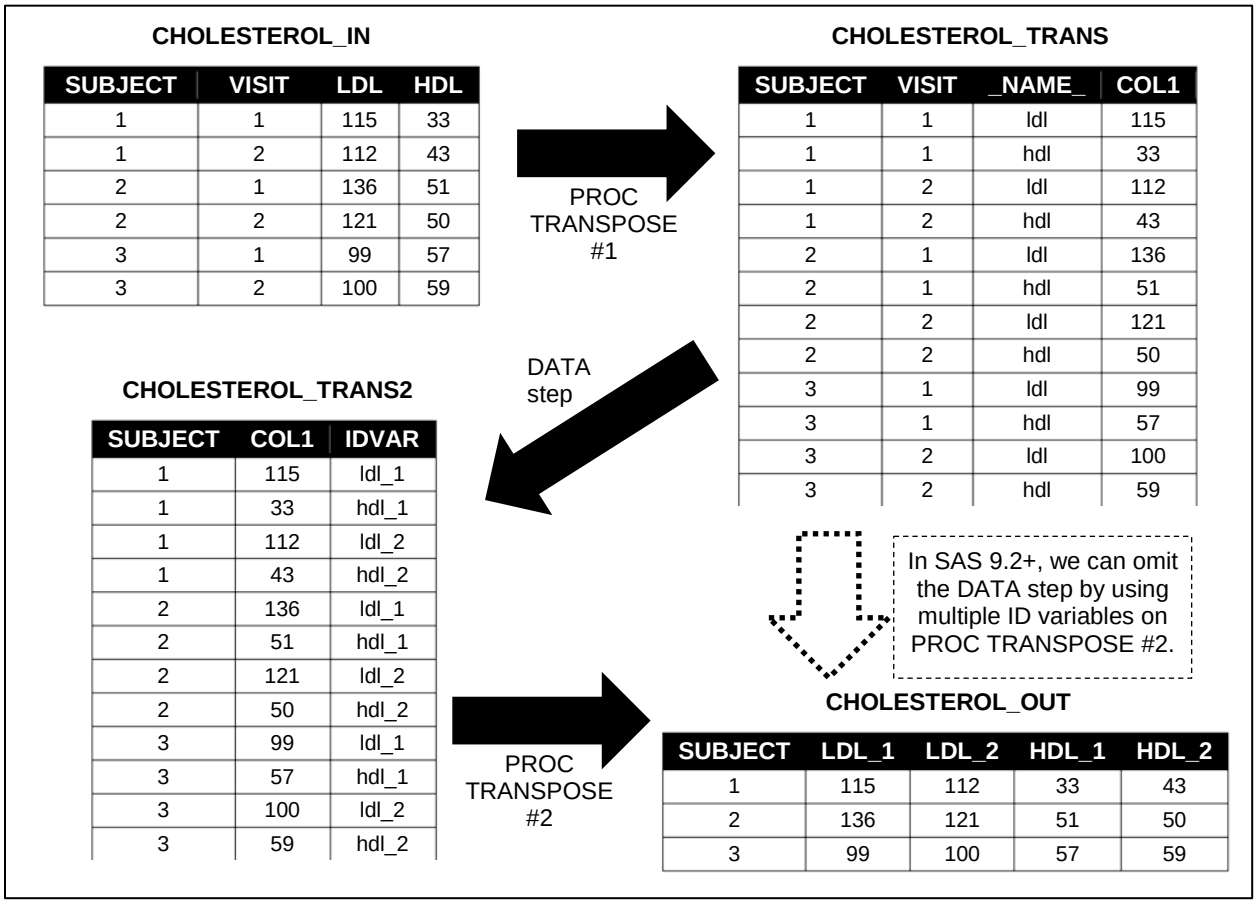


Figure 5. Logical flow of Method #3 using CHOLESTEROL data

Regardless of which version of SAS is being used, this method holds up fairly well in response to changes to the structure of the incoming data. Both subjects and visits can be added or removed with impunity. Adding a new response variable requires only the slight change of adding the variable name to the VAR statement in the first PROC TRANSPOSE. Similarly, adding a new ID variable requires only two changes. The new ID variable must be added to the BY statement of the first PROC TRANSPOSE and it must also be included as part of the construction of the IDVAR variable created in the DATA step (or simply included on the ID statement of the second PROC TRANSPOSE in SAS 9.2 or later).

METHOD 4: BRUTE FORCE USING ARRAYS

In our fourth method, we will see that transposing data doesn't require the use of PROC TRANSPOSE at all. Such transformations can also be performed using the DATA step. This requires the programmer to manually perform much of the work normally done by PROC TRANSPOSE. This can be characterized as a "brute force" approach, but it has the advantage of being very flexible.

There are many ways to accomplish this task within a DATA step. Our approach will rely on the use of arrays. We create an array corresponding to each response variable. The dimension of each array matches the number of rows in the input data set in each BY group. In our cholesterol example, the arrays have a dimension of two because there are two visits per subject. We process all of the records in each BY group, copying the values to the appropriate places in the arrays. At the beginning of each BY group, we initialize the array values to missing so that we don't inadvertently carry over values from the previous group. Once an entire BY group has been read, a single record is written to the output data set using the values stored in the arrays.

```
data cholesterol_out;
   keep subject ldl_1-ldl_2 hdl_1-hdl_2;
   set cholesterol_in;
   by subject;
   array ldlarray(2) ldl_1-ldl_2;
   array hdlarray(2) hdl_1-hdl_2;
   retain ldl_1-ldl_2 hdl_1-hdl_2;
   if first.subject then do i=1 to 2;
      ldlarray[i]=.; hdlarray[i]=.;
      end;
   ldlarray[visit] = ldl;
   hdlarray[visit] = hdl;
   if last.subject then output;
run;
```

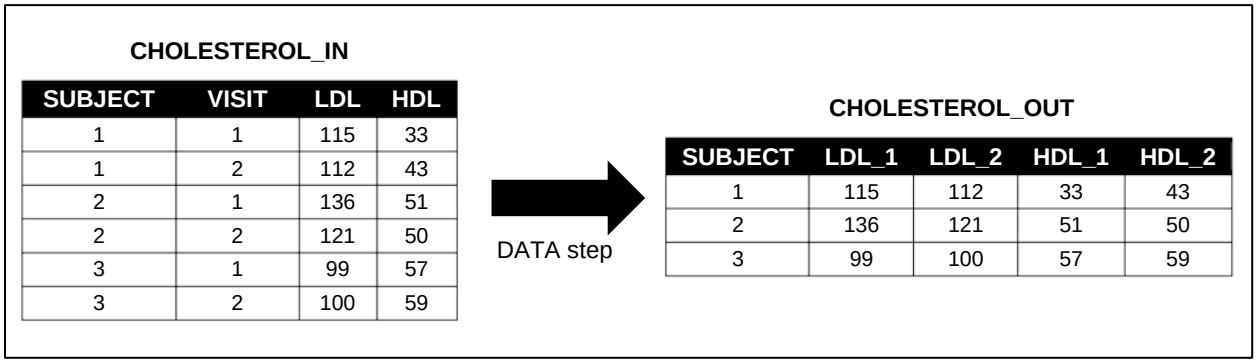


Figure 6. Logical flow of Method #4 using CHOLESTEROL data

The vast flexibility of this method also turns out to be its Achilles' heel in terms of extending the code to work with other data. Because the logic is highly specific, significant changes are required. As with all the prior methods, adding or removing subjects is not an issue. Adding or removing visits, however, requires changes to the array definitions, modification of the RETAIN and KEEP statements, and adjustment of the loop boundaries.

It should also be noted that the code shown above benefits from the fact that our visit numbers are sequential integers. Were this not the case, our array definitions, RETAIN statements, and KEEP statements would not be able to make use of the variable list notation. Instead, we would need to list each variable explicitly in each of those places. We also would not be able to use VISIT as the subscript in an array reference, which is a significant limitation.

Adding a new response variable requires another array definition, additional assignment statements, and changes to the RETAIN and KEEP statements. Things get really hairy if we decide to add an additional ID variable to this method. Multidimensional arrays and more complex conditional logic are needed to track all of the values as they are read in and to assign them to the correct place in the record being written to the output data set.

METHOD 5: PROC SQL

The final method we will consider uses the SQL procedure to perform the transposition. During the processing of the SQL query, the desired output variables are created for every row in the input data set but only those variables pertaining to the data in that row are populated, while the others are set to missing. However, our SQL query includes a GROUP BY clause so that only one row will be returned for each BY group, and that row will consist of the maximum value of each variable within that group. Since non-missing values are by definition larger than missing values, this has the effect of collapsing only the non-missing values into a single row for each BY group. A conceptual representation of this process is shown in Figure 7.

```
proc sql noprint;
   create table cholesterol_out as
      (select subject,
         max(case when visit = 1 then ldl else . end) as ldl_1,
         max(case when visit = 2 then ldl else . end) as ldl_2,
         max(case when visit = 1 then hdl else . end) as hdl_1,
         max(case when visit = 2 then hdl else . end) as hdl_2
      from cholesterol_in group by subject);
   quit;
run;
```

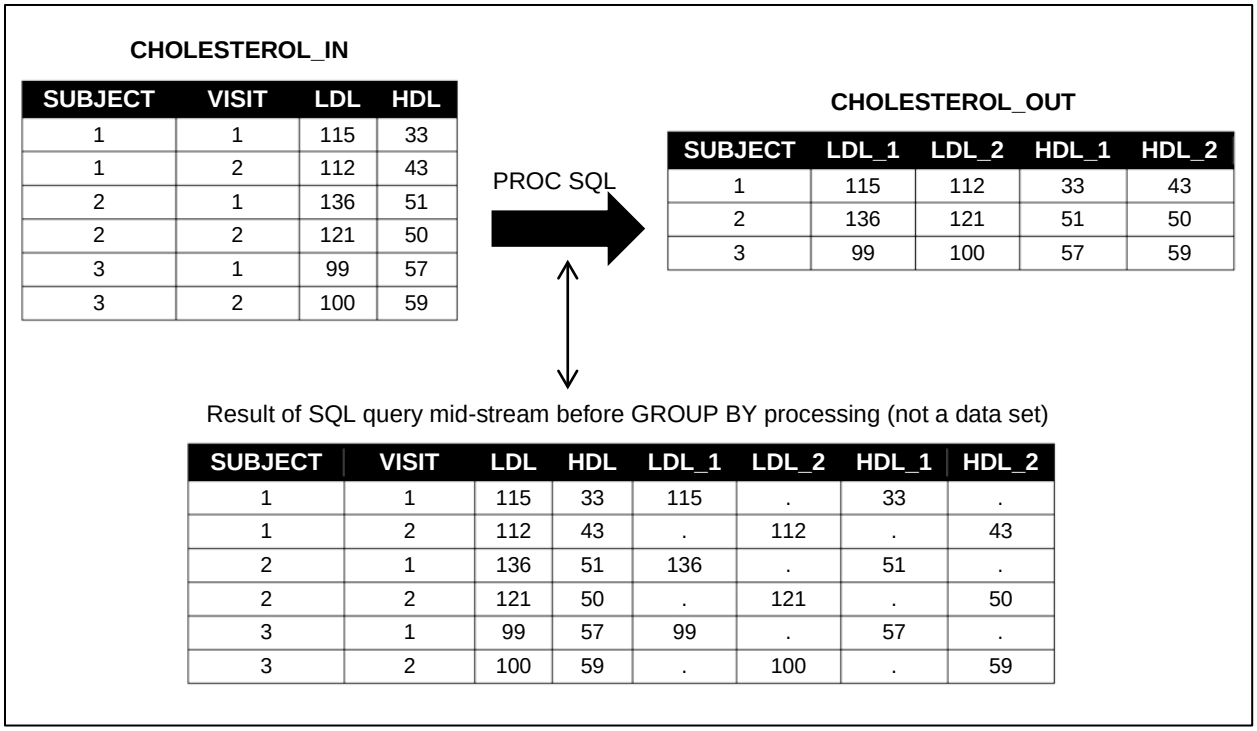


Figure 7. Logical flow of Method #5 using CHOLESTEROL data

Like the previous method, this method also does not extend itself easily. While subjects can be added or removed without any changes to the code, adding another visit or another response variable requires additional terms to be added to the SELECT statement within the call to PROC SQL. Adding an additional ID variable will require having a term for each combination of values of the ID variables. If your ID variables have more than just a few values, this can become quite cumbersome. Conway (2008) proposes a macro to solve this problem.

CONCLUSION

All of the methods described will produce the desired results, but there are other factors to consider when selecting a method to use for a job. The table below summarizes our findings pertaining to the extensibility of the code for each method.

Method	Changes required to the code if we add another…			
	value of a BY variable	value of an ID variable	response variable	ID variable
#1. One Transpose Per Variable	None	None	Additional PROC TRANSPOSE	Pre-9.2: Additional DATA Step 9.2+: Modify ID statements
#2: Single Transpose	None	Additional RENAME= data set options	Modify VAR statement and MERGE statement	Same as above
#3: Double Transpose	None	None	Modify VAR statement	Modify ID statement (and modify DATA step in pre-9.2)
#4: Brute Force using Arrays	None	Modify array definitions, RETAIN, KEEP statements	Add array definitions and assignment statements; modify RETAIN and KEEP statements	Use multi-dimensional arrays and more complex conditional logic
#5: PROC SQL	None	Add additional terms to SELECT statement	Add additional terms to SELECT statement	Add additional terms to SELECT statement

While none of these methods is the clear winner in terms of the ease with which the code can be extended and reused in the situations we considered, method #3 (two transposes) does appear to be the most robust in general. However, that is not to say that its use is recommended in all circumstances. There may be cases in which one of the other methods can better accommodate some peculiarity of the task at hand. Understanding how and why each method works will better equip the SAS programmer to make a judgment call when such a situation arises.

REFERENCES

Conway, Ted. "It's a Bird, It's a Plane, It's SQL Transpose!" *Proceedings of the SAS Global Forum 2008 Conference.* Cary, NC: SAS Institute Inc., 2008. Available at http://www2.sas.com/proceedings/forum2008/089-2008.pdf.

SAS Institute: "Sample 44637: Double PROC TRANSPOSE method for reshaping your data set with multiple BY variables." Available at http://support.sas.com/kb/44/637.html.

CONTACT INFORMATION

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

Name: Joshua M. Horstman
Organization: Nested Loop Consulting
Address: 8921 Nora Woods Drive
City, State ZIP: Indianapolis, IN 46240
Work Phone: 317-815-5899
Email: josh@nestedloopconsulting.com