Paper CC-17

Using Arrays to Handle Transposed Data

Michael Leitson, Wellstar Research Institute, Marietta, GA

ABSTRACT

Often times, particularly in the health care field, programmers and analysts need to deal with multiple repeated observations (or patients), with the task of searching for a desired string, number or other trait within a certain variable. PROC TRANSPOSE can be utilized to transpose the data so that each record has a unique observation, with the transposed variable displayed across multiple columns. Then an array is processed throughout the multiple columns, optionally using FIND, SCAN, SUBSTR or other character function, and returns a new variable with a value of 1 if the transposed variables contain the desired information. Additional code must be written to return a value of 0 if the transposed variables do not contain the desired information.

All techniques are intended for the working knowledge of the Base SAS® programmer, with the hope that the provided simple, straightforward array code will make the task of seeking characteristics of observations much easier.

INTRODUCTION

The use of arrays in SAS, in conjunction with the proper DO loops, can easily save a programmer a multitude of hours of repeated and unnecessary coding. Imagine that you are a Base SAS programmer in the health care industry and have elementary knowledge of SQL. Suppose that you are asked to analyze a data set from a clinician's office that includes the patient's unique ID, and any risk factors the patient has, among other variables. Since patients can visit the clinician's office multiple times, the same patient can make up numerous observations. The main goal is to find out what risk factors are more prevalent among these patients.

Since the patients take up varying number of rows, the most natural instinct would be to transpose the data, so that each unique patient only appears in one row. After transposing the data successfully, you discover that you have 300 risk factor columns (due to patients coming to the office many times over or having an abundance of risk factors), many of which are duplicates. You brainstorm about what to do next – you ultimately want a data set with only the patients' IDs and an indicator of if they have the risk factors or not. This is where arrays come in handy – the Base SAS programmer has seen many calculations done within arrays, but using IF-THEN-ELSE statements within arrays sheds a whole new capability and flexibility of arrays to the Base programmer.

Please note that the above scenario is from my own experience but this technique can be applied to any coder in all fields. Such examples include a business analyst needing to analyze several customer transactions, where customers can have more than one transaction, or manufacturing repeated products for quality control.

SYNTAX

Suppose that you have an initial SAS data set that is similar to the following:

ID	Risk Factor
1	Coronary Artery Disease
1	Congestive Heart Failure
1	Coronary Artery Disease
2	Hypertension
2	Diabetes
2	Diabetes
2	Diabetes
.	
.	
.	
1036	Leukemia
1036	Renal Insufficiency Syndrome
1036	Hypertension
1036	Diabetes
1036	Diabetes
1037	Hypertension
1037	Congestive Heart Failure
1037	Congestive Heart Failure

Table 1: Original Data Set

Basically, this data set has information about 1,037 patients and risk factors that each patient has. The next obvious task would be to transpose the data. If the data is not already sorted by the identifying variable, a PROC SORT step should be done to accomplish that. Next, the following code on the next page would be implemented to transpose the data:

```
PROC TRANSPOSE DATA=DATASET OUT=DATASET1;
       BY ID;
       VAR RISK_FACTOR;
RUN;
```

Note that the OUT option is optional, but it would be very wise to exercise this, as one error in coding can cause you to lose your data set. Now, your new data set should look like the following:

ID	Name of Former Variable	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8
1	Risk Factor	Coronary Artery Disease	Congestive Heart Failure	Coronary Artery Disease					
2	Risk Factor	Hypertension	Diabetes	Diabetes	Diabetes				
.									
.									
.									
1036	Risk Factor	Leukemia	Renal Insufficiency Syndrome	Hypertension	Diabetes	Diabetes			
1037	Risk Factor	Hypertension	Congestive Heart Failure	Congestive Heart Failure					

Table 2: Transposed Data Set

After this point has been reached, you begin to wonder if additional transformation is needed in order for each risk factor to appear only once within each patient. A quick glance at the actual data set in SAS conveys that the columns do not stop at COL8. One patient has 300 recorded risk factors; hence the many empty columns for all other patients. You want to assign patients the standard labeling of values – a 0 if they do not have a certain risk factor and a 1 if they do have the same risk factor. Certainly, there must be a way to achieve the desired task instead of writing 300 IF-THEN-ELSE statements for each risk factor.

That is where the beauty of the functionality of SAS comes in. The common Base SAS programmer knows the basic methods of coding, but the simple flexibility of SAS allows users to combine functions within one another. A better

method to achieve the task on hand would be to use IF-THEN-ELSE statements within an array. First, a review of the concepts of arrays in SAS is needed (SAS Institute Inc, 2011):

1. A SAS array exists only for the period of the DATA step.
2. Do not use variable names in the DATA step as the array name.
3. Remember to include the array dimension, followed by the variables, in order that they appear in the data set.

The first statement to code (aside from the DATA and SET statements) would be to write an array statement, with a corresponding DO statement:

```
ARRAY RISKS {300} COL1--COL300;

DO I = 1 TO 300;
```

You are simply assigning an array called 'RISKS' that is referencing the 300 columns in which the risk factors can be found. The fact that the columns are conveniently called COL1 through COL300 makes the task easier. The DO loop signals that the next lines of code will be performed on the array. After the DO loop, the following can be written:

```
IF RISKS(I) = 'Coronary Artery Disease' THEN CAD = 1;
IF RISKS(I) = 'Congestive Heart Failure' THEN CHF = 1;
IF RISKS(I) = 'Diabetes' THEN DIABETES = 1;
IF RISKS(I) = 'Hypertension' THEN HYPERTENSION = 1;
IF RISKS(I) = 'Leukemia' THEN LEUKEMIA = 1;
IF RISKS(I) = 'Renal Insufficiency Syndrome' THEN RIS = 1;
```

The alternative coding assignment of the 0's should not be written as the ELSE statement below each corresponding IF statement, as that would lead to erroneous results. Assignment of the 0's for the non-risk factors will take place in a moment. For the new data set, however, the 300 columns holding the risk factors are not of any concern, so we can drop them, which is essentially the same as keeping the desired variables.

```
KEEP ID CAD CHF DIABETES HYPERTENSION LEUKEMIA RIS;
END;
```

An END statement is necessary to enclose the DO loop. Now the assignment of the 0's is necessary, as it is much easier to work with 0's rather than missing values. At this point, the new data set that is still in process of being created has a 1 for each risk factor that the patient has and a period (.) if the patient does not have a certain risk factor. You can simply write another array statement to take care of the non-risk factors. However, note that it is not needed to reference the original 300 columns, as those columns are not part of the data set now. Instead, the missing values will appear in the 6 newly created columns. So an array should be written that references these 6 columns.

```
ARRAY NONRISKS {6} CAD--RIS;
       DO I = 1 TO 6;
       IF NONRISKS(I) = . THEN NONRISKS(I) = 0;
END;
```

Note that it is very important to reference the columns in the array exactly as you coded them in the original IF statements. This is why it is recommended to keep the variables, rather than dropping them, so you can make sure you are counting the correct number of variables in the arrays by double-checking. After you have concluded the process with a RUN statement, the newly created data set should look like the following:

ID	CAD	CHF	DIABETES	HYPERTENSION	LEUKEMIA	RIS
1	1	1	0	0	0	0
2	0	0	1	1	0	0
.						
.						
.						
1036	0	0	1	1	1	1
1037	0	1	0	1	0	0

Table 3: Final Data Set

An ideal data set has been reached! It can be readily identified which patients have predictors and which do not. You can now perform frequency table analysis on all these predictors with much more ease than the original data set. If there is an outcome variable, such as a certain brand of medicine or the number of hospital visits, you can perform regression analysis to see which predictors are significant.

OTHER TIPS

Often times, the data will not be as clean as the examples used. Strings may be misspelled or numeric data may be used. Some strings may be capitalized, other may not. For example, if you have 'hypertension' as one risk factor and 'Hypertension' as another risk factor, the human brain knows that those two strings are equivalent, but SAS initially will not. To overcome this, use:

```
IF FIND(RISK(I),'hypertension','i') > 0 THEN HYPERTENSION = 1;
```

The FIND function can be used to ignore case sensitivity when 'i' is denoted in the 3rd argument (note that this is vastly different than the 'I' specified in the DO loop). Other times, when using numeric data, such as ICD-9 codes, the goal will be to search for numbers in a certain order within a specific string. The ICD-9 code for hypertension depends on what specific kind of hypertension one is referring to, but it can take values of 401.1, 401.9, etc., with 401 referring to general hypertension. A code can simply be written to generalize this:

```
IF SUBSTR(RISK(I),1,3) = '401' THEN HYPERTENSION = 1;
```

A SCAN function or other character function can also be used.

CONCLUSION

Arrays are useful tools for saving time to repeatedly write code. When combined with IF-THEN-ELSE statements, arrays can be utilized to bring transposed data with repeating variables to a single, clean data set that informs the user of the characteristics of the observations. Proper coding is the key here, so a coder must remember the rules of arrays and to code the 1s before the 0's. In the instance that a substring is desired, then a simple SUBSTR, SCAN or FIND function can be nested within the IF-THEN-ELSE statement. SAS has powerful flexibility and functionality when combining two or more concepts. The possibilities for the Base programmer are endless.

REFERENCES

SAS Institute Inc. 2011. *SAS® Certification Prep Guide: Base Programming for SAS®9, Third Edition*. Cary, NC: SAS Institute, Inc.

ACKNOWLEDGEMENTS

Thanks to Dr. Louise Lawson for helping with the syntax and sparking the idea to write about it.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author by email at:
Michael.Leitson@wellstar.org.

TRADEMARK CITATION

APPENDIX

The following is the full SAS code that was used as the primary component for obtaining the desired data set:

```
*TRANSPOSING THE DATA;;

PROC TRANSPOSE DATA=DATASET OUT=DATASET1;
BY ID;
VAR RISK_FACTOR;
RUN;



DATA DATASET2;
SET DATASET1;

*CREATING THE ARRAY TO HANDLE THE 1 VALUES;

ARRAY RISKS {300} COL1-COL300;
DO I = 1 TO 300;
IF RISKS(I) = 'CAD' then CAD = 1;
IF RISKS(I) = 'Congestive Heart Failure' then CHF = 1;
IF RISKS(I) = 'Diabetes' then DIABETES = 1;
IF RISKS(I) = 'Hypertension' then HYPERTENSION = 1;
IF RISKS(I) = 'Leukemia' then LEUKEMIA = 1;
IF RISKS(I) = 'Renal Insufficiency Syndrome' then RIS = 1;

*KEEPING THE DESIRED VARIABLES;

KEEP ID  RIS CHF CAD DIABETES HYPERTENSION LEUKEMIA ;
END;

*CREATING THE ARRAY TO HANDLE THE 0 VALUES;

ARRAY NONRISKS {6} CAD-RIS;
DO I = 1 TO 6;
IF NONRISKS(I) = . THEN NONRISKS(I) = 0;
END;
RUN;
```