Paper 234-31

The TRANPOSE Procedure or How to Turn It Around

Janet Stuelpner, Left Hand Computing, Inc., New Canaan, CT

ABSTRACT

So many times we need to take our data and turn it around. One of the reasons that this is done is that it is more efficient to store your data in a vertical format and processing the data is easier in a horizontal format. That means that we need to change the format of the data before we process or analyze it. There are many ways to accomplish this in a DATA step. Another way to change the data is to use a PROC TRANSPOSE. This paper will show you, step by step, how to change the format of the data. You will be taken from the easiest way of doing without any options to a more complex manner using a whole host of options.

INTRODUCTION

The transpose procedure restructures the data by changing the variables into observations. How this is done and what variables are chosen to transform are determined by the options that are chosen when running the procedure. The TRANSPOSE procedure can eliminate the need to write a complex DATA step that requires the use of one or more PROC SORT. The output from the procedure is a data set that contains the transposed data or reformatted data. The output data set can be used for analysis, reporting or further manipulation of the data. The output from the procedure can be used in other reporting procedures such as PROC PRINT or PROC REPORT

VERTICAL VS HORIZONTAL DATA

We classify data as either vertical or horizontal. The format in which we store data is very different from the format that we need to analyze the data. There are very good reasons why we need to be able to change the layout of the data based on what we need to do with the data. The best method of data storage is to keep it in a vertical format. This is also called stacked data that has a few variables and many rows. There is an identifier on each record with a little bit of information. If there is a piece of information that is missing, there will not be a row in the data. Take a look at the example below. You will see that there are only records for a student if they took a class. There aren't any rows that are blank.

STUDENT	CLASS	GRADE	CREDIT
Ann	Math101	A	4
Ann	English101	B+	4
Ann	Biology101	B+	4
Ann	French111	A	4
Ann	Biolab	A-	2
Bob	Math101	A	4
Bob	Chemisty101	A-	4
Bob	Chemlab	B	2
Carol	Spanish101	B	4
Carol	French101	B	4
Carol	History102	C	4
Carol	PoliSci111	B	4
David	Italian	C	4
David	Math210	C	4
David	Lit200	B	4
Fred	Chem101	B	4
Fred	Chemlab	B	2
Fred	Anthro111	C	4
Fred	Math110	A	4

Table 1: Vertical Data Example

The other format for our data is called horizontal. With this type of data layout, we can easily analyze the data because all of the variables that are needed for analysis are on the same record. Horizontal data has many variables with few rows. There will be empty cells in the data if there are any missing values. In the example above, we would have one record for each student with a variable for each class and a variable for each grade. If the data were formatted in a horizontal manner, it would be easy to calculate the grade point average for each student.

STUDENT	CLASS1	GRADE1	CLASS2	GRADE2	CLASS3	GRADE3	CLASS4	GRADE4
Ann	Math	A	English	B+	Biology	B+	French	A
Bob	Math	A	Chemistry	A-				
Carol	Spanish	B	French	B	History	C	PoliSci	B
David	Italian	C						

Table 2: Horizontal Data Example

Note in the example above, there are empty cells for Bob and David because they are not taking a full load of courses. Carol and Ann are taking 4 classes each and have every cell filled in.

So we can see through the examples above that the format for data storage is very different than the format for data analysis. We need a mechanism to transform the data back and forth, depending on our goal of either data storage or data manipulation and analysis.

DATA STEP MANIPULATION

How do we create many observations from one observation? In other words, how do we take data that is in a horizontal format and create a dataset that is in a vertical format. This is not very difficult but can take a great deal of coding. There are many ways to tackle this problem. There are two obvious methods to accomplish this. The first method shows careful use of the OUTPUT statement for each value than needs to be output. The OUTPUT statement is used several times for each row of the output dataset. Take a look at the example below.

Data for Examples:

```
data vitals;;
  input pat test $ visit1 visit2 visit3 visit4;
cards;
16 SBP 112 118 120 114
35 SBP 120 155 140 130
93 SBP 110 115 110 115
;
run;
```

Example 1:

```
data sbp;
   set vitals;
   if visit1 ne . then result=visit1;
   output;
   if visit2 ne . then result=visit2;
   output;
   if visit3 ne . then result=visit3;
   output;
   if visit4 ne . then result=visit4;
   output;
   keep pat test result;
run;
```

The second method is characterized by the use of arrays. Arrays allow us to use iterative processing with a minimum amount of code. The result is the same that we would get by using the code above. The major difference is that we don’t have to write multiple statements for each variable that we want to output as a line in the output dataset. In this example, we have four variables that we want to put out on each line. However, what if we wanted to put out 10 or 50 variables. The method above would be tedious because we would have to type 10 assignment statements and ten output statements. Arrays make it easier to code and produce the iterative processing with fewer statements. Let’s take a look at an example.

Example 2:

```
data sbp;
   set vitals;
   array vitals[4] visit1-visit4;
   do I=1 to 4;
      result=vitals(i);
      output;
   end;
   keep pat test result;
run;
```

If we run a PROC PRINT on the output dataset, we see that we now have a variable named RESULT that contains the values of all of the systolic blood pressures that were taken for each patient at each visit.

Output:

```
PAT  TEST  RESULT
 16   SBP    112
 16   SBP    118
 16   SBP    120
 16   SBP    114
 35   SBP    120
 35   SBP    155
 35   SBP    140
 35   SBP    130
 93   SBP    110
 93   SBP    115
 93   SBP    110
 93   SBP    115
```

How do we create one observation from many observations? In other words, how do we take data that is in a vertical format and create a dataset that is in a horizontal format? We can do this with a DATA step and array processing or we can use FIRST. And LAST. processing. If we choose to use FIRST. and LAST. processing, the data must be sorted first before the dataset is created. The idea is to read in all records for a subject and then output only the last record while we retain a bunch of information. If we choose to use array processing, we need to make sure that all of the data is of the same type because of the limitations of array processing. Along with that it is not a simple task as it has to be done with multidimensional arrays and nested DO loops. Because of the type of arrays that must be used, this can be extremely complicated. Let’s take a look at an example.

Data for Example:

```
data one;
  input student $ class $ grade $ credit $;
datalines;
Ann Math101 A 4
Ann Eng101 B+ 4
Ann Bio101 B+ 4
Ann Fren111 A 4
Ann Biolab A- 2
;
run;
```

Example:

```
data define ( drop=student class grade credit i j );
  set one nobs=nobs;
  array all(5,4)$ a1-a20;
  array vars(*) $ student class grade credit;
  retain a1-a20;
  i+1;
  do j=1 to 4;
    all( i,j )=vars( j );
  end;
 put a1-a20;  /* for information only */
  if _n_=nobs then output;
run;
```

Output:

```
A1   A2   A3  A4  A5   A6  A7  A8  A9   A10  A11  A12  A13  A14  A15  A16  A17  A18  A19  A20
Ann  M1   A   4   Ann  E1  B+  4   Ann  B1   B+   4    Ann  F1   A    4    Ann  BL   A-   2
```

This is all very easy to do with Proc Transpose. We are going to take a look at the various options that can be used with the procedure and how each option can be used to help us to create a dataset that transforms the data exactly the way we need to have it. We can take a look at the procedure with the use of any options all the way to using several to tailor the data so that it fits the analysis plan.

PROC TRANSPOSE

Let's start with the simplest process and work up to something more complex. We will use this simple vital statistics dataset in our examples.

PROTOCOL	INV	PAT	VISIT	VSDT	SBP	DBP	PULSE	WEIGHT
DRG2005	227	27001	1	6-Jan-04	122	80	65	119.1
DRG2005	227	27001	2	13-Jan-04	118	82	60	119.1
DRG2005	227	27001	3	16-Jan-04	120	72	66	119.1
DRG2005	227	27001	4	3-Feb-04	114	82	60	119.1
DRG2005	227	27001	9	4-May-04	104	70	60	119.1
DRG2005	227	27001	10	10-May-04	118	76	66	119.1
DRG2005	227	27001	11	1-Jun-04	113	81	62	119.1
DRG2005	227	27001	12	29-Jun-04	122	76	66	119.1
DRG2005	227	27001	13	27-Jul-04	110	76	72	119.1
DRG2005	227	27001	14	23-Aug-04	124	82	60	119.1
DRG2005	227	27001	15	28-Sep-04	116	76	66	119.1
DRG2005	227	27002	1	26-Jan-04	136	86	72	110.4
DRG2005	227	27002	2	2-Feb-04	136	80	84	110.4
DRG2005	227	27002	3	5-Feb-04	136	84	84	110.4
DRG2005	227	27002	4	19-Feb-04	138	76	82	110.4
DRG2005	227	27002	9	5-Apr-04	112	80	72	110.4

Table 3: Vital Statistics Vertical Dataset

We will now explore some of the important options and statements in the transpose procedure. The simplest case is to use Proc Transpose without any options at all. What happens when we do this?

This data is stored in a vertical format and we need to transform it into a horizontal format. Notice that each subject has multiple records and that the identifier information is the protocol number, investigator number and patient identifier. There is one record per subject per visit. If a subject has 5 visits, there will be 5 records for that subject. Take a look at the code below. The first use of the transpose procedure will not use any options.

```
        proc transpose data=vitals; run;
        proc print; run;
```

Since Proc Transpose always produces an output dataset, we need to run a Proc Print to see the data. If you do not specify a name with the OUT= option, SAS will give it a default name. So which variables in the original dataset are transposed? Without any additional statements, the procedure will transpose only numeric variables. In this example all of the variables other than the identifiers are transposed because they are all numeric variables. There is one row in the output dataset for each numeric variable in the original dataset. So we have one row for the visit, one for the visit date, one for the systolic blood pressure, one for the diastolic blood pressure, one for the pulse and one for the weight. Take notice of the variable VSDT. This is a date variable that is stored with the DATE9. format attached to it. As you can see in the sample above, when the data is printed, the date is formatted to read ddmmmyy. However, in the print out below, the date is the original value of the number of days since January 1, 1960. The formatting is lost after the procedure is invoked, in other words, the date is now unformatted. We can try to change it to a character variable, however, the default for the transposition is to transpose only numeric variables. There must be a better way! SAS includes some default variables in the output dataset. The first is _NAME_ which is the name of the variables that were transposed. The next default variable is _LABEL_ which takes the label of the variables and puts them in a variable itself. The last thing to note is the variable names. Each variable was given a default name beginning with the characters COL and a number (in reality, this dataset has 775 columns and the printout is only a subset of the output dataset). We have lost the variables and associated values of the protocol, investigator and patient. The new dataset has one column for each visit of each patient. The only way to tell where one patient ends and the next patient begins is that the visit number (which is the first row of data) starts again at one after all of the data is displayed for a patient. This makes the evaluation of the data very confusing. So, what do we do to fix this? We need to add some options to our program so that the data in the output dataset is very clear.

NAME	_LABEL_	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10	COL11
VISIT	Visit Number	1	2	3	4	9	10	1	2	3	4	9
VSDT	Date of Vital Signs	16076	16083	16086	16104	16195	16201	16223	16251	16279	16306	16342
SBP	Systolic BP	122	118	120	114	104	118	113	122	110	124	116
DBP	Diastolic BP	80	82	72	82	70	76	81	76	76	82	76
PULSE	Pulse Rate	65	60	66	60	60	66	62	66	72	60	66
WEIGHT	Weight	119.1	119.1	119.1	119.1	119.1	119.1	110.4	110.4	110.4	110.4	110.4

Output: Proc Transpose Without Using Options

ADDING OPTIONS

There are a number of options that you can add to the statements in a Proc Transpose in order to achieve results that are clear and data that is ready for analysis. At first, it will seem like a great deal of trial and error to get your output dataset in the format that you need. The Proc Transpose statement has a number of options. Then there are other statements with options of their own that dictate how the data is to be transformed. Let take a look at a few of the most important ones.

PROC TRANSPOSE statement

The result of adding some options on the PROC statement is to change the output dataset a bit. These options give you the ability to use more meaningful names for your variables. You can control how your output dataset is defined with the extra options on the Proc Transpose statement. There are several options that can be added here. The options that we are going to discuss here are the LABEL, NAME and PREFIX options.

Below is a sample of code that we are going to use in our discussion. It is a simple Proc Transpose statement. We are going to refer to the data in the vital signs data set.

```
proc transpose data=vitals
              label=DESC
              name=VAR
              prefix=VAL;
run;
```

Remember, you can always use the data set options the DATA= and OUT= options. These tell you what data to use and what the name of the output dataset will be. If you omit these options, the default dataset that is used will be the last one that is created. The default output dataset name will be DATA1. As noted above, the default variables (_NAME_, _LABEL_ and the COL1, COL2, COL3, etc.) are a part of the output dataset. Of course, you can drop any variables that you don't want with a simple DROP option or keep specific variables by using the KEEP option on the DATA= option or you can rename them to something that is more meaningful (the RENAME=(oldname=newname) works as well). Remember that you cannot use both the DROP and KEEP option in the same step. In the example below, the _NAME_ is renamed to VAR and the _LABEL_ is renamed to DESC. The last option on the Proc Transpose statement below is called PREFIX. This option specifies a prefix to use in constructing the names of the variables in the output dataset. So, we are going to change COL1, COL2, COL3, COL4 to VAL1, VAL2, VAL3, VAL4.

VAR	DESC	VAL1	VAL2	VAL3	VAL4	VAL5	VAL6	VAL7	VAL8	VAL9	VAL10	VAL11
VISIT	Visit Number	1	2	3	4	9	10	11	12	13	14	15
VSDT	Date of Vital Signs	16076	16083	16086	16104	16195	16201	16223	16251	16279	16306	16342
SBP	Systolic BP	122	118	120	114	104	118	113	122	110	124	116
DBP	Diastolic BP	80	82	72	82	70	76	81	76	76	82	76
PULSE	Pulse Rate	65	60	66	60	60	66	62	66	72	60	66
WEIGHT	Weight	119.1	119.1	119.1	119.1	119.1	119.1	119.1	119.1	119.1	119.1	119.1

Output: Proc Transpose statement Using Options

Again, remember that this is a subset of the original data. The output will still have 775 columns. We haven't yet worked with any options that will change the number of columns in the output dataset.

BY Statement

This example shows how we can use the BY statement to change how the data is processed. Suppose that we want to have a record for each subject for each vital sign. How would we accomplish this? We introduce the BY statement into the code. As with any SAS procedure that uses a BY statement, the data must be sorted before the procedure is invoked. The BY statement that we will use can be seen in the sample code below. Proc Transpose does not transpose the BY group. So, the end result is exactly what we want. Now the data is beginning to look more believable and be easier to analyze.

```
proc transpose data=vitals
    label=DESC
              name=VAR
              prefix=VAL;
     by protocol inv pat; run;
```

Notice in the data below, there are missing values for the VAL6 and VAL7 columns. This is because this patient only had 5 visits. As we have seen with any horizontal data structure, there can be missing data from some cells. However, this makes it easier to perform an analysis of the data. For example, it we want to calculate the minimum and maximum diastolic blood pressure, it can be done very easily with a function or a line of code.

PROTOCOL	INV	PAT	VAR	DESC	VAL1	VAL2	VAL3	VAL4	VAL5	VAL6	VAL7
drg2005	227	27001	VISIT	Visit Number	1	2	3	4	9	10	11
drg2005	227	27001	VSDT	Date of Vital Signs	16076	16083	16086	16104	16195	16201	16223
drg2005	227	27001	SBP	Systolic BP	122	118	120	114	104	118	113
drg2005	227	27001	DBP	Diastolic BP	80	82	72	82	70	76	81
drg2005	227	27001	PULSE	Pulse Rate	65	60	66	60	60	66	62
drg2005	227	27001	WEIGHT	Weight	119.1	119.1	119.1	119.1	119.1	119.1	119.1
drg2005	227	27002	VISIT	Visit Number	1	2	3	4	9	.	.
drg2005	227	27002	VSDT	Date of Vital Signs	16096	16103	16106	16120	16166	.	.
drg2005	227	27002	SBP	Systolic BP	136	136	136	138	112	.	.
drg2005	227	27002	DBP	Diastolic BP	86	80	84	76	80	.	.
drg2005	227	27002	PULSE	Pulse Rate	72	84	84	82	72	.	.
drg2005	227	27002	WEIGHT	Weight	110.4	110.4	110.4	110.4	110.4		

Output: Use of the BY statement

VAR Statement

The VAR statement is the one that is needed to list the variables that need to be transposed. As we noted previously, the default for Proc Transpose is to transpose only numeric variables. The VAR statement is needed to transpose character variables. So, if you have character variables that you want to be transposed, you must list them here.

```
proc transpose data=student
               out=s1
               prefix=info;
    by name;
    var class grade credit;
run;
```

If we take a look at the student data, the transpose statement specifies that we are create an output dataset called S1 and all of the variables that are created will have a prefix of INFO. The new statement that we have added is the VAR statement. In the VAR statement, we are stating that we want CALSS, GRADE and CREDIT to be transposed. If you look at the data, you will see that the CLASS variable (which is the name of the class in which the student was enrolled) and the GRADE variable (which is the letter grade that the student received as a score) are character variables. The credit variable is a numeric variable that represents the number of credits for a course. With these three variables, we have all the information that we need to calculate a grade point average for a student once we transpose the data. If we didn't use the VAR statement, the only variable that would be transposed would be the variable CREDIT because it is the only numeric variable. Remember, the default is to only transpose the numeric data unless we use the VAR statement.

Notice in the output below, there is a mixture of character and numeric variables in the new variables INFO1 – INFO6. The interesting thing to notice is the way that SAS justifies the data within the variable. The numeric variables are right justified and the character variables are left justified. Remember, however that because there is a mixture of data types, SAS will create the variables as a character field. The fact that some data is right justified and some data is left justified could make the analysis difficult. We can overcome this problem by using the LEFT function whenever we want to reference the value of the variable.

Name	_NAME_	INFO1	INFO2	INFO3	INFO4	INFO5	INFO6
Ann Adams	Class	CHEM101	MATH110	LIT100	PHY101	FREN102	CHEMLAB
Ann Adams	Grade	A	A	B	A	B	B
Ann Adams	Credit	4	4	4	4	4	2
Bob Benz	Class	CHEM101	BIO110	PSY101	ANT100	CHEMLAB	
Bob Benz	Grade	B	C	B	A	B	
Bob Benz	Credit	4	4	4	4	2	
Carl Jones	Class	SPAN101					
Carl Jones	Grade	A					
Carl Jones	Credit	4					
Dave Fine	Class	BIO200	MATH210	LIT200			
Dave Fine	Grade	C	C	C			
Dave Fine	Credit	4	4	4			
Edna James	Class	ART220					
Edna James	Grade	A					
Edna James	Credit	4					
Fred Fame	Class	CHEM101	MATH110	ANT111	CHEMLAB		
Fred Fame	Grade	B	A	C	B		
Fred Fame	Credit	4	4	4	2		
Greg Gain	Class	PSY211	ITAL101				
Greg Gain	Grade	A	A				
Greg Gain	Credit	4	4				
Hanna Hand	Class	CHEM212	BIO203	PSY101	HIST325	SPAN202	
Hanna Hand	Grade	B	A	B	B	A	
Hanna Hand	Credit	4	4	4	4	4	

Output: Use of the VAR statement

ID Statement
Suppose you want to use the value of a variable as the variable name in the output dataset. Using the ID statement is the method to accomplish this. The ID statement specifies a variable in the input dataset whose value is used to name a variable in the output dataset. If I had data for each season and wanted the variables to be named WINTER, SPRING, SUMMER and FALL, I would use the variable SEASON in the ID statement of the Proc Transpose to accomplish this.

```
proc transpose data=student
  by name;
  var grade;
  id class;
run;
```

This code above shows us how we can use the ID statement with our student dataset to create variables for each class. The values of the variables are used to name the columns. In this example, the classes that have been taken by the students are the variable and the grade that they each received are the values of the variables. I can calculate the grade point average of each student very easily.

So far, we have talked about the values of a character variable. What happens if the variable that you want to have as the name of the transposed variable is a numeric value? SAS will change the name to a valid SAS name by affixing an underscore as the first character. If there are any missing values in your data, SAS will not transpose the data and issue a warning message in the log. If the value in the data is longer than 32 characters, SAS will truncate the value to 32 characters. If you have the system option

VALIDVARNAME set to V6 where the variable names can not be any longer than 8 characters, SAS will truncate the value to 8 characters.

If you have duplicate values in the input dataset or in a BY group, Proc Transpose will issue a warning message in the log. It will stop processing and you will not get the results that you expect in the output dataset.

Name		CHEM101	MATH110	LIT100	PHY101	FREN100	LAB	SPAN200
Ann Adams	Grade	A	A	B	A	B	B	
Bob Benz	Grade	B	A	A	C	B	B	C
Carl Jones	Grade	B		A	C	C	A	
Dave Fine	Grade	B	A	B	A	A	A	B
Edna James	Grade		B	B	A	B		B
Fred Fame	Grade	B	A				B	A
Greg Gain	Grade	C		C	B	D	B	C
Hanna Hand	Grade		C	A	B	A		A

Output: Use of the ID statement

CONCLUSION

We have explored some of the features and options that can be used in a Proc Transpose. From its simplest invocation without any options to some of the most used ones, this procedure can be very powerful and helpful when analyzing data. With a minimal amount of code, the data can be manipulated so that the analysis is simple and easy. Because the procedure creates an output dataset and not a report, you can use any type of printing procedure to report on the output dataset. This is a very powerful tool in the base SAS product.

REFERENCES

SAS Procedures Guide, Version 8, SAS Publishing, 1999
Cody, Ron, *Longitudinal Data and SAS*, SAS Publishing, 2001

ACKNOWLEDGMENTS

I would like to thank my husband Bob for his unending support and tireless proof-reading comments. Without his encouragement, I could not have completed my paper.

AUTHOR CONTACT

Janet Stuelpner
Left Hand Computing, Inc
326 Old Norwalk Road
New Canaan, CT 06840
(203) 966-7520 voice
(203) 966-8027 fax
jstuelpner@usa.net