

# Proc Transpose: Making Data Easier to Process

Jimmy DeFoor, Brierley & Partners, Dallas

## Why Use Transpose

Data, regardless of how it is stored, is processed in either of two structures: horizontal or vertical. The horizontal form expresses the attributes as fields or columns and stores the values within those entities. As a result, there is no need to repeat the primary identifiers. An example would be:

| Name           | Lot Number | Color | Texture | Manufacturer | Price   |
|----------------|------------|-------|---------|--------------|---------|
| Blue Jean      | 1010001    | Blue  | Rough   | Old Tents    | \$15.00 |
| Athletic Socks | 102000     | White | Ribbed  | Smelly Cat   | \$3.00  |
| Towels         | 1030001    | Red   | Rough   | Floor Covers | \$10.00 |

The horizontal form saves space and is also more efficient for selecting all data which have a certain attribute value, such as the color blue or the manufacturer Smelly Cat. The problem with the horizontal form, however, is that more coding is required to analyze it using the various SAS Procedures. Hence, the value of Proc Transpose. It allows data to be stored in horizontal form and then processed in the vertical form.

This is the basic Proc Transpose that would transform the horizontal form to the vertical form:

```
Proc Transpose Data = Horizontal
      Out = Vertical(Rename=(_Name_=Trait Coll=Value);
  By Name LotNumber;
  Var Color Texture Manufacturer Price;
Run;
```

Notice the renaming of the automatic variables *\_Name\_* and *\_Col1\_* to *Trait* and *Value*. The *Trait* field contains the variable names (now attributes) of the horizontal data. The *Value* field contains the values of each of those variables.

| Name      | Lot Number | Item         | Value     |
|-----------|------------|--------------|-----------|
| Blue Jean | 1010001    | Color        | Blue      |
| Blue Jean | 1010001    | Texture      | Rough     |
| Blue Jean | 1010001    | Manufacturer | Old Tents |
| Blue Jean | 1010001    | Price        | \$15.00   |

Going from the vertical to horizontal would also be done with Proc Transpose. However, an additional statement would be required, one that specifies the input variable that contains the attributes that will become the new variables in the output data set. That statement is the ID statement.

```
Proc Transpose Data = Vertical
      /* Drop _Name_ because it contains only the string Trait */
      /* Coll doesn't occur because of the ID Variable */
      Out = Horizontal(Drop=_Name_);
  By Name LotNumber;
  Var Answer;
  Id Item; /* Provides names of new variables */
Run;
```

The automatic variable `_Name_` was dropped in the transpose of vertical to horizontal because all observations will only contain the string *Trait* since that was the name of the transposed ID variable. This is shown below:

| Name           | Lot Number | _Name_ | Color | Texture | Manufacturer | Price   |
|----------------|------------|--------|-------|---------|--------------|---------|
| Blue Jean      | 1010001    | Trait  | Blue  | Rough   | Old Tents    | \$15.00 |
| Athletic Socks | 102000     | Trait  | White | Ribbed  | Smelly Cat   | \$3.00  |
| Towels         | 1030001    | Trait  | Red   | Rough   | Floor Covers | \$10.00 |

## Details of Transpose

```
Proc Transpose Data = Horizontal
      Out = Vertical(Rename=(_Name_=Trait Coll=Value);
  By Name LotNumber;
  Var Color Texture Manufacturer Price;
Run;
```

## Transpose Statement

The Transpose statement defines the input and output data sets, controls the renaming or dropping of the automatic variables, and allows the use of options such as `Let`, which will be discussed in the `ID` statement.

## Var Statement

The `Var` statement specifies the variables that will be transposed. If a `Var` statement is not present, only the numeric variables will be transposed. Character variables are only be transposed if they are listed in the `Var` statement. To transpose all character variables, use the `Var _Character_` statement. To transpose all variables, use the `Var _All_` statement.

## \_Name\_ Variable

The `_Name_` variable contains the names of the variables that are being transposed. For the above example, it would contain `Color`, `Texture`, `Manufacturer`, and `Price`, hence the reason for renaming the field `Trait`.

## Col Variables

The `Col` variables contain the values of the variables specified in the `Var` Statement. The number created by `Transpose` is dependent upon the number of observations associated with the variables in the `Var` statement - for each `By` group. For this code, which is from our first example, there is one observation per `By` group. If there had been two observations for each `By` group, perhaps for an earlier price of that combination of `Color`, `Manufacturer`, and `Texture`, there would have been a second `Col` variable. The appropriate renaming then would be `Col1=Value1`, `Col2=Value2`. This is shown in the next example:

### *Two Observations per By group*

| Name      | Lot Number | Color | Texture | Manufacturer | Price   |
|-----------|------------|-------|---------|--------------|---------|
| Blue Jean | 1010001    | Blue  | Rough   | Old Tents    | \$15.00 |
| Blue Jean | 1010001    | Blue  | Rough   | Old Tents    | \$10.00 |

*Resulting Transpose has two Col variables.*

| Name      | Lot Number | Item         | Col1      | Col2      |
|-----------|------------|--------------|-----------|-----------|
| Blue Jean | 1010001    | Color        | Blue      | Blue      |
| Blue Jean | 1010001    | Texture      | Rough     | Rough     |
| Blue Jean | 1010001    | Manufacturer | Old Tents | Old Tents |
| Blue Jean | 1010001    | Price        | \$15.00   | \$10.00   |

Col variables occur when there is not an ID variable to provide the output names for the transposed data.

### **By Statement**

By Name LotNumber;

The By statement determines the variables which will not be transposed. In general, it will contain the variables that either will be or were the unique groupings of the transposed data.

*Note: Do not put the same variable in the By statement that you put in the ID statement. Your data will be unintelligible and you will get no warning that you did it.*

### **Id Statement**

ID Trait;

The ID statement replaces the Col variables. It creates the names of the output variables from the values of the variable in the ID statement. If those values were Color, Texture, Manufacturer, and Price, those values would be the names of the new transposed variables. However, if the variable had a format associated with it, the names will be generated from it. For example, suppose the ID variable was Month, that it had values of 1 to 12, and that it was associated with the Month format, which had 01 = Jan, 02 = Feb, etc. Then the output variables would be Jan through Dec.

Id Variables with unformatted numeric values will create output variables with a beginning underscore, such as \_1998, unless the Prefix option is used on the Proc Transpose statement. Then, that prefix will be attached to each numeric value. For example, Prefix=YR would create output variables YR1998, YR1998, YR2000 instead of \_1998, \_1999, \_2000.

The key issue for the ID statement are duplicate values in the ID variable for the variables in the By group. If they exist, Proc Transpose will issue a warning message and stop. Using the LET option on the Proc Transpose statement will prevent this. Then, Transpose will issue a warning message and output only the last occurrence of the duplicate Id value.

The LET option should not be used casually. Duplicate Id values are usually an indication of problem and then termination is the best choice. For example, the occurrence of multiple SumYrAmts for the video category of an account in 1998 would create duplicate Id values and should. No customer could have two yearly totals for video rentals for the same account. But is the first sum the correct one, the second, or should the two rows be summed? Only an examination of the data and previous processing can determine the correct solution.

| Account | YrField | SumYrAmt |
|---------|---------|----------|
| 1101110 | Bev98Yr | 50.55    |
| 1101110 | Can98Yr | 75.30    |
| 1101110 | Vid98Yr | 200.30   |
| 1101110 | Vid98Yr | 35.30    |

The data on the records of Id Variables with missing values will be dropped from the output data.

### ***Guidelines for Using Proc Transpose***

These guidelines are based upon my uses of Transpose, which certainly may not be comprehensive.

- ❖ Decide which variables should to be transposed and put those in the Var statement.
- ❖ Decide which variables should not be transposed and put those in the By statement. If the data is not sorted or grouped by the variables in the By statement, precede the transpose with a sort.
- ❖ Does the Var statement have multiple variables or only one?
  - Multiple: Then an ID statement is probably not appropriate. Rename the generated Col variables.
  - One: Then an ID statement is probably appropriate. Use the field that contains the names of the transposed variables. Examples are classification variables such as Category, Item, and Type.
- ❖ If a sort is required for the By statement, add the Id variable as the last variable in the Sort's By statement. Do not put the ID variable in the Transpose's By statement. It creates bad output.
- ❖ Col variables will be generated if an ID field is not used. One will be generated for each observation associated with the By statement. Rename the Col variables to reflect the meaning of the values that were in the fields in the Var statement. Example: Value, Answer, and Amount. If more than one Col variable is generated, they should probably have sequential names such as Amt1 - Amt10.
- ❖ A \_Name\_ variable will always be generated. If the Transpose has an ID statement, the \_Name\_ variable can probably be dropped. Otherwise, the \_Name\_ variable should be renamed to reflect the meaning of the names of the fields in the Var statement. Example: Item, Questn, and Category.

## **Structure of Data Output from Proc Transpose**

This information is directly from the SAS Online Help Documentation. I can't say it any better.

### **Attributes of Transposed Variables**

- All transposed variables are the same type and length.
- If all variables that the procedure is transposing are numeric, the transposed variables are numeric. Thus, if the numeric variable has a character string as a formatted value, its unformatted numeric value is transposed.
- If any variable that the procedure is transposing is character, all transposed variables are character. Thus, if you are transposing a numeric variable that has a character string as a formatted value, the formatted value is transposed.
- The length of the transposed variables is equal to the length of the longest variable being transposed.

### **Names of Transposed Variables**

PROC TRANSPOSE names transposed variables using the following rules:

1. An ID statement specifies a variable in the input data set whose formatted values become names for the transposed variables.
2. The PREFIX= option specifies a prefix to use in constructing the names of transposed variables.
3. If you do not use an ID statement or the PREFIX= option, PROC TRANSPOSE looks for an input variable called `_NAME_` from which to get the names of the transposed variables. (*It uses `_Name_` as an ID field.*)
4. If you do not use an ID statement or the PREFIX= option, and the input data set does not contain a variable named `_NAME_`, PROC TRANSPOSE assigns the names COL1, COL2, . . . , COLn to the transposed variables.

## Shortcomings of Transpose

Transpose, like Tabulate, formulates its output structure based upon the order and content of the data transposed in the first group of the variables of the BY statement. Thus, if the By statement is for Account and the ID is Year, the data will be transposed according to the order and content of the values of Year within the first account in the data. For example, if the first account has the Years 1998, 1998, and 2000, the output variables will be \_1998, \_1999, and \_2000.

The problem comes when the second account number is encountered. It has years 1997 and 1996 in addition to 1998 - 2000. Because of that, Transpose adds \_1996 and \_1997 to the output data vector after 1998 - 2000. This creates havoc with the desire to look at the results in context from within SAS Explorer, or by saving the data as an HTML or CSV file, or through the Output window using Proc Print.

Several solutions exist, but all automated methods require using macros and pre-processing the data to determine the unique entries of the ID variable. The simplest automated method passes the transposed data through a subsequent data step in which the transposed variables are reordered using a Length statement that is placed before the Set statement referencing the transposed data set.

In the non-automated way, the transposed data set would be visually examined and then a data set created with appropriate length statements. For example, say the transformed data has this form:

| Account  | _1998 | _1999 | _2000 | _1996 | _1997 |
|----------|-------|-------|-------|-------|-------|
| 10100185 | 2000  | 3000  | 4000  | 0     | 1000  |

A subsequent step would then be written to reorder the fields:

```
Data Restructured;
  Length Account 8
         _1996 8
         _1997 8
         _1998 8
         _1999 8
         _2000 8
  ;
  Set Transformed;
  Output;
Run;
```

In the automated way, the code would be put in a macro and the data would be pre-processed before transposition;

```
%Macro Restructure(Form=Char);
/* The data is sorted by the By variable and ID variable */
Proc sort data = work1(Keep=Year Value);
    Out = work2
    by Year Value;
Run;
*;
/* The maximum length of the ID field is determined. */
/* It will be used to set the length of the transposed fields */
Data _null_;
    Set work2 end=eof;
    By Year;
    %if &Form eq Char %then
        %do;
            Retain maxlength 0;
            if length(Value) gt Maxlength then
                maxlength = length(Value);
        %end;
    %else
        %do;
            Retain maxlength 8;
        %end;
    /* The macro variables Var1 through VarN are created. */
    /* Into them are loaded the values of the transposed vars. */
    if first.Year then
        do;
            Count + 1;
            Var='Var' || Left(Put(Count,3.0));
            Col='_' || Year;
            Call Symput(Var,Col);
            output;
        end;
    /* The number of transposed vars and their maximum length */
    /* loaded into macro variables. */
    if eof then
        do;
            Call Symput('TotVars',Put(Count,3.0));
            Call Symput('length',Put(Maxlength,3.0));
        end;
run;
*;
/* Some of the macro vars are displayed as an audit. */
%put &var1 &var5 &var9;
%put &length &totvars;
;
Proc Transpose data = VertYears
    out = HortYears(Drop=_Name_);
    By Account;
    Id Year;
run;
*;
```

```

Data Restructured;
  /* Generate Length statement with transposed variables */
  /* ordered correctly */
  Length Account $10.0
  %do K = 1 %to &TotVars;
    &&Var&K $&&length..0
  %end;
  ;
  Set HortYears;
  output;
Run;
%Mend Restructure;

```

## ***Reasons for Using Transpose***

### **Working with Survey Results**

Proc Transpose can be very useful with survey results. Imagine that we have horizontal data representing answers to questions on service ratings and number of roundtrip flights.

| Account  | SurveyType    | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 |
|----------|---------------|----|----|----|----|----|----|----|----|
| 44444444 | Domestic      | 3  | 5  | 7  | 6  | 12 | 9  | 35 | 20 |
| 55555555 | Domestic      | 6  | 7  | 4  | 5  | 18 | 5  | 20 | 30 |
| 66666666 | International | 4  | 5  | 6  | 2  | 16 | 4  | 18 | 10 |

This code would transpose the data:

```

Proc Transpose Data = HortQuestn
  Out = VertQuestn(Rename=(_Name_=Questn Coll=Answer);
  By Account SurveyType;
  Var Q1 - Q8;
Run;

```

This would be its transposed form:

| Account  | SurveyType | Questn | Answer |
|----------|------------|--------|--------|
| 44444444 | Domestic   | Q1     | 3      |
| 44444444 | Domestic   | Q2     | 5      |
| 44444444 | Domestic   | Q3     | 7      |
| 44444444 | Domestic   | Q4     | 6      |
| 44444444 | Domestic   | Q5     | 12     |
| 44444444 | Domestic   | Q6     | 9      |
| 44444444 | Domestic   | Q5     | 35     |
| 44444444 | Domestic   | Q6     | 20     |

The vertical form of the data requires less coding to create and print the statistical moments associated with the answers to the questions. Look at this Proc Univariate and Proc Print code used to analyze and report the results for just three questions. Compare the vertical form to the horizontal.

|                        |                                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                  |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Vertical Form</b>   | <pre>Proc Univariate Data = VertQuestn noprint;   By SurveyType Questn;   Var Answers;   Output Out = VertStats     Mean = Avg P10 = P10 Q1 = P25     Median = P50 P90 = P90 Q3 = P75     N = Cnt ; Run;</pre>                                                                                                                                    | <pre>Proc Print Data = VertStats;   By SurveyType Questn;   Id N;   Var Avg P10 P25     P50 P75 P90   ; Run;</pre>                                                                                                                                                               |
| <b>Horizontal Form</b> | <pre>Proc Univariate Data = HortQuestn noprint;   By SurveyType;   Var Q1 Q2 Q3;   Output Out = HortStats     Mean = Q1_Avg Q2_Avg Q3_Avg     P10 = Q1_P10 Q2_P10 Q3_P10     P25 = Q1_P25 Q2_P25 Q3_P25     P75 = Q1_P75 Q2_P75 Q3_P75     P90 = Q1_P90 Q2_P90 Q3_P90     Median = Q1_P50 Q2_P50 Q3_P50     N = Q1_Cnt Q2_Cnt Q3_Cnt ; Run;</pre> | <pre>Proc Print Data = HortStats;   By SurveyType; /* Q1 only */   Id Q1_Cnt;   Var Q1_Avg Q1_P10 Q1_P25     Q1_P50 Q1_P75 Q1_P90; Run;  Proc Print Data = HortStats;   By SurveyType; /* Q2 only */   Id Q2_Cnt;   Var Q2_Avg Q2_P10 Q2_P25     Q2_P50 Q2_P75 Q2_P90 Run;</pre> |

Obviously, more coding is required to code the horizontal form of the data than the vertical. This is because the variables Q1, Q2, and Q3 are attributes of the field Questn in the vertical form, which allows the By statement to be used to control processing. The horizontal form requires that the variables be named in the Univariate and Print steps and also forces the need for multiple Print steps.

Good SAS macro programmers will attempt to resolve many of these issues by placing the Univariate and the Print procs into a macro that includes a where clause for the question being analyzed and reported. They would create this code:

```
%Macro Report (Questn=,StatData= );

Proc Univariate Data = HortQuestn (Where=(Questn=&Questn)) noprint;
By SurveyType;
  Var &Questn;
  Output Out = &StatData
    Mean = &Questn_Avg
    P10 = &Questn_P10
    P25 = &Questn_P25
    P75 = &Questn_P75
    P90 = &Questn_P90
    Median = &Questn_P50
    N = &Questn_Cnt ;
Run;
*;
Proc Print Data = &StatData;
  By SurveyType;
  Id &Questn_Cnt;
  Var &Questn_Avg &Questn_P10 &Questn_P25
    &Questn_P50 &Questn_P75 &Questn_P90;
Run;
*;
%Mend Report;
```

This code works very well, particularly if only a few questions are being reported. However, if many questions are being processed, the source data will have to read multiple times. Also, if the survey was created so that responses to subsequent questions are related to previous answers, the output data sets will have to be merged in order to easily compare the relationship of the responses. Proc Transpose avoids these issues.

For example, say the responses to Question 5 and Question 6 were to be examined based upon a '2' answer to question 3. Proc Transpose and the horizontal format of the data make this quite easy.

```
Proc Transpose Data = HortQuestn
  Out = VertQuestn(Rename=(_Name_=Questn Coll=Answer
    Where=(Questn=2));

  By Account SurveyType;
  Var Q5 Q6;
Run;
```

Proc Univariate and Proc Print would then process the transposed data using By Questn statements, as shown in the Vertical Format table on the previous page.

## Summing Categories of Values

Transpose can also facilitate working with the summing of categories of sales data. Imagine that sales data by account was stored in this horizontal structure.

| Account | Bev9801 - Bev9812 | Can9801 - Can9812 | Vid9801 - Vid9812 |
|---------|-------------------|-------------------|-------------------|
|---------|-------------------|-------------------|-------------------|

Summing these into yearly buckets without a transpose would require this type of code.

```
Data SumYR98;  
  Keep Account BevYr98 CanYr98 VidYr98;  
  Set  Month98;  
  BevYR98 = Sum(of Bev9801 - Bev9812);  
  CanYR98 = Sum(of Can9801 - Can9812);  
  VidYR98 = Sum(of Vid9801 - Vid9812);  
  /* Or, using the prefix list method, Sum(of BevYR:) */  
  Output;  
Run;
```

Simple and sparse code, right? But what if the data was already being transposed so that the monthly data could be analyzed more easily? And what if there were separate data sets for each of 10 years and 50 such 'types' of fields in each data set. Then, the yearly analysis process would require 50 sum statements in the program and 10 such programs (unless a macro with year substitution was used) in addition to the transpose of the monthly data.

Instead, once transposed, a single program could be used to summarize every year of data:

```

/* The Transpose renamed _Name_ to Category and _Coll_ to Amount. */
Data Work1;
  Drop YrValue;
  Length YrValue $2.0;
  Retain YrValue ' ';
  Set Transposed end=eof;
  If _N_ eq 1 then
    do; /* Creates a field containing the year value */
      YrValue = Substr(Category,4,2);
      /* Places the Year value in a macro */
      Call Symput('Yr'||YrValue);
    end;
    /* Creates attributes for the yearly sums of each category */
    /* Examples are Bev98Yr, Vid98Yr, etc. */
    YrField = Substr(Category,1.3)||'YR'||YrValue;
  Output;
Run;
*;
/* A sort will only be necessary if the data is ungrouped */
Proc Sort Data = Work1;
  By Account YrField ;
Run;
*;
Data SumYr&YR;
  Keep Account YrField SumYRAmt;
  Set Work1;
  /* NotSorted allows the data to be only grouped instead of sorted */
  By Account YrField NotSorted;
  SumYrAmt + Amount;
  If last.YrField then
    do;
      output;
      SumYrAmt = 0;
    end;
Run;

```

The resulting data would be:

| Account | YrField | SumYrAmt |
|---------|---------|----------|
| 1101110 | Bev98Yr | 50.55    |
| 1101110 | Can98Yr | 75.30    |
| 1101110 | Vid98Yr | 200.30   |