Paper BB-59

PROC TRANSPOSE® For Fun And Profit

John J. Cohen, Advanced Data Concepts, LLC

Abstract

Occasionally we are called upon to transform data from one format into a "flipped," sort of mirror image. Namely if the data were organized in rows and columns, we need to transpose these same data to be arranged instead in columns and rows. A perfectly reasonable view of incoming lab data, ATM transactions, or web "click" streams may look "wrong" to us. Alternatively extracts from external databases and production systems may need massaging prior to proceeding in SAS®. Finally, certain SAS procedures may require a precise data structure, there may be particular requirements for data visualization and graphing (such as date or time being organized horizontally/along the row rather than values in a date/time variable), or the end user/customer may have specific deliverable requirements.

Traditionalists prefer using the DATA step and combinations of Array, Retain, and Output statements. This approach works well but for simple applications may require more effort than is necessary. For folks who intend to do much of the project work in, say, MS/Excel®, the resident transpose option when pasting data is a handy short cut. However, if we want a simple, reliable method in SAS which once understood will require little on-going validation with each new run, then PROC TRANSPOSE is a worthy candidate. We will step through a series of examples, elucidating some of the internal logic of this procedure and its options. We will also touch on some of the issues which cause folks to shy away and rely on other approaches.

Introduction

We will take a simple dataset and transpose it at the simplest level using a standard Proc Sort/Data step approach. We will demonstrate the same using Excel transpose. And finally we will use Proc Transpose to transform the same data. We will then explore Proc Transpose in increasing complexity as we test some of the many options. We will demonstrate certain errors and test strategies for overcoming these. When completed, we expect that you will also agree that Proc Transpose has value as another tool in your SAS tool kit.

The dataset we will use for our examples consist of a modified/simplified version of a standard sample dataset provided with most SAS installations, sashelp.retail. It will look substantially similar to Figure 1:

Figure 1 – sashelp.retail dataset

VIEWTABLE: Work.Retail

	Retail sales in millions of $	date
1	$998	1994-04-01
2	$991	1993-10-01
3	$920	1993-07-01
4	$909	1993-04-01
5	$889	1992-10-01
6	$876	1994-01-01
7	$826	1992-07-01
8	$807	1991-10-01
9	$797	1992-04-01
10	$758	1993-01-01

Simple Transpose Using SAS Data step

We will be using a DATA step to take in our dataset, restructure it, and output the same data, but in a different shape from our starting point. The first step in re-structuring the dataset is to sort the data into the desired order. PROC SORT will do the job nicely. And in this simple example we are sorting by date as in Figure 2. We will also capture the number of observation in the dataset using PROC SQL, outputting the result into a macro variable **&record_count**.

Figure 2 – Transpose Using DATA Step - Preparation

```
              /*** order dataset using PROC SORT ***/
proc sort data=retail; by date; run;
NOTE: The data set WORK.RETAIL has 58 observations and 2 variables.
                   /*** capture observation count ***/
proc sql noprint;
   select count(*) into: record_count trimmed
   from retail;
quit;
%put record count is &record_count.;
record count is 58
```

The results should look much like the data displayed in Figure 3, same data as in Figure 1, except now sorted. We now know that this dataset contains 58 observations, a fact which we have stored in the macro variable **&record_count.** It is also worth noting that there are two variables in this dataset, ***sales*** and ***date***.

Figure 3 – Transpose Using DATA Step – "Prepared" Dataset

VIEWTABLE: Work.Retail

	Retail sales in millions of $	date
1	$220	1980-01-01
2	$257	1980-04-01
3	$258	1980-07-01
4	$295	1980-10-01
5	$247	1981-01-01
6	$292	1981-04-01
7	$286	1981-07-01
8	$323	1981-10-01
9	$284	1982-01-01
10	$307	1982-04-01

With these tools in place, the DATA step will then allow us to complete the transformation using the program displayed in Figure 4. We will start with a dataset containing 2 variables and 58 observations and which is sorted by date. We will transform this into a dataset with 58 variables and 1 observation (or is it still 2 observations?).

Figure 4 – Transpose Using DATA Step – The Restructuring Step

```
                                                /*** Restructure dataset in DATA Step ***/
data transposed_retail;
   set retail end=eof;
   ***by date;                                                  /*** (just for show) ***/
   retain date1 - date&record_count. i 0;
   array dates {*} date1 - date&record_count.;
   i = i + 1;
   dates{i} = sales;
   ***if last.date then output;                                 /*** (just for show) ***/

   if eof then output;  /*** note only one record will be output- i.e., @eof ***/
   keep date1 - date&record_count.;
run;

                                                                    /*** partial log ***/
NOTE: There were 58 observations read from the data set WORK.RETAIL.
NOTE: The data set WORK.TRANSPOSED_RETAIL has 1 observations and 58 variables.
```

In some detail we will walk through how the "old school" DATA step will achieve our goal. (Our goal is not to understand this process in extreme detail – as our interest is in learning newer, simpler techniques. But understanding some of the basics of the logic will inform the following discussions.)

END = EOF

First note that we take the sorted dataset RETAIL and set by DATE (although readers with good eyes will notice that the BY statement is, in fact, commented out). In more complex constructs – say with multiple observations per value of DATE, we may end up processing across these observations (say, filtering out certain observations, finding the maximum or minimum values, and/or summing across these) and then wanting to output one observation for each value of date. Likely then we would also be employing FIRST.DATE and/or LAST.DATE. For our purposes here, a simpler approach is appropriate. On the SET statement we include an END= option to create a logical variable named EOF (for End Of File). If an incoming observation is NOT the last observation in the file, the value of is set to "0". In contrast, the value of EOF for the last observation in the dataset RETAIL will be set to "1". Thus, when the last observation is processed and SAS reaches the IF statement – IF EOF – the OUTPUT instruction will be executed – **only for this last observation**.

Macro Variable &RECORD_COUNT.

Next note the use of the macro variable **&RECORD_COUNT** in three locations, on the RETAIN, the ARRAY, and the KEEP statements. In all three instances we are defining a range of variables named DATE1, DATE2, DATE3, through DATE&RECORD_COUNT., which for these data will resolve to DATE%* (for the 58 observation in the incoming dataset RETAIL).

Additional Details

The **RETAIN** statement means that values we assign to the variables in the list DATE1 – DATE58 (remember, this is what the macro variable &RECORD_COUNT will resolve to) will *retain* those values across observations. (Without the RETAIN, each variable's value will be re-initialized as the next observation is read in.

The **ARRAY** statement initializes a list of variables which we can then refer to using simply the array name of DATES.

The **KEEP** statement controls the variable list which will appear on the out-going dataset TRANSPOSED_RETAIL. We start with 2 variables, DATE and SALES and 58 observations. We end with 58 variables, DATE1 – DATE58 and one observation.

Program Logic

We initialize our counter variable **i** to zero, so we increment **i** by 1 as we read in the first observation. We assign a value to the **i**-th element of array DATES the corresponding value of SALES. The first element (**i**=1) of variable array DATES will be DATE1. To that we assign the value of 220. It is not the end-of-file/last observation, so we do not execute the output statement.

Instead we return to the top of the DATA step and read in observation #2. We are retaining the value of DATE1, and now we increment **i** by 1, meaning that i=2. The array DATES{i} will now refer to DATES{2}, or DATE2, the second element of the array. We assign to DATE2 the corresponding value of SALES for the second observation, or 257.

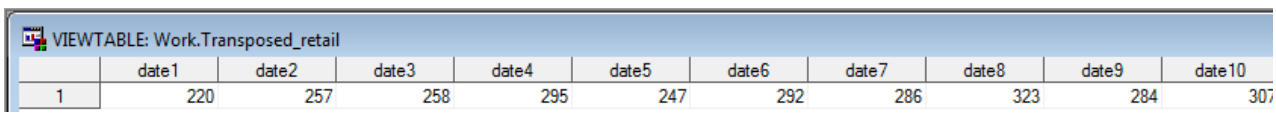
In a similar fashion we continue to read in each successive observation, increment the counter variable **i**, find the value of SALES for the **i**-th observation, and assign that value to the appropriate element of the array DATES. When we have finished reading in all 58 observations of dataset RETAIL we will have fully populated the values of every element of array DATES (thanks to the RETAIN statement) and when the end-of-file flag EOF is finally triggered, now we output a single observation. The KEEP statement confirms that we are keeping the 58 new variables from our array (DATE1 – DATE58) and discarding the original variables DATE and SALES from the incoming dataset RETAIL. A partial view of the results is displayed in Figure 5.

Discussion

We may note the following: we started with two perfectly fine variables, DATE and SALES. We finish with a new dataset which faithfully retains all the values of SALES, now transposed from multiple observations within a single variable (or from rows to columns). But the other variable DATE, aside from having the correct order being reflected in the variable (or column) names as DATE1 through DATE58, has any additional precision regarding the original values of date being lost. (Note that even though we did not set BY DATE, had we not pre-sorted dataset RETAIL by DATE, the order of the variables/columns would have been incorrect.)

There are ways to capture and carry forward more precise information from the incoming variable DATE. This information can then be transformed into the new variable names (or at least variable labels) in the outgoing dataset. (For instance, see Cohen, “Hands Free: Automating Variable Name Re-Naming Prior to Export” for one approach.) However, these take an already not-so-simple approach and add further complexity. Our goal is to explore easier solutions. We will do so looking first at the Transpose function in Excel.

Figure 5 – SAS Dataset Transposed Using SAS Data Step


VIEWTABLE: Work.Transposed_retail

	date1	date2	date3	date4	date5	date6	date7	date8	date9	date10
1	220	257	258	295	247	292	286	323	284	307

Simple Transpose Using Excel

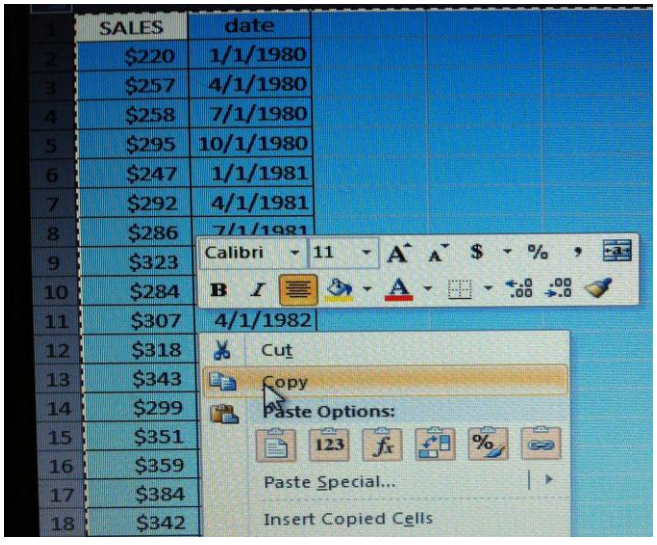
With Excel we start with the same dataset. How you get your data into Excel (e.g., Proc Export, writing out a flat file or comma-separated file and then importing within Excel, ODS to XML, data entry) is not a part of this discussion. While Excel may employ different formatting from SAS in its display of these data, Figure 6 shows the same dataset as we saw in Figure 3. As before, we have sorted these data by DATE.

Figure 6 - Simple Transpose – Initial Excel Spreadsheet

	A	B
1	Sales	Date
2	$220	1/1/1980
3	$257	4/1/1980
4	$258	7/1/1980
5	$295	10/1/1980
6	$247	1/1/1981
7	$292	4/1/1981
8	$286	7/1/1981
9	$323	10/1/1981
10	$284	1/1/1982
11	$307	4/1/1982

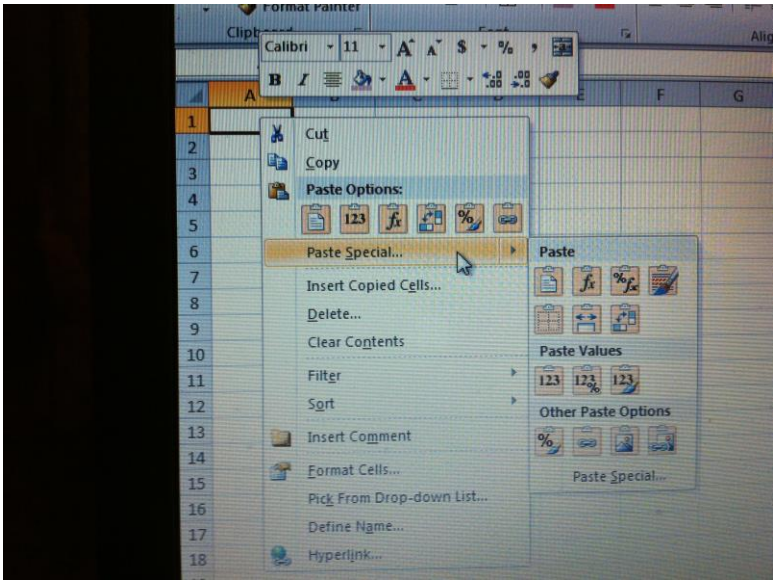
In Figure 7 we highlight the cells we want, then right click to bring up the edit menu. As in the example depicted, we want to click on the COPY function. Then we locate our cursor in the target cell location.

Figure 7 - Simple Transpose – Excel Copy Process



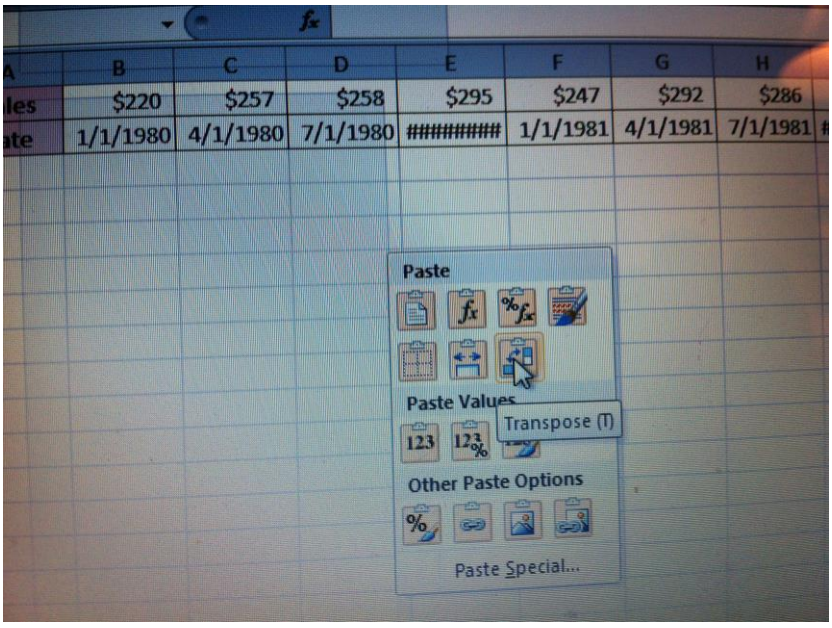
Next we right-click again and as depicted in Figure 8 we drop our cursor down to the “Paste Special” line in the drop-down menu. We click on the “Paste Special”, which in turn brings up a “Paste” sub-menu.

Figure 8 - Simple Transpose – Excel "Paste Special"



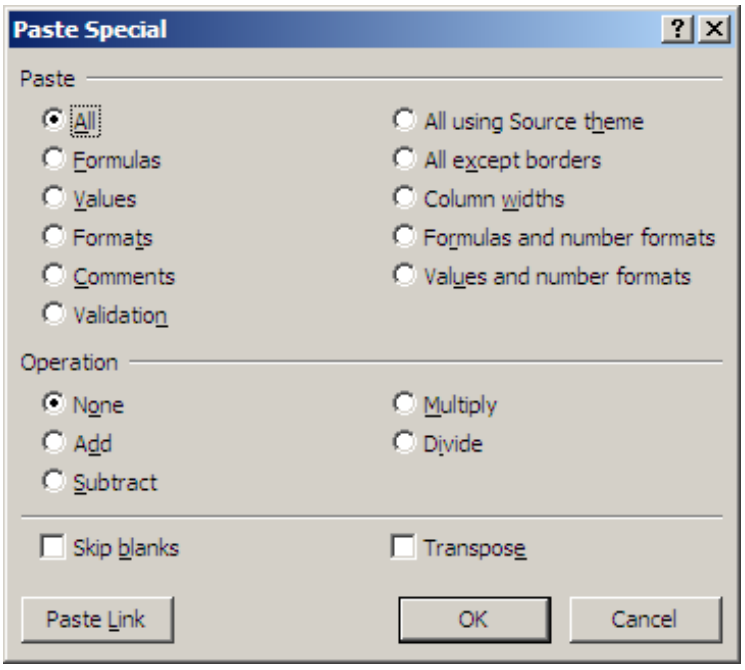
Within the "Paste" sub-menu displayed in Figure 9 is a "Transpose" option – as will be presented in Excel 2010.

Figure 9 - Simple Transpose – Excel "Transpose" (Excel 2010)



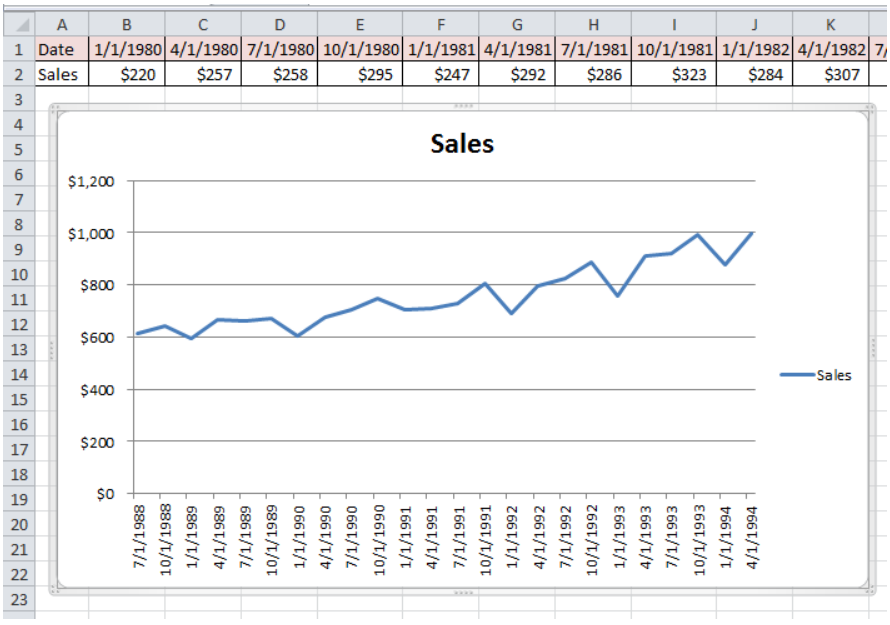
In Figure 10 we see the "Transpose" option as will be presented under Excel 97-2003 as a check box at the bottom right.

Figure 10 - Simple Transpose – Excel "Transpose" (Excel 97-2003)



With a click on the "Transpose" choice under Excel 2010 or selecting the "Transpose" check box, then "OK" under Excel 97-2003, the result will be something like the new table displayed in Figure 11.

Figure 11 - Simple Transpose – Excel – Transposed Table (and Resulting Graph)



	A	B	C	D	E	F	G	H	I	J	K
1	Date	1/1/1980	4/1/1980	7/1/1980	10/1/1980	1/1/1981	4/1/1981	7/1/1981	10/1/1981	1/1/1982	4/1/1982
2	Sales	$220	$257	$258	$295	$247	$292	$286	$323	$284	$307

Simple Transpose Using SAS Proc Transpose

We start with the same retail dataset as in the prior two approaches. Again we sort the dataset by date. Finally we run a simple Proc Transpose step. Notice that all we specify is the name of the input and output datasets. These steps are displayed in Figure 12.

Figure 12 - Simple Transpose – Proc Transpose

```
proc sort data=retail; by date; run;

Proc Transpose data=retail out=proc_transposed_retail;
run;
```

In Figure 13 we see the results. Note that as in the dataset displayed in Figure 5 – the results of the DATA step transpose – we again have a table with one observation, but now we have 60 variables, as is evident in Figure 14, a partial view of Proc Contents output run against our transposed dataset.

Figure 13 - Simple Transpose – SAS Dataset Transposed Using Proc Transpose

VIEWTABLE: Work.Proc_transposed_retail

	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10
1	SALES	Retail sales in millions of $	$220	$257	$258	$295	$247	$292	$286	$323	$284	$307

Notice that the formatting of Dollar10. is carried forward into our new dataset. Notice also that the names of the newly-transposed variables now will be COL1 through COL58 (instead of DATE1 through DATE58). Finally, the reason for ending up with 60 variables is that Proc Transpose creates two new variables in the output dataset, _NAME_ and _LABEL_. Both of these two new variables contain variable labels automatically by the SAS procedure.

Figure 14 - Proc Transpose Results via (partial) Proc Contents

```
                          The CONTENTS Procedure

Data Set Name        WORK.PROC_TRANSPOSED_RETAIL      Observations          1
Member Type          DATA                             Variables             60

                       Variables in Creation Order

          #    Variable    Type    Len    Format       Label

          1    _NAME_      Char      8                 NAME OF FORMER VARIABLE
          2    _LABEL_     Char     40                 LABEL OF FORMER VARIABLE
          3    COL1        Num       8    DOLLAR10.
          4    COL2        Num       8    DOLLAR10.
          5    COL3        Num       8    DOLLAR10.

         //     //         //        //      //        /** (COL4 - COL55) **/

         58    COL56       Num       8    DOLLAR10.
         59    COL57       Num       8    DOLLAR10.
         60    COL58       Num       8    DOLLAR10.
```

Discussion

With a substantially simpler approach than in either of the previous examples we have achieved similar results. Using the SAS Data step approach (see Figure 2 **AND** Figure 4), an approach requiring a certain amount of customizing for each new dataset, we achieve a basic desired result – in 13 lines of code. With the Proc Transpose (see Figure 12) we achieve nearly identical results in just 2 lines of code – which are nearly generic for any application. After all, DATA= and OUT= as the only "customized" elements in this latter approach are likely elements one should be updating in any self-documenting program.

We do lose information in both of these approaches, that is, the dates corresponding to the sales data in the individual columns – information available to us in the original RETAIL dataset. Variable names of DATE1 through DATE58 or COL1 through COL58 retain the date order, but exactly what they correspond to is lost. In contrast, these are retained in the Excel approach (see Figure 11). This approach contains more steps than Proc Transpose, but if the goal is output in Excel in any case, then this approach gathers interest. However, in this instance with a simple additional option to the Proc Transpose we can transfer the date information from the original dataset into the transposed dataset as follows.

Proc Transpose – Options - ID

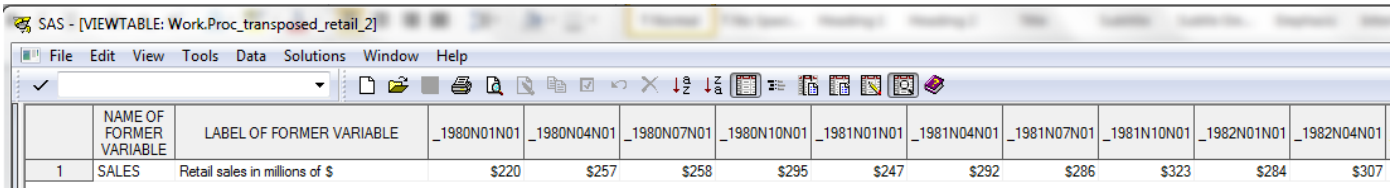
In our first example we have used as simple an invocation of Proc Transpose as is possible. This procedure offers several options which give one additional control of the results. As this is a very simple dataset, very little additional effort is of value. But here we will take control of the output variable names using the ID option as in Figure 15.

Figure 15 - Proc Transpose – ID Option Program

```
Proc Transpose data=retail out=proc_transposed_retail_2;
   id date;
run;
```

With this option we instruct SAS to construct the variable names for the transposed variables in the output dataset using the values of the variable DATE from the incoming dataset. As will be apparent in Figure 16, the date information is retained and is certainly of greater value than, say, COL1 through COL58. It is not nearly as elegant as the results in Excel (see Figure 11), however – the hyphens are placed with "N"s.

Figure 16 - Proc Transpose – ID Option - Output



	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	_1980N01N01	_1980N04N01	_1980N07N01	_1980N10N01	_1981N01N01	_1981N04N01	_1981N07N01	_1981N10N01	_1982N01N01	_1982N04N01
1	SALES	Retail sales in millions of $	$220	$257	$258	$295	$247	$292	$286	$323	$284	$307

Discussion

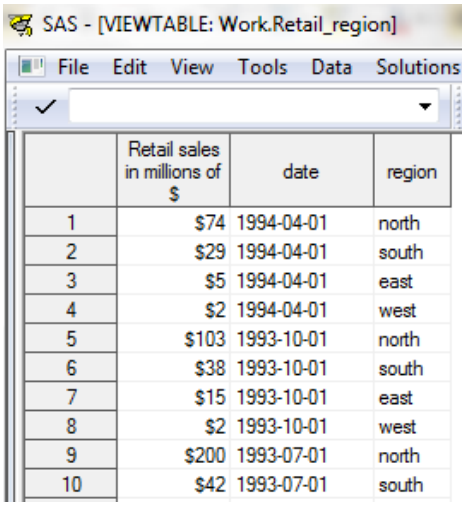
We are perhaps lulled into a false sense of security regarding the use of Proc Transpose with the simplicity of this example. A challenge not immediately noticeable is that Proc Transpose automatically transposes numeric variables – of which SALES qualifies – but not character variables unless these are specified on a VAR statement. Secondly, the nature of one's output dataset are fundamentally controlled by the contents of the input dataset. If we desire different outcomes, we may need to manipulate our

original data in some fashion in advance. And then some of the benefits of the simplicity of Proc Transpose relative to, say, the Data Step approach, are lost. We will explore additional challenges next.

Proc Transpose – Options - BY

We will look at a slightly more complex version of the RETAIL dataset, one with similar sales date broken out by REGION (north, south, east, west). These data are displayed in part in Figure 17.

Figure 17 – Retail SAS Dataset by Region



	Retail sales in millions of $	date	region
1	$74	1994-04-01	north
2	$29	1994-04-01	south
3	$5	1994-04-01	east
4	$2	1994-04-01	west
5	$103	1993-10-01	north
6	$38	1993-10-01	south
7	$15	1993-10-01	east
8	$2	1993-10-01	west
9	$200	1993-07-01	north
10	$42	1993-07-01	south

We will sort by REGION and DATE and then run our Proc Transpose procedure, including a BY statement. Please note that we have 232 observations and 3 variables in the dataset RETAIL_REGION.

Figure 17 – Retail SAS Dataset by Region Proc Contents Output

```
                          The CONTENTS Procedure

Data Set Name        WORK.RETAIL_REGION                Observations          232
Member Type          DATA                              Variables               3


                       Variables in Creation Order

      #    Variable    Type    Len    Format       Label

      1    SALES       Num       8    DOLLAR10.    Retail sales in millions of $
      2    date        Char     10
      3    region      Char      5
```

In Figure 18 we see the Proc Transpose program, new sort, new dataset, but otherwise same program. We know we want to include the BY option but it will be instructive to see what happens without.

Figure 18 – Proc Transpose on Retail_Region with No Changes

```
proc sort data=retail_region; by region date; run;

Proc Transpose data=region out=proc_transposed_region;
   id date;
run;
```

We find our next challenge when we examine our log, namely that the data do not flow through the Proc Transpose engine. In Figure 19 we see the log. In this new dataset, the values of the ID variable DATE are not unique. This is so as we have an observation for each date for each of 4 Regions – what we want in our data. But this is not immediately conducive to creating our transposed dataset. Please note that SAS fails on the 59th observation. We sorted by REGION and date, so we expect SAS to read through all 58 observations (58 unique dates) for Region = east. Next we read the first observation for Region = north, the 59th in our sorted dataset, and an observation with the same/duplicate date value as we expect in the subset of Region = east.

Figure 19 – Proc Transpose on Retail_Region with No Changes – Log Output

```
134  Proc Transpose data=region out=proc_transposed_region;
135     id date;
136  run;

ERROR: The ID value "_1980N01N01" occurs twice in the input data set.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: There were 59 observations read from the data set WORK.REGION.
```

Fortunately we are already heading towards our solution. We will include a BY option on REGION, which will cause Proc Transpose to output one observation per region, and the ID variable DATE resumes having unique values within REGION. As we touched on earlier, Proc Transpose actually will only transpose numeric variables unless we list character variables on a VAR statement. In an effort to be complete and self-documenting, we will include a VAR statement (for the SALES variable). This is not needed in this instance, but likely is a good programming protocol to adopt. The program is displayed in Figure 20.

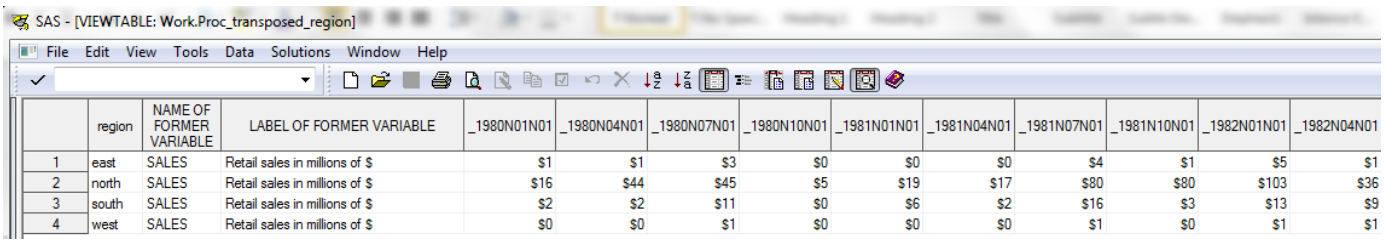
Figure 20 – Proc Transpose on Retail_Region with BY Option

```
Proc Transpose data=retail_region out=proc_transposed_region;
   by region;                    /*** will make date values unique ***/
   id date;
   var sales;                 /*** not necessary here, but good form ***/
run;


                                                    /*** Partial Log ***/
NOTE: There were 232 observations read from the data set WORK.REGION.
NOTE: The data set WORK.PROC_TRANSPOSED_REGION has 4 observations and 61 variables.
```

The resulting dataset is displayed in Figure 21. Proc Transpose takes in 232 observations – one row for each DATE/SALES combination (58) for each of 4 REGIONs (58 x 4 = 232). The output dataset contains four observations, one for each REGION and 61 variables. Here we have the 58 DATE values, the BY variable REGION, and the two variables created by Proc Transpose _NAME_ and _LABEL_. Note that if we want to dispense with these latter two variables, a simple KEEP or DROP statement as appropriate (as a dataset option on the Proc Transpose here, in a later SAS programming step, etc.).

Figure 21 –Transposed Region Dataset Using the BY Option



	region	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	_1980N01N01	_1980N04N01	_1980N07N01	_1980N10N01	_1981N01N01	_1981N04N01	_1981N07N01	_1981N10N01	_1982N01N01	_1982N04N01
1	east	SALES	Retail sales in millions of $	$1	$1	$3	$0	$0	$0	$4	$1	$5	$1
2	north	SALES	Retail sales in millions of $	$16	$44	$45	$5	$19	$17	$80	$80	$103	$36
3	south	SALES	Retail sales in millions of $	$2	$2	$11	$0	$6	$2	$16	$3	$13	$9
4	west	SALES	Retail sales in millions of $	$0	$0	$1	$0	$0	$0	$1	$0	$1	$1

Discussion

A simple additional option – BY – allows us to easily deal with the challenge we just encountered. The resulting dataset is intuitively satisfying, and offers a convenient model for similar problems. But not all datasets are this simple nor flow through the Proc Transpose engine as smoothly. We will next examine certain issues around "difficult" incoming datasets. Sadly, not all issues will generate a SAS error message – which alert us to there being something needing attention. For example, if we neglected to sort the dataset we just used by REGION in advance of run Proc Transpose BY REGION, SAS would let us know that we had made a mistake as in the log shown in Figure 22.

Figure 22 – Error with Helpful Message from SAS

```
251   *** Proc Sort data=retail_region; by region;     /*** we forgot to uncomment ***/

252   Proc Transpose data=retail_region out=proc_transposed_region;
253      by region;
254      id date;
255      var sales;
256   run;

ERROR: Data set WORK.RETAIL_REGION is not sorted in ascending sequence. The current
BY group has region = south and the next BY group has region = east.
NOTE: The SAS System stopped processing this step because of errors.
```

Proc Transpose – Subtle "Errors" #1 – Unsorted/Improperly Ordered Incoming Data

We will look at another example of a Proc Transpose issue which will create problematic output but NOT generate an immediate warning. If we take our original RETAIL dataset – 58 observations, one-per-date value – and neglect to sort by DATE, the Proc Transpose engine will have no difficulty in creating an output dataset. Our starting point can be seen in Figure 23. Note that the observations are essentially randomly ordered.

Figure 23 – Unsorted Dataset

VIEWTABLE: Work.Retail_notsorted

	Retail sales in millions of $	date
1	$323	1981-10-01
2	$797	1992-04-01
3	$758	1993-01-01
4	$705	1990-07-01
5	$284	1982-01-01
6	$318	1982-07-01
7	$384	1983-10-01
8	$385	1984-07-01
9	$295	1980-10-01
10	$998	1994-04-01

Sadly for us, using the program displayed in Figure 24, Proc Transpose will take these data and transpose observation #1 and move the value of $323 into new variable COL1, then observation #2 and move the value of $797 into COL2, $758 will be placed in COL3, and so on until all 58 observations are accounted for. We see this result in Figure 25, a perfectly fine SAS dataset, no errors in our log, and nevertheless **ALL WRONG!!!!!**

Figure 24 – Proc Transpose Unsorted Dataset

```
Proc Transpose data=retail_notsorted1 out=proc_transposed_notsorted;
   var sales;            /*** not necessary here, but good form ***/
run;
```

Discussion

All wrong because COL1 is not actually in the correct column. As we can see just in the first ten observations, observation #9 ought to be listed before observation #1 (if we hope to retain real DATE order). A useful option in Proc Transpose, the PREFIX option will allow us to control the transposed variable names in the output dataset. In Figure 26 we see the slightly changed Proc Transpose program – now including the PREFIX option. Note that this is an option to the Proc Transpose and not a separate statement – unlike the ID and BY statements we used earlier.

Figure 25 – Transposed Unsorted Dataset

SAS - [VIEWTABLE: Work.Proc_transposed_notsorted]

File Edit View Tools Data Solutions Window Help

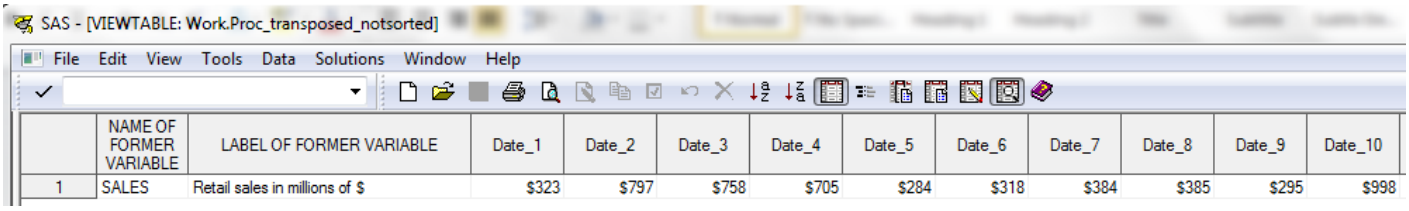
	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10
1	SALES	Retail sales in millions of $	$323	$797	$758	$705	$284	$318	$384	$385	$295	$998

Figure 26 – Proc Transpose Unsorted Dataset – PREFIX Option

```
Proc Transpose data=retail_notsorted
                out=proc_transposed_notsorted
              prefix=Date_;
   var sales;
run;
```

The Proc Transpose PREFIX option certainly gives us a useful tool for certain circumstances, say when simply the order of the columns, along with our "prefixed" description (say, PATIENT1 – PATIENT20), is all the information we need to capture. In Figure 27, however, it becomes entirely misleading because even with "better" variable names, the data are still in random order rather than the DATE order implied by the variable names.

Figure 27 – Transposed Unsorted Dataset – PREFIX Option

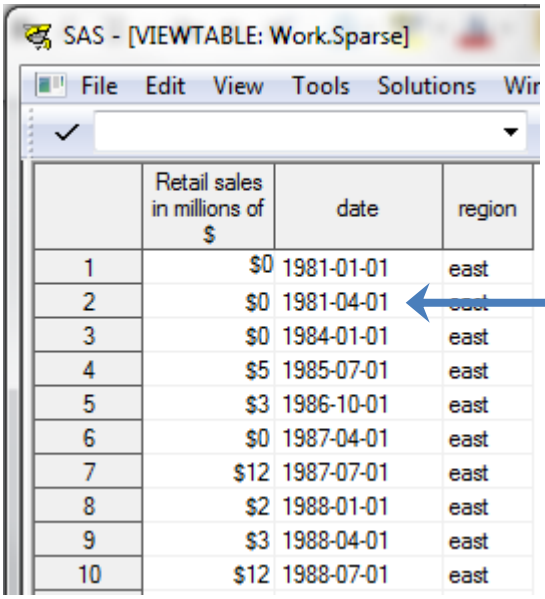


	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	Date_1	Date_2	Date_3	Date_4	Date_5	Date_6	Date_7	Date_8	Date_9	Date_10
1	SALES	Retail sales in millions of $	$323	$797	$758	$705	$284	$318	$384	$385	$295	$998

Proc Transpose – Subtle "Errors" #2 – Sparse Datasets

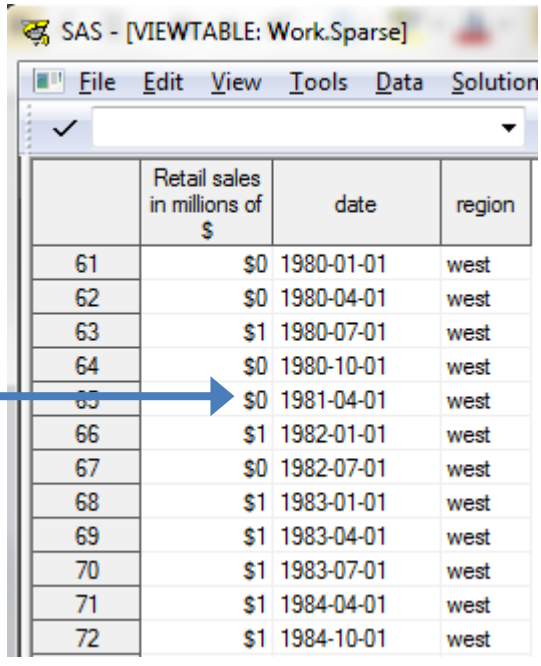
Our final example will look at another challenge, that of "sparse" datasets. As apparent in the unsorted data example, Proc Transpose looks for cues on how to structure the outgoing dataset based on the incoming dataset. With these data we need to sort by DATE to assure that the SALES data will be ordered in the desired column order. These data are partially displayed in Figures 28 and 29.

Figure 28 – "Sparse" Dataset – Top



	Retail sales in millions of $	date	region
1	$0	1981-01-01	east
2	$0	1981-04-01	east
3	$0	1984-01-01	east
4	$5	1985-07-01	east
5	$3	1986-10-01	east
6	$0	1987-04-01	east
7	$12	1987-07-01	east
8	$2	1988-01-01	east
9	$3	1988-04-01	east
10	$12	1988-07-01	east

Figure 29 – "Sparse" Dataset – Bottom



	Retail sales in millions of $	date	region
61	$0	1980-01-01	west
62	$0	1980-04-01	west
63	$1	1980-07-01	west
64	$0	1980-10-01	west
65	$0	1981-04-01	west
66	$1	1982-01-01	west
67	$0	1982-07-01	west
68	$1	1983-01-01	west
69	$1	1983-04-01	west
70	$1	1983-07-01	west
71	$1	1984-04-01	west
72	$1	1984-10-01	west

"Sparse" Data

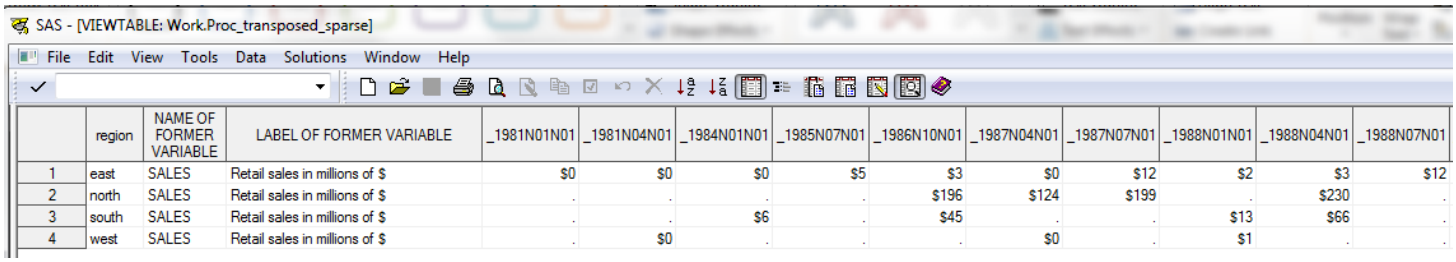
In earlier versions of these data we have an observation for each value of DATE for each value of REGION. In this new dataset, this is not necessarily the case, as for some values of REGION there may not be SALES data for every value of DATE. If we compare Figure 28 – a partial display of data for REGION = east – to Figure 29 – partial display of data for REGION = west – we can see the arrow matching up SALES data to the corresponding date of 1981-04-01 in east and west. Note that the east data contain data for another date in 1981 (1981-01-01), but nothing in the year of 1980, nor 1982 nor 1983. In contrast, the west data contain four data points in 1980, two in 1982, and three in 1983.

Figure 30 – Proc Transpose – "Sparse" Dataset

```
Proc Transpose data=sparse out=proc_transposed_sparse;
   by region;
   id date;                    /*** nothing new here ***/
   var sales;
run;
```

Figure 30 displays our Proc Transpose step – identical to our previous effort except that we are executing this against the "sparse" dataset. Proc Transpose will take the first observation, or the collection of observations in the first BY-group, to create the structure for the remainder of the data. As a result, east being the first BY-group displayed in Figure 28, we should expect the first column in our transposed dataset to correspond to 1981-01-01, the second column should correspond to 1981-04-01, the third column to 1984-01-01, the fourth to 1985-07-01, and so forth. Figure 31 proves this to be correct.

Figure 31 – Transposed "Sparse" Dataset


SAS - [VIEWTABLE: Work.Proc_transposed_sparse]

File Edit View Tools Data Solutions Window Help

	region	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	_1981N01N01	_1981N04N01	_1984N01N01	_1985N07N01	_1986N10N01	_1987N04N01	_1987N07N01	_1988N01N01	_1988N04N01	_1988N07N01
1	east	SALES	Retail sales in millions of $	$0	$0	$0	$5	$3	$0	$12	$2	$3	$12
2	north	SALES	Retail sales in millions of $	.	.	.	.	$196	$124	$199	.	$230	.
3	south	SALES	Retail sales in millions of $	.	.	$6	.	$45	.	.	$13	$66	.
4	west	SALES	Retail sales in millions of $	.	$0	.	.	.	$0	.	$1	.	.

Discussion

Two issues arise from the sparsity in our incoming dataset. The first is that at least in this example we have an output dataset with many, many empty cells (or in SAS-speak, missing values). This can be addressed in a variety of ways appropriate to the use of these data, other steps in our final process, and the like. It may make sense to change missing values to zero or to apply interpolation techniques for some applications, in others to leave these observations as missing, or even as a special missing value. (One can look to a series of discussions in SAS documentation on this topic of Special Missing Values.) Nevertheless, if there are no data for a particular combination of the variables REGION and DATE, then some would offer that, "it is what it is."

Elimination of Data by Proc Transpose

A more alarming challenge is that Proc Transpose has eliminated non-missing data, namely data for any values of DATE which are not present in the first (east) BY-group, but may be very much present in subsequent BY-groups. For example, any data present for 1980, 1982, and 1983 has been eliminated as the first BY-group east has no data for any of those values of DATE (see Figure 31). This occurs, again,

as the first BY-group acts as a template for structuring any subsequent data. If we can manipulate this "template" and assure that the first "row" of data processed is a corrected version of what would otherwise present to the Proc Transpose engine, we will be able to get an improved result. Figure 32 suggests such an approach.

Figure 32 – Proc Transpose – "Sparse" Dataset Corrected

```
                    /*** dynamically generate list of dates (columns) ***/
proc sort data=sparse(keep=date) out=list nodupkey; by date; run;

data template;                              /*** create "template" dataset ***/
   format region $5. sales DOLLAR10.3;
   retain region ' ' sales . ; /** or could be 0? **/
   set list;                 /*** SET statement AFTER FORMAT and RETAIN ***/
run;

data sparse_corrected;  /*** apply Template to correct dataset sparse ***/
   set template    /** must appear first on SET statement **/
       sparse;     /** already sorted appropriately **/
run;

Proc Transpose data=sparse_corrected
                out=proc_transposed_sparse_corrected
                    (where=(region NE ' '));    /*** no longer needed ***/
   by region;
   id date;
   var sales;
run;
```

Discussion

Briefly, we first use Proc Sort to dynamically generate a list of the entire span of DATE values – to be transposed into our columns – which appear in the incoming dataset. In the next DATA step we create a "template" dataset – the definition or skeleton of the structure we wish to present to the proc Transpose engine. We add certain formatting structure to the additional variables REGION and SALES up front – rather than see what might happen if we left it up to chance, as it were. Note that we are suggesting a slightly different format for SALES, still DOLLAR10, but now allowing the display of decimal values. (Are those zeros really zeros, or instead values rounding to less than 1 million?)

We next "apply" the template to the SPARSE dataset – listing TEMPLATE first on the SET statement – resulting in a "corrected" version of the SPARSE dataset. (I.e., the Proc Transpose engine will be presented with a combined dataset, and as the first BY-group is the contents of the TEMPLATE, the structure created will be as constructed and controlled by our previous steps.) Finally, we run a nearly identical Proc Transpose step – only removing the "template" date – on the corrected SPARSE dataset. The datasets LIST, TEMPLATE, and SPARSE_corrected are (partially) displayed in Figures 33, 34, and 35 respectively. Our final, corrected transposed data are shown in Figure 36.

Note that while our results are substantially superior to the uncorrected version, there are, in fact, certain "theoretically possible" values of DATE not encompassed here. This is because the dataset SPARSE is sufficiently sparse to not include every DATE value which appeared in the original sashelp.RETAIL dataset. In practice this is certainly a possibility. If this outcome is unacceptable, instead of dynamically creating the template elements one could instead specify the required bounds using a DO loop within a

DATA step to assure that all values of DATE, or REGION, or whichever variables are of interest for a given problem will be present – even if all associated data are missing – in the resulting dataset.

Figure 33 – Dataset LIST

SAS - [VIEWTABLE: Work.List]

File Edit View Tools

	date
1	1980-01-01
2	1980-04-01
3	1980-07-01
4	1980-10-01
5	1981-01-01
6	1981-04-01
7	1981-07-01
8	1982-01-01
9	1982-07-01
10	1982-10-01

Figure 34 – Dataset TEMPLATE

SAS - [VIEWTABLE: Work.Template]

File Edit View Tools Data Solution

	region	sales	date
1		.	1980-01-01
2		.	1980-04-01
3		.	1980-07-01
4		.	1980-10-01
5		.	1981-01-01
6		.	1981-04-01
7		.	1981-07-01
8		.	1982-01-01
9		.	1982-07-01
10		.	1982-10-01

Figure 35 – Dataset SPARSE_Corrected

SAS - [VIEWTABLE: Work.Sparse_corrected]

File Edit View Tools Data Solution

	region	Retail sales in millions of $	date
43		.	1993-01-01
44		.	1993-04-01
45		.	1993-10-01
46		.	1994-01-01
47		.	1994-04-01
48	east	$0.215	1981-01-01
49	east	$0.212	1981-04-01
50	east	$0.103	1984-01-01
51	east	$4.853	1985-07-01
52	east	$3.393	1986-10-01

Notice that the TEMPLATE formatting for SALES (of DOLLAR10.3) has overridden the format (of DOLLAR10.0) in the incoming dataset – because Proc Transpose encounters the observations from the TEMPLATE dataset first (it appears first on the SET statement). It is a small point, but underlines the need to understand the structure and contents of your data in some detail in order to have a likelihood of good outcomes.

Figure 36 – Transposed Dataset SPARSE_Corrected

SAS - [VIEWTABLE: Work.Proc_transposed_sparse_corrected]

File Edit View Tools Data Solutions Window Help

	region	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	_1980N01N01	_1980N04N01	_1980N07N01	_1980N10N01	_1981N01N01	_1981N04N01	_1981N07N01	_1982N01N01	_1982N07N01	_1982N10N01
1	east	sales	Retail sales in millions of $	.	.	.	.	$0.215	$0.212	.	.	.	.
2	north	sales	Retail sales in millions of $	.	.	.	.	.	.	.	.	.	$31.492
3	south	sales	Retail sales in millions of $	.	$2.196	.	$0.426	.	.	$15.790	$12.523	.	.
4	west	sales	Retail sales in millions of $	$0.310	$0.314	$0.926	$0.012	.	$0.151	.	$0.675	$0.270	.

Conclusion

We have suggested here that Proc Transpose may prove a straightforward approach to what is a fairly complex data manipulation problem. However, the simple dataset used as an example and the step-by-step directed process belies some of the trickier aspects of using this SAS procedure. With any approach there is no substitute for having a fair understanding of the underlying structure of your data as well as a clear picture of what you want in your result set. For simpler data constructs Proc Transpose will hold considerable appeal. In contrast, as an increasing amount of data manipulation is required, the value of using Proc Transpose (instead of the traditional DATA step approach) begins to erode. And if much of your final output needs to be in alternate formats (such as Excel), other solutions begin to gain interest.

As with learning any new technique, starting simply and building incrementally are usually the ideal approach. (Or find good examples of other's successful programs and leverage these.) While there is an unfortunate tendency to find much of one's efforts reduced to a brute force, trial-and-error approach, a substantial investment in development and validation are likely justified when a program will be run (or cloned) many times over. As with any technique, if you find success using it, be sure to document extensively and store examples of working code in locations from which retrieval will be easy. Once this is accomplished, you will have added another tool to your solution box.

References

There are many helpful papers in the proceedings of the collected SAS User Group Conferences. Several specific to Proc Transpose are listed here along with an approach to variable naming manipulation (see Cohen, 2011) and links to SAS documentation on Proc Transpose and on Special Missing Values.

Cohen, John, "Hands Free- Automating Variable Name Re-Naming Prior to Export", Proceedings of the 2011 NorthEast SAS Users Group (NESUG) Conference, .September, 2011.

Joseph, Katie, "Swap the DATA Step for PROC TRANSPOSE", Proceedings of the 2007 NorthEast SAS Users Group (NESUG) Conference, November 2007.

Lavery, Russ, "An Animated Guide: PROC TRANSPOSE", Proceedings of SAS Global Forum 2007 Conference, April 2007.

Leighton, Ralph W., "Some Uses (and Handy Abuses) of PROC TRANSPOSE", Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference, May 2004.

Stuelpner, Janet, "The TRANSPOSE Procedure or How to Turn It Around", Proceedings of the Thirty-First Annual SAS® Users Group International Conference, March 2006.

SAS Institute, Inc., "Missing Values: Creating Special Missing Values", http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/a000992455.htm, 2010-02-11.

SAS Institute, Inc., " The TRANSPOSE Procedure: PROC TRANSPOSE Statement", Base SAS(R) 9.2 Procedures Guide, http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/a000063663.htm, 2010-05-25.

Acknowledgements

The author would like to thank Meenal Sinha for certain suggestions regarding the approach to this paper and for her encouragement during its preparation.

Contact Information

Your comments and questions are valued and encouraged. You may contact the author at:

John Cohen
Advanced Data Concepts LLC
Newark, DE
(302) 559-2060
jcohen1265@aol.com