Techniques For Improving Coding and Processing Efficiency in the DATA Step

Herb Reynolds, Viar and Company, Inc.
Thomas Jacob, Viar and Company, Inc.

ABSTRACT

The SAS® System is heavily used in certain large data processing environments. As with any programming language, if not used efficiently, programs can be a burden on the computer system. Although it takes anywhere from a few weeks to several months for one to get a basic understanding of the SAS system, experience and imagination are needed to do complex programming tasks in an efficient manner.

This paper will discuss several different ways of solving certain programming tasks. Some of the techniques described are very efficient in coding, others are efficient in processing. With a little imagination, any programming task, however difficult it may seem, can be done in an easy and efficient manner. The concepts behind these examples show the power of the SAS programming language in controlling processing in the data step.

TOPIC I. TRANSPOSING DATA: Changing observations into variables.

An occasional and often difficult data manipulation task for the SAS programmer is the job of inverting a dataset or changing observations into variables. This is sometimes necessary when data are arranged vertically and must be presented horizontally in a report. A common example of this is the task of summing data by month for a series of values.

The following discussion demonstrates and compares three techniques for transposing observations into variables. First an example input dataset is described and the desired output dataset is shown. Three methods of manipulating this dataset into the desired output are then demonstrated. The advantages, disadvantages and limitations of each method are also discussed.

The example dataset shown below (LABDATA) consists of a number of observations containing information about laboratories. Each observation contains three variables: MONTH, which is the number of the month, LAB, which contains the name of the laboratory, and DEFECT, which indicates the number of defects reported at that lab for that month.

<table>
<thead>
<tr>
<th>LAB</th>
<th>MONTH</th>
<th>DEFECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>X1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>X1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>X1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>X2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>X2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>X3</td>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>

There can be a maximum of 12 observations per lab or one for each month (M1 - M12) of the year. The number of laboratories may vary and there may not be an observation for each laboratory for each month. The objective is to convert or interchange the observations in the LABDATA dataset into one observation per lab.

The output dataset (TARGET) must have the following form:

<table>
<thead>
<tr>
<th>LAB</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>...</th>
<th>M12</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X3</td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

METHOD I: Using FIRST, and LAST, variables.

The most frequently used method to create a single observation from many observations in the original dataset is to make use of the "FIRST," and "LAST," variables of the BY variable LAB. After sorting the dataset by the variable LAB the following code may be executed.

```
DATA TARGET(KEEP = LAB M1-M12);
SET LABDATA;
BY LAB;
RETAIN M1-M12;
IF FIRST.LAB THEN DO;
   M1 = ; ; M2 = ; ; M3 = ; ; ; M12 = ;
END;
IF MONT = 1 THEN MI = DEFECT;
ELSE IF MONT = 2 THEN M2 = DEFECT;
ELSE IF MONT = 3 THEN M3 = DEFECT;
   ;
   ;
ELSE IF MONT = 12 THEN M12 = DEFECT;
IF LAST.LAB THEN CUNIT;
```

The dataset created from this code is the desired TARGET dataset. This method is logically simple and easily coded for small and uncomplicated datasets, i.e. datasets with a small number of values of the variable to be interchanged. If the variable being interchanged has an unknown number of values, it is almost impossible to code this method. For more complex applications this method may require a large number of lines of code which can increase the chance of making mistakes.
METHOD 2: Using PROC TRANSPOSE.

The SAS System procedure PROC TRANSPOSE can be used to attain the same desired TARGET dataset if the number of observations per lab is constant. The code for the use of the TRANSPOSE procedure is as follows:

```
PROC TRANSPOSE DATA=LABDATA PREFIX=M;
   BY LAB;
   VAR DEFECT;
```

The TRANSPOSE procedure will interchange the data in one simple step, provided there are no missing observations or "holes" in the dataset. The transposed dataset is shown below.

Output of TRANSPOSE Procedure.

<table>
<thead>
<tr>
<th>LAB</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>...</th>
<th>M12</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X3</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice that the output dataset is not correct and is far from the desired results - certain values are shifted to different months. Since there were observations for each month for LAB "X1" the transposed data is correct for LAB "X1". For LAB "X2", the values of the "MONTH" variables (M1-M12) have been shifted, the output dataset shows the value of "2" for DEFECT for "M1", even though no data exists for "M1" in the original dataset. When using this procedure, as demonstrated by this example, one has to make sure that there are no missing observations within each "BY" group. This greatly limits the use of the TRANSPOSE procedure. If there are missing data - and it may be very difficult to tell if there are missing data in large datasets - this procedure cannot be used.

METHOD 3: The use of an ARRAY.

In this method we make use of an array. The dataset must first be sorted by the variable 'LAB' as in Method 1.

```
DATA SAMPLE;
   KEEP LAB M1-M12;
   ARRAY LIST(M2-M12)
   DO MONTH = 1 TO 12;
      SET LABDATA;
      BY LAB;
      LIST(MONTH) = DEFECT;
      IF LAST.LAB THEN DO;
         OUTPUT;
      END;
   END;
```

Since the variable MONTH is numeric, we can create an index variable with the same name to point to cells in the array called LIST. The array LIST contains M1 through M12 (month 1 through month 12). Since the SET statement is within the DO LOOP, the value of the index variable MONTH is dictated by the value of the variable MONTH. With this careful manipulation, we force the value of DEFECT to go to the correct place, even though there may be missing data points within the "BY" variable LAB. By referencing the array implicitly, we do not have to know beforehand the maximum number of observations within a LAB. When the number of observations within the "BY" variable is large, this technique becomes very efficient. Since the number of lines of code needed for this method is relatively small, this method is less prone to programming errors.

TOPIC II: USE OF THE "END" OPTION

Another common task encountered by SAS programmers is the addition of totals or other data values to all observations in a dataset. This task can be accomplished by several simple methods which will not be discussed here. We will discuss in detail the method which is probably the most efficient and easiest to use - the use of the second dataset.

Consider the two datasets SAMP and TOTAL shown below. The dataset SAMP contains three observations with one variable named DEFECT. The dataset TOTAL, which was created by summing the DEFECT field in the dataset SAMP, contains one observation and one variable named TOT.

Dataset: SAMP                      Dataset: TOTAL
SAMPLE  DEFECT  TOT
ABI23    2        9
ABI24    3        9
ABI25    4        9

The objective here is to assign the value of TOT from the TOTAL dataset to each observation in the dataset SAMP. This can be accomplished by the following simple lines of code.

```
DATA SAMPLE;
   SET SAMP;
   IF _N_ = 1 THEN SET TOTAL;
```

The resulting output dataset is:

<table>
<thead>
<tr>
<th>SAMPLE</th>
<th>DEFECT</th>
<th>TOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABI23</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>ABI24</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>ABI25</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
This method will work fine if the dataset TOTAL contains observations. If the dataset TOTAL is empty and contains no observations, this code will not work and the resulting dataset will be null. If any of the datasets input into this data step run out of observations SAS will stop building the output dataset. This null dataset can cause unexpected and sometimes fatal errors in the code following this step.

If the TOTAL dataset happened to contain no observations we would want the output dataset to look like the following:

<table>
<thead>
<tr>
<th>SAMPLE</th>
<th>DEFECT</th>
<th>TOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB123</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>AB124</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>AB125</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

The problem of the empty dataset can be solved by using the 'END' variable which is added to the program data vector and initialized at compile time. The code below demonstrates this concept.

```sas
DATA samp;
SET samp;
IF EOF = 0 THEN TOTAL END=EOF;
```

Notice where the "END=EOF" code is located. The addition of the 'END' variable will cause the SAMP dataset to be output with the TOT variable added even if the dataset is empty. The value of the TOT variable will be missing, however, the addition of the missing value for TOT created by this code will allow subsequent data steps which use this output dataset to continue execution.

**TOPIC III. ADDING SUMMARY STATISTICS**

This topic deals with the problem of adding summary statistics or totals to a dataset for a report. The calculations may include accumulating a total across a "BY GROUP" and saving it for later use. In this discussion we will show two methods, one using standard SAS code and one using SQL, for generating summary statistics for a dataset.

**EXAMPLE:**

There exist two datasets, one for department information (ANALYST) and another for tracking defects by analyst (DEFECT).

**Dataset: ANALYST**

<table>
<thead>
<tr>
<th>DEPT</th>
<th>ANALYST</th>
<th>DEFECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Herb</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>Ruth</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>Thomas</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Joyce</td>
<td>4</td>
</tr>
</tbody>
</table>

**Dataset: DEFECT**

We need to generate a report showing the percentage of defects made by each analyst within a particular department. The percentage of defects for an analyst is calculated with respect to their own department. The output dataset or report should look like the following:

**Percentage of Defects by Department and Analyst**

<table>
<thead>
<tr>
<th>DEPT</th>
<th>ANALYST</th>
<th>PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Herb</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>Ruth</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>Thomas</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>Jim</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>Joyce</td>
<td>20</td>
</tr>
</tbody>
</table>

**Method 1: Using Standard SAS Code.**

The following code shows how this task would be done using standard SAS code. Notice that to accomplish our goal we must do three "sorts", two "merges", one "data step", and then print the report.

```sas
PROC SORT DATA=ANALYST; BY ANALYST;
PROC SORT DATA=DEFECT; BY ANALYST;
DATA ANALYST;
MERGE ANALYST DEFECT;
BY ANALYST;
PROC SORT DATA=ANALYST; BY DEPT;
DATA SUMMARY(DEPT=DEFECT SUMTOTAL=);
SET ANALYST;
BY DEPT;
RETAIN SUBTOTAL;
IF FIRST.DEPT THEN SUBTOTAL = 0;
SUBTOTAL = SUBTOTAL + DEFECT;
IF LAST.DEPT THEN OUTPUT;
DATA ANALYSIS;
MERGE ANALYST SUMMARY;
BY DEPT;
PERCENT = (DEFECT/SUBTOTAL)*100;
PROC PRINT DATA=ANALYSIS;
TITLE 'Percentage of Defects by Department and Analyst';
```

**Method 2: Using PROC SQL.**

We can optimize and simplify the above process by using the SAS version 6.06 "PROC SQL". The Structured Query Language (SQL) is a standardized, widely used language that retrieves data from relational databases. Most versions of SQL require two passes through the data to accomplish the percentage calculations shown above. The SAS system enhancement to SQL accomplishes this calculation with only one pass through the file. The SQL code shown below achieves the same result as the standard code shown in Method 1.
PROC SQL;
SELECT A.ANALYST,DEPT,(SUM(A.DEFECT)*100) AS PERCENT
FROM ANALYST AS A, DEFECT AS B
WHERE A.ANALYST = B.ANALYST
GROUP BY DEPT;

Note:
The "SELECT" statement selects the columns to be printed.
The "FROM" statement links the two relational databases.
The "WHERE" statement links the databases in the "FROM" statements by a common variable.

The results of this SQL procedure are the same as the results in Method 1. One can easily see that the SQL procedure uses fewer lines of code than the standard SAS code shown in Method 1 and it accomplishes the same task. It was also found that the SQL procedure in Method 2 is more efficient and runs much faster (uses less CPU time) than Method 1.

CONCLUSIONS

It is important to understand that in any programming language there are always more than one way to solve a given problem. The various programming solutions have their respective advantages and disadvantages which must be considered before choosing one method over another. Seemingly complex programming tasks can sometimes have relatively simple solutions if one has a good understanding of the SAS system. With experience and imagination any SAS programming task can be made manageable and efficient.

SAS® is a registered trademark of SAS Institute Inc., Cary, N.C., USA.

Questions or comments regarding this paper should be directed to:

Herb Reynolds or Thomas Jacob
Vic and Company, Inc.
300 North Lee Street
Alexandria, Virginia 22314
phone: 703-684-5678