Challenges in Reshaping & Reporting DATA - Overcome with ARRAYS & 'V' Function

Manjusha Alapati, Mumbai, India
Vidyasagar Palli, Mumbai, India

ABSTRACT
Reshaping or restructuring a dataset is one of the most frequent approaches for reporting data. The most common approaches used are PROC TRANSPOSE or using ARRAYS in DATA step. PROC TRANSPOSE might not cover all the situations that users may encounter in real programming world and can make the work tedious, repetitive and time consuming. Here we discuss how to overcome those by using ARRAYS in DATA step which has control over the reshaping process. Handling the task of restructuring data to switch SAS variables with logical record key (BY Group), we need to consider about details like,

I. No of variables need to be transposed
II. Variable Types
III. Report Structure/Layout

Considering a real-time scenario with LAB listing with its Mock-up, we will try to convey efficient way to achieve reshaping with ARRAY and "V" functions.

INTRODUCTION
If we want to conduct an analysis/operation across observations in a data set we can use SAS Procedures. Or, if we want to conduct an analysis within observations we can use SAS functions. However, data needs to be rearranged to perform computation on logical record key (BY group) or to report the data in a meaningful way. Rearranging in this context seems to converting variables into observations or it's vice versa.

This paper will examine both the techniques, PROC TRANSPOSE and ARRAY with certain tasks/questions for comparing advantages and limitations of each.

PROC TRANSPOSE
The TRANSPOSE procedure creates an output data set by restructuring the values in a given SAS data set, transposing selected variables into observations. It consists of set of statements and options. Its basic syntax as follows,

PROC TRANSPOSE <DATA=input-data-set> <OUT=output-data-set> <PREFIX=prefix> <SUFFIX=suffix>;
BY <DESCENDING> variable-1 <…<DESCENDING> variable-n> <NOTSORTED>;
ID variable;
VAR variable(s);

BY - Transpose will take place from a set of observations comprising of unique BY group values.
ID - Specifies a variable whose values name the transposed variables.
VAR - Variable(s) which needs to be transposed.

PREFIX option specifies a prefix to use in constructing names for transposed variables in the output data set.

While transposing the data we need to take care of two aspects as below,

i. Variable which is used as ID variable or the variable which names the variables of the output dataset with its value must be unique for the logical record key (BY group).
ii. ID variable values should not be missing and missing value records will not be transposed.
ARRAY PROCESSING

ARRAY is a temporary grouping of SAS variables that are arranged in a particular order and identified by an array-name. Array may allow us to simplify our processing and helps us to analyze repetitive data with a minimum of coding.

Basic ARRAY processing involves the following steps:
- Grouping variables into ARRAY
- Selecting a current variable for an action
- Repeating an action

Simple ARRAY statement is as follows,

    array array-name { n } <$> <length><array-elements>;

The ARRAY statement provides the following information about the SAS array:
- array-name – Any valid SAS name
- n – Number of elements within the array
- $ - Indicates the elements within the array are character type variables
- length – A common length for the array elements
- elements – List of SAS variables to be part of the array

Array-name is the name given to identify the group of variables specified as array-elements. We can group up and create new variables of same type i.e., either character or numeric but not mixed. The array name should not be the same name as any variable on the SAS data set.

An array must be defined within the data step prior to being referenced. Defining an array within one data step and referencing the array within another data step will cause errors, because ARRAYS exist only for the duration of the data step in which they are defined. Once an array is defined, the array name and an index reference the elements of the array.

When we define the number of elements in an ARRAY, we can either use an asterisk (*) enclosed by braces to count the number of elements or to specify the number of elements. If we specify the number of elements explicitly, we can omit the names of the variables or ARRAY elements in the ARRAY statement. When the asterisk is used, it is not necessary to know how many elements are contained within the array. SAS will count the number of elements for you.

When an array is defined with the ARRAY statement SAS creates an array reference. The variable name and the array reference are interchangeable. When an array has been defined in a data step either the variable name or the array reference may be used. We can reference the variable in an ARRAY with its location, ARRAYS are often processed in iterative loops and references to the array are usually found within DO groups. The combination of ARRAYS and do loops in the data step lend incredible power to programming.

RESTRUCTURING: SIMPLE CASE

Let’s begin with simple example where the subject is having multiple records of parameter and visit that needs to be transposed and the resultant dataset should have one record per subject per parameter.

Display 1: Sample Data Vitals

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>PARAM</th>
<th>VISIT</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001</td>
<td>DBP</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1001</td>
<td>DBP</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1001</td>
<td>SBP</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1001</td>
<td>SBP</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1002</td>
<td>DBP</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1002</td>
<td>SBP</td>
<td>1</td>
</tr>
</tbody>
</table>

Using PROC TRANSPOSE,

    proc transpose data=vitals out=trans prefix=visit;
    by subj id param;
    id visit;
    var result;
    run;
Using ARRAYS,

```
data array;
  set vitals;
  by subjid param;
  array arl(*) visit1 visit2;
  if first.param then do i=1 to dim(arl);
    arl(i)=.;
  end;
  arl(visit)=result;
  retain arl;
  if last.param then output;
run;
```

Display 2: Transposed data

With both the logics we can achieve the same output as shown in Display 2, but there are few advantages and limitations with both the logics as below,

- As the syntax was simple and easy to remember PROC TRANSPOSE is suitable for above data.
- While using ARRAYS we need to handle variables carefully across the iterations by using RETAIN statement and for the new logical key record retain variables must be initialized to missing.
- With the existence of visit as numeric variable it helped in indexing the variables. If the values for visit are different from 1, 2 or not numeric then further implementation of logic is required.

RESTRUCTURING MULTIPLE VARIABLES

If the context is to turnaround more than one variable into observation with a logical record key (BY group) is not an easy task by using PROC TRANSPOSE. It follows with a transpose, merge of split datasets and renaming the variables to avoid overlapping. But the whole thing can be performed in a single DATA step using ARRAYS. To transpose the data shown in Display 3 for one record per parameter,

```
proc transpose data=vitals out=trans prefix=trt;
  by param;
  id trtgrpn;
  var n mean median;
run;
```

Display 3: Descriptive statistics of vital data

<table>
<thead>
<tr>
<th>PARAM</th>
<th>TRTGRPN</th>
<th>N</th>
<th>MEAN</th>
<th>MEDIAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBP</td>
<td>1</td>
<td>9</td>
<td>791</td>
<td>79</td>
</tr>
<tr>
<td>SBP</td>
<td>2</td>
<td>8</td>
<td>52</td>
<td>66</td>
</tr>
<tr>
<td>DBP</td>
<td>3</td>
<td>7</td>
<td>569</td>
<td>64</td>
</tr>
<tr>
<td>SBP</td>
<td>1</td>
<td>6</td>
<td>33</td>
<td>124</td>
</tr>
<tr>
<td>SBP</td>
<td>2</td>
<td>5</td>
<td>5.34</td>
<td>1.36</td>
</tr>
<tr>
<td>SBP</td>
<td>3</td>
<td>6</td>
<td>7.28</td>
<td>1.42</td>
</tr>
</tbody>
</table>
With PROC TRANSPOSE the resultant data is as below in Display 4,

<table>
<thead>
<tr>
<th>PARAM</th>
<th><em>NAME</em></th>
<th>trt1</th>
<th>trt2</th>
<th>trt3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DBP</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>MEAN</td>
<td>731</td>
<td>592</td>
<td>589</td>
</tr>
<tr>
<td>3</td>
<td>MEDIAN</td>
<td>79</td>
<td>86</td>
<td>84</td>
</tr>
<tr>
<td>4</td>
<td>SBP</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>MEAN</td>
<td>631</td>
<td>634</td>
<td>720</td>
</tr>
<tr>
<td>6</td>
<td>MEDIAN</td>
<td>124</td>
<td>136</td>
<td>142</td>
</tr>
</tbody>
</table>

Display 4: Transpose output of vital data

data trans_fin;
merge trans(where=(lowcase(_name_) eq "n")
rename=(trt1=trt1_n trt2=trt2_n trt3=trt3_n))
trans(where=(lowcase(_name_) eq "mean")
rename=(trt1=trt1_mean trt2=trt2_mean trt3=trt3_mean))
trans(where=(lowcase(_name_) eq "median")
rename=(trt1=trt1_median trt2=trt2_median trt3=trt3_median));
by param;
run;

With the above merge the resultant dataset is as below in Display 5 with an observation per parameter,

<table>
<thead>
<tr>
<th>PARAM</th>
<th>trt1_n</th>
<th>trt2_n</th>
<th>trt3_n</th>
<th>tt1_mean</th>
<th>tt2_mean</th>
<th>trt1_median</th>
<th>trt2_median</th>
<th>trt3_median</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DBP</td>
<td>9</td>
<td>8</td>
<td>731</td>
<td>592</td>
<td>589</td>
<td>79</td>
<td>84</td>
</tr>
<tr>
<td>2</td>
<td>SBP</td>
<td>6</td>
<td>5</td>
<td>631</td>
<td>634</td>
<td>720</td>
<td>124</td>
<td>136</td>
</tr>
</tbody>
</table>

Display 5: Transposed output with one observation per parameter

By using ARRAYS we can achieve the output as in Display 5 with the below code in single DATA step,

data array_vital(drop=mean median n trtgrp);
set vitals;
array arn(*) trt1_n trt2_n trt3_n;
array armean(*) trt1_mean trt2_mean trt3_mean;
array armedian(*) trt1_median trt2_median trt3_median;
by param;
if first.param then do i=1 to dim(arn);
  arn(i)=.;
  armean(i)=.;
  armedian(i)=.;
end;
arn(trtgrp)=n;
armean(trtgrp)=mean;
armedian(trtgrp)=median;
retain arn armean armedian;
if last.param then output;
run;

Code 1 Using ARRAYS

To restructure the data shown in Display 6 we can still make use of above code for Transpose.

<table>
<thead>
<tr>
<th>PARAM</th>
<th>CAT</th>
<th>N</th>
<th>MEAN</th>
<th>MEDIAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DBP</td>
<td>BL</td>
<td>9</td>
<td>731</td>
</tr>
<tr>
<td>2</td>
<td>DBP</td>
<td>PB</td>
<td>6</td>
<td>728</td>
</tr>
<tr>
<td>3</td>
<td>SBP</td>
<td>BL</td>
<td>6</td>
<td>831</td>
</tr>
<tr>
<td>4</td>
<td>SBP</td>
<td>PB</td>
<td>5</td>
<td>634</td>
</tr>
</tbody>
</table>

Display 6: Descriptive statistics of vitals data per param per category
**PhUSE 2014**

```sas
proc transpose data=vitals out=trans prefix=trt;
    by param;
    id cat;
    var n mean median;
run;

data trans_fin;
    merge trans(where=(lowcase(_name_) eq "n")
        rename=(bl=bl_n pb=pb_n))
    trans(where=(lowcase(_name_) eq "mean")
        rename=(bl=bl_mean pb=pb_mean))
    trans(where=(lowcase(_name_) eq "median")
        rename=(bl=bl_median pb=pb_median));
    by param;
run;
```

With the above code, the transposed data as below in Display 7

Display 7: Transposed data of vitals data per parameter

<table>
<thead>
<tr>
<th>PARAM</th>
<th>bl_n</th>
<th>pb_n</th>
<th>bl_mean</th>
<th>pb_mean</th>
<th>bl_median</th>
<th>pb_median</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DBP</td>
<td>9</td>
<td>6</td>
<td>731</td>
<td>726</td>
<td>73</td>
</tr>
<tr>
<td>2</td>
<td>SBP</td>
<td>6</td>
<td>5</td>
<td>831</td>
<td>534</td>
<td>124</td>
</tr>
</tbody>
</table>

But the above Code 1 Using ARRAYS won’t work for the data shown in Display 6 due to non existence of numeric variable for indexing the position of the variable in the ARRAY. But with a slight modification in code 1 will make it efficient for restructuring/turn around the data with multiple variables. Before that we need to evaluate both the logics used for transposing data with multiple variables.

While transposing multiple variables with PROC TRANSPOSE we need to split the output dataset in to 'n' number of datasets if we are transposing 'n' variables. Those 'n' number of datasets are merged on basis of logical key record (BY group), while merging these datasets care should be taken to rename the variables to avoid overlapping.

While using ARRAYS we can handle the whole process with unique observation for each logical key record (BY group) in a single DATA step by initializing ‘n’ number of ARRAYS to transpose ‘n’ variables. This logic helps in implementation if there is an existence of numeric and suitable for indexing the specific variable in an ARRAY. For example as per the data shown in Display 6 to transpose data we need to implement a reliable technique which helps in placing the result to the correct variable irrespective of the order in which the variables were grouped in an ARRAY.

This can be achieved with the use of additional ARRAY as a look-up and “VNAME” function shown as below,

```sas
data array_vital(drop=mean median n cat i j bl pb);
    set vitals;
    by param;
    array lookup(*) BL PB;
    array arn(*) bl_n pb_n;
    array armean(*) bl_mean pb_mean ;
    array armedian(*) bl_median pb_median ;
    if first.param then do i=1 to dim(arn);
        arn(i)=.;
        armean(i)=.;
        armedian(i)=.;
    end;
    do j=1 to dim(lookup);
        if strip(upcase(cat)) eq vname(lookup(j)) then do;
            arn(j)=n;
            armean(j)=mean;
            armedian(j)=median;
        end;
    end;
    retain arn armean armedian;
    if last.param then output;
run;
```

With the above code we can achieve the data transposition shown as in Display 7. Here the ARRAY initialized as
LOOKUP has grouped the distinct values of CAT variable as variables which all need to be unique for respective logical key record (By group) for a successful transpose. So, with this array logic we are creating the variables with the help of ARRAY initialization which all are required for transpose before they all come in to existence. Where as in Transpose while merging the datasets we are renaming the variables which all are created from transpose output.

In addition to the ARRAY which we are using for look-up we need to initialize ARRAYS depending on number of variables need to be transposed and the number of elements in each ARRAY should be same as the number of elements in the look-up ARRAY.

With the help of VNAME function we can identify the location of the variable from the look-up ARRAY and using that information as index value to reference the variable and to assign the value of the variable transposed to the respective variable in the respective ARRAY where all the variables grouped are assumed to be the same order as in look-up ARRAY.

Example: As per the data shown in Display 6, with the help of look-up ARRAY how the variables were transposed is shown as below,

```do j=1 to dim(lookup);
  if strip(upcase(cat)) eq vname(lookup(j)) then do;
    arn(j)=n;
    armean(j)=mean;
    armedian(j)=median;
  end;
end;
retain arn armean armedian;
if last.param then output;
```

Do loop will run twice as the number elements in the ARRAY retuned from DIM (LOOKUP) is 2 and for the first record in the dataset the IF condition satisfies with index variable j = 1 with this,

```
arn(1) referenced to BL_N assigned with 9
armean(1) referenced to BL_MEAN assigned with 791
armedian(1) referenced to BL_MEDIAN assigned with 79
```

And with the use of retain statement these variables are retained to next iteration and for the second record IF condition satisfies with index variable j=2 with this,

```
arn(2) referenced to PB_N assigned with 6
armean(2) referenced to PB_MEAN assigned with 728
armedian(2) referenced to PB_MEDIAN assigned with 142
```

As the IF condition satisfies the last.param the record was outputted as below for the first BY group,

<table>
<thead>
<tr>
<th></th>
<th>PARAM</th>
<th>bl_n</th>
<th>pb_n</th>
<th>bl_mean</th>
<th>pb_mean</th>
<th>bl_median</th>
<th>pb_median</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DBP</td>
<td>9</td>
<td>6</td>
<td>751</td>
<td>728</td>
<td>79</td>
<td>142</td>
</tr>
</tbody>
</table>

RESTRICTURING FOR REPORT LAYOUT

Sometimes we need to reshape the data to report it in a meaningful way for that we might need to transform our data from its longitudinal structure to wider. Here we will try to elaborate the limitation of PROC TRANSPOSE while reshaping the data from longitudinal to wide for reporting the data with the help of two lab listing mock shells,

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>LBTEST</th>
<th>RANGE</th>
<th>UNITS</th>
<th>VISIT</th>
<th>COLLDATE</th>
<th>LBDAY</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>CALCIUM</td>
<td>2-10.6</td>
<td>mg/dL</td>
<td>1</td>
<td>20JAN2012</td>
<td>21</td>
<td>9.9</td>
</tr>
<tr>
<td>1001</td>
<td>CALCIUM</td>
<td>2-10.6</td>
<td>mg/dL</td>
<td>2</td>
<td>27JAN2012</td>
<td>28</td>
<td>9.8</td>
</tr>
<tr>
<td>1102</td>
<td>ALT</td>
<td>0-59</td>
<td>IU/L</td>
<td>1</td>
<td>22JAN2012</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>1102</td>
<td>ALT</td>
<td>0-59</td>
<td>IU/L</td>
<td>2</td>
<td>30JAN2012</td>
<td>28</td>
<td>35</td>
</tr>
</tbody>
</table>

Data 1: Lab data
Table 1: Lab listing Mock-1

For the layout shown in Table 1, the data in Data 1 needs to be transposed subject wise with one record per lab test by using ARRAYS and transpose procedure. PROC TRANSPOSE is a better approach to implement as we need to transpose single variable VISIT and the values in VISIT were unique across the subjects.

```plaintext
proc transpose data=lab out=tran prefix=visit_;  
    by trtgroup subjid lbtest range units;  
    id visit;  
    var result;  
run;
```

With the above code we can transpose the data in to required shape from narrow to wide with all the visit data for each lab test by using transpose procedure with its basic syntax and prefix option.

But we can’t reshape the data shown in Data 1 for the lay out shown in Table 2 with transpose procedure, Table 2: Lab listing Mock-2

```plaintext
proc transpose data=lab out=tran;  
    by trtgroup subjid lbtest range units;  
    id colldate;  
    var result;  
run;
```

Data 1: Output dataset created by PROC TRANSPOSE

According to the Mock shell provided in Table 2, we need to display data one record per lab test for a subject with their collection date (COLLDATE). Even here we need to transpose single variable COLLDATE but the values of this variable are not unique across the subjects which leads in wider data and the resultant data is not feasible in reporting subject wise from the transposed dataset which is an outcome from PROC TRANSPOSE.

For example, as per the data shown in Data 1 lab data Subjid 1001 didn’t have any lab assessment done on 22JAN2012 but the other subjects might have some assessments done on that date like Subjid 1102. So, after transposing the whole data the resultant number of columns will be the count of distinct number of COLLDATE values in addition to the logical key records used for transpose.

So, the limitation of PROC TRANSPOSE in transforming the data for Table 2 layout is, if the Unique values in ID variable increases restrctured data set results too wider and in turn some of the resultant variable values will be missing for the respective logical key record.

But with the help of ARRAYS we can transpose the data with required number of variables to accommodate the data for reporting as below,
**Step-1:** Count the number of distinct number of dates across each subject and determine maximum number of dates out of it, in order to define ARRAY size to transpose data. Resultant dataset MAXDT is as shown below, macro variable MAXDTCNT helps in defining array size.

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>DTCNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>12</td>
</tr>
<tr>
<td>1002</td>
<td>3</td>
</tr>
<tr>
<td>.......</td>
<td>.......</td>
</tr>
<tr>
<td>1102</td>
<td>15</td>
</tr>
</tbody>
</table>

**Data 2:** Distinct number of dates per subject

```sql
proc sql noprint;
  select max(distinct dtcnt) into: maxdtcnt from maxdt;
quit;
```

**Step-2:** With the help of maximum number of distinct dates per subject we can define the ARRAY with that element size to accommodate the transposed data, for that we have to create a dataset with one observation per subject with all the lab assessment dates for the respective subject as below in Data 4, DT1-DTn variables are numeric variables in the same way we will create set of character variables DTC1-DTCn to hold the character date values.

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>DT1</th>
<th>DT2</th>
<th>.........</th>
<th>DTN</th>
<th>DTC1</th>
<th>.........</th>
<th>DTCN</th>
<th>DTCNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1002</td>
<td>20JAN2012</td>
<td>01FEB2012</td>
<td>.........</td>
<td>20JAN2012</td>
<td>.........</td>
<td>20JAN2012</td>
<td>.........</td>
<td>3</td>
</tr>
<tr>
<td>.......</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.......</td>
</tr>
<tr>
<td>1102</td>
<td>22JAN2012</td>
<td>30JAN2012</td>
<td>.........</td>
<td>22JAN2012</td>
<td>.........</td>
<td>22JAN2012</td>
<td>.........</td>
<td>15</td>
</tr>
<tr>
<td>.......</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.......</td>
</tr>
</tbody>
</table>

**Data 3:** Dataset with one record per subject with lab assessment dates

**Step-3:** Merge the dataset shown in Data 4 with the lab dataset which need to be transposed.

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>LBTEST</th>
<th>COLLDATE</th>
<th>.........</th>
<th>RESULT</th>
<th>DT1</th>
<th>.........</th>
<th>DTCN</th>
<th>DTCNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>CALCIUM</td>
<td>20JAN2012</td>
<td>.........</td>
<td>9.9</td>
<td>20JAN2012</td>
<td>.........</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>CALCIUM</td>
<td>27JAN2012</td>
<td>.........</td>
<td>9.8</td>
<td>20JAN2012</td>
<td>.........</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>.......</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.......</td>
<td></td>
</tr>
<tr>
<td>1102</td>
<td>ALT</td>
<td>22JAN2012</td>
<td>.........</td>
<td>38</td>
<td>22JAN2012</td>
<td>.........</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>.......</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.........</td>
<td>.......</td>
<td></td>
</tr>
</tbody>
</table>

**Step-4:** “N” number of variables (maximum number of distinct dates) needs to be created and grouped by initializing an ARRAY to transpose RESULT variable.

```plaintext
array rslt (&maxdtcnt);
```

Macro variable MAXDTCNT holds the value maximum number of distinct dates for a subject.

Result variables which were grouped under array RSLT will be retained till the last record for respective by group. Retain variables are initialized to missing for the new by group value to avoid misplacing the values of the new by group values with the previous by group values.

**Step-5:** Using VNAME function with ARRAY

We can overcome following issue while transposing data,

If subj 1001 has no record for CALCIUM lab test on 27JAN2012 and the next lab assessment for subj 1001 was collected on 03FEB2012 for CALCIUM, while transposing the data instead of keeping the RSLT2 value as missing lab result will get wrongly populated with 03FEB2012 result.

To avoid this, we can use VNAME function to return the correct index value to the array reference.

```plaintext
array ar1(*) dtcl-dtc&maxdtcnt.;
array ar3(*) rslt1-rslt&maxdtcnt.;
retain ar3;
if first.srt then do i= 1 to &maxdtcnt.;
  ar3(i)=.;
end;
do i=1 to &maxdtcnt.;
  if _colldt eq ar1(i) then var=vname(ar1(i));
end;
```
val=input(compress(var,"dt"),2.);
ar3(val)=result;

<table>
<thead>
<tr>
<th>SUBJID</th>
<th>LBTEST</th>
<th>RANGE</th>
<th>RSLT1</th>
<th>RSLT2</th>
<th>......</th>
<th>RSLTN</th>
<th>DT1</th>
<th>......</th>
<th>DTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>CALCIUM</td>
<td>2-10.6</td>
<td>9.9</td>
<td>9.8</td>
<td>......</td>
<td>.</td>
<td>20JAN2012</td>
<td>......</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>BASOPHILS</td>
<td>XX-XX.X</td>
<td>XX.X</td>
<td>.</td>
<td>......</td>
<td>.</td>
<td>20JAN2012</td>
<td>......</td>
<td></td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>......</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>1102</td>
<td>ALT</td>
<td>0-59</td>
<td>38</td>
<td>35</td>
<td>......</td>
<td>.</td>
<td>22JAN2012</td>
<td>......</td>
<td></td>
</tr>
</tbody>
</table>

**Step-6:** DTRET variables were created to hold the DATE values concatenated with DAY. These DTRET variables were used while creating the macro variables for respective result variables which can be used for LABELLING purpose in PROC REPORT.

**Step-7:** Reporting the restructured data by running the Proc Report in a loop by sub setting the data for each subject. The key steps involved for reporting is as follows,

I. For each time the loop iterates, DTCNT value for that respective subject defines number of variables needs to be placed in COLUMN statement for each subject. Macro variable MACVAR holds the DTCNT variable value and helps in resolving the required number of variables in the column statement as below,
   
   columns trtgroup subjid sex age race lbtest units range
   
   %do r=1 %to &macvar.;
   
   rslt&r. %end;;

II. To subset the data for this looping subject was assigned with alias.

III. In DEFINE statement macro variables created for labeling gets resolved. For this we have assigned the labels to macro variables as below and variable DMACn will get resolved for respective result column. CNT is the alias variable provided to each subject on required sorting order for easy processing,

   data _null_;
   
   set lab(where=(cnt=&i));
   
   by trtgrpn trtgroup subjid sex age race ;
   
   if first.subjid and cnt eq &i then call symputx("macvar",dtcnt);
   
   if first.subjid then do;
   
   array dtmac(*) dtret1-dtret&macvar. ;
   
   %do m=1 %to &macvar. ;
   
   call symputx('dmac'||strip(put(&m,2.)),dtret&m);
   
   %end;
   
   end;
   
   run;

**CONCLUSION**

With very large data the performance of PROC TRANSPOSE breaks down and becomes highly inefficient in which DATASTEP with ARRAY subscript processing becomes more increasingly more efficient. That potential originates in the powerful SAS DATA step language which is well designed processing tool. Like all the other SAS procedures PROC TRANSPOSE might not cover all the situations. You should choose the right method depending on your programming objective and efficiency.

Complete code is placed in Appendix 1.

**REFERENCES**

- SAS support website:
  

**CONTACT INFORMATION**

Your comments and questions are valued and encouraged. Contact the author at:

Name: Manjusha Alapati

Address: Mumbai, India
E-mail: manjualapati@gmail.com

Name: Vidyasagar Palli
Address: Mumbai, India
E-mail: vidyasagarpalli@gmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.
APPENDIX 1

data lab;
  set lab(where=(lbtest ne ""));
  RANGE = right(minnorm)||"-"||left(maxnorm);
  _COLLDT=upcase(put(colldate,date9.))||"@DAY@"||strip(put(labday,best.));
run;

proc sort data=lab out=lab_coldt(where=(colldate eq .) keep= trtgrpn trtgroup subjid lbtest colldate units range);
  by trtgrpn trtgroup subjid lbtest;
run;

/* To get the maximum number of assessment dates across all the subjects*/
proc sql noprint;
  create table datecnt as
    select count(distinct _COLLDT) as datenum,subjid
    from lab_pop
    group by subjid;
  select max(datenum) into: maxdtcnt
  from datecnt;
quit;
%let maxdtcnt=&maxdtcnt;
proc sort data=lab_pop out=rslt_sub(keep= subjid labday range sex age race trtgrpn trtgroup
  result lbtest colldate units _colldt);
  by trtgrpn trtgroup subjid colldate lbtest;
run;

data rslt_sub;
  length srt $ 200;
  set rslt_sub(in=b);
  by trtgrpn trtgroup subjid colldate;
  srt=cats(subjid,lbtest,units,range);
run;

data dates_persub;
  set rslt_sub;
  by trtgrpn trtgroup subjid colldate lbtest;
  array ar1(*) dt1-dt&maxdtcnt.; /* Numeric dates */
  array ar2(*) $20 dtc1-dtc&maxdtcnt.; /* Character dates */
  retain ar1 ar2;
  if first.subjid then do;
    dt=0;
    do i=1 to &maxdtcnt;
      ar1(i) = .;
      ar2(i) = "";
    end;
  end;
  if first.colldate then dt+1;
  ar1(dt) = colldate;
  ar2(dt) = _colldt;
run;
/* this macro helps in determining the order of records if multiple records per colldate
were present for same labtest for a subject and labseqid is missing  with the data*/
%macro duplvl;
  options nosymbolgen nomprint nomlogic;
  %global set;
  %do chk=1 %to 99;
%if &chk eq 1 %then %let dupchk=dates_persub;;

proc sort data=&dupchk. out=chk&chk. nodupkey dupout=chkdup&chk.;
   by trtgrpn trtgroup subjid lbtest units range srt colldate;
run;

proc sql noprint;
   select count(*) into:anydup
   from chkdup&chk.;
quit;

%if &anydup. eq 0 %then %do;
   %let duplvl=&chk.;;
   %let chk=99;
%end;
%else %if &anydup ne 0 %then %do;
   %let dupchk=chkdup&chk.;;
%end;
%end;
%do srt=1 %to &duplvl;

   %if &srt. eq 1 %then %let srtdup=dates_persub;;
   %if &srt. ne 1 %then %do;
      %let prev=%eval(&srt-1);
      %let srtdup=dup&prev.;;
   %end;
%end;
proc sort data=&srtdup. out=nodup&srt. nodupkey dupout=dup&srt.;;
   by trtgrpn trtgroup subjid lbtest units range srt colldate;
run;

data onelb_persub&srt.;;
   length var $ 40;
   set nodup&srt.(keep=dt1-dt&maxdtcnt. dtc1-dtc&maxdtcnt. _colldt subjid result srt subjid
      units range lbtest sex age race trtgroup colldate);
   by trtgrpn trtgroup subjid lbtest units range srt colldate;
   array ar1(*) dtc1-dtc&maxdtcnt.;
   array ar3(*) rslt1-rslt&maxdtcnt.;
   retain ar3;
   if first.srt then do i= 1 to &maxdtcnt.;;
      ar3(i)=.;
   end;
   do i=1 to &maxdtcnt.;;
      if _colldt eq ar1(i) then var=vname(ar1(i));
   end;
   val=input(compress(var,"dtc"),2.);
   ar3(val)=result;
   if last.srt then do;
      flg=&srt.;;
      output onelb_persub&srt.;;
   end;
run;
%end;

data _null_;;
   length cat $150;
   array ar1(&duplvl.) $20 onelb1-onelb&duplvl.;;
   do i=1 to &duplvl.;;
      ar1(i)=cats("onelb_persub",put(i,3.));
   end;
   cat=catx(" ",of onelb1-onelb&dupvl.);
cat=strip(cat);  
call symputx('set',cat);  
run;

data dup_obs_dt;  
set &set.;  
run;

%mend duplvl;

%mend duplvl;

proc sort data=dup_obs_dt out=dup_obs_dt_srt;  
by trtgrpn trtgroup subjid lbtest units range srt colldate flg;  
run;

data retdate_subj;  
set dup_obs_dt_srt;  
by trtgrpn trtgroup subjid sex age race;  
array dtc(*) $20 dtc1-dtc&maxdtcnt.;  
array arret(*) $20 dtret1-dtret&maxdtcnt.;  
retain arret;  
if first.subjid then do i=1 to &maxdtcnt.;  
arret(i)="";  
end;  
do i=1 to &maxdtcnt.;  
if dtc(i) ne "" then arret(i) = dtc(i);  
end;  
if last.subjid then do;  
macvar=&maxdtcnt-cmiss(of dtret1-dtret&maxdtcnt.);  
end;  
run;
data macvar_dummy(keep=macvar trtgroup subjid sex age race dtret1-dtret&maxdtcnt. dtc1-dtc&maxdtcnt.);  
set retdate_subj;  
by trtgrpn trtgroup subjid sex age race;  
if last.subjid;  
run;

data retdate_subj;  
merge retdate_subj(drop=macvar dtret1-dtret&maxdtcnt. dtc1-dtc&maxdtcnt.) macvar_dummy;  
by trtgrpn trtgroup subjid sex age race;  
run;

proc sort data=retdate_subj out=final_rep;  
by trtgrpn trtgroup subjid sex age race units range flg;  
run;

data final_rep;  
set final_rep;  
by trtgrpn trtgroup subjid sex age race units range flg;  
if first.subjid then do;  
ord_pg=0;  
cnt+1;  
end;  
ord_pg+1;  
pgval=ceil(ord_pg/24);  
run;

data l_lab;  
set final_rep;  
by trtgrpn trtgroup subjid sex age race lbtest units range dtret1-dtret&maxdtcnt. rslt1-rslt&maxdtcnt. ord_flg ord_pg;  
run;
title;  
footnote;  

title "Laboratory data";  

options nodate nonumber orientation=landscape nobyline nosymbolgen ls = 152 ps = &_ps ;  

**********Report Creation*****************************************************************;  

%macro rep(numcol=, replbtest=);   /* NUMCOL parameter is to place number of result columns per page */  

data maxlen;  
  set l_lab;  
  unlen=lengthn(units);  
  rnglen=lengthn(range);  
  lblen=lengthn(lbtest);  
  run;  

proc sql noprint;  
  select count(distinct subjid) into: maxsub  
    from l_lab;  
  select max(unlen) into:maxunlen  
    from maxlen;  
  select max(rnglen) into:maxrnglen  
    from maxlen;  
  select max(lblen) into:maxlbtstlen  
    from maxlen;  
  select max(macvar) into:maxmacvar  
    from maxlen;  
  quit;  

%let maxsub = &maxsub;     /* maximum number of subjects to run the report loop */  
%let maxunlen = &maxunlen;  
%let repunlen = %sysevalf(&maxunlen+1);    /* to provide unit length in the report */  
%let maxrnglen = &maxrnglen;  
%let reprnglen = %sysevalf(&maxrnglen+2);   /* to provide range length in the report */  
%let maxlbtstlen = &maxlbtstlen;  
%let repblen = %sysevalf(&maxlbtstlen+2);    /* to provide labtest length in the report */  
%let fix=%sysevalf(&replbtest.+&reprnglen.+&repunlen.);  

%do i=1 %to &maxsub.;  
  %if &i=1 %then %do;  
    proc printto new file="fileref.l_lab.txt";  
    run;  
  %end;  
  %else %do;  
    proc printto file="fileref.l_lab.txt";  
    run;  
  %end;  

data _null_;  
  set l_lab(where=(cnt=&i));  
  by trtgrpn trtgroup subjid sex age race;
if first.subjid and cnt eq &i then call symput("macvar","macvar");
if first.subjid then do;
array dtmac(*) dtret1-dtret&macvar.;
  %do m=1 %to &macvar;
    %let m=&m;
    call symput('dmac'||strip(put(&m,2.)),dtret&m);
  %end;
end;
r
run;

proc report data=l_lab(where=(cnt eq &i)) nowindows nocenter headline spacing=0 missing formchar(2)='-' split="@";
  columns ord_pg trtgroup subjid sex age race lbtest units range %do r=1 %to &macvar.; rslt&r. %end;
  by trtgrpn trtgroup subjid sex age race;
  define trtgroup / order order=data noprint;
  define subjid / order=data noprint;
  define sex / order=data noprint;
  define age / order=data noprint;
  define race /  order=data noprint;
  define ord_pg / order=data noprint;
  define lbtest / order=data "Date:@@Test" width=35 left flow;
  define units/ order=data "Units" width=&repunlen. left  flow;
  define range /id display "   Reference@    Range" width=&reprnglen. left flow;
  %let remain=%sysevalf(&ls.-&fix.);
  %let forcolwd=%sysevalf(&remain./&numcol.);
  %let width=%sysevalf(%sysfunc(int(&forcolwd.))); /* Width for result columns */
  %do r=1 %to &macvar;
    define rslt&r. / "&&dmac&r." width=&width. flow;
  %end;
  title3 j=l "Subject ID : #byval3   Age #byval5 years  Race: #byval6";
  title5 j=l "Treatment Group : #byval2";
compute after ord_pg;
  line @1 &ls."*-*.*";
endcomp;
compute before _page_; 
  line @1 &ls."*-*.*";
endcomp;
break after ord_pg/page;
run;
%end;
%mend;
%rep(numcol=6,repLBtest=35);
proc printto;
run;