Paper 376-2013

Horizontal Data Sorting and Insightful Reporting: A Useful SAS® Technique

Justin Jia, Canadian Imperial Bank of Commerce (CIBC), Toronto, Ontario, Canada

Amanda Lin, Bell Canada, Toronto, Ontario, Canada

ABSTRACT

Sorting and ordering of data is a fundamental skill in SAS® data analysis. Data sorting can be vertical sorting, across rows, or horizontal sorting, across columns. Compared to vertical sort, horizontal sort is used less frequently, and it requires the user to employ multiple sophisticated SAS skills such as Transpose, Rotate, Array, Macro, etc. It is also an important and useful technique for advanced data analysis and reporting in customer profiling and metrics, which can significantly enhance the format and layout of data reporting, and thus provide informative insights into data. This paper will discuss the different approaches and methods of performing horizontal sorting and presentation of SAS data, which can also expand our horizon on data manipulation and SAS programming skills.

INTRODUCTION

In the process of data manipulation and data analysis through the use of SAS programming, it is often required to sort and arrange data in a specified order: either in ascending or descending sequence. This kind of data sorting or ordering can be arranged either in a vertical direction or in a horizontal direction. Vertical sorting of data arranges the observations of a data set according to the values of one or more variables, such that the elements are ordered vertically across the observations. This method is very common and not difficult to execute in SAS, and we can use PROC SORT or PROC SQL (with ORDER BY) methods to realize the new arrangement. Although horizontal sort is not used as often as the vertical sort, we may encounter occasions that require data to be sorted across variables or columns as well. For example, in a cross-sell marketing campaign, we have customers' monthly purchase data as shown below. The arrangement of products and purchase quantities is not in a defined order, which makes it very inconvenient for us to look up or compare.

Table 1. Sample Raw Data of A Cross-Sell Campaign

Client ID	Product 1	Qty	Product 2	Qty	Product 3	Qty	Product 4	Qty
101	Printer	10	Computer	25	Games	6	TV	3
107	TV	17	Software	32	Fax Machine	11		

To do a better data reporting, for each client, we need to present the purchase data either in an alphabetical order based on the product names, or in a descending order according to the purchase quantities. This horizontal sorting and presentation of data will make the report easy to read and provide insightful information in understanding customer behaviors.

Under most circumstances, sorting and ordering of data in a horizontal way is much more complicated and challenging as compared to vertical sorting. It requires the sophisticated utilization of multiple advanced SAS skills such as TRANSPOSE, ARRAY, DO-LOOP, MACRO and other techniques. Although it is not often used, horizontal sorting is an important and useful technique in advanced data analysis and reporting. In this paper, we will use two sample projects to illustrate the different approaches and methods of sorting and reporting data horizontally.

PROJECT 1: STUDENT PROFILING PROJECT--- HORIZONTAL SORTING OF NUMERIC VARIABLES.

In our previous work, our client once asked us to create a profile on the students who participated in a series of simulation exams for psychological studies. In this project, study subjects were randomly selected from a large student database which contained the personal and academic information of undergraduate students from various universities in Canada. These study subjects were required to take a series of simulated psychological tests. Each subject could choose any of the 12 available tests; however, they had to take at least 3 tests and at most 12 tests for the purpose of statistical analysis. Below is a partial printout of the test score data, please note that some test scores may have missing values because it was not mandatory for each student to take all of the 12 tests, therefore the ones that a student skipped will have missing values accordingly.

Table 2. Raw Data of Students' Test Scores in MACCA Psychological Studies.

ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	Test_10	Test_11	Test_12
A001	82	99	100	69	34	99	94	42	42	97	.	.
A003	.	45	67	.	45	88	95	.	.	.	63	32
A004	52	.	35	66	.	77	79	.	68	.	58	79
A006	86	82	71	45	78	.	52	.	45	31	.	78
A008	50	67	59	63	58	.	70	.	53	94	.	61

Our client also required us to report the personal profile and the descriptive statistics of test scores of each participant in HTML format using their template as shown below. As such, the personal and academic information of each student was to be presented in a separate table. Please note the test scores are arranged in a descending order in the test summary item.

Figure 1. Template and Layout of the Output HTML Report as Required by Client

Personal Profile and Test Summary Report for Students Participating in 2011 MACCA Study.

ID=A019

Item	Content
Subject ID	A019
Hobby	Shopping, Basketball, Hiking.
Last_Name	Taylor
First_Name	Janet
Gender	F
Birth_Date	05/12/1987
Phone	905-602-7777
Home_Address	99 Conestoga College Blvd., Kitchener, ON, N2P 2N5
Grade	1
Department	Music
University	University of Manitoba
Test Summary	Subject ID#: A019. Test scores are: 86, 67, 67, 48, 40, 38, 32, 30 respectively. Score statistics: 51.0 +/- 20.1 (n= 8).

Funded by *Natural Sciences and Engineering Research Council* of Canada (*NSERC*), this research is conducted by MACCA Study Team of McMaster University on a voluntary participation basis.

We will illustrate and discuss the different approaches to generating this profiling report as follows.

Method 1A: Bubble Sort Approach.

```
libname P '/file path/';
data Sort_A(drop= I J temp);
set P.test;
array S(12) test_1-test_12;
do I=1 to 12;
do J=1 to 12-I;
if S(J) < S(J+1) then do; *Sort data in descending order.;
Temp = S(J);
S(J)= S(J+1);
S(J+1) = Temp;
end; end; end;
run ;
```

The Bubble Sort is one of the simplest data sorting algorithm, and it is very popular and widely used in practice. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. In brief, it works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.

Therefore our first method is to apply the bubble sort approach to sort the data by using array and do-loop in a SAS DATA step. As shown above, we define a numeric array S(12) of 12 elements to hold the values of 12 tests, and then a do-loop and IF-THEN conditional processing to compare each pair of adjacent elements. For descending sort, we do this in a way that is opposite to the ascending sort. If the larger(heavier) one of the two adjacent elements is on the right side of its neighbor, they swap places. Thus the largest(heaviest) element bubbles to the surface and at the end of each iteration it appears on the top. In this way, the test scores are sorted in a descending order, the horizontally-sorted data are shown below (partial printout):

Table 3. Partial Printout of the Generated SORT_A Data Set.

ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	Test_10	Test_11	Test_12
A001	100	99	99	97	94	82	69	42	42	34	.	.
A003	95	88	67	63	45	45	32	.	.	.	.	.
A004	79	79	77	68	66	58	52	35	.	.	.	.
A006	86	82	78	78	71	52	45	45	31	.	.	.
A008	94	70	67	63	61	59	58	53	50	.	.	.

Then we use a DATA step to calculate the descriptive statistics and create summary text for reporting uses. We put the DATA step in a SAS macro because we will utilize it repetitively in other methods as well.

```
%macro summary(method=);
data Summary_&Method(drop=Test_1-Test_12  I);
length Summary $200;
set Sort_&Method;

N=N(of test_1-test_12);
Mean= Mean(of test_1-test_12);
STD= STD(of test_1-test_12);

format Mean STD 4.1;

array Score (12) test_1-test_12;
Summary="Subject ID#: " ||ID||".  Test scores are:   ";

do I=1 to 12;
if I=1 then Summary=strip(Summary) ||put(Score(I), 3.);
else if I> 1 and Score(I) ^=. then Summary= strip(Summary) || ", " || put(Score(I),
3.) ;
else if I=12 then Summary= strip(Summary) || " respectively.  Score statistics: "
|| put(mean, 4.1)||" +/- " ||put(STD, 4.1)||' (n= '||put(N, 2.0)||').' ;
end;
run;
%mend;
%summary(method=A);
```

In the above DATA step, a new numeric array SCORE(12) of 12 elements is defined to contain the test scores which have been previously sorted in descending order. Three new variables N, MEAN, STD are created to calculate the sample size (the number of tests taken), average and standard deviation of test scores for each participating student. The long character variable SUMMARY (length= $200 as specified by LENGTH statement) is used to contain all these descriptive statistical information. We then use a do-loop to concatenate the values of ID, test scores and their statistics into a summary text. For this purpose, we need use PUT function to convert the numeric values into character values using the specified formats, and STRIP function to remove unnecessary leading and trailing blanks. After dropping unwanted variables (DROP = TEST_1-TEST_12 I), we eventually obtain a data set as shown below (partial printout):

Table 4. Partial Printout of the Generated SUMMARY_A Data Set.

ID	N	Mean	STD	Summary
A001	10	75.8	27.0	Subject ID#: A001. Test scores are:100, 99, 99, 97, 94, 82, 69, 42, 42, 34 respectively. Score statistics: 75.8 +/- 27.0 (n= 10).
A003	7	62.1	23.3	Subject ID#: A003. Test scores are: 95, 88, 67, 63, 45, 45, 32 respectively. Score statistics: 62.1 +/- 23.3 (n= 7).
A004	8	64.3	15.4	Subject ID#: A004. Test scores are: 79, 79, 77, 68, 66, 58, 52, 35 respectively. Score statistics: 64.3 +/- 15.4 (n= 8).

Then we need merge or join this test summary data set with the student information data set for creating the final profiling report. We achieved this by using PROC SQL to join the two tables together.

```
********** Join above Summary dataset with student info dataset***********;
proc sql;
create table P.profile as
select b.*, a.summary
```

```
from Summary_A(keep=ID Summary) a inner join P.students  b
on a.ID=b.ID
where b.ID is not null
order by b.ID;
quit;
```

In order to follow client's report template, we need transform the joined data set. We can do it by using PROC TRANSPOSE procedure, or using array in a DATA step.

```
************************ Transform By Proc Transpose ********************;
proc sort data=P.profile; by ID; run;

proc transpose data=P.profile out=Report(rename=(col1=Content));
by ID;
var  _ALL_;
run;
```

As shown above, using the special SAS name list _ALL_ in the PROC TRANSPOSE will transpose all variables(including the ID variable) in the input data set, the output REPORT data set is shown in the following table(partial printout). Please note that numeric values are automatically converted into character values by using the BEST12 format during this process.

Table 5. Partial Printout of the Generated REPORT Data Set.

ID	Col_Name	Content
A003	ID	A003
A003	Hobby	Sewing, Beatboxing, Socializing.
A003	Last_Name	Thompson
A003	First_Name	Judy
A003	Gender	F
A003	Birth_Date	08/06/1987
A003	Phone	647-235-0987
A003	Home_Address	16 Isabella, Saint Jacobs, ON, N0B 2N0
A003	Grade	3
A003	Department	Psychology, Neuroscience & Behaviour
A003	University	Colombia College
A003	Summary	Subject ID#: A003. Test scores are: 95, 88, 67, 63, 45, 45, 32 respectively. Score statistics: 62.1 +/- 23.3 (n= 7).

Alternatively, using DATA step array is also an important method in transforming a data set. Compared with PROC TRANSPOSE, array method is much more flexible and versatile; therefore it is widely used in SAS data manipulation. Especially when we have special needs and PROC TRANSPOSE fails to work, array is often the solution due to its flexibility. Below is the coding used in the DATA step array approach, which produces a data set identical to the one obtained by using the PROC TRANSPOSE method.

```
************************ Transform By Data Step Array ***************************;
proc sort data=P.profile; by ID; run;

data Report(keep=ID Col_Name Content);
set P.profile;
by ID;
array Col(12) $200 ID Hobby Last_Name  First_Name Gender Birth_Date Phone
                   Home_Address   Grade Department University Summary;
do I=1 to 12;
Col_Name= Vname(Col(I));
Content =Col(I);
output;
end;
run;
```

The final step is to use SAS ODS and PROC PRINT to create the final profiling report, a separate table for each student, below is the SAS coding and SAS output.

```
/********************************************** ODS Output to create HTML Report. ***************************************/
```

```
filename Report '/file path/Report.html';
proc format;
value $Col
  'ID'='Subject ID'     'hobby'='Hobby'    'Summary'='Test Summary';
run;

ODS listing close;  ODS HTML file=Report style= sasweb;
Title "Personal Profile and Test Summary Report for Students Participating in 2011
MACCA Study." ;
Footnote "Funded by Natural Sciences and Engineering Research Council of Canada
(NSERC), this research is conducted by MACCA Study Team of McMaster University on a
voluntary participation basis." ;

proc print data= Report noobs label;
by ID;
label Col_Name="Item";  format Col_Name $Col.;
run;
ODS HTML close;  ODS listing;
```

Method 1B: Rotate and Transpose Approach.

In addition to Bubble Sort approach presented in Method 1A, we can use rotate and transpose approach to achieve the same goal.

```
****************** Method 1B: Rotate and Transpose Approach. *************;
proc sort data=P.test;  by ID; run;
data B_rotate(drop= I test_1-test_12);
set P.test;
by ID;
array S(12) test_1-test_12;
if first.ID then do I=1 to 12;
Score=S(I);
if not missing(score) then output;
end;
run;
proc sort data= B_rotate;  by ID descending score; run;
proc transpose data= B_rotate out=Sort_B(drop=_name_)  prefix=test_ ;
by ID ;
var score;
run;
%summary(method=B);
```

By using array again, the above program will rotate the original TEST data set into vertical form: the test scores of each ID will transform from values of 12 columns into values of 12 observations of a single numeric variable SCORE. Please note that we only keep the observations with non-missing values.

Following data set rotation, we first use PROC SORT to sort data set observations in descending order based on the value of SCORE, and then employ PROC TRANSPOSE to transpose the sorted dataset back to the original horizontal form. The generated data set SORT_B is identical to data set SORT_A obtained in Method 1A. We then use the same MACRO %SUMMARY to calculate the required descriptive statistics of test scores and create the standardized summary text. Next, by following the same Join/Rotate/ODS steps as shown in Method 1A, we can create the same profiling report as produced in Method 1A.

This method is relatively easier in comparison to Method 1A, and it is applicable to the sorting of both numeric and character variables. However, it requires a good understanding and utilization of rotate and transpose techniques.

Method 1C: Make Use of new LARGEST function.

The third method illustrated here is to use the new LARGEST function introduced into SAS 9.0 and later versions. It may be not well known and widely used in SAS programming, but it is very useful to meet some special needs.

We know that the MAX and MIN functions allow us to easily identify the maximum and minimum values from a selection of variables, however, sometimes we also need to find the second highest value (or lowest) or the k^{th} highest(or lowest) value of a list of variables. In these cases, we can make use of the new SAS 9 LARGEST function to do it easily.

As stated in SAS Support website, the basic syntax[1] is:

"Syntax: LARGEST (k, value-1<, value-2 ...>)

Arguments: k is a numeric constant, variable, or expression that specifies which value to return.

Value specifies the value of a numeric constant, variable, or expression to be processed.

This function returns the k^{th} largest non-missing value. If k is missing, less than zero, or greater than the number of values, the result is a missing value and _ERROR_ is set to 1. Otherwise, if k is greater than the number of non-missing values, the result is a missing value but _ERROR_ is not set to 1."

Below is the SAS coding to do the required sort by using the LARGEST function, which will sort and produce the same data set as in Methods 1A and 1B.

```
****************** Method 1C: Sort by Use of Largest function. ******************;
data Sort_C(drop= T1-T12 J);
set P.test(rename=(Test_1=T1 Test_2=T2 Test_3=T3 Test_4=T4
                   Test_5=T5 Test_6=T6 Test_7=T7 Test_8=T8
                   Test_9=T9 Test_10=T10 Test_11=T11 Test_12=T12)) ;
array Test(12) Test_1 - Test_12;
do J=1 to 12;
Test(J)=largest(J, of T1-T12);
end;
run;
%summary(method=C);
```

As indicated above, using array and the LARGEST function provides a better solution to our needs: the SAS coding is much more concise and straightforward than that in Method 1A and Method 1B.

Similarly, the SMALLEST function is used to return the k^{th} smallest non-missing value. Actually, we can also employ SMALLEST function to perform the descending sort, as long as we apply it in the reverse way:

```
data Sort_C(drop= T1-T12 J);
set P.test(rename=(Test_1=T1 Test_2=T2 Test_3=T3 Test_4=T4
                   Test_5=T5 Test_6=T6 Test_7=T7 Test_8=T8
                   Test_9=T9 Test_10=T10 Test_11=T11 Test_12=T12)) ;
array Test(12) Test_1 - Test_12;
do J=1 to 12;
Test(J)= smallest(13-J, of T1-T12);
end;
run;
```

Thus, we can see that these two new SAS functions provide more alternative solutions to our needs and can be very useful in SAS data analysis.

Method 1D: Make Use of the new SAS feature --- CALL SORTN routine .

Starting from SAS9.2, two very useful new call routines come into use: CALL SORTC and CALL SORTN. They are useful in sorting a small number of values within a running DATA step. The basic syntax[2] is:

CALL SORTC (variable-1<, ..., variable-n>) ;

CALL SORTN (variable-1<, ..., variable-n>) ;

They can be used to sort the values of a list of variables passed to it, for example, the variables specified in the arguments, or the elements of a defined array[3]. CALL SORTN is applied for sorting numeric values, while CALL SORTC for character values (all character variables must be of the same length when using CALL SORTC). Please note that "the sorts by these two new routines are always done in ascending sequence, however, sorts in descending sequence can be effectively achieved by specifying the variables in a reverse order[3]". This is a very helpful trick in using them.

```
********************** Method 1D: Use Call SortN routine. ********************;
data Sort_D;  set P.test;
call sortN(Test_12, Test_11, Test_10, Test_9, Test_8, Test_7, Test_6, Test_5,
Test_4, Test_3, Test_2, Test_1);
run;
```

```
%summary(method=D);

data Sort_E;
set P.test;
array Test(12)  Test_12- Test_1;
call sortN(of Test(*));
run;
%summary(method=E);
```

As shown above, CALL SORTN routine is applied to sort the scores of 12 tests in a running DATA step. By default, CALL SORTN will sort the values of the 12 tests into an ascending sequence, therefore the required descending sort is realized by arranging the 12 variables in a reverse order, from TEST_12 to TEST_1.

The above DATA step produced an identical data set as those in Methods 1A-1C, however, this ingenious approach by using the new CALL SORT routines provides the simplest method to achieve our goal with the most concise SAS coding: only one DATA step is needed to do the horizontal sorting, even eliminating the use of array and do-loop. However, combining using an array with CALL SORTN can produce the same result with much less typing.

PROJECT 2: MONTHLY PURCHASE DATA REPORTING--- HORIZONTAL SORTING OF BOTH NUMERIC AND CHARACTER VARIABLES.

In this project, the raw cross-sell purchase data as shown below is received in no order and therefore it is very difficult to read and look up. To produce a better report, we can: 1) arrange the purchase data of each client in the alphabetical sequence of product names, which will make the information easier to read through or look up for a specific product; 2) or, present data in the descending sequence of purchased quantities, which will provide insightful information into customers' purchase tendency and preferences. This is definitely a much better data presentation strategy for creating useful customer profiling and customer marketing pictures.

Table 6. Sample Raw Data of A Cross-Sell Campaign

Client ID	Product 1	Qty	Product 2	Qty	Product 3	Qty	Product 4	Qty	Product 5	Qty	Product 6	Qty
101	Printer	10	Computer	25	Games	6	TV	3	Software	8		
107	TV	17	Software	32	Fax Machine	11						
216	Books	53	Clothes	16	Glass Ware	24	Phones	9	TV	2	DVD	38

This kind of horizontal sort and presentation of data is much more challenging than those discussed in Project 1, because they involve two different kinds of variables. The product name is character in nature, while the purchase quantity is numeric, and even more important, they are relevant to each other rather than independent of each other: since each purchase quantity corresponds to a specific product, we can NOT sort and arrange them only on the basis of one variable, otherwise the purchase data will be messed up totally. Therefore, it is a remaining challenge.

We hereby present and illustrate three different creative methods to achieve this goal by using Bubble Bust, Rotate and Transpose, and Call Sort routines. Although we use only two kinds of variables (product names and purchase quantities) to demonstrate for simplicity, however, all the methods are generalizible and applicable to multiple variable cases.

Method 2A: Bubble Bust Approach.

Similar to the discussion in Method 1A, we can apply the classic Bubble Bust algorithm to horizontally sort the purchase data again.

```
************ Horizontal Sort in the Ascending Order of Product Name************;
data report_By_product(drop=I  J  temp_P temp_Q )    ;
set  P.purchase;
array P(6)  $30  Product_1- Product_6;
array Q(6)  Qty_1-Qty_6;


do I=1 to 6;
do J=1 to 6-I;
if not missing(P(J+1)) and P(J) > P(J+1) then do;
*sort product name in ascending order.;
Temp_P = P(J);
P(J)= P(J+1);
P(J+1) = Temp_P;
```

```
Temp_Q=Q(J);
Q(J)= Q(J+1);
Q(J+1) = Temp_Q;
end; end; end;
run;

***********Horizontal Sort in the Descending Order of Purchase Quantity*****;
data report_By_quantity(drop=I  J  temp_P temp_Q ) ;
set  P.purchase;
array P(6)  $30  Product_1- Product_6;
array Q(6)  Qty_1-Qty_6;
do I=1 to 6;
do J=1 to 6-I;
if not missing(P(J+1)) and Q(J) < Q(J+1) then do;
*sort purchase quantity in descending order;

Temp_Q=Q(J);
Q(J)= Q(J+1);
Q(J+1) = Temp_Q;

Temp_P = P(J);
P(J)= P(J+1);
P(J+1) = Temp_P;
end; end; end;
run;

********************** ODS Output to create MS Excel report. ********************;
filename Purchase '/file path/P2_Report.xls';
ODS listing close; ODS MSOFFICE2K file=Purchase style= journal;
Title "Overview of Monthly Customer Purchases." ;

proc print data= report_by_product  noobs label;
var Client_ID product_1  qty_1  product_2  qty_2  product_3  qty_3
       product_4  qty_4  product_5  qty_5  product_6  qty_6  ;
label      Product_1 ='Product 1'      Qty_1 = 'Purchase Qty'
           Product_2 = 'Product 2'     Qty_2 = 'Purchase Qty'
           Product_3 = 'Product 3'     Qty_3 = 'Purchase Qty'
           Product_4 = 'Product 4'     Qty_4 = 'Purchase Qty'
           Product_5 = 'Product 5'     Qty_5 = 'Purchase Qty'
           Product_6 = 'Product 6'     Qty_6 = 'Purchase Qty';
run;

proc print data= report_by_quantity  noobs label;
var Client_ID product_1  qty_1 product_2  qty_2 product_3  qty_3
    product_4  qty_4  product_5  qty_5 product_6  qty_6  ;
label      Product_1 ='Product 1'      Qty_1 = 'Purchase Qty'
           Product_2 = 'Product 2'     Qty_2 = 'Purchase Qty'
           Product_3 = 'Product 3'     Qty_3 = 'Purchase Qty'
           Product_4 = 'Product 4'     Qty_4 = 'Purchase Qty'
           Product_5 = 'Product 5'     Qty_5 = 'Purchase Qty'
           Product_6 = 'Product 6'     Qty_6 = 'Purchase Qty';
run;
ODS MSOFFICE2K close;
ODS listing;
```

As illustrated above, a character array P(6) of 6 elements(length=$30) is defined to hold the values of product names, and a numeric array Q(6) of 6 elements to hold the values of purchase quantities. Then a do-loop is performed to sort the data in an ascending sequence of product names by using Bubble Bust approach. Please note the IF-THEN conditional processing is based on the comparison of product names(character) of each pair of adjacent elements. If the smaller one is at the right side of the pair, and the right one is not null, then they exchange values with each other, at the mean time, the values of purchase quantities swap places accordingly. With the execution of all the do-loop iterations, the data set will eventually get sorted in the ascending order of product names.

Similarly, if we want to sort the data in a descending order of purchase quantities, we apply the Bubble Bust approach in an opposite way. We perform the IF-THEN conditional processing according to the comparison of purchase quantities(numeric) of each pair of adjacent elements. If the larger one is at the right side of the pair, and the right one is not null, then they exchange values with each other, at the mean time, the values of product names swap places accordingly. Consequently, the data set will be sorted in the descending order of purchase quantities.

Then we use SAS ODS facility and PROC PRINT to output the sorted data to generate the Microsoft Excel report. The ordering of columns in the Excel report is achieved by arranging them in the desired order in the VAR statement in PROC PRINT procedure. Also, application of meaningful labels can enhance the Excel report greatly. Below is the printout of the generated Excel report.

Table 7. Overview of Monthly Customer Purchases: Order By Ascending Product Name.

Client_ID	Product 1	Purchase Qty	Product 2	Purchase Qty	Product 3	Purchase Qty	Product 4	Purchase Qty	Product 5	Purchase Qty	Product 6	Purchase Qty
101	Computer	25	Games	6	Printer	10	Software	8	TV	3		.
107	Fax Machine	11	Software	32	TV	17		.		.		.
216	Books	53	Clothes	16	DVD	38	Glass Ware	24	Phones	9	TV	2

Table 8. Overview of Monthly Customer Purchases: Order By Descending Purchase Quantity.

Client_ID	Product 1	Purchase Qty	Product 2	Purchase Qty	Product 3	Purchase Qty	Product 4	Purchase Qty	Product 5	Purchase Qty	Product 6	Purchase Qty
101	Computer	25	Printer	10	Software	8	Games	6	TV	3		.
107	Software	32	TV	17	Fax Machine	11		.		.		.
216	Books	53	DVD	38	Glass Ware	24	Clothes	16	Phones	9	TV	2

Method 2B: Rotate and Transpose Approach.

In this method, the original data set is rotated into vertical form by using the character array P(6) for product names and the numeric array Q(6) for purchase quantities. The rotated values of product names and purchase quantities are assigned to two new variables PRODUCT and QUANTITY respectively.

Following data set rotation, we first use PROC SORT to sort the transformed data across observations by CLIENT_ID and PRODUCT, then apply array in DATA step to transpose the sorted data set back to horizontal form. RETAIN statement is necessary to retain the values across data step iterations, and a counter variable CNT is created by SUM statement to ensure the correct assignments of variable values into corresponding elements in arrays. These data manipulations will produce a final data set sorted in the ascending order of product names. In a similar way, we can also sort it horizontally by descending purchase quantity. Below presents the SAS codes for achieving the goals.

```
**********Horizontal Sort in the Ascending Order of Product Name**********;
data Rotate(keep=client_ID product quantity);
set P.purchase;
array P(6)  $30  product_1 - product_6;
array Q(6)  Qty_1 - Qty_6;
do I=1 to 6;
product=P(I);
quantity=Q(I);
if not missing(product) then output;
end;
run;

proc sort data= Rotate; by Client_ID product; run;

data report_by_product(drop=I CNT product quantity);
set Rotate;
by Client_ID product;
array P(6)  $30  product_1 - product_6;
array Q(6)  Qty_1 - Qty_6;
retain product_1 - product_6  Qty_1 - Qty_6 ;

if first.Client_ID then do I=1 to 6;
CNT=0;
call missing(P(I), Q(I));
end;
```

```
CNT+1;
P(CNT)=product;
Q(CNT)=quantity;

if last.Client_ID then output;
run;

***********Horizontal Sort in the Descending Order of Purchase Quantity***********;
proc sort data= Rotate; by Client_ID descending quantity; run;

data report_by_quantity(drop=I CNT product quantity);
set Rotate;
by Client_ID descending quantity;
array P(6)  $30  product_1 - product_6;
array Q(6)  Qty_1 - Qty_6;
retain product_1 - product_6  Qty_1 - Qty_6 ;

if first.Client_ID then do I=1 to 6;
CNT=0;
call missing(P(I), Q(I));
end;

CNT+1;
P(CNT)=product;
Q(CNT)=quantity;

if last.Client_ID then output;
run;
```

Please note that, for this transpose requirement, the PROC TRANSPOSE procedure will NOT work due to its limitations. For this reason, we must use flexible data step array to realize it. The two final data sets (REPORT_BY_PRODUCT and REPORT_BY_QUANTITY) obtained by above rotate and transpose approaches are identical to those gained in Method 2A. The next step is to use the same ODS program to create the identical Excel report, which we will not repeat here.

Method 2C: Make Use of Call SortN/SortC routines.

In addition to approaches discussed in Methods 2A and 2B, we can also utilize the new SAS features CALL SORTN/SORTC call routines to perform our task. Although this method is much trickier and more complicated compared to the straightforward Methods 2A and 2B, it can remarkably expand our horizon and skills in SAS programming, therefore we illustrate and discuss it in details as well.

```
*************** Method 2C: Make Use of Call SortC/SortN Routines. ***************;
proc sort data=P.purchase; by Client_ID; run;

data concatenation(drop=I Product_1- Product_6 Qty_1-Qty_6);
set P.purchase;
by Client_ID;
array P(6)  $30  Product_1- Product_6;
array Q(6)       Qty_1-Qty_6;
array sort_By_prod(6) $60  P1-P6;
array sort_By_Qty(6)  $60  Q1-Q6;

do I=1 to 6;
if not missing(P(I)) then do;
sort_By_prod(I)= strip(P(I))||"/"||put(Q(I), Z4.);
sort_By_Qty(I) = put(Q(I), Z4.)||"/"|| strip(P(I));
end; end;

call sortC(of P1-P6); *sort by product name in ascending order;
call sortC(of Q6-Q1); *sort by purchase quantity in descending order;
run;
```

In this method, we can NOT use CALL SORTC or CALL SORTN routine directly because it can only sort a variable independently each time when we recall it. However, each purchase quantity is relevantly corresponding to a specific product, it does not exist independently. After sorting the data by product name with CALL SORTC, the sequence and position of array elements will change and differ from the original ones. However, as obvious as it is, we must match the purchase quantity to the specific product that it belongs to, otherwise the purchased data will be messed up and become useless. This is a big challenge indeed.

One way to accomplish this is to use the text concatenation approach. The creative idea is to concatenate the product name together with the purchase quantity (Note: purchase quantity must be converted from numeric into character values in advance).

If we want to sort the data by product name, as shown below, we need to concatenate product name in ahead of the converted purchase quantity. Then we sort the concatenated text in ascending order by using CALL SORTC routine, which is equivalent to sort data by product name. After that we can then use SCAN function to retrieve the product name from purchase quantity, and INPUT function to convert the purchase quantity back to numeric values. In this way, the purchase quantity always moves together with the product name. Thus the product name and purchase quantity will match correspondingly to each other in the output data set.

Table 9. Sort Data By Product Name: Concatenating Product Name In Ahead of Purchase Quantity.

Product	Qty	Concatenated Text	Product	Qty	Concatenated Text	Product	Qty	Concatenated Text
Games	6	Games/0006	Printer	10	Printer/0010	Computer	25	Computer/0025

Similarly, if we want to sort the data by purchase quantity in descending order, we must concatenate the converted purchase quantity text in ahead of product name, then use CALL SORTC to sort the concatenated text in descending order, followed by SCAN, INPUT functions to separate purchase quantity from product name, and convert the quantity into numeric values. Consequently, it leads to the desired sort result.

Table 10. Sort Data By Purchase Quantity: Concatenating Purchase Quantity In Ahead of Product Name.

Product	Qty	Concatenated Text	Product	Qty	Concatenated Text	Product	Qty	Concatenated Text
Games	6	0006/Games	Printer	10	0010/Printer	Computer	25	0025/Computer

As shown in the above SAS codes, the first step is to concatenate the corresponding product name and purchase quantity together. Four arrays P(6) Q(6) SORT_BY_PROD(6) SORT_BY_QTY(6) are therefore defined to hold the original values and concatenated texts respectively. Then a do-loop is used to do the concatenation, STRIP function is to remove leading and trailing blanks, and PUT function for numeric-character conversion with Z4. format. Please note that Zn. format is a new feature in SAS 9, which will pad character text converted from numeric values with leading zeros. Using this format is better than other ones because the padded leading zeros are more straightforward than invisible leading blanks. After concatenation, the CALL SORTC routine is recalled to sort the text either by ascending product name, or by descending purchase quantity. Below is the printout of the generated CONCATENATION data set.

Table 11. Partial Printout of the Generated CONCATENATION Data Set.

Client_ID	P1	P2	P3	P4	P5	P6	Q1	Q2	Q3	Q4	Q5	Q6
101		Computer/0025	Games/0006	Printer/0010	Software/0008	TV/0003	0025/Computer	0010/Printer	0008/Software	0006/Games	0003/TV	
107				Fax Machine/0011	Software/0032	TV/0017	0032/Software	0017/TV	0011/Fax Machine			
216	Books/0053	Clothes/0016	DVD/0038	Glass Ware/0024	Phones/0009	TV/0002	0053/Books	0038/DVD	0024/Glass Ware	0016/Clothes	0009/Phones	0002/TV

The next step is to use SCAN and INPUT function to separate product name from purchase quantity and convert purchase quantity into numeric values. Below codes show how to sort data in the descending order of purchase quantity. Please note slash "/" is defined as a delimiter in the SCAN function.

```
/******************Sort by Purchase Quantity: descending order.******************/
data report_By_quantity(drop=I  P1-P6 Q1-Q6);
set concatenation;
array P(6)  $30  Product_1- Product_6;
array Q(6)  Qty_1-Qty_6;
array sort_By_prod(6) $60  P1-P6;
array sort_By_Qty(6)  $60  Q1-Q6;
do I=1 to 6;
P(I)=scan(sort_By_qty(I),  2, "/");
Q(I)=input(scan(sort_By_qty(I),  1, "/"),  4.);
end;
run;
```

However, if we want to sort the data in an ascending order of product name, it requires additional work due to the missing values of some products. Because missing values are always regarded as the smallest values, thus the columns with missing values will appear in ahead of columns with non-missing values in the defined array SORT_BY_PROD(6). The ordering of columns actually appears like:

Table 12. Structure of Data Before Left Alignment.

Client_ID	P1	P2	P3	P4	P5	P6	Q1	Q2	Q3	Q4	Q5	Q6
107				Fax Machine	Software	TV	.	.	.	11	32	17

Therefore, we need move the non-blank values of P4, P5, P6 forward into P1, P2, P3 respectively for left alignment, and same thing for Q4-Q6 into Q1-Q3 respectively.

```
/*********************Sort by Product Name: ascending order.**********************/
data report_By_product(drop=I  J M P1-P6 Q1-Q6 T1-T6 );
set concatenation;
array P(6)  $30  Product_1- Product_6;
array Q(6)  Qty_1-Qty_6;
array sort_By_prod(6) $60  P1-P6;
array sort_By_Qty(6)  $60  Q1-Q6;
array Temp_Prod(6)    $30  T1-T6;
array Temp_Qty(6)     _temporary_;

do I=1 to 6;
Temp_Prod(I)=scan(sort_By_prod(I),  1, "/");
Temp_Qty(I)=input(scan(sort_By_prod(I),  2, "/"),  4.);
end;
M= Cmiss(of T1-T6);
do J=1 to 6-M;
P(J)=Temp_prod(J+M);
Q(J)=Temp_Qty(J+M);
end;
run;
```

As illustrated above, to beat this challenge, the great idea is to define two more temporary arrays TEMP_PROD(6) $30 and TEMP_QTY(6) to hold the sorted and separated values of product names and purchase quantities respectively, which have been accomplished by the previous Concatenation DATA step. Then we use CMISS function and one more do-loop to realize it. The CMISS function can count how many missing values present in each observation, and we already know that the columns with missing values always appear ahead of columns with non-missing values because the dataset is already horizontally sorted in the ascending sequence of product name. Therefore we can use the last part of codes(the bolded part) to “move” the columns with non-missing values to the left, the final result is shown as follows, which is exactly what we pursue toward.

Table 13. Structure of Data After Left Alignment.

Client_ID	P1	P2	P3	P4	P5	P6	Q1	Q2	Q3	Q4	Q5	Q6
107	Fax Machine	Software	TV				11	32	17	.	.	.

Therefore, through above approaches, we can sort the purchase data either by product name or by purchase quantity. This method generates the same datasets(REPORT_BY_PRODUCT and REPORT_BY_QUANTITY) as those produced by Method 2A and 2B, then we can use the ODS program to create the identical Excel report.

BUSINESS APPLICATION EXAMPLE: RFM ANALYSIS OF RETAIL SALES DATA.

The methods and approaches illustrated above can have important application and uses in business analytics and business intelligence fields. It can greatly help and improve our work in data analysis, data reporting, customer profiling and database marketing and so on.

For example, it is very useful for creating insightful report for RFM analysis. RFM represents Recency (How recently did the customer purchase?), Frequency (How often do they purchase?), and Monetary Value (How much do they spend?) respectively. RFM analysis can give in-depth insights into customers’ business behavior and attitudes, and thus provide valuable information for designing strategies for customer marketing. Below example reveals its vital application. As shown in Appendix, the RFM data set contains the online purchase data of a retail business for each client at a given time period: Category_1-Category_5, Freq_1-Freq_5, Sales_1-Sales5, Last_Date_1-Last_Date_5 are the product category, frequency, monetary value and last order date of customers’ online purchases, respectively.

Unfortunately, these raw data are in no order at all and thus is of little use for business intelligence and customer marketing. Therefore we need rearrange and present them in the descending sequence of either frequency or recency or monetary value, which will then provide valuable insights into customer purchase behaviors and preferences.

To meet this request, we can use any one of the 3 approaches illustrated above to perform the horizontal sort and reporting. Here we just show Bubble Bust Method as an example to accomplish it.

```
data RFM_Report;
set  P.RFM;
array C(5)  $30  Category_1- Category_5;
array R(5)  Last_Date_1- Last_Date_5;
array F(5)  Freq_1-Freq_5;
array M(5)  Sales_1-Sales_5;
do I=1 to 5;
do J=1 to 5-I;

if not missing(C(J+1)) and F(J) < F(J+1) then do; *sort data in the descending
order of purchase frequency.;
/*if not missing(C(J+1)) and M(J) < M(J+1) then do;*sort order frequency in
descending order;*/

Temp_C=C(J);
C(J)= C(J+1);
C(J+1) = Temp_C;

Temp_R=R(J);
R(J)= R(J+1);
R(J+1) = R_R;

Temp_F = F(J);
F(J)= F(J+1);
F(J+1) = Temp_F;

Temp_M = M(J);
M(J)= M(J+1);
M(J+1) = Temp_M;
end;  end;  end;
run;

proc print data=RFM_Report(keep=Client_ID category_1-category_3 Freq_1-Freq_3)
noobs label;
label      Category_1    =  "Most Frequent Purchase Category"
Category_2 =  "2nd Most Frequent Purchase Category"
Category_3 =  "3rd Most Frequent Purchase Category"
Freq_1="Purchase Freq." Freq_2 ="Purchase Freq."  Freq_3="Purchase Freq." ;
run;
```

The above program will sort the data in the descending order of purchase frequency, below is the PROC PRINT output, please note that we only keep the top 3 most frequent purchases. Similarly, we can sort the data in the descending order of purchase value or last purchase date just by changing the IF---THEN conditional processing(the commented part) and PROC PRINT labels accordingly, and then we can create insightful customer purchase reports on the basis of Monetary Value or Recency. Below tables showcase the RFM reports based on the Frequency and Monetary Value of online purchases.

As we can obviously see from the above tables, these horizontally-sorted RFM analysis reports are very informative and helpful in understanding customer behaviors. Therefore, it is very useful in business analytics, reporting and profiling of customer data.

Table 14. RFM Analysis of Online Purchases: Purchase Frequency (in descending order).

Client ID	Most Frequent Purchase Category	Purchase Freq.	2nd Most Frequent Purchase Category	Purchase Freq.	3rd Most Frequent Purchase Category	Purchase Freq.
101	Patio & Garden	25	Auto Accessories	19	Software	17
105	Clothes	26	Office Equipments	22	Jewelry	10
117	Sports	49	Computers	14	Office Equipments	11
126	Clothes	16	Jewelry	9	Computers	7
139	Exercise & Fitness	23	Patio & Garden	20	Appliances	18
203	Household Essentials	35	Exercise & Fitness	22	Jewelry	15

Table 15. RFM Analysis of Online Purchases: Purchase Value (in descending order).

Client ID	Most Monetary Purchase Category	Purchase Value($)	2nd Most Monetary Purchase Category	Purchase Value($)	3rd Most Monetary Purchase Category	Purchase Value($)
101	Patio & Garden	$2,752	Auto Accessories	$2,483	Household Essentials	$2,387
105	Clothes	$1,884	Jewelry	$1,464	Office Equipments	$1,276
117	Sports	$3,568	Auto Accessories	$1,297	Office Equipments	$932
126	Computers	$2,524	Clothes	$2,404	Jewelry	$1,938
139	Appliances	$2,838	Exercise & Fitness	$2,726	Sports	$2,360
203	Household Essentials	$2,774	Clothes	$2,377	Exercise & Fitness	$1,550

CONCLUSION

As discussed in this paper, horizontal data sorting and insightful reporting is an important and challenging technique in advanced data analysis and manipulation through SAS programming. Utilization of this technique can significantly improve our work in customer and business data analytics and reporting, customer profiling and database marketing, and help to present valuable information and provide competitive insights into customers and businesses. Therefore, it can have important applications in a wide variety of business analytics and business intelligence fields.

REFERENCES:

[1] SAS Support Website,
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a002154862.htm

[2] SAS Support Website,
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a003106052.htm

[3] Phil Mason, “The Most Useful New Parts of SAS® 9”, SAS Forum Conference Proceedings, NESUG 2006.

CONTACT INFORMATION

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

Justin Jia
CISR, Customer Marketing
CIBC Canada
Email: justin.jia@cibc.com

Amanda Lin
Credit Risk Management
Bell Canada
Email: amanda_shan_shan.lin@bell.ca