Paper 056-30

Customer Segmentation by Using CALL SYMPUT, Arrays and DO Loops

Rodger Zhang, TD Canada Trust, Toronto, Ontario, Canada

ABSTRACT

Customer grouping or segmentation is a common task in the financial sector. Some customers might have only one product while others might have multiple products. Identifying customers with correct products will help you improve your business in areas such as cross selling. *SAS*® offers many ways not only to customize your financial reports but also to effectively put your customer in the right position. This paper will focus on how to check the maximum number of products one customer can have and how to apply that information to your reporting by using the CALL SYMPUT statement. You can use the CALL SYMPUT statement to create a macro variable that is assigned the value of a variable from a data set and transpose the data with efficient and powerful arrays and the associated DO loops facility. Base SAS is used and the examples shown are sample data only. This paper is for the beginner or intermediate SAS programmer.

INTRODUCTION

Customer data in the financial sector are usually stored at two levels, customer level and product level. A customer might have one or more products, such as Mortgage, Line of Credit, Deposit, etc. It is becoming more popular to summarize the product data at customer level. You are likely to see increasing need to understand how to manipulate such data.

This paper presents an efficient method to handle the data containing customers with unknown number of products by using the CALL SYMPUT statement, arrays and the associated DO loops facility. An example is used to illustrate the method that includes the following steps.

- Identify the type of data
- Find the maximum number of products a customer might have and apply that information to the reporting
- Transpose data horizontally
- Summarize products at customer level

METHOD

SAMPLE DATA

The following data include customer number, branch number, account number, product name and account open date.

```
data rawdata;
  infile datalines;
  input cxno branchno accountno product: $15. opendate;
datalines;
```

1234567 1234 1234567 Credit_1 20000810
1234567 1234 1234568 Credit_A 20011012
1234567 0096 5246890 Credit_B 19961201
1234567 2583 3348560 Credit_E 20030120
1234567 2583 3348500 Credit_8 19980321
1234567 1234 4234567 Deposit_1 20040821
2234568 2234 2234567 Credit_1 20011114
3344569 3334 3344567 Deposit_A 20040220
4455660 4455 4567890 Deposit_1 19871230
4455660 4455 5578920 Deposit_A 19871230
1234567 1234 1234577 Credit_2 20010105
1234567 1234 1234588 Credit_G 20011220
1234567 0096 5246800 Credit_6 20030103
1234567 2583 3348590 Credit_F 20031123
1234567 2583 3348520 Credit_H 20001024
1234567 1234 4234599 Deposit_B 19990201
2234568 2234 2234577 Credit_A 20030501
3344569 3334 3344597 Deposit_B 20040521

```
3344569 3334 5544597 Credit_B 20040521
4455660 4455 4567899 Deposit_9 19870201
4455660 4455 5578922 Deposit_B 20040201
4455660 4455 9278922 Credit_1 20040201
;
run;
```

Since you create the data set RAWDATA by reading the instream data, normally you don't need the INFILE statement. It won't hurt if you use INFILE statement here, but you have to add the DATALINES after it to tell SAS that the raw data is internal just following the DATALINES; statement. When the internal file contains non-standard format such as comma separated values, option DSD has to be used, then the INFILE statement is mandatory. For reading external raw data, what you need to do is to just change the DATALINES after keyword INFILE to the path the raw data are located with single quotation marks, or double quotation marks if macro variables are included in the path. In list input, the default length of a variable is 8, so if character variable's length is more than 8, a colon : has to be added after the variable name followed by the informat of the variable, or the value will be truncated after the 8th character.

Only two categories of products that are **CREDIT** and **DEPOSIT** are involved here. Each category has two types of sub-products, namely, **CREDIT** includes 10 *NORMAL LOANS* with the name ending with alphabetic letter from A to J and 10 *MORTGAGE LOANS* with the name ending with numbers from 1 to 10 while in the same naming convention, **DEPOSIT** includes 10 *TERM DEPOSIT* products with the name ending with alphabetic letter from A to J and 10 *DEMAND DEPOSIT* products with the name ending with numbers from 1 to 10. Just for demonstration purposes, the product names as specified above are used. Actually the real product names can be entirely different from each other, that's why the full product names are listed in the codes later on.

Now a campaign targeting at the existing customers is going to be launched. A letter will be sent to the customers to promote the current products (normal loan, mortgage loan, demand deposit or term deposit) if the customer doesn't have that particular product right now. This means you are going to correctly identify the customers with correct products so that the letter will be properly sent.

FIND MAXIMUM NUMBER OF PRODUCTS

In order to define the arrays used in the next step, it is necessary to find the maximum number of products a customer can have in the population. What you need to do first is to count the number of products for each customer.

```
proc sort;
  by cxno;
run;
data checkacct;
  set rawdata;
  by cxno;
  retain maxnum;
  if first.cxno then maxnum=0;
  maxnum+1;
  if last.cxno;
  keep cxno maxnum;
run;
```

Then you sort the data set in ascending order so that you know that the last customer (observation) will be the one with highest number as following:

```
proc sort;
  by maxnum;
run;
data _null_;
  set checkacct end=last;
  if last;
  call symput('mxnum',left(maxnum));
run;
```

Alternatively, you can sort the data set in descending order but then the customer with the highest number will become the first observation as following:

```
proc sort data=checkacct;
```

```
by descending maxnum;
run;
data _null_;
  set checkacct;
  if _n_ eq 1;
  call symput('mxnum',left(maxnum));
run;
```

The following points need to be noticed from codes above:

1. RETAIN Statement: it causes a variable to retain its value from one iteration of the DATA step to the next. It takes the form

```
RETAIN variable(s) initial value
```

This statement can only appear in DATA step, but it can appear anywhere in the DATA step. All variables before the initial value will start the first iteration of the DATA step with that value. If no initial value is specified, by default 0 will be taken as the initial value. With the RETAIN statement, it will continue to be the initialized value until it is replaced by another different value.

2. SAS automatic variables: Some variables are generated temporarily during the SAS DATA step. Among these variables, *FIRST.variable, LAST.variable* and _N_ are most frequently used variables. _N_ indicates the number of times SAS DATA step has executed. When a BY statement in a DATA step is used, FIRST.variable and LAST.variable will be automatically generated. The FIRST.variable will have a value of 1 when SAS is processing an observation with the first occurrence of a new value for that variable and a value of 0 for the other observations. For the LAST.variable's value, it will be 1 when SAS is processing an observation with the last occurrence of a value for that variable and 0 for the other observation. *FIRST.variable* and *LAST.variable* always appear in pairs, one pair for each BY variable on a BY statement.

3. SUM statement: A SUM statement also retains values from the previous iteration of the DATA step. It automatically retains the variable in the statement (in the example above, it's MAXNUM). It is used to cumulatively add the value of an expression to a variable. It takes the form of

```
Variable + expression;
```

It assigns the summed value of variable and the expression to the variable and specifies the variable to be retained.

In the example, within the same customer number MAXNUM is retained and initial value is set to 0 by default. Therefore the first time when MAXNUM+1; is executed, you are adding 0+1=1 and assigning this value to MAXNUM. With each subsequent iteration of the DATA step, the value of MAXNUM is increased by 1 for the next observation and MAXNUM will have a value of 2, and so forth. When it encounters the last observation of the same customer number, that would be the total number of accounts for that customer.

4. CALL SYMPUT: The SYMPUT routine allows you to create a new macro variable or assign a new value to an existing one. You can use this statement to call inside a DATA step to produce a global variable. But keep in mind that macro variable being created in this statement has to be enclosed in quotation marks. In the example above, you use the CALL SYMPUT routine to assign value of maximum number of accounts, which is the value of variable MAXNUM from data set CHECKACCT, to macro variable MXNUM which can be referred later in your codes.

5. _NULL_ Statement: Your purpose is to create a macro variable and assign value of the maximum number of accounts you obtained from the previous DATA step to this macro variable so that you can reference to it later in your code, so you just want to execute a DATA step but do not want to create a SAS data set.

6. END=variable: It is a useful SAS option that can be used when you read SAS DATA set using SET statement. It specifies a temporary SAS variable. Any valid SAS name can be chosen as the variable name. Normally the value of the variable is set to 0, but when the end of the data has been reached or when the input statement encounters the last record in the SAS DATA set being read, its value will be set to 1.

RESHAPE DATA – MANY TO ONE

The original data set with multiple observations per customer has to be transformed to the format of one observation per customer for the analysis purpose.

```
data transprod;
    set rawdata;
    by cxno;
    retain prod1-prod&mxnum;
    array prd(*) $15 prod1-prod&mxnum;
    if first.cxno then do;
      i=1;
      do j=1 to &mxnum;
```

```
            prd(j)=' ';
          end;
        end;
        prd(i)=product;
         if last.cxno then output;
       i+1;
         keep cxno prod1-prod&mxnum;
    run;
```

After being transposed, the products in the new data set will be as follows.

1234567 Credit_1 Credit_A Credit_B Credit_E Credit_8 Deposit_1 Credit_2 Credit_G Credit_6 Credit_F Credit_H Deposit_B
2234568 Credit_1 Credit_A
3344569 Deposit_A Deposit_B Credit_B
4455660 Deposit_1 Deposit_A Deposit_9 Deposit_B Credit_1

In the codes above, first, you use a RETAIN statement to remember the values of product from 1 to the maximum number possible for a customer until it is time to output them. For detail about RETAIN statement, please refer to the section above. Then you use ARRAY and iterative DO statements that provide an easy way to process a group of variables identically. As for the ARRAY statement, it takes the form of

```
ARRAY array-name{dimension}$length elements;
```

It begins with the keyword ARRAY followed by an array name that must be a valid SAS name and in order to have a correct result from array, an array name that is the same as a SAS function name should be avoided. Following the array name is the {dimension} that indicates the number of elements. This can be an asterisk, a number, or a range of numbers used to describe the number and arrangement of the elements in this array. By default, array elements are treated as numeric. If array elements are character, then after the subscript, a $ followed by the length of the character variable referenced by the array must be included. The last part is the elements that can be used to define the variables that the array will reference. They can be the existing variables or new variables to be created. In the example you are creating new variables prod1 to prod with the maximum number of accounts a customer might have.

SAS arrays are very often used together with an iterative DO loop which begins with the iterative DO statements, followed by other SAS statements and ends with an END statement. The DO statement contains an index variable whose name can be chosen, as you like. In array processing you generally want the loop to execute once for each element in the array, so you specify the values of your index variables as 1 TO the number of elements in the array. By default SAS steps through the elements by increasing the value of the index variable by 1 before each new iteration of the loop.

Here you create two index variables I and J. The loop is processed as many times as the maximum number of the accounts possible for a customer and the variable being processed changes as the value of the index variable changes, so on the first pass through the loop the array element prd{1}, which maps to the variable prod1 is used. The index variable I and J are increased to 2 on the second pass, so the array element prd{2} or variable prod2 is processed next, and so forth through all the variables. So each time when a new cxno is read in, the elements of ARRAY prd, (prod1-prod&mxnum) are initialized to missing. Prd(i) = product statement instructs the program to set the element of prd with a subscript equal to the current value of I, equal to the current value of prod until LAST.cxno is encountered where the observation is output to new data set TRANSPROD. And only cxno and prod1-proc&mxnum are kept. The process is then repeated for the next cxno.

Alternatively, you can have the same result by using PROC TRANSPOSE statement if only one variable is transposed at one time. However, with arrays and the associated DO loops in DATA step discussed above, it's more flexible where you can transpose multiple variables at a time by defining different arrays. For your reference here is how PROC TRANSPOSE works:

```
proc transpose data=rawdata out=transprod(drop=_name_) prefix=prod;
by cxno;
var product;
run;
```

This is a very powerful and useful procedure. By default, the variable created starts with col and ends with a numeric number from 1 to the maximum number of the value of the variable transposed in the VAR statement, such as col1, col2, etc. But with PREFIX= option, the variable name to be created will start with the name defined in the option, it's prod here in the example.

IDENTIFY THE PRODUCTS

Now you can summarize product data at customer level to identify who had the current products so that the promotion letter

will be sent out accordingly.

```
data prodgroup;
  set transprod;
  by cxno;
  array c(*) c1-c&mxnum;  /* define array c{*} to create new credit variables          */
  array d(*) d1-d&mxnum;  /* define array d{*} to create new deposit variables         */
  array m(*) m1-m&mxnum;  /* define array m{*} to create new mortgage variables        */
  array nm(*) nm1-nm&mxnum;/*define array nm{*} to create new non-mortgage variables   */
  array dd(*) dd1-dd&mxnum;/*define array dd{*} to create new demand deposit variables */
  array t(*) t1-t&mxnum;  /* define array t{*} to create new term deposit variables    */
  array prod(*) $8 prod1  /* define array prod{*} to reference existing character      */
              -prod&mxnum;/* variables created by array prd{*} from previous data step */
   do f=1 to &mxnum;
     c(f)=prod(f) in
                 ('Credit_1','Credit_2','Credit_3','Credit_4', 'Credit_5','Credit_6',
                  'Credit_7','Credit_8','Credit_9','Credit_10','Credit_A','Credit_B',
                  'Credit_C','Credit_D','Credit_E','Credit_F', 'Credit_G','Credit_H',
                  'Credit_I','Credit_J');
     d(f)=prod(f) in
               ('Deposit_1','Deposit_2','Deposit_3','Deposit_4','Deposit_5','Deposit_6',
               'Deposit_7','Deposit_8','Deposit_9','Deposit_10','Deposit_A', 'Deposit_B',
               'Deposit_C','Deposit_D','Deposit_E','Deposit_F','Deposit_G','Deposit_H',
               'Deposit_I','Deposit_J');
     m(f)=prod(f) in
                  ('Credit_1','Credit_2','Credit_3','Credit_4','Credit_5',
                   'Credit_6','Credit_7','Credit_8','Credit_9','Credit_10');
     nm(f)=prod(f) in
                   ('Credit_A','Credit_B','Credit_C','Credit_D', 'Credit_E',
                    'Credit_F','Credit_G','Credit_H', 'Credit_I','Credit_J');
     dd(f)=prod(f) in
                  ('Deposit_1','Deposit_2','Deposit_3','Deposit_4', 'Deposit_5',
                   'Deposit_6','Deposit_7','Deposit_8','Deposit_9','Deposit_10');
     t(f)=prod(f) in
                 ('Deposit_A','Deposit_B','Deposit_C','Deposit_D', 'Deposit_E',
                  'Deposit_F','Deposit_G','Deposit_H', 'Deposit_I','Deposit_J');
   end;
   totalc=sum(of c1-c&mxnum);
   totald=sum(of d1-d&mxnum);
   totalm=sum(of m1-m&mxnum);
   totalnm=sum(of nm1-nm&mxnum);
   totaldd=sum(of dd1-dd&mxnum);
   totalt=sum(of t1-t&mxnum);
length creditgroup  prodgroup depositgroup $ 40;
 /* customer in the population must have a deposit product or a credit  */
/* product, otherwise the following prodgroup will have another value  */
/* as 'product - none'                                                 */
if totalc ne 0 and totald ne 0 then prodgroup='both credit and deposit';
else if totalc ne 0            then prodgroup='credit only';
else                                prodgroup='deposit only';
if totalm ne 0 and totalnm ne 0 then creditgroup=
                                          'credit - both mortgage and non mortgage';
```

```
   else if totalm  ne 0            then creditgroup='credit - mortgage only';
   else if totalnm ne 0            then creditgroup='credit - non mortgage only';
   else                                 creditgroup='credit - none';
   if totaldd ne 0 and totalt ne 0 then depositgroup='deposit - both demand and term';
   else if totaldd ne 0            then depositgroup='deposit - demand deposit only';
   else if totalt  ne 0            then depositgroup='deposit - term deposit only';
   else                                 depositgroup='deposit - none';
   keep cxno prodgroup creditgroup depositgroup;
run;
```

Variable=variable1 eq value is special SAS statement which is equivalent to *if variable1 eq value then variable=1; else variable=0;.* The value of variable will be 1 or 0 depending on the value of the variable1. So you use this statement combined with SAS ARRAY to check all the products a customer might have to identify what type of products s/he has. If customer has mortgage product then m1-m&mxnum will have a value of 1, if customer doesn't have mortgage products, then m1-m&mxnum will have a value of 0. The sums of d1-d&mxnum, c1-c&mxnum, nm1-nm&mxnum,dd1-dd&mxnum and t1-t&mxnum will be handled similarly. Please notice that the full product names are listed in the statement because in reality, the product names can be completely different from each other (otherwise, you may use some sorts of function to shorten your codes as the names have some kinds of commonality such as CREDIT, DEPOSIT, 1 to 10 and A to J etc.) The final result is as following:

cxno	prodgroup	creditgroup	depositgroup
1234567	both credit and deposit	credit - both mortgage and non mortgage	deposit - both demand and term
2234568	credit only	credit - both mortgage and non mortgage	deposit - none
3344569	both credit and deposit	credit - non mortgage only	deposit - term deposit only
4455660	both credit and deposit	credit - mortgage only	deposit - both demand and term

From the result, you can determine that you don't need to send any letters to the first customer because the customer already has 4 types of current products both credit (including mortgage and normal loans) and deposit (including demand and term deposit). But for the second customer, you should send a letter asking customer if s/he is interested in your deposit products (both demand and term) while a letter promoting mortgage loan and demand deposit and a letter with normal loan promotion only will be sent to the third and fourth customer, respectively.

CONCLUSION

SAS® offers many powerful ways to manipulate the data and generate customized reports. Whenever you need to process a large number of variables, the optimal option will be using arrays and the associated DO loops. They can reduce or dramatically simplify the coding, flexibly manage your data and effectively solve your business problem. What has been demonstrated is the typical usage of arrays and the associated DO loop facility. They have other complicated and advanced features that are not covered here. Multiple records per customer can be transposed to one record per customer by using either arrays and the associated DO loops or PROC TRANSPOSE procedure if only one variable needs to be restructured at one time. If you need to create a macro variable that is assigned the value of a variable from a DATA set to be referenced later in your code, you can use the CALL SYMPUT statement to accomplish this. When you use the CALL SYMPUT routines, you need to remember that the macro variable being created in the statement has to be enclosed in quotation marks.

REFERENCES

SAS OnlineDoc®, Version 8, Cary, NC: SAS Institute Inc., 1999

Cody, Ronald P., and Raymond Pass. *SAS*® Programming by Example. Cary, NC: SAS Institute Inc., 1995. 337pp.

Delwiche, Lora D., and Susan J. Slaughter. The Little SAS Book: A Primer, Third Edition. Cary, NC: SAS Institute Inc., 2003

ACKNOWLEDGMENTS

The author would like to thank Business & Information Senior Manager Mr. Joseph Virey for his support. Special thanks go to Debbie Buck, Louie Luo and Isabel Huang for their excellent suggestions and help in reviewing this paper. Also of special note are two papers from previous SUGIs written by Stephen Keelan (Paper 66-27) and Lynn Palmer (Paper 88-26) and one paper from NESUG 16 written by Ronald Fehd (CC015).

CONTACT INFORMATION

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

Rodger Zhang

TD Canada Trust
77 King Street W.,
Toronto, Ontario, Canada M5K 1A2
Work Phone: 416 307 6056
Email: rodger.zhang@td.com