PharmaSUG 2014 - Paper CC05

Having Fun with RACE Derivation in DM Domain

Chunxia Lin, InVentiv Health Clinical, Ossining, NY
Deli Wang, Tarrytown, NY

ABSTRACT

Race in SDTM DM domain is an expected variable; generally it is quite easy to derive. However, there are times when raw database is set up with multiple race variables (RACE_X), and a subject may have multiple race values. Some clients require that, if a subject has more than one race values, set RACE as “MULTIPLE”, otherwise, set to the single selected race value. The logic looks simple, however SAS programmers have to figure out two important steps before coding: 1. whether the subject has more than one race variables selected; 2. which race variable has nonmissing value. The Multiple IF-THEN/ELSE statements work fine but the coding is a little tedious. This paper introduces four methods to derive RACE using the logical expression and SAS functions. Each of the methods is discussed separately depending on the attributes of the data (character/numeric).

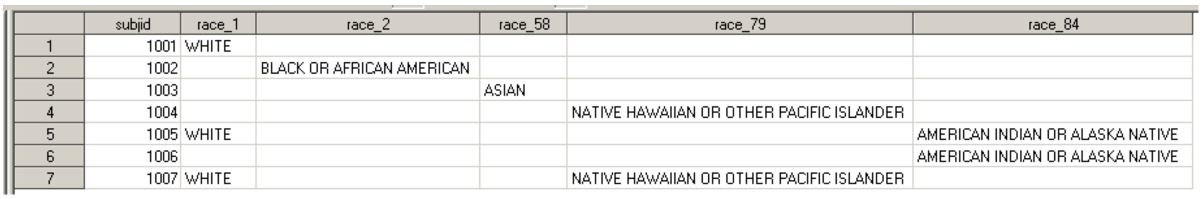
INTRODUCTION

The SAS system provides an extensive library of “Built-in” functions. The collection of SAS functions and call routines allow SAS programmers to do extensive manipulation on all sorts of data. Logical (Boolean) expression yields the result “true” or “false”, and represented in SAS by the values 1 and 0 respectively. Logical expression can make the code much easier to understand and maintain, particularly if the conditions are complex and/or are repeated several times in the code. This paper takes RACE derivation as an example to illustrate the simplicity, power and beauty of logical expression and SAS functions in our daily SAS programming.

The following example shows that the race values in raw data are organized in a horizontal structure and represented by different RACE variables (e.g RACE_84, RACE_58, RACE_2, RACE_79, RACE_1).The client require that if more than one races are valued for the subject, then set RACE as ‘MULTIPLE’, and output corresponding RACE values in SUPPDM domain; Otherwise, set to the single selected race value. The focus of this paper is on determining cases in which more than one race value was selected when deriving DM RACE. How to output distinct race values in SUPPDM for those cases is out of this paper’s discussion scope.

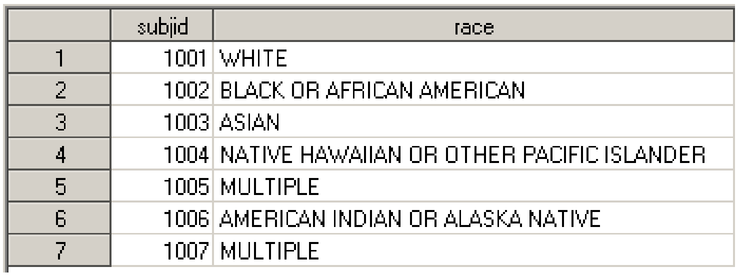
SITUATION I: COLLECTED CRF DATA IS CHARACTER

Table1 showed partial character RACE data collected from CRF. The client require that If more than one races are valued for the subject, then set to ‘MULTIPLE’; Otherwise, set to the single value selected in RACE_84, RACE_58, RACE_2, RACE_79 or RACE_1.



	subjid	race_1	race_2	race_58	race_79	race_84
1	1001	WHITE				
2	1002		BLACK OR AFRICAN AMERICAN			
3	1003			ASIAN		
4	1004				NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER	
5	1005	WHITE				AMERICAN INDIAN OR ALASKA NATIVE
6	1006					AMERICAN INDIAN OR ALASKA NATIVE
7	1007	WHITE			NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER	

Table 1. Character CRF RACE Dataset: RACE_C



	subjid	race
1	1001	WHITE
2	1002	BLACK OR AFRICAN AMERICAN
3	1003	ASIAN
4	1004	NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER
5	1005	MULTIPLE
6	1006	AMERICAN INDIAN OR ALASKA NATIVE
7	1007	MULTIPLE

Table 2. Expected RACE Output in DM Domain

METHOD A: PROC SQL (CMISS/COALESCE)

CMISS function counts the number of missing arguments. The COALSCE function in PROC SQL returns the first nonmissing value in a list of variables, which enables us to process RACE data with a single function call instead of writing multiple IF-THEN/ELSE statements. Dataset race_c1 created by Method A showed exactly same result as Table 2.

```
PROC SQL NOPRINT;
     CREATE TABLE race_c1 AS
     SELECT subjid, case when
        cmiss(race_1, race_2, race_58, race_79, race_84) <4 then "MULTIPLE"
       else coalesce(race_1, race_2, race_58, race_79, race_84)
       end as race format=$100.
  from race_c;
QUIT;
```

METHOD B: DATA STEP (ARRAY)

SAS array is a convenient way of temporarily identifying a group of variables for processing within data step. Method B takes advantage of an array and a loop to simplify RACE derivation. Dataset race_c2 showed same result as Table 2.

```
DATA race_c2;
    SET race_c;
    length race $100;

    array arace {5} race_1 race_2 race_58 race_79 race_84;

       do i=1 to 5;
         if arace {i} ne '' then do;
           if cmiss(race_1, race_2, race_58, race_79, race_84) =4 then race=arace{i};
           else race="MULTIPLE";
         end;
       end;

    keep  subjid race ;

RUN;
```

METHOD C: DATA STEP (LOGICAL EXPRESSION/SUM/CATX)

Race_x>' ' in Method C is a logical expression yielding the value “1” when race_x>' ‘condition is true, otherwise yields “0” when false. SUM function adds the result of an expression to an accumulator variable and treats an expression that produces a missing value as zero. CATX function concatenates character strings, strips both leading and trailing blanks, and inserts separators. The first argument to CATX specifies the separator. Like other methods, Method C outputs same results as TABLE 2.

```
DATA race_c3;
    SET race_c;
    length race $100;

    if sum(race_1>'',race_2>'',race_58>'',race_79>'', race_84>'')=1 then
            race=catx('',race_1, race_2,race_58, race_79, race_84);
    else race="MULTIPLE";

    keep  subjid race ;

RUN;
```

METHOD D: DATA STEP (MACRO)

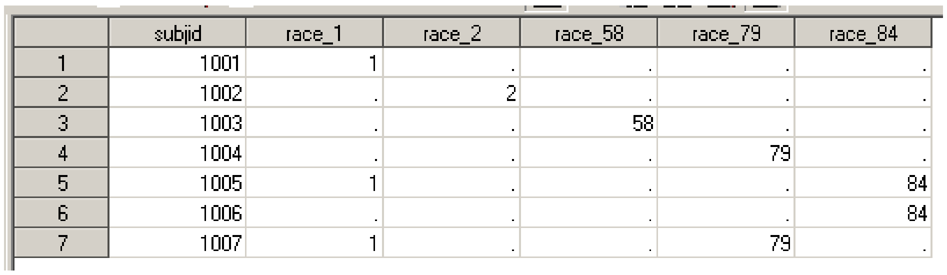
Method D takes benefits of macro language to process RACE data instead of writing multiple IF-THEN/ELSE statements. Like other methods, Method D outputs same results as TABLE 2.

```
%MACRO race_c(var);
   if &var^='' then race=&var;
```

```
%MEND;

DATA race_c4;
    SET race_c;
    length race $100;
    if sum(race_1='', race_2='',race_58='',race_79='', race_84='')=4 then do;
          %race_c(race_1);
          %race_c(race_2);
          %race_c(race_58);
          %race_c(race_79);
         %race_c(race_84);

    end;
    else race="MULTIPLE";

    keep subjid race;
RUN;
```

SITUATION 2: COLLECTED CRF DATA IS NUMERIC

Table 3 showed partial numeric RACE data collected from CRF. The client requires that If more than one race are valued for the subject, then set to ‘MULTIPLE’; Otherwise, set to the decoded value for the single selected value in RACE_84, RACE_58, RACE_2, RACE_79 or RACE_1. PROC FORMAT defines the decoded values for the collected RACE_x. Expected DM_RACE output shown in TABLE 2.



	subjid	race_1	race_2	race_58	race_79	race_84
1	1001	1	.	.	.	.
2	1002	.	2	.	.	.
3	1003	.	.	58	.	.
4	1004	.	.	.	79	.
5	1005	1	.	.	.	84
6	1006	.	.	.	.	84
7	1007	1	.	.	79	.

Table 3. Numeric CRF RACE Dataset: RACE_N

```
PROC FORMAT;
   VALUE racec

   1="WHITE"
   2="BLACK OR AFRICAN AMERICAN"
   84="AMERICAN INDIAN OR ALASKA NATIVE"
   58="ASIAN"
   79="NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER"
   other=" "
   ;
 RUN;
```

METHOD A: PROC SQL (NMISS/COALESCE)

Different from CMISS function, NMISS function counts the number of missing numeric arguments. The COALSCE function returns the single selected race_x numeric value, and then PUT function returns the decoded RACE value using a specified RACEC format. Dataset race_n1 created by Method A showed exactly same result as Table 2.

```
PROC SQL NOPRINT;
      CREATE TABLE race_n1 AS
      SELECT subjid, case when
        nmiss(race_1, race_2, race_58, race_79, race_84) <4 then "MULTIPLE"
        else put(coalesce(race_1, race_2, race_58, race_79, race_84), racec.)
       end as race format=$100.
   from race_n;
QUIT;
```

METHOD B: DATA STEP (ARRAY/N)

Similarly, SAS array is still used to process numeric RACE data. The N function returns the number of argument variables which have nonmissing values. Dataset race_n2 showed exactly same result as Table 2.

```
DATA race_n2;
    SET race_n;
    length race $100;

      array arace {5} race_1 race_2 race_58 race_79 race_84;
         do i=1 to 5;
           if arace {i} ne . then do;
             if n(race_1, race_2, race_58, race_79, race_84) =1 then
               race=put(arace{i},racec.);
             else race="MULTIPLE";
           end;
         end;
    keep  subjid race ;

RUN;
```

METHOD C: DATA STEP (LOGICAL EXPRESSION/SUM)

Logical expression (RACE_x>.) is similarly used to handle numeric data here. The first SUM function added logical expression yielded values together to determine how many race values selected for the subject. The second SUM function returns the numeric race value when single race is selected. Like other methods, Method C outputs same results as TABLE 2.

```
DATA race_n3;
    SET race_n;
    length race $100;

    if sum(race_1>.,race_2>.,race_58>.,race_79>., race_84>.)=1 then
       race=put(sum(race_1,race_2,race_58, race_79,race_84), racec.);
      else race="MULTIPLE";

    keep  subjid race ;

RUN;
```

METHOD D: DATA STEP (MACRO/MISSING)

Different from RACE character code, Method D added PUT function in the macro to decode the numeric race value, and replaced logical expression with MISSING function within SUM function. The MISSING function checks if a value is a null or missing and returns a numeric result 1/0. Like other methods, Method D outputs same results as TABLE 2.

```
%MACRO race_n(var);
   if &var^=. then race=put(&var,racec.);
%MEND;

DATA race_n4;
    SET race_n;
    length race $100;
    if sum(missing(race_1), missing(race_2),missing(race_58),missing(race_79),
            missing(race_84))=4 then do;
          %race_n(race_1);
          %race_n(race_2);
          %race_n(race_58);
          %race_n(race_79);
          %race_n(race_84);

    end;
    else race="MULTIPLE";
    keep subjid race;
RUN;
```

CONCLUSION

A comprehensive knowledge of the appropriate SAS® functions can often make difference between a difficult and a fun programming task. The DM_RACE derivation methods introduced in the paper proved to be such an example to illustrate the robustness of SAS functions. The methods could be useful when SAS programmers need to derive a single variable where its multiple values are organized horizontally in raw data and represented by different variables. In addition, this paper also serves as an illustration of how SAS functions and logical expressions can be used creatively in SAS programming.

REFERENCES

1. Study Data Tabulation Model Implementation Guide: Human Clinical Trials, Version 3.1.2. Published by CDISC November 12, 2008.
2. SAS/BASE Software: Version 9, SAS® Institute Inc., Cary NC
3. SAS/MACRO Software: Version 9.1.3, SAS® Institute Inc., Cary NC

ACKNOWLEDGMENTS

Many thanks to Sandra VanPelt Nguyen in providing the valuable suggestions and support.

CONTACT INFORMATION

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

Name: Chunxia Lin
Enterprise: InVentiv Health Clinical
Work Phone: (914) 9230173
E-mail: chunxia.lin@inventivhealth.com

Name: Deli Wang
E-mail: deli.wang@hotmail.com