Paper 890-2017

Getting Classy: A SAS® Macro for CLASS Statement Automation

Erica Goodrich M.S., Brigham and Women's Hospital, Boston, MA
Daniel Sturgeon, M.S., Brigham and Women's Hospital, Boston, MA
Kathryn Schurr M.S., Quest Diagnostics, Hudsonville, MI

INTRODUCTION

When working with a variety of statistical models, it's important to treat categorical variables, no matter the data type, different than continuous variables for proper analysis and interpretation. To account for this, a large number of statistical procedures in SAS® have a CLASS statement. When including variables in the class statement, SAS® treats included variables as categorical. Variables not included in the CLASS statement, but included in the MODEL statement as continuous variables. In large models or models which may change frequently, there may instances where user error can lead to variables being not added to the CLASS statement or are kept in the CLASS statement when they are not included in the MODEL. These additions or omissions can also lead to incorrect results. To solve this issue we created the % MasterCLASS macro. The macro explained in further detail below, will allow for creating an accurate and automatic CLASS statement for analysis. In addition to the macro, a few simple input prompts listed below will provide the output needed for automatically designating categorical variables to the class statement along with a desired model statement.

Input:

(Code) %MasterCLASS(DATA = <dataset>, VARS = <variables>, LEVELS = <*number of levels*>);

(Example) %MasterCLASS(DATA=SASHELP.CARS, VARS=ORIGIN MAKE TYPE MPG_CITY WEIGHT CYLINDERS DRIVETRAIN ENGINESIZE HORSEPOWER MPG_HIGHWAY WEIGHT WHEELBASE LENGTH);

Output:

Three macro variables: &CLASS. &VARIABLE. &MODEL.
The first is &MODEL, which is a list of all of the covariates mentioned for the model. This matches the list provided in the VARS.

```
%put &MODEL.;
CYLINDERS DRIVETRAIN MAKE ORIGIN TYPE MPG_CITY WEIGHT ENGINESIZE
HORSEPOWER MPG_HIGHWAY WEIGHT WHEELBASE LENGTH
```

Two macro variables are created which can be used in the class statement. &CLASS provides a list using the param=reference method, whereas &VARIABLE provides a list of covariates which should be included in the class statement. This second method will work with ascending and descending options.

```
%put &CLASS.;
CYLINDERS (REF='3') DRIVETRAIN(REF='All') MAKE(REF='Acura') ORIGIN(REF='Asia')
TYPE(REF='Hybrid')

%put &VARIABLE.;
CYLINDERS DRIVETRAIN MAKE ORIGIN TYPE
```

THE MACRO

Even though intimate knowledge of the working parts of the macro is not necessary for use, we decided to walk through the steps of what is going behind the scenes of this program. The macro starts with three parameters; DATA, which is the name of the set you are using, VARS, which are the variables of interest for your model, and the optional LEVELS, which indicates the lowest number of levels allowed in a continuous variable allowed before considering it a categorical. The variables entered for VARS must be separated by a space. These variables will be evaluated inside the macro to determine if they should be considered categorical or numeric. The option LEVELS has a default value of ten. This means any values which are numeric character type with less than 10 distinct values will be treated as numeric. This may vary depending on datasets and specifications of data values. For this example we will be using the SASHELP.CARS data set. In this example we will be trying to predict MSRP using the string variables make, type, origin and the numeric variables engine size, cylinders, horsepower, weight. Cylinders consists of only seven levels, so for this example we want to treat it as a string. Please note that the variables are being used for demonstration purposes and reproducibility, rather than statistical rigor. Code needed to invoke the macro is seen above, whereas the code for the entire %MASTERCLASS macro is found in the appendix of this paper.

Table 1. Number of Levels for Numeric Variables

CYLINDERS	ENGINESIZE	HORSEPOWER	WEIGHT
7	43	110	348

TECHNICAL DETAILS

The SAS macro code includes steps to categorize the variables using PROC CONTENTS and later PROC SQL, and setting up the data in a way that gives us the variables we want with a reference. This involves arrays within the data step. Firstly, PROC CONTENTS is invoked.

```
PROC CONTENTS DATA=&DATA out=OUTCONTENT NOPRINT;
RUN;
```

Through PROC CONTENTS, we export a list of all covariates in the dataset of interest along with additional information, such as variable type. Type indicates whether the variable is numeric (1) or character (2). This isn’t enough to decide which variables should be entered into the class statement. As mentioned previously, for this example we believe the variable cylinders should be treated as a character even though is coded as a numeric. Table 2 shows an example of the PROC CONTENTS output.

Table 2. Example PROC CONTENTS Variable Type:

Obs	LIBNAME	MEMNAME	MEMLABEL	TYPEMEM	NAME	TYPE	LENGTH
1	SASHELP	CARS	2004 Car Data		Cylinders	1	8
2	SASHELP	CARS	2004 Car Data		DriveTrain	2	5
3	SASHELP	CARS	2004 Car Data		EngineSize	1	8
4	SASHELP	CARS	2004 Car Data		Horsepower	1	8
5	SASHELP	CARS	2004 Car Data		Invoice	1	8
6	SASHELP	CARS	2004 Car Data		Length	1	8

To address variables with multiple levels that are coded as numeric values the next step is used to figure out how many levels each has:

```
%DO I = 1 %TO &CNT1;
      %LET VAR = %SCAN(&VARS.,&I.,' ');

      PROC SQL;
            CREATE TABLE _A AS
            SELECT DISTINCT
                  "&VAR" AS NAME,
                  COUNT(DISTINCT &VAR) as LEVELS
            FROM &DATA;
      QUIT;

      /*Table where we keep variable information regarding levels*/
      DATA _VARS;
            SET _VARS _A;
      RUN;
%END;
```

The first statement counts how many variables are being included. This is used for the do-loop counter of inside the macro. The %SCAN function goes through each variable, one at a time, and inserts it into the PROC SQL statement below to get the levels using the count function. The results are added to a new dataset. We now combine the information with that result to the type information we retrieved from the PROC CONTENTS so we have two pieces of information:

Table 3. Example Result:

NAME	LEVELS	TYPE
MAKE	38	2
TYPE	6	2
ORIGIN	3	2
MPG_CITY	28	1
WEIGHT	348	1
Cylinders	7	1

From this information we automatically move all type character to one group along with type numeric that contains fewer levels than what we define. The rest are moved to their own dataset.

Next, two arrays are set up: one for the numeric variables to be used as character and another for the variables already considered string. We do two arrays because the array needs to be the same type of variable.

```
DATA CAT2;
      SET &DATA;
      ARRAY CATVAR[&CNT3] &CATEGORICAL2;

      DO J = 1 TO &CNT3;
      VARIABLE = SCAN("&CATEGORICAL2",J,' ');
      LEVEL = CATVAR[j];
      OUTPUT;
      END;

      KEEP VARIABLE LEVEL;
RUN;

DATA CAT3;
      SET &DATA;
      ARRAY CATVAR[&CNT2] &CATEGORICAL1;
      DO J = 1 TO &CNT2;
            VARIABLE = SCAN("&CATEGORICAL1",J,' ');
            LEVEL = COMPRESS(INPUT(CATVAR[j],$255.));
            LEVRAW = CATVAR[j];
            OUTPUT;
      END;

      KEEP VARIABLE LEVEL LEVRAW;
RUN;
```

The arrays create tables that consist of both the variable and the level. An example would be if the variable has seven levels, there will be seven lines for it. For the numeric variables there is extra code that converts the levels to character and creates a variable 'levelraw' for sorting purposes.

Finally, we combine everything and define reference variables. The reference is based on the first variable alphabetically, and for the numbers it is based on smallest (which is why we had the extra sort variable). This can be changed in the code below if a change in ordering logic is desired.

```
/*Combining the results and defining the variable references*/
PROC SQL NOPRINT;
      CREATE TABLE CAT_ALL AS
      SELECT DISTINCT
            VARIABLE,
            LEVEL,
            LEVRAW,
            COMPRESS(VARIABLE||"(REF='"||LEVEL||"')") AS CLASS
      FROM CATCOMB
      WHERE LEVEL NE ''
      GROUP BY VARIABLE
      ORDER BY 1, 3, 2;
QUIT;

DATA CAT_ALL;
      SET CAT_ALL;
      BY VARIABLE;
      IF FIRST.VARIABLE NE 1 THEN DELETE;
RUN;
```

The final table created inside the %MACROCLASS can be seen below in Table 4. This table is used for metadata to create two additional macro variables: One named &class which should be added to the class statement when parameter=reference methods are desired. The other is &variable, which can be used for the class statement if parameter=ascending or descending methods are used. After the %MasterCLASS macro is compiled, the &model macro variable will be created as well which can be used as a list of covariates inside of the statistical model.

Table 4. Final Table:

Obs	VARIABLE	LEVEL	LEVRAW	CLASS
1	Cylinders	3	3	Cylinders(REF='3')
2	DriveTrain	All	0	DriveTrain(REF='All')
3	MAKE	Acura	0	MAKE(REF='Acura')
4	ORIGIN	Asia	0	ORIGIN(REF='Asia')
5	TYPE	Hybrid	0	TYPE(REF='Hybrid')

```
%PUT &MODEL;
CYLINDERS DRIVETRAIN MAKE ORIGIN TYPE MPG_CITY WEIGHT ENGINESIZE HORSEPOWER
MPG_HIGHWAY WEIGHT WHEELBASE LENGTH
```

```
%PUT &CLASS;
CYLINDERS(REF='3') DRIVETRAIN(REF='All') MAKE(REF='Acura') ORIGIN(REF='Asia')
TYPE(REF='Hybrid')
%PUT &VARIABLE;
CYLINDERS DRIVETRAIN MAKE ORIGIN TYPE
```

These put statements show some example code of what will be created inside the model, class and variable macros.

CONCLUSION

The %MasterCLASS macro allows for flexible use and easy coding which can be easily programmed for additional use inside model creation or could work for a quick validation process on hand written code examples. Our appendix shows how this macro can be quickly used and applied to Multiple Linear Regression as well as Logistic Regression. With small changes inside of the macro even more customizable options can be available.

CONTACT INFORMATION

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

Erica Goodrich, M.S.
Biostatistician
TIMI Study Group, Division of Cardiovascular Medicine
Brigham and Women's Hospital
Building for Transformative Medicine
60 Fenwood Road, 7022B
Boston, MA, 02120
Work Email: EGoodrich@bwh.harvard.edu

Dan Sturgeon, M.S.
Sr. Biostatistician
Center for Surgery and Public Health (CSPH)
Brigham and Women's Hospital
One Brigham Circle
1620 Tremont Street, 4-020
Boston, MA, 02120
Work Email: DSturgeon@bwh.harvard.edu

Kathryn Schurr, M.S.
Health Information Analyst
Quest Diagnostics
Hudsonville, MI, 49426
Work Email: Kathryn.M.Schurr@QuestDiagnostics.com

APPENDIX: CODE

```
/*****************************************************************************************************
* MasterCLASS Macro
* This macro determines which variables should be added into the class statement
* Author: Dan Sturgeon, Erica Goodrich
*****************************************************************************************************/
/*****************************************************************************************************
MasterCLASS data prompts:
DATA =   Dataset name of interest
VARS =   list of variables to be added to the model of interest as covariates.
LEVELS = Lower limit for numeric variables before considering it categorical. The default is set at 10.
*****************************************************************************************************/

%MACRO MasterCLASS(DATA=, VARS=, LEVELS=10);

        /*Grabs metadata from the dataset of interest*/
        PROC CONTENTS DATA = &DATA out = OUTCONTENT noprint;
        RUN;

        /*Creates empty datasets for later use*/
        DATA _VARS;
                LENGTH NAME $255.;
                SET _NULL_;
        RUN;

        DATA CAT;
                LENGTH NAME $255.;
                SET _NULL_;
        RUN;

        DATA CONT;
                LENGTH NAME $255.;
                SET _NULL_;
        RUN;

        /*Count the number of variables of interest*/
        %LET CNT1 = %SYSFUNC(countw(&VARS,' '));

        /*Setting up the do loop through the variable list*/
        %DO I = 1 %TO &CNT1;
                /*This will scan through the list of &VARS. one at a time separated by a space, until it
                reaches the end.*/
                %LET VAR = %SCAN(&VARS.,&I.,' ');

                /*Get the count of levels to determine type of numeric*/
                PROC SQL;
                        CREATE TABLE _A AS
                        SELECT DISTINCT
                                "&VAR" AS NAME,
                                COUNT(DISTINCT &VAR) as LEVELS
                        FROM &DATA;
                QUIT;

                /*Create a table where variable information regarding levels is kept*/
                DATA _VARS;
                        SET _VARS _A;
                RUN;
        %END;

        /*Combine level counts with data type for the categorical variables that have many levels*/
        PROC SQL;
                CREATE TABLE _VARS1 AS
                SELECT A.NAME,
                          A.LEVELS,
                          B.TYPE
                  FROM _VARS A,
                          OUTCONTENT B
                 WHERE UPCASE(A.NAME) = UPCASE(B.NAME);
        QUIT;

        /*If the number of levels < than the minimum we defined OR the type is string then they are
        lumped into categorical*/
        DATA CONT CAT;
                SET _VARS1;
                IF LEVELS < &LEVELS OR TYPE = 2 THEN OUTPUT CAT;
                ELSE OUTPUT CONT;
        RUN;

        PROC SQL NOPRINT;
                SELECT COUNT(*)
                   INTO :CATROW
                FROM CAT;
                SELECT COUNT(*)
                   INTO :CONTROW
                FROM CONT;
        QUIT;


        /*Now the continuous variables are noted*/
        %IF &CONTROW GE 1 %THEN %DO;

                PROC SQL NOPRINT;
                        SELECT NAME
                          INTO :CONTINUOUS SEPARATED BY " "
                        FROM CONT;
                QUIT;

        %END;

        %IF &CONTROW GE 1 %THEN %DO;

                /*Combine two categorical types: those that are numeric and those that are not.*/
                PROC SQL NOPRINT;
                        SELECT NAME
                          INTO :CATEGORICAL1 SEPARATED BY " "
                         FROM CAT
                        WHERE TYPE = 1;
                QUIT;

                PROC SQL NOPRINT;
```

```
          SELECT NAME
            INTO :CATEGORICAL2 SEPARATED BY " "
           FROM CAT
          WHERE TYPE = 2;
QUIT;

/*Gather counts for both of those groups*/
PROC SQL NOPRINT;
          SELECT COUNT(*)
            INTO :CNT2
            FROM CAT
          WHERE TYPE = 1;
QUIT;

PROC SQL NOPRINT;
          SELECT COUNT(*)
            INTO :CNT3
            FROM CAT
          WHERE TYPE = 2;
QUIT;

/*First go through the string categorical variables*/
%IF &CNT3 GE 1 %THEN %DO;

          DATA CAT2;
          /*LENGTH LEVEL $255.;*/
                    SET &DATA;
                    ARRAY CATVAR[&CNT3] &CATEGORICAL2;
                    DO J = 1 TO &CNT3;
                              VARIABLE = SCAN("&CATEGORICAL2",J,' ');
                              LEVEL = CATVAR[j];
                              OUTPUT;
                    END;

                    KEEP VARIABLE LEVEL;
          RUN;

%END;

%ELSE %DO;
          DATA CAT2;
                    SET _NULL_;
          RUN;
%END;

%IF &CNT2 GE 1 %THEN %DO;

/*Now with numeric variables, since type must match since the arrays cannot handle
different types*/
          DATA CAT3;
                    SET &DATA;
                    ARRAY CATVAR[&CNT2] &CATEGORICAL1;
                    DO J = 1 TO &CNT2;
                              VARIABLE = SCAN("&CATEGORICAL1",J,' ');
                              LEVEL = COMPRESS(INPUT(CATVAR[j],$255.));
                              LEVRAW = CATVAR[j];
```

```
                OUTPUT;
            END;

            KEEP VARIABLE LEVEL LEVRAW;
        RUN;

    %END;

    %ELSE %DO;
        DATA CAT3;
            SET _NULL_;
        RUN;
    %END;

    DATA CATCOMB;
        LENGTH LEVEL $255.;
        SET CAT2 CAT3;
        IF LEVRAW = . THEN LEVRAW = 0;
    RUN;

    /*Combines the results and defining the variable references*/
    PROC SQL NOPRINT;
        CREATE TABLE CAT_ALL AS
        SELECT DISTINCT
                 VARIABLE,
                 LEVEL,
                 LEVRAW,
                 COMPRESS(VARIABLE||"(REF='"||LEVEL||"')") AS CLASS
         FROM CATCOMB
        WHERE LEVEL NE ''
        GROUP BY VARIABLE
        ORDER BY 1,3,2;
    QUIT;

    DATA CAT_ALL;
        SET CAT_ALL;
        BY VARIABLE;
            IF FIRST.VARIABLE NE 1 THEN DELETE;
    RUN;

%END;

%GLOBAL MODEL CLASS VARIABLE; /*Macro variables we will use in our model*/

PROC SQL NOPRINT;
    SELECT VARIABLE,
             CLASS
     INTO :VARIABLE SEPARATED BY ' ',
             :CLASS SEPARATED BY  ' '
     FROM CAT_ALL;
QUIT;

%LET MODEL = &VARIABLE &CONTINUOUS;

/*Housekeeping: Delete all tables except the working table we are using*/
PROC DATASETS NOPRINT;
```

```
                DELETE CAT CAT2 CAT3 CAT_ALL _VARS _VARS1 CAT CONT OUTCONTENT _A;
        RUN;
        QUIT;

        /*Housekeeping: Clear macro variables used during run in case of re-run*/
        %LET CATROW =;
        %LET CONTROW =;
        %LET CONTINUOUS =;
        %LET CATEGORICAL1 =;
        %LET CATEGORICAL2 =;
        %LET CNT2=;
        %LET CNT3=;
%MEND;

/************************************************************************************************************
EXAMPLES
************************************************************************************************************/
/*Example using built in SAS dataset, CARS. This is a terrible model and should not be used outside of
showing how the variables can be filled in*/
%MasterCLASS(DATA = SASHELP.CARS, VARS=ORIGIN MPG_CITY WEIGHT CYLINDERS
DRIVETRAIN ENGINESIZE HORSEPOWER MPG_HIGHWAY WEIGHT WHEELBASE LENGTH);

PROC GLM DATA=SASHELP.CARS;
        CLASS &CLASS./ PARAM=REF;
        MODEL MSRP = &MODEL.;
RUN;

/*Changes made to create a binary variable*/
DATA CARS;
        SET SASHELP.CARS;
        IF MSRP LT 25000 THEN UNDER25K = 1;
        ELSE UNDER25K = 0;
        LABEL UNDER25K = "CAR MSRP UNDER 25K? (1 = YES, 0 = NO)";
RUN;

/*An example could be using this model to show a saturated model and then use a selction method*/
PROC LOGISTIC DATA = SASHELP.CARS DESCENDING;
        CLASS &CLASS./PARAM = REF;
        MODEL UNDER25K = &MODEL./SELECTION = BACKWARD;
RUN;

% MasterCLASS (DATA=SASHELP.CARS, VARS = ORIGIN MPG_CITY WEIGHT DriveTrain
DRIVETRAIN ENGINESIZE HORSEPOWER MPG_HIGHWAY WEIGHT WHEELBASE LENGTH);

DATA CARS;
SET SASHELP.CARS;
        IF MSRP LT 25000 THEN UNDER25K = 1;
        ELSE UNDER25K = 0;
        LABEL UNDER25K = "CAR MSRP UNDER 25K? (1=YES, 0=NO)";
RUN;

PROC LOGISTIC DATA = CARS DESCENDING;
        CLASS &CLASS./PARAM = REF;
        MODEL UNDER25K = &MODEL.;
RUN;
```