SAS® GLOBAL FORUM 2017

April 2 – 5 | Orlando, FL

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

USERS PROGRAM

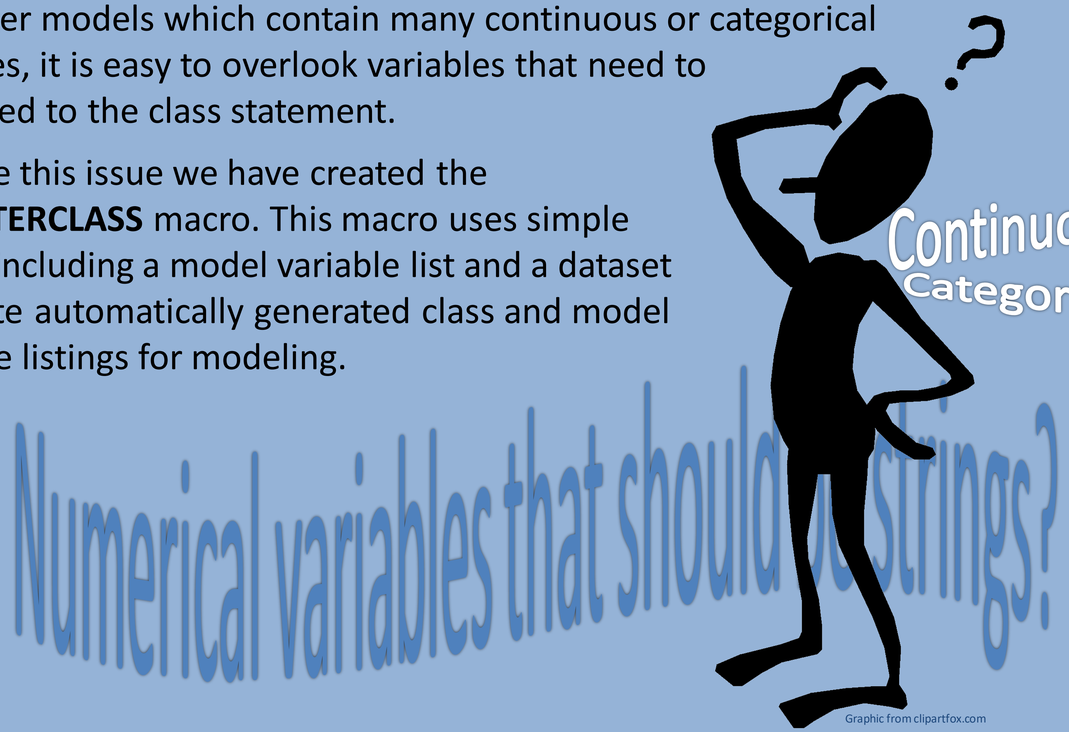
Getting Classy: A SAS® Macro for CLASS Statement Automation

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

ABSTRACT

When creating statistical models that include multiple covariates, it is important to address which variables are considered categorical and continuous for proper analysis and interpretation in SAS®. Categorical variables, regardless of SAS® data type, should be added in the model statement along with an additional class statement. For larger models which contain many continuous or categorical variables, it is easy to overlook variables that need to be added to the class statement.

To solve this issue we have created the **%MASTERCLASS** macro. This macro uses simple inputs including a model variable list and a dataset to create automatically generated class and model Variable listings for modeling.



Graphic from clipartfox.com

Example of a Statistical Model using a CLASS statement

Below is an example of code which could be used for a Logistic Regression. While the model has multiple covariates included, only two of them are listed in the class statement.

```
PROC LOGISTIC DATA=CARS DESCENDING;
  CLASS DRIVETRAIN(REF='All') ORIGIN(REF='Asia')/PARAM=REF;
  MODEL UNDER25K = ORIGIN MPG_CITY WEIGHT DRIVETRAIN ENGINESIZE
   HORSEPOWER MPG_HIGHWAY WEIGHT WHEELBASE LENGTH;
RUN;
```

This is a simple process when you are familiar with your dataset or have a small list of covariates considered in a model.

This becomes much more work when multiple models, or large numbers with small changes inside each model, or models with an extremely large number of covariates. Our macro helps improve efficiency and quality of class statement creation.

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

EXAMPLE DATASET: SASHELP.CARS

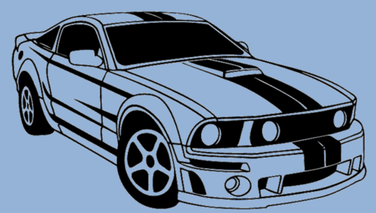
The example used for this macro will be shown with the SASHELP.CARS dataset with a small modification. Variable names, types, and formats are listed below.

Here's a bit of background on the CARS dataset:

- Includes 16 Variables:
  - **5 character types**
  - **10 numeric type s**
  - **1 numeric with a small number of levels**
  - **1 created binary variable**
  - **(Created based on MSRP </≥ $25,000)**

- Mock examples show modeling for MSRP or costs above or below 25k MSRP.

Variable	Type	Len	Format
Cylinders	Num	8	
DriveTrain	Char	5	
EngineSize	Num	8	
Horsepower	Num	8	
Invoice	Num	8	DOLLAR8.
Length	Num	8	
MPG_City	Num	8	
MPG_Highway	Num	8	
MSRP	Num	8	DOLLAR8.
Make	Char	13	
Model	Char	40	
Origin	Char	6	
Type	Char	8	
UNDER25K	Num	8	
Weight	Num	8	
Wheelbase	Num	8	



Graphics from clipartfox.com

INPUTS AND FINAL OUTPUTS

All you need is the dataset of interest, the **%MasterCLASS** macro, the model of interest covariates. From there, the macro is simple to use with the example prompt listed below:

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.

%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

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

Macro information and code available by phone!

Very little of the details are needed to work this macro! However, behind the scenes this is what's going on:

- User identifies dataset and variables of interest
- Metadata is gathered from PROC CONTENTS
- The metadata variables are separated by numeric and character types
- Logic checks for variables assigned as numeric which should be identified as categorical types to be included in CLASS statement
  - e.g. 0/1 binary variables or the "Cylinders" variable within SASHELP.CARS
- Unique levels are obtained for all variables in the class statement
- Macro variables are created for class and model statement.
  - Auto generated references groups (i.e. Type(REF='Hybrid')) can be auto-included for the CLASS statement
- End user just needs to know macro prompts!

%MacroCLASS code can be found with the QR code above or at https://tinyurl.com/h6mcd45

Getting Classy: A SAS® Macro for CLASS Statement Automation

%MasterCLASS code: everything needed for an automated class statement!

This code is also available in the published paper and online at the QR code posted on the previous slide.

```
%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; /*These are the 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
			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;
```

SAS® GLOBAL FORUM 2017

April 2 – 5 | Orlando, FL