

Automatic Indicators for Dummies:

A macro for generating dummy indicators from category type variables

Matthew Bates, Affusion Consulting, Columbus, OH

ABSTRACT

Dummy Indicators are critical to building many statistical models based on data with category type predictors. Most programmers rely on the “class” option within various procedures to temporarily build such predictors behind the scenes. This method carries with it a variety of limitations that can be overcome by auto-generating dummy indicators of all variables below a reasonable threshold of cardinality prior to running such procedures. Statistical modelers may find this topic a real effort and time saver while advanced SAS® programmers looking for creative techniques of efficiently automating processes may find this macro worth geeking out over.

INTRODUCTION

The topic of this paper is targeted to those who are either in need of a method to generate dummy indicators from category variables (character or numeric) or advanced SAS programmers (comfortable with writing macros) looking for alternative creative techniques to manage and automate processes. Those in the former group who are not too concerned with the details and getting lost in weeds are encouraged upon reading through the introduction section to skip to the conclusion section then utilize the macro as given at the end. For those in the latter section you will be exposed to innovative advanced techniques such as:

- Generation of lists of variable and strings as macro variables.
- Creating do loops based off the macro variable lists.
- Dynamic creation and calling of hash lookup tables.

Dummy Indicators have always been a powerful tool for building models that result in coefficients which are easily explainable while providing a simple standardized method for comparisons between the coefficients. Typically, the task of creating indicator variables is often left to SAS procedures to process behind the scenes. Two downsides to this method is that the raw data may not be readily available for the programmer to view/work with and this method forces all indicators to be used in the procedure even if some of them are sparse or not otherwise desired.

An alternative solution may be to create manual static code with ‘if then’ or ‘case’ statements which could be very time consuming, error prone and all around painful to maintain. In many scenarios this solution is not even feasible given the large number of columns and categories of variables a data set could contain.

The solution to this challenge presented is a macro a macro called “auto_dummy” given at the end of this paper. It allows the user to control for the following as 3 parameters:

1. Maximum cardinality (# of levels or distinct values a variable has) to be a candidate for conversion
2. Convert only character type only or consider numeric as well
3. Leave out final Indicator per variable conversion

PROCESS OF CONVERTING CATEGORY “TYPE” VARIABLES TO INDICATORS

Note that the title states category “type” because a SAS variable that is classified as a categorical variable one might not consider it a category type variable, and likewise a numeric variable might really be regarded as a category type variable. For example, if a data set had a series of customer identifiers that contains values like “CUST0001” that would probably be considered more of a key than a category variable. As for numeric data as with observing profiles of automobiles a field like # of cylinders, although

technically numeric might be regarded as categorical as there are only a few distinct values one will find among all vehicles.

As for converting all category variables the process could be broken down like this:

1. Establish macro list of variables for potential conversion
2. Start loop for each variable in list
3. Calculate the cardinality.
4. If the cardinality is below the maximum cardinality then:
 - a) Append variables meeting criteria to macro variable list
 - b) Create an identity matrix data set of the distinct values (see Figure 1 for example)
 - c) Append data set with Variable names and Indicator Variable names to a master data set
5. Repeat loop (steps 2-4) for all variables in list
6. Prepare macro variables for hash table lookups
7. Merge original data set to all identity matrix data sets

Total rows: 6 Total columns: 7

Type	Type_Hybrid	Type_Sedan	Type_Sports	Type_SUV	Type_Truck	Type_Wagon
1 Hybrid	1	0	0	0	0	0
2 Sedan	0	1	0	0	0	0
3 Sports	0	0	1	0	0	0
4 SUV	0	0	0	1	0	0
5 Truck	0	0	0	0	1	0
6 Wagon	0	0	0	0	0	1

Figure 1. "Type" Identity table from sashelp.cars data set

ESTABLISH LIST OF POTENTIAL VARIABLES FOR CONVERSIONS

One method of retrieving variable names using SAS code is by querying the view "sashelp.vcolumn". by entering the library name (sashelp below) as well as the data set name (cars below) the view can be restricted to just those columns.

```
*PRODUCE LIST OF VARIABLES AS CANDIDATES FOR DUMMY CODING AS A GLOBAL VARIABLE;
```

```
PROC SQL NOPRINT;
  SELECT NAME INTO: VARS_TODUMMY
  SEPARATED BY " "
  FROM SASHELP.VCOLUMN
  WHERE MEMTYPE='DATA'
  AND LIBNAME=UPCASE("SASHELP")
  AND MEMNAME=UPCASE("CARS")
  AND UPCASE(TYPE)="CHAR";
QUIT;
```

By using the "into:" in conjunction with the "separated by" functionality a single macro variable is created called VARS_TODUMMY which resolves to: Make Model Type Origin DriveTrain

START LOOP & DETERMINE CARDINALITY CONDITION

The do loop to cycle through all potential variables for converting to indicators requires knowing the number variables there are. Fortunately with the use of the "%sysfunc()" most formulas can be executed

on data out side of data steps- and in this case allow the use of the “countw()” function which counts the number of words in the string.

```
%DO I=1 %TO %SYSFUNC (COUNTW (&VARS_TODUMMY.)) ;
```

By leveraging the scan function each individual word can be extracted out of the string according to the i-th index. This allows the following query to create a table (DISTVAR) of only distinct values for one variable within the iteration of the loop.

```
%LET CURVAR=%SCAN (&VARS_TODUMMY., &I) ;  
PROC SQL ;  
  CREATE TABLE DISTVAR AS SELECT DISTINCT &CURVAR.  
  FROM SASHELP.CARS  
  ORDER BY 1 ;  
QUIT ;
```

The cardinality for the specific variable can then be efficiently determined by extracting the size of DISTVAR. If this value is less then the parameter of CARDINALITY then the loop will continue on to produce the Identity matrix in the next step.

```
*CARDINALITY CALCULATED AS THE SIZE OF THE DATA SET ;  
DATA _NULL_ ;  
  IF 0 THEN SET WORK.DISTVAR NOBS=N ;  
  CALL SYMPUTX ('CARDINALITY', N) ;  
  STOP ;  
RUN ;  
  
%IF &CARDINALITY.<=&MAX_CARDINALITY. %THEN %DO ;
```

CREATE IDENTITY MATRICES

If a variable is found to have cardinality less than or equal to the parameter of &CARDINALITY then the variable name is appended to the macro VARLIST in order to keep track of all variables with low cardinality.

```
%LET VARLIST=&VARLIST. &CURVAR. ;
```

The first part of creating the identity matrix of the indicators variable is to create good names for the indicators. The methodology chosen combines the name of the variable being converted with an “_” and the individual values as “INDICATOR_NAME”:

```
DATA PREP ;  
  SET DISTVAR END=EOF ;  
  FORMAT INDICATOR_NAME $1000. ;  
  INDICATOR_NAME= COMPRESS ("&CURVAR._" || &CURVAR.) ;  
  X=1 ; *POPULATES INDICATOR VALUES IN PROC TRANSPOSE ;  
RUN ;
```

Total rows: 3 Total columns: 3

	Origin	INDICATOR_NAME	X
1	Asia	ORIGIN_Asia	1
2	Europe	ORIGIN_Europe	1
3	USA	ORIGIN_USA	1

Figure 2: “prep” table on variable "Origin" from sashelp.cars

```
PROC TRANSPOSE DATA=PREP OUT=&CURVAR. (DROP=_NAME_);
  ID INDICATOR_NAME;
  BY &CURVAR.;
RUN;
```

Total rows: 3 Total columns: 4

	Origin	ORIGIN_Asia	ORIGIN_Europe	ORIGIN_USA
1	Asia	1	.	.
2	Europe	.	1	.
3	USA	.	.	1

Figure 3: “Origin” Identity table from sashelp.cars data set

```
*MISSING VALUES NOT ON DIAGONAL REPLACED WITH ZEROS;
DATA &CURVAR.;
  SET &CURVAR.;
  ARRAY A{*} _NUMERIC_;
  DO I=1 TO DIM(A);
    IF A(I)=. THEN A(I)=0;
  END;
  DROP I;
RUN;
```

Total rows: 3 Total columns: 4

	Origin	Origin_Asia	Origin_Europe	Origin_USA
1	Asia	1	0	0
2	Europe	0	1	0
3	USA	0	0	1

Figure 4: “Origin” Identity Matrix table from sashelp.cars data set

PREPARE MACRO VARIABLES FOR HASH TABLES

Within the inner do loop of variables with low cardinality each prep table is appended to table “Indicator_labels”. Each set of records is indexed by the number of variables to be converted to indicator variables. This is used down stream to prepare inputs for combining the data sets using macro variable lists.

```
DATA INDICATOR_LABELS;
  SET INDICATOR_LABELS PREP(IN=P KEEP=INDICATOR_NAME PRE_LABEL);
  *ADDING CNTER TO INDEX SOURCE VARIABLES;
```

```

        IF P THEN CNTER=COUNTW("&VARLIST.");
RUN;

```

After the loop is completed the following loop is processed.

This returns the number of variables to be converted to Indicator variables:

```
%LET VAR_UPDATES=%SYSFUNC(COUNTW(&VARLIST.));
```

Macro variable list created to reference all indicators being appended to main data set:

```

PROC SQL NOPRINT;
    SELECT INDICATOR_NAME INTO:INDICATORS SEPARATED BY " "
    FROM INDICATOR_LABELS;
QUIT;

```

```
*CREATE QUOTED VARIABLES WITH INDEXES FOR REFERENCES IN HASH TABLES;
```

```

%DO I=1 %TO &VAR_UPDATES.;
    %LET V=%SCAN(&VARLIST.,&I.);
    DATA _NULL_;
        CALL
SYMPUTX(COMPRESS('INDICATOR_SOURCE_Q' || &I.), COMPRESS("'" || "&V." || "'"))
;
    RUN;

```

```

PROC SQL NOPRINT;
    SELECT COMPRESS("'" || INDICATOR_NAME || "'") INTO:SOURCE_VALUES&I.
    SEPARATED BY "," FROM INDICATOR_LABELS WHERE CNTER=&I.;
QUIT;
%END;

```

Using the sashelp.cars data set as an example each respective variable resolves as shown:

```

INDICATOR_SOURCE_Q1='Type'
INDICATOR_SOURCE_Q2='Origin'
SOURCE_VALUES1='Type_Hybrid', 'Type_SUV', ... 'Type_Truck', 'Type_Wagon'
SOURCE_VALUES2='Origin_Asia', 'Origin_Europe', 'Origin_USA'

```

These variables will be called in a do loop to be passed as arguments for hash tables when merging values. The Indicator source is both the same of the SAS data set to use as a hash table as well as the name of the key to join on in the upcoming example.

MERGE ALL IDENTITY MATRICES AS HASH TABLES

Once all identity matrix data set has been created for each variable to be converted, the original data set can be merged with each of those tables in order to create all of the necessary indicators and drop the original category variables being converted. The most efficient way to do this provided there is enough memory is to utilize the hash method in the data step. This is accomplished in a dynamic fashion by iterating each hash table as "h1,h2...etc." with corresponding inputs of the macro variables created in the preceding step.

```

DATA &DATA_OUT.;
    IF _N_=1 THEN DO;
        %DO I=1 %TO &VAR_UPDATES.;
            DECLARE HASH H&I.(DATASET: &&INDICATOR_SOURCE_Q&I.);
            H&I..DEFINEKEY(&&INDICATOR_SOURCE_Q&I.);
            H&I..DEFINEDATA(&&SOURCE_VALUES&I.);

```

```

H&I..DEFINEDONE ();
%END;

*CALL MISSING ELIMINATES UNNECESSARY NOTES IN THE LOG;
%DO I=1 %TO %SYSFUNC (COUNTW (&INDICATORS.)) ;
    CALL MISSING (%SCAN (&INDICATORS., &I.)) ;
%END;

END;

SET SASHELP.CARS;
FORMAT &INDICATORS. 8.0;

%DO I=1 %TO &VAR_UPDATES.;
    RC=H&I..FIND ();
%END;

DROP I RC &VARLIST;
RUN;

```

	Model	Origin	Origin_Asia	Origin_Europe	Origin_USA
1	3.5 RL 4dr	Asia	1	0	0
2	3.5 RL w/Navigation 4dr	Asia	1	0	0
3	300M 4dr	USA	0	0	1
4	300M Special Edition 4dr	USA	0	0	1
5	325Ci 2dr	Europe	0	1	0
6	325Ci convertible 2dr	Europe	0	1	0

Figure 5: Output data set with “Origin” Indicators added

DEALING WITH NUMERIC DATA

Prior to some technical update the macro as describe previously would not work on numeric data such as “Cylinders” in the sashelp.cars data set only if missing values were not present. Because the methodology places the value in the name of the indicator variable an error would be thrown because “.” is not a valid character for SAS variable names. To remedy this the following snippet was added to the prep data set to recode numeric values indicator names as_MISS.

```

*RECODE FOR NUMERIC MISSING VALUES;
IF FIND (INDICATOR_NAME, ".") > 0 THEN DO;
    INDICATOR_NAME = COMPRESS (COMPRESS (INDICATOR_NAME, ".", "") || "MISS");
    PRE_LABEL = INDICATOR_NAME;
END;

```

The second issue that would break the macro when trying to create indicator variables of numeric value is after the transpose data would accidentally set all key values to missing because the array was built using “_numeric_”. To fix this issue the indicator variable names for then identity matrix are extracted from the prep data set as macro “VALUES” and then references to replace off diagonal missing values with zero.

```

PROC SQL NOPRINT;
    SELECT INDICATOR_NAME INTO :VALUES SEPARATED BY " " FROM PREP;
QUIT;

DATA &CURVAR.;
    SET &CURVAR.;

```

```

ARRAY A{*} &VALUES.;
DO I=1 TO DIM(A);
    IF A(I)=. THEN A(I)=0;
END;
DROP I;
RUN;

```

PRODUCE NON-FULLRANK DATA

Usually when modeling data with indicator variables the data is chosen to leave one of the indicator variables out so that the meaning of every coefficient can be weights as the impact against the value left out. Models that have all indicator variables are called “over-parameterized” because the last indicator could always be constructed by all of the others – so there is more information than is needed. Furthermore the interpretation of the coefficients is not a straight forward.

As an extra parameter to the macro to control for this scenario all indicators except the last one for each converted variable can be left out if the ‘fullrank’ is set to ‘N’. The result is the last record of the prep data set is eliminated from the matrix prior to creating the identity matrix.

```

IF EOF THEN DO;
    %IF &FULLRANK.=N %THEN %DO;DELETE;%END;
END;

```

Total rows: 2 Total columns: 3

	Origin	Origin_Asia	Origin_Europe
1	Asia	1	0
2	Europe	0	1

Figure 6: Output data set with “Origin” Indicators added- Non-Fullrank

This method still had a problem prior to the final fix because when using hash tables to join to the original table there would be missing keys in the hash table. The fix to this is to rely on the return code if the key exist- and if the return code was anything other than 0 set the missing values of the array to zero.

```

ARRAY IND{*} &INDICATORS.;
%DO I=1 %TO &VAR_UPDATES.;
    RC=H&I..FIND();
    *IF KEY NOT FOUND IN HASH TABLE ALL INDICATORS SET TO ZERO;
    IF RC NE 0 THEN DO I=1 TO DIM(IND);
        IF IND(I)=. THEN IND(I)=0;
    END;
%END;

```

CONCLUSION

The use of macros to automate processes such as creating indicator variables is what makes the almost impossible possible. This proposed macro can be easily leveraged even with out understanding all the details as long as the following three parameters are understood:

CARDINALITY-only variables with the # of distinct values this much or less will be converted into indicator variables

CHARONLY-if set to 'Y' then only character type variables will be candidates for conversion

FULLRANK-if set to 'N' the last indicator of each converted variable is dropped since they can be re-created from the other indicator variables.

Here are the 4 types of scenarios demonstrated (be sure to define the macro prior to testing of course):

```
%AUTO_DUMMY(SASHELP.CARS, CARSOOT_CHAR_FR, 10, Y); *CHARACTER VARIABLES ONLY- FULL RANK;
%AUTO_DUMMY(SASHELP.CARS, CARSOOT_ALL_FR, 10); *ALL VARIABLES -FULL RANK;
%AUTO_DUMMY(SASHELP.CARS, CARSOOT_CHAR_NONFR, 10, Y, N); *CHARACTER VARIABLES ONLY-NOT FULL RANK;
%AUTO_DUMMY(SASHELP.CARS, CARSOOT_ALL_NONFR, 10, N, N); *ALL VARIABLES -NOT FULL RANK;
```

A final note that the auto_dummy macro as presented below has some variations and additions from what was presented above. The reason for the differences was to focus on the heart of what the process is doing without getting lost in the details the define the labels and properly identifies the library and data set name from the given data set reference.

```
%MACRO AUTO_DUMMY(DATA_IN, DATA_OUT, MAX_CARDINALITY, CHARONLY, FULLRANK);
*****
THE PURPOSE OF THIS MACRO IS TO AUTOMATICALLY REPLACE
EACH INDIVIDUAL VARIABLE IN A DATA SET WITH MULTIPLE
INDICATOR VARIABLES IF THE INDIVIDUAL VARIABLE HAS
CARDINALITY BELOW THE SPECIFIED VALUE
*****

AUTHOR:MATT BATES
LAST MODIFIED:12APR2018
CONTACT INFO: (740)963-2751

*CHARONLY=Y MEANS ONLY CHARACTER VARIABLES WILL BE REPLACED AS INDICATORS;
*FULLRANK=N MEANS LAST INDICATOR PER VARIABLE WILL BE DROPPED;
%MACRO EXTRACT_LIBNAME(D);
*****
THE PURPOSE OF THIS MACRO IS TO SEPARATE AND RETRIEVE
THE LIBRARY NAME FROM THE DATA SET NAME REGARDLESS
OF WHETHER OR NOT THE WORK LIBRARY IS EXPLICITLY LISTED
*****;

%GLOBAL LIBN DATA;
%LET POINT=%SYSFUNC(FIND(&DATA_IN., %STR(.)));
%IF &POINT.>0 %THEN %DO;
%LET LIBN=%SYSFUNC(SUBSTR(&DATA_IN., 1, %SYSFUNC(SUM(&POINT., -1))));
%LET DATA=%SYSFUNC(SUBSTR(&DATA_IN., %SYSFUNC(SUM(&POINT., 1))));
%END;
%ELSE %DO;
%LET LIBN=WORK;
%LET DATA=&DATA_IN.;
%END;
%MEND;
%EXTRACT_LIBNAME(&DATA_IN.);

*PRODUCE LIST OF VARIABLES AS CANDIDATES FOR DUMMY CODING AS A GLOBAL VARIABLE;
PROC SQL NOPRINT;
SELECT NAME INTO:VARS_TODUMMY SEPARATED BY " " FROM SASHELP.VCOLUMN
WHERE MEMTYPE='DATA' AND LIBNAME=UPCASE("&LIBN.") AND MEMNAME=UPCASE("&DATA.")
%IF &CHARONLY.=Y %THEN %DO; AND UPCASE(TYPE)="CHAR" %END;;
QUIT;
```

```

*DEFINE RETAINING LISTS FOR LOOPS;
%LET VARLIST=;
DATA INDICATOR_LABELS;
    FORMAT INDICATOR_NAME PRE_LABEL $1000.;
    STOP;
RUN;

*LOOP THROUGH VARIABLES IN &VARS_TODUMMY. LIST TO:
    1) DETERMINE CARDINALITY
    2) CREATE INDICATOR TABLES TO BE HASHED IF CARDINALITY IS LOW ENOUGH
    3) POPULATE INDICATOR_LABELS TABLE;
%DO I=1 %TO %SYSFUNC(COUNTW(&VARS_TODUMMY.));
    %LET CURVAR=%SCAN(&VARS_TODUMMY., &I);

    PROC SQL;
        CREATE TABLE DISTVAR AS SELECT DISTINCT &CURVAR. FROM &DATA_IN. ORDER BY 1;
    QUIT;

    *CARDINALITY CALCULATED AS THE SIZE OF THE DATA SET;
    DATA _NULL_;
        IF 0 THEN SET WORK.DISTVAR NOBS=N;
        CALL SYMPUTX('CARDINALITY', N);
        STOP;
    RUN;

    %IF &CARDINALITY.<=&MAX_CARDINALITY. %THEN %DO;
        %LET VARLIST=&VARLIST. &CURVAR.;

        DATA PREP (DROP=LEN);
            SET DISTVAR END=EOF;
            FORMAT INDICATOR_NAME PRE_LABEL $1000.;

            RETAIN LEN 1;
            PRE_LABEL="&CURVAR._"||TRIM(&CURVAR.);
            X=1;*POPULATE INDICATOR VALUES IN PROC TRANSPOSE;
            LEN=MAX(LEN, LENGTH(PRE_LABEL));
            INDICATOR_NAME=COMPRESS(PRE_LABEL);

            *RECODE FOR NUMERIC MISSING VALUES;
            IF FIND(INDICATOR_NAME, ".")>0 THEN DO;
                INDICATOR_NAME=COMPRESS(COMPRESS(INDICATOR_NAME, ".", "")||"MISS");
                PRE_LABEL=INDICATOR_NAME;
            END;

            IF EOF THEN DO;
                CALL SYMPUTX('LEN', LEN);
                %IF &FULLRANK.=N %THEN %DO; DELETE; %END;
            END;
        RUN;

        %IF &LEN.>32 %THEN %DO;
            *RECODES INDICATOR_NAME IF RISK OF EXCEEDING
            32 CHARACTER SAS NAME RESTRICTION.
            REPLACES VARIABLE VALUE WITH INDEX OF ORDER;
            DATA PREP;
                SET PREP;

            INDICATOR_NAME=COMPRESS(SUBSTR("&CURVAR.", 1, MIN(LENGTH("&CURVAR."), 25))||"_IND"||_N_);
            RUN;
        %END;

        DATA INDICATOR_LABELS;
            SET INDICATOR_LABELS PREP(IN=P KEEP=INDICATOR_NAME PRE_LABEL);
            IF P THEN CNTER=COUNTW("&VARLIST."); *ADDING CNTER TO INDEX SOURCE
VARIABLES;
        RUN;

        PROC TRANSPOSE DATA=PREP(KEEP=INDICATOR_NAME &CURVAR. X)
            OUT=&CURVAR.(DROP=_NAME_);
            ID INDICATOR_NAME;
            BY &CURVAR.;

```

```

RUN;

PROC SQL NOPRINT;
    SELECT INDICATOR_NAME INTO:VALUES SEPARATED BY " " FROM PREP;
QUIT;

DATA &CURVAR.;
    SET &CURVAR.;
    ARRAY A{*} &VALUES.;
    DO I=1 TO DIM(A);
        IF A(I)=. THEN A(I)=0;
    END;
    DROP I;
RUN;

%END;
%END;

%LET VAR_UPDATES=%SYSFUNC (COUNTW(&VARLIST.));

DATA INDICATOR_LABELS;
    SET INDICATOR_LABELS;
    FORMAT LAB $1000.;

    FIRST_UNDERSCORE=FIND(PRE_LABEL,"_");
    LAB=COMPRESS(INDICATOR_NAME||"="_);
        ||SUBSTR(INDICATOR_NAME,1,FIRST_UNDERSCORE-1)||" = "
        ||TRIM(TRANSLATE(SUBSTR(PRE_LABEL,FIRST_UNDERSCORE+1)," ","_"))
        ||" INDICATOR'";
    DROP FIRST_UNDERSCORE;
RUN;

PROC SQL NOPRINT;
    SELECT INDICATOR_NAME ,LAB
        INTO:INDICATORS SEPARATED BY " " ,:LAB SEPARATED BY " "
    FROM INDICATOR_LABELS;
QUIT;

*CREATE QUOTED VARIABLES WITH INDEXES FOR REFERENCES IN HASH TABLES;
%DO I=1 %TO &VAR_UPDATES.;
    %LET V=%SCAN(&VARLIST.,&I.);
    DATA _NULL_;
        CALL SYMPUTX(COMPRESS('INDICATOR_SOURCE_Q'||&I.),COMPRESS("'"||"&V."||"'"));
    RUN;

    PROC SQL NOPRINT;
        SELECT COMPRESS("'"||INDICATOR_NAME||"'") INTO:SOURCE_VALUES&I.
            SEPARATED BY "," FROM INDICATOR_LABELS WHERE COUNTER=&I.;
    QUIT;
%END;

DATA &DATA_OUT.;
    IF 0 THEN SET &DATA_IN. (OBS=0);
    FORMAT &INDICATORS. 8.0;
    LABEL &LAB.;
    IF _N_=1 THEN DO;
        %DO I=1 %TO &VAR_UPDATES.;
            DECLARE HASH H&I. (DATASET: &&INDICATOR_SOURCE_Q&I.);
            H&I..DEFINEKEY (&&INDICATOR_SOURCE_Q&I.);
            H&I..DEFINEDATA (&&SOURCE_VALUES&I.);
            H&I..DEFINEDONE ();
        %END;

        *CALL MISSING ELIMINATES UNNECESSARY NOTES IN THE LOG;
        %DO I=1 %TO %SYSFUNC (COUNTW(&INDICATORS.));
            CALL MISSING(%SCAN(&INDICATORS.,&I.));
        %END;
    END;

    SET &DATA_IN.;

    ARRAY IND{*} &INDICATORS.;

```

```
%DO I=1 %TO &VAR_UPDATES.;
  RC=H&I..FIND();
  *IF KEY NOT FOUND IN HASH TABLE ALL INDICATORS SET TO ZERO;
  IF RC NE 0 THEN      DO I=1 TO DIM(IND);
                      IF IND(I)=. THEN IND(I)=0;
  END;
%END;

DROP I RC &VARLIST.;
RUN;

PROC DELETE DATA=INDICATOR_LABELS PREP DISTVAR &VARLIST.;RUN;
%MEND;
```

ACKNOWLEDGMENTS

Special thanks to James Gearheart for proposing this challenge.

CONTACT INFORMATION

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

Matthew Bates
Affusion Consulting Inc
(740)963-2751
matt_d_bates@msn.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.