An Easy Route to a Missing Data Report with ODS+PROC FREQ+A Data Step

Mike Zdeb, University at Albany School of Public Health, Rensselaer, NY

ABSTRACT

A first step in analyzing data is making a decision on how to handle missing values. That decision could be deletion of observations and/or variables with excess missing data, substitution of imputed values for missing values, or taking no action at all if the amount of missing data is insignificant and not likely to affect the analysis. This paper shows how to use an ODS OUTPUT statement, PROC FREQ, and some data step programming to produce a missing data report showing the percentage of missing data for each variable in a data set. Also shown is a method for identifying and dropping from a data set all variables with either all or a high percentage of missing values. The method of producing the missing data report is less complicated and more compact than several methods already proposed in other papers.

INTRODUCTION

There are a number of different SAS® procedures that can produce counts of the amount of missing values in a data set, for example PROC FREQ, PROC TABULATE, PROC SQL, or PROC UNIVARIATE (for numeric variables only). PROC FREQ requires the least amount SAS code for producing counts and percentages of missing values for all the variables in a data set and will be the procedure used in all the examples. Example 1 uses the data set SASHELP.CLASS to create a small data set with some missing character and numeric data. That SAS code produces the data set shown on the right.

```
* EXAMPLE 1;
data class;
set sashelp.class;
if ranuni(987) le .5;
if ranuni(987) le .2 then
   call missing(weight);
if ranuni(987) le .1 then
   call missing(sex, age);
run;
```

Obs	Name	Sex	Age	Height	Weight
1	Alice		.	56.5	84.0
2	Jane	F	12	59.8	84.5
3	Janet	F	15	62.5	112.5
4	John	M	12	59.0	.
5	Joyce	F	11	51.3	50.5
6	Judy	F	14	64.3	90.0
7	Mary	F	15	66.5	112.0
8	Philip	M	16	72.0	.
9	Robert		.	64.8	.
10	Thomas	M	11	57.5	85.0

PROC FREQ only needs the name of a data set to produce a table of values for each variable in a data set ...

```
proc freq data=class;
run;
```

However, the default behavior of PROC FREQ is to ignore missing data. The above syntax will produce counts of missing values but not percentages since missing values are not included within tables as shown in the table of one variable located directly below the data set. Adding one line to PROC FREQ moves the missing values into tables ...

Sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	5	62.50	5	62.50
M	3	37.50	8	100.00

Frequency Missing = 2

```
proc freq data=class;
tables _all_ / missing;
run;
```

and the last table on the right shows both a count and percentage of missing data for one variable. Given a data set with both a small number of variables and a small number of different values for those variable, the above three lines of SAS code will produce a missing data report. However, even with a small number of variables, one or more of those variables might contain a large number different values. That would result in a large amount of output with the missing data information buried within large tables. What is needed is some way to group the non-missing values and that will be done with formats, one each for numeric and character variables.

Sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
	2	20.00	2	20.00
F	5	50.00	7	70.00
M	3	30.00	10	100.00

Example 2 creates the two formats, NM. for numeric variables and $CH. for character variables. Since SAS has twenty-seven different ways of representing missing numeric, a period plus .a through .z, you could use a range for missing numeric data in example 2 ...

```
* EXAMPLE 2;
proc format;
value nm
. = 'MISSING'
other = 'OK'
;
value $ch
' ' = 'MISSING'
other = 'OK'
;
run;
```

```
. - .z = 'MISSING'
```

However, the values .a through .z must be user-assigned to missing numeric data using an informat or some data step logic so the examples in this paper will assume that missing numeric values are represented in the default manner, by a period.

Using the formats as shown in example 3 condenses all the tables to just two entries, OK and MISSING. An example is shown on the right below example 3. If the data set contains a small number of variables, a collection of such tables might suffice as a missing data report. However, when the number of variables in a data set is large, even the condensed tables would require a lot of room. If the results of example 3 could be sent to a data set rather than to the display manager output window or a file, the information could be reformatted.

```
* EXAMPLE 3;
proc freq data=class;
tables _all_ / missing;
format _numeric_ nm.
       _character_ $ch.;
run;
```

Sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
MISSING	2	20.00	2	20.00
OK	8	80.00	10	100.00

One common way to direct the results of PROC FREQ to a data set is to use procedure options as shown in example 4. The NOPRINT option is used to suppress printed results and the OUT option directs the results to a data set name TABLES. Though example 4 produces a data set, that date set only contains a table of values for the last variable in the data set. PROC CONTENTS shows that variable is WEIGHT.

```
* EXAMPLE 4;
proc freq data=class;
tables _all_ / noprint missing
               out=tables;
format _numeric_ nm.
       _character_ $ch.;
run;
```

Printing data set TABLES gives the output shown below on the left. The values of WEIGHT look as intended, but what about the variables COUNT and PERCENT?

Remember in example 4, the format NM. was applied to all numeric variables. Thus, when you print the data set, you see formatted values for all variables in the table. If you add a statement to PROC PRINT (format _all_;) you instruct PROC PRINT to display the unformatted values of all the variables (shown below on the right), but you can see that this introduces another small issue. The values of the variable WEIGHT are now missing and the lowest value of WEIGHT in the data set (used to represent all the non-missing values in the data set).

Obs	Weight	COUNT	PERCENT
1	MISSING	OK	OK
2	OK	OK	OK

Obs	Weight	COUNT	PERCENT
1	.	3	30
2	50.5	7	70

ODS + PROC FREQ

ODS offers an alternative to procedure options for an easy way to produce a data set with tables of all variables in a data set. An ODS OUTPUT statement can be used to direct all or parts of the output of any procedure to a data set. The key piece of information is the ODS table name for PROC FREQ output of one-way tables (i.e. tables with only one variable). You can find that name by running PROC FREQ with the ODS TRACE option turned on and looking in the LOG for the ODS table name. You can also find that name in SAS online help and the name that ODS assigns to that output is ONEWAYFREQS.

One small change is made to the formats in example 6. Missing values are given a value of 0 and non-missing are grouped together with a value of 1. ODS statements are used to first close the LISTING destination and then to direct the various tables to a data set named TABLES. Note that using the procedure option NOPRINT to suppress output rather than an ODS statement to close the LISTING destination will result in this note in the log ...

NOTE: There are no valid requests for output data sets or printed output, so processing will terminate.

plus a much longer warning message that includes ...

For example, verify that the NOPRINT option is not used.

After closing the OUTPUT destination and reopening the LISTING destination, the data set TABLES is printed suppressing all the formats and it is shown below. In contrast to when the procedure option was used to produce a data set, this data set contains a table for each variable in the order that the variables appear in the data set CLASS.

```
* EXAMPLE 5;
proc format;
value nm    . = '0' other = '1';
value $ch ' ' = '0' other = '1';
run;

ods listing close;
ods output onewayfreqs=tables;

proc freq data=class;
tables _all_ / missing;
               format _numeric_ nm.
              _character_ $ch.;
run;

ods output close;
ods listing;

proc print data=tables;
format _all_;
run;
```

Obs	Table	F_Name	Name	Frequency	Percent	CumFrequency	CumPercent	F_Sex	Sex	F_Age	Age	F_Height	Height	F_Weight	Weight
1	Table Name	1	Alice	10	100	10	100				.		.		.
2	Table Sex			2	20	2	20	0			.		.		.
3	Table Sex			8	80	10	100	1	F		.		.		.
4	Table Age			2	20	2	20			0	.		.		.
5	Table Age			8	80	10	100			1	11		.		.
6	Table Height			10	100	10	100				.	1	51.3		.
7	Table Weight			3	30	3	30				.		.	0	.
8	Table Weight			7	70	10	100				.		.	1	50.5

Also notice: though we are looking at unformatted values of all the variables, the formatted values 0 and 1 also appear for each variable with a new name comprising an "F_" plus the variable name; the only variables that are needed for a missing data report are TABLE, all "F_" variables, Frequency, Percent.

Thus, example 6 can be modified with a KEEP data set option in the first ODS OUTPUT statement ...

ods output onewayfreqs=tables (keep=table f_: frequency percent);

PROC CONTENTS shows that a data set produced with that KEEP data set option contains the variables shown on the right with their attributes. Notice that all the "F_" variables are character. This data set will be used to produce a missing data report.

#	Variable	Type	Len	Format	Label
6	F_Age	Char	1		Age
7	F_Height	Char	1		Height
2	F_Name	Char	1		Name
5	F_Sex	Char	1		Sex
8	F_Weight	Char	1		Weight
3	Frequency	Num	8	BEST8.	
4	Percent	Num	8	6.2	
1	Table	Char	256		

ODS+PROC FREQ+A DATA STEP
The final data step that produces the missing data report is shown in example 6. A LENGTH statement assigns a length of 32 to the variable VAR (the maximum length of a SAS variable name) ①. Since the variable VAR is created using the SCAN function later in the data step, without the LENGTH statement, the length of VAR would be 200 ... the default when a variable is created with the SCAN function.

A DoW loop (see the references) is used to read the data set TABLES one variable at a time and to create new variables used in the report ②. The combination of the statements ...

```
do until (last.table); ②
```

and ...

```
set tables; ③
by variables notsorted;
```

processes observations in data set TABLES in groups based on the value of the variable TABLE. Look at the data set TABLE on the previous page. There are no missing values for the variable NAME so there is only one observation for that variable (the same is true for HEIGHT). For each variable with both missing and non-missing values there are two observations.

In data set TABLE you will also see that the information that tells us that we are looking at counts and percentages for missing or non-missing values is located in the "F_" variables. The appropriate 0/1 values for those variables are found by setting up an array NAMES ③. The first pass through the data step uses the variable F_NAME (the first element in the array NAMES) in the SELECT statement ④. Each time a pass is made through the data step, the variable _N_ is incremented by 1 and subsequent "F_" variables are used. The SELECT statement assigns the values of FREQUENCY and PERCENT as missing or non-missing based on the value of the "F_" variable ⑤.

```
* EXAMPLE 6;
data report;
length var $32; ①

do until (last.table); ②
   set tables; ③
   by table notsorted;
   array names(*) f_: ;
   select (names(_n_)); ④
     when ('0') do; ⑤
        miss   = frequency;
        p_miss = percent;
     end;
     when ('1') do; ⑤
        ok   = frequency;
        p_ok = percent;
     end;
   end;
end;

miss   = coalesce(miss,0); ⑥
ok     = coalesce(ok,0);
p_miss = coalesce(p_miss,0);
p_ok   = coalesce(p_ok,0);

var    = scan(table,-1); ⑦

keep var miss ok p_: ; ⑧
format miss ok comma7. p_: 5.1;
label
miss   = 'N_MISSING'
ok     = 'N_OK'
p_miss = '%_MISSING'
p_ok   = '%_OK'
var    = 'VARIABLE'
;
run;
```

The first pass through the data step only assigns values to the variables OK and P_OK since there are no missing values for name. Thus, MISS and P_MISS will have missing values, not zero. A series of COALESCE statements is used to change the value of variables to zero if the value is missing ⑥. Note that when a variable with both missing and non-missing values is processed, the COALESCE statements make no changes to the variables. The name of the variable is extracted from the variable TABLE using the SCAN function, taking the first word starting from the right ⑦. The remaining statements do some data set "housekeeping": only necessary variables are kept; formats and labels are assigned to variables ⑧.

Data set REPORT is shown on the right. Notice that though there are no zero counts and percentages for NAME and HEIGHT in the data set TABLES, the COALESCE statements in example 6 successfully replaced the missing values for those counts and percentages with zeros. Hopefully you have noticed that examples 5 and 6 are not specific to any data set. We should be able to create a missing data report for any data set using the SAS code in those examples.

VARIABLE	N_MISSING	%_MISSING	N_OK	%_OK
Name	0	0.0	10	100.0
Sex	2	20.0	8	80.0
Age	2	20.0	8	80.0
Height	0	0.0	10	100.0
Weight	3	30.0	7	70.0

The data set SASHELP.HEART was used in place of data set class and the combination of the SAS code in examples 5 and 6 produced the report shown on the right. The only change made was to replace the name of the data set in PROC FREQ in example 5.

VARIABLE	N_MISSING	%_MISSING	N_OK	%_OK
Status	0	0.0	5,209	100.0
DeathCause	3,218	61.8	1,991	38.2
AgeCHDdiag	3,760	72.2	1,449	27.8
Sex	0	0.0	5,209	100.0
AgeAtStart	0	0.0	5,209	100.0
Height	6	0.1	5,203	99.9
Weight	6	0.1	5,203	99.9
Diastolic	0	0.0	5,209	100.0
Systolic	0	0.0	5,209	100.0
MRW	6	0.1	5,203	99.9
Smoking	36	0.7	5,173	99.3
AgeAtDeath	3,218	61.8	1,991	38.2
Cholesterol	152	2.9	5,057	97.1
Chol_Status	152	2.9	5,057	97.1
BP_Status	0	0.0	5,209	100.0
Weight_Status	6	0.1	5,203	99.9
Smoking_Status	36	0.7	5,173	99.3

Variable labels can be added to the missing data report without to much extra SAS code. The first step is to place the variable names and labels into a data set. We will continue to use SASHELP.HEART since a number of the variables have labels ...

```
proc contents data=sashelp.heart
  noprint out=labels
  (keep=name label
   rename=(name=var)
   index=(var));
run;
```

Notice that the data set has an index and the variable used as the index (VAR) matches the name of the variable in example 6 that contains the names of data set variables.

Near the lower portion of example 6: a statement is added to read the label from the data set LABELS; LABEL is added to the KEEP statement; a label is assigned to the variable LABEL ...

```
var    = scan(table,-1);
* new statement;
set labels key=var/unique;
keep var label miss ok p_: ;
format miss ok comma7.
       p_: 5.1;
label
miss   = 'N_MISSING'
ok     = 'N_OK'
p_miss = '%_MISSING'
p_ok   = '%_OK'
var    = 'VARIABLE'
label  = 'LABEL'
;
```

VARIABLE	LABEL	N_MISSING	%_MISSING	N_OK	%_OK
Status		0	0.0	5,209	100.0
DeathCause	Cause of Death	3,218	61.8	1,991	38.2
AgeCHDdiag	Age CHD Diagnosed	3,760	72.2	1,449	27.8
Sex		0	0.0	5,209	100.0
AgeAtStart	Age at Start	0	0.0	5,209	100.0
Height		6	0.1	5,203	99.9
Weight		6	0.1	5,203	99.9
Diastolic		0	0.0	5,209	100.0
Systolic		0	0.0	5,209	100.0
MRW	Metropolitan Relative Weight	6	0.1	5,203	99.9
Smoking		36	0.7	5,173	99.3
AgeAtDeath	Age at Death	3,218	61.8	1,991	38.2
Cholesterol		152	2.9	5,057	97.1
Chol_Status	Cholesterol Status	152	2.9	5,057	97.1
BP_Status	Blood Pressure Status	0	0.0	5,209	100.0
Weight_Status	Weight Status	6	0.1	5,203	99.9
Smoking_Status	Smoking Status	36	0.7	5,173	99.3

PROC CONTENTS assigns a label ("Variable Label") to LABEL. That is why that variable is added to the LABEL statement. The new report on the right shows that not all variables had labels, but now you know that "MRW" stands for "Metropolitan Relative Weight" (now you have to find out what that means !).

There is another way to add the label to the report. Rather than use PROC CONTENTS to create data set LABELS and a SET statement in the data step to read the data set LABELS, the VLABELX function can be used ...

```
var    = scan(table,-1);
* new statement;
label  = vlabelx(var);
```

However, that requires removing the KEEP data set option shown on the bottom of page 3 changing ...

```
ods output onewayfreqs=tables (keep=table f_: frequency percent);
```

to ...

```
ods output onewayfreqs=tables;
```

Using VLABELX does cause a change in the output since if a variable has no label, the VLABELX function returns the name of the variable. Thus, every variable in the report will look as if it has a label even if one did not exist. If you use PROC CONTENTS to produce data set LABELS and a SET statement with an indexed read to find the label for each variable, variables with no label in the data set have a blank label in the missing data report.

It would be quite easy to combine the SAS code in examples 5 and 6 plus the extra statements for label addition into a missing data report macro since use from one data set to the next only requires changing one parameter, the name of the data set involved in the report. An example of such a macro is shown in the appendix B.

MISSING DATA PATTERNS

PROC MI is a SAS/STAT procedure used to impute values for missing data. However, it can also be used to produce a report showing the pattern in missing data values. Example 7 uses the data set CLASS shown on page 1. An ODS SELECT statement specifies that the only output should be a report on the pattern of missing values.

```
*EXAMPLE 7;
proc mi data=class;
ods select misspattern;
run;
```

The output from example 7, below on the left, shows that only numeric variables are in the report. There are six observations with no missing numeric values (Group 1). The other three groups have either one or two missing values in the patterns. There is no way to suppress the group means.

PROC FREQ with the LIST option on the TABLE statement can produce the similar results ...

```
proc format;
value nm    . = '.' other = 'X';
value $ch ' ' = '.' other = 'X';
run;

proc freq data=class;
table Name*Sex*Age*Height*Weight / list missing nocum;
format _numeric_ nm. _character_ $ch.;
run;
```

The results, below on the right, show that all variables are included and there are no group means. The order of the patterns is different from the PROC MI but the information is the same.

Missing Data Patterns

						Group Means		
Group	Age	Height	Weight	Freq	Percent	Age	Height	Weight
1	X	X	X	6	60.00	13.000000	60.316667	89.083333
2	X	X	.	2	20.00	14.000000	65.500000	.
3	.	X	X	1	10.00	.	56.500000	84.000000
4	.	X	.	1	10.00	.	64.800000	.

Name	Sex	Age	Height	Weight	Frequency	Percent
X	.	.	X	.	1	10.00
X	.	.	X	X	1	10.00
X	X	X	X	.	2	20.00
X	X	X	X	X	6	60.00

Since asterisks are required between the variable names, the shortcut _all_ cannot be used in the TABLE statement. When there are only a few variables, writing that table statement is not too arduous. However, if there are many variables in the data set, one can use PROC SQL to create the list of variables in a macro variables and then use that macro variable in PROC FREQ ...

```
proc sql noprint;
select name into :vars separated by '*'
from dictionary.columns
where libname eq 'WORK' and memname eq 'CLASS' ;
quit;
```

then ...

```
proc freq data=class;
tables &vars / list missing nocum out=miss;
format _numeric_ nm. _character_ $ch.;
run;
```

ELIMINATING VARIABLES WITH MISSING DATA

The information in a missing data report can be used for a task such as dropping variables that have all missing values or a given percentage of observations with missing values. We will again use the data set SASHELP.HEART. The variable P_MISS in data set REPORT contains the percentage of missing data for each variable in the data set.

In example 8, PROC SQL is used to create a macro variable named &droplist that lists all the variables in data set SASHELP.HEART that have 50%+ observations with missing data. That macro variable is used in a DROP data set option to create the new data set HEART, eliminating all variables with 50%+ missing data.

```
*EXAMPLE 8;
proc sql noprint;
select var into :droplist separated by ' '
from report
where p_miss ge 50;
quit;

data heart;
set sashelp.heart(drop=&droplist);
run;
```

If the task is to eliminate observations with all missing data, there is another approach that uses and NLEVELS option in PROC FREQ. That approach is shown in the appendix C.

REFERENCES

You can read about the DoW loop in the following ...

The DOW-Loop Unrolled ... http://support.sas.com/resources/papers/proceedings09/038-2009.pdf

ACKNOWLEDGMENTS

This work was funded in part by NIH grant HHSN267200700019C from the Eunice Kennedy Shriver National Institute of Child Health and Human Development

TRADEMARK CITATION

CONTACT INFORMATION

The author can be contacted using e-mail... Mike Zdeb msz03@albany.edu

APPENDIX A: APPEARANCE OF SAS OUTPUT

The output from various procedures used in this paper was produced using the following ODS statements ...

```
ods listing close;
ods html file = <destination for output.html> style=barrettsblue;

<procedure code>

ods html close;
ods listing;
```

The contrasting color in the output at the bottom of pager 5 was produced by using the variables VAR and LABEL in an ID statement in PROC PRINT ...

```
proc print data=report label noobs;
id var label;
run;
```

APPENDIX B: MISSING DATA REPORT MACRO

The SAS code in examples 5 and 6 plus the extra code for adding labels to the report can be assembled into a missing data report macro that should work with any data set.

```
%macro miss_report(din, fout, label=no);
* formats to group observations;
proc format;
value nm   .  = '0' other = '1';
value $ch ' ' = '0' other = '1';
run;

* PROC FREQ produces data set TABLES with counts of missing/non-missing values;
ods listing close;
ods output onewayfreqs=tables;
proc freq data=&din;
tables _all_ / missing;
format _numeric_ nm. _character_ $ch.;
run;
ods output close;
ods listing;

* place variable labels in a data set;
proc contents data=&din noprint
              out=labels (keep=name label rename=(name=var) index=(var));
run;

* create a macro variable &LABEL_EXIST ... value 1 if there are labels;
data _null_;
length check $5000;
set labels end=last;
check = cats(check,label);
if last then call symputx('label_exist',(lengthn(check) gt 0));
run;
```

```
* create the missing data report as a data set;
data report;
length var $32;
do until (last.table);
   set tables;
   by table notsorted;
   array names(*) f_: ;
   select (names(_n_));
     when ('0') do; miss = frequency; p_miss = percent; end;
     when ('1') do; ok   = frequency; p_ok   = percent; end;
   end;
end;
miss   = coalesce(miss,0);
ok     = coalesce(ok,0);
p_miss = coalesce(p_miss,0);
p_ok   = coalesce(p_ok,0);
var    = scan(table,-1);

set labels key=var/unique;
keep var label miss ok p_:;
format miss ok comma7. p_: 5.1;
label
miss   = 'N_MISSING'
ok     = 'N_OK'
p_miss = '%_MISSING'
p_ok   = '%_OK'
var    = 'VARIABLE'
label  = 'LABEL'
;
run;

* print the report ... output directed to an HTML file;
ods listing close;
ods html file="&fout" style=barrettsblue;

proc print data=report label noobs;
%if &label ne no and &label_exist ne 0 %then %do;
    id var label;
    var miss p_miss ok p_ok;
%end;
%else %do;
    id var;
    var miss p_miss ok p_ok;
%end;
run;

ods html close;
ods listing;
%mend;
```

Some examples using data sets in the SASHELP library ...

```
%miss_report(sashelp.class, z:\class.html);
%miss_report(sashelp.heart, z:\heart.html, label=yes)
%miss_report(sashelp.zipcode, z:\zicode.html, label=yes);
```

APPENDIX C: USING NLEVELS TO ELIMINATE VARIABLES WITH 100% MISSING VALUES

A small data set is created in example C1. All observations for three variables (GENDER, HEIGHT, WEIGHT) have missing values. Notice that some special numeric missing values are used.

```
* EXAMPLE C1;
data test;
input name : $10. gender : $1.
      age height weight;
datalines;
MIKE . 21 . .L 200
MARY . .Z . .H 120
MARK . 45 . .H 110
;
run;
```

In example C2, formats are created that can be used to group both numeric and character values into two groups, missing and OK ①. ODS statements are used to first close the LISTING destination and then to direct output from PROC FREQ (NLEVELS) to a data set named TABLES ②.

```
* EXAMPLE C2;
proc format; ①
value nm  low - high = 'OK'
          other      = .;
value $ch ' '        = ' '
          other      = 'OK';
run;

ods listing close; ②
ods output nlevels=tables;

proc freq data=test nlevels; ③
format _numeric_ nm. ④
       _character_ $ch.;
run;

ods output close; ⑤
ods listing;

data tables; ⑥
retain nnonmisslevels -1;
set tables;
run;

proc sql noprint; ⑦
select tablevar into :droplist
separated by ' '
from tables
where nnonmisslevels eq 0;
quit;

data new;
set test (drop=&droplist); ⑧
run;
```

The NLEVELS option is specified in PROC FREQ ③ and the previously defined formats are used so there are only two values produced in the table for each variable ④. ODS statements are used to close the OUTPUT destination and reopen the LISTING destination.

What is an NLEVELS table? PROC FREQ in example C2 produced a data set named TABLES that looks as follows ...

TableVar	NLevels	NMissLevels	NNonMissLevels
name	1	0	1
gender	1	1	0
age	2	1	1
height	1	1	0
weight	1	1	0

Notice that any variable with all missing values produces a value of zero for the variable NNONMISSLEVELS. However, if there were no variables in data set TEST with missing values, the only two variables in data set TABLES would be TABLEVAR and NLEVELS. The RETAIN statement in the data step that follows PROC FREQ in example C2 insures that data set TABLES will always contain the variable NNONMISSLEVELS ⑥.

As was done in example 8 earlier in the paper, PROC SQL is used to create a macro variable named &droplist ⑦. In example C2, that macro variable is to contain the names of all variables with 100% missing values (variables with a value of zero for NNONMISSLEVELS).

The last data step uses the macro variable in DROP data set option to eliminate the variables GENDER, HEIGHT, and WEIGHT from the data set ⑧.

I would like to acknowledge the help of John King whose posting on SAS-L about using the NLEVELS option in PROC FREQ is the basis for this example ...

http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0810C&L=sas-l&P=R32251

Thanks John.

You can read more about NLEVELS at ... http://support.sas.com/kb/30/867.html