Paper SA-08 - 2015

Are You Missing Out? Working with Missing Values to Make the Most of What is not There

Arthur L. Carpenter, California Occidental Consultants

ABSTRACT

Everyone uses and works with missing values, however many SAS® programmers are unaware of the variety of tools, options, and techniques associated with using missing values. Did you know that there are 28 types of numeric missing values? Did you know that the numeric missing value (.) is neither the smallest or largest possible numeric missing value? Are you aware of the System options, DATA step functions, and DATA step routines that specifically deal with missing values? Do you understand how the macro null value is the same, and different from DATA step missing values? Are you aware that observations with missing classification variables may or may not be excluded from analyses depending on the procedure and various options?

This paper explores various aspects of the world of missing values. The above questions and others are discussed. Learn more about missing values and make sure that you are not missing out.

KEYWORDS

Missing Value, CALL MISSING, MISSING Function, MISSING System option, MISSTEXT, null value

INTRODUCTION

In the SAS programming language when a data value is unknown we mark it as such using what is known as a missing value. There are two basic types of missing values; one for numeric variables, and one character variables. Character missing values are designated using a blank
(‘ ‘) and numeric missing values are typically designated with a dot (.). As placeholders for unknown values, missing values are automatically taken into consideration (are not used) when calculating statistics and for levels of classification variables.

```
x=.;
wrong='.';
name=' ';
```

In assignment statements, missing values can be designated by specifying a . for numeric variables, and a blank for character variables. For numeric variables the dot (.) should not be quoted [see the assignment statement for the variable WRONG in the box to the right], as this will result in either a character to numeric conversion or the creation of a character variable.

Arithmetically numeric missing values are essentially minus infinity (*i.e.* super small). Missing values are therefore smaller than any non-missing value, and any arithmetic operations performed on a missing value will result in a missing value. Most numeric functions that operate on lists of values, such as the MEAN and STDERR functions, will automatically ignore missing values.

There are a number of functions and options that have been specifically created for working with missing values. In addition a number of procedures have internal methods for handling missing values in various situations.

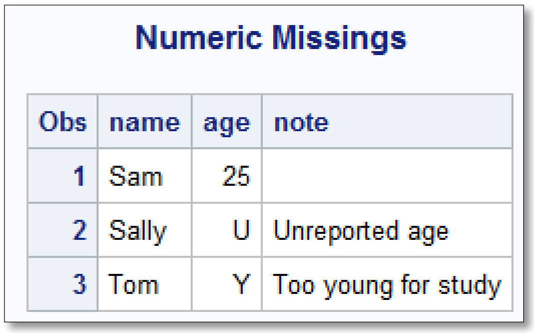
MORE ON NUMERIC MISSINGS

Although we usually think of a period (.) as the only symbol used for a numeric missing value, there are actually 28 different numeric missing values. In addition to the period, which is most commonly used, numeric missing values can also be designated by preceding each of the 26 letters of the alphabet (a through z) as well as the underscore with a period. These different values can then be used to distinguish between different kinds of missing values, such as a dropped sample as opposed to a sample that was not taken.

These special missing values can be directly coded on incoming data by placing a dot in front of the letter to be used to indicate the missing value. You can use the same notation to indicate the special missing value in your code. Here the IF expressions are testing for the three potential types of missing values of the numeric variable AGE.

```
data patients;
input name $ age;
if age=.y then note='Too young for study';
else if age=.o then note='Too old for study';
else if age=.u then note='Unreported age';
datalines;
Sam 25
Sally .u
Tom .y
run;
```

When a PROC PRINT is executed for the data set PATIENTS we see the two special missing values capitalized and without the dot.



Numeric Missings

Obs	name	age	note
1	Sam	25	
2	Sally	U	Unreported age
3	Tom	Y	Too young for study

If the MISSING statement is used, the incoming values do not have to have the preceding dot in the data. Otherwise the remainder of the code including the output of a procedure such as PROC PRINT is the same.

```
data patients;
missing o u y;
input name $ age;
if age=.y then note='Too young for study';
else if age=.o then note='Too old for study';
else if age=.u then note='Unreported age';
datalines;
Sam 25
Sally u
Tom y
run;
```

Including the use of letters and the underscore there are 28 different possibilities for numeric missing values. Although all essentially take on a value of minus infinity, there is a hierarchy associated with the 28 numeric missing values. This hierarchy can become critical when comparisons between them are made. In terms of size (sort order) the traditional missing value (.) is neither the smallest nor the largest of the 28 types of numeric missing values. The ._ is smallest, and is the only missing value smaller than (.). The largest numeric missing value is .z.

Suppose we want to subset for all valid dates in a data set. The WHERE clause or subsetting IF statement might be written as `where date > .;` . However, this expression would only eliminate two of the 28 potential types of numeric missing values (. and ._). The other 26 are greater than . and would be included. In order to guarantee that all numeric missing values are eliminated, the expression should be written as `where date > .z;` . Conversely, if you are searching for the smallest numeric value, (._) is smaller than the traditional missing (.).

FUNCTIONS AND ROUTINES

While some functions can accept missing values as arguments and still return the correct value, there are a number of functions and routines that have been written to specifically work with missing values.

Taking Missing into Consideration

Since arithmetic calculations involving missing values always result in a missing value, functions that receive a variable list of numeric values in order to calculate a result like a statistic, including functions such as MEAN, MEDIAN, STD, ignore missing values and return a value as if the missing value had not been included.   This allows us to calculate a statistic without first determining which if any of the arguments are missing.

```
data tryit;
array list {3} a b c (2 . 4);
n_val1 = n(a,b,c);
n_val2 = n(of list{*});
mean_val1 = mean(of a b c);
mean_val2 = (a+b+c)/3;
put (_all_) (=);
run;
```

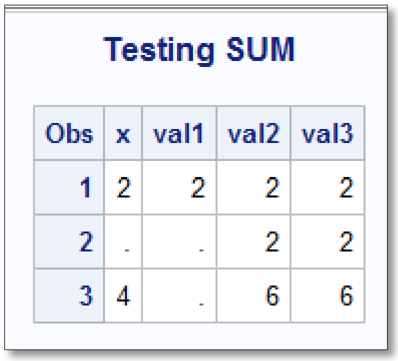
The three variables A, B, and C, in the example to the left have been given values, however B is missing.   The LOG shows that the N and MEAN function have correctly taken the missing value into consideration, however the equation that calculates the mean directly (MEAN_VAL2) results in a missing value.

```
a=2 b=. c=4 n_val1=2 n_val2=2 mean_val1=3 mean_val2=.
```

Unlike an assignment statement, which results in a missing value when operating on a missing value, the SUM statement does not.   This is because the SUM statement calls the SUM function behind the scenes, and like the MEAN and N functions shown above, the SUM function will ignore missing values when calculating a total.

```
data testsum;
input x;
retain val1 val2 0;
val1 = val1 + x;
val2 = sum(val2,x);
val3 + x;
datalines;
2
.
4
run;
```

In this example the structure of the expression that calculates VAL2 is most like the way that the DATA step handles the SUM statement used to calculate VAL3.

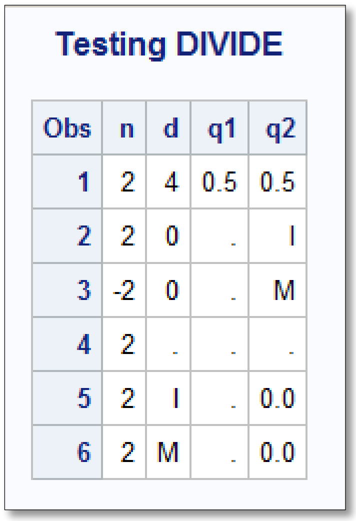


Testing SUM

Obs	x	val1	val2	val3
1	2	2	2	2
2	.	.	2	2
3	4	.	6	6

The DIVIDE function performs division, but is designed to both accept and correctly handle resultant missing values.   This includes division by zero as well as noting the differences between having a missing value in the numerator as opposed to being in the denominator. It does this by returning different types of missing values. These include: .I (positive infinity), .M (minus infinity), and ._ (either the numerator or the denominator contains a ._).

```
data showdivide;
input n d;
q1 = n / d;
q2 = divide(n,d);
datalines;
2 4
2 0
-2 0
2 .
2 .i
2 .m
run;
```



Testing DIVIDE

Obs	n	d	q1	q2
1	2	4	0.5	0.5
2	2	0	.	I
3	-2	0	.	M
4	2	.	.	.
5	2	I	.	0.0
6	2	M	.	0.0

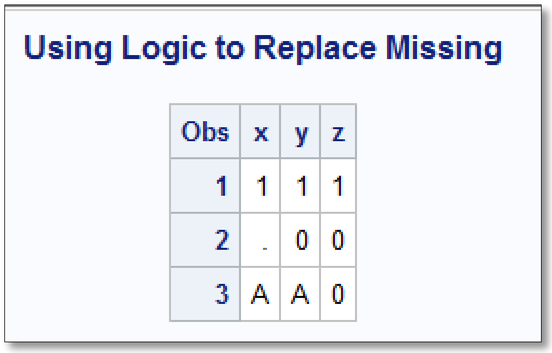
Replacing Missing Values

The MISSING system option and TABULATE's MISSTEXT option (both described below) can change the way a missing value is displayed, but neither will replace the missing value with another value *in the data*. Because they tend to be less efficient, we should try to avoid when possible the use of IF-THEN/ELSE statements. Fortunately there are a number of alternatives.

In this example IF-THEN/ELSE is used to detect a missing value. For demonstration purposes new variables are created here, but we could just have easily replaced the value of X.

```
data logic;
input x;
if x=. then y = 0; else y=x;
if x < .z then z=0; else z=x;
datalines;
1
.
.a
run;
```

Notice that we do not get the same answer for the two logical expressions. If there is a possibility that non-standard missing values might be used, it is *NOT* sufficient to use X=. as the logical expression.



Using Logic to Replace Missing

Obs	x	y	z
1	1	1	1
2	.	0	0
3	A	A	0

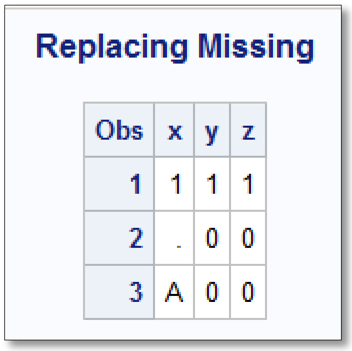
The IFN and IFC functions are ideally suited for this type of logical assignment. IFN returns numeric values and IFC returns character values. The first argument of both of these functions is a logical expression. If that expression is true the value of the second argument is returned, if false the value of the third argument, and if it evaluates to missing the fourth argument is returned. The code shown here will assign all numeric missing values of X to 0.

```
x = ifn(x,x,x,0);
```

The COALESCE function is designed for the task of replacing missing values with another value. The COALESCE function returns the first non-missing value that it encounters in its list of arguments. In this example Y will take on the value of zero whenever X is missing. Of course you could replace the missing with any other value, not just zero.

```
data replace;
input x;
y = coalesce(x,0);
z = sum(x,0);
datalines;
1
.
.a
run;
```

When you do want to replace the missing with zero , you could also use the SUM function as shown here. Missing values are ignored and non-missing values will not be altered.

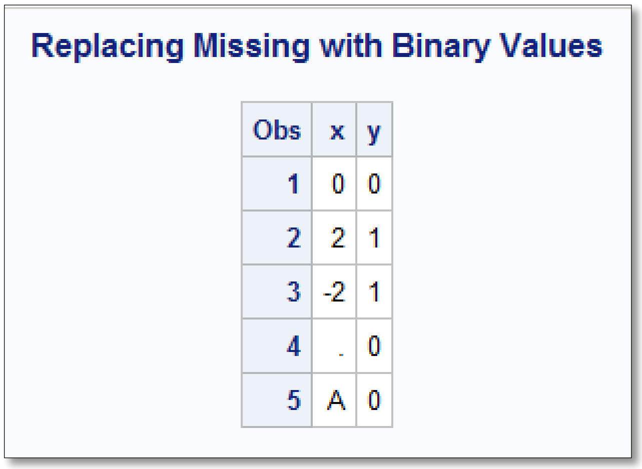


Replacing Missing

Obs	x	y	z
1	1	1	1
2	.	0	0
3	A	0	0

If you need to construct a binary (0,1) value based on your incoming data you can take advantage of the fact that SAS considers false to be 0 and true to be 1. Missing values are considered to be false, therefore we can easily convert missing values to 0 (false). This is most easily accomplished by using a double logical negation.

```
data binary;
input x;
y = ^^x;
datalines;
0
2
-2
.
.a
run;
```



Replacing Missing with Binary Values

Obs	x	y
1	0	0
2	2	1
3	-2	1
4	.	0
5	A	0

Detecting Missing Values - MISSING

The MISSING function determines if the argument contains a missing value (numeric or character). This function returns a 0 (not missing) or a 1 (missing), and since it can be used on either a numeric or character value, it can be used when a variable type is unknown such as after a PROC TRANSPOSE. In this example the DO block will be executed if the variable VAR3 is not missing.

```
if missing(var3) then do;
```

The MISSING function anticipates a single argument, however it is not unusual to see programmers incorrectly use it with a list of variables. Two or more comma separated variables will cause an error, however if connected with an AND or OR, as it is here, the argument is seen as a logical expression. If both variables are numeric the result will be either TRUE or FALSE, 1 or 0, either way the MISSING function will return a 0. If either X or Y is character the expression will be incorrect syntactically since a comparison operator is not present.

```
ncnt=missing(x or y);
```

Counting Missing Values

The CMISS and NMISS functions can be used to count the number of missing values in a list of values. Each of these functions can have from one to as many arguments as needed. NMISS counts the occurrences of any of the 28 types of numeric missing values (character values are converted to numeric first). CMISS counts character and numeric missing values (this includes values with zero length). Both functions will return a zero if no missing values are found. In the PROC PRINT shown here, CMISS is used in a WHERE clause to cause PRINT to list only those observations with one or more missing values in the selected variables.

```
proc print data=missdemo.clinics;
var symp diag proced;
where cmiss(symp, diag, proced);
run;
```

In the following example these two functions are used together to return the total number of missing values (numeric and character) on the PDV. The variable list abbreviations, _NUMERIC_ and _CHARACTER_, are used to list all variables of each type. In these function calls a non-missing constant has also been included to make sure that each function has at least one argument. Of course since TOTMISS is numeric, there will always be at least one numeric variable.

```
retain totmiss 0;
totmiss = nmiss(of _numeric_,1) + cmiss(of _character_,'a');
```

The NMISS function expects numeric arguments, however since the CMISS function will handle either numeric or character arguments. The previous calculation of TOTMISS could be simplified using only CMISS.

```
retain totmiss2 0;
totmiss2 = cmiss(of _all_);
```

Setting values to Missing

While the MISSING function determines if a value is missing, the CALL MISSING routine can be used to set one or more values to missing. The arguments to this routine can be numeric, character, or both, and when preceded with the keyword OF the argument can be an array call. In the CALL MISSING routine shown in this example all the elements of the array ANNUAL are set to missing. There is no need to step through the array one element at a time.

```
retain q1-q4 .;
array annual {4} q1-q4;
if first.year then call missing(of annual{*});
```

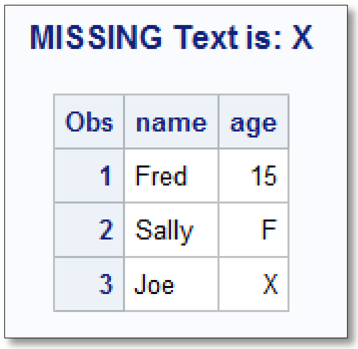
SYSTEM OPTIONS

The MISSING system option allows you to specify a character to display other than the period (.). Like all system option settings, once specified the replacement value remains in effect, persists, until the end of the SAS session, job, or until reset.

The data set SHOWMISS has three observations and two missing values, the special missing value .f and a standard missing value. The MISSING option will not change how a missing value is read or how it is used in an expression; however, it does change how the missing value is displayed. Here the MISSING system option is given the value of ‘X’ (the use of the quotes is optional on the OPTIONS statement.

```
data showmiss;
input name $ age;
datalines;
Fred 15
Sally .f
Joe .
run;
options missing=X;
title1 'MISSING Text is: X';
proc print data=showmiss;
run;
```

Examination of the PROC PRINT results shows that special missing values (.f) are not replaced; however, the missing value for Joe’s age has been replaced with an X.



MISSING Text is: X

Obs	name	age
1	Fred	15
2	Sally	F
3	Joe	X

Because you are limited to a single character when using the MISSING system option, it is often far more flexible to write and use a user-defined format to recode missing values (see the PROC FORMAT examples a bit later in this paper).

MISSING CLASSIFICATION VARIABLES

Throughout SAS, when classification variables are missing, their associated observation is excluded from the analysis. This is true for procedures with explicit CLASS statements, such as PROC TABULATE, MEANS, and PROC GLM, as well as for those with implicit classification variables, such as PROC FREQ and PROC REPORT. Sometimes this is the behavior that you want; however, often it is important that these observations not be removed. The MISSING option allows missing values to be valid levels of the classification variable.

In the data set MISSDEMO.CLINICS the classification variables RACE and SEX do not have any missing values, however the variable SYMP has 12 missing values out of 80 observations. In the current versions of SAS (9.3 and later) both of the tables generated by this FREQ step reflect the correct number of observations (SYMP has 68, while RACE*SEX has 80), however in earlier versions the missing values of SYMP could affect the number of observations in the second table (both could have 68).

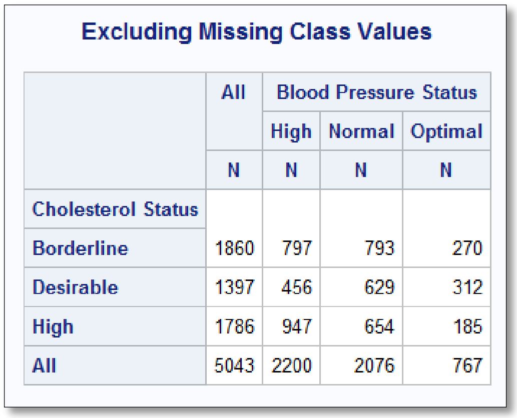
```
proc freq data=missdemo.clinics;
   table symp race*sex;
   run;
```

The MISSING option can be used with most procedures that have either implicit or explicit classification variables. This option can be used on a CLASS statement or on the PROC statement. When used on the PROC statement the option applies to all the classification variables; however, when it is used on the CLASS statement it is only applied to those specific classification variables. In PROC FREQ the MISSING option can also be used as an option on the TABLES statement, and in PROC REPORT it can appear on the DEFINE statement. Including the MISSING option on the TABLE statement forces the inclusion of all observations regardless of the version of SAS and the presence of missing values.

```
proc freq data=missdemo.clinics;
   table race*sex/missing;
   run;
```

While the FREQ procedure determines observations to be included in a table using only the classification variables used in that table, this is not necessarily true for other procedures. When tables are generated by PROC TABULATE, all the classification variables are examined, and any observation with a missing classification variable is eliminated from all the tables – even those that do not utilize the classification variable with the missing value. This is demonstrated by including two classification variables (WEIGHT_STATUS and SMOKING_STATUS) that do not even appear within any TABLE statement.

```
proc tabulate data=sashelp.heart;
class chol_status bp_status
      weight_status smoking_status;
var weight;
table chol_status all, all bp_status*weight=' '*n;
run;
```



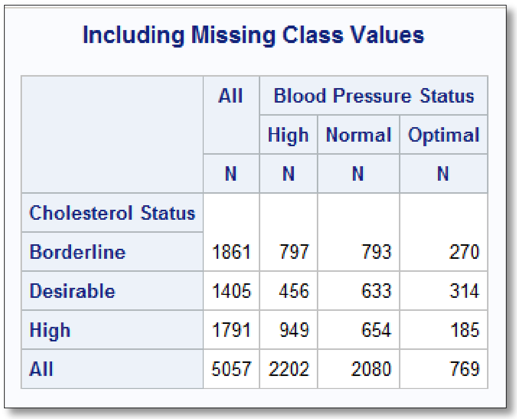
Excluding Missing Class Values

	All	Blood Pressure Status		
		High	Normal	Optimal
	N	N	N	N
Cholesterol Status				
Borderline	1860	797	793	270
Desirable	1397	456	629	312
High	1786	947	654	185
All	5043	2200	2076	767

Simply applying a MISSING option to the classification variables that are *not used* is sufficient to change the counts for the table that uses the *other* classification variables. This shows that in TABULATE the unused classification variables influence the counts of unrelated tables!

```
class chol_status bp_status ;
class weight_status smoking_status/missing;
```

Placing the MISSING option on the PROC statement would apply the option to all classification variables. Again the counts would change, because now missing values of CHOL_STATUS and BP_STATUS would also be included in the table.



Including Missing Class Values

	All	Blood Pressure Status		
		High	Normal	Optimal
	N	N	N	N
Cholesterol Status				
Borderline	1861	797	793	270
Desirable	1405	456	633	314
High	1791	949	654	185
All	5057	2202	2080	769

REASSIGNING MISSING VALUES

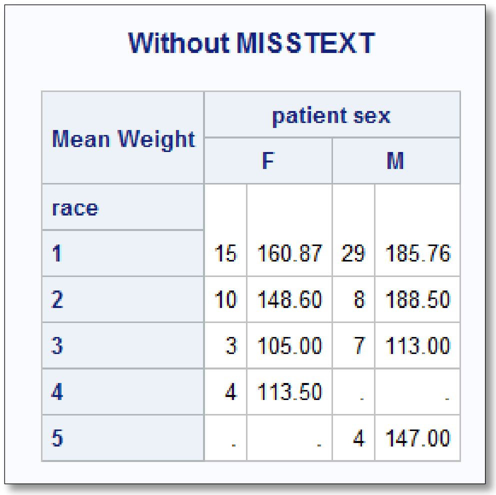
As SAS programmers we are used to seeing missing values displayed as dots in our reports, however most of those for whom the report is designed are not. This means that most of the time when we create reports the missing values must be replaced by some other symbol. We have already seen a several of ways to do this. For special numeric missing values such as .A, only the upper case letter is displayed and often a footnote can be provided as way of explanation. For the standard numeric missing dot, the MISSING system option can be specified. But these are not our only options.

In TABULATE Output

When using PROC TABULATE you can replace missing calculated values through the use of the MISSTEXT option. The TABULATE table demonstrated here shows that there are no observations, and hence missing values for Males of Race 4 and Females of Race 5.

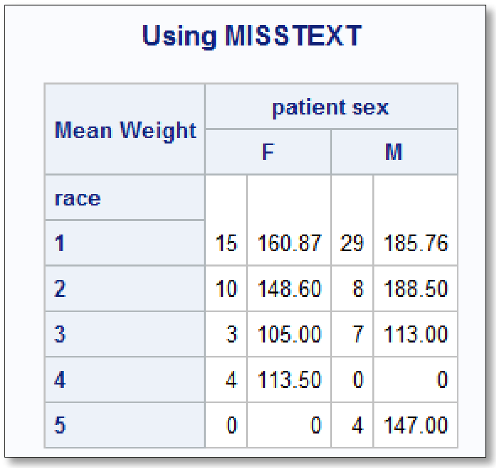
Including the MISSTEXT= option on the TABLE statement allows us to replace the missing values with specified text. Here they are to be replaced with a zero (0).

```
proc tabulate data=missdemo.clinics;
   class race sex;
   var wt;
   table race
         ,sex*wt=' '*(n=' ' mean=' ')
         /box='Mean Weight'
          misstext='0';
   run;
```



Without MISSTEXT

Mean Weight	patient sex			
	F		M	
race				
1	15	160.87	29	185.76
2	10	148.60	8	188.50
3	3	105.00	7	113.00
4	4	113.50	.	.
5	.	.	4	147.00



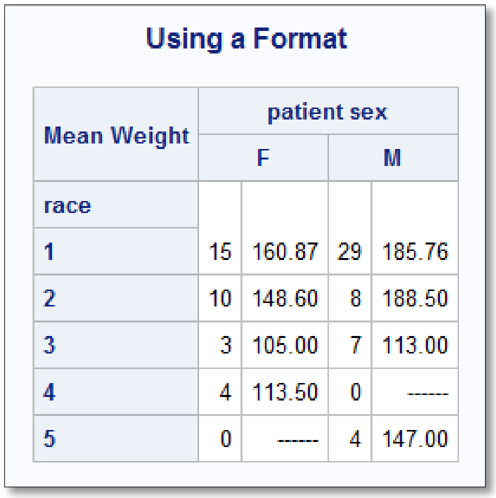
Using MISSTEXT

Mean Weight	patient sex			
	F		M	
race				
1	15	160.87	29	185.76
2	10	148.60	8	188.50
3	3	105.00	7	113.00
4	4	113.50	0	0
5	0	0	4	147.00

Although the zero is appropriate for the N, it is less than desirable for the mean weight. Fortunately we can further refine this table with a user defined format.

Using User Defined Formats

User defined formats offer us a great deal of flexibility and control in what is to be displayed. In the TABULATE example shown to the right we would like to replace the zero associated with the mean weight with six dashes (the zero is still to replace the missing values of N).



Using a Format

Mean Weight	patient sex			
	F		M	
race				
1	15	160.87	29	185.76
2	10	148.60	8	188.50
3	3	105.00	7	113.00
4	4	113.50	0	------
5	0	------	4	147.00

The VALUE statement names the format

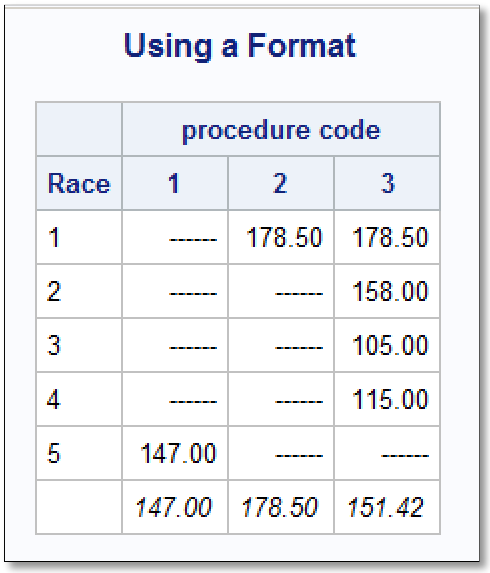
```
proc format;
   value misswt
      . = '------'
      other=[6.2];
   run;
```

(MISSWT.) and defines the mapping for the incoming values. In this format missing is mapped to 6 dashes while everything else maps to a 6.2 format. The format is then associated with the mean through the use of the asterisk (*).

```
table race
      ,sex*wt=' '*(n=' ' mean=' '*f=misswt.)
      /box='Mean Weight'
       misstext='0';
```

Missing Values and User Defined Formats in REPORT

The formatting techniques shown in the previous section can also be used in PROC REPORT. However there is one case where the detection of a missing value by the format is different. In the report shown to the right the missing values in the report have been replaced with dashes using the same format (MISSWT.) that was used in the TABULATE example.



Using a Format

	procedure code		
Race	1	2	3
1	------	178.50	178.50
2	------	------	158.00
3	------	------	105.00
4	------	------	115.00
5	147.00	------	------
	147.00	*178.50*	*151.42*

What if we would like to replace the missing RACE in the RBREAK summary line with text? Following the example of filling missing values using a format we could create the format $MISSRACE., where a missing value is mapped to the word 'All'. This format is then applied to the character grouping variable RACE.

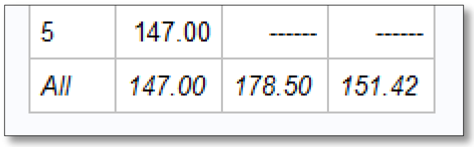
```
proc format;
   value $missrace
      ' ' = 'All';
   run;
proc report data=missdemo.clinics
            nowd;
   column ('Race' race) proced,wt;
   define race / group ' '
                 f=$missrace.;
   define proced / across;
   define wt   / mean ' '
                f=misswt.;
   rbreak after / summarize;
   run;
```

Unfortunately this does not work! We get the same table as before! Missing values on summary lines are not handled the same, as other values, when applied to a format.

Instead we need to change the value to a something that is non-missing and map the format to that value rather than to missing. Since it never appears in the data, the solution code shown here uses a lower case 'x'. A compute block is then added to the REPORT step which resets the value on the summary line from missing to 'x'. The remainder of the step remains unchanged.

```
proc format;
   value $missrace
      'x' = 'All';
   run;
```

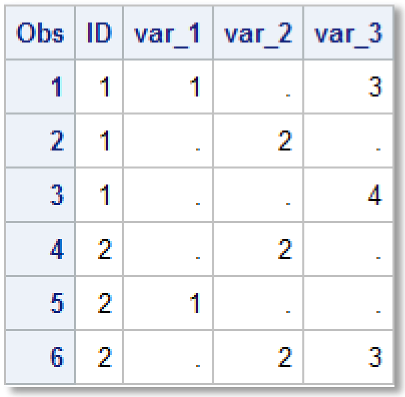
```
compute race;
   if _break_='_RBREAK_' then race='x';
endcomp;
```



5	147.00	------	------
All	*147.00*	*178.50*	*151.42*

Collapsing Data Using the UPDATE Statement

If a data set contains multiple observations per BY group, and you would like to collapse the data to one observation per BY group and at the same time replace missing values with non-missing values (when available), the UPDATE statement can be used. In this example the initial data set (WORK.HAVE) has a series of variables that contain a mixture of missing and non-missing values. We want to collapse the data set into one row per BY group and at the same time save the latest non-missing value for each variable of interest.



Obs	ID	var_1	var_2	var_3
1	1	1	.	3
2	1	.	2	.
3	1	.	.	4
4	2	.	2	.
5	2	1	.	.
6	2	.	2	3

An approach using assignment statements requires the user to know and use variable names. Here the COALESCE function is used to return the first non-missing value of the arguments. Because the incoming value (TEMPx) is listed first, the latest non-missing value will be stored. This approach requires the user to know and code the various variables. The coding can be simplified using the UPDATE statement.

```
data want1(keep=id var_:);
   set have(rename=(var_1=temp1 var_2=temp2 var_3=temp3));
   by id;
   retain var_1 var_2 var_3 .;
   if first.id then call missing(of var_1 var_2 var_3);
   var_1 = coalesce(temp1,var_1);
   var_2 = coalesce(temp2,var_2);
   var_3 = coalesce(temp3,var_3);
   if last.id then output want1;
   run;
```

Obs	ID	var_1	var_2	var_3
1	1	1	2	4
2	2	1	2	3

The resulting data set (WANT1) will have one observation for each BY group, which in this case is each level of the ID variable.

The UPDATE statement assumes that the first data set listed is the primary data set and that the second data set is a transaction data set whose observations are used to 'update' the primary. The BY statement is used to identify (presumably to the row level) the rows to which the transactions are to be applied. Missing values in the transaction data set are ignored and non-missing values replace values in the primary. In this application the same data set is used in both roles.

The first occurrence of HAVE places the BY group variables onto the PDV first. Notice that no observations are read from this usage. The second usage of HAVE is as the transaction data set. During compilation the variables VAR-1 – VAR_3 are added to the PDV. During the execution phase of the DATA step, each observation is read as a transaction. It is at this time that missing values are replaced by non-missing values and observations are essentially collapsed. The resulting data set is the same as was achieved in the DATA step using the COALESCE function, however the coding is substantially easier.

```
data want2;
  update have (obs=0 keep=id)
         have;
  by id;
run;
```

This technique was suggested by @Tom in a SAS Forums thread.

MACRO LANGUAGE NULL VALUES

The macro language does not support the concept of a missing value. While a macro variable can take on the value of a blank or a period, these values are not treated as missing values by the macro language. Unlike data set variables, a macro variable can take on a null value; that is, the macro variable can store nothing. This is generally not possible for variables on a data set.

When working with null macro variables the syntax may at first look odd to the DATA step programmer. The %IF statement shown here is considered to be standard syntax for comparing a macro variable (&CITY) to a null value. Notice that there is nothing between the equal sign (comparison operator) and the %THEN. Since DATA step comparisons must have something on the right of the comparison operator, this statement form often makes newer macro programmers uneasy.

```
%if &city= %then %do;
```

Acceptable alternative forms of this same comparison can include the use of the %LENGTH and %STR functions. Since the macro variable can contain nothing the %LENGTH function can return a zero, and this can also be used to detect a null value in a macro variable.

```
%if %length(&city) = 0 %then %do;
```

```
%if &city = %str() %then %do;
```

Although it generally works correctly, using quotes to satisfy the *need* to have 'something on the right side of the comparison operator' is not considered good

```
%if "&city"="" %then %do;
```

programming practice. The quotes are parsing characters in the DATA step, but not so in the macro language. To do something similar without stepping out of the macro world, you could use one of the macro quoting functions to put 'something to the right of the equals sign'. Here the %STR function is used (with nothing between the parentheses).

Although commonly used, Chung and King (2009) showed that each of the previous comparisons can fail under various circumstances. Their tests show that the most robust test of a null value uses a combination of %SUPERQ and the %SYSEVALF functions.

```
%if %sysevalf(%superq(city)=,boolean) %then %do;
```

SUMMARY

We encounter missing values on an ongoing basis. We need to know the basics of how to detect and use them, but we also need to be well versed in various tools within SAS are used to detect, count, convert, and utilize missing values. These tools are many and varied, and it is a part of our job to be able to use them effectively – if only to make sure that we are not missing out.

ABOUT THE AUTHOR

Art Carpenter's publications list includes; five books, two chapters in *Reporting from the Field*, and numerous papers and posters presented at SAS Global Forum and SAS user conferences. Art has been using SAS since 1977 and has served in various leadership positions in local, regional, and national user groups.

Art is a SAS Certified Advanced Professional Programmer, and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
www.caloxy.com

View my paper presentations page at:
http://www.sascommunity.org/wiki/Presentations:ArtCarpenter_Papers_and_Presentations

REFERENCES

Carpenter, Art, 2012, *Carpenters Guides to Innovative SAS® Techniques*, SAS Press, SAS Institute Inc., Cary NC.

In her 2006 paper “MISSING! - Understanding and Making the Most of Missing Data” Suzanne Humphreys includes a number of nice examples and explanations on the use of missing values. The paper was published in the *Proceedings of the Thirty-first Annual SAS Users Group International Conference*, 2006, NC: SAS Institute Inc., paper 025-31.
http://www2.sas.com/proceedings/sugi31/025-31.pdf

A careful and detailed discussion concerning the testing for null macro values is presented by:
Chung, Chang Y., and John King, 2009, “IS THIS MACRO PARAMETER BLANK?” . The paper was published in the *Proceedings of the SAS Global Forum Conference*, 2009, NC: SAS Institute Inc., paper 022-2009.
http://support.sas.com/resources/papers/proceedings09/022-2009.pdf

A comparison of the IS MISSING and IS NULL comparisons used in PROC SQL is discussed in the LinkedIn thread:
https://www.linkedin.com/groups/IS-NULL-IS-MISSING-Operators-2356262.S.5880340836025131008?view=&item=5880340836025131008&type=member&gid=2356262&trk=eml-b2_anet_digest-null-5-null&fromEmail=fromEmail&ut=2GPr7n15XwASg1

TRADEMARK INFORMATION

SAS and SAS Certified Advanced Professional are registered trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.