WHAT SAS* SOFTWARE DOES AND DOESN'T DO: THINGS YOUR MOTHER NEVER TOLD YOU

Calvin D. Croy, National Demographics & Lifestyles

It has been said that people learn more from mistakes and failure than from success. Yet learning by self discovery can cost time, composure, and sometimes money. Because of these costs, most people have probably sometime in their lives had occasion to reflect "Gosh, I wish someone had told me. . .". The purpose of this paper is to help prevent such afterthoughts concerning SAS code. Listed in this paper are a few of the SAS software version 82.3 peculiarities I or my colleagues have encountered while submitting batch jobs within a VS1 operating system. We've all "discovered" instances where SAS software handles data in unexpected ways. The few here might take more than a little digging through the user's manuals to unearth. They're listed below according to decreasing likelihood of being encountered.

Surprise #1. Little elementary-level documentation in Basics User's Manual. User familiarity with SAS software insufficient to select the best Procedure.

Though perhaps analysts above the novice stage won't consider this a SAS software peculiarity, many beginning SAS software users do have problems deciding which Proc to select. For example, the Basics User's Manual doesn't directly help identify which Proc a user should select if all the user wants to do is "compare two means". The advice of a more advanced analyst may not be available, and at such an elementary level reference to the Basics User's Manual probably won't help much. Should the neophyte select Proc Means? Proc Summary? Proc Univariate? Proc Freq? Proc Corr? Proc T-test? Proc GLM? Proc ANOVA? All these procedures can be used to perform statistical analyses useful in comparing two means. For now, the only answer is to talk to a more experienced analyst, get other manuals, read a statistical textbook, or all three--depending upon the potential user's background.

Surprise #2. The message printed in the SAS log as a result of an error may not have straightforward connection to the error made. Restated, error messages in the SAS log do not always help identify an error and can actually misidentify an error.

Example 1. Consider a title statement imbedded in a 300 line SAS program. The user unwittingly places a single apostrophe within the title rather than the double apostrophe required by SAS software. The resulting error message "character literal has more than 200 characters" can be confusing to those uninitiated into the ways of SAS software. This message is especially confusing if the programmer realizes that no character values have been used in the program.

Example 2. The programmer places a semicolon in the wrong place. Consider a data step opened with this accidentally split line of code:

```
Data First(Keep= X); Second(Keep= Y);
```

The resulting error message "Explicit subscripting of array variables is not supported in this release of SAS" means little. No array has been referenced though SAS software has mistaken Second for one.

Surprise #3. The logical comparison SAS software makes may not be the one wanted. The user should take care to understand the Boolean logic behind SAS software logical comparisons.

Example: The statement "if X or Y > 5" will always be true, regardless of the value of X and Y. The Boolean interpretation of this statement is "if X = X or Y > 5". Since X = X is always true, all records will pass this test. Similarly, the test "if X = 10 or 20" will always be true because 20 is always equal to 20. The correct syntax for these tests are "if X > 5 or Y > 5" and "if X = 10 or X = 20". Problems such as these with Boolean logic frequently snare unwary SAS software users new to computer languages in general.

Surprise #4. Because of the way SAS software represents numbers internally using floating point decimal, tests that are actually true can be taken as false. Consequences can be disastrous!

Example:

```
Data Two; Set One;
If Area ** .5 = 5 then go to Code1;
Else go to Code2;
```

If the variable Area's value is 25, where will the flow of execuation be carried? To label Code1 or to label Code2?

Answer: Code2. Floating point decimal representations can cause minute errors in the right most bits SAS software uses to store the results of arithmetic operations. SAS software unfortunately uses the floating point binary representations of numbers to make comparisons. As this example shows, the binary representation of 25 raised to the power of .5 is not perfectly equal to the binary representation of 5. If the variable Area's value is printed however, the value printed will be 5. This occurs because SAS software converts the internal binary code to the nearest base 10 number prior to printing. The printed value is base 10, but the tested value is binary floating point. The execution of unintended code despite the seeming correctness of values when printed can be almost impossible to debug

in lengthy SAS programs.

Moral: For critical applications, round values to a user-specified level of precision before testing for perfect matches.

Surprise #5. SAS software may interpret input statements differently from the way you think it does.

Consider the following code:

```
Data Example;
Length A $ 10;
Input A Y 1. ;
Cards;
abcdefghij8
;;
```

A logical interpretation of how the length and input statements interact would be to assume that since character variable A has been declared to have a length of 10, then numeric variable Y would be sought for beginning in byte 11. In this example, the anticipated value for A is 'abcdefghij' and Y is 8. However, when this code runs, SAS software will return a missing value for variable Y. A is read using list input which will search for a blank to separate the two variables' values. Since no blank separates A's value from Y's value, Y's value is not found and is set to missing.

Note however that it is safe to use "Input Y 1. A;" with data of 8abcdefghij. Since Y is read using an informat, no blank in the data is searched for, and A's value will be sought beginning in byte 2.

Surprise #6. SAS software may interpret arguments to functions differently than you do.

```
Example:  X1 = 1; X2 = 100; X3 = 100;
                  X4 = 100; X5 = 100;
          Total = SUM(of X1, X2 - X5);
```

The value of Total above is 1, not 401.

The reason here is that because of the preceding comma, SAS software considers X2 - X5 to be a subtraction to be performed prior to the operation of the SUM function. Since the value of X2 subtract X5 is 0, X1 + 0 = 1 + 0 = 1. The solution here is to remove the comma: SUM(of X1 X2 - X5) = 401.

Surprise #7. Functions may not generate missing values, or may return unexpected values, compared to doing the calculation "by hand".

```
Example:  Given X = 1        Y = 2
                T = missing value
          Z = SUM(of X  Y  T ) = 3 but
          Z = X + Y + T = missing value.
```

Many SAS software functions will exclude variables with missing values and operate on the remaining variables. Hence one should try to avoid using functions to critically screen data. For example, suppose one would like to keep records where variables M1 through M5 have values greater than or equal to 1. Using the code "if SUM(of M1 - M5) >= 5" will merely insure that at least one of the variables has a value of 5 or more. Note that if M1 = 5 and M2 - M5 all have missing values, then SUM(M1 - M5) will equal 5 and the test is satisfied even though 4 out of 5 variables had missing values. Or perhaps one wants to make sure retained records have no missing values for variables M1 - M5. The test "if SUM(of M1 - M5) = . then delete" will delete records only where all 5 variables have missing values. If M1 = 1 and M2 - M5 all have missing values, then SUM(of M1 - M5) will return a value of 1, not a missing value. Of interest too is that if A = 10 B = 20 C = 30, and X = 20 Y = . Z = . , then MEAN(of A B C) = MEAN(of X Y Z) = 20.

Surprise #8. Using the trim function to remove trailing blanks from character values doesn't work unless you include a length statement. (Unexpected, though well documented in 82.3 Basics User's Manual on p. 194).

The trim function removes trailing blanks from a character variable's value, and the resulting trimmed value has the length specified in an accompanying length statement. If no length statement is specified, the trim function in effect does nothing since any trimmed blanks are reinserted.

```
Example:  A = 'XYZ   ';
          B = TRIM(A);
```

Here the value of B will be 'XYZ '. To get a value returned of 'XYZ' the user needs to supply the statement: Length B $ 3;

Surprise #9. Variables created from the substring function retain the length of the variable from which they were created, unless specified otherwise.

```
Example:  Length Y $ 130;
          H = SUBSTR(Y,1,2);
```

As can be revealed with a Proc Contents, the length of H will be 130, not 2.

In order to have H be of length 2, its length needs to be explicitly declared: Length H $ 2;

Surprise #10. How you merge determines what you get. But is it what you want? Merging can create unexpected holes in your data.

Consider the merge of Data B with Data A in order to have the variable TOTAL appended to each observation in the resulting Data C.

Data A

OBS	GROUP	SEX
1	1	M
2	1	M
3	1	M

Data B

OBS	GROUP	SEX	TOTAL
1	1		20

Data C; Merge A B; By GROUP;

Data C will appear as:

OBS	GROUP	SEX	TOTAL
1	1		20
2	1	M	20
3	1	M	20

Why does observation #1 have a blank for the value of the character variable SEX when observations 2 and 3 have the anticipated value of 'M'? Answer: The blank value arises because Data A's value for SEX was replaced by Data B's value, a blank. Also "when multiple observations occur within a by group [such as in Data A], the value in the new dataset is the value from the dataset mentioned latest in the merge statement that is still contributing information to the by group" (p. 105, 82.3 Basics User's Manual). Thus for observation #1, the contributing dataset mentioned latest in the merge statement is Data B, and the blank value for SEX is taken from B. But for observations 2 and 3, the contributing dataset mentioned latest in the merge statement is the only contributing dataset--Data A. For observations 2 and 3, the value of M is taken from Data A, the only contributing dataset. Why the value 20 for TOTAL is kept for all records is succinctly stated as well on p. 105 of the 82.3 Basics User's Manual: "When a dataset runs out of observations for a by value, values from the last observation contributed by the dataset are retained for the remaining observations in the by group".

Surprise #11. The sign of the t-value generated by Proc T-test depends upon the spellings of the class variable's values.

Let's say you always want to compare another company's data to your company's data. If the other company's mean is significantly larger than your company's mean, the t statistic should have a positive value. If the other company's mean is significantly less than yours, you expect the t value to be negative. But in practice it doesn't work this way with Proc T-test.

Proc T-test orders the values of the class variable alphabetically on the printout, and the mean of the data represented lowest is subtracted from the mean printed above. Depending on the name you furnish to designate the other company's data, their mean will appear either above or beneath yours, regardless of the direction of difference in means. Hence comparing the same sets of data ordered the same way can lead to different signs for the t value depending upon the spelling selected.

Example: Other company's variable TIME has mean of 5. Class variable GROUP given value of 'other'.

Your company's variable TIME has mean of 2. Class variable GROUP given value of 'ours'.

Proc T-test; Class GROUP; Var TIME;

The listing from this program will show a positive t value since 'other' will physically appear above 'ours'. Alphabetically, 'ot' comes before 'ou'.

If the same data are run again, but the class value for your company's data is changed from 'ours' to 'mine', the same program will generate a negative t value of the same magnitude. This is because 'mine' will physically appear above 'other' since alphabetically 'm' precedes 'o'.

Surprise #12. To use Proc Score with coefficients from Proc Reg, a label must have been present on the Model statement of Proc Reg. This is true even though "the label is optional".

Proc Score uses the value of the _MODEL_ variable in the scoring coefficient dataset from Proc Reg. When used with regression coefficients, Proc Score takes the value of the _MODEL_ variable to name the scores it creates. The value of the _MODEL_ variable is the label the user supplied with the original Model statement of Proc Reg. If the user supplies no label, then the value of _MODEL_ in the coefficient dataset is blank, and Proc Score will create scores for a variable without a useable name since its name will consist entirely of blanks. A variable name made of nothing but blanks makes such scores very hard to later reference!

Example 1: Without use of "optional" label.

```
Proc Reg data= Readin  Outest= Coefs;
     Model SALES = PRICE;
Output out= Regout Predicted= YHAT;
Proc Score data= New score= Coefs
out= Scored type= OLS;
Var PRICE;
Proc Print data= Scored;
```

Data Scored

PRICE	
10	2000
50	500
30	1000

Example 2: With use of "optional" label.

```
Proc Reg data= Readin Outest= Coefs;
Score: Model SALES = PRICE;
Output out= Regout Predicted= YHAT;
Proc Score data= New score= Coefs
out= Scored type= OLS;
Var PRICE;
Proc Print data= Scored;
```

Data Scored

PRICE	SCORE
10	2000
50	500
30	1000

Moral: If the regression coefficients created by Proc Reg are to be used later with Proc Score, always use a label in the Proc Reg's Model statement.

Surprise #13. Using a permanent file for work space by giving it a dd name of Work in the JCL can cause headaches.

When one creates or uses a permanent external file by having a dd name of Work in the JCL, the last line number of source code is stored in the external file regardless of the number or position of the other internal pointers. This facilitates using statements such as %include. When the physical external space is reused for work space, SAS software begins counting lines of source code beginning with the line stored in the external file. With each run the stored end line number is added to. However, a SAS program can contain only 32,000 lines of SAS code. Thus when large SAS programs run frequently and reuse the external file, the 32,000 line limit will eventually be reached and an error generated.

We have had no success in resetting the line number stored in the permanent external file. Neither have we been able to destroy it using the kill option to Proc Datasets. The only way we at NDL have been able to remove the offending line pointer has been to run a small PL1 program to open and close the external file, thus purging all pointers set in the file.

Moral: Don't use Work as a dd name in JCL!

Surprise #14. Under CMS the device type must be supplied by the user either in the options statement of the program executing SAS/GRAPH* or in a SAS profile.

When using SAS/GRAPH under OS, SAS software can get the device type from the operating system so the user needn't explicitly declare it. But to run SAS/GRAPH under CMS, the user does need to supply the device type since it can't be obtained from the operating system. The real surprise here is that such requirements are listed in the 82.3 SAS Basics User's Manual rather than in the 82.3 SAS Companion for the VM/CMS Operating System.

Surprise #15. Using Proc Freq with the Weight statement on very large datasets can produce incorrect and inconsistent counts.

Example: 206,395 records having GROUP values A through E and an integer weight value for variable X were run twice through the following code.

```
Proc Freq; Tables GROUP; Weight X;
```

Results

	Group	Frequency	Cumulative Frequency
Run #1	E	1,430,393	64,284,606
Run #2	E	1,430,393	64,161,260

In the above example, the final total frequency calculated was first 64,284,606 and when rerun, 64,161,260--a difference of 123,346 or 0.5976 for each record processed. Neither of these totals was correct.

The true count total was 64,517,076 as calculated using counters in a data step and also by using Proc Summary with a Freq statement. The inaccuracy of Proc Freq in this use remains a mystery.

Moral: Use Proc Summary with a Freq statement to process very large sets of data.

Surprise #16. SAS software writes a missing value (.) using packed decimal format as -0. However, when such -0 values are read using the same packed decimal format with which they were written, the value input is 0. The negative sign is lost. In datasets written using packed decimal format, the distinction between a missing value and 0 is lost when the data are read using a packed decimal informat.

Surprise #17. The column binary informats Row 0.6 and Row 10.6 are not equal, despite what the 82.3 Basics User's Manual shows.

Consider a puch occurs in row 10. The informat Row 0.6 will return a value of 0 whereas Row 10.6 will return the value of 1. In general, using Row 0.6 returns a value of 1 less than the true value!

Suggestion: User Row 10.d instead of Row 0.d.

*SAS and SAS/GRAPH are registered trademarks of SAS Institute Inc., Cary, NC, USA.

[1] Doyle Bishop, Chris Hamlin, Don Hinman, Bernie Schneider, Bill Schneider, Duane Schulte, and Frank Scoviak

For further information contact:

Dr. Calvin D. Croy
Senior Statistical Analyst
National Demographics & Lifestyles
1621 Eighteenth Street
Denver, Colorado 80202
USA
(303) 292-5000