Paper 069-2011

Nobody Warned Me! Be Careful With SAS® Invisible Errors

Ieva Brauksa, University of Latvia, Riga, Latvia

ABSTRACT

This paper can be useful for SAS programmers with different experience levels. It points out several logical and programming errors, sometimes called SAS gotchas, which can lead you to incorrect results without producing warnings or errors that would be easy to spot in the log. These mistakes arise from misinterpreting coding principles in a way that SAS OnlineDoc® or the SAS System Help don‘t warn you to be careful about.

INTRODUCTION

When you start to learn SAS and run your first programs, there is a great joy when they finally run without errors or warnings. Later, you face situations when you wish there would have been at least a warning in the log – that is when you deliver some results that seems perfectly correct but actually have some logical or other error that just can‘t be noticed so easily. And all you can say is that nobody warned you about them.

This paper gives you examples of some of these cases that produce no warnings and also in SAS online documentation there are no notes about possibilities to misinterpret the results.

1. DAYS OF THE WEEK MIX – USEFUL WHEN ON MONDAYS YOU WISH IT WAS STILL SUNDAY

Isn‘t it great that SAS can tell you what day it was on 22JUL1985? Sometimes that can be helpful if you are doing some analysis where day of the week is important factor. On which days do customers usually visit stores, or maybe on which days are sales relatively lower? How many visits were there to hospitals on Saturdays in July of last year?

So the first step would be to define what day of the week particular dates are. Let us assume that you have heard something about weekday function, even more – you have read SAS documentation and know that weekday=1 is Sunday and not Monday as it could be thought in some parts of the world.

Then you find out that there is a great format WEEKDATEw. that changes dates into names of the days of the week (DOWNAMEw. would give the same results). So the code is written:

```
data day_of_week;
input fulldate date9.;
day=weekday(fulldate);
day_number=day;
day_correct=fulldate;
format fulldate date9. day weekdate3. day_correct weekdate3.;
datalines;
04APR2011
 05APR2011
 06APR2011
 07APR2011
;
run;
```

And you get results:

fulldate	day	day_number	day_correct
4-Apr-11	Sun	2	Mon
5-Apr-11	Mon	3	Tue
6-Apr-11	Tue	4	Wed
7-Apr-11	Wed	5	Thu

Problem is that variable ”day“ is not what you had expected. April 4th, 2011, which happens to be first day of SAS Global Forum, is Monday, but as frequently in Mondays you could wish that it was still the weekend and not the beginning of new week. In this case, it seems that SAS feels the same; it says: Hey, day number is “2”, it could be Sunday (day=’Sun’). Why does SAS do so if we know that if no other options are used day number “2” stands for Monday?

Here we come again to the question of how SAS treats dates. In a way, how day_number is defined in the code above, results in it no longer storing information about fulldate – the date it was taken from. Now it is just a number “2”. But, if you tell SAS that it is a date that should be changed to the name of weekday, SAS looks into memory where “2” as a date means 3JAN1960, which was indeed Sunday.

For using the WEEKDATEw. format you don't need day but full date (like for "day_correct"). There are no warnings and in such a way just following your own logic, you can find yourself in situation where results about performance on different days could be surprising.

Lesson learned: Be sure that when you use format WEEKDATEw. or DOWNAMEw., you use full date, not just the number that indicated day of the week.

2. REALLY NO OBSERVATIONS?

Those that work with SAS for some time know that warnings and errors are not the only things you should be careful with. For those that are new to SAS here is an example how checking other types of messages can help you to avoid delivering false results.

Let as assume that you have some data which you sort by some variable. Later you want to check if there are repeated variables (is this "something" repeated more than once in the data?). There could be several ways to do that. For example, you could just simply use SORT procedure with NODUPKEY option and see if some observations are not deleted because they were repeated. Another way that could be useful in some situations is using FIRST. And LAST. variables. You can define that you want to keep those observations where variable "something" is repeated so you could see which they are.

You could write a program. The line commented out below is a necessary line that is needed to have correct results. Here we discuss the situation where we forget to add this line.

```
proc sort data=in_data;
by  something;
run;

data out_data;
set in_data;
/*by  something;*/
if first.something ne last.something;
run;
```

Unfortunately, the BY statement is missing. In this case data set out_data will have no observations no matter whether there are multiple observations with the same "something" value in in_data or not. So these results are misleading and you could have incorrect idea about data structure.

Similarly, this no-warning error could be important if you wanted to do any other action with FIRST.variable or LAST.variable So be careful and don't forget to add a BY statement.

Unlike other errors mentioned in this paper, this one actually is not so invisible at all. True, there are no warnings or errors, but there is a note in log:

```
NOTE: Variable 'first.something'n is uninitialized.
NOTE: Variable 'last.something'n is uninitialized.
```

So no warnings, but if you are careful, you can avoid this mistake, because name "uninitialized" usually isn't what you want to see in your SAS log and you should check that everything really worked the way it was supposed to work. Here we come back to old rule: read your log, that can help you to avoid mistakes.

Actually there is a nice undocumented option dsoptions=note2err which would convert this NOTE into ERROR. For more undocumented SAS features, check (Cheng 2004).

Lesson learned: You should remember that ERROR or WARNING messages are not the only things that tell that something is wrong. Important information could also be written in notes or other types of messages.

3. WHERE CONTENTS MATTERS

There are several great things about SAS and some of my favorites are the CONTENTS procedure and the WHERE option. Isn't it nice sometimes to quickly get some information about data set using PROC CONTENTS? And I like the WHERE option even more – in many cases it just makes data processing more efficient and who doesn't like efficiency? Thing is that sometimes using two good things together doesn't guarantee that the result would be good as well.

Let us make a simple data set for this example.

```
data data;
input name $ age;
datalines;
Cris 5
Dave 4
Jane 5
;
run;
```

Now you want to check how many kids there are at the age of 5. At the same time, you would like to also have some information about the data set – like, for example, list of variables. So the following step could seem logical:

```
proc contents data=data (where=(age=5));
run;
```

Only there is one problem – the CONTENTS procedure doesn‘t support the WHERE option. So in this case, it will give the same results as if there were no WHERE option. No warning is written in the log. You get information about data set as you wished, only the number of observations is written to be 3. That is the total number of observations, not only those where age is 5.

So in this case, other procedures could be used to check the results you wanted to get. PROC CONTENTS is good for getting overall information about data set. If you want more information about variables, for example, you want to count how many different values a variable has, you can still check it with other procedures (for example, PROC FREQ).

Trying to be efficient is great and sometimes it is better to use only one DATA step or procedure with several options and statements instead of getting results from several that could take some extra time processing data. But there are times when striving for greater efficiency overcomes the limits SAS can offer and you just need to accept that there are some things that can't be done in one step but take several steps to achieve correct results.

I should mention that contrary to previous versions of documentation, in SAS OnlineDoc 9.2 there is a restriction stated that "you cannot use the WHERE option to affect the output because PROC CONTENTS does not process any observations". So if you use that version of documentation then you will be at least warned.

Lesson learned: PROC CONTENTS doesn‘t support WHERE option. Using this options don‘t change anything in the output, but there are no errors or warnings issues in the log.

4.MULTIPLE SIMILAR VALUES IN IN= DATA SET OPTION

This point is not much of a SAS trick or misunderstanding of what you have coded. This is included to remind you that you should be careful about different things – not only misunderstanding the code, but also in creating your code or simply typing.

Let us have an example where you want to merge three data sets by a common variable "something":

```
data results;
merge first_data   (in=a)
      second_data  (in=b)
      third_data   (in=a)
;
by something;
if a;
run;
```

Results from such a step would include data that would be at least in first_data or in third_data. Of course there could be cases when that is exactly what you want. When using the IN option, to not have any surprises, you should know that if the IN option has the same value for several data sets, observation would be included if that is present in at least one of them.

If you would like to have only those observations that are in both of these data sets, you should use different values of this option and indicate it in the IF statement as illustrated in the code below:

```
data results;
merge first_data   (in=a)
      second_data  (in=b)
      third_data   (in=c)
;
by something;
if a and c;
run;
```

Logical errors are the ones that are hardest to spot. In cases like this, the number of observations in the input and output data could give you some idea if results are what you wanted them to be. Anyway, be careful with logic as logical errors usually don't produce warning or error messages in the log.

Lesson learned: Be careful how you use logic in the IN data set option when using the MERGE statement. There is nothing implicitly illogical, but you should be sure what results you want to get and reflect that in your code.

5.PROC TRANSPOSE VARIABLE ORDER

Do you remember the rule: first in will be first out? That is what you find out when you first start to learn SAS. That is the way in which order variables are stored in data sets – they are kept in SAS memory in the order of how the program first meets them.

The TRANSPOSE procedure can be very useful in some cases, but there are also some things you should be careful with. Thing is that this first in – first out rule applies both for creating and for modifying data sets, like, for example, with the TRANSPOSE procedure. SAS uses information about variable order to create output data sets, but this is not specifically stated in the PROC TRANSPOSE documentation. Seems that this global truth is so important and applies to everything so that it can't be mentioned again and again for each procedure and statement.

Here we will look at the PROC TRANSPOSE ID statement in a way when knowing variable order is important to receive correct results. If your data are not sorted by ID variable in a way you would like to have them, you should remember that also variable order for transposed data set will not be in your wished order. An example data set:

```
data test_data;
input day $ type $ value ;
datalines;
Mon one 1.0
Tue one 3.0
Wed one 5.5
Fri one 3.0
Sun one 2.0
Mon two 4.0
Tue two 3.5
Thu two 2.5
Fri two 4.0
Sat two 5.0
;
run;
```

Now you transpose data to have two rows for each type of values.

```
proc transpose data=test_data out=transposed (drop=_name_);
by type;
id day;
var value;
run;
```

Results you get are the following:

type	Mon	Tue	Wed	Fri	Sun	Thu	Sat
one	1	3	5.5	3	2	.	.
two	4	3.5	.	4	.	2.5	5

Thing is that SAS is not so smart that it would order other variables (days of the week) in the order that days of the week actually follow. Similarly, for example, if you had some other variable and expected output variables to be in alphabetical order, most likely that is not what you would get.

So what could follow from this false perception about variable order? Here you would like to make a data set by each type that would show its value in each day. More likely, you will add some more variables to make the final data set. One more thing you would like to add is total value for each type for all week. So the code is written:

```
data results;
set transposed;
 (... adding other variables..);
weekly_value=sum(of Mon--Sun);
 (... adding other variables..);
run;
```

I suppose you have heard about different ways to sum using SAS. One that can sometimes be useful is using SUM (OF ..). But if you use it as illustrated in this example, a logic error occurs. SAS sums all variables starting from Monday to Sunday. But there is a little problem – variables Thursday and Saturday are stored in the data set after variable Sunday, so they are not included into this sum.

So what could be solution – specifically naming all of the variables you want to sum? You are happy if those are only 7 days of the week. There could be a much larger number with fancy variables you meet in real-life data sets. Maybe SUM (OF _NUMERIC_) could help. But also here you should be careful as there could be that those are not the only numeric variables and something else is included in the sum.

And once again, the SAS OnlineDoc 9.2 proves to be more comprehensive than previous versions of SAS documentation. There is a note included to avoid this problem saying that "when a variable name is being formed in the transposed (output) data set, the formatted values of all listed ID variables will be concatenated in the same order that the variables are listed on the ID statement".

Lesson learned: Pay attention to variable order, especially if you want to use this order in some calculations, for example, telling SAS which observations you want to sum. SAS just puts variables in such an order as he first meets them, not paying attention to alphabet or any other logical order we are used to.

6.NOT EVERYTHING COUNTS – HOW TO MAKE LIFE 25 PERCENT POINTS BETTER

Be careful what you count. This one is just an error you can make if you are not paying enough attention to what you actually do. Sometimes it is very important that you sort data with the NODUPKEY option before making further calculations, for example so that you wouldn't sum the same attribute twice. So - basket id, basket value, items – use PROC SORT NODUPKEY by basket id – and only then sum basket values to know the total sum of sales. But, if items come from purchase and the data set is sorted with the NODUPKEY option, the results are no longer correct because part of the valuable data is missing.

So here is a simple example. You have data from survey how people evaluate their life on three different days.

```
data attitude_data;
length person $1. day $9.;
input person $ day $ quality $ age;
datalines;
1 Monday Good 25
1 Tuesday Bad 25
1 Wednesday Bad 25
2 Monday Good  26
2 Tuesday Good 26
2 Wednesday Bad 26
3 Moday Bad 32
3 Tuesday Bad 32
3 Wednesday Bad 32
4 Monday Good 18
4 Tuesday Bad 18
4 Wednesday Bad 18
;
run;
```

For some calculations you may need single entry for each person. So you may have sorted this initial data set using the NODUPKEY option:

```
proc sort nodupkey data=attitude_data out=attitude_nodup;
by person;
run;
```

Then at some point, you think that you need to see what the proportion of bad and good life quality estimate is. Sounds simple, you just need to use PROC FREQ, right? Something like:

```
proc freq data=attitude_nodup;
tables quality;
run;
```

Sure, it is simple, but the thing is, it is correct only if you use the correct data set. You need to pay attention to where data are taken from. So in this case if you want to see results about overall distribution between good and bad day evaluations, you should take the complete data set "attitude_data" not the one that is sorted NODUPKEY. So the correct step would be:

```
proc freq data=attitude_data;
tables quality;
run;
```

When comparing results, data frequency and in most cases also perceptual distribution will differ. So for our example results can be seen in the following table:

	Incorrect data set		Correct data set		difference (percent points)
quality	**freq**	**percent**	**freq***	**percent***	
good	2	50	3	25	25
bad	2	50	9	75	-25

Of course, statisticians and your boss may hate these results, but isn't it nice that with such a move you can make attitude towards life 25 percent point better?

Lesson learned: Always be careful what data you use. Know your data, but double check that you chose the correct data set and don't' misunderstand something.

7.NOT MERGING DIFFERENT FORMATTED VARIABLES

I could guess that MERGE is one of the most used features in SAS, but it also has some tricks that could surprise you. Let's make some simple example datasets. Both of them contain variable "something" that is formatted to be whole numbers and variable "name":

```
data data1;
input name $ something_1 ;
datalines;
a 1
b 2
c 3
;
run;

data data2;
format something_2 8.;
input name $ something_2;
datalines;
a 1.1
b 2.2
c 3.3
;
run;
```

So if you merge these datasets by name, you get:

name	something_1	something_2
a	1	1
b	2	2
c	3	3

Now suppose that for some reason you would like to merge these data sets not by name, but by "something" (best reason: if that is the only variable you have). So I have a question for you: how many observations would the outdata have?

```
data outdata;
merge data1 (rename=(something_1=something))
      data2 (rename=(something_2=something));
by something;
run;
```

Maybe you are more careful now as you know that this paper is about topics that seem logic and simple but hide some specific point that may create unexpected results. In this case outdata would have 6 observations. Though variables "something" look the same and we could expect that they merge, they don't.

In a table below you can see how the results after merge look (first two columns).

name	something	something_else
a	1	1.0
a	1	1.1
b	2	2.0
b	2	2.2
c	3	3.0
c	3	3.3

There is added variable "something_else" that shows this common variable formatter with one decimal place.

SAS still remembers that in data2 this variable actually has different values, they are just formatted to look the same. So during merge these observations are not combined. SAS has quite good memory and just outer appearance can't trick it as well as it sometimes tricks us.

Lesson learned: When using MERGE, if observations have different values that are formatted to look the same, don't be surprised if they don't actually merge, but instead create different entries.

CONCLUSIONS

This paper mentions a few ways that SAS can trick you, and how you could perceive incorrect data as being perfectly fine. Hopefully, you will not fall for these tricks, though there are surely others that SAS still has prepared for you. The only way to avoid making mistakes is to be careful in situations when you think you understand what your code means.

Quality check from your peers is great, but you need to double check the results yourself as well. If there are some surprising results, that is even better reason to check them one more time to be sure that these surprises don't arise from some misunderstanding. Even if you are experienced and program seems simple, logical errors are the hardest to avoid. I wish you good luck and wariness to also spot those invisible errors!

REFERENCES

SAS OnlineDoc 9.1.3. SAS Institute Inc. Cary, NC, USA http://support.sas.com/onlinedoc/913/docMainpage.jsp

SAS OnlineDoc 9.2. SAS Institute Inc. Cary, NC, USA http://support.sas.com/documentation/cdl_main/index.html

Wei Cheng. 2004. Helpful Undocumented Features in SAS. Paper 040-29
http://www2.sas.com/proceedings/sugi29/040-29.pdf

Rob Krajcik. 2009. Why Does SAS Run Clockwise?, http://www.nesug.org/Proceedings/nesug09/bb/bb10.pdf

SAS gotcha's. http://www.sascommunity.org/wiki/SAS_gotcha's

ACKNOWLEDGMENTS

I want to thank my colleagues for helping me to learn SAS and answering all my questions during that process. And, of course, thanks to my mentor M.A.Raithel for the support in editing this paper.

CONTACT INFORMATION

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

Name Ieva Brauksa
Address Ausekla 16, Cesis, LV4100, Latvia
Email ieva.brauksa@inbox.lv