

Debugging your SAS® Output

by Lauren Haworth
Kaiser Permanente Center for Health Research, Portland, Oregon

Abstract

There is nothing more embarrassing than an end user finding an error in your results. Before you deliver your SAS output, it's important to be confident that your program is running correctly.

ERROR messages are an obvious sign of trouble, but what about those pesky WARNING messages and NOTES? And what about the programming mistakes that SAS can't spot? This paper is a step-by-step guide to reviewing your SAS output. I will cover common problems and solutions, and provide a basic checklist to use in reviewing your output.

Introduction

Even experienced SAS programmers make mistakes. The sign of a good programmer is that they find and fix their mistakes before delivering their output.

This paper came about as the result of my experiences working as an analyst for the coordinating center of a national clinical trial. We had a team of four analysts working for a dozen investigators based all over the country. Many of the investigators were not familiar with the data, and some were not familiar with complex analyses, so it fell on our shoulders to carefully check and re-check our output to catch mistakes.

We chose to implement a formal review process for every SAS program we wrote. This paper will describe the series of tests and checks we ran, and propose a review checklist that can be used for any SAS program.

Check #1: ERROR messages

When you review the log file produced by a SAS run, you get a lot of help from SAS. Most critical errors are already identified for you, flagged with a message that starts with "ERROR: ..."

So the first step in debugging your SAS code is to look for and fix these errors. Even if you are running a production program that you have run hundreds of times before, it is important to check the log for ERROR messages every single time.

For example, maybe an important dataset was accidentally deleted. The next time you run your usual programs you will get an error. For example:

```
DATA THISDATA;  
    SET SD2.THATDATA;  
ERROR: File SD2.THATDATA.DATA does  
not exist.  
    C=A+B;  
RUN;
```

Depending on your operating system and option settings, SAS may continue to run your program even after an error like the one above. If you are lucky, it will be obvious from your output that something went wrong, but if you just look at the output and never check the log file, you might miss a major error message.

Check #2: WARNING messages

Once you have fixed the obvious errors, you can move on to the next level of checks. When you review your log, you will also see messages from SAS that are labeled as a WARNING. These messages are not necessarily a problem, so SAS does not terminate the job. They are simply highlighted for your review.

One warning message that almost always spells trouble is the following:

```
DATA THISDATA (KEEP=B C);  
    SET THATDATA;  
    C=A+1;  
RUN;  
WARNING: The variable B in the  
DROP, KEEP, or RENAME list has  
never been referenced.
```

This message is also telling you that variable B does not exist in the dataset THATDATA. This could be the result of a spelling mistake (the variable is actually BEE), or you may have forgotten to include B in a previous KEEP statement.

Another common warning message is the spelling correction warning. Depending on the version of SAS you are running, it may have some limited capabilities to correct simple spelling errors in your code. In the following example, instead of typing THEN, I mistakenly typed THE:

```
DATA THISDAT2;
  SET THISDAT;

  IF DRUG>2 THE DELETE;
  ---
  1

RUN;

WARNING 1-322: Assuming the symbol
THEN was misspelled as THE.
```

SAS spotted the problem and fixed it for me. While the code ran correctly, it is a good practice to go back and correct the spelling. Also, this spelling correction feature is fairly primitive, and it doesn't always make the right decision. Always review the output carefully and correct the code.

The previous examples were just a few of the many WARNING messages that SAS can display. Most messages are fairly clear, but if you can not figure out why you are getting a certain warning message, print out some test records before and after that section of code.

The main thing to remember about WARNING messages is never to take them lightly. When reviewing your log file you should explore and explain every WARNING message.

Check #3: NOTES

Some problems encountered by SAS seem so benign that SAS only generates a NOTE message. When you run a SAS program, your log will display numerous NOTE messages. Most NOTE messages just tell you things like how many observations are in a dataset, or how many minutes SAS took to run a PROC.

However, a number of nasty errors are only reported in NOTE messages. Be sure to take the time to read each and every one of these messages in your log, so that you catch these subtle errors. Every time you review a log, you want to make sure you can explain the reason for every NOTE message.

For example, you want to watch out for the following message:

```
DATA BOTH;
  MERGE THISDATA THATDATA;
  BY ID;

RUN;
NOTE: MERGE statement has more
than one data set with repeats of
BY values.
```

This message can almost always be assumed to be an error. The message means that you are trying to merge two datasets that each have more than one record for each value of the BY variable (see the SASBASE manual¹ for an explanation). You probably will not like how SAS handles such a merge. To solve the problem, remove the multiple records on one of the two datasets, or use PROC SQL² to execute a more appropriate many-to-many merge.

Another messages that is almost always an error is:

```
DATA THISDATA;
  SET THATDATA;
  C=A+B;

RUN;
NOTE: Variable B is uninitialized.
```

This message means that you have tried to perform some operation using a variable that does not exist. In this case, variable C can not be calculated because variable B does not exist. This may be the result of misspelling a variable name, a mistake in a drop or keep statement or an error in a previous data step.

A more subtle error which generates a note message is following situation:

```
DATA THISDAT2
  SET THISDAT;

RUN;

NOTE: The data set WORK.THISDAT2 has
1 observations and 0 variables.
NOTE: The data set WORK.SET has 1
observations and 0 variables.
NOTE: The data set WORK.THISDAT has 1
observations and 0 variables.
```

¹ "Examples of Match Merging: Duplicate Values of the BY Variable", p. 153, *SAS Language: Reference*.

² See *SAS Guide to the SQL Procedure: Usage and Reference*.

What happened here is that I forgot to put a semicolon after the data statement. This causes SAS to interpret the SET statement as the name of a dataset to be created. The result is that I've just created two garbage datasets: THISDAT2 and SET, and on top of that I've just overwritten my perfectly good THISDAT dataset. Be sure to check your log for any NOTES that refer to datasets you didn't expect to create (like WORK.SET above).

Some note messages may not be a problem. For example, the "NOTE: missing values were generated ..." message may be caused by a few missing values in your data, or it may be a sign that a calculation has gone seriously awry.

```
DATA THISDATA;
  SET THATDATA;
  E=A+B+C+D;

RUN;
NOTE: Missing values were
generated as a result of
performing an operation on missing
values.
```

Each place is given by:
(Number of times) at
(Line):(Column).
72 at 25:16

In this case, there are a few observations with missing data for A, and a bunch of observations with missing data for B. You should review your data whenever you get this message to be sure you expected to find missing data for these variables. Also, any time you get a message about missing data, it is important to see if you can rewrite the code to better take advantage of the data you do have.

The syntax E=A+B+C+D is causing a problem because, to calculate E, it requires that A, B, C, and D be non-missing. In this case, you can rewrite this code as: E=SUM(A,B,C,D). This tells SAS to sum the non-missing values of A, B, C and D. The result is that you will have more observations with a non-missing value of E.

Another note message that you should always check, and possibly try to fix, is the character-numeric conversion message:

```
DATA THISDATA;
  SET THATDATA;
  WEEKS=DAYS/7;

RUN;
NOTE: Character values have been
converted to numeric values...
```

You may have intended to do such a conversion, or this may be a sign that one of your variables is not formatted as you expected. In this case, the days variable was formatted as a character, '1', '2', ..., rather than a number, 1,2, ..., as the code expected.

Finally, there are notes that are not usually a concern. For example, if your label is too lengthy, SAS tells you that it has been truncated. Generally, losing a few characters off the end of a label is not an issue, but consider the following example:

```
DATA THISDAT2;
  SET THISDAT;
  LABEL TESTA='PARTICIPANT RESULTS
FROM LABORATORY TEST #3402'
  TESTB='PARTICIPANT RESULTS
FROM LABORATORY TEST #4098';
```

```
NOTE: Label value for variable
TESTA has been truncated to a
length of 40.
NOTE: Label value for variable
TESTB has been truncated to a
length of 40.
RUN;
```

These two labels will both be truncated to 40 characters, which results in both variables having the same label: "PARTICIPANT RESULTS FROM LABORATORY TEST". The information about test number is lost.

This problem also comes up when you use PROC FORMAT to label variable values. Any labels longer than 16 characters will be truncated. Be sure to double-check and/or correct your variable and format labels if you get the truncation message.

Check #4: Data Merges

Once you've explored and either fixed or explained all of the ERROR, WARNING, and NOTE messages in your SAS log, the harder part of debugging begins. You have now found all of the problems that SAS knows how to identify. Now you have to start looking for the kind of problems that only a programmer can spot.

The first place to start checking is any merge statements in your program. If you have a mistake here, it will have a major impact on your results.

One of the most common MERGE errors is to merge two datasets and forget to include a BY statement. While there are times that you will want to do this, they are extremely rare.

```
DATA BOTH;
  MERGE THISDATA THATDATA;
RUN;
NOTE: The data set WORK.BOTH has
      250 observations and 6 variables.
```

As you can see, SAS does not generate a WARNING or even a NOTE when the BY statement is missing. All that happens is that you get a one-to-one merge (see the SAS\BASE manual³ for a description of what happens).

The other main check to do on your MERGE statements is to be sure you have correctly handled the cases where a record is on one dataset or the other but not both. If SAS does not find a matching observation for one observation in a merge, the default is to add it to the output dataset anyway.

You may want this to happen. If you don't, you need to write code to handle the mismatches. The following example shows how you can use the IN= statement⁴ to control your merges.

```
DATA HASBOTH OOPS;
  MERGE THISDATA (IN=INTHIS)
        THATDATA (IN=INTHAT);
  BY ID;
  IF NOT(INTHIS AND INTHAT)
    THEN OUTPUT OOPS;
  ELSE OUTPUT HASBOTH;
RUN;
PROC PRINT DATA=OOPS;
  VAR ID THISVAR THATVAR;
  TITLE 'ERROR: OBSERVATIONS NOT
        FOUND ON BOTH DATASETS';
RUN;
```

Creating a dataset called OOPS is a great technique to use throughout your code. Any time you test for an error, output the observations with the error to a dataset called OOPS. Then when you review the log file, any time you see a dataset called OOPS with more than 0 observations, you know you need to check for a problem in the program.

```
NOTE: Dataset OOPS has 15
      observations;
```

...

Check #5: Observation Counts

SAS gives you messages at every step of your program that tell you how many observations are in each dataset. It is important to check these numbers carefully to be sure you get the number of observations you expected.

For example, in the following output, the number of observations drops dramatically after the data step.

```
PROC SORT DATA=MOREDAT; RUN;
NOTE: The data set WORK.MOREDAT
      has 224 observations and 10
      variables.
```

```
DATA LESSDAT;
  SET MOREDAT;
  IF AGE>21;
RUN;
NOTE: The data set WORK.LESSDAT
      has 9 observations and 10
      variables.
```

Perhaps this was supposed to happen, or maybe this is an unintended result of missing data on the AGE variable. This is a case where SAS can't help you out. You need to know your data and have an idea what to expect.

If you do not know why the observation count changed, print out some records before and after the data step or procedure and review the changes.

³ "Examples of One-to-One Merging", p. 148, *SAS Language: Reference*.

⁴ "Data Set Options: IN=", p. 723, *SAS Language: Reference*.

Check #6: Constructed Variables

Most SAS programs have data steps where you calculate new variables from existing data. Sometimes these calculations can go awry. Whenever you are calculating a new variable, it's a good idea to print out some observations and double-check the calculation.

For example, in the following code, a PROC PRINT is used to print out 50 test records from the output dataset to check the result of the calculations.

```
DATA THATDAT2;  
  SET THATDAT;  
  IF AGE<5 THEN KID=1;  
  ELSE KID=0;  
RUN;  
PROC PRINT DATA=THATDAT2 (OBS=50);  
  VAR AGE KID;  
RUN;
```

TEST CALCULATION KID

OBS	AGE	KID
1	4	1
2	.	1
3	6	0
4	5	0
5	9	0
6	.	1
7	8	0
8	.	1
9	6	0
...		

Here the missing data cause KID to be set to 1 when age is unknown. This is because a value of missing is less than 5 (the test condition). To fix this problem, the syntax should be IF 0<AGE<5 THEN KID=1.

Check #7: Data-Specific Checks

The final check on the list is up to you. Each dataset is different, and may require its own checks. For example, if your data file contains academic data for high school freshman, you might want to do a range check on the age variable to make sure that you don't have any 54-year-old freshmen (unless the school in question caters to older students).

Another type of check you can do is some basic frequency distributions. If your school data can be broken down by class, you might want to check to see that the class sizes look reasonable. A class with 2000

students or a class with 1 student might be a sign of a data problem.

Some datasets may need a review of their missing data codes. If your data come from another software package, odds are that the missing data are not identified by periods, as in SAS. The data might turn up with a special numeric code (0, 9, or 99999, for example) or with a character code (X or N/A). In order for SAS to understand the data they need to be recoded to the SAS missing value code.

You can build up a checklist of your own that lists specific things to look for in your data. Another way to build a custom checklist is to think back over the last few months and write up a list of all of the mistakes that showed up in your code or output.

Now take this list and figure out which problems would have been spotted if you had implemented checks 1-7 above. Cross those out. Then figure out what kind of check it would take to find the remaining problems on your list. These are the items that you need to add to the checklist.

In the future, if a mistake slips through even after you've followed the checklist, just come up with an appropriate new check and add it to the list.

A complete sample checklist is shown in Appendix A.

Conclusions:

Reviewing and debugging your SAS output can save you a lot of embarrassment, but it takes time. You may not want to take the time to review every program you write in such detail, but there are situations where output review is essential.

Any time you are creating a production program (a program that will be run again and again), take the time to check it thoroughly. Any time you've created a complex program and you're not 100% convinced you've got it exactly right, ask someone to review the output for you.

In fact, any time you can, swap output with another programmer when you do these reviews. You are more likely to spot problems in someone else's output. Also, you may pick up some new programming techniques by looking at other people's work.

I encourage you to build your own checklist and give it a trial run. Our experience is that you gain both confidence and peace of mind by taking the time to carefully check your work.

References:

SAS Institute Inc., *SAS Language: Reference*,
Version 6, First Edition, Cary, NC: SAS Institute
Inc., 1990.

SAS Institute Inc., *SAS Guide to the SQL Procedure:
Usage and Reference*, Version 6, First Edition, Cary,
NC: SAS Institute Inc.

Contacting the Author:

Please direct any questions or feedback to the author
at:

Kaiser Permanente Center for Health Research
3800 N. Kaiser Center Drive
Portland, OR 97227-1098

E-mail: haworthla@chr.mts.kpnw.org

Appendix A: Sample Checklist

Program: _____		Date: _____	
Reviewer: _____			
Check	CHECK ONE:		Notes
	No Problems	Fixed/ Explained	
ERROR Messages	☐	☐	
WARNING Messages	☐	☐	
NOTE Messages	☐	☐	
Data Merges	☐	☐	
Observation Counts	☐	☐	
Constructed Variables	☐	☐	
Other:	☐	☐	
Other:	☐	☐	