

Don't Get Blindsided by PROC COMPARE

Joshua Horstman, Nested Loop Consulting, Indianapolis, IN
Roger Muller, Data-to-Events.com, Carmel, IN

ABSTRACT

"NOTE: No unequal values were found. All values compared are exactly equal." Do your eyes automatically drop to the end of your PROC COMPARE output in search of these words? Do you then conclude that your data sets match? Be careful here! Major discrepancies may still lurk in the shadows, and you'll never know about them if you make this common mistake. This paper describes several of PROC COMPARE's blind spots and how to steer clear of them. Watch in horror as PROC COMPARE glosses over important differences while boldly proclaiming that all is well. See the gruesome truth about what PROC COMPARE does, and what it doesn't do! Learn simple techniques that allow you to peer into these blind spots and avoid getting blindsided by PROC COMPARE!

INTRODUCTION

PROC COMPARE is a SAS® procedure which provides the programmer with a simple facility for comparing two data sets. While the procedure is capable of providing exhaustive listings of the differences between two data sets, it is frequently the case that the programmer is simply interested in confirming that two data sets are the same.

For example, a programmer may compare an output data set to a previous version of the same data set to verify that changes to the code which produced the data set have had no unintended consequences. When accuracy of results is critical, data sets are often "double programmed". The specified output is independently constructed by two separate programmers and the results are then compared.

In any case, the programmer often scans the PROC COMPARE output for a note at the bottom saying "No unequal values were found. All values compared are exactly equal." This note is taken as a definitive statement that the two data sets are absolutely identical. This is a serious mistake which can result in the programmer being blindsided.

Fortunately, there are ways to avoid being caught in this trap. Just as a rearview mirror provides a driver with additional information to help avoid being struck by an unseen vehicle, so the COMPARE procedure output provides the programmer with additional information to help avoid being struck by unseen data discrepancies. In both cases, the operator needs to be looking in the right places to eliminate any blind spots.

In this paper, we will review some of the common blind spots that occur in PROC COMPARE and describe where the savvy programmer should be looking to avoid them. The examples use clinical trial data, but the concepts here apply broadly to any industry or type of work.

BLIND SPOT #1: MISSING VARIABLES

Our first example will use the two demography (DM) data sets shown below. We have a production data set, DM_PROD, and a validation or QC data set, DM_QC. The validation programmer wishes to use PROC COMPARE to determine if the two data sets are the same.

DM_PROD			DM_QC	
SUBJID	AGE	SEX	SUBJID	AGE
101	45	M	101	45
102	37	F	102	37
103	61	F	103	61

Observe that the QC data set does not contain all of the variables that are in the production data set. Specifically, the SEX variable is not present. Below are the PROC COMPARE code and output.

```
proc compare base=dm_prod compare=dm_qc; run;
```

Output 1. Output from PROC COMPARE of DM_PROD and DM_QC.

```

                                The COMPARE Procedure
          Comparison of WORK.DM_PROD with WORK.DM_QC
                        (Method=EXACT)

                        Data Set Summary

Dataset              Created          Modified  NVar      NObs
WORK.DM_PROD  30MAR13:17:23:18  30MAR13:17:23:18      3         3
WORK.DM_QC    30MAR13:17:23:18  30MAR13:17:23:18      2         3

                        Variables Summary

Number of Variables in Common: 2.
Number of Variables in WORK.DM_PROD but not in WORK.DM_QC: 1.

                        Observation Summary

Observation          Base  Compare
First Obs            1      1
Last Obs             3      3

Number of Observations in Common: 3.
Total Number of Observations Read from WORK.DM_PROD: 3.
Total Number of Observations Read from WORK.DM_QC: 3.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 3.

NOTE: No unequal values were found. All values compared are exactly equal.
```

Notice the last line of the procedure output: “NOTE: No unequal values were found. All values compared are exactly equal.” The careless programmer will interpret this to mean that the job is finished. However, a more thorough inspection of the output reveals an important blind spot.

PROC COMPARE alerts us to the blind spot in two different ways. In the first section of the report, the “Data Set Summary”, we see that one data set has 3 variables while the other has only 2. Further down in the “Variables Summary”, the report indicates that there is 1 variable in WORK.DM_PROD but not in WORK.DM_QC.

These items are red flags to the programmer. Even though “all values compared are exactly equal,” not all values were compared. The COMPARE procedure simply compares the data sets based on the variables in common and reports its findings based on that comparison. If the goal of the validation programmer was to ensure that the SEX variable was derived correctly in the production data set, that goal has not been accomplished.

BLIND SPOT #2: MISSING OBSERVATIONS

Our second example will use the two vital signs (VS) data sets shown below. We have a production data set, VS_PROD, and a validation or QC data set, VS_QC. As before, the validation programmer wishes to use PROC COMPARE to determine if the two data sets are the same.

VS_PROD				VS_QC			
SUBJID	VISITNUM	SYSBP	DIABP	SUBJID	VISITNUM	SYSBP	DIABP
101	1	120	80	101	1	120	80
101	2	126	84	101	2	126	84
102	1	132	90	102	1	132	90
102	2	131	85				

Observe that the QC data set does not contain all of the observations that are in the production data set. Specifically, the record for subject 102 at visit 2 is not present. Below are the PROC COMPARE code and output.

```
proc compare base=vs_prod compare=vs_qc; run;
```

Output 2. Output from PROC COMPARE of VS_PROD and VS_QC.

```

The COMPARE Procedure
Comparison of WORK.VS_PROD with WORK.VS_QC
(Method=EXACT)

Data Set Summary

Dataset              Created              Modified  NVar      NObs
WORK.VS_PROD  30MAR13:17:23:18  30MAR13:17:23:18      4         4
WORK.VS_QC    30MAR13:17:23:18  30MAR13:17:23:18      4         3

Variables Summary

Number of Variables in Common: 4.

Observation Summary

Observation      Base  Compare
First Obs        1      1
Last Match       3      3
Last Obs         4      .

Number of Observations in Common: 3.
Number of Observations in WORK.VS_PROD but not in WORK.VS_QC: 1.
Total Number of Observations Read from WORK.VS_PROD: 4.
Total Number of Observations Read from WORK.VS_QC: 3.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 3.

NOTE: No unequal values were found. All values compared are exactly equal.

```

We find that familiar note at the bottom of the procedure output: "NOTE: No unequal values were found. All values compared are exactly equal." Once again, however, a more careful reading of the output is necessary.

Notice the "Data Set Summary" indicates one data set contains 4 observations while the other has only 3. We also see the "Observation Summary" that there is 1 observation in WORK.VS_PROD but not in WORK.VS_QC. Again, these are red flags. The comparison was performed based only on the observations in common. If there are any problems with the last record in the production data set (the record that is missing from the QC data set), we won't discover it because we have a blind spot.

Those familiar with the COMPARE procedure may be wondering if this blind spot can be avoided though the use of the ID statement. The ID statement is used to specify key variables that the COMPARE procedure should use to match observations. The logical ID variables in this example would be SUBJID and VISITNUM because the combination of these two variables uniquely identifies every observation. While the ID statement is extremely useful in certain situations, it does not help us here. The resulting report is very similar and still concludes with "NOTE: No unequal values were found. All values compared are exactly equal."

BLIND SPOT #3: CONFLICTING TYPES

For our third example, consider the two laboratory results (LB) data sets shown below. We have a production data set, LB_PROD, and a validation or QC data set, LB_QC. Once again, the validation programmer wishes to use PROC COMPARE to determine if the two data sets are the same.

LB_PROD				LB_QC			
SUBJID	VISITNUM	LBTESTCD	LBORRES	SUBJID	VISITNUM	LBTESTCD	LBORRES
101	1	ALB	3.6	101	1	ALB	13.6
101	1	ALP	47.2	101	1	ALP	57.2
101	1	AST	13.5	101	1	AST	23.5
101	1	BILI	0.8	101	1	BILI	10.8

This time, the difference is more subtle. In the production data set, LBORRES (the original lab result) is a numeric variable, but in the QC data set, it is a character variable. In addition to this difference, the results themselves have been altered in the QC data set to highlight the danger here. Below are the PROC COMPARE code and output.

```
proc compare base=lb_prod compare=lb_qc; run;
```

Output 3. Output from PROC COMPARE of LB_PROD and LB_QC.

The COMPARE Procedure				
Comparison of WORK.LB_PROD with WORK.LB_QC				
(Method=EXACT)				
Data Set Summary				
Dataset	Created	Modified	NVar	NObs
WORK.LB_PROD	30MAR13:18:57:21	30MAR13:18:57:21	4	4
WORK.LB_QC	30MAR13:18:57:21	30MAR13:18:57:21	4	4
Variables Summary				
Number of Variables in Common: 4.				
Number of Variables with Conflicting Types: 1.				
Listing of Common Variables with Conflicting Types				
Variable	Dataset	Type	Length	
lborres	WORK.LB_PROD	Num	8	
	WORK.LB_QC	Char	12	
Observation Summary				
Observation	Base	Compare		
First Obs	1	1		
Last Obs	4	4		
Number of Observations in Common: 4.				
Total Number of Observations Read from WORK.LB_PROD: 4.				
Total Number of Observations Read from WORK.LB_QC: 4.				
Number of Observations with Some Compared Variables Unequal: 0.				
Number of Observations with All Compared Variables Equal: 4.				
NOTE: No unequal values were found. All values compared are exactly equal.				

Despite the fact that the two data sets contain completely different results for LBORRES, we once again see the message “No unequal values were found. All values compared are exactly equal.” Even checking the number of variables and the number of observations in the “Data Set Summary” does not help us here.

We must look further down to the “Variables Summary” to see that there is 1 variable with conflicting types. The report also contains a new section called “Listing of Common Variables with Conflicting Types.”

What is important to note here is that variables with conflicting types are excluded from the comparison. This is a huge blind spot! The inattentive validation programmer is at serious risk here of declaring valid a production data set that may in fact be erroneous.

BLIND SPOT #4: MISMATCHED ID VARIABLES

As mentioned earlier, one solution sometimes suggested to avoid the blind spots described so far is the inclusion of an ID statement in the call to PROC COMPARE. Without an ID statement, PROC COMPARE simply matches observations based on their positions within the two data sets. The ID statement specifies one or more key variables PROC COMPARE will use to match observations. For that reason, the set of variables included in the ID statement should have a unique combination of values for every record within each data set.

While the ID statement can be very useful, it is not foolproof. It not only leaves us exposed to the previous blind spots but also has the potential to introduce a new blind spot. Consider the drug exposure (EX) data sets below. Again, we have a production data set, EX_PROD, and a validation or QC data set, EX_QC. As before, the validation programmer wishes to use PROC COMPARE to determine if the two data sets are the same.

EX_PROD			EX_QC		
SUBJID	VISITNUM	DOSE	SUBJID	VISITNUM	DOSE
101	1	3	101	1	3
101	2	4	101	2	4
102	1	5	102	1	5
102	2	6	102	3	7

The two data sets differ only in the last record. Note that both the visit number and the dose differ. In this example, the programmer uses an ID statement to ensure PROC COMPARE matches the records properly. Since there should only be one record for a given subject at a particular visit, the SUBJID and VISITNUM variables are included on the ID statement. The code and output are given below:

```
proc compare base=ex_prod compare=ex_qc;
  id subjid visitnum;
run;
```

Output 4. Output from PROC COMPARE of EX_PROD and EX_QC.

The COMPARE Procedure				
Comparison of WORK.EX_PROD with WORK.EX_QC				
(Method=EXACT)				
Data Set Summary				
Dataset	Created	Modified	NVar	NObs
WORK.EX_PROD	31AUG13:14:55:43	31AUG13:14:55:43	3	4
WORK.EX_QC	31AUG13:14:57:58	31AUG13:14:57:58	3	4
Variables Summary				
Number of Variables in Common: 3.				
Number of ID Variables: 2.				
Observation Summary				
Observation	Base	Compare	ID	
First Obs	1	1	subjid=101 visitnum=1	
Last Match	3	3	subjid=102 visitnum=1	
Last Obs	4	.	subjid=102 visitnum=2	
	.	4	subjid=102 visitnum=3	
Number of Observations in Common: 3.				
Number of Observations in WORK.EX_PROD but not in WORK.EX_QC: 1.				
Number of Observations in WORK.EX_QC but not in WORK.EX_PROD: 1.				
Total Number of Observations Read from WORK.EX_PROD: 4.				
Total Number of Observations Read from WORK.EX_QC: 4.				
Number of Observations with Some Compared Variables Unequal: 0.				
Number of Observations with All Compared Variables Equal: 3.				
NOTE: No unequal values were found. All values compared are exactly equal.				

Even though the two data sets have different values in the fourth observation, we are presented with the message “No unequal values were found. All values compared are exactly equal.” As with the previous example, checking the number of variables and the number of observations in the “Data Set Summary” doesn’t indicate the problem either.

However, if we dig into the “Observation Summary” section of the output, there are a few clues. In particular, PROC COMPARE explicitly tells us that EX_PROD contains 1 observation not in EX_QC, and vice-versa. It also tells us that there are 3 observations in common, even though each data set contains 4 observations.

Why then are we told that all values compared are exactly equal? Since we included an ID statement, PROC COMPARE only matches observations based on the values of those ID variables. If one data set contains an observation with ID variables having a combination of values that does not appear in the other data set, then no comparison can be made. That record is effectively excluded from the comparison. This is yet another stunning blind spot that can easily leave vulnerable even an experienced programmer.

RECOMMENDATIONS

In order to avoid leaving yourself exposed to one of the above blind spots, there are several good habits the SAS programmer should develop.

First, “know thy data”. Know what variables you have in each data set and their types. Know how many observations are in each data set and where they came from. Examine your data sets carefully to make sure they contain what you are expecting. While automated tools like the COMPARE procedure are tremendously powerful, they are no substitute for a good, working knowledge of the data.

Second, always review the entire PROC COMPARE output, particularly if your goal is to confirm that two data sets are identical. Don't be lulled into a false sense of security by seeing "NOTE: No unequal values were found. All values compared are exactly equal." Check the number of variables and the number of observations. If these don't match, all other conclusions about the data are suspect. Carefully inspect the "Variables Summary" and "Observations Summary" sections of the output to see if anything is amiss.

Finally, if your validation data set contains extraneous variables that aren't part of the production data set being validated, drop them before you call PROC COMPARE. In so doing, you will be able to more easily recognize when one data set is missing variables because the variable counts won't match.

CONCLUSION

Just as a skilled craftsman must be thoroughly familiar with the proper use of his tools, so the SAS programmer must understand how the various SAS procedures work. It's not enough to throw input at a procedure and blindly trust that the results mean what you think they should mean. In the case of PROC COMPARE, this means understanding what is actually being compared and what is not being compared.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Joshua M. Horstman
Nested Loop Consulting
8921 Nora Woods Drive
Indianapolis, IN 46240
317-815-5899
josh@nestedloopconsulting.com

Roger D. Muller
Data-to-Events.com
14475 Stephanie St.
Carmel, IN 46033
317-846-5782
rdmuller@hotmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.