Paper 129-2019

Tips for Correctly and Efficiently Comparing Two Files in SAS®

Aaron Brown, South Carolina Department of Education

ABSTRACT

This paper gives some tips for correctly and efficiently comparing two data files via SAS® programs, not just to locate if discrepancies exist but where they exist. This can be helpful if you need to compare two different versions of a file. The tips include thoughts on different methods of reading data into SAS, then code examples for the COMPARE and SQL procedure to compare the datasets.

INTRODUCTION

It is a common occurrence that we need to compare two different versions of a file. It could be that there are two different versions of the data and we are told to check what changed. Or, we are transitioning from one method or tool for making a file to a new one, and we want to make sure that they produce the same output from identical input. Also, it could be that two versions were generated as a way to validate the data. (For example, you assign two programmers to independently generate the same file, then check if their results are the same as verification. It is unlikely they would both make the identical mistake.)

For this paper, we will use an imaginary report, using dummy data. Let's say an office is told to generate a report of student enrollment counts. The report should include rows for the following counts:

- by School
- by School and Grade
- by School and Gender
- by School, Gender, and Grade

Also, if any student-level record has a missing Grade value, exclude it from the counts as an invalid record. Our data contains 1 school with 40 students, 10 of which have missing Grade values.

For our example, we will say (somewhat absurdly) that the company has decided to verify the file by having three programmers independently generate it. This will enable us to highlight how different types of differences appear.

Our dataset data1 is correct: it has all the required combinations of data and properly excluded records without a grade. Our dataset data2 has all the required combinations of data but failed to exclude records without a grade. (Note that it contains 40 students total, instead of just 30.) Our dataset data3 is as data1, except that it excluded the by School by Grade and Gender combination. The datasets are shown below.

dataset 1

SchoolID	Grade	Gender	Count
0001	03	M	10
0001	03	F	10
0001	03		20
0001	05	M	5
0001	05	F	5
0001	05		10
0001		M	15
0001		F	15
0001			30

dataset 2

SchoolID	Grade	Gender	Count
0001	03	M	10
0001	03	F	10
0001	03		20
0001	05	M	5
0001	05	F	5
0001	05		10
0001		M	20
0001		F	20
0001			40

dataset 3

SchoolID	Grade	Gender	Count
0001	03		20
0001	05		10
0001		M	15
0001		F	15
0001			30

Table 1. Contents of our Sample Datasets

First, in Step 1, we will look at issues when getting the data ready for comparison. Then, in Step 2, we will compare the data with the COMPARE procedure. In Step 3, we will look into specific details of discrepancies via the SQL procedure.

See appendices 1 and 2, at the end of this paper, for the full SAS code to generate all output in this paper.

STEP 1: READING YOUR DATA INTO SAS

If you are comparing SAS datasets, then this step is essentially a non-issue; the data is already ready-to-use in SAS. But, often, we are required to generate CSV files or Excel files. When that is the case, you need to read the data in those files back into SAS before you can compare them. It is important to make sure you read them in in such a way that your comparison is valid. (Plus, it is embarrassing to tell someone they screwed up, when it was actually your program that screwed up.)

If your data is a CSV or text file, I highly recommend using the DATA step to read in data. The IMPORT procedure can be used, but it can cause unexpected results, such as converting something to numeric when you want it as a character string.

As an example, here is what our data1 dataset looks like if read in through PROC IMPORT.

SchoolID	Grade	Gender	Count
1	3	M	10
1	3	F	10
1	3		20
1	5	M	5
1	5	F	5
1	5		10
1	.	M	15
1	.	F	15
1	.		30

Output 1. Dataset 1 via PROC IMPORT

All the observations were read into SAS, but SchoolID and Grade were rendered as numeric variables. This caused their leading zeroes to be dropped. This change will negatively impact our comparisons via PROC COMPARE and PROC SQL.

If you must use PROC IMPORT, it can be helpful to compare the character types and lengths of the datasets you plan to compare. You could do this programmatically through the CONTENTS and COMPARE procedures or by glancing at the datasets.

If importing Excel spreadsheets through PROC IMPORT, the MIXED option can sometimes help prevent variable type issues. You can also try using LIBNAME XLSX, to link directly to the Excel document; a blog with details about using that is in the references of this paper. But, in general, using Excel means things will be a little bit trickier and you might need to add some code to undo any assumptions SAS made while reading in the data.

For the rest of this paper, we will assume that the data were read into SAS properly.

STEP 2: COMPARE THE DATA

Comparing the datasets is relatively easy by using PROC COMPARE. If you expect no discrepancies (as is our expectation here), I recommend using the LISTALL option; if you expect several discrepancies, you may want to exclude it lest it generate too much clutter. If the order of the observations should be identical, you can simply run code like the below:

```
PROC COMPARE DATA=data1 COMPARE=data2 LISTALL;
RUN;
```

We see that the 7th, 8th, and 9th observations had discrepancies.

Obs	Base Count	Compare Count	Diff.	% Diff
7	15.0000	20.0000	5.0000	33.3333
8	15.0000	20.0000	5.0000	33.3333
9	30.0000	40.0000	10.0000	33.3333

Output 2. PROC COMPARE Without ID Statement

However, this method will only identify differences by observation number. Although that is fine for telling you that discrepancies exist, it is not very helpful for troubleshooting where or why discrepancies occurred. To see more details, or if order does not matter, you can declare some ID variables.[1] First use the SORT procedure to sort the datasets by the variables, then run PROC COMPARE, as follows:

```
PROC SORT DATA=data1; BY SchoolID Grade Gender; RUN;
PROC SORT DATA=data2; BY SchoolID Grade Gender; RUN;
PROC COMPARE DATA=data1 COMPARE=data2 LISTALL;
     ID SchoolID Grade Gender;
RUN;
```

Now we can see that the discrepancies occurred at the SchoolID-level and the SchoolID-by-Gender-level.

SchoolID	Grade	Gender	Base Count	Compare Count	Diff.	% Diff
0001			30.0000	40.0000	10.0000	33.3333
0001		F	15.0000	20.0000	5.0000	33.3333
0001		M	15.0000	20.0000	5.0000	33.3333

Output 3. PROC COMPARE With ID Statement

That is much more helpful for telling the programmers what is wrong and what they need to double-check.

PROC COMPARE can also tell you if something is in one dataset but missing from another one. We will show that by comparing data1 and data3 (after sorting data3). Its comparison shows no differences in counts!

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

Output 4. PROC COMPARE With No *Compared* Values Unequal

However, if we look in the rest of PROC COMPARE's output, we see that some observations were missing.

[1] You could save the ID variables to a macro variable to limit the number of times you need to type them out. Doing so can help decrease errors via typos.

```
Observation 5 in WORK.DATA1 not found in WORK.DATA3: SchoolID=0001
 Grade=03 Gender=F.

Observation 6 in WORK.DATA1 not found in WORK.DATA3: SchoolID=0001
 Grade=03 Gender=M.

Observation 8 in WORK.DATA1 not found in WORK.DATA3: SchoolID=0001
 Grade=05 Gender=F.

Observation 9 in WORK.DATA1 not found in WORK.DATA3: SchoolID=0001
 Grade=05 Gender=M.
```

Output 5. PROC COMPARE Showing Missing Observations

The LISTALL option will also explicitly list if any variables are in one dataset but not in the other. Such is not the case in our examples, but it good to be aware that this procedure does check it. (Without the LISTALL option, some of the output will state there are a different number of variables, but it is not as easy to notice.)

When using PROC COMPARE, it is important to be aware of how its output flags different types of discrepancies as well as situations when it is hard to notice that discrepancies occurred. For more detail on how to safely understand PROC COMPARE output, check the Recommended Reading section.

TOPIC 3: PROC SQL TO FIND DISCREPANCIES

If there are a lot of records in one dataset but not in another, PROC COMPARE can become too cluttered to really understand what is going on. When I was faced with that situation, I found a way to use PROC SQL to view the discrepancies in a cleaner, more orderly manner by using the EXCEPT set operator. [2] (And even if PROC COMPARE is comprehensible, this output is probably easier for explaining the issue to someone not used to PROC COMPARE.)

The EXCEPT operator limits the SQL output to just observations that are not in whatever you are excepting from. In a sense, it means "SELECT everything EXCEPT these". To fully compare two datasets, you run two SQL queries. In the example below, the first query checks for everything in data1 not in data2, and the second query checks for anything in data2 not in data1.

```
PROC SQL NUMBER;
      title2 'records in data1 not in data2';
      SELECT * FROM data1
      EXCEPT
      SELECT * FROM data2
      ;
      title2 'records in data2 not in data1';
      SELECT * FROM data2
      EXCEPT
      SELECT * FROM data1
      ;
QUIT;
```

[2] If your columns are in a different order between the datasets, use EXCEPT CORR instead of just EXCEPT. You can just use EXCEPT CORR in either case if you want to make certain SAS is comparing the correct columns. Note that other set operators also exist and can be handy for organizing data or conducting quality control. See the online documentation or the *Advanced Programming SAS Certification* guide for details.

PROC SQL Checks

records in data1 not in data2

Row	SchoolID	Grade	Gender	Count
1	0001			30
2	0001		F	15
3	0001		M	15

PROC SQL Checks

records in data2 not in data1

Row	SchoolID	Grade	Gender	Count
1	0001			40
2	0001		F	20
3	0001		M	20

Output 6. PROC SQL Data1 and Data2 (with Counts)

If you do not want to see discrepancies in counts, but rather just see missing observations, you can accomplish this by DROPping the variable that changes: in our example, Count. Here is the comparison between data1 and data3, dropping the Count variable. (The second query has no output since data1 was not missing any data.)

```
PROC SQL NUMBER;
      title2 'records in data1 not in data3';
      SELECT * FROM data1(DROP=Count)
      EXCEPT
      SELECT * FROM data3(DROP=Count)
      ;
      title2 'records in data3 not in data1';
      SELECT * FROM data3(DROP=Count)
      EXCEPT
      SELECT * FROM data1(DROP=Count)
      ;
QUIT;
```

PROC SQL Checks

records in data1 not in data3

Row	SchoolID	Grade	Gender
1	0001	03	F
2	0001	03	M
3	0001	05	F
4	0001	05	M

PROC SQL Checks

records in data3 not in data1

Output 7. PROC SQL Data1 and Data3 (No Counts)

Either method gives useful information. The first gives more details, but might seem cluttered to someone if they care more about missing observations. The second omits the Count discrepancies, but highlights missing observations.

CONCLUSION

This paper has presented some tips and tools for checking if two files are identical or not. My hope is that, whether for validation via dual processing, checking between versions, or some other purpose, the ideas in this paper will be helpful. I should also note that similar results could, at least in theory, be gained through checksums or hashing methods; however, I am not familiar enough with those methods to recommend them, nor have I found an instance where that would be preferable to methods shown here.

APPENDIX 1: TEXT FILES

To run the SAS code in Appendix 2, save the contents below into three text files and set the file directory as the macro variable &dir.

data1.csv
SchoolID,Grade,Gender,Count
0001,03,M,10
0001,03,F,10
0001,03,,20
0001,05,M,5
0001,05,F,5
0001,05,,10
0001,,M,15
0001,,F,15
0001,,,30

data2.csv
SchoolID,Grade,Gender,Count
0001,03,M,10
0001,03,F,10
0001,03,,20
0001,05,M,5
0001,05,F,5
0001,05,,10
0001,,M,20
0001,,F,20
0001,,,40

data3.csv
SchoolID,Grade,Gender,Count
0001,03,,20
0001,05,,10
0001,,M,15
0001,,F,15
0001,,,30

APPENDIX 2: SAS CODE

```
*STEP 1: READING IN DATA;
*&dir is a macro variable with the folder location for the CSV files;
*in the DATA steps below, lrecl is set to be arbitrarily large;
data data1;
      infile "&dir\data1.csv" firstobs=2 dsd missover pad lrecl=1000;
      input SchoolID :$4. Grade :$2. Gender :$1. Count :8.;
run;
proc print noobs; title 'dataset 1'; run;
data data2;
      infile "&dir\data2.csv" firstobs=2 dsd missover pad lrecl=1000;
      input SchoolID :$4. Grade :$2. Gender :$1. Count :8.;
```

```
run;
proc print noobs; title 'dataset 2'; run;
data data3;
     infile "&dir\data3.csv" firstobs=2 dsd missover pad lrecl=1000;
     input SchoolID :$4. Grade :$2. Gender :$1. Count :8.;
run;
proc print noobs; title 'dataset 3'; run;

Title 'dataset 1 via PROC IMPORT';
proc import datafile="&dir\data1.csv" dbms=csv replace out=data1_imp;
     getnames=yes;
run;
proc print noobs; run;

*STEP 2: PROC COMPARE;
Title 'Comparing the Datasets';
Title2 'no ID statement';
PROC COMPARE DATA=data1 COMPARE=data2 LISTALL;
RUN;
title2 'with ID statement';
PROC SORT DATA=data1; BY SchoolID Grade Gender; RUN;
PROC SORT DATA=data2; BY SchoolID Grade Gender; RUN;
PROC COMPARE DATA=data1 COMPARE=data2 LISTALL;
     ID SchoolID Grade Gender;
RUN;

PROC SORT data=data3; BY SchoolID Grade Gender; run;
PROC COMPARE DATA=data1 COMPARE=data3 LISTALL;
     ID SchoolID Grade Gender;
RUN;

*STEP THREE: PROC SQL;
Title 'PROC SQL Checks';
PROC SQL NUMBER;
     *show all discrepancies;
     title2 'records in data1 not in data2';
     SELECT * FROM data1
     EXCEPT
     SELECT * FROM data2
     ;
     title2 'records in data2 not in data1';
     SELECT * FROM data2
     EXCEPT
     SELECT * FROM data1
     ;


     *using DROP;
     title2 'records in data1 not in data3';
     SELECT * FROM data1(DROP=Count)
     EXCEPT
     SELECT * FROM data3(DROP=Count)
```

```
      ;
      title2 'records in data3 not in data1';
      SELECT * FROM data3(DROP=Count)
      EXCEPT
      SELECT * FROM data1(DROP=Count)
      ;
QUIT;
```

ACKNOWLEDGMENTS

The author thanks the WUSS chair and staff for this conference and the opportunity to present.

RECOMMENDED READING

Hemedinger, Chris. “Using LIBNAME XLSX to read and write Excel files.” https://blogs.sas.com/content/sasdummy/2015/05/20/using-libname-xlsx-to-read-and-write-excel-files/. Accessed 21 December 2018.

Horstman, Joshua and Roger Muller (2016), “Don’t Get Blindsided by PROC COMPARE,” 2016 Western Users of SAS Software, San Francisco, CA, USA.

SAS Institute Inc. 2011. *SAS® Certification Prep Guide: Advanced Programming for SAS®9, Third Edition*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Aaron R. Brown
South Carolina Department of Education
1429 Senate Street, Columbia, SC 29201
Work Phone: 803-734-8858
E-mail: ARBrown@ed.sc.gov