PharmaSUG 2012 – Paper TF14

Comparing Datasets: Using PROC COMPARE and Other Helpful Tools

Deb Cassidy
PPD, Morrisville, NC

ABSTRACT

There may be many reasons to compare datasets including working in a double-programming environment, determining if your code revisions worked as expected, and determining the impact from raw data updates. PROC COMPARE works great in many cases and you need nothing more than the lines

```
  proc compare data=old_data comp=new_data;
  run;
```

However, sometimes you get so many pages of differences that you are at a loss as to where to begin. If you want your datasets to be identical, this paper will cover examples of PROC COMPARE options and other helpful tools to get to everyone's favorite line of output:

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

If you are expecting differences, the paper will cover ways of making it easier to see your differences.

INTRODUCTION

At first glance, the COMPARE procedure is a simple procedure in that it compares two datasets and provides you a brief summary about the two datasets and shows you variables that differed. The basic output lists:

Output Item	**Used to check:**
Name of each dataset.	Do you have the right libname/datasets being compared?
Time each dataset was created and last modified.	Do you have the right version of the datasets?
Number of variables in each dataset.	Do you have the expected number of variables and are they the same for both datasets?
Number of observations in each dataset.	Do you have the expected number of observations and are they the same for both datasets?
Number of variables in common. If all variables are not in common, the number of differences will be listed.	Do you have the same variables?
Number of observations in common, number read from each dataset, number with all variables the same and number with some variables different	Do you have any differences in the number of observations or values in those observations?
If everything didn't match, a list of the differences up to the maximum allowed.	Which observations didn't match and what the values were from each dataset.

Now that you have the basic output and know there are differences, where do you begin? This paper will first address the case where you what your datasets to be identical. This will be followed at how to look at results a little differently when they should not match.

STEP 1 – GETTING THE RIGHT VARIABLES

I believe there are two things you must always resolve first before looking at the actual values. Some people will jump to the end of the output and see

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

They think they are done and move on. Unfortunately, this doesn't always mean the dataset was an exact match. If you don't have the same number of observations and the same number of variables, you can hide problems or create problems where they really don't exist.

My preference is to make sure I have the same number of variables and the exact same variables before I check anything else. The first part of the output has a dataset summary

```
                 Data Set Summary

Dataset              Created          Modified     NVar    NObs

WORK.OLD_DATA  23MAR12:14:14:58  23MAR12:14:14:58    20       1
WORK.NEW_DATA  23MAR12:14:14:58  23MAR12:14:14:58    20       1
```

Looks great, right? When you look at the next lines in the output, you discover a problem.

```
          Variables Summary

Number of Variables in Common: 19.
Number of Variables in WORK.OLD_DATA but not in WORK.NEW_DATA: 1.
Number of Variables in WORK.NEW_DATA but not in WORK.OLD_DATA: 1.
```

Even though both datasets have 20 variables, they aren't the same 20 variables. The default output leaves you guessing as to which variables aren't in the both datasets. I recommend you ALWAYS add the LISTALL option so you don't have to run the compare twice when this situation happens.

```
Listing of Variables in WORK.OLD_DATA but not in WORK.NEW_DATA
          Variable  Type  LengtH
            alpha     Num        8


Listing of Variables in WORK.NEW_DATA but not in WORK.OLD_DATA
            Variable  Type  Length
             alphab    Char       1
```

You quickly spot that dataset NEW_DATA has an extra 'b' in the variable name. You think it shouldn't be there so you modify your dataset and run again.

You get somewhat different output this time. Yes, all the variables are now in both datasets but you see information about a conflict.

```
                               Variables Summary

                 Number of Variables in Common: 20.
                 Number of Variables with Conflicting Types: 1.


           Listing of Common Variables with Conflicting Types

                Variable  Dataset        Type  Length

                 alpha     WORK.OLD_DATA  Num        8
 .                         WORK.NEW_DATA  Char       1
```

If you had looked a little closer at the previous output, you would have noticed that not only were ALPHA and ALPHAB not spelled the same, they are not the same variable type. SAS can't compare values in this case either. A numeric variable equal to 2 and a character variable equal to 2 just aren't the same even though your eyes might think so.

If your variables are the same type but you have other attributes that differ such as labels or formats, you'll also see that difference in your output. In many cases, this would not impact the rest of your checking but you would need to clean up the issues to have truly identical datasets.

What happens in the case where you really do need to have different variable names but you need them to be compared? You just need to add a couple statements to your proc.

```
proc compare data=old_data comp=new_data;
   var  alpha   beta ;
   with alphab  beta;
run;
```

Just remember to list ALL your variables and not just the ones with different names. The comparison will only be done on the variables in the VAR statement. You may also have a reason that you only want to compare a few variables so the VAR statement is useful to restrict the comparisons to the ones you want. If have datasets with a large number of variables, you will probably want to restrict the list of variables checked with each pass so you can resolve issues in manageable chunks.

STEP 2 – GETTING THE RIGHT OBSERVATIONS

So you've finally straightened out your variable names and attributes. Now you can move on to looking at your number of observations.

If you have a dataset with one observation per a single identifying variable such as one observation per subject, figuring out where you differ usually isn't too hard. Look in the output for that identifying variable and check the values for the first case that differs. You may be able to quickly spot other differences as well. Your challenge is to figure out why one dataset includes that record and the other one doesn't. Sometimes it will be real obvious such as one dataset includes all subjects while the other dataset was restricted to ITT subjects. Other cases may take more investigation.

However, what if you have more than one record per identifying variable? I find it easier to use to step away from proc compare to start identifying the problems. The following code counts the number of records for the identifying variable and creates a dataset to let you check for differences.

```
proc sql;
create table old_count as
 select subject, count(*) as old_count
 from old_data
group by subject;

 create table new_count as
 select subject, count(*) as new_count
 from new_data
group by subject;

quit;

data diffs;
     merge old_count
            new_count;
     by subject;
run;
```

Now I can look at my diffs dataset interactively to start identifying my problems. Check whether you have subjects in one dataset and not the other. When you do have matching subjects, do they have the same number of observations? If not, does one dataset consistently have more observations? Once you see patterns, you can start to figure out why those subjects are different. There will be times where you just don't see a pattern and you may feel like someone used a random-number generator to give you the observation counts. You may have no choice but to start looking at subjects one at a time and manually figuring out which dataset has the right number of observations. Also, you may have multiple issues causing the differences,

If you have several variables that are needed to uniquely identify records, you'll want to repeat the above process with additional variables once you finally have the same number of records per subject. For example, let's assume you have lab data with multiple visits and multiple lab tests. You'll want to recheck your observation count with something like:

```
proc sql;
create table old_count as
 select subject, labtest count(*) as old_count
 from old_data
group by subject, labtest;

create table new_count as
 select subject, labtest, count(*) as new_count
 from new_data
group by subject, labtest;

quit;
```

```
data diffs;
     merge old_count
            new_count;
     by subject labtest;
run;
```

You might discover that you have the same number of observations for a subject but different lab tests or a different number of visits. Keep working through the process and adding variables until you are comfortable that you really have the same number of observations. I've been known to use all the variables required to identify a unique record. That gave me a count of 1 for both new_count and old_count. If any of your key variables were derived, you may really want to consider doing this step.

STEP 3 – COMPARING VALUES

Now that you finally have the right variables and matching number of observations, you can actually start looking at the comparison of your data values.

In addition to the dataset and variable summary sections discussed above, the standard output provides an observation summary and a value summary. This lets you know how "severe" your differences are. If you only have 1 variable that differs for all records you'll probably have a lot less work than if you have 20 variables that differ for half your records.

```
                                Observation Summary

                            Observation      Base  Compare

                            First Obs           1        1
                            First Unequal       1        1
                            Last  Unequal     100      100
                            Last  Obs         100      100

      Number of Observations in Common: 100.
      Total Number of Observations Read from WORK.OLD_DATA: 100.
      Total Number of Observations Read from WORK.NEW_DATA: 100.

      Number of Observations with Some Compared Variables Unequal: 100.
      Number of Observations with All Compared Variables Equal: 0.


                       Values Comparison Summary

      Number of Variables Compared with All Observations Equal: 19.
      Number of Variables Compared with Some Observations Unequal: 1.
      Total Number of Values which Compare Unequal: 100.
      Maximum Difference: 2.


                    Variables with Unequal Values

             Variable  Type  Len   Compare Label    Ndif   MaxDif

             alpha     NUM     8   This is a label   100    2.000
```

```
                         Value Comparison Results for Variables

           __________________________________________________________
                       ||  This is a label
                       ||       Base    Compare
                   Obs ||      alpha      alpha      Diff.     % Diff
               ________  ||  _________  _________  _________  _________
                         ||
                      1  ||     5.0000     7.0000     2.0000    40.0000
                      2  ||     5.0000     7.0000     2.0000    40.0000
. . . additional differences . . .

                     49  ||     5.0000     7.0000     2.0000    40.0000
                     50  ||     5.0000     7.0000     2.0000    40.0000

NOTE: The MAXPRINT=50 printing limit has been reached for the variable alpha. No more
values will be printed for this comparison.
                ______________________________________________________
```

By default, you'll see just the first 50 differences. If you have a large number of variables with lots of differences, you may also get this message:

```
NOTE: The MAXPRINT=(50,500) printing limit has been reached. No more values will be
printed.
```

This means that the overall number of differences was stopped at 500 so you won't even see all the variables with differences. This is to prevent you getting hundreds or thousands of pages of output. You can override this with the maxprint option. For example, the following code would restrict the differences to 20 per variable and 1000 overall.

```
proc compare data=old_data comp=new_data listall maxprint=(20,1000);
run;
```

If you have a large number of differences, you may want to use the VAR statement so you only see selected variables first. This might be "key" variables or perhaps variables that have a logical relationship with each other such as all your date variables. I've had datasets with 200-300 variables and 20,000+ records so multiple submissions with just a few variables made the task manageable. Just don't forget that your final run would need all variables. There's always a chance that corrections to one variable caused differences in a variable that was previously matching.

CLOSE BUT NOT EXACT

Sometimes you may look at your output and wonder why there is a difference. For example, you think both datasets have a value of 3. However, when you look more closely at the data you see the first dataset really has a value of 3.000000001 and the second dataset is exactly 3. This is common when you have a derived variable where the two programs used a different combination of functions to do the calculations. It might also occur if the datasets were generated under difference operating systems as precision can vary. You will want to confirm that neither person did rounding or something that really created different values.

```
                         ||       Base    Compare
                     Obs ||          q          q      Diff.     % Diff
               ________  ||  _________  _________  _________  _________
                         ||
                     1  ||     3.0000     3.0000      -1E-9  -3.333E-8
                     2  ||     3.0000     3.0000      -1E-9  -3.333E-8
```

Once you have confirmed these values really should be treated as equal, you just need to specify the CRITERION option with a large number of decimal places. I suggest using 1 decimal place less than what you have identified as the real value. You don't want to use something like .01 because it can make real differences appear equal.

```
proc compare data=old_data comp=new_data listall criterion=.00000001;
  var q;
run;
```

Your output will now show that the values were treated as equal but not EXACTLY equal.

```
                    Total Number of Values which Compare Unequal: 0.
                    Total Number of Values not EXACTLY Equal: 100.
                    Maximum Difference Criterion Value: 3.3333E-10.
```

You can also decide that you want to consider a missing value as an acceptable match. NOMISSBASE will take any missing values in the base dataset and consider it as matching the comparison dataset. NOMISSCOMP does the same for the comparison dataset. And of course there needs to be an option for matching missing values in either dataset so NOMISSING is available.

IDENTIFYING RECORDS

The observation number isn't always meaningful. Adding an ID statement can provide additional information that may speed up the process to determine why you have differences. For example, in the output below I can quickly tell that subjects from site 0003 had differences but subjects in sites 0001 and 0002 matched. Then I just need to figure out why one site might be different.

```
proc compare data=old_data comp=new_data listall;
  var q;
  id subject;
run;
```

```
                     Value Comparison Results for Variables

           __________________________________________________________
                      ||  This is a label
                      ||       Base    Compare
            subject   ||      alpha      alpha       Diff.     % Diff
            _________ ||  __________ __________  __________  __________
                                               ||
        0003-001  ||     5.0000     7.0000     2.0000    40.0000
        0003-002  ||     5.0000     7.0000     2.0000    40.0000
        0003-003  ||     5.0000     7.0000     2.0000    40.0000
        0003-004  ||     5.0000     7.0000     2.0000    40.0000
        0003-005  ||     5.0000     7.0000     2.0000    40.0000
        0003-006  ||     5.0000     7.0000     2.0000    40.0000
        0003-007  ||     5.0000     7.0000     2.0000    40.0000
```

You can have several ID variables.

```
proc compare data=old_data comp=new_data listall;
  var q;
  id subject trt;
run;
```

In this example, it is obvious that the differences are tied to treatment B. How many ID variables you can have depends on their length. If you have more than can fit, you will have a message in the output indicating that LINESIZE was too small and listing which ID variables were not printed. Depending upon your system and printer setup, you can increase the LINESIZE and/or use NOCENTER to fit more onto the page.

```
                 Value Comparison Results for Variables

    _______________________________________________________________
                      ||  This is a label
                      ||       Base    Compare
     subject   trt    ||      alpha      alpha       Diff.     % Diff
     _________ ___    ||  __________ __________  __________  __________
                      ||
     0003-001  B      ||     5.0000     7.0000      2.0000    40.0000
     0003-003  B      ||     5.0000     7.0000      2.0000    40.0000
     0003-005  B      ||     5.0000     7.0000      2.0000    40.0000
     0003-007  B      ||     5.0000     7.0000      2.0000    40.0000
     0003-009  B      ||     5.0000     7.0000      2.0000    40.0000
     0003-011  B      ||     5.0000     7.0000      2.0000    40.0000
```

If your ID variables don't yield unique records, you will get warnings in both the log and output. Of course, if you have done the SQL steps above to get the same number of records and identified your key variables, you won't have this problem.

If your datasets are not sorted by the key variables, you can still do a comparison. Adding NOTSORTED to the ID statement will cause a one-to-one comparison.

```
proc compare data=old_data comp=new_data listall;
  var q;
  id subject trt notsorted;
run;
```

BY STATEMENT

Almost every proc allows a BY statement and PROC COMPARE is no different. If you add a BY statement, you get the same proc compare results split out by each BY group. There is one major drawback to this option. The number of pages of output will increase. The output for my sample dataset went from 4 pages to 201 pages by adding a BY statement. You can also add NOTSORTED to your BY statement if the data isn't sorted. A one-to-one match will be performed. You can also use the DESCENDING option if variables in the BY statement are sorted that way. Just specify DESCENDING as part of the BY statement immediately before the appropriate variable(s).

MORE PRINTING OPTIONS

There are even more options that can control the output. You can use NOVALUES just to see the summary. NOSUMMARY will give you the differences without the summary. BRIEFSUMMARY (or BRIEF) will give you a reduced summary of differences. STATS provides summary stats for all numeric variables that had differences. ALLSTATS will provide summary stats for all numeric variables.

The following options are not recommended with a large dataset as you could end up with thousands of pages. ALLOBS gives you all records including the ones with matching values but just the variables that didn't have a 100% match. This option ignores the MAXPRINT option to restrict the number of observations displayed. ALLVARS gives you records including matching ones for all variables but does apply MAXPRINT to limit the output. PRINTALL combines ALLVARS, ALLOBS, ALLSTATS, LISTALL and WARNING.

LISTBASE, LISTCOMP, LISTOBS, LISTBASEOBS, LISTCOMPOBS, LISTBASEVAR, LISTCOMPVAR, LISTEQUALVAR, LISTVAR all control which observations and variables appear. These options could be very beneficial when you are comparing datasets that you know should not match such as when you are comparing datasets after a code change.

TRANSPOSE provides a different way of looking at the output. Instead of seeing a single variable with all the differences followed by the next variable and its differences, you look at everything for a single record followed by the next record. This method could be very useful if you have a set of related variables and you want to see which matched and which didn't for each record.

```
                  Comparison Results for Observations

    _OBS_1=1 _OBS_2=1:
    Variable    Base Value       Compare         Diff.        % Diff
       alpha      5.000000      7.000000      2.000000     40.000000
           q      3.000000      3.000000         -1E-9  -3.333334E-8

    _OBS_1=2 _OBS_2=2:
    Variable    Base Value       Compare         Diff.        % Diff
       alpha      5.000000      7.000000      2.000000     40.000000
           q      3.000000      3.000000         -1E-9  -3.333334E-8

    _OBS_1=3 _OBS_2=3:
    Variable    Base Value       Compare         Diff.        % Diff
       alpha      5.000000      7.000000      2.000000     40.000000
           q      3.000000      3.000000         -1E-9  -3.333334E-8
```

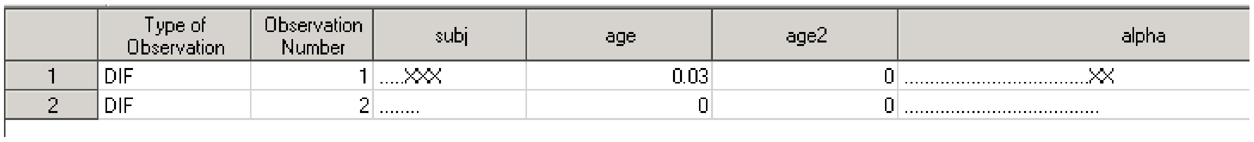
WRITING DIFFERENCES TO A DATASET

Personally, I rarely use most of the options above. I work in interactive SAS and like a dataset where I can move related variables together to see patterns or filter records. To write the differences out to a dataset, you simply use:

```
proc compare data=one comp=two out=comparisons;
Run;
```

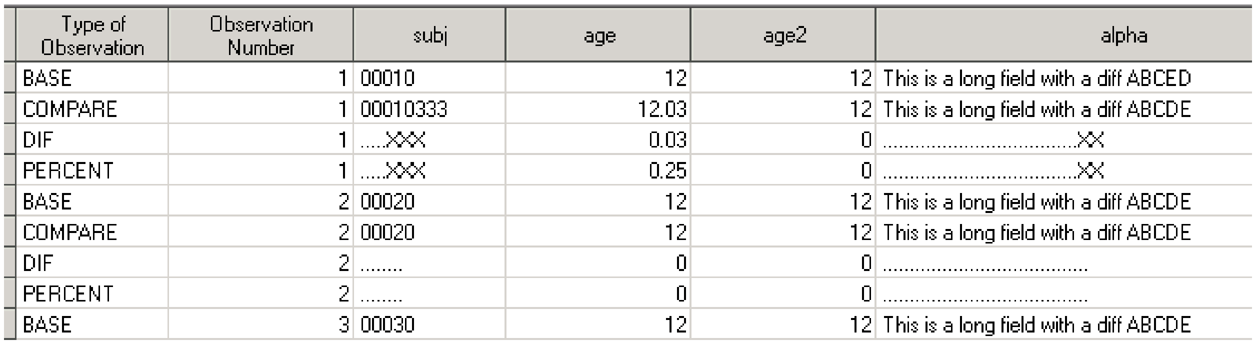
If you don't want to see output as well, also include NOPRINT.

The resulting dataset will have a "DIF" row for each corresponding observation. Character variables that are identical will display periods. The number of periods will be the number of characters in the data. Character variables that differ will display periods for characters that match and X for characters that differ. If you are working interactively, you can set the font of the variable to a fixed width font such as courier. This will line the X up the differences. If you have numeric variables, the DIF row will either show 0 for matching variables or the difference for non-matching.



	Type of Observation	Observation Number	subj	age	age2	alpha
1	DIF	1	XXX	0.03	0	XX
2	DIF	2	....	0	0	....

However, many times you will want more information. The OUTBASE option will write out a record for each observation in the base dataset and OUTCOMP will write out a record for each observation in the comparison dataset. There is also an OUTPERCENT option that will write one record per corresponding observation with the percentage difference for the numeric variables. The latter option could be beneficial when you are checking the impact of code or data change. Catch is, when you start specifying what records to write out, DIF is no longer written to the dataset unless you specify OUTDIF. By the way, spelling it with 2 F's also works. To simplify code, if you want all four records, you just need OUTALL.



Type of Observation	Observation Number	subj	age	age2	alpha
BASE	1	00010	12	12	This is a long field with a diff ABCED
COMPARE	1	00010333	12.03	12	This is a long field with a diff ABCDE
DIF	1	XXX	0.03	0	XX
PERCENT	1	XXX	0.25	0	XX
BASE	2	00020	12	12	This is a long field with a diff ABCDE
COMPARE	2	00020	12	12	This is a long field with a diff ABCDE
DIF	2	....	0	0	....
PERCENT	2	....	0	0	....
BASE	3	00030	12	12	This is a long field with a diff ABCDE

Notice that you also get BASE or COMPARE records for any extra observations that did not have a corresponding record. This dataset could become very overwhelming. OUTNOEQUAL used in conjunction with the other options will eliminate all the records where every variable was an exact match such as observation #2.

You can also get summary stats for numeric variables by including the OUTSTATS= option and specifying a dataset name. This is the same as the ALLSTATS option provides it in the printed output.

DATASETS WITH INTENTIONAL DIFFERENCES

PROC COMPARE works great for finding differences when the goal is to having identical datasets. However, what if you are comparing a dataset back to the previous version before you changed code or received new data? First, make sure you keep a copy of the old dataset before you overwrite it. You will need to determine what variables/observations you expect to change and what really should not change. For example, if the records will still be in the same order and just a few variables will change, you can probably don't need more than the basic proc and including MAXPRINT so it shows all records that changed.

It gets more complicated if the records are now in a different order. One of the things you might want to consider is reading in your old dataset and adding temp variables for the purpose of sorting. Once you dataset is sorted, drop these temp variables as a dataset option when running the proc.

```
proc compare data=old_data(drop=temp:) comp=new_data listall;
run;
```

Comparisons become more complicated if you now have more records than you did before. The SQL code used earlier can benefit here. You can use that dataset to identify “old” and “new” records. You then merge it to your new dataset to enable you to split your new dataset into two datasets with one containing the “old” records that would correspond to the original dataset and one with the “new” records. You can then compare the “old” records to see what changed. The new records would need to be reviewed to see if it made sense that those records were added. The code might look something like this.

```
data new_data2_oldrecs
      new_data2_newrecs;
  merge new_data
              diffs;
  by subject labtest;
  if old_count=new_count then output new_data2_oldrecs;
else if old_count ne new_count then output new_data2_newrecs;
run;
proc compare data=old_data comp=new_data_oldrecs;
run;
```

Unfortunately this method won’t work in every case. At times, you may have no choice except to look through your dataset and review it by hand to ensure the code didn’t change anything you didn’t expect or that the new data didn’t contain anything you code didn’t expect.

CONCLUSION

PROC COMPARE has many options to help you identify differences in datasets. These may be differences that you don’t want so you will need to modify your code or differences that you are confirming were changed correctly by updated code or new raw data. By combining the use of PROC COMPARE with other code and tools such as PROC SQL, the data step or working with a dataset interactively, you have a very flexible, efficient tool to meet most situation.

CONTACT INFORMATION

Deb Cassidy
deborah.cassidy@ppdi.com