

Are You Sure That Is Correct?: An Overview Of Good Practices For Dataset And Output Validation

Gregory Weller, Rho[®]; Alex Buck, Rho

ABSTRACT

In the world of statistical programming, it is imperative to ensure that the source data are accurately represented in the datasets, tables, listings, and figures submitted to regulatory agencies and academic journals. The gold-standard for ensuring correctness is double-independent programming where a production programmer and validation programmer aim to produce the same output. SAS[®] PROC COMPARE is the most commonly-used tool to compare the outputs and determine differences. While double-independent programming using PROC COMPARE can be a very useful process, it is important to recognize its limitations. In certain cases, this may not be the most efficient or complete method of validation required. Further, failure to establish and follow sound procedures and practices for validation can lead to disaster.

In this article, we will discuss good practices for double-independent programming, the proper use of PROC COMPARE, common pitfalls, and explore the different techniques needed for validating datasets, tables, listings, and figures. While there will never be one true and final answer for validation best practice, it is the hope of the authors to provide a starting point for discussion.

INTRODUCTION

The discussion of validation best practices is an ongoing one. There are many possible methods and many more possible implementations of each method. It is the hope of the authors that the article provides a starting point for discussion between users and helps avoid common pitfalls of validation programming. This article includes introductions to methods of validations and PROC COMPARE, overviews of good practices in validation and PROC COMPARE, as well as validation recommendations for datasets, listings, tables, and figures. A general knowledge of PROC COMPARE is not required, but may be helpful in reviewing the good practices and cautions. During this paper the authors will refer to a “production programmer” as the programmer that creates the final output to be used or submitted and a “validation programmer” as the second programmer tasked to confirm that the output is correct.

METHODS OF VALIDATION

The general flow of validation should be as follows:

1. Production programmer creates output based on specifications.
2. Validator creates a second set of output.
3. Validator compares the production output to the validation output.
4. Validator and production programmer discuss and resolve differences.

The gold standard for validation is double-independent programming. In the context of statistical programming, double-independent programming is the process of two or more independent people starting with the same information (specifications and datasets) and trying to create an identical output. This output can be another dataset, a listing, a table, or a figure. The production and validation versions are compared and any differences are investigated and fixed. This is an iterative process and ends when both programmers end up with the same product and agree that it is correct. A caveat to double-independent programming is that while it minimizes the risk, it is still possible to have an incorrect result. This may occur when both programmers make the same mistake, leading to the validation of the same incorrect output. It is important to note that while the programmers are expected to work together to discuss differences and investigate possible causes, no code is to be exchanged and the programmers are not to look at code used by the other side.

The comparison method between the two outputs can vary. One method is to use PROC COMPARE. This will ensure that the output values and formatting are consistent between the two outputs produced. In the case of displays the final SAS dataset used for the display is output and compared. Another method is a visual confirmation by the validator when comparing the two outputs. In this case the validation output could be in a similar structure and format to the production output or could be raw output from PROCs which require more attention to the differences caused by formatting.

A validation method commonly used for listings is a visual comparison between the input dataset and the listing output by an independent reviewer. A full visual comparison can be done for smaller listings, but larger listings would prove time consuming. The risk of skimming quickly over values and missing issues increases as the reviewer goes further down the listing. A spot check involves comparing randomly selected observations in the listing to the dataset.

This allows for thorough review of these observations, but does not constitute a complete review. We would never recommend this method when validating tables.

GENERAL INTRODUCTION TO PROC COMPARE

As explained in the previous section, the end result of the initial programming effort of double-independent programming is two versions of the same intended output. However that is only the first part of the process. Once the production and validation version exist they still have to be compared. Manually checking every value is possible but usually impractical given the large amount of data typically contained in an output. To solve this problem SAS created a method of electronically comparing two datasets called PROC COMPARE.

PROC COMPARE is a very powerful tool that will check two datasets observation by observation and report any differences it detects. In addition to checking the actual data values PROC COMPARE also checks variable attributes such as lengths, formats, informat, and labels.

Below is an example of PROC COMPARE comparing two WORK datasets called DATASET1 and DATASET2. The BASE and COMPARE arguments are the minimum amount of information a programmer must specify. Other features of PROC COMPARE will be discussed later in this paper. When comparing datasets it does not matter which one the programmer specifies as BASE and which one is they specify as COMPARE. PROC COMPARE would give the same results if DATASET2 were defined as BASE and DATASET1 were defined as COMPARE.

```
proc compare base=dataset1 compare=dataset2;
run;
```

GOOD PRACTICES AND CAUTIONS FOR PROC COMPARE

As with most PROCs in SAS PROC COMPARE has many options and features that can greatly enhance its functionality. However, use of these features can lead to major problems in the validation process. This section will cover some of the most commonly used features of PROC COMPARE plus the cautions a programmer must exercise in using them.

We recommend using the LISTALL option when using PROC COMPARE. This will give you the most complete information about what is mismatching in both datasets, and we do not see any advantage in using other options which restrict the amount of information that is displayed.

The VAR statement will limit PROC COMPARE to checking a specified subset of variables. There are some circumstances in which checking only a subset of variables is necessary, such as validating table columns, and utilizing the VAR statement is the best way to accomplish this. In circumstances where all variables must be compared the VAR statement should only be used when investigating differences and then removed as soon as possible. Leaving a VAR statement in can lead to a validator erroneously thinking a dataset is validated when there are in fact differences in other variables that were not checked.

The example below shows a PROC COMPARE of DATASET1 and DATASET2 but only checking the variables VARIABLE1 and VARIABLE2.

```
proc compare base=dataset1 compare=dataset2;
  var variable1 variable2;
run;
```

Another helpful feature found in PROC COMPARE is the ID statement. This allows a programmer to define a set of key variables to match the datasets on and can be very helpful in zeroing-in on problem observations. These key variables are usually defined in the output specifications as the final sort order and must uniquely define an observation. If the defined ID variables do not uniquely identify an observation PROC COMPARE will output a warning. Occasionally this warning will indicate a true duplicate observation in a dataset, but more often it will signal to the validator that they need to update the list of key variables.

It is important to note that PROC COMPARE will not directly check the values of the ID variables. Any mismatches in ID variables will show up in a compare as existing in one dataset but not the other.

Consider the following example dataset containing vital signs results. USUBJID is a subject identifier, TEST is the description of the test, DATECOL is the numeric date the vital sign was measured (with a DATE9. format), and RESULT is the numeric result of the vital sign measurement. The key variables which uniquely define an observation

	usubjid	test	datecol	result
1	ABC-1-1	Pulse	05MAR2016	70
2	ABC-1-1	Pulse	09MAR2016	75
3	ABC-1-1	Systolic Blood Pressure	09MAR2016	130
4	ABC-1-2	Pulse	15AUG2016	67
5	ABC-1-2	Systolic Blood Pressure	30AUG2016	122
6	ABC-1-2	Pulse	31AUG2016	71
7	ABC-1-3	Diastolic Blood Pressure	01JUL2016	85
8	ABC-1-3	Systolic Blood Pressure	01JUL2016	124
9	ABC-1-3	Pulse	30AUG2016	73

Display 1. Example Vital Signs Dataset

We will also create another version of the dataset that is identical except for the last observation having a result of 74 instead of 73. PROC COMPARE will be used to compare the two datasets to each other defining the dataset above as BASE and the new dataset (with the RESULT value of 74) as COMPARE. With no ID statement defined the differences will show up in the PROC COMPARE as follows:

Value Comparison Results for Variables

Value Comparison Results for Variables					
Obs		Test Result			
		Base	Compare	Diff.	% Diff
		result	result		
9		73.0000	74.0000	1.0000	1.3699

Note: The 'Obs' column to the left indicates the difference in RESULT is in the 9th observation of the dataset.

Display 2. PROC COMPARE Results from Vital Signs Dataset

When dealing with derived numeric variables value differences in far-out decimal places is quite common. Even though these differences are usually negligible PROC COMPARE will still flag them as a difference. Using the CRITERION option will define the maximum number of decimal places for SAS to check for a difference. If a difference is detected in a value past the cutoff defined by CRITERION it will be counted as "not exactly equal" but will not show up as a difference. CRITERION should be used cautiously, and only when the need for fuzzy validation is identified and confirmed by all project team members.

For the following example the result of the Systolic Blood Pressure observation for subject ABC-1-2 is changed from 122 to 122.0001 in the dataset VERIFY.BASECRIT. A CRITERION of .001 is specified so any differences less than .001 will be considered equal.

```
proc compare base=derive.base compare=verify.basecrit criterion=.001 listall;
run;
```

In the output below all 9 observations are considered to have all variables equal. However the "Values Comparison Summary" notes the value of 122.0001 in VERIFY.BASECRIT is "...not EXACTLY Equal..." to the value of 122 in the DERIVE.BASE dataset. The "not EXACTLY equal" row only counts the number of instances where two values are within the difference specified by CRITERION. It does not identify the affected observations.

```

      Observation Summary

      Observation      Base  Compare

      First Obs          1      1
      Last  Obs          9      9

Number of Observations in Common: 9.
Total Number of Observations Read from DERIVE.BASE: 9.
Total Number of Observations Read from VERIFY.BASECRIT: 9.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 9.

      Values Comparison Summary

Number of Variables Compared with All Observations Equal: 4.
Number of Variables Compared with Some Observations Unequal: 0.
Total Number of Values which Compare Unequal: 0.
Total Number of Values not EXACTLY Equal: 1.
Maximum Difference Criterion Value: 8.1967E-07.

```

Display 3. PROC COMPARE Results Using CRITERION

When PROC COMPARE detects differences the output will only display the first 20 characters of any variable. If this does not provide sufficient information programmers will need to output a dataset of the differences from PROC COMPARE.

GENERAL BEST PRACTICES FOR PROGRAMMING AND VALIDATION

We consider the validation process to be collaborative process. The production programmer should be available for discussion of issues and should feel confident to review the PROC COMPARE output and determine possible causes. Traditionally, the PROC COMPARE is only included in the validation version of a program. This leads to a tendency for the validation programmer to be the only one investigating differences and in essence being forced to prove that they are correct. We recommend including PROC COMPARE on both sides of the programming while going through the validation process. If both programmers are active participants in investigating differences the process tends to go quicker and smoother than if only the validator was looking into the differences.

Production programmers should open their output when created and confirm that it appears correct before informing the validator that the display is ready. When the output is a display it is good practice to review the titles and footnotes to confirm the analysis or summary within the display is consistent with any description in titles and footnotes. This can save rework during the final review.

To create traceability and transparency, the production and validation versions of a program should both output a permanent version of their final output dataset. This hold for tables and figures as there is always a final dataset used to create the display. An added benefit to this is that both programmers can investigate differences simultaneously. If a complicated model or analysis is leading to differences in the final output we recommend outputting the analysis input dataset on both sides and comparing. We have found that a simple difference in the input dataset is a common cause of output differences.

In the case of tables and figures it is the recommendation of the authors that every display be a PROC away from output. The display program should include the analysis and any formatting/structure changes for the display, but data manipulation should only be included in the dataset program. This will simplify the display validation. If a subset is needed we recommend that the flag be available in the dataset. We realize that this is not always possible (i.e. adverse event incidence $\geq 15\%$ tables), but in most cases the display program should not be used to create new variables for subset.

Table validation should include validation of display values as well as formats and structure. If the production dataset includes blank rows for the purposes of table structure then the validation program should also include those rows. Deletion of blank rows when pulling in the production dataset for comparison may cause unforeseen issues. The authors have found it beneficial to include formatting of values on both the production and validation side. Differences in the method of formatting values, such as using PUT versus PUTN, can result in a difference of final values. The validator should not assume that the raw values would match the formatted values in the production table.

The logs should be reviewed on both the production and validation side. An error can occur on both sides leading to matching values that are not accurate. Errors and warnings in the log may also inform the programmer if an analysis may not be appropriate for the data.

Occasionally, a programmer will run into a situation where they are having difficulty getting a matching PROC COMPARE output while sure that one version of the output is correct. In these situations every effort should be made to determine the cause of the differences and any possible fixes. If no more progress can be made the output may need to be left without a matching validation output. This should only be used as a last resort after thorough investigation and agreement by the project team that one version of the output is correct.

DATASET VALIDATION

Datasets should be validated through double-independent programming using PROC COMPARE. The size of most datasets alone makes any non-programmatic validation highly impractical.

After the validation programmer outputs their version of the dataset to a permanent location, the PROC COMPARE in the validation program should only read in the permanent production and validation datasets directly from their home folder. It is common to see validation programs where a programmer will read in the production dataset to the WORK library then perform programmatic manipulation to it (such as using a PROC SORT) before feeding it into PROC COMPARE. When datasets are programmed from a set of specifications there should be no reason for a validator to have to do any manipulation to the production version to make it match. If a validator must make adjustments to the production dataset in order to get it to match that is a clear sign that the output is not programmed as per the specs. A simple way to summarize this is that the word "WORK" should not appear in front of the dataset name in the PROC COMPARE output. The example below shows the datasets DATASET1 and DATASET2 being pulled in directly from the libraries DERIVE and VERIFY, respectively.

```
proc compare base=derive.dataset1 compare=verify.dataset2;
run;
```

In addition to the values within a dataset matching, it is important to make sure all variable attributes match as well. This includes variable labels, formats, informats, and lengths. To ensure variables have the proper formats and informats we recommend a two-step approach. The first step is to use a data step to remove all formats and informats from variables. Formats and informats should then be applied in a subsequent data step. Variable labels and lengths should also be applied at this last data step, though to avoid log warnings a programmer should keep variable lengths consistent at all data steps throughout their program.

The example below shows how PROC COMPARE identifies mismatches in variable attributes and labels in the "Variables Summary" section. DERIVE.BASE is the same vital signs dataset from before. The VERIFY.V_BASEFMT dataset has identical values to DERIVE.BASE, but the DATE9. format is removed from the variable DATECOL and the label of RESULT is "Incorrect Label".

```

Variables Summary

Number of Variables in Common: 4.
Number of Variables with Differing Attributes: 2.

Listing of Common Variables with Differing Attributes

Variable  Dataset              Type  Length  Format  Label
-----
datecol   DERIVE.BASE             Num    8      DATE9.  Date of Collection
          VERIFY.V_BASEFMT     Num    8              Date of Collection
result    DERIVE.BASE             Num    8              Test Result
          VERIFY.V_BASEFMT     Num    8              Incorrect Label
```

Display 4. PROC COMPARE Identifying Attribute Differences

WHERE and VAR statements should be avoided entirely when doing dataset validation since utilizing them can cause a programmer to accidentally compare only a subset of the data instead of the entire dataset. The ID statement is very helpful for investigating differences but should be removed after all differences have been resolved. The CRITERION option can be utilized if necessary but it must be used with caution. If rounding numeric values to a reasonable decimal place is not an option the project team must agree on an acceptable precision to check values to.

LISTING VALIDATION

A listing is a presentation of data without summary or analysis. We do not recommend double-independent programming for listings as an initial method of validation. Our best practice validation for listings is a visual comparison of the listing output to the input dataset. If the listing is short enough to allow for a full comparison that would be the preferred method. However we acknowledge that this is not always possible. In the case of a large listing output we would recommend a spot check method where listing observations are randomly chosen for a thorough comparison to the dataset observations. This should not include so many observations that the comparison suffers, but should include enough observations that the reviewer feels confident that listing is sufficiently validated.

The assumption that leads to using visual comparison over programmatic validation is that the listing is a simple output of the dataset. There should be no derivations or multiple datasets within a listing. If derivations are needed it is our recommendation that the dataset be updated with a new variable and that the listing uses that variable. If it is the case that multiple datasets are needed for a listing our recommendation is that double-independent programming be used for that listing. The comparison recommendations for double-independent programming would follow that of the table validation.

TABLE VALIDATION

A table is an output that may include a summary of the data or output from analyses of the data. Examples of tables include frequency displays, descriptive statistics (n, mean, standard deviation, median), and estimates and p-values from statistical analyses (logistic regression, repeated measures, etc.). Table validation should be done using double-independent programming. PROC COMPARE should be used to compare the dataset used to produce the final production output to the final validation dataset. This will require the production and validation programmer to confer with regards to the format of the values as well as structure of the output. The descriptive column, usually the first column of a table, will not regularly require validation. It is the belief of the authors that validation of this column will take more time than is necessary and will not produce meaningful results in terms of confidence of validation. If the descriptive column includes endpoints that can change with each data cut (12 month response versus 24 month response) or includes a dynamic list (all genomic sequences meeting criteria) we would recommend validating this column.

The input dataset going into the display should be used for the comparison. Libnames would be used to create a permanent dataset in the production and validation folders for the tables. When reviewing PROC COMPARE the programmer should ensure that the number of variables is expected and that the number of observations is equal. We do expect that in the case of tables the number of variables may differ. We recommend using the VAR statement of PROC COMPARE to specify the variables/columns of interest for comparison with the caveat that the validator confirms that the variables listed include all columns of interest for the table. When using previous validation code to create a new program the VAR statement must be checked for updates. While PROC REPORT does include an option to create an output dataset it is the recommendation of the authors to use the input dataset. PROC REPORT can set order, formats, and structure of the output within the procedure. We recommend avoiding these options and making the input dataset as close to the final desired output as possible to ensure that validation can produce the same results.

In the example below the input dataset includes three columns of a frequency table. The column values are formatted into "XX (XX.X%)" and blank observations are included to create spaces between rows in the final output. It is the expectation that the validation dataset would follow similar structure and formatting as the production dataset. The call to PROC COMPARE could include the VAR statement to only compare COL2 and COL3 as those are the values of interest.

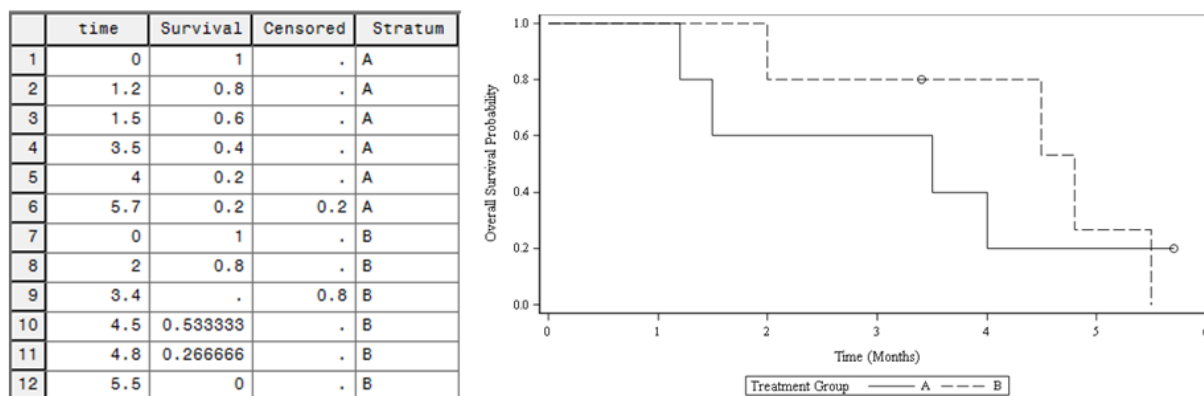
	col1	col2	col3		Treatment A (N=55)	Treatment B (N=45)
1	Gender			Parameter		
2	Female	30 (54.5%)	25 (55.6%)	Gender		
3	Male	25 (45.5%)	20 (44.4%)	Female	30 (54.5%)	25 (55.6%)
4				Male	25 (45.5%)	20 (44.4%)
5	Age			Age		
6	<65 years	50 (90.9%)	42 (93.3%)	<65 years	50 (90.9%)	42 (93.3%)
7	>=65 years	5 (9.1%)	3 (6.7%)	>=65 years	5 (9.1%)	3 (6.7%)

Display 5. Input Dataset and Final Table Output

FIGURE VALIDATION

Figure validation tends to be the most disputed validation item. One option is to compare the figure output to a validated table as most figures are created to be a graphical representation of information presented in tables. We do not find this to be sufficient as comparing graphics to numbers may be misleading. Another method is double-independent programming where the validator programs a second figure and visually compares the validation figure to the production figure. While this may catch most validation issues it leaves the possibility for 'close but not quite' issues to slip through. One example the authors have seen is a figure where the median $\pm$ the standard error was to be graphed. Production was deriving the standard error of the median whereas validation was using the standard error from PROC MEANS, or the standard error of the mean. While these values were close enough appear similar in the two graphs, the values themselves were different. One may also run into a resourcing issue where it becomes difficult to find two programmers that feel confident with creating figures. It has been our experiences that finding one programmer with confidence in creating figures and another programmer confident in creating a summary dataset is more reasonable.

Our recommendation is to have double-independent programming where the input dataset used for the figure is validated. Libnames would be used to create a permanent dataset in the production and validation folders for the figures. The validation program would then pull in the production dataset for comparison. In the case of a Kaplan-Meier plot the input dataset would include the survival estimate at each time point by group. In the case of a line graph with 95% CI it would include the mean, the lower limit, and the upper limit at each time point by group. The production and validation programmer should discuss the expected format, structure, and variable names of the final output dataset prior to starting programming. We do not find this extra step on the side of the production programmer to be burdensome and the confidence in the validation gained outweighs the possible extra time on the validation side.



Display 6. Input Dataset and Final Figure Output

CONCLUSION

There are many options for the method of validation and many more options for implementing the method. While the authors do not claim to have all the answers they have attempted to create a starting point for discussion and validation decisions. It is the sincere believe of the authors that using the best practices described throughout the paper will increase the quality of the output as well as the confidence in its accuracy by users.

REFERENCES

"The COMPARE Procedure" SAS® 9.2 Documentation.
<http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a000057814.htm>

"Usage Note 22174: PROC COMPARE output truncates character variables at 20 characters" SAS® Knowledge Base. <http://support.sas.com/kb/22174.html>

CONTACT INFORMATION

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

Name: Alex Buck
Enterprise: Rho
Address: 6330 Quadrangle Dr
City, State ZIP: Chapel Hill, NC 27517
E-mail: Alexandra_Buck@Rhoworld.com

Name: Gregory Weller
Enterprise: Rho
Address: 6330 Quadrangle Dr
City, State ZIP: Chapel Hill, NC 27517
E-mail: Greg_Weller@Rhoworld.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.