

When Good Looks Aren't Enough

Lisa Eckler, Lisa Eckler Consulting Inc., Toronto, ON

ABSTRACT

Have you ever looked over a SAS® dataset or report you've prepared and concluded that it "looks good" or worse still, "good enough"? One of the great advantages SAS programming – the ease with which you can manipulate data – is also a potential hazard. It's far too easy to create something that's close, but not exactly correct. Beyond instinct or some sort of sniff test, how do you know that your results are correct? How can you prove it to others? As SAS programmers or users, we should be striving for clear and accurate data, not just "good enough" data.

A SAS log showing no syntax or data errors or warnings and some reasonable numbers of observations is only a small step. This paper offers a common-sense approach to simple data validation, expanding on some of the most frequently used validation techniques. Some may consider these recommendations obvious. Experience has proven that they are obvious only if you follow them.

INTRODUCTION

Before tackling the challenge of checking the results of our own or someone else's SAS programming, we need to figure out what the scope of the exercise is. Depending on the situation, we may need to address

- ☐ Data Verification: Are the data accurate and consistent?
- ☐ Process Validation: Does our program or process produce accurate and complete results which meet the specified requirements?
- ☐ Or, some combination of the two which might be described as Quality Assurance (Does the overall process perform as it should and deliver the result it should?)

Whether we call it verification, validation, or quality assurance (QA), applying appropriate checks means the exercise will take less time and effort and is far more likely to identify any issues which need attention. After applying some human intelligence to defining the problem, much of the tedious work can be done in SAS, letting the machine do the work and freeing the analyst (or the programmer) to concentrate on the parts that require knowledge and judgment. Just as with programming activities, the key here is to identify the right questions in order to define the scope of the QA required. A mantra to approach this with is:

- ☐ Check your assumptions
- ☐ Confirm similarities
- ☐ Focus on differences

We tend to view QA as something that happens after the programming is done, and is often handled by a different person or team than the programming. This division of labour helps reinforce the examination of assumptions. Is QA actually a programming activity? Yes! At least much of it can be. First, the programmers should be exploring the result to ensure it meets their assumptions and expectations before passing it along. Next, the QA reviewer should be taking advantage of programs (ie, fully-tested, automated, repeatable processes) to confirm similarities and isolate differences. The same applies whether the QA is to be completed by the same person or team as the initial development or not. Why not let the machine do the work it can do best? Examples in this paper apply simple Base SAS® programming techniques to explore the new dataset and compare it to known things, testing assumptions and confirming similarities. Once we've used SAS to quickly identify any differences, concerns or unexpected results, we can concentrate most of our analytical time and energy on exploring or explaining the differences rather than on finding them.

BEFORE YOU GET STARTED

Checking of assumptions begins as a human task, first by being aware of what you expect or assume to be true. This awareness is something to consider -- or work on -- *before* you begin to look at the result set. While you may know something to be true in the universe as a whole, is it reasonable to expect it to apply in your subset of data? For example, while we may know that a typical gender ratio at birth means that 51% of babies born are male, does that mean we expect the male to female ratio in our universe of data to be 51:49? Doesn't it depend on knowing the specific population we are looking at? If the population under consideration is seniors aged 80 and above, then surely the anticipated ratio would not be the same as at birth. Beware of what you may think are common sense assumptions. There is no substitute for understanding the data you are working with. It may require someone with more applied knowledge than the programmer to identify what to expect.

Also before looking at the result set, consider what the required result is. In order to plan the review to be most effective, you must have a clear and detailed understanding of the expected result, down to the structure and format of the dataset. This may come from program specifications or knowledge of *why* there is to be a new dataset. Should it be the same as -- or are at least some parts the same as -- something which has already been through a substantial review process and is considered to be reliable? If the process under review, for example, is an optimization of an existing process, then we would expect the resulting data to be identical. If it's a parallel run from an application conversion then the obvious base for comparison is the end product of the old system, and the expectation would be an identical dataset or a dataset with predefined differences.

Some of the key questions

- ☐ Is this dataset supposed to be identical to something we've seen before?
- ☐ If not identical to something existing, what is the dataset supposed to contain?
 - Is this similar to something we have seen before because it includes something we had before with different or additional data (either rows or columns)?

- Are there expected differences reflecting corrections to some data or just different or additional rows representing a different population or time period? Is it reasonable to compare the new with the old to look for patterns?
- If there are new columns or all the rows are new, can we check these against some other source for reasonableness?

Answers to these questions will highlight what the expectations and assumptions are and help to distinguish what *should* be similar and what *should* be different from existing datasets. The steps above can be done before touching the new dataset and even before the new dataset is ready. Preparing the dataset and preparing to validate it can be done concurrently. Especially if different people or teams are doing development and QA, this can reduce elapsed time to delivery significantly.

CONSIDERING THE SITUATION

Knowing *why* we have a new dataset will determine which of the following questions are relevant:

- ☐ Does the dataset contain all the columns it should?
- ☐ Does the dataset contain all the rows it should?
- ☐ Is there some reality check on the reasonableness of the data?
- ☐ If the product is a customized or heavily formatted report, does it accurately reflect the underlying data?

If the dataset results from a regular daily, weekly or monthly process *and* the process hasn't changed, we may bypass checking the structure and do only the most basic reality check. If, however, this is a new or newly-modified process, some or all of the other checks may be necessary.

Once there is a clear understanding of what the *relevant* assumptions are for this particular data, confirm that those assumptions are met. Some simple, automated summary statistics and cross-tabulations can go a long way.

Confirmation of similarities can be primarily a 'machine' task, where a simple program can churn through the data and distill some simple facts. There is no point in spending a lot of time and human energy on this step. Just get it out of the way before focusing on the more important piece. Any differences between what is known or expected and what is received deserve the majority of time and attention. Analyzing the differences will be a machine-assisted, human task.

In real work life, our QA requirements may not be nearly as neat as above. One release may involve a combination of optimizing an existing process, delivering more or different rows and columns, and correcting errors in previous data all at once. The simple steps proposed in this paper can easily be combined to address more complex requirements, using the same approach: Check your assumptions and expectations as to what should be new and what should match existing, confirm

that the expected similarities are present, then focus your attention on the integrity of the columns or values that should be different.

Consider some general scenarios where QA may be needed. Some simple steps of Base SAS code are included for tackling each case. Although it does take some time to build each of these testing steps, they will form a utility toolkit to make subsequent QA activities faster, easier, more accurate and easily repeatable. To reinforce the idea that these steps form a reusable toolkit, macro variables are used to reference new and existing datasets. (Figure 1) There may be occasions when the dataset name will be identical and only the library name will differ, and other occasions when the old and new datasets may be stored in the same directory. Either way, only the macro values need to be changed.

Figure 1

```
** define macro variables **;  
  
%let NEW_LIB = C:\LISA\QA_LIBRARY;  
%let NEW_DATASET = QA_TEST;  
%let OLD_LIB = C:\LISA\PRODUCTION_LIB;  
%let OLD_DATASET = ORIG_PROD;  
  
** using macro variables means that only code above this line **;  
** would need to be touched when comparing other datasets **;  
  
libname NEW_LIB "&NEW_LIB.";  
libname OLD_LIB "&OLD_LIB.";  
  
proc contents  
    data = NEW_LIB.&NEW_DATASET.;  
run;  
  
proc contents  
    data = OLD_LIB.&OLD_DATASET.;  
run;
```

Case I: New dataset is expected to be identical to an existing dataset.

The first – and possibly only – thing required would be to run PROC COMPARE. (See Figure 2) If the results are exactly the same, no further review is necessary. A listing of contents, using PROC CONTENTS or DICTIONARY TABLES, and a dump of some sample records might be appropriate as an enhancement to program documentation. These are, however, beyond essential QA if the new dataset is supposed to be identical to an existing one which has already been fully validated *and* it is has proven to be identical.

If both the dataset for comparison and NEW_DATASET *are* identical in number of rows and columns and values in each *and in the same sequence*, PROC COMPARE will provide a quick and concise confirmation that they are identical. It may be necessary to sort the datasets so they are in the same

sequence. If there are differences in values, specifying which key column to compare on is essential to making the output meaningful.

Figure 2

```

** This and subsequent coding examples assume that the      **;  

** statements from Figure 1, above, up to and including the  **;  

** LIBNAMES, have executed.                                  **;  
  

proc compare  

    data = NEW_LIB.&NEW_DATASET.  

    compare = OLD_LIB.&OLD_DATASET.;  

run;
```

Case II: New dataset is expected to be completely new to us, based on new data and process.

- If we are working with an unfamiliar result set, we must rely heavily on program specifications and knowledge and understanding of the requirements. Ideally, column specifications such as variable names, formats and labels would have been provided in an electronic format such as an Excel spreadsheet. If so, those specifications can be imported and treated as the assumed contents of a model dataset which can be compared to the contents of NEW_DATASET. The model dataset could replace what's in CONTENTS_OLD in Figure 9, below. That would allow us to break up the complex Case II QA exercise and deal with the structural part of it as an easily-automated Case III situation.
- A formal code review done with the development and QA person(s) or team(s) can be tedious but may help to clarify expectations and contribute to an improved common understanding of whether the program satisfies the requirements.
- Review data for reasonableness. This *requires* knowledge of relevant data and what is expected in this dataset. Are there unreasonable values like supposed actual dates which haven't occurred or negative ages? Figures 3 and 4 show how to find clues as to which variables warrant further analysis. Either of these may generate a high volume of output, so they are best reserved for occasions when you don't know where else to begin. After isolating relevant variables, some cross-tabulations can help to identify more meaningful patterns or look for unexpected values or what should be mutually-exclusive combinations. Applying standard or user-defined formats with PROC MEANS, PROC FREQ or PROC SUMMARY to categorize data is a very quick way to highlight the existence of unexpected data which require further examination. (See Figure 5)

Figure 3

```

** PROC MEANS with no CLASS statement is a fair starting point **;
** if you aren't familiar with the data. It will determine **;
** the number of non-missing values, the mean, standard **;
** deviation, minimum and maximum of all numeric variables **;
** as the default. **;

proc means
  data = NEW_LIB.&NEW_DATASET.;
run;

```

Figure 4

```

** PROC FREQ with the _ALL_ option for tables will provide **;
** frequency counts for values of all variables in the dataset. **;

proc freq
  data = NEW_LIB.&NEW_DATASET.;
  tables _all_ / list missing;
run;

```

Note that the code in Figure 4 might produce a very large listing if any of the numeric values taken on continuous values, or even a large range of discrete values. Using a user-defined FORMAT to group expected numeric values can make the results of PROC FREQ easier to read.

Figure 5

```

** PROC FORMAT is used to create a user-defined format. This **;
** one will map acceptable values for gender (M,F,m,f) so that **;
** missing or unacceptable values stand out in the summary. **;

proc format;
  value $genfmt
    'm','M' = 'Male      '
    'f','F' = 'Female    '
    other   = 'Unexpected'
  ;
run;

proc freq
  data = NEW_LIB.&NEW_DATASET.;
  tables gender / list missing;
  format gender $genfmt.;
run;

```

The converse of applying a standard or user-defined format to categorize data is looking at underlying *unformatted* data in a dataset. The example below shows how to remove stored formats from all (`_ALL_`) variables in the dataset. Note that the dataset is *copied* from its live location into a work directory before removing formats from the *copy*. Changing attributes does modify the stored data, so a copy is worked on to *avoid altering the live dataset*.

Figure 6

```
proc datasets
    library=work;

    ** Take a copy of NEW_DATASET from its regular location in      **;
    ** NEW_LIB into WORK to avoid damaging the real one.          **;

    copy in=NEW_LIB out=work;
    select &NEW_DATASET.;

    ** Remove stored formats and other attributes from ALL        **;
    ** variables in the copy of NEW_DATASET.                      **;

    modify &NEW_DATASET.;
    attrib _all_ format=;
quit;
```

Although these code steps can help uncover concerns, the reasonableness and relevance determination relies on human insight. Full automation of this activity is impossible.

- Considering a sample of rows from a new dataset and dumping those for visual review or validation of derived variables is a common approach. Careful selection of the sample is less common. Only if rows in the dataset are already in *random* order – not in sorted sequence or in careless order – does it make sense to dump the first or last n rows. There are many possible ways around this:
 - If the data are sorted, we may select every n^{th} record in a data step.
 - We may use a SAS function to assign a random value which is used as a sort key and then select the first n records.
 - If the data include categorizations or treatments of records based on values derived by program logic, however, it is best to select representative records for review. The minimum number of rows that need to be looked is not predetermined, but depends on how many decision variables are used in the program and how many possible values each decision variable takes. Figure 7 shows one way to capture one representative row for each branch in a decision tree for which data exist in the result set. This intelligent sampling reduces effort required to validate derivations in program logic.

Figure 7

```

** This assumes that NEW_DATASET is sorted by whatever      **;
** decision variables are used to derive categorical variables **;
** or treatments. For example, say that is YEAR, REGION and  **;
** GENDER.                                                    **;

** Select one representative record per combination of       **;
** decision variables which exists in dataset. The resulting **;
** SAMPLE will have the same number of records as there would **;
** be rows in the output of a PROC FREQ on a cross-tabulation **;
** of those variables.                                         **;

data SAMPLE;
    set NEW_LIB.&NEW_DATASET.;
        by year region gender;
    if first.gender;
run;

```

Case II forms the most complex and time-consuming QA situation. Fortunately, it is also the least common, since more often our new dataset is based on -- or related to -- something we are already familiar with.

Case III: New dataset is expected to be similar to an existing dataset, plus or minus some columns.

This can be broken down into sub-problems:

- Review the name, label and format of columns using PROC CONTENTS. (Basic code is shown in Figure 1, above.) Default sequence is alphabetical by variable name. Confirm the absence of any columns that should not be included in NEW_DATASET and the presence of any additional columns that should be included in NEW_DATASET.
- Compare those columns which are expected to be common to OLD_DATASET and NEW_DATASET using PROC COMPARE to ensure they are present and identical in value. This is similar to Case I, but with specific variables listed. (Figure 8)
- Consider the integrity of the data in any additional columns (using Case II methodology, above).

Figure 8

```

** If we expect that OLD_DATASET contains variables a, b, c    **;
** OLD_DATASET contains variables b, c, d, COMPARE b and c     **;
** from each to confirm they are identical.                    **;

proc compare
    data = NEW_LIB.&NEW_DATASET. (keep = b c)
    compare = OLD_LIB.&OLD_DATASET. (keep = b c);
run;

```

The volume and nature of the data will influence how much of the analysis needs to be automated. There's a big difference between comparing a dataset containing 5 columns with one containing 6

columns and comparing a dataset containing 150 columns with one containing 160 columns. It may not seem time-saving to write code to compare small (narrow) datasets, but if written with macro variables, that code can easily be reused to handle large or small datasets.

For larger datasets, rather than listing the result of each PROC CONTENTS, a subset of each of the results can be routed to a dataset using the OUT= option, and then the result subsets are *combined* into a simple listing for visual comparison. These steps, in effect, turn structure information about the dataset – or *metadata* -- into data, allowing two dataset structures to be compared in an automated way. (Figure 9)

Figure 9

```
** Create a dataset containing the name, type and length of    **;  
** each variable in the OLD dataset.                          **;  
  
proc contents  
    data = OLD_LIB.&OLD_DATASET.  
    out = CONTENTS_OLD(keep=name type length);  
run;  
  
** Create a dataset containing the name, type and length of    **;  
** each variable in the NEW dataset.                          **;  
  
proc contents  
    data = NEW_LIB.&NEW_DATASET.  
    out = CONTENTS_NEW(keep=name type length);  
run;  
  
** Combine the lists of variables and their attributes from    **;  
** each of the two source datasets.                          **;  
  
data CONTENTS_BOTH;  
    merge CONTENTS_OLD(rename=(type = type1 length = length1))  
          CONTENTS_NEW(rename=(type = type2 length = length2));  
    by name;  
run;  
  
proc print data = CONTENTS_BOTH;  
    id name;  
    title1 'Display combined CONTENTS of each dataset';  
run;
```

Dictionary tables can be used instead of PROC CONTENTS to achieve a very similar effect. Because the format of results from Dictionary Tables is slightly different than that of PROC CONTENTS, it's best to stick with one approach to keep the results easily comparable.

For larger (wider) datasets, a program combining PROC CONTENTS and PROC COMPARE can be a huge help in confirming similarities and isolating differences in structure between two datasets. Figure 10 uses the code similar to Figure 2, above, as a follow-on to what's done in Figure 9. The effect is an automated comparison of the metadata from our OLD_DATASET and NEW_DATASET. This will confirm whether the column structure is identical in the two datasets or highlight those columns which differ.

Figure 10

```
** COMPARE the variables in the two datasets. Each of those **;  
** variables represents a column in OLD_DATASET or NEW_DATASET.**;  
  
proc compare  
    data = CONTENTS_NEW  
    compare = CONTENTS_OLD;  
    by name;  
run;
```

Case IV: New dataset is expected to be similar to an existing dataset, plus or minus some rows.

This can be broken down into sub-problems:

- Compare those rows which are expected to be common to OLD_DATASET and NEW_DATASET *and* identical, using PROC COMPARE on a subset of data. This sub-problem is like Case I. (see Figure 11) This comparison will confirm the continued integrity of what should not have changed.
- Confirm the presence of any additional rows that are expected and the absence of any rows that should be excluded. (Use PROC FREQ on key variables.)
- Consider whether any new rows in NEW_DATASET follow the expected pattern of data.
- Consider the integrity of the data in any additional rows in NEW_DATASET. (This sub-problem is like Case II.)

Figure 11

```
** For example, assume OLD_DATASET contained history for years **;  
** 2006 and 2007 and NEW_DATASET is supposed to contain **;  
** history for 2007 and 2008 only. Use the common year, 2007, **;  
** for comparison to ensure integrity of common rows. **;  
  
proc compare  
    data = NEW_LIB.&NEW_DATASET. (where=(year = 2007))  
    compare = OLD_LIB.&OLD_DATASET. (where=(year = 2007));  
run;
```

IF THE RESULT IS A REPORT

If the product you are reviewing is a report rather than a dataset, there are ways to apply very similar techniques, again depending on the scope of the exercise. If this is a totally new process then ideally you will have access to the underlying dataset(s) and be able to do your own manipulation, treating the data as described above. If the process under review is similar to some existing process, and you have access to the data behind both the new and the existing, then it may be sufficient to compare those two sets of data, again with emphasis on confirming the similarities and looking at the differences. The questions here are: (1) What are the expected differences? and (2) Are the expected

differences present? If you only have the report and no access to the underlying data, but have the report in soft copy, there is still some potential for time-saving. You may be able to IMPORT the data from the existing and new reports using SAS and then COMPARE the SAS datasets as described above. Unfortunately, though, formatting can easily skew the IMPORT, so this approach is less reliable with reports than with datasets. If the new report represents primarily formatting changes then visual inspection may be the most effective review.

CONCLUSION

Applying an intelligent approach to planning and performing data quality assurance can save time and improve effectiveness. Building up a toolkit of simple SAS steps can streamline the process and let us complete the most tedious parts programmatically, so that human resources can be focused to the part of the exercise that requires attention. If properly built, these program steps are easily reusable, so that subsequent QA reviews take very little additional time and expected results are easy to confirm.

ACKNOWLEDGEMENTS

Thanks to the many colleagues and clients I've worked with who have learned their lessons the long, hard way and been willing to share the benefits of their experience.

CONTACT INFORMATION

Your comments are welcomed. Happy programming!

Lisa Eckler

lisa.eckler@sympatico.ca

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 registered trademarks or trademarks of their respective companies.