Paper 7740-2016

How to STRIP Your Data: Five Go-To Steps to Assure Data Quality

Michael Santema and Fagen Xie, Kaiser Permanente

ABSTRACT

Managing large datasets comes with the task of providing a certain level of quality assurance, no matter what the data is used for. We present here the fundamental SAS® procedures to perform when determining the completeness of a dataset. Even though each dataset is unique and has its own variables that need more examination in detail, it is important to first examine the size, time, range, interactions, and purity (STRIP) of a dataset to determine its readiness for any use. This paper covers first steps you should always take, regardless of whether you're dealing with health, financial, demographic, environmental, or other data.

INTRODUCTION

Data volume across the globe has been increasing at a rapidly accelerating rate. Most organizations have drastically increased their storage capacity in the past few years. With this new wealth of data comes a new breadth of solutions for the 21st century. However, as the demand for big data has increased, handling vast quantities of data carefully becomes more of a challenge as time constraints for performing quality assurance remain tight.

How does one assure quality for a massive dataset, let alone series of massive datasets that are updated frequently? As the task of maintaining high-quality data is receiving more attention than ever before; there is no shortage of best practices, checklists, protocols, standards, etc. for how to manage large datasets. We distill many of these methods into 5 easy-to-remember steps: examine the Size, Time, Range, Interactions, and Purity (STRIP) of every multi-gigabyte table that lands on your server and you'll go from knowing very little about a dataset to being fairly confident in its quality, no matter what industry you work in.

Many of the tools useful to complete the STRIP method are built-in to every version of SAS®. These are already basic go-to procedures for SAS® programmers everywhere, and can be conducted with relatively little code-writing.

The very first step for checking a dataset produced by a program is, of course, to check the log for errors and warnings. However, even an error-free log doesn't necessarily mean the dataset is ready for use. Supposing you've either checked the log, or not received one because someone else made the datasets, these 5 steps will guide you to a better understanding the state of your datasets.

OVERVIEW

Following are the characteristics of any dataset you should examine for basic quality assurance. Just remember STRIP:

- Size
- Time
- Range
- Interactions
- Purity

SIZE

The first thing you will notice when you create or receive a dataset is the size, in bytes. Even if you're unfamiliar with the type of data contained in the file, you'll have a reasonably good idea if it's noticeably small (<100KB) or large (>100GB). The data size is an important piece of information when deciding what procedures and steps to use to efficiently manipulate the data, such as PROC SQL versus a merge statement in a data step. Whatever the case, a simple run of PROC CONTENTS is a free way to find out more about the metadata of the dataset:

```
proc contents
    data= work.dataset;
run;
```

Contents information is attached to every SAS dataset, so even if the data you're looking at is over a petabyte in size, it costs you no time. On top of that, as shown above, there's very little code required to check the contents, so typing it out costs you almost no time as well! Once you go beyond the actual size of the file in bytes, you can begin to examine the size of the data in fields. Check the number of variables. Is it what you expected? If not, which variables are missing, or which variables are added? Often superfluous variables are misspelled versions of other variables. Missing variables are either dropped or not correctly added in the creation of the dataset. Having the correct variables is an easy way to check your data quality before moving forward.

Look for unusual trends, are there very few observations for a huge dataset? This could be due to large formats being used. If you see a large format, consider shortening those. Formats can also be too short, or just plain wrong. Dates and times are difficult to work with if they're only in numeric form (i.e., a count of days from January 1 1960), or worse, character.

Also, when you look at the contents of your dataset, make sure that that your dataset and variables follow standard naming procedures you may have already developed. Dealing with renaming early can save you unnecessary headaches later on.

TIME

Virtually all data analysis involves a time component. Whether you're looking at environmental, financial, health, or political data, there is some record somewhere of how the data changes. That's why, in almost any given dataset, you will find either a date or a time variable related to when the data was recorded. If you're able to look at the distribution of your data over time, all sorts of valuable information is suddenly revealed.

PROC FREQ lends itself to producing quick distributions that are easy to read. Often, you can use the format step within PROC FREQ to adjust your time variable for the relevant cycles. Consider formatting your time variable so that the results table prints on 1-2 pages. Date (i.e., day) is useful if the data you're using are over a few weeks. However, if you have several years of data, the output of data-per-day is less useful, and you'll need to put it in a more user-friendly scale. For this example using format var YEAR4.; to show data-per-year is more appropriate:

```
proc freq
data= work.dataset;
  format date YEAR4.;
  table date / missing;
run;
```

Perhaps your data covers a few hours instead of days, and the only time variable available to you is time-of-day. Again, you should keep your frequency table short, but long enough to give some information about the data, usually >4 rows and <2 pages if possible.

Once you have a frequency of your data over time, look at your total time span. You may find that your time variable is compromised if there are dates after the current date, or if you find dates spanning back hundreds of years. Next, look for unusual spikes in your frequency counts. Often, a growing dataset will show a gradual increase in available data with time. Are there years, days, or hours that have much more or much less data than those before or after them? Make sure you can explain spikes or drops, they might be normal, but they might be due to duplicates or merging errors between different data sources.

RANGE

Sometimes, even if the data is of the proper size and distribution over time, you may discover that some of the values are simply wrong by some error of commission. Perhaps the data was collected improperly or there was some manipulation or calculation that happened earlier that left errors, or perhaps the data was incorrectly recorded in the beginning. Whatever the case, you can never be too careful when checking that the data you have falls within an expected range. For numeric variables, PROC MEANS is an invaluable tool. Using the standard options, in the results the range is given to you as minimum and maximum, as well as commonly used percentiles:

```
proc means
    data=work.dataset;
run;
```

For character variables, PROC MEANS will do you little good. PROC FREQ to the rescue once again. Looking at the frequency of your data, you should look for misspellings and things that are the same but have different capitalizations. If you're working with a truly massive dataset, you already know you'll be waiting for PROC FREQ for a long time. If you only have time to do a quick spot-check, consider a randomized sample using a DATA STEP:

```
data sample (drop=i);
    choice=int(ranuni(0)*n)+1;
    set work.dataset point=choice nobs=n;
    i+1;

    /* Enter the desired sample size, 20 in this case */
    if i>20 then stop;
run;
```

After checking the frequency distribution of your time variable in the last step, you may have noticed some missing variables. However, if time is not a necessary variable for your data, you should also check the missingness of other key variables. The fastest way to do this is to use PROC FREQ, with the / missing option activated.

For large and complex datasets, formatting is your friend. Consider creating a format with PROC FORMAT that lumps similar results together. For example, say you want to find how many times the word "aspirin" appears in your data, but the different options available to you are "aspirin 100mg", "aspirin 50mg" or "aspirin 10% HCl", etc. Creating a new format with the following options would solve your issue without having to change the data:

```
proc format;
   value $aspirin
   (like '%aspirin%')='Aspirin'
   other = 'Not Aspirin';
run;
```

You may have variables that are related, where unrealistic values are contingent upon other variables. For example, if you have demographic information such as whether someone is a smoker or not, a value of "yes" for your smoking variable might not be realistic if the person in question is 4 years old. It is worth taking some time to imagine some unlikely scenarios in your data to help you screen for errors in related variables. Here a cross table in PROC FREQ becomes useful:

```
proc freq
   data= dataset;
   table smoker*age / nopercent nocum nocol norow;
run;
```

INTERACTIONS

Many datasets have a key variable used for linking to other datasets. Identifying this variable may be obvious and by design, or it may be some variable that happens to occur in two datasets, like time. Whatever the case, in order to make use of a dataset, it almost always becomes important to note the relationship your data has with other data. This, of course, means you will need to check the data that you plan to link to your dataset. Additionally, you can assemble one or two datasets you might link this data to in the future. The more established and used by other people the data is, the better.

First, if you haven't already (and you really should have, when you checked the range above), do a quick PROC FREQ and see what percentage of your key linking variable is missing values. You won't be able to link on records that are missing your linking variable. Decide if missing values present a problem for the work you need to do, and if so, figure out why that variable is missing. If not, move forward.

Next, check the format of your key variable against the format in your outside dataset. If they're different, which one is wrong? Obviously don't try to change a character variable to a numeric if there are letters in it, but consider what, if anything will be lost if you change the format of this variable.

Finally, you'll need to check that your dataset is actually to its related datasets. It may seem like you're starting to use your data and not doing quality assurance anymore, but to be really sure everything matches up well, you'll need to do a join in PROC SQL.

```
proc sql;
  create table test_join as
    select a.key_variable as my_key,
           b.key_variable as their_key
    from dataset as a
    full outer join other_set as b
           on a.key_variable=b.key_variable
   where a.key_variable is not null
  ;
quit;
```

From there, you've got the simple task of seeing how complete of a join you've really got. Tacking on a flag in a DATA STEP and running a PROC FREQ usually works. Making your flag really small and using PROC FORMAT will keep your dataset from growing too large:

```
data test_join;
set test_join;
  if my_key is null
   then flag = 1;
  else if their_key is null
    then flag = 2;
  else flag = 3;
run;


proc format;
  value who_missing
    1 = "missing in mine"
    2 = "missing in theirs"
    3 = "Match!";
run;


proc freq
data= test_join;
   format flag who_missing.;
   table flag / missing;
run;
```

This is your moment of truth! How good was that join? Is the level of interaction what you expected? If not, add corresponding variable to the table statement of PROC FREQ above (e.g., flag*var), and see if there are patterns associated with those missing keys. Time variables (sound familiar?) are perfect because they're almost always there, and any missingness associated with a certain period of time is easy to recognize.

PURITY

Finally, after checking that the nuts-and-bolts of much of your data is ready for use, you'll need to check to make sure your data is pure. Check for the many different impurities that are prone to happen in different datasets: whether there are unusual duplicates, redundant variables, and whether the data is being stored efficiently. This step represents a final comb-through of your data to check for anything you may have missed on the first four major passes.

Often, there is little reason for there to be duplicate records. Duplicate records may be caused by the exclusion of a key variable that would have been the sole unique variable between records, or a joining error. Running PROC SORT with the noduprecs option easily takes care of this to ensure each record is unique:

```
proc sort noduprecs
data= work.dataset;
   by key_var;
run;
```

or a distinct count in PROC SQL:

```
proc sql;
   select count(distinct key_var) as Ndistinct,
   count(*) as Nobs
   from dataset;
quit;
```

Going beyond that, you may have a key variable that you also require to be unique. Perhaps, for example, your dataset is a registry of customers where each record is supposed to represent one person, and no person is in the dataset twice. Using the nodupkey option in proc sort, and sorting by your key variable (in this case, maybe a customer ID number) or variables (like first, middle, and last names) in the "by" statement.

Even with major advances in data storage, and terabytes being cheaper than ever before, many databases reach the limits of what an organization is able to store. In anticipation of potential storage issues, it's always important to make sure your data is stored efficiently. Extraneous megabytes compromise your data's purity, and using compression should always be standard practice. Make sure you compress with the statement:

```
options compress=yes;
```

This option is usually the most useful for compressing, but there is also compress=binary for situations when your data is largely numeric variables. Use compress=yes as standard practice, and if you have data that is mostly numeric variables and you have time, try both ways to see which way reduces your data's size the most. Even with mostly numeric data, using compress=binary may not be the best option because sometimes one character variable is all it takes to monopolize the size of a dataset.

CONCLUSION

It's crucial to perform careful quality assurance of any dataset you work with. However, since almost everyone has more data than ever before, the amount of time allotted for quality assurance may be limited. Remembering a simple 5-letter acronym like STRIP can be a useful tool for providing at least a basic level of quality assurance for every dataset. Remember to check the Size (unusual growth or reduction in bytes, missing or extra variables), Time (unusual gaps or spikes in data over time), Range (most values are in an expected range, missingness occurs at acceptable rate, number of unique values is expected), Interactions (linking variable is unique and links to target data), and Purity (no duplicates, data is generally stored efficiently). Using these 5 steps regularly will boost your colleagues' confidence in your data.

REFERENCES

Ferriola, F. and A. Long. 2015. "Standardizing the Standardization Process." *Proceedings of the SAS Global 2015 Conference*, Cary, NC: SAS Institute, Inc. Available at http://support.sas.com/resources/papers/proceedings15/3209-2015.pdf.

Martín, E. and G. Ballard. 2010. Data Management Best Practices and Standards for Biodiversity Data Applicable to Bird Monitoring Data. U.S. North American Bird Conservation Initiative Monitoring Subcommittee. Online at http://www.nabci-us.org/.

Winiwarter, W., J. Mangino, A. Ajavon, A, McCulloch, and M. Woodfield. 2006. "Chapter 6: Quality Assurance / Quality Control and Verification." *2006 IPCC Guidelines for National Greenhouse Gas Inventories*. Geneva: IPCC. Available at http://www.ipcc-nggip.iges.or.jp/public/2006gl/pdf/1_Volume1/V1_6_Ch6_QA_QC.pdf.

Mitchell, R. M. 2007. "Finding Your Mistakes Before They Find You: A Quality Approach For SAS® Programmers." *Proceedings of the SAS Global 2007 Conference*, Cary, NC: SAS Institute, Inc. Available at http://www2.sas.com/proceedings/forum2007/128-2007.pdf.

Purvis, T. and C. Pearson. 2015. "Creating a Data Quality Scorecard." *Proceedings of the SAS Global 2015 Conference*, Cary, NC: SAS Institute, Inc. Available at http://support.sas.com/resources/papers/proceedings15/3261-2015.pdf.

Reynolds, H. and D. Trees. 1988. "A Generalized Approach to Data Quality Assurance and Control of Large, Complex Data Structures Using SAS." *IASSIST Quarterly*, 12:25-52. Available at http://www.iassistdata.org/sites/default/files/iq/iqvol124trees.pdf.

Shiralkar, P. 2012. "Quality Assurance: Best Practices in Clinical SAS® Programming." *Proceedings of the 2012 NorthEast SAS Users Group Conference*, Cary, NC: SAS Institute, Inc. Available at http://www.lexjansen.com/nesug/nesug12/ma/ma05.pdf.

CONTACT INFORMATION

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

Michael Santema
Department of Research & Evaluation
Kaiser Permanente
626-564-7842
Michael.L.Santema@kp.org