PharmaSUG 2023 - Paper SD-183

Was the load okay?

Lisa Eckler, Lisa Eckler Consulting Inc.

ABSTRACT

After a series of database tables are loaded from multiple data sources and before using the data to feed automated reports and business intelligence tools, we want to know whether the load was complete and correct. This goes beyond confirming that the jobs ran without errors. The more precise concerns are:

- Were all of the data sources ingested?
- Were the right number of rows of data added or updated in each table?
- Were all of the appropriate columns populated?
- Do the data values make sense?
- Are the values of categorical variables different than expected?

This paper describes a fully automated SAS® process for comparing a new set of data loaded to one or more tables with previous sets to check for reasonableness and completeness and highlight potential problems. It can also be used more generally as a data comparison tool for two or more subsets of similar data.

INTRODUCTION

This paper will use the requirements for, and development and implementation of, a verification process for one particular use case and then explore how a similar process could be applied to others. Our challenge was to verify the loading of several related tables encompassing several hundred columns of data. The tables are loaded monthly but on different days. Some tables have rows added and partially populated one day and the other columns populated another day. Recognizing that the load jobs have run is easy. We receive an automated email confirmation when each database update job ends. Knowing that the jobs ended successfully is a necessary and important first step but it's not sufficient for knowing that the data are correct and complete. As data is gathered from more sources and the feeds have more dependencies, there is greater risk of an input being missed or an upstream change not being properly integrated. Working on the analytics and reporting side, we don't want to be bogged down by the details of every data source but we do rely on our data loads being correct and complete. So, we trust but we verify. Having an automated process to do the verification reinforces our ability to trust.

We anticipate slow monthly growth for many values in the data and seasonal variations for some others. It would actually be a concern if there was absolutely no change from month to month. Looking at the change in distribution of values from month to month is helpful for identifying concerns. Most of the time there's nothing interesting to see in the verification reports and that's good news.

WHAT CHANGES ARE SIGNIFICANT?

For an example, refer to Output 1 for a snapshot of one regional metric for 2 months that may be distributed in a report to the regions. From the trend and our knowledge of the data and circumstances, we expect variation of up to 15% per month under normal circumstances. Anything beyond 15% would warrant investigation. Output 2 shows the results for a third month. Based on that, Region B would be a concern because of a sharp drop in volume. Only by looking at Output 3 and considering the whole situation can we see that a new region, Region D, has opened up in month 3. Our subject matter expert would know that Region D was created to divide Region B into 2 because it had been so growing so rapidly in the past. Even before reports to the regions have been prepared, we can share news of the

continued success. If results for Region D didn't appear in our database then we could be proactive in investigating the problem and advising the regions about missing data.

Report for May

region	month	volume
Region A	2022-05-31	28

region	month	volume
Region B	2022-05-31	45

region	month	volume
Region C	2022-05-31	35
		108

Report for June

region	month	volume
Region A	2022-06-30	30

region	month	volume
Region B	2022-06-30	44

region	month	volume
Region C	2022-06-30	40
		114

Dump Expected Trend Based on May - June

region	month	volume_05	volume_06	absolute_difference	percent_difference
Region A	2022-06-30	28	30	2	7.14

region	month	volume_05	volume_06	absolute_difference	percent_difference
Region B	2022-06-30	45	44	1	2.22

region	month	volume_05	volume_06	absolute_difference	percent_difference
Region C	2022-06-30	35	40	5	14.29

Output 1: Data for two months with month over month trend

Report for July

region	month	volume
Region A	2022-07-31	32

region	month	volume
Region B	2022-07-31	25

region	month	volume
Region C	2022-07-31	45

region	month	volume
Region D	2022-07-31	35
		137

Output 2: Data for third month

Anaylsis for May - July with Flagging

region	month	volume_05	volume_06	volume_07	absolute_difference	percent_difference	flag
Region A	2022-07-31	28	30	32	2	6.67	
Region B	2022-07-31	45	44	25	19	43.18	*
Region C	2022-07-31	35	40	45	5	12.50	
Region D	2022-07-31	.	.	35	.	.	*
		108	114	137			

Output 3: Data for 3 months with concerns flagged

The approach described in this paper is easy to set up. It's metadata-driven to apply default assumptions while allowing for overrides. It's adaptable to changes in data and provides a comprehensive report on the data with potential concerns flagged. The results are easy to review in order to identify concerns so it makes efficient use of the reviewer's time.

The techniques explored here expand on earlier work to perform high-level validation of results following program or platform changes by comparing results to confirm expected similarities and explore expected differences. The distinction here is that, with periodic database updates, we expect differences in data values. Unlike system conversion activities, having exactly the same values and distributions in periodic loads would indicate a problem with the data feed.

We will now look at how this approach was implemented for a specific situation, what sort of customization was done and how this could be generalized for other uses.

BACKGROUND

Data verification is the process of confirming the accuracy and consistency of data. In situations where we are loading massive amounts of new data without external opportunities to confirm the completeness, we rely on confirming reasonableness by comparing to previous results. When there are anticipated

changes at the row level as well as expected growth in the number of rows, it can be more difficult to assess newly loaded data and recognize possible failures in populating upstream data. We can, however, apply some expectations of aggregate changes and flag aggregated values that are outside of the expected range for investigation.

ALTERNATIVE APPROACHES

There are many approaches we may use formally or informally to verify data using SAS®. Sampling is quick and easy but for our use case has been insufficient to identify unexpected values or an entirely missing data source, such as the absence of inputs from one of several regions.

PROC COMPARE will detect differences in column values at the row level. When two data sets are expected to be identical or at least most rows or columns are expected to be identical then it will identify the exceptions where values differ. That would be useful for a conversion or validation effort. In the case of our high-volume monthly tables, however, differences are to be expected. Using PROC COMPARE for this would generate voluminous output without adding value.

There are, of course, trade-offs between the extent of analysis of the new data and the expense of the analysis. We have found that once this robust verification process is set up, running a full analysis of every load is well worth the cost when we consider the risk and potential human effort if invalid or incomplete data were propagated.

A COMPREHENSIVE APPROACH

Rather than sampling or comparing row by row, we classify the columns and assign each column, by name, to one or more treatment groups. Each treatment group will get a specific type of comparative analysis that's appropriate for the data in that column. The treatments may include:

- Finding the distribution of values for dimension columns and identifying new values that appear
- Counting overall rows and missing or null values
- Calculating and comparing the sums, minimum, maximum and average values for measure columns for continuous variables.

Refer to Appendix 1 for additional treatment suggestions.

Experience has shown that full verification of the newly loaded data, totaling hundreds of millions of rows at a time for us, is necessary. To minimize the effort and make it sustainable, we need a process that is:

- Thorough
- Repeatable
- Needing little or no maintenance
- Making efficient use of the reviewer's time.

Our process is primarily data-driven, actually metadata-driven, but allowing overrides in treatment group assignment minimizes the need for other customization when a new data source needs to be analyzed. A key to ensuring the analysis is comprehensive is to analyze all the columns in a table, or at least default to analyzing all columns but allow overrides at the column name level for those columns that really don't need to be analyzed. We also use optional include or exclude lists when not all columns in a table are populated at the same time. It gives us the flexibility to analyze the results of each load without waiting for all columns in the table to be populated.

If data values are unexpectedly skewed, we flag them for investigation. If, after investigating, we determine that the data is accurate but unusual, we can advise the downstream data consumers before they notice and question our data. Anticipating questions and communicating the answers improves confidence in our data and reporting and saves time for multiple data consumers who might otherwise need to explore this on their own.

HOW TO IMPLEMENT A COMPREHENSIVE SOLUTION

Tackle developing the analysis one table at a time to keep the process manageable and the output organized. Develop one treatment at a time, beginning with how that treatment would be applied to just one column. Treatment group overrides based on column name can be embedded in a macro that handles all tables since the assignment logic is only executed once per table.

GETTING STARTED

Know Your Data

While the verification process described in this paper leads to deeper knowledge of the data being loaded, some "local knowledge" and domain knowledge makes for a smoother start in establishing the verification process. It helps to begin with subject matter expertise regarding expected values and an understanding of expected metadata. Also, work with a subject matter expert to determine:

- What columns are most important
- What degree of variance is anticipated from one time period to another
- Whether there are any columns that merit special treatment
- What columns should not be analyzed or be analyzed only at the highest level because they contain wide-ranging or unreliable data.

If your database architects enforce standards for consistent naming of columns across tables, be grateful and take advantage of that when assigning analytical treatments. If, for example, every date column starts with "Date" or includes "_Date" in the column name, it will be easier to identify and classify dates for a different sort of treatment than other numeric data. Similarly, there may be database naming conventions around columns like:

- Components of addresses – Street, City, Zip or Postal Code may all have an "Address" prefix
- Flags or Boolean data – may all have "_Flag" in the column name
- Codes, which may have numeric representation but represent categorical values – may have "_Code" in the column name
- Quantities or amounts in columns containing being continuous numerics – may include "_Amount" or "QTY" or "Dollars". These may suggest a large range of values where the exact value per row isn't important but the cumulative value OR presence of positive, zero and negative values matters.

What else do we know about the data? Apart from considering naming conventions, data types and formats may help determine the best analytical treatment. Some other things to consider when deciding on treatments are:

- What are the key columns? Keys will have a large range of distinct values and the actual values aren't important BUT missing or zero or null values might be important.
- Are there columns that have so many values that the specific values don't matter but the presence of some value as opposed to null is worth counting?
- Are there any exceptional or unreasonable values such as null, zero, or negative amounts that should be counted because a change in the frequency distribution warrants investigation?
- Is there seasonality or expected natural variation by day of week/month? We might compare the most recent month to previous month or month to same month last year. Or the same analysis could be applied to compare week over week, quarter over quarter, quarter over same quarter last year, etc.
- What is the default treatment for columns that don't fall into the above? Perhaps default to one treatment for columns with numeric data and another for character data.

In the absence of enforced naming standards and familiarity with the data, it may be worth running PROC SUMMARY on all the numeric columns and PROC FREQ on the character columns initially to see the distinct values before making a decision on whether to treat each of those columns as continuous, categorical or not needing to be verified:

```
proc summary data = SASHELP.SNACKS nway missing;
    var _numeric_;
    output out = SUMMARY(drop=_type_);
run;

proc freq data = SASHELP.SNACKS;
    tables _character_ / list missing;
run;
```

Make Decisions

Begin with the table with the most columns in it. Find all data column names, types and formats in the table and save that information in a temporary SAS data set. This can be achieved using PROC CONTENTS or querying Dictionary Tables to get the simplest form of metadata. Initial decisions at the design stage involve what sort of treatment each data column requires and what level of detail is best for quantifying the results. It helps to have some knowledge of the data and expected differences in order to categorize columns but analyzing every column in detail as a first attempt and then tweaking the assignment logic later is okay.

Establish Treatment by Column Name

We let the metadata drive the default selection of analytical treatment but can specify overrides based on naming conventions or specific column names. Most columns with numeric data type (other than dates) are measures but there may be numeric values used for categorical variables as well. Using the temporary SAS data set containing column names as input to a DATA STEP, assign each column name to a treatment group depending on the type of analysis that's appropriate for that data. The choice of treatment may be based on some combination of data type, format, database column naming convention, knowledge of expected values or usage. If there are nuances to the data values that require more specialized treatment, it can be achieved by further subdividing the treatments OR by using formats which could be passed as macro variables. For example, if it's important to compare some dates down to the day level but other dates to the year level only, that could be achieved by assigning some columns to a treatment group that generates frequency counts with format year4. and other columns to a treatment group that generates counts with format yymmdd10. Output 4 below shows the result of varying the granularity with a user-define date format.

Refer to Appendix 1 for some suggested treatments by data type and subtype.

Choose Appropriate Granularity

When verifying the data, the level of granularity that's of interest might be very different from what's needed by the ultimate consumer of the data. Someone in a management or decision-making role will be interested in changes in distribution of order volume over time but, from a data quality perspective, we want clues as to whether all the data was loaded. Consider what level of granularity makes sense for each column. Perhaps ranges of values are sufficient. A subject matter expert can help suggest what level of detail is best for columns that take on a large range of values.

We may control the granularity of frequency counts for amounts or volumes using a user-defined format such as:

```
proc format;
    value amt_fmt
       low -     -1  = 'Negative       '
         0           = 'Zero           '
         1 -   10000 = 'Low Volume     '
     10001  - 100000 = 'Medium Volume  '
    100001 - high    = 'High Volume    '
        .            = 'Missing        '
;
run;
```

If our database has regular updates and the columns include START_DATE which could be any day, it might not matter at the table level exactly what the distribution of new dates is but it would matter if there are no new dates being added or if there's a reduced frequency of older dates. In this case, summarizing to the year or quarter level might be sufficient for analysis. We might expect to see an increasing count of rows for the most recent time period and a static or non-increasing count for previous years. We might use SAS date formats to control the granularity or apply a user-defined format for even more flexibility. For example, to group together all the years prior to 2001, then group together 10 year bands, then one year and then quarterly detail:

```
proc format;
value my_date
              low - '31dec2000'd = '2000 or earlier'
      '01jan2001'd - '31dec2010'd = '2001 - 2010    '
      '01jan2011'd - '31dec2020'd = '2011 - 2020    '
      '01jan2021'd - '31dec2021'd = '2021           '
      '01jan2022'd - '31mar2022'd = '2022 Q1        '
      '01apr2022'd - '30jun2022'd = '2022 Q2        '
      '01jul2022'd - '30sep2022'd = '2022 Q3        '
      '01oct2022'd - '31dec2022'd = '2022 Q4        '
;
run;
```

First Month Summarized with User-Defined Format

start_date	_FREQ_
2000 or earlier	13
2001 - 2010	87
2011 - 2020	60
2021	8
2022 Q1	1
2022 Q3	1
	170

Second Month Summarized with User-Defined Format

start_date	_FREQ_
2000 or earlier	13
2001 - 2010	87
2021	8
2022 Q1	1
2022 Q3	21
	130

Output 4: Summarizing two subsets of data

Output 4 shows what happens when our data is printed using the format MY_DATE as defined above to categorize the date values. Output 5 shows combining the two sets of data shown in Output 4 and draws

attention to the absence of data for 2011 – 2020 that might not be noticed in Output 4. That would lead us to investigate whether data for some time period was intentionally purged or was missed in the second month.

First and Second Month Combined

start_date	freq1	freq2
2000 or earlier	13	13
2001 - 2010	87	87
2011 - 2020	60	.
2021	8	8
2022 Q1	1	1
2022 Q3	1	21
	170	130

Output 5: Comparing the two monthly subsets

Expected Variability

We need to consider our tolerance for variation to determine what should be flagged for investigation. That is, should we flag a 2% difference in the sums of a metric month over month or is it only noteworthy if the change is 5%? Or does only greater than 10% change matter? What about the percentage difference in distribution of values for attributes? There may need to be different levels of flagging for different tolerances. For example, flag +/- 5% with one style and flag +/- 10% with another.

Whether for metrics or attributes, we look at the absolute value of the difference, but that is definitely a choice that might be different for other use cases.

Building the Logic

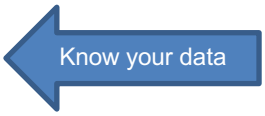
The components can be rearranged depending on the requirement. The building blocks for the process are simple PROCs and DATA STEPS that are used to:

- Define macro variables for the selection of the subsets (such as different month end dates)
- Define and code what sort of treatments will should be applied. (Refer to Appendix 1 for suggestions.) These will be described as TREATMENT1, TREATMENT2, TREATMENT3, etc.
- Create or filter for the subsets of data to be compared. For our example that will be a series of snapshot dates
- Capture the column names, types and formats from the most recent subset of data from the table in a temporary SAS data set
- Decide how each treatment group should be analyzed for verification
- Assign column names to treatment groups as described in the “Establish Treatment by Column Name” section above. Depending on the environment, treatment types assigned based on naming conventions may be common to just one table, a data mart or an entire database. The treatment assignment logic may hard-coded in the assignment step, included as an %INCLUDE module or passed as a parameter depending on the scope of the analysis.
- Create a list per treatment of the column name(s) that treatment should be applied to
- For each table, apply each treatment:
    - to every column named in the column list for that treatment via the column list

- for each of the comparative subsets of data
- combine and compare the results of analyzing each subset to identify variability across the subsets
- compare the variability to our tolerance and flag concerns
- write the results with the flags to our output destination.

Let's work through an example of how to build and apply the logic blocks, substituting SASHELP.SNACKS for our database table. First, we'll see how to capture the metadata to drive the treatment group assignments. Running the code shown here produces the results shown as Output 6:

```
proc contents data = SASHELP.SNACKS
              out  = CONTENTS(keep=name type format label);
run;
```



Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
3	Advertised	Num	8		Advertised (1=yes)
5	Date	Num	8	DATE9.	Date of sale
4	Holiday	Num	8		Holiday (1=yes)
2	Price	Num	8		Retail price of product
6	Product	Char	40		Product name
1	QtySold	Num	8		Quantity sold

Output 6: Results of running PROC CONTENTS on SASHELP.SNACKS

Continuing the example from above, we can see from the column labels that columns "Advertised" and "Holiday" are Boolean. "Price" and "QtySold" are both numeric. We can learn from running a preliminary PROC SUMMARY or PROC FREQ or browsing the data set that "Price" actually has only a handful of distinct values and "QtySold" has a large number of values. "Product" is character but takes on a limited number of values, so a count of the distribution of those values would be meaningful. Column "Date" is a date value but rather than the specific date, summarizing to monthly level is sufficient for our purposes. So, we'd want three different treatment types for our data. Table 1 shows the treatments. This classification into treatment groups seems like the best fit for this particular data but it is subjective. Others might choose different classification or different treatment for each group of columns.

Treatment Group	Description	Analysis Assigned
1	Categorical, whether character or numeric (including Boolean values)	Frequency count by value
2	Numeric, continuous	Sum
3	Date	Frequency count by value, summarized to monthly level

Table 1: Treatments to be applied

The step to assign treatments might look like this:

```
data TREATMENTS(drop=upcase_name);
    set CONTENTS;
    upcase_name = upcase(name);

    if index(format, 'DATE') then treatment_group = 3;
        else if index(upcase_name, 'QTYSOLD') then treatment_group = 2;
        else treatment_group = 1;
run;
```

Note that we default to assigning column names to a treatment group (Treatment Group 1 in this example) if they don't have a special treatment. This lets us manage by metadata. The process is dynamic so if new columns are added to our database table, they won't be inadvertently left out of the verification. Output 7 shows the result of the DATA step.

Dump the TREATMENTS data set

NAME	TYPE	LABEL	FORMAT	treatment_group
Advertised	1	Advertised (1=yes)		1
Holiday	1	Holiday (1=yes)		1
Price	1	Retail price of product		1
Product	2	Product name		1
QtySold	1	Quantity sold		2
Date	1	Date of sale	DATE	3
N = 6				

Output 7: Results of assigning column names to treatment groups

Now that every column name has been assigned to a treatment group, based on data type, data format, possibly the column naming convention plus what we know about the column usage, we'll create a macro variable list of column names per treatment group:

```
proc sql noprint;
    select name into: column_list1 separated by ' '
    from TREATMENTS
    where treatment_group  = 1 ;

    select name into: column_list2 separated by ' '
    from TREATMENTS
    where treatment_group  = 2 ;

    select name into: column_list3 separated by ' '
    from TREATMENTS
    where treatment_group  = 3 ;
    ;
quit;
```

Again, the ongoing process is dynamic and won't need to be modified when new columns are added to the table unless we need to define a whole new treatment group. Of course, we could turn the PROC SQL step above into a macro loop that creates a macro string named column_list&i where the treatment_group = &i and is repeated over the number of distinct treatment groups in the TREATMENTS table. Output 8 shows the results of the macro variables created above.

```
** Macro variable column_list1 contains: Advertised Holiday Price Product **
 ** Macro variable column_list2 contains: QtySold **
** Macro variable column_list3 contains: Date **
```

Output 8: The result of %PUT statements for our macro lists

We've decided on the type of analysis that's appropriate for each category of variables (which we'll describe as Treatment1, Treatment2 and Treatment3), assigned a treatment group for each column name and generated a macro variable list of the names of columns to be subject to each style of analysis (which we called column_list1, column_list2 and column_list3).

Define – or derive based on current date or maximum date in the data table – what time points are to be analyzed and assign those dates to macro variables. Continuing with the SASHELP.SNACKS data set, which only holds data up to 2004, let's choose 4 months ending in that year to compare and assign these values:

```
%let date0 = '31oct04'd;
%let date1 = '30sep04'd;
%let date2 = '31aug04'd;
%let date3 = '31jul04'd;
```

Date macro variables

For an actual database table, the date macro variables above would be in the form dates are stored in that database. Split the data into subsets for comparison. We will work with 4 subsets of data and compare the latest time period to data from the previous time period.

Now it's time to code the actual treatments. We'll start coding to analyze a single column, using what will we designate as Treatment 1. The flow for analyzing one column of data with Treatment 1 is illustrated in Figure 1. Figure 2 shows Treatment 1 integrated into the overall flow.

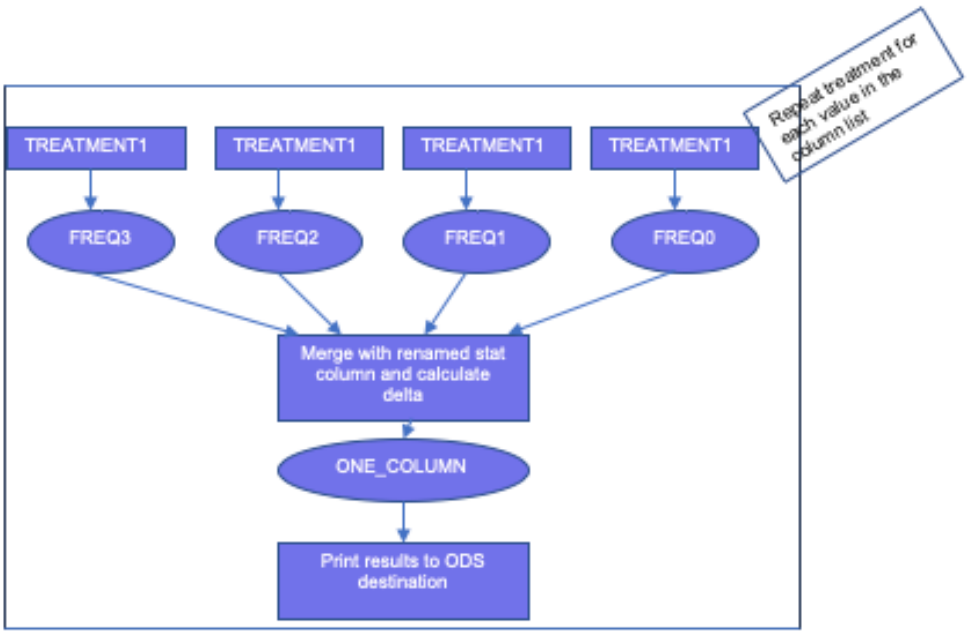


Figure 1: Flow for analyzing one column using Treatment 1

The first attempt at coding the above might look like this, for just one specific column name:


```
proc freq data = SET0;
     tables Advertised / list missing
                    out = FREQ0(keep = Advertised count
                                rename=(count = count0));
run;

proc freq data = SET1;
     tables Advertised / list missing
                    out = FREQ1(keep = Advertised count
                                rename=(count = count1));
run;

proc freq data = SET2;
     tables Advertised / list missing
                    out = FREQ2(keep = Advertised count
                    rename=(count = count2));
run;

proc freq data = SET3;
     tables Advertised / list missing
                    out = FREQ3(keep = Advertised count
                                rename=(count = count3));
run;

data ONE_COLUMN;
    merge FREQ3 FREQ2 FREQ1 FREQ0;
        by Advertised;
    abs_diff = abs(count0 - count1);
    pct_diff = 100* abs_diff / count1;
run;

proc print data = ONE_COLUMN noobs;
run;
```

Once that works, the code could be converted to a macro to reduce the overall number of lines of code:

```
%macro TREATMENT1;

    %do i = 0 %to 3;
        proc freq data = SET&i;
             tables Advertised / list missing
                            out = FREQ&i(keep = Advertised count
                            rename=(count = count&i));
        run;
    %end;

%mend TREATMENT1;

%TREATMENT1;

data ONE_COLUMN;
    merge FREQ3 FREQ2 FREQ1 FREQ0;
        by Advertised;
    abs_diff = abs(count0 - count1);
    pct_diff = 100* abs_diff / count1;
run;

proc print data = ONE_COLUMN noobs;
    title 'Treatment 1 using simple loop macro';
run;
```

Then the macro could be adapted to use a variable column name:

```
%macro TREATMENT1(sets_to_compare_to_base, column_name);

    %do i = 0 %to &sets_to_compare_to_base;
        proc freq data = SET&i noprint;
             tables &column_name. / list missing
                                    out = FREQ&i(keep = &column_name. count
                                    rename=(count = count&i));
            run;
        %end;

    data ONE_COLUMN;
        merge FREQ3 FREQ2 FREQ1 FREQ0;
            by &column_name.;
        abs_diff = abs(count0 - count1);
        pct_diff = 100* abs_diff / count1;
        if pct_diff > 20 then flag = '*';
    run;

    proc print data = ONE_COLUMN noobs;
        format pct_diff 8.2;
        title "Treatment 1 for &column_name. using loops macro with parameters";
        footnote "Tolerance for flagging here is 20%";
    run;
    %mend TREATMENT1;
```

Code each Treatment as a macro

If we repeat TREATMENT1 over all the columns in &column_list1, the results will be as shown in Output 9. It shows a large number of flags so we'd want to consider whether there was a problem with the data and if there wasn't, then reconsider the type and sensitivity of the treatment applied to the columns.

We will develop the additional two treatments. The structure for the TREATMENT2 macro is quite similar to TREATMENT1 but the analysis uses PROC SUMMARY instead of PROC FREQ. Figure 3 illustrates Treatment 2.

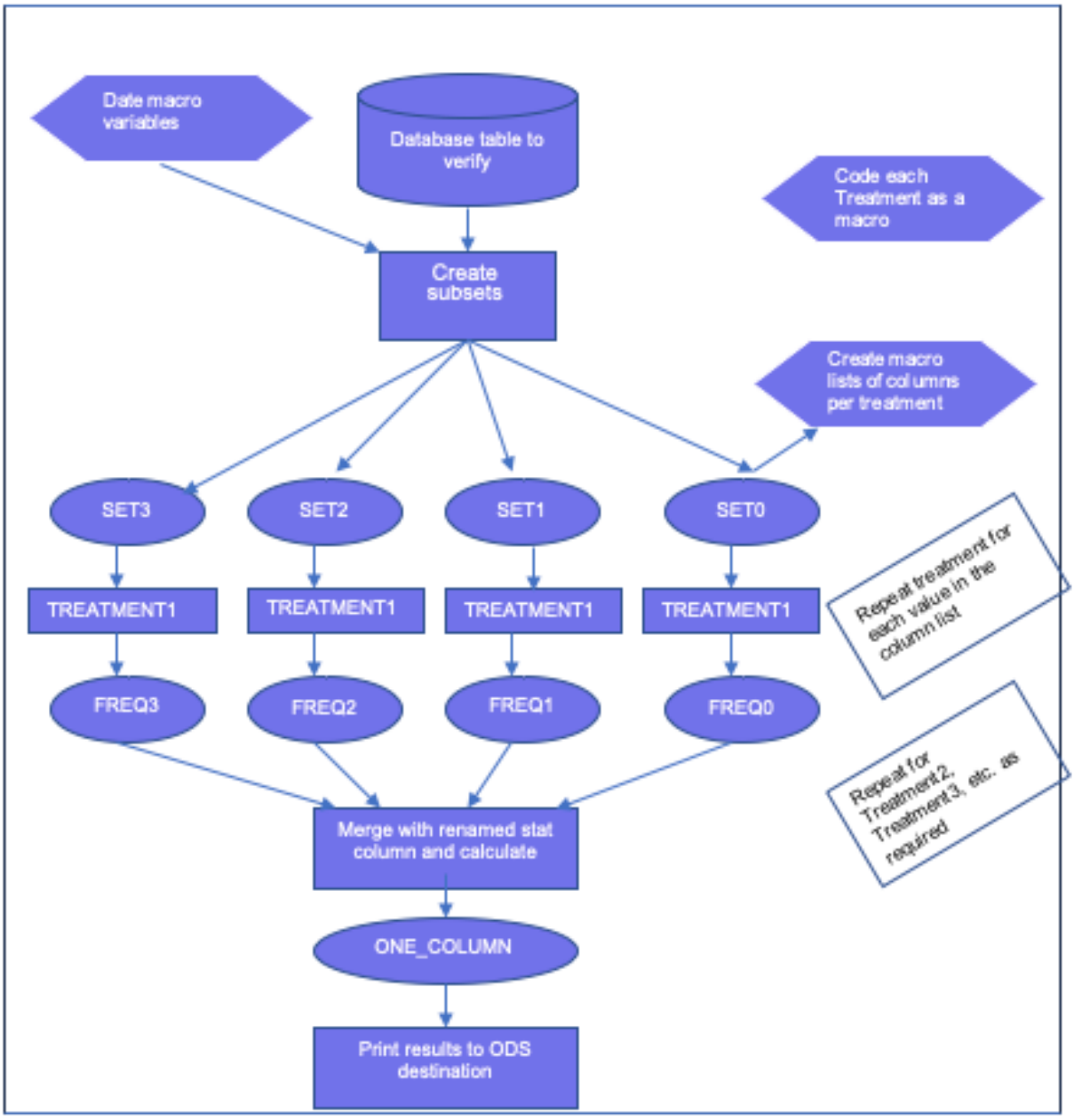


Figure 2: Overall flow for our month over month verification process for Treatment 1

Treatment 1 for Advertised using loops macro with parameters

Advertised	count3	count2	count1	count0	abs_diff	pct_diff	flag
0	988	962	982	550	432	43.99	*
1	97	123	68	80	12	17.65	

Tolerance for flagging here is 20%

Treatment 1 for Holiday using loops macro with parameters

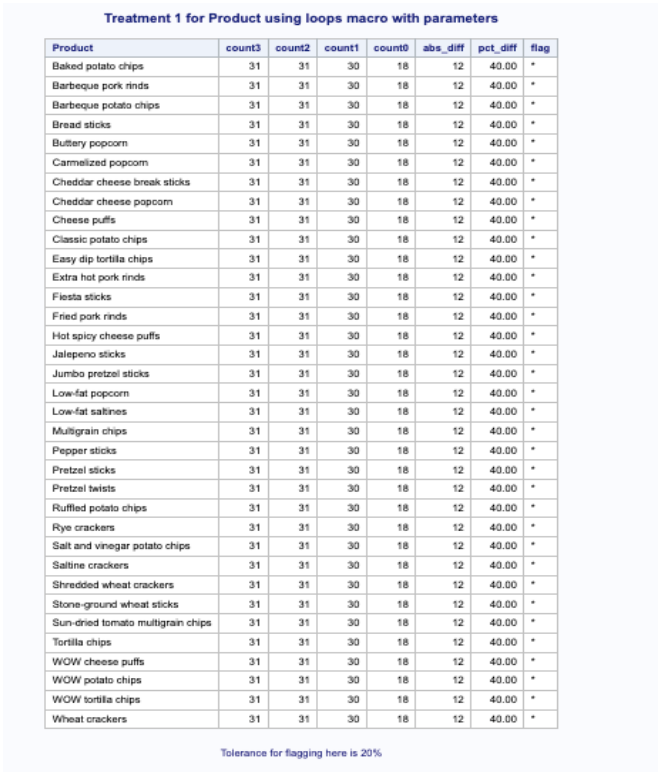
Holiday	count3	count2	count1	count0	abs_diff	pct_diff	flag
0	875	1085	805	630	175	21.74	*
1	210	.	245	.	.	.	

Tolerance for flagging here is 20%

Treatment 1 for Price using loops macro with parameters

Price	count3	count2	count1	count0	abs_diff	pct_diff	flag
0.99	155	155	150	90	60	40.00	*
1.49	138	131	147	72	75	51.02	*
1.99	265	278	250	162	88	35.20	*
2.49	149	149	113	104	9	7.96	
2.99	347	341	360	184	176	48.89	*
3.49	31	31	30	18	12	40.00	*

Tolerance for flagging here is 20%



Treatment 1 for Product using loops macro with parameters

Product	count3	count2	count1	count0	abs_diff	pct_diff	flag
Baked potato chips	31	31	30	18	12	40.00	*
Barbeque pork rinds	31	31	30	18	12	40.00	*
Barbeque potato chips	31	31	30	18	12	40.00	*
Bread sticks	31	31	30	18	12	40.00	*
Buttery popcorn	31	31	30	18	12	40.00	*
Carmelized popcorn	31	31	30	18	12	40.00	*
Cheddar cheese break sticks	31	31	30	18	12	40.00	*
Cheddar cheese popcorn	31	31	30	18	12	40.00	*
Cheese puffs	31	31	30	18	12	40.00	*
Classic potato chips	31	31	30	18	12	40.00	*
Easy dip tortilla chips	31	31	30	18	12	40.00	*
Extra hot pork rinds	31	31	30	18	12	40.00	*
Fiesta sticks	31	31	30	18	12	40.00	*
Fried pork rinds	31	31	30	18	12	40.00	*
Hot spicy cheese puffs	31	31	30	18	12	40.00	*
Jalepeno sticks	31	31	30	18	12	40.00	*
Jumbo pretzel sticks	31	31	30	18	12	40.00	*
Low-fat popcorn	31	31	30	18	12	40.00	*
Low-fat saltines	31	31	30	18	12	40.00	*
Multigrain chips	31	31	30	18	12	40.00	*
Pepper sticks	31	31	30	18	12	40.00	*
Pretzel sticks	31	31	30	18	12	40.00	*
Pretzel twists	31	31	30	18	12	40.00	*
Ruffled potato chips	31	31	30	18	12	40.00	*
Rye crackers	31	31	30	18	12	40.00	*
Salt and vinegar potato chips	31	31	30	18	12	40.00	*
Saltine crackers	31	31	30	18	12	40.00	*
Shredded wheat crackers	31	31	30	18	12	40.00	*
Stone-ground wheat sticks	31	31	30	18	12	40.00	*
Sun-dried tomato multigrain chips	31	31	30	18	12	40.00	*
Tortilla chips	31	31	30	18	12	40.00	*
WOW cheese puffs	31	31	30	18	12	40.00	*
WOW potato chips	31	31	30	18	12	40.00	*
WOW tortilla chips	31	31	30	18	12	40.00	*
Wheat crackers	31	31	30	18	12	40.00	*

Tolerance for flagging here is 20%

Output 9: Result of running TREATMENT1 over &column_list1

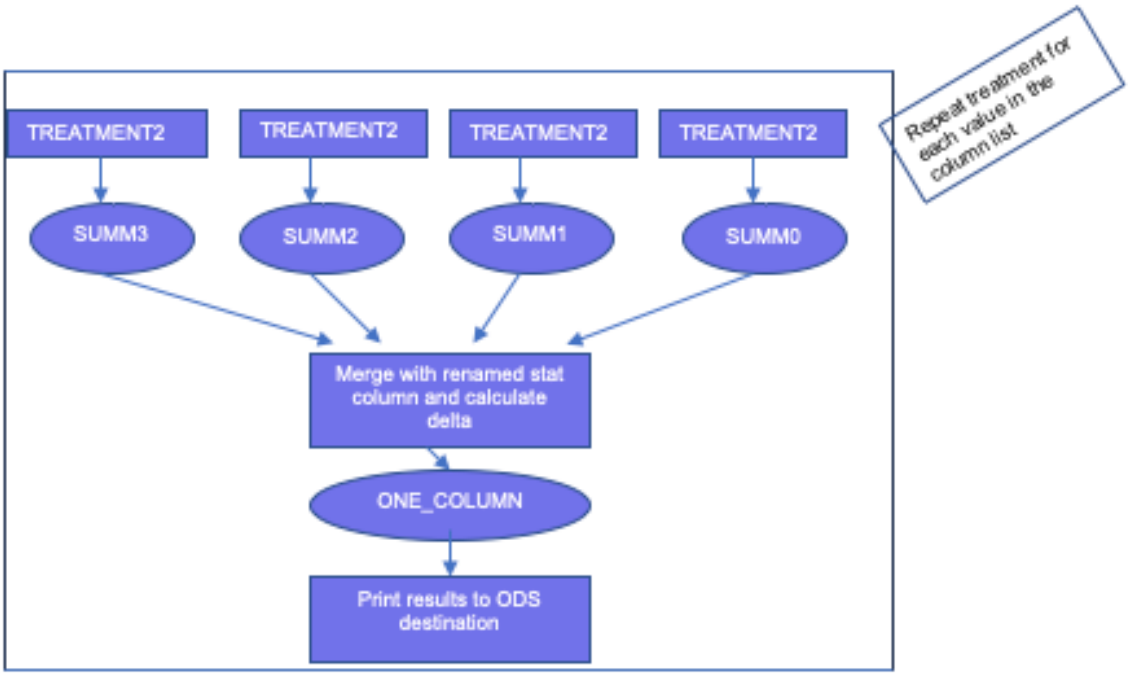


Figure 3: Flow for analyzing one column using Treatment 2

The flow for TREATMENT3 is the same as TREATMENT1. The only difference is the use of the MONYY7. format to group the counts by month instead of by day. The code is shown here:

```
%macro TREATMENT3(sets_to_compare_to_base, column_name);

    %do i = 0 %to &sets_to_compare_to_base;
        proc freq data = SET&i noprint;
             tables &column_name. / list missing
                                    out = FREQ&i(keep = &column_name. count
                                    rename=(count = count&i));
             format &column_name. monyy7.;
            run;
        %end;

    data ONE_COLUMN;
        merge FREQ3 FREQ2 FREQ1 FREQ0;
            by &column_name.;
        abs_diff = abs(count0 - count1);
        pct_diff = 100* abs_diff / count1;
        if pct_diff > 20 then flag = '*';
    run;

    proc print data = ONE_COLUMN noobs;
        format pct_diff 8.2;
        title "Treatment 3 for &column_name. using loops macro with parameters";
        footnote "Tolerance for flagging here is 20%";
    run;
    %mend TREATMENT3;
```

Granularity

To eliminate the need for TREATMENT3, we could enhance the coding of TREATMENT1 by adding another macro variable for the format.

For each run, we use a main macro that is executed once per table being analyzed, with parameters to control the execution. Since different columns in one table may be refreshed on different days, some tables may have multiple passes of the verification process on different days in the same month where each pass explicitly includes or excludes some columns. What the main macro does:

- Iterate through the list of column names assigned to each style of analysis and run the analysis (frequency by value, sum, count, etc.) for each of the data subsets by calling the Treatment macro and consolidating the results for each column.
- Combine the results of the analyses for each subset, one column name at a time, side-by-side and calculate % change between the most recent and the previous period or data set.
- Flag each row of output if the change is greater than the defined tolerance. We may be comparing sums, averages, counts, or the distribution of values.
- We use macro variables for an optional DROP or KEEP list of columns to verify. If not all columns in a table are updated on the same date, there may be a need to run the comparison for only certain columns. For example if <Column1, Column2 and Column3> get updated on the first day of each month but <Column4> doesn't get updated until the 15th of the month, there's no point in showing a 100% change in Column4 at the time it makes sense to be analyzing the first 3 columns. Running the verification in two passes, with DROP Column4 as a parameter on the first pass and KEEP Column4 on the second pass will be more efficient and more meaningful.

There are parameters to supply the table name and specify column names to include or exclude in case different columns are populated at different times. Without an include or exclude list, the default is to analyze all columns, subject to the treatment group assignment.

USING THE COMPREHENSIVE VERIFICATION PROCESS

Structuring the Process to Invoke the Logic

To maintain flexibility, our periodic verification process uses a macro to process one table name at a time, comparing subsets of data from that table as of different dates. It is standardized to derive the comparison dates but these could be passed as parameters to allow even greater flexibility. Other parameters are formats for appearance or summarization and an include or exclude column list.

Call the main macro to analyze each table. One driver program may invoke the macro multiple times, once for each table. Figure 4 shows the steps that one invocation of the driver might include. This would analyze one table at 4 different points in time using 3 different treatments.

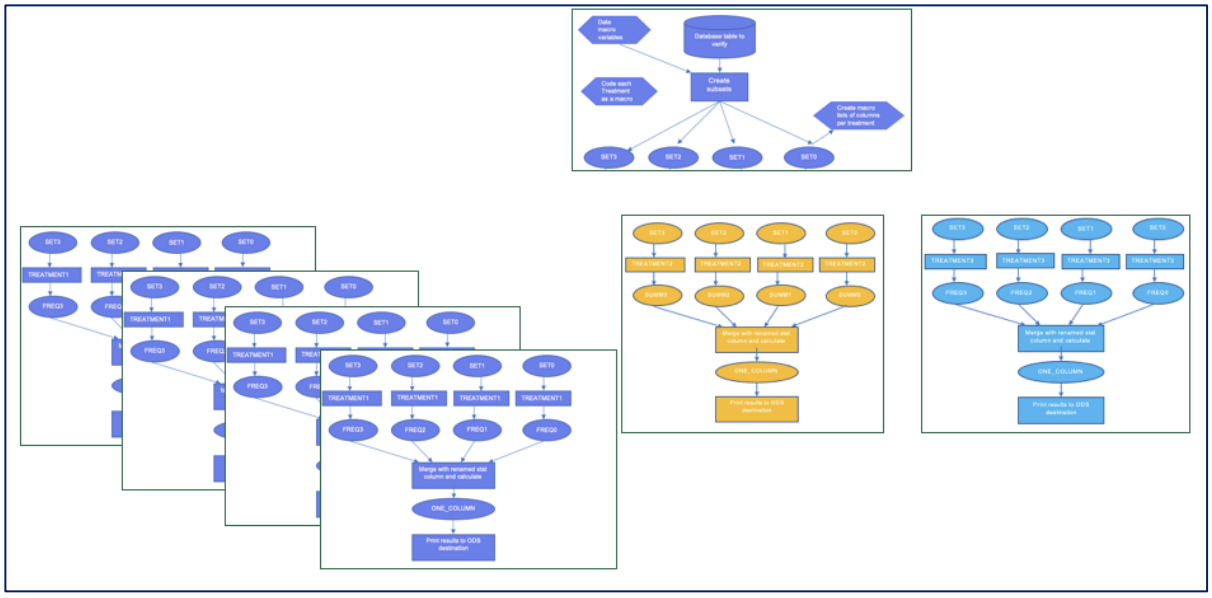


Figure 4: Overall flow for our month over month verification process, comparing 4 points in time with 3 treatments iterated over column names

Analyzing Periodic Loads

These generated analyses aren't helpful unless someone with sufficient understanding of the data reviews them promptly and follows up on any flagged concerns. Comparing values across months can generate a lengthy report. We make easy to review the reports by using visual clues to draw attention to potential concerns with the data.

A dramatic increase or even a month over month trend of increases in the frequency of zeroes, nulls, "n/a" or "Other" values might alert us to a new problem with a data feed. These could be a sign of data from a new source being added to our tables without being properly coded, for example.

Data Insights: An Unexpected Benefit

The intention was for this analysis of data to be preliminary to any actual use of the data to provide insights to our data consumers. There was an unexpected benefit, though. Applying this comprehensive automated comparison process to multiple tables with many columns can produce screen after screen filled with columns of numbers representing detailed distributions. While it isn't the slickest looking report, a quick visual review of the results can provide a lot of insight and a reminder about correlations in the data for someone who is familiar with using the data. We can also quickly confirm that any expected new values are reflected.

Additional Use Cases

Once we have a comprehensive verification process built, it can be used for more than just comparing similar subsets of data at different points in time. Other applications include system validation and data profiling.

Validation

A very similar process can be applied to the data validation we might do after a system change. It would involve comparing just two sets of data instead of three but the comparative logic per type of data column would be the same. Wrapping the logic in macros allows it to be redeployed this way. In this case, we expect the data to be the same before and after the change, or at least mostly the same but with some expected differences. Instead of comparing subsets of data from the same database at different dates, we would compare data from two different databases as of the same date. This would typically involve

just 2 comparison sets, ‘before’ and ‘after’ or ‘original’ and ‘modified’. Other than that, the same logic can be applied to confirm equivalence or highlight differences at a summarized level. In this case, the desired outcome is little or no difference. The tolerance for change in values would be zero and any differences would need to be investigated. Figure 4 shows the flow of that for one treatment, as an example.

A combination of our original requirement and the original versus modified comparison approach could be used for verifying loads of batches of data that aren’t loaded on a regular schedule.

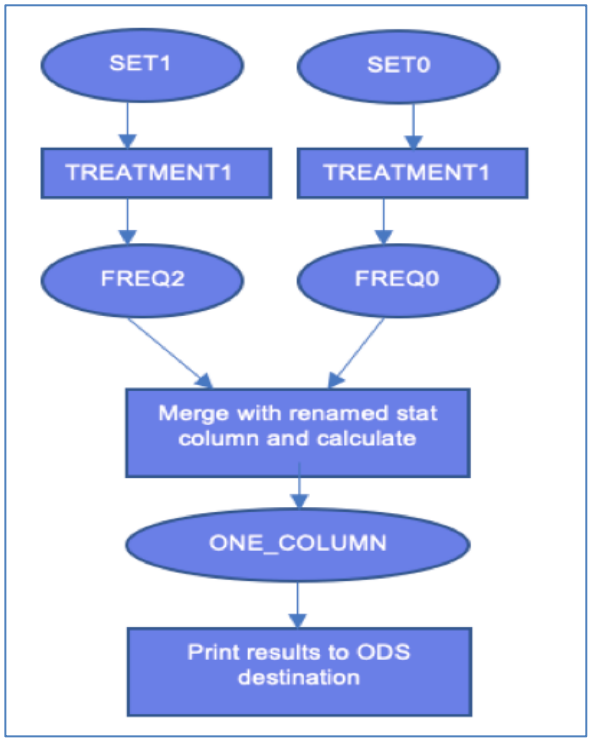


Figure 4: Flow for comparing Before and After data

Data Profiling

Because the comprehensive verification analyzes the distribution of all metrics and dimensions in a set of data, it could also be easily adapted to profile one subset of data against another. While our initial requirement was to compare subsets of a table at different points in time, it could also be used to profile the metrics of one region against others as of the same point in time to identify regional differences beyond the obvious ones like location. Once the logic described above is built, it can be repurposed and profiling is just a WHERE clause away.

This can also be used as a problem-solving step. If the periodic verification of data by point in time raised concerns and there was a hypothesis as to where there was a problem in the data, rerunning the same periodic verification logic with a WHERE clause added when creating the subsets could be used validate the hypothesis.

Although the process described throughout this paper refers to multiple database tables, it could easily be applied to a collection of SAS data sets or series of subsets of one SAS data set or any other collection of data that could be readily imported into a SAS data set for comparison.

Adding a WHERE parameter apart from the date to the verification process would make the process ready to be applied for data profiling at any time.

ENHANCING THE PROCESS

Our process continues to evolve as it gets applied to new data sources and we find new ways to make the analyses valuable. There are also lots of possible enhancements to make the process friendlier and more flexible.

Adding Visual Cues

We've arrived at a fairly simple method of flagging potential concerns in the analysis report: adding an alert in the rightmost column of the output. If the distribution is similar to the prior period then the alert column is blank. If the distribution is unexpected then there are a series of asterisks, where number of asterisks corresponds to the degree of difference. That means scanning down one column of the output will identify potential concerns.

There are several SAS papers and presentations available on how to use traffic-lighting with ODS output. This would be a good way to draw attention to discrepancies of concern with colour-coding.

With such a powerful process, there's a long list of possible enhancements to make it even more broadly applicable, incorporating the additional use cases described above. Some of the wish list items are:

- Make the number of comparison sets variable. Our preferred number for comparing corresponding sets of data at different points in time, such as monthly, is the base (most recent) set plus 3 prior sets. Making the number of subsets macro driven would allow the expansion to additional time periods, address use for validating program or feed changes by comparing "Before" and "After" data sets and use for data profiling. This would make the number of iterations of every analysis step macro variable driven and also make the lists of data sets in the program logic macro driven. Implementing this is non-trivial but definitely doable.
- Make the tolerances for flagging variability parameters (percentage change in sum and percentage change in distribution) so they can be varied at the table level. Combining this with the use of include or exclude column list would give ultimate control of tolerances at the column level.

Further Automation

There are some enhancements that could make the ongoing use of the process friendlier:

- Kick off the verification of each table or set of columns automatically when the data is populated. Instead of relying on a manual intervention here, we could use an automated process to either query a checkpoint table that is updated when the load completes and appears to be successful or query the relevant tables directly to look for new snapshot dates.
- Make the process entirely data-driven to further reduce the need for customization for new data sources or columns. Use PROC SUMMARY to check number of distinct values occurring for each column and use the result of that to automatically assign the columns that have a very large number of values to an analysis category that doesn't list all the details.
- Highlight columns requiring follow-up in an email message. Currently, at the completion of the analysis run there is an automated email message to advise that the results are ready for review in a particular location. Flagging of results that are outside of our tolerances is already done in a DATA STEP within the process. The resulting data set could be filtered to find which data columns and values are outside of our tolerances and a filtered PRINT of that data set could be incorporated into the email message.

CONCLUSION

Once a macro has been written to handle processing each of the analysis treatments over a macro list of column names and initial assignments and overrides have been established for a table or set of tables, we can get an automated analysis of data loaded monthly, in a convenient format that makes it easy to spot concerns or recognize changes.

This solution is comprehensive. Because it can automatically generate analysis for all columns, there is no temptation to only verify critical or most-used columns. Sometimes a significant concern is identified based on a pattern of discrepancies in the distribution of values in less frequently used data columns. It has proven to be absolutely worth the processing time and brief human review time to catch problems as well as unexpected results which are not data problems but may generate questions or uncertainly about the downstream results.

This approach is a complement to, not a substitute for, subject matter expertise. It allows for efficient use a subject matter expert's time. Verifying data helps us to trust the data we use and share with our data consumers. Having early insight into data changes and results, as well as possible data issues, also improves our credibility.

ACKNOWLEDGMENTS

Thanks to Marje Fecht for her very enthusiastic encouragement for enhancing this verification process and writing about it.

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

Lisa Eckler

Lisa Eckler Consulting Inc.

lisa.eckler@sympatico.ca

Appendix 1: Possible Treatment by Usage

Data Type	Sub-Type or Description	Examples	Possible Analyses for Verification
Numeric	Discrete	Data that comes from a limited set of values, also known as discrete values (often but not always integers) like: • Grade level • Age (in years) • Year of birth • Tenure (in years) • Number of interactions last month • Number of people in household • Sales Rep ID (if numeric) • Shoe size	Frequency count per individual value or range
Numeric	Continuous	Data that can take on a large range of values, like: • Hours or Days Since Inception • Bank Balance • Net Worth • Sales Volume	Sum, average, maximum, minimum, count nulls OR frequency count per range
Numeric	Dates	Data stored as numeric in SAS that can take on a vast range of value like: • Date of Birth • First Interaction Date • Transaction Date • Month End Date • Most Recent Interaction Date	Frequency count per range of values
Character	Categorical data	Data that comes from a limited set of values like: • Province/State Code • Country • Sales Region • True/False/Null • Yes/No/Maybe • Age Range • Letter Grade (ranking) • Sales Rep ID (if alphanumeric)	Frequency count per individual value
Character	String	Data that can take on many values like: • Customer name • City/Town name • Comment • Description	Count blank strings or those containing “n/a” or other default indicator
Character	Structured string	Data string with an expected length or pattern like: • Postal Code or Zip Code • Phone number	Perl Regular Expression
Character or Numeric	Key	Data that takes on a large range of unique values but should never be blank or null, such as: • Account Number • Employee ID	Frequency count of blank or null values