Paper 3047-2015

Examination of Three SAS® Tools for Solving a Complicated Data Summary Problem

Carry Croghan, US-EPA;

ABSTRACT

When faced with a difficult data analysis problem, a SAS® programmer has many options to solve the problem. In this paper, three different approaches are evaluated and compared in terms of effectiveness, efficiency, debugging, and clarity of the code. The three methods include linearizing the data, using SQL Cartesian joins, and using sequential data processing. Inconsistencies in the raw data caused the data linearization to be problematic. The output generated by a multi-stage, air monitoring instrument that collected several measurements every 5 minutes for 9 years resulted in more than 800,000 records. The large number of records and the need for many-to-many merges resulted in a long run time for the SQL code. The sequential data processing, although older technology, provided the most time efficient and error-free results.

INTRODUCTION

The US EPA deploys air pollution monitors in various locations that collect a wide range of measurements used to address many different research questions (US EPA Monitoring Network Information, Bereznicki 2012, Kimbrough 2013). The output from one complex multi-stage instrument is the subject of this paper; however, methods described here can be applied to a number of other continuous datasets. These complex monitors not only collect continuous measurement of multiple chemicals, but they also perform automated data quality checks, self- cleaning, and recalibration. The results of all of these stages of the monitors may be output to a single data output stream. Once the data collection has been completed, and the instrument files are returned to the laboratory, the data must be cleaned, data qualifications applied, summarized and composite variables calculated.

The particular instrument cycled phases which included measurement chemical compound and performing analyses for alternate forms as well as a cleaning and calibration phase. The five phases of data collection together are referred to as a cycle. The instrument is automated and controls the flow of air from an ambient source and a clean air canister of known quality. It also turns additional testing devices on and off, and logs the instrument data. Each phase is given a data qualifier so the operator knows what measurement is being taken and when the instrument is cleaning or calibrating.

The automated data collection cycle for the instrument was as follows: collect Ambient Measurements of a pollutant every 5 minutes for 2 hours, clean the instrument for 15 minutes, perform an analysis for an alternate form of the chemical (Test A) for 20 minutes, immediately perform an analysis of another form of the chemical (Test B) for 15 minutes, then clean the instrument again for 10 minutes before returning to Ambient Measurement collection (see Figure 1). During cleaning the instrument is flushed with a clean air canister. This serves two functions: (1) it cleans the instrument of any remaining ambient air and (2) is used to identify the zero level which is used to calibrate the results.

At this specific EPA location, there were several measurements of the pollutant being continuously monitored at each time point. Since the processing and calculations are identical, this paper will limit the discussion to just one of the measurements consisting of a single pollutant and two of its related forms.

Ambient Measurement Data

Date	Time	Status	phase	ng/m3		
01-09-13	08:30:01	OK	1	3.273		
01-09-13	08:35:01	OK	1	0.672		
01-09-13	08:40:01	OK	0	1.936		
01-09-13	08:45:01	OK	0	0.765		
01-09-13	08:50:01	OK	0	2.383		
01-09-13	08:55:01	OK	0	0.816		
01-09-13	09:00:01	OK	0	2.293		
01-09-13	09:05:01	OK	0	0.898	Ambient	
01-09-13	09:10:01	OK	0	2.336	Measurements	
01-09-13	09:15:01	OK	0	0.849		
…	…	…	…	…		
01-09-13	10:25:01	OK	0	0.984		
01-09-13	10:30:01	OK	0	2.967		
01-09-13	10:35:01	OK	0	1.074		
01-09-13	10:40:01	OK	1	11.111		One Cycle
01-09-13	10:45:01	OK	1	4.609	Cleaning	
01-09-13	10:50:01	OK	1	11.960		
01-09-13	10:55:01	OK	2	8.740		
01-09-13	11:00:01	OK	2	152.335	Test A	
01-09-13	11:05:01	OK	2	6.969		
01-09-13	11:10:01	OK	2	10.999		
01-09-13	11:15:01	OK	3	229.037		
01-09-13	11:20:01	OK	3	5.909	Test B	
01-09-13	11:25:01	OK	3	1.813		
01-09-13	11:30:01	OK	1	2.992	Cleaning	
01-09-13	11:35:01	OK	1	531.788		

Figure 1. Sample of raw data from the instrument demonstrating one cycle

For each cycle, the following summary calculations were needed.

1. The average of all the Ambient Measurement values (except the first one).
2. The sum of the Test A values minus three times the previous cleaning value.
3. The sum of the Test A values (except the first one) minus two times the previous cleaning value.
4. The sum of the Test B values minus two times the subsequent cleaning value.

The challenges presented with this complex instrument are many. The individual cycles and special values for each cycle must be identified. Special values include for example, the first ambient measurement or the first value measured during Test A. Because of the mechanics of the monitor a switch in air source is not instantaneous, thus the first measurement after an air source switch is not of the full sampling duration and is inaccurate. Therefore, this measurement is not to be used in the calculations. It is easier to subtract it from the total, than to determine the total of all values except it. Complications are added due to incomplete cycles caused by instrument failure and regular servicing when the instrument was shut down. There was some inconsistency in the timing of individual cycles. All problem cycles had to be identified and removed along with any cycles containing data with a status other than 'OK'. An additional challenge was introduced because the instrument was replaced with a similar, but not identical model during the 9 years of monitoring. Only after addressing these issues could the requested summaries for the remaining 800,000 records be calculated.

SAS®, like most robust programming languages, offers a wide variety of methods to solve most problems. If correctly applied, the final results are the same and the decision of which tool to use is a matter of efficiency, ease of implementation, and personal preference. Three methods will be discussed in this paper: Linearized Approach, SQL Cartesian Joins Approach, and Sequential Processing Approach.

LINEARIZED APPROACH

The first approach was implemented by processing all the data within a single DATA step using multiple LAG functions to "horizontalize" the data. The intension of this approach is to linearize all data for each cycle on a single record. Linearizing is equivalent to transposing a tall dataset to a wide one, which is also called unstacking. Once each complete cycle was on a single record, the required composite variables were calculated using summation functions.

The code for the Linearized Approach to process one of the six measurements is:

```
data outfile;
        set in_file;

          lag1Phase=lag1(Phase);        lag1ng_m3=lag1(ng_m3);
          lag2Phase=lag2(Phase);        lag2ng_m3=lag2(ng_m3);
          lag3Phase=lag3(Phase);        lag3ng_m3=lag3(ng_m3);
          lag4Phase=lag4(Phase);        lag4ng_m3=lag4(ng_m3);
[30 lines not shown]
          lag34Phase=lag34(Phase);      lag34ng_m3=lag34(ng_m3);
          lag35Phase=lag35(Phase);      lag35ng_m3=lag35(ng_m3);
          lag36Phase=lag36(Phase);      lag36ng_m3=lag36(ng_m3);



 if lag1Phase=1 and lag2Phase=3 and lag3Phase=3 and lag4Phase=3 and
    lag5Phase=2 and lag6Phase=2 and lag7Phase=2 and lag8Phase=2 and
    lag9Phase=1 and  lag10Phase=1 and lag11Phase=1 and lag12Phase=0 and
    lag13Phase=0 and lag14Phase=0 and lag15Phase=0 and lag16Phase=0 and
    lag17Phase=0 and lag18Phase=0 and lag19Phase=0 and lag20Phase=0 and
    lag21Phase=0 and lag22Phase=0 and lag23Phase=0 and lag24Phase=0 and
    lag25Phase=0 and lag26Phase=0 and lag27Phase=0 and lag28Phase=0 and
```

```
    lag29Phase=0 and lag30Phase=0 and lag31Phase=0 and lag32Phase=0 and
    lag33Phase=0 and lag34Phase=0 and lag35Phase=0 and lag36Phase=1
then do;

    TestA = (lag7ng_m3+lag6ng_m3+lag5ng_m3)-(3*lag9ng_m3);

    TestA_2 = (lag8ng_m3+lag7ng_m3+lag6ng_m3+lag5ng_m3)-(4*lag9ng_m3);

    TestB = (lag4ng_m3+lag3ng_m3+lag2ng_m3)-(3*lag1ng_m3);

    AverageAmb = mean(lag12ng_m3, lag13ng_m3, lag14ng_m3,lag15ng_m3,
                lag16ng_m3, lag17ng_m3, lag18ng_m3, lag19ng_m3, lag20ng_m3,
                lag21ng_m3, lag22ng_m3, lag23ng_m3, lag24ng_m3, lag25ng_m3,
                lag26ng_m3, lag27ng_m3, lag28ng_m3, lag29ng_m3, lag30ng_m3,
                lag31ng_m3, lag32ng_m3, lag33ng_m3, lag34ng_m3,
                lag35ng_m3);

end;

run;
```

Since a cycle consisted of 36 records with seven variables (6 measurements and the phase) being maintained, a full set of measurements required 252 LAG function calls, thus expanding the dataset from only 10 variables to 262 variables. When limiting the analysis to only one measurement variable, this method still resulted in a dataset that was 14 times larger than the original. Keeping track of all the results from the LAG function calls and dealing with incomplete cycles was difficult using this method. In addition, the LAG function has known complications (Dunn & Chung 2005, Yunchao 2009). Decisions concerning whether the cycle was complete was based solely on the values of the Phase variable for each of its lagged value. There were instances when the instrument was offline for hours and just looking for the correct pattern in the Phase variable failed to identify a problem. Since there were known problems with this solution, and it would require maintaining even more variables across the records, this method was not fully debugged nor made operational.

CYCLE IDENTIFICATION

Unlike the Linear Approach, SQL and Sequential Processing require the identification of the different cycles. It is the primary key for matching different elements together in the SQL statements. The code used to locate the cycles and generate a Cycle_ID was the same for both approaches.

To aid in locating special values, each phase was given a unique value. In the original dataset, all cleaning events are coded with Phase = 1. There are two separate cleaning phases, and different members of these phases are needed for the calculations (e.g., last cleaning value before Test A and first cleaning value after Test B). To simplify locating these records, the Phase for the cleaning events that occur after Test B were recoded to “4”.

In addition to making it easier to locate the special values, the recoding of the Phase made it easier to find the end of a cycle. Although the cycles were designed to last two hours and the data were to be collected in defined increments, there were numerous instances where the situation deviated from the design. Because of this inconsistency, the number of records cannot be used to define a cycle, the changing of the Phase value was used to identify the cycles instead. With the redefinition of the Phase variable, this identified that a new cycle started when the current value was 0 and the previous value had been 4. The RETAIN statement was used to maintain the value of the Phase. For this procedure, the LAG function could have been used. However, since a RETAIN was necessary for the enumeration of the cycles and counting the number of records in each cycle, having additional variables retained did not present any hardship.

This is the DATA step for identification of the cycles:

```
data stepping;
    set in_file;
    retain Cycle_ID cycle_line 0 lastphase currentphase . ;

    ** recoding the phase = 1 that follow a phase = 3 to 4.  This will aid
    in later calculations**;

    if lastphase = 3 & phase = 1 then phase = 4;
    else if lastphase = 4 & phase = 1 then phase = 4;

    if startoffile then do;
          Cycle_ID = Cycle_ID+1;
          lastphase = .;
          cycle_line = 0;
    end;

    else
    if lastphase = 4 & phase = 0 then do;
          Cycle_ID+1;
          lastphase=.;
          end;

    **Failed to complete cycles ;
     else
     if phase < lastphase then do;
          Cycle_ID+1;
          lastphase=.;
          end;

    else  lastphase = currentphase;

    cycle_line + ;

    currentphase = phase;

run;
```

The variable CURRENTPHASE is a holder variable used to carry the value of one record to the next record. The variable STARTOFFILE indicates whether a new data log has started due to regular servicing of the instrument.

SQL CARTESIAN JOINS APPROACH

Since data that were key to the calculations of the summarization values were on separate records, the second approach investigated used SQL joins. SQL allows multiple datasets to be combined on different keys. There are also ways to perform many-to-many merges which are problematic with the traditional SAS® DATA step.

Each of the phases of the cycle was processed in separate SQL code. This was more of a convenience for de-bugging than a requirement of the language.

For the Ambient Measurements, the data summary functions were used to calculate the sum, number of non-missing observations, minimum, and maximum value of the measurements for each cycle. In order to eliminate invalid or interrupted cycles, the starting date and time and the ending date and time were also determined. Finally, the record number of the first Ambient Measurement is determined so it can be

subtracted from the total. It is easier to subtract the first measurement from the total, than to determine the total of all values excluding that first measurement.

The summation functions are used with a GROUP statement creating one record for each of the cycles:

```
** phase = 0 obs;

proc sql;
    create table temp0 as
    select Cycle_ID, sum(ng_m3) as SumAmb, n(ng_m3) as NAmb, min(datetime)
       as starttime,  max(datetime) as endtime0, min(ng_m3) as minAmb,
       max(ng_m3) as MaxAmb, min(num) as first
    from stepping
    where phase = 0
    group Cycle_ID;
quit;

proc sql;
    create table Amb as
    select b.*, a.ng_m3 as Amb_1
    from temp0 as b, stepping as a
    where a.Cycle_ID=b.Cycle_ID & b.first = a.num
    order by Cycle_ID;
quit;
```

The ORDER BY statement sorts the output dataset.

The SQL statements for Test A and B are identical except for the record of interest to be used for later matching. For Test A, the zero value prior to the start of the testing is used to calibrate the results so the first record identification with phase equal to 2 was needed, whereas, Test B used the zero value after to completion of the test thus the last record identification was needed.

The code for phase = 2 was:

```
proc sql;
    create table TestA as
    select Cycle_ID, sum(ng_m3) as SumTestA, n(ng_m3) as NTestA, min(num)
       as first2
    from stepping
    where phase = 2
    group by Cycle_ID;
quit;
```

The code for phase = 3 was:

```
proc sql;
    create table TestB as
     select Cycle_ID, sum(ng_m3) as SumTestB, n(ng_m3) as NTestB, max(num)
       as last3
    from stepping
    where phase = 3
    group by Cycle_ID;
quit;

proc sql;
    create table cycles as
    select distinct a.Cycle_ID, max(a.datetime) as endtime
    from stepping as a
```

```
    group by a.Cycle_ID;
quit;
```

All special values for each cycle were located and output to a single row. These special values included the zero reading of the instrument as well as the initial Test A results. Each of the special values is identified with a separate sub-query. The results of the sub-queries were joined together, effectively creating a three-table join.

The SQL code for the Cleaning Phase is:

```
proc sql;
    create table Zero as
    select a2.Cycle_ID, a2.ng_m3 as prior1, a3.ng_m3 as post1, a4.ng_m3 as
          TestA_1
    from (select a.Cycle_ID, a.ng_m3
          from stepping (where= (phase=1)) as a, TestA as b
          where a.Cycle_ID=b.Cycle_ID & a.num= b.first2-1) as a2,
          (select a.Cycle_ID, a.ng_m3
          from stepping (where= (phase=4)) as a, TestB as b
          where a.Cycle_ID=b.Cycle_ID & a.num= b.last3+1) as a3,
          (select a.Cycle_ID, a.ng_m3
          from stepping (where= (phase=2)) as a, TestA as b
          where a.Cycle_ID=b.Cycle_ID & a.num= b.first2) as a4
    where a2.Cycle_ID = a3.Cycle_ID & a3.Cycle_ID=a4.Cycle_ID ;
quit;
```

The final calculations of the composite variables were done in a DATA step. The output records are limited to those having reasonable elapsed times and number of ambient measurements as well as having the components necessary for the calculation of all of the composite variables.

DATA step for final calculations is:

```
data &outfile;
    merge Amb
           Zero (in=inref)
          TestB
          TestA
          cycles;
    by Cycle_ID;

    if missing(NAmb) |  missing(NTestA) |  missing(NTestB) | not(inref)
    then delete;

    if NAmb < 14  | NAmb > 30 then delete;

    if not(missing(Amb_1)) then AverageAmb = (SumAmb -Amb_1) / (NAmb-1);
    else AverageAmb = (SumAmb) / (NAmb);

    ElapseTime0 = round((endtime0-starttime+5*60)/60/60,.01);
    ElapseTime  = round((endtime -starttime+5*60)/60/60,.01);

    if 1.5 < ElapseTime0 < 2.25;

    format endtime0 endtime starttime datetime15.;

    TestA = (SumTestA -TestA_1) - ((NTestA-1)*prior1);
    TestA_2 = SumTestA - (NTestA*prior1);
```

```
    TestB = SumTestB - NTestB*post1;


run;
```

A subset of the data was used and, a solution was successfully coded and debugged using SQL. However, once the solution was applied to the entire dataset, the program was bogged down. In Cartesian joins, all combinations are generated prior to filtering of the results. This created a dataset too large for the computing platform being used and resulted in 'insufficient' memory errors.

Although, it is possible to subdivide the data, process each portion, and then assemble the results, this lacked elegance, and a different solution was sought.

SEQUENTIAL PROCESSING

The third solution to the problem was to use a SAS® DATA step to 'walk through' the data, identifying the special cases and retaining the values. PROC MEANS was used to summarize the data and output the results for each cycle phase. Then the special values and the summary values were combined using another DATA step and the final calculations were made.

The code for identification of the cycles for Sequential Processing and is described in a previous section. Once the Cycle_ID was defined for all records, then the location of special records for each Phase was possible. The data were sorted by Cycle_ID, Phase (here again the recoding was beneficial), and datetime.

The special records needed for calculations were (1) the first sample of a cycle, (2) the first sample of Test A, (3) the cleaning value prior to the start of Test A, and (4) the cleaning value following Test B. By using the RETAIN statement, the values of special records were saved and output to a dataset with single row per cycle.

The DATA step for locating the special records is:

```
 data cycles (keep = Cycle_ID cycleflag Amb_1 TestA_1 prior1 post1
         starttime endtime0 endtime)
       stepping (drop = Amb_1 TestA_1 prior1 post1 starttime endtime0
          endtime);
    set stepping;
    by Cycle_ID phase datetime;

    format starttime   endtime0     endtime datetime15.;

    retain cycleflag 0 Amb_1 TestA_1 prior1 post1 starttime endtime0
endtime .;

** resetting retained values for new cycle;

    if first.Cycle_ID then do;
          cycleflag = 0;
          Amb_1     = .;
          TestA_1   = .;
          prior1    = .;
          post1     = .;
          starttime = .;
          endtime0  = .;
          endtime   = .;
    end;
```

```
    if first.Cycle_ID & phase not= 0 then cycleflag = 1;

    else if first.Cycle_ID then do;
          Amb_1 = ng_m3;
          starttime = datetime;
    end;

    if last.phase & phase = 0 then endtime0 = datetime;

    if first.phase & phase = 2 then TestA_1 = ng_m3;

    if last.phase & phase = 1 then prior1 = ng_m3;

    if first.phase & phase = 4 then post1 = ng_m3;

    if last.Cycle_ID then do;
          endtime = datetime;
          if missing(starttime) then cycleflag = 2;
          else if missing(endtime0) then cycleflag = 3;
          else if (endtime0 - starttime)> 8100 then cycleflag = 4;
          else if (endtime0 - starttime) < 6300 then cycleflag = 5;
          lapsedtime0 = endtime0-starttime;
          lapsedtime = endtime -starttime;
          output cycles;
    end;

    ** Limiting the records that will be used in the PROC MEANS;
    if cycleflag = 0 & phase in (0,2,3) then output stepping;

run;
```

The MEANS procedure was used to calculate the mean, sum, and N of the measurements for each Cycle_ID and Phase. The output statement allows for the creation of a dataset with the requested statistical results. Using an NWAY option limits the output to records with the highest level of the grouping (Welbrock, 2000) therefore containing one record for each combination of Cycle_ID and Phase.

The code for the PROC MEANS is:

```
proc means data=stepping nway noprint;
    by Cycle_ID phase;
    var ng_m3;
    output out=stats n=n mean=mean min=min max=max sum =sum;
run;
```

A final dataset merged the results for the different phases and the special values dataset by Cycle_ID. With all the necessary elements for the equations on one row, calculations were easily performed using this approach. In addition, each cycle was tested for completeness and integrity prior to the calculations, thus improving the run time and processing efficiency.

DATA step for final calculations is:

```
data &outfile (drop = phase _TYPE_ _FREQ_ nAmb     meanAmb minAmb   maxAmb
      sumAmb   nTestA     meanTestA minTestA   maxTestA sumTestA   nTestB
      meanTestB minTestB maxTestB   sumTestB   Amb_1    TestA_1    prior1
      post1);
```

```
    merge stats (where = (phase = 0) rename = (n=nAmb mean=meanAmb min=minAmb
              max=maxAmb sum =sumAmb))
         stats (where = (phase = 2) rename = (n=nTestA mean=meanTestA
              min=minTestA max=maxTestA sum =sumTestA))
         stats (where = (phase = 3) rename = (n=nTestB mean=meanTestB
              min=minTestB max=maxTestB sum =sumTestB))
         cycles (where= (cycleflag = 0) in = keepit);
     by Cycle_ID;

     ** Only cycles that without errors;
     if keepit;

     ** keeping only those cycles with all 3 phases;

     if missing(NAmb)| missing(NTestA)| missing(NTestB) then delete;

     if NAmb < 14 | NAmb > 30 then delete;

     if not(missing(Amb_1)) then AverageAmb = (SumAmb -Amb_1) / (NAmb-1);
     else AverageAmb = (SumAmb) / (NAmb);


     TestA = (SumTestA -TestA_1) - ((NTestA-1)*prior1);
     TestA_2 = SumTestA - (NTestA*prior1);
     TestB = SumTestB - NTestB*post1;

     if missing(TestA) | missing(TestB) then delete;

     ElapseTime0 = round((endtime0-starttime+5*60)/60/60,.01);
     ElapseTime  = round((endtime -starttime+5*60)/60/60,.01);
     format endtime starttime datetime15.;

run;
```

The STATS dataset is read in multiple times with a WHERE statement controlled which records are read in and renaming the summation variables to indicate the phase.

CONCLUSION

Three different approaches were used to process and calculate summary variables from a complicated dataset. The dataset consisted of various data collection phases with all the results output to a single variable. The quality of the data were dependent on instrument performance as well as the calibration values that were collected both previously to and after the measurements. The methods included using the Linear Approach where a LAG function was applied multiple times, joining the data to itself using SQL Approach, and DATA step processing using a Sequential Approach utilizing the RETAIN statement.

The Linear Approach solution was producing erroneous results and was not time efficient. The Linear Approach using the LAG function expanded the dataset without removing the unnecessary records. With more than 800,000 records, this method was cumbersome and problematic, (i.e. resource hog). This coupled with the complexity of debugging the program caused the Linear Approach using the LAG function to be quickly abandoned after a short investigation.

The second solution was to use PROC SQL to summarize the data, locate special records, merge the data, and perform final calculations. Although all the required data processing could have been done with one SQL call, each Phase was processed within its own statements to improve clarity and simplify debugging. Using a two-year subset of the data, this method was fully developed and debugged. Once

applied to the entire nine years of data, however, the program quickly ran out of resources and failed to complete. Investigation into the cause of the failure uncovered that SQL was performing a full Cartesian join on all data and only afterwards limiting the output to the desired records based on the WHERE condition. The size of the Cartesian join was too large for the limited computer power that was available.

The final solution was to perform basic DATA step processing using a Sequential Approach. Using such features as RETAIN and automatically created FIRST. and LAST. variables, it was only necessary to 'walk through' the entire dataset twice. Once the data were summarized using PROC MEANS the number of records that were necessary to process was greatly reduced. This method had no difficulty with memory resources, and was easy to debug and verify.

Using a better computing platform, the results from the second method obtained and compared to those from the third method. All the values were identical. A comparison between the third method and the first resulted in thousands of differences (45% of the values) and no attempt made to identify the issue. The processing time for the first method, limited to only one measurement value, was approximately one minute, whereas the time requirement for both the second and third method was 1.7 and 1.5 seconds, respectively. The time requirement for the second method increased exponentially when additional measurement values were analyzed, while the time requirement for the third method remained level.

SAS® provides multiple tools to accomplish any given task. Some tools are more elegant than others, while other tools only work well in certain situations. There are many on-line resources to investigate alternative solutions to problems including: SAS® on-line documentation (http://support.sas.com/documentation/), SAS® on-line communities (http://support.sas.com/documentation/), and local SAS® user groups. It is important to remember that when the current tool is not working you can explore other tools and other paths to reach your goal.

REFERENCES

Bereznicki, Sarah D.; Sobus, Jon R.; Vette, Alan F.; et al. 2012. "Assessing spatial and temporal variability of VOCs and PM-components in outdoor air during the Detroit Exposure and Aerosol Research Study (DEARS)", *Atmospheric Environment*, 61: 159-168.

Dunn, Toby and Chang Y. Chung (2005). Retaining, lagging, leading, and interleaving Data. In Proceedings of the Pharmaceutical SAS® Users Group Conference Paper TU09.

Ferriola, Frank, (2002) 'What's Your _TYPE_? How to find the CLASS You Want in PROC SUMMARY' Available at http://www2.sas.com/proceedings/sugi27/p077-27.pdf.

Kimbrough, S., Baldauf, R., Hagler, G., Shores, R.C., Mitchell, W., Whitaker, D.A., Croghan, C.W. and Vallero, D.A., (2013). "Long-term continuous measurement of near-road air pollution in Las Vegas: Seasonal variability in traffic emissions impact on local air quality." *Air Quality, Atmosphere & Health* 6: 295-305.

US EPA Monitoring Network Information, Accessed: March 24, 2015, Available at http://www.epa.gov/ttn/amtic/qapollutant.html.

Welbrock, Peter R. (2000) "PROC MEANS: More than just your average procedure" Available at http://www.ats.ucla.edu/stat/sas/library/nesug00/p068.pdf.

Yunchao, Tian (2009). LAG – the Very Powerful and Easily Misused SAS ® Function. Available at http://support.sas.com/resources/papers/proceedings09/055-2009.pdf.

CONTACT INFORMATION

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

Carry Croghan
US-EPA
croghan.carry@epa.gov

DISCLAIMER

Although this work was reviewed by the US-EPA and approved for publication, it may not necessarily reflect official Agency policy. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.