Paper 1069 - 2017

Know Your Tools Before You Use

Justin Jia, TransUnion Canada, Burlington, Ontario, Canada

Jenny Jia, University of Western Ontario, London, Ontario, Canada

ABSTRACT

When analyzing data with SAS®, we often use SAS DATA step and PROC SQL to explore and manipulate data. Though they both are useful tools in SAS, however, many SAS users do not fully understand their differences, advantages and disadvantages and thus have unnecessary biased debates on them. Therefore, this paper is aimed to illustrate and discuss these aspects with real work examples, which will give SAS users deep insights in utilizing them. Using the right tool for a given circumstance not only provides an easier and more convenient solution, it will also save time and work in programming and thus improve work efficiency. Furthermore, the illustrated methods and advanced programming skills can be used in a wide variety of data analysis and business analytics fields.

INTRODUCTION

In data analysis and business analytics with SAS programming, we often need to perform a lot of data exploration and manipulations, such as sort, dedupe, subset, append, merge, join, reshape and summarize data etc. To accomplish these tasks, DATA step and Proc SQL (or native SQL queries via SAS pass-through facility) are the frequently used tools, each one has its strengths and drawbacks. Despite that SQL was not originated from SAS, it is a very useful and necessary part of SAS now after Proc SQL was developed to support the use of SQL language in SAS.

However, from my work experiences, I heard about many debates and arguments from colleagues on SQL and DATA Step, often during lunch time or internal learning sessions. Some people, who gained SQL experience before using SAS, prefer to use Proc SQL in SAS, and claim that SQL queries are much better than DATA Step to use in SAS programming. On the contrary, other people, who started with SAS first, may feel that SQL is much harder than SAS DATA step. They argue that DATA step is more straightforward, flexible and convenient than Proc SQL to use. Sometimes the debate even goes quite fierce.

As a fair comment on these debates, I would like to say, they are both wrong! Frankly speaking, these people debate because they are superficial in using both SQL and DATA step. They don't fully understand and grasp the pros and cons of SAS Data step and Proc SQL, which lead to their biased debates. If they could know their tools better, they would understand that: there is no best universal tool at all, but there exists the most suitable tool for a specific circumstance.

Therefore, this paper will use real work examples to show the distinctive differences between DATA step and SQL queries. We will present and discuss their advantages and disadvantages under different application circumstances. The illustrated methods and advanced programming skills can have important applications in a wide variety of analytic work fields.

THE FUNFAMENTAL DIFFERENCE BETWEEN DATA STEP AND SQL QUERIES

Before we illustrate with examples, we want to call the reader's attention to the fundamental difference between DATA step and SQL queries. First, DATA Step is an iterative process and it only processes one record in each iteration. Second, DATA step processes data in two phases--- a compilation phase and an execution phase. In the compilation phase, SAS scans and checks syntax to make sure it is error-free. After it is successfully compiled, the execution phase begins and the DATA step processes the input data line by line.

On the contrary, Proc SQL or other SAS procedures are NOT iterative in nature, they process input data all together. However, it is important to note that the clauses in a SQL query do have sequences during

execution. The syntax in DATA step is executed from top to bottom sequentially, but in a SQL query, the clauses are not executed in their apparent order. Below gives the order of operations in a SQL query:

FROM clause

WHERE clause

GROUP BY clause

HAVING clause

SELECT clause

ORDER BY clause

Because the SELECT clause is executed after WHERE clause, therefore when using PROC SQL, we must use the CALCULATED keyword if we want to use the results of an expression in the WHERE clause or in the same SELECT clause. Please note that it is valid only when used to refer to columns that are calculated in the immediate query expression.

A good understanding of the above order of operations of SQL clauses will be very helpful for the sophisticated use of PROC SQL or native SQL queries.

CASE I: CREATE A CONSECUTIVE ROW NUMBER FOR EACH RECORD

In data analysis, sometimes we need to assign a consecutive unique number to each row for further data manipulations. For example, such a unique sequence ID allows us to join back of an output dataset to the input data set if the input data does not have a unique key. It can also enable us to subset data or to create polished reports by using traffic lighting technique. In the following case studies, we will show its important practical uses, such as selecting top records from each group, calculating moving statistics etc.

For example, we have below sample Class data as shown in Table 1, which contains 25 observations and three columns: Student_ID, Class and Grade. Our aim is to create a consecutive sequence number for each record.

As everyone knows, it is super easy to accomplish it by DATA step due to the iterative nature of DATA step. We can utilize the automatic variable _N_, or the SUM statement to create the sequence number column Seq_ID. The SAS code is given below. It is worth to note that other automatic variables created during DATA step execution, such as END, NOBS, IN contributor etc., also provide great convenience to data manipulations. We can utilize them for a wide variety of applications.

If we use SQL to create such a sequence number, it will be a tough job because SQL is not an iterative process. Actually, the only way to achieve it in SAS is to use the MONOTONIC function. Please note that this function is undocumented by SAS Institute and you use it at your own risk. In native SQL language such as Oracle or DB2, we can use the ROW_NUMBER() OVER() function to do it. Please refer to the related SQL documents for details if interested.

```
********Case I: Create consecutive sequence ID for each record. ********;
proc sort data=A.Class;
by Class Student_ID;
run;

data Sequence;
set A.Class;
by Class Student_ID;

Seq_ID=_N_;
/*Seq_ID + 1;*/
run;
```

```
*****************************SQL method. ********************************;
proc sql;
create table Sorted as
select *
from  A.Class
order by Class, Student_ID;

create table Sequence as
select *,
       Monotonic() as Seq_ID
from Sorted;
quit;
```

Table 1. Contents of Class data. **Column to be created.**

Student_ID	Class	Grade		Seq_ID
A119	A	83		1
A199	A	83		2
A219	A	83		3
A220	A	92		4
A277	A	83		5
A356	A	75		6
A392	B	58		7
A400	B	77		8
A404	B	90		9
A418	C	99		10
A475	C	43		11
A508	D	78		12
A512	D	51		13
A521	D	66		14
A523	D	95		15
A549	D	19		16
A660	E	33		17
A701	E	58		18
A709	E	77		19
A762	E	77		20
A774	F	94		21
A951	F	89		22
A961	F	76		23
A963	F	76		24
A981	F	76		25

CASE II: SELECT TOP RECORDS FROM EACH GROUP

In our daily work, we frequently need to select several top records from each group from a big database. For example, in the above Class data, we want to select the top 3 grades from each class. How can we do it?

It is simple and easy if we use SAS DATA step. As shown below, we first sort the data by Class and descending Grade, then we create a counter variable CNT by SUM statement in the following DATA step. We utilize this counter variable to easily select and output the top 3 records from each class. The code works well with tier values in data. Table 2 shows the output.

```
**************Case II: select top records for each group. ***************;
proc sort data=A.Class;
by Class descending Grade;
run;

data Top_1;
set A.Class;
by Class descending Grade;

if first.Class then CNT=0;
CNT+1;

if CNT <=3;
drop Student_ID;
run;
```

Table 2. Printout of Top_1 data set (correct result)

Class	Grade	CNT
A	92	1
A	83	2
A	83	3
B	90	1
B	77	2
B	58	3
C	99	1
C	43	2
D	95	1
D	78	2
D	66	3
E	77	1
E	77	2
E	58	3
F	94	1
F	89	2
F	76	3

Table 3. Printout of Top_2 data set (incorrect)

Class	Grade
A	92
B	90
B	77
B	58
C	99
C	43
D	95
D	78
D	66
E	77
E	77
E	58
F	94
F	89

On the contrary, if we use Proc SQL only to do it, it is difficult and challenging and requires much more code. At some online SAS communities, some people gave below SQL code by using self-join or correlated sub-query. It is important to point out that these codes are error-prone because they fail to work if the grades have tier values. Table 3 shows the results produced by these codes, you can see it does NOT work properly for Class A and Class F due to the tier values in data.

```
******************** SQL Method: Error-prone SQL code. ******************;
proc sql;
create table Top_2 as
select Class, Grade
from A.Class  a
where (select count(*) from A.Class b where b.Class=a.Class
       and b.Grade >=a.Grade) <=3;
quit;

proc sql;
create table Top_2 as
select a.Class, a.Grade
from  A.Class  a, A.Class b
```

```
where a.Class=b.Class and b.Grade >=a.Grade
group by a.Class, a.Grade
having count(*) <=3
order by a.Class, a.Grade desc;
quit;

************************Correct SQL method. ****************************;

proc sql;
create table Sorted as
select *
from A.Class
order by Class, Grade desc ;

create table Sequence as
select *,   Monotonic() as Seq_ID
from Sorted;

create table Top_3 as
select  a.Class, a.Seq_ID, a.Grade
from    Sequence  a, Sequence  b
where   a.Class=b.Class and b.Seq_ID <=a.Seq_ID
group by a.Class, a.Seq_ID, a.Grade
having count(*) <=3
order by a.Class, a.Seq_ID ;
quit;
```

So, what is the correct solution? In this case, we have to create a consecutive sequence number and then use it to select the top records. As shown above, we first sort the data and create a unique sequence ID Seq_ID by using MONOTONIC function. Then we join the Sequence table with itself and subset the results by using the WHERE, GROUP BY and HAVING clauses. This code works with tier values and gives results identical to Table 2. Obviously, this SQL method requires advanced SQL skills.

In respect to this work example, we can see that DATA step is a much better tool than SQL because of its iterative nature and flexibility. So DATA step beats SQL in this round!

CASE III: CREATE A CROSS-JOIN CARTESIAN PRODUCT

Under some circumstances, we need to cross join two data sets and then perform further calculations based on the cross-join product. If we use PROC SQL or native SQL to do it, it is just a piece of cake because a cross join Cartesian product is the default product of SQL join. However, if we want to create it with DATA step, it is not easy at all.

For example, in mathematics, suppose we have 5 data points A, B, C, D, E in a plane, we want to calculate the distance between any two different points. To perform the calculations, a straightforward solution is to create a cross-join Cartesian product by self-joining the input data. It is a simple job by using Proc SQL or native SQL, we give the SQL code below and show the results in Table 4.

```
*****************Case III: create a cross-join product. *****************;
data Points;
input Point $1 X Y   @@;
cards;
A 1 3     B -5 2      C 9 15      D 6 -1      E 12 4
;
run;
```

```
*********************************SQL method.*****************************;
proc sql;
create table Distance as
select a.Point        as Point1,
       a.X            as X1,
       a.Y            as Y1,
       b.Point        as Point2,
       b.X            as X2,
       b.Y            as Y2,

(case when b.Point=a.Point then ' ' else '|'||(a.Point)||(b.Point)||'|'
end)  as From_To,

(case when b.Point=a.Point then .   else SQRT((b.X-a.X)**2+(b.Y-a.Y)**2)
end)  as Distance format=comma6.2
from Points a, Points b;
quit;

*************************DATA step Method I. ***************************;
data Cartesian;
set Points(rename=(Point=Point1 X=X1 Y=Y1));

do I=1 to M;
set Points(rename=(Point=Point2 X=X2 Y=Y2))  NOBS= M end=EOF point=I;
output;
if EOF=1 then stop;
end;
run;

*************************DATA step Method II. ***************************;
data Cartesian;
set Points End=EOF;

array _Point_(5)      $1    P1-P5;
array _X_(5)    X1-X5;
array _Y_(5)    Y1-Y5;

retain P1-P5 X1-X5 Y1-Y5;

_Point_(_N_)=Point;
_X_(_N_)=X;
_Y_(_N_)=Y;

if EOF=1 then do I=1 to 5;
Point1=_Point_(I);
Temp_X1=_X_(I);
Temp_Y1=_Y_(I);

do J=1 to 5;
Point2=_Point_(J);
Temp_X2=_X_(J);
Temp_Y2=_Y_(J);
output;
end;
end;

keep Point1 Point2 Temp_X1 Temp_X2 Temp_Y1 Temp_Y2;
```

```
rename Temp_X1=X1  Temp_X2=X2 Temp_Y1=Y1  Temp_Y2=Y2;
run;

data Distance;
set Cartesian;
if Point2 ^=Point1 then do;
From_To='|'||(Point1)||(Point2)||'|';
Distance=SQRT( (X2-X1)**2+(Y2-Y1)**2);
end;

else do;
From_To=' ';
Distance=.;
end;

format Distance comma12.2;
run;
```

Table 4. Printout of Distance data set

Point1	X1	Y1	Point2	X2	Y2	From_To	Distance
A	1	3	A	1	3		.
A	1	3	B	-5	2	|AB|	6.08
A	1	3	C	9	15	|AC|	14.42
A	1	3	D	6	-1	|AD|	6.4
A	1	3	E	12	4	|AE|	11.05
B	-5	2	A	1	3	|BA|	6.08
B	-5	2	B	-5	2		.
B	-5	2	C	9	15	|BC|	19.1
B	-5	2	D	6	-1	|BD|	11.4
B	-5	2	E	12	4	|BE|	17.12
C	9	15	A	1	3	|CA|	14.42
C	9	15	B	-5	2	|CB|	19.1
C	9	15	C	9	15		.
C	9	15	D	6	-1	|CD|	16.28
C	9	15	E	12	4	|CE|	11.4
D	6	-1	A	1	3	|DA|	6.4
D	6	-1	B	-5	2	|DB|	11.4
D	6	-1	C	9	15	|DC|	16.28
D	6	-1	D	6	-1		.
D	6	-1	E	12	4	|DE|	7.81
E	12	4	A	1	3	|EA|	11.05
E	12	4	B	-5	2	|EB|	17.12
E	12	4	C	9	15	|EC|	11.4
E	12	4	D	6	-1	|ED|	7.81
E	12	4	E	12	4		.

However, if we use DATA step to do it, it is challenging and requires advanced skills. As shown above, the first method is to use multiple SET statements along with the POINT= Variable option. This option specifies a temporary variable whose numeric value determines which observation is read. POINT= causes the SET statement to use random (direct) access to read a SAS data set rather than sequential access[1]. It is important to note that we must include a STOP statement to stop DATA step processing when we use the POINT= option, otherwise continuous loops may take place.

Another method is to use ARRAY and DO-LOOP. We first use ARRAY along with RETAIN statement to transpose the input data from vertical to horizontal. At the end-of-file, we use the DO-LOOP to create the

cross-join product and rotate it back to vertical format. These two DATA step methods produce a cross join Cartesian product exactly same as the SQL method. After creating the Cartesian product, it is easy to compute the distance between any two points. The generated results are identical to those shown in Table 4.

Please note that a cross-join product may have other important applications in data analysis. For example, we can utilize it to calculate the distance between two places by using GPS coordinates, to calculate the relative altitude difference between any two cities. In business analytics, we can use it to compute the frequency, stacking and clustering characteristics of customer purchases. It is a very useful programming skill in data analysis actually.

CASE IV: CALCULATE THE MONTH OVER MONTH CHANGES

In business analytics, we often need to compute and report the month over month change of data for market trends study or business intelligence purposes. This kind of changes is usually called the MOM Variance. For example, in financial industry, we often report the MOM variances of new, existing and total clients, total balances and managed funds etc.

In the below example, the Sales data set contains monthly sales for the 12 months of 2001. We want to calculate the month over month variances.

```
*************************Case IV: MOM Variance.*************************;
data Sales;
input Year_Month Sales dollar7.0   @@;
datalines;
200101  $1000 200102  $1200 200103  $1800 200104  $2000
200105  $2300 200106  $2400 200107  $2380 200108  $2500
200109  $2700 200110  $2900 200111  $3000 200112  $3200
;
run;

***********************Method I: use LAG/DIF functions. **************;
data MOM;
set Sales;
retain N Year_Month Sales;

MOM=Sales-Lag(Sales);
MOM_PCT=MOM/lag(Sales);

/*MOM=DIF(Sales);*/
/*MOM_PCT=MOM/(Sales-MOM);*/

format MOM dollar6.0 MOM_PCT percentn9.2;
run;

***********************Method II:  use RETAIN statement. **************;
data MOM;
set Sales;
retain Last_MTH_Sales;

if _N_=1 then do;
MOM=.;
MOM_PCT=.;
output;
end;

else do;
MOM=Sales-Last_MTH_Sales;
```

```
MOM_PCT=MOM/Last_MTH_Sales;
output;
end;

Last_MTH_Sales=Sales;

format MOM dollar6.0 MOM_PCT percentn9.2;
run;

********************Method III:  use multiple SET statements**************;
data Mom;
set Sales;

if _N_ > 1 then set Sales(keep=Sales rename=(Sales=Last_MTH_Sales));

if _N_=1 then do;
MOM=. ;
MOM_PCT=.;
end;

else do;
MOM = Sales- Last_MTH_Sales;
MOM_PCT= MOM/Last_MTH_Sales;
end;

format MOM dollar6.0 MOM_PCT percentn9.2;
run;

**********************Method IV: use data step merge**************;
data Prepare;
set  Sales;
N=_N_;
M=_N_ + 1;
run;

data MOM;
merge Prepare(in=A)
      Prepare(in=B keep=M Sales rename=(M=N Sales=Last_MTH_Sales)) ;
by N;

if A=1;
MOM=Sales-Last_MTH_Sales;
MOM_PCT=MOM/Last_MTH_Sales;

format MOM dollar6.0 MOM_PCT percentn9.2;
drop M N;
run;

********************Method V: SQL query method ******************;
proc sql;
create table MOM as
select a.*,
       b.Sales                    as Last_MTH_Sales,
       a.Sales-b.Sales            as MOM format=dollar6.0,
       calculated MOM/b.Sales     as MOM_PCT format=percentn9.2

from Sales a left join Sales b
```

```
on a.Year_Month=b.Year_Month+1;
quit;
```

First, let's use DATA step to do it. As mentioned above, DATA step is an iterative process and it processes only one observation in each iteration. Therefore we have got a challenge: we only have one observation in each loop, how can we compute the difference of Sales between current observation and previous observation?

We have several methods to beat the challenge. As an instinct, some people may think to reshape data from vertical to horizontal by using Proc Transpose or DATA step ARRAY, then calculate the MOM changes and rotate it back to vertical layout. Though it works, but it is really a clumsy method, especially when the input data is big. For this reason, we do not give the SAS code and we do not recommend to use it.

As shown above, the first method we give is to use LAG or DIF function, which can returns values from a queue[2]. However, the drawback of using LAG/DIF function is that it is very tricky and not easy to control the process, especially when we have complicated conditional processing in the program. The second method is to use RETAIN statement to realize it. RETAIN can prevent a variable that is created by an INPUT or assignment statement from initialization and retain its value from one iteration of the DATA step to the next[3]. Therefore, as shown in Method II, the variable Last_MTH_Sales is used to retain the value of Sales from previous iteration. Then the MOM Sales changes are computed and the record is written out to the output data set by the explicit OUTPUT statement. After output, we replace the value of Last_MTH_Sales with the value of Sales in the current observation, and the new value is carried forward to the next iteration. In this way, the MOM sales change is computed and Table 5 displays the printout of the generated data set. Similarly, a simpler but better alternative method is to use multiple SET statements as shown in the Method III. We use the IF-THEN conditional processing to SET the same dataset Sales starting from the second iteration and rename Sales as Last_MTH_Sales. In this way, the Last_MTH_Sales variable actually carries on the value of Sales of previous month to current observation so that we can compute the MOM sales changes. This tricky method produces the same results as shown in Table 5, but it is more concise in SAS coding.

Table 5. Printout of Created MOM data set.

Year_Month	Sales	Last_MTH_Sales	MOM	MOM_PCT
200101	1000	.	.	.
200102	1200	1000	$200	20.00%
200103	1800	1200	$600	50.00%
200104	2000	1800	$200	11.11%
200105	2300	2000	$300	15.00%
200106	2400	2300	$100	4.35%
200107	2380	2400	($20)	-0.83%
200108	2500	2380	$120	5.04%
200109	2700	2500	$200	8.00%
200110	2900	2700	$200	7.41%
200111	3000	2900	$100	3.45%
200112	3200	3000	$200	6.67%

The other approach is to self-join the Sales data via DATA step merge or SQL join. We demonstrate these two methods in Method IV and V, and they produce identical results as shown in Table 5.

As illustrated above, in this case, both DATA step and SQL query provide equally effective solutions for us. So they get equal scores in this round of boxing.

CASE V: COMPUTE ROLLING AND MOVING STATISTICS

In analysis of financial data (like stock prices, returns or trading volumes) and macroeconomic time series data, moving statistics are often calculated and used. For example, a moving average is commonly applied to smooth out short-term fluctuations and highlight longer-term trends or cycles. It is a very useful data-smoothing technique.

There are three different types of moving statistics: **simple moving statistics, central moving statistics and cumulative moving statistics**[4]. For example, in financial applications, a simple moving average (SMA) is the unweighted mean of the previous n data. However, in science and engineering the mean is normally taken from an equal number of data on either side of a central value. This is called a central moving average and it ensures that variations in the mean are aligned with the variations in the data rather than being shifted in time. Under some circumstances, we need to get the average of all of the available data up to the current datum point, this is known as a cumulative moving average or running average. For example, an investor may want the average price of all of the stock transactions for a particular stock up until the current time.

SAS does have a procedure PROC EXPAND which is developed to process time series data and it supports a wide array of data transformations[5]. Therefore we can use it to compute many kinds of moving statistics. However, it is important to note that this procedure is included in the SAS/ETS software package rather than in the base programming package. SAS/ETS is a special software package which includes a wide range of tools for analyzing time series data such as modeling, forecasting and simulation. For the users who don't have this package or don't know this procedure, it can be accomplished through base programming. We demonstrate the programming skills by using the above Sales data set.

Cumulative Moving Statistics

First, we illustrate how to compute cumulative moving statistics by using DATA step and SQL. As shown below, we first use DATA step to compute the moving statistics of Sales from January up to the current month. When using DATA step, we need to use ARRAY and RETAIN statement to carry the value of previous observations to current observation, and then use the summary functions such as SUM, MEAN, MEDIAN, STD etc to compute the moving statistics. Alternatively, though we can use SUM statement to calculate the cumulative moving sum and average, but we cannot use it to compute other moving statistics such as median, standard deviation etc. This is the drawback of using SUM statement. Our approach is therefore more versatile and generalizable. Table 6 shows the generated results.

```
**********Cumulative moving statistics. **********;
data Cumulative;
set Sales;
array _Sales_(12) S1-S12;
retain S1-S12;

_Sales_(_N_)=Sales;
MTH_Count=_N_;
MoveSum=Sum(of _Sales_(*) );
MoveMean=Mean(of _Sales_(*));
MoveSTD=STD(of _Sales_(*));

drop S1-S12;
format MoveSum MoveMean MoveSTD dollar6.0;
run;

**********SQL method. **********;
proc sql;
create table Sequence as
select *, monotonic() as  N
from Sales
order by Year_Month;
```

```
create table Cumulative as
select a.Year_Month, a.Sales, a.N,
       count(*)      as MTH_Count format=comma6.0,
       sum(b.Sales)  as MoveSum         format=dollar6.0,
       mean(b.Sales) as MoveMean  format=dollar6.0,
       std(b.Sales)  as MoveSTD   format=dollar6.0

from  Sequence a,  Sequence  b
where a.N >= b.N
group by a.Year_Month, a.Sales, a.N;
quit;

*************** SQL method by use of correlated sub-query. ***************;
proc sql;
create table moving as
select a.*,
(select sum(Sales)
from Sequence  b where b.N <=a.N) as MoveSum  format=dollar6.0,

(select mean(Sales)
from Sequence  b where b.N <=a.N) as MoveMean format=dollar6.0,

(select STD(Sales)
from Sequence  b where b.N <=a.N) as MoveSTD  format=dollar6.0

from Sequence a;
quit;
```

Alternatively, we can also use SQL to do it. Before computing the statistics, we need to create a consecutive sequence variable N. Please note this sequence number N is mandatory and critically important to accomplish our task. Then we can calculate all the moving statistics by self-joining or correlated sub-query. Self-joining method is preferred because it is more straightforward and requires less SAS code. The above SQL methods produce the exactly same results as shown in Table 6.

Table 6. Printout of created Cumulative data set.

Year_Month	Sales	MTH_Count	MoveSum	MoveMean	MoveSTD
200101	1000	1	$1,000	$1,000	.
200102	1200	2	$2,200	$1,100	$141
200103	1800	3	$4,000	$1,333	$416
200104	2000	4	$6,000	$1,500	$476
200105	2300	5	$8,300	$1,660	$546
200106	2400	6	$10,700	$1,783	$574
200107	2380	7	$13,080	$1,869	$571
200108	2500	8	$15,580	$1,948	$574
200109	2700	9	$18,280	$2,031	$592
200110	2900	10	$21,180	$2,118	$622
200111	3000	11	$24,180	$2,198	$647
200112	3200	12	$27,380	$2,282	$682

Simple Moving Statistics

Next, let's compute the simple moving statistics of sales of the previous 4 months and current month, totally 5 data points. Again, we use DATA step along with ARRAY and RETAIN statement to retain values. Table 7 shows the created DATA set.

```
************************Simple moving statistics************************;
data Simple;
set Sales;
array _Sales_(5) S1-S5;
retain S1-S5;

do I=5 to 2 by -1;
_Sales_(I)=_Sales_(I-1);
end;

_Sales_(1)=Sales;

if _N_>=5 then do;
MoveSum=Sum(of _Sales_(*));
MoveMean=Mean(of _Sales_(*));
MoveSTD=STD(of _Sales_(*));
end;

format MoveSum MoveMean MoveSTD dollar6.0;
drop I;
run;

*******************************SQL Method. ****************************;
proc sql;
create table Simple as
select a.Year_Month, a.Sales, a.N,
       count(*)      as MTH_Count,
       sum(b.Sales)  as MoveSum   format=dollar6.0,
       mean(b.Sales) as MoveMean  format=dollar6.0,
       std(b.Sales)  as MoveSTD   format=dollar6.0

from  Sequence a,   Sequence  b
where a.N >=5   and  b.N between a.N-4 and a.N
group by a.Year_Month, a.Sales, a.N;
quit;
```

Table 7. Printout of Simple data set created by data set.

Year_Month	Sales	S1	S2	S3	S4	S5	MoveSum	MoveMean	MoveSTD
200101	1000	1000	.	.	.	.	.	.	.
200102	1200	1200	1000	.	.	.	.	.	.
200103	1800	1800	1200	1000	.	.	.	.	.
200104	2000	2000	1800	1200	1000	.	.	.	.
200105	2300	2300	2000	1800	1200	1000	$8,300	$1,660	$546
200106	2400	2400	2300	2000	1800	1200	$9,700	$1,940	$477
200107	2380	2380	2400	2300	2000	1800	$10,880	$2,176	$264
200108	2500	2500	2380	2400	2300	2000	$11,580	$2,316	$190
200109	2700	2700	2500	2380	2400	2300	$12,280	$2,456	$154
200110	2900	2900	2700	2500	2380	2400	$12,880	$2,576	$221
200111	3000	3000	2900	2700	2500	2380	$13,480	$2,696	$261
200112	3200	3200	3000	2900	2700	2500	$14,300	$2,860	$270

Table 8. Printout of Simple data set created by SQL method.

Year_Month	Sales	N	MTH_Count	MoveSum	MoveMean	MoveSTD
200105	2300	5	5	$8,300	$1,660	$546
200106	2400	6	5	$9,700	$1,940	$477
200107	2380	7	5	$10,880	$2,176	$264
200108	2500	8	5	$11,580	$2,316	$190
200109	2700	9	5	$12,280	$2,456	$154
200110	2900	10	5	$12,880	$2,576	$221
200111	3000	11	5	$13,480	$2,696	$261
200112	3200	12	5	$14,300	$2,860	$270

We can also use SQL to perform our task by self-joining and through the created sequence variable N. Table 8 shows the generated results, which are identical to those in Table 7, except that it leaves out the first 4 months because of insufficient data points.

Central Moving Statistics

Finally, we demonstrate how to compute the central moving statistics of sales, i.e., the statistics of sales of current month and its prior-/post- two months. Totally, we have 5 data points too. Compared to the cumulative and simple moving statistics, it involves more challenges when we use DATA step to do it. Attributed to the iterative nature of DATA step, we not only need to retain values from previous observations, and we also need to pre-use sales values post current observation. How can we realize it?

As shown below, the creative solution is to use multiple SET statement along with the FIRSTOBS data set option. In this way, we can both retain sales values from previous months and pre-read them from post months, then we can use the summary functions to calculate the required statistics. Table 9 shows the results produced from the DATA step code.

```
************************Central moving statistics************************;
data Central;
set Sales end=EOF;

call missing(of S1-S5);

if _N_> 2 and _N_ <=10 then do;

set Sales(keep=Sales rename=(Sales=S1));
set Sales(firstobs=2 keep=Sales rename=(Sales=S2));
S3=Sales;
set Sales(firstobs=4 keep=Sales rename=(Sales=S4));
set Sales(firstobs=5 keep=Sales rename=(Sales=S5));

MoveSum=Sum(of S1-S5);
MoveMean=Mean(of S1-S5);
MoveSTD=STD(of S1-S5);
end;

format MoveSum MoveMean MoveSTD dollar6.0;
run;

*******************************SQL Method ******************************;
proc sql;
create table Central as
select a.Year_Month,  a.Sales, a.N,
       count(*)       as MTH_Count,
       sum(b.Sales)   as MoveSum  format=dollar6.0,
       mean(b.Sales)  as MoveAvg format=dollar6.0,
       std(b.Sales)   as MoveSTD format=dollar6.0
```

```
from  Sequence a,  Sequence  b
where a.N >=3 and a.N<=10 and  b.N between a.N-2 and a.N+2
group by a.Year_Month, a.Sales, a.N;
quit;
```

Table 9. Printout of Central data set created by DATA step method.

Year_Month	Sales	S1	S2	S3	S4	S5	MoveSum	MoveMean	MoveSTD
200101	1000	.	.	.	.	.	.	.	.
200102	1200	.	.	.	.	.	.	.	.
200103	1800	1000	1200	1800	2000	2300	$8,300	$1,660	$546
200104	2000	1200	1800	2000	2300	2400	$9,700	$1,940	$477
200105	2300	1800	2000	2300	2400	2380	$10,880	$2,176	$264
200106	2400	2000	2300	2400	2380	2500	$11,580	$2,316	$190
200107	2380	2300	2400	2380	2500	2700	$12,280	$2,456	$154
200108	2500	2400	2380	2500	2700	2900	$12,880	$2,576	$221
200109	2700	2380	2500	2700	2900	3000	$13,480	$2,696	$261
200110	2900	2500	2700	2900	3000	3200	$14,300	$2,860	$270
200111	3000	.	.	.	.	.	.	.	.
200112	3200	.	.	.	.	.	.	.	.

In this case, it is easier to use SQL method to achieve our goal. As shown in Table 10, the above SQL code gives the same results as DATA step, except for the exclusion of the first two months and last two months because they don't include sufficient data points for calculating central moving statistics.

Table 10. Printout of Central data set created by SQL method.

Year_Month	Sales	N	MTH_Count	MoveSum	MoveAvg	MoveSTD
200103	1800	3	5	$8,300	$1,660	$546
200104	2000	4	5	$9,700	$1,940	$477
200105	2300	5	5	$10,880	$2,176	$264
200106	2400	6	5	$11,580	$2,316	$190
200107	2380	7	5	$12,280	$2,456	$154
200108	2500	8	5	$12,880	$2,576	$221
200109	2700	9	5	$13,480	$2,696	$261
200110	2900	10	5	$14,300	$2,860	$270

Therefore based on above demonstrations, we can see that SQL method is a better tool than DATA step to calculate moving statistics. In this round of competition, SQL is the winner!

CONCLUSION

As demonstrated and discussed in this paper, DATA step and SQL queries are both useful tools in data analysis and manipulation with SAS programming. As a data analyst and SAS programmer, we need to be very clear of both their pros and cons so that we can choose the most suitable tool under a circumstance. Using the right tool will provide an easier and more convenient solution in programming, which can save both time and work and lead to improved work efficiency.

REFERENCES

[1] SAS Support Web,
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000173782.htm

[2] SAS Support Website,
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000212547.htm

[3] SAS Support Website,
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000214163.htm

[4] WIKI Website,
https://en.wikipedia.org/wiki/Moving_average

[5] SAS Support Website,
http://support.sas.com/documentation/cdl/en/etsug/63939/HTML/default/viewer.htm#expand_toc.htm

CONTACT INFORMATION

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

Justin Jia TransUnion Canada, Burlington, Ontario, Canada