Paper 029-2007

Calculating Duration via the Lagged Values of Variables

Pon Su, Health Economics Resource Center (HERC),
Department of Veterans Affairs, Menlo Park, CA

ABSTRACT

Duration calculation is common in many SAS programming tasks. Whenever an event occurs between a start date and an end date, one may need to compute the elapsed duration (in seconds, minutes, days) for the said event. The programming challenge in SAS often arises from having to perform operations across many observations for the same person. As most books and papers about SAS have not fully explored the LAG function in the context of duration calculation, the author now wishes to leverage the LAG function and offer a different programming approach.

INTRODUCTION

Real life data records generally include more than one person; hence we use more than one subject ID values (101, 102). Records are usually entered in random order; therefore, a sorting of 'ID, STARTDT, STOPDT' is warranted to ensure proper processing. The sort step is skipped in the following example because the records presented are already in the correct ascending order. The LAG function as well as the nested IF are demonstrated here.

Please note that the records with out-of-sequence dates or dates containing missing values are not discussed here. They belong in the data cleaning stage.

DATA and SAS PROGRAM

```
options center nodate nonumber ls=64;

data temp;
   input id  startdt:mmddyy8. stopdt:mmddyy8. ;
   datalines;
101 01/01/05 01/09/05
101 01/05/05 01/20/05
101 01/16/05 01/18/05
101 01/26/05 01/31/05
102 01/01/05 01/07/05
102 01/05/05 01/07/05
102 01/06/05 01/17/05
102 01/14/05 01/27/05
102 01/29/05 01/31/05
;
run;

data temp1;
   set  temp;

 format    startdt stopdt lagstop  mmddyy8. ;

 real_days = stopdt - startdt + 1;

 lagstop =lag(stopdt);
```

```
 if id = lag(id) then
    if lagstop >= stopdt   then
       real_days = 0;
    else if lagstop >= startdt  then
       real_days = stopdt - lagstop;
 run;

*************************************************************************;
* The following comments are copied from SAS Procedure Guide manuals.   ;
* The TIMEPLOT procedure plots one or more variables over time intervals.;
* OVERLAY plots all requests in one plot statement on one set of axes.  ;
* HILOC connects the leftmost plotting symbol to the rightmost with hyphens.;
*************************************************************************;

title "SGF 2007 Coders' Corner Demo";
proc timeplot data=temp1;
   plot startdt ='<' stopdt='>' / overlay ref='15JAN05'd   hiloc;
   by    id;
run;

proc print data=temp1;
run;

proc sql;
   select id, sum(real_days) as duration_in_days
       from temp1
       group by id
       order by id;
quit;
```

TECHNIQUES and RESULTS

SAS TIMEPLOT procedure is provided here to better understand the nature of the data. The visual demonstration highlights the gaps, overlaps, time intervals and complete embedding of one interval within another.

```
---------------------------- id=101 -----------------------------
 startdt       stopdt      min                               max
                           01/01/05                      01/31/05
                          *-------------------------------------*
01/01/05     01/09/05     |<--------->      |                   |
01/05/05     01/20/05     |     <-----------|----->             |
01/16/05     01/18/05     |                 |<->                |
01/26/05     01/31/05     |                 |           <---->  |
                          *-------------------------------------*
```

```
---------------------------- id=102 ----------------------------

 startdt       stopdt       min                                 max
                            01/01/05                         01/31/05
                           *-----------------------------------*
01/01/05     01/07/05      |<------>          |                  |
01/05/05     01/07/05      |     <->          |                  |
01/06/05     01/17/05      |      <----------|->                |
01/14/05     01/27/05      |                <|-------------->   |
01/29/05     01/31/05      |                  |                <>|
                           *-----------------------------------*
```

The LAG<n>(argument) function (LAG1, LAG2...LAG100) returns the value of the argument from the last LAG execution. LAG1 is often abbreviated into LAG.

Programmers are often alerted against conditional execution of LAG since undesirable lagged values will be returned. The following assignment is executed to store the lag(stopdt) value:

```
lagstop =lag(stopdt);
```

Why the above? Because the variable 'lagstop' is later used three times in the following condition:

```
if lagstop >= stopdt   then
       real_days = 0;
else if lagstop >= startdt  then
       real_days = stopdt - lagstop;
```

Now, the question comes up why there is a preceding 'if id = lag(id) then' in the program before the above 'if' block? Since this conditional LAG is the first statement and executed every time, no undesirable lagged value of 'id' is stored and returned.

The simple Proc Print produces the following:

SGF 2007 Coders' Corner Demo

Obs	id	startdt	stopdt	lagstop	real_ days
1	101	01/01/05	01/09/05	.	9
2	101	01/05/05	01/20/05	01/09/05	11
3	101	01/16/05	01/18/05	01/20/05	0
4	101	01/26/05	01/31/05	01/18/05	6
5	102	01/01/05	01/07/05	01/31/05	7
6	102	01/05/05	01/07/05	01/07/05	0
7	102	01/06/05	01/17/05	01/07/05	10
8	102	01/14/05	01/27/05	01/17/05	10
9	102	01/29/05	01/31/05	01/27/05	3

Ideally, the lagstop for FIRST.id should contain missing values for obs (1, 5). Since lagstop contains transient values, the author skips the effort.

The main driver variable is 'real_days'. When lagstop is coupled with the preceding 'if id = lag(id) then', it generates the desired 'real_days' results in the printout.

Use of the nested IF is a personal preference to eliminate SAS statement replications. Oftentimes, a properly hierarchically aligned IF's block provides the most easy-to-read and traceable programming logic. The solution is by no means exhaustive, and can be re-written in many different ways.

Using the above approach yields two advantages:
(1) No relatively complex SAS statements (Arrays, RETAIN, DO loops, FIRST.id Temporary variable).
(2) Simple processing with no generation of extra records and swelling of the temporary files.

Finally, we can use a simple SQL to summarize the durations for the subjects:

SGF 2007 Coders' Corner Demo

id	duration_ in_days
101	26
102	30

CONCLUSION

The LAG function is an extremely valuable tool because it enhances program efficiency and flexibility. Since the argument in LAG<n>(argument) can either be numeric or character, the potential usage extends well beyond the duration calculation.

Readers are also advised to explore the DIF function. The DIF function operates in a manner similar to the LAG function except that the difference between a value in the current observation and a value from one or more (up to 100) previous observations is computed. For example, DIF(X) is equivalent to X - LAG(X).

Although we have highlighted the utility of the LAG function in this article, important considerations such as clarity, conciseness and efficiency should also influence the choice of a specific programming approach for calculating duration.

ACKNOWLEDGEMENT

The author would like to thank his organization HERC, VA, for its full support. He is also grateful for the invaluable critique and proofreading of the article by his colleagues, Sam King and Andrea Shane.

He would also like to acknowledge Alice M. Cheng, from Scios Inc., of Fremont, CA. Her WUSS 2005 Coders' Corner award-winning article "Duration Calculation from a Clinical Programmer's Perspective" originally generated the author's interest.

CONTACT INFORMATION
Please contact the author with your questions and comments:

Pon Su
VA Palo Alto Health Care System
795 Willow Road (152 MPD)
Menlo Park, CA 94025
Pon.su@va.gov