PharmaSUG 2020 - Paper QT-253

Implementing a LEAD Function for Observations in a SAS® DATA Step

Timothy J. Harrington, Navitas Data Sciences, Inc.;

ABSTRACT

A common situation in DATA step processing is the need to reference the value of a variable (column) in a prior or later observation. The SAS® system provides the functions LAG and DIFF to return the value of the variable in the prior observation or the difference between the current value of the variable, in the Program Data Vector (PDV), and the prior value. LAGn and DIFFn refer to the nth prior value, where 1<=n<=100 and must not refer to before the first observation (_N_=1) in the dataset. However, there is no corresponding LEAD function which looks at values in observations still to be read into the PDV. This paper demonstrates three different methods of implementing a LEAD function functionality. The modus operandi of each method is illustrated with examples of SAS code and the advantages and disadvantages of each method are discussed, as is the suitability of each method for specific types of programming situations.

INTRODUCTION

The SAS system can only reference prior values of a variable by successively storing the value at each observation in a queue which retains those prior values. Values farther down the dataset, not yet read into the PDV, cannot yet be known unless they have been stored in some way from a earlier run. Otherwise, the SAS system does not have any way of looking ahead at observations not yet read. The three methods of such a LEAD function implementation described here are:

1. Inverting a data set and using LAG
2. Storing all the values of a column transposed as array elements to be referenced on the observation in the PDV. All three of these methods pre-process the dataset and create new columns concatenating the carried backwards values, which can now be referenced inside a SAS DATA step or a SAS PROCedure (including PROC SQL).
3. A macro which uses _N_ to identify each observation and then performs a MERGE to a copy of the observations using an observation identifier offset.

HOW A DATA STEP PROCESSES A DATASET

When the execution of a DATA Step begins a data structure called the Program Data Vector (PDV) is created. This contains all of the variables in the dataset, which are all initially set to missing. The first observation is then read into the PDV so now each variable contains its corresponding value in the first observation. The system variable _N_ is set to 1. The contents of any item in the PDV may now be read or modified until the end of the DATA step is reached, at which point the PDV contents are written to the output dataset. (An OUTPUT statement performs such a write to the output dataset with the PDV content values at that point in the DATA Step.) When execution returns to the beginning of the DATA step the contents of the PDV are reset to missing before being loaded with the values of the corresponding variables in the second observation and _N_ is incremented to 2. This process repeats until the last observation has been processed. (A DELETE statement sets everything in the PDV to missing at that point and the contents are not written to the output data set.) After each iteration of the DATA Step the PDV contents are reset to missing, so prior values, even if output to the output data set, are lost in the current DATA step. However, the RETAIN statement prevents the specified variable(s) from being reset to missing at the start of the next iteration, though if that variable exists in the input dataset it will be overwritten with the value in the current observation (including missing) in the input data set.

THE LAGN AND DIFN FUNCTIONS

The LAGn function retains the values from prior observations of the specified variable in a queue. For example, if the variable is called X, LAG(X) (or LAG1(X)) will hold the prior value of X. LAG2(X) will hold the value of X from two observations ago. The LAGn function refers to the nth prior value of a variable. The DIFn function is the difference between the current value (currently in the PDV) and the nth prior value. Using the example dataset below LAG(TIME) and LAG(AMT) would have the values shown in the columns below:

PATIENT	VISIT	PCTPT	TIME	EVID	AVALC	AMT	LAG (TIME)	DIF (TIME)	LAG (AMT)
1000	DAY 1	PREDOSE	2020-02-24 T08:47:00	0	LTR		.	.	
1000	DAY 1		2020-02-24 T08:55:00	1		60mg	2020-02-24 T08:47:00	00:08:00	
1000	DAY 1	30 MIN POST DOSE	2020-02-24 T09:31:00	0	2.24		2020-02-24 T08:55:00	00:36:00	60mg
1000	DAY 1	4 HRS POST DOSE	2020-02-24 T13:03:00	0	0.46		2020-02-24 T09:31:00	03:32:00	
1000	DAY 2	PREDOSE	2020-02-25 T09:02:00	0	0.05		2020-02-24 T13:03:00	19:59:00	
1000	DAY 2		2020-02-25 T09:10:00	1		65mg	2020-02-25 T09:02:00	00:08:00	
1000	DAY 2	30 MIN POST DOSE	2020-02-25 T09:37:00	0	2.53		2020-02-25 T09:10:00	00:27:00	65mg
1000	DAY 2	4 HRS POST DOSE	2020-02-25 T13:08:00	0	0.66		2020-02-25 T09:37:00	03:31:00	
1000	DAY 3	PREDOSE	2020-02-26 T08:52:00	0	0.06		2020-02-25 T13:08:00	19:44:00	
1000	DAY 3		2020-02-26 T09:02:00	1		70mg	2020-02-26 T08:52:00	00:10:00	
1000	DAY 3	30 MIN POST DOSE	2020-02-26 T09:31:00	0	2.72		2020-02-26 T09:02:00	00:29:00	70mg
1000	DAY 3	4 HRS POST DOSE	2020-02-26 T13:24:00	0	0.63		2020-02-26 T09:31:00	03:53:00	

Example PK data (PKRES) showing dosing (EVID=1) and PK sampling (EVID=0) observations

PATNUM = Patient ID, VISIT = Visit Text, PCTPT = Visit Timepoint,, TIME = Actual Date and Time of Sample Collection or Dose Administration (numeric with format is8601dt.). EVID=Event identifier (0=PK sample taken e.g. blood draw, 1=dose given to patient). AVALC = Character Result from Sample (LTR=Less Than Readable), AMT = Amount of Dose Administered

Points to note here are:

1. DIF only applies to numeric variables. DIF(TIME) is formatted as time8.

2. DIF is the subtraction of the prior value from the current value, since TIME is increasing the DIF values are all positive

3. DIF is missing if either the current or prior observation value are missing, or if the prior value reference is before the first observation (n in DIFn is less than _n_-1)

4. The default value of n is 1, but it can be any integer up to and including 100, but cannot be dynamically changed during the execution of the DATA step.

5. Care must be taken when grouping observations by one or more sorted variables that 'rollover' into the next group (in this example PATNUM and VISIT) is prevented. In the above example the TIME for the last VISIT (PCTPT='4 HRS POST DOSE') is carried into the first observation for the next VISIT (and in some cases the next PATNUM). Hence, when determining LAG1 at the first VISIT (and the first PATNUM) the LAG1 result should be ignored and be set to missing.

EXAMPLE OF THE NEED FOR A LEAD

In some programming situations there is a requirement to be able to 'look ahead' to a variable's value in the next or a 'future' observation. In above example there may be a need to determine the time the PREDOSE sample was taken before the dose was administered at a given VISIT. (The predose 'trough' prior to dosing). In this case the time the PREDOSE sample was taken at the DAY 2 visit would be 'lead1(time)-time, or 2020-02-25T09:02:00 - 2020-02-25T09:10:00 = 8 minutes earlier. Unfortunately there is no such LEAD function, the reason being the next observation has not yet been read into the PDV and hence cannot yet be known.

Below are described the three methods for implementing a LEAD function. All three methods require a one-time set up for the data set, using additional variables and in some cases additional temporary datasets, however once the set-up has been performed accessing the LEADn values is immediate and convenient.

METHOD 1: USING AN INVERTED LAG

This method is to sort the dataset by the group variables (PATNUM and VISIT) and sort the variable to be ordered for the carry forward (TIME) within the innermost group in reverse order and then use the LAG (or DIF) function on the reversed data to store the carried forwards value in a new column. The observations are then resorted back into the original order, so that 'carried forwards' has now become 'carried backwards'. Once this set-up has been performed the created column(s) can now be referred to in a current observation. An advantage of this method is no temporary datasets are needed to hold intermediate values.

Using the example data shown above, stored as columns and observations in a dataset named PKRES, the code below creates two new variables LEAD_DIF_TIME and LEAD_AMT. LEAD_DIF_TIME is used for storing the difference between TIME in the current observation and TIME in the next observation. LEAD_AMT will hold the amount of the dose about to occur. The AMT for the dose about to occur will be carried back to be LEAD_AMT at each of PREDOSE observations.

The first step is to sort the data by the grouping variables, in this case PATNUM and VISIT, and also sort TIME in reverse order within the innermost group (VISIT). Hence TIME is sorted using the keyword DESCENDING.

```
proc sort data=pkres;
  by patnum visit descending time;
run;
```

If a variable being referenced is a character string, in this case AMT, the column being created, LEAD_AMT, to hold the carried forward value must be declared with sufficient length to ensure the longest string encountered is not truncated. A numeric variable OBSCOUNT is used to keep track of the observation sequence number within each VISIT. The RETAIN statement prevents OBSCOUNT from being reset to missing at the start of each DATA step iteration. OBSCOUNT is reset to 1 at FIRST.VISIT. The LAGn or DIFn function is used to store the prior sorted backwards value (next value when sorted forwards) in the applicable LEAD_ column. An important point to note is the LEAD_ variables are set as missing on the first OBSCOUNT. If this were not done values would roll over into the next VISIT. The rule is if the carry back is by N observations the values of the LEAD_ variables must be set to missing when the observation counter is less than or equal to N. (obscount<=n).

```
data pkres(drop=obscount);
  set pkres;
  by patnum visit descending time;
  retain obscount 0;
  format lead_dif_time time8.;
  length lead_amt $20;
  obscount=obscount*(first.visit=0)+1;
  lead_dif_time=dif(time);
  lead_amt=lag(amt);
  if obscount=1 then do;
    lead_dif_time=.;
    lead_amt=' ';
  end;
run;
```

Once this DATA step is complete all that remains to be done is to drop OBSCOUNT and re-sort the observations back into their original order, so 'LEAD_' will then refer to the next instead of prior observation.

```
proc sort data=pkres;
  by patnum visit time;
run;
```

LEAD_DIF_TIME is now missing at the last observation for each VISIT and is otherwise negative because of the reversed order of TIME resulting from a later time being subtracted from an earlier time (The time is 'before' instead of 'after'). The PREDOSE 'trough' time until the next dose, and the amount of the dose about to be administered are now LEAD_DIF_TIME and LEAD_AMT respectively on the observations where PCTPT='PREDOSE'. Note: The assumption is being made that the data is 'clean', that is all the VISITs, EVID, PCTPT time-points, and TIME dates and times are all valid and correctly sequenced with respect to each other.

Note: LEAD_DIF_TIME is formatted as time8. instead of is8601dt. so '1959-12-31:' is not shown in front of the time

PATIENT	VISIT	PCTPT	TIME	EVID	AVALC	AMT	LEAD_DIF_TIME	LEAD_AMT
1000	DAY 1	PREDOSE	2020-02-24 T08:47:00	0	LTR		-00:08:00	.60mg
1000	DAY 1		2020-02-24 T08:55:00	1		60mg	-00:36:00	
1000	DAY 1	30 MIN POST DOSE	2020-02-24 T09:31:00	0	2.24		-03:32:00	
1000	DAY 1	4 HRS POST DOSE	2020-02-24 T13:03:00	0	0.46		.	
1000	DAY 2	PREDOSE	2020-02-25 T09:02:00	0	0.05		-00:08:00	65mg
1000	DAY 2		2020-02-25 T09:10:00	1		65mg	-00:27:00	
1000	DAY 2	30 MIN POST DOSE	2020-02-25 T09:37:00	0	2.53		-03:31:00	
1000	DAY 2	4 HRS POST DOSE	2020-02-25 T13:08:00	0	0.66		.	
1000	DAY 3	PREDOSE	2020-02-26 T08:52:00	0	0.06		-00:10:00	70mg
1000	DAY 3		2020-02-26 T09:02:00	1		70mg	-00:29:00	
1000	DAY 3	30 MIN POST DOSE	2020-02-26 T09:31:00	0	2.72		-03:53:00	
1000	DAY 3	4 HRS POST DOSE	2020-02-26 T13:24:00	0	0.63		.	

PKRES data after evaluating the time before the next dose or sample time, and the next dose amount

METHOD 2: USING AN ARRAY OF TRANSPOSED VALUES

This method is most suitable for data sets with a large number occurrences of group variables (e.g. the patient ID) and a small number of occurrences of the data items to be referenced within each group. First, the data is sorted by the group variable(s) and the variable to be carried backwards, then, using PROC TRANSPOSE, the variables are stored as columns for each group. The first value becomes the first created column, the second value becomes the second column, and so on. These created columns are then referenced by number (or could be stored in an array). An advantage of this method is there is no need to declare lengths for character holding variables, since PROC TRANSPOSE creates columns of an applicable length.

Using the above PKRES data as a simple example, the first step is to sort the observations into the appropriate order:

```
proc sort data=pkres;
  by patnum visit time;
run;
```

The next step is to create a data set with the column to be carried backwards transposed so the data is now in a horizontal format. In this case there will be four columns produced, corresponding to each of the four times at each VISIT. If the number of observations for the innermost BY variable varies, any excess column values would be set to missing. The PREFIX option names the transposed times as t1 – t4 (instead of col1-col4) and the transposed dose amounts as a1 - a4.

```
proc transpose data=pkres out=_t01 prefix=t;
  by patnum visit;
  var time;
run;
```

PATNUM	VISIT	_NAME_	T1	T2	T3	T4
1000	DAY 1	TIME	2010-02-24 T08:47:00	2010-02-24 T08:55:00	2010-02-24 T09:31:00	2010-02-24T 13:03:00
1000	DAY 2	TIME	2010-02-25 T09:02:00	2010-02-25 T09:10:00	2010-02-25 T09:37:00	2010-02-25 T13:08:00
1000	DAY 3	TIME	2010-02-26 T08:52:00	2010-02-26 T09:02:00	2010-02-26 T09:31:00	2010-02-26 T13:24:00

```
proc transpose data=pkres out=_t02 prefix=a;
  by patnum visit;
  var amt;
run;
```

PATNUM	VISIT	_NAME_	A1	A2	A3	A4
1000	DAY 1	AMT		60mg		
1000	DAY 2	AMT		65mg		
1000	DAY 3	AMT		70mg		

These PROC TRANSPOSE functions arrange the time and amount values as sequentially ordered columns which can now be joined back with the original data using a MERGE with the group identifiers, PATNUM and VISIT, as the BY variables.

```
data _t02;
  merge pkres(in=a) _t01(in=b) _t02(in=c);
  by patnum visit;
  if a and b and c;
run;
```

This next DATA step now sets LEAD_AMT and LEAD_DIF_TIME at the PCTPT=’PREDOSE’ observations by referencing the second column of transposed data. This second column is from the second observation in each VISIT, that is the dose taken (EVID=1), assuming all the values are valid and correctly ordered in the original data.

```
data _t03(drop=_name_ a1-a4 t1-t4);
  set _t02;
  length lead_amt $20;
  format lead_dif_time time8..;
  if pctpt='PREDOSE' then do;
    lead_amt=a2;
    lead_dif_time=time-t2;
  end;
run;
```

PATIENT	VISIT	PCTPT	TIME	EVID	AVALC	AMT	LEAD_DIF_TIME	LEAD_AMT
1000	DAY 1	PREDOSE	2020-02-24 T08:47:00	0	LTR		-00:08:00	.60mg
1000	DAY 1		2020-02-24 T08:55:00	1		60mg	.	
1000	DAY 1	30 MIN POST DOSE	2020-02-24 T09:31:00	0	2.24		.	
1000	DAY 1	4 HRS POST DOSE	2020-02-24 T13:03:00	0	0.46		.	
1000	DAY 2	PREDOSE	2020-02-25 T09:02:00	0	0.05		-00:08:00	65mg
1000	DAY 2		2020-02-25 T09:10:00	1		65mg	.	
1000	DAY 2	30 MIN POST DOSE	2020-02-25 T09:37:00	0	2.53		.	
1000	DAY 2	4 HRS POST DOSE	2020-02-25 T13:08:00	0	0.66		.	
1000	DAY 3	PREDOSE	2020-02-26 T08:52:00	0	0.06		-00:10:00	70mg
1000	DAY 3		2020-02-26 T09:02:00	1		70mg	.	
1000	DAY 3	30 MIN POST DOSE	2020-02-26 T09:31:00	0	2.72		.	
1000	DAY 3	4 HRS POST DOSE	2020-02-26 T13:24:00	0	0.63		.	

PKRES data after taking the second column of the transposed time and amount, ie: from the dose (EVID=1) observation, and joining to the PREDOSE observation

OBSERVATION OFFSET PAIRING

This method uses the _N_ variable, provided by SAS to identify the observation number in a DATA step. Each observation is matched to a copy of the dataset's key columns and the variable(s) to be carried backwards. The matching is performed using an offset of the key values so each column value to be carried backwards is displaced back by the offset. The detailed steps of this process are:

1. The observations are sorted by specified BY variables, one of these BY variables is defined as the group variable, the carry backwards will take place within each distinct value of this group variable.

2. If the data is not to be sorted, ie: the carry backward is for the entire dataset as one group, a dummy group variable is created with the same value (e.g. 1) on all observations.

3. A numeric group key, GRPKEY, is assigned to each block of observations with the same group variable value.

4. A numeric key, KEY0, is incrementally assigned to each observation within each group, reset to 1 at FIRST.GRPKEY

5. An offset key, KEYN, is created by subtracting the observation displacement (n) from KEY0

6. The data is sorted by GRPKEY and KEY0 and these output to a temporary dataset, A.

7. The original dataset is now sorted by GRPKEY and the offset key KEYN. These two key variables and the variable being carried backwards (VAR) are output to a second temporary dataset, B. KEYN is to KEY0.

8. The datasets A and B are now joined using a MERGE with GRPKEY and KEY0 as the BY variables, keeping only matching observations in dataset A. When the match is in A and B, VAR is overwritten with the new carried backwards value (or output to another specified column). When in A only, the absence of a matching value in B results in VAR being set to missing, this is for the last N observations in each grouping, where there would be no following VAR value to carry backwards from. Key values in B only are dropped, these would be the 'lost' values of VAR pushed back to before the beginning of each group.

9. The dataset created in the prior step is now the output dataset, which is resorted by the original BY variables. The temporary keys GRPKEY, KEY0, and KEYN are dropped.

The following illustration shows KEY0 being added as a sequential key for each value of GRPKEY in dataset A, To carry the value of X backwards by one observation a second dataset (B) is created and 1 is subtracted from KEY0 and the result stored in KEYN. In dataset B KEY0 is dropped, KEYN is renamed to KEY0, and X is renamed as LEAD_X. The two datasets are now MERGED by GRPKEY and KEY0. The last observation for each GRPKEY will have LEAD_X missing (NXTGRP1 in this example) because there is no matching value of GRPKEY and KEY0.

GRPKEY	KEY0	X
999	4	PRIGRP4
1000	1	FIRST
1000	2	SECOND
1000	3	THIRD
1000	4	FOURTH

Dataset A

GRPKEY	KEYN (KEY0-1) (Renamed to KEY0 before merge)	KEY0 (Dropped before merge)	LEAD_X
1000	0	1	FIRST
1000	1	2	SECOND
1000	2	3	THIRD
1000	3	4	FOURTH
1001	0	1	NXTGRP1

Dataset B

Below is a macro LEADN, which performs these functions, and an example call using the dataset from the prior examples. The parameters are:

INDS: Name of the input dataset

OUTDS: Name of the output dataset, if left blank the created carried backwards variable is added to INDS.

BYVARS: The BY variables to sort the dataset by. If none specified, the whole dataset is taken as one group.

GRPVAR: BYVARS column to be used for grouping (carry backward within the observations with a distinct value)

N: Number of observations to carry backwards, default is 1.

VAR: Name of the column to be carried backwards

NXTVARCB: Name of a new variable to contain the carried backward values of VAR (defaults to &var._cb)

Existing columns in the input dataset are unchanged.

```
%macro leadn(inds=&syslast, /*** Macro to simulate a LEADn function ***/
             outds=,
             byvars=,
             grpvar=,
             n=1,
             var=,
             nxtvarcb= );

  %if &outds= %then %do;
    %let outds=&inds;/*If no out data set, add &nxtvarcb to input data set*/
  %end;

  %if &nxtvarcb= %then %do; /* If no variable specified to hold the lead */
    %let nxtvarcb=&var._cb; /* value create a column named as &VAR with  */
  %end;                     /* suffix _cb */

  %local sortkey;
  %let sortkey=bvar1 &byvars;   /* Define the sortkey */
  %if &sortkey=bvar1 %then %do; /* If no BY variables specified use one */
    %let grpvar=&sortkey;       /* dummy sort-key for the whole dataset  */
  %end;

    data _t1;
      set &inds;
      bvar1=1;
    run;

    proc sort data=_t1;
      by &sortkey;
    run;

    data _t2;
      set _t1;
      by &sortkey;
      retain grpkey key0 0;
      grpkey=grpkey+first.&grpvar;   /* Group of distinct values key */
      key0=key0*(first.&grpvar=0)+1; /* Obs.  count within each group */
      key&n=key0-&n;                 /* Key with the N obs. offset */
    run;
```

```
  proc sort data=_t2 out=_t3(drop=key&n); /* Create a copy of the */
    by grpkey key0;               /* data set sorted by the group */
  run;                            /* observation key */

  proc sort data=_t2(keep=grpkey key&n &var)
    out=_t4(rename=(key&n=key0 &var=&nxtvarcb));
    by grpkey key&n;              /* Create an index dataset of the */
  run;                            /* key and offset key and the new VAR */


  data _t5;
    merge _t3(in=a) _t4(in=b);    /* Join by the group and the */
    by grpkey key0;               /* corresponding observation keys */
    if a;
  run;


  proc sort data=_t5 out=&outds(drop=bvar1 grpkey key0);
    by &sortkey;                  /* Resort by original BY variables */
  run;                            /* drop the temporary key variables */


  proc datasets nolist;           /* Remove the temporary data sets */
    delete _t1-_t5;
  run;

%mend leadn;
```

The following macro calls create a column NEXTTIME with LEAD1(TIME) within each VISIT and a column NEXT_AMT with LEAD1(AMT) within each VISIT. Subtracting NEXTTIME from TIME would yield the difference, the equivalent of DIF(TIME) but using the next instead of the prior observation:

```
%leadn(inds=pkres,
        outds=pk1,
        byvars=patnum visit time,
        grpvar=visit,
        n=1,
        var=time,
        nxtvarcb=nxttime);

%leadn(inds=pkres,
        outds=pk1,
        byvars=patnum visit time,
        grpvar=visit,
        n=1,
        var=time,
        nxtvarcb=nxttime);
```

PATIENT	VISIT	PCTPT	TIME	EVID	AVALC	AMT	NEXTTIME	NEXT_AMT
1000	DAY 1	PREDOSE	2020-02-24 T08:47:00	0	LTR		2020-02-24 T08:55:00	.60mg
1000	DAY 1		2020-02-24 T08:55:00	1		60mg	2020-02-24 T09:31:00	
1000	DAY 1	30 MIN POST DOSE	2020-02-24 T09:31:00	0	2.24		. 2020-02-24 T13:03:00	
1000	DAY 1	4 HRS POST DOSE	2020-02-24 T13:03:00	0	0.46		.	
1000	DAY 2	PREDOSE	2020-02-25 T09:02:00	0	0.05		2020-02-25 T09:10:00	65mg
1000	DAY 2		2020-02-25 T09:10:00	1		65mg	2020-02-25 T09:37:00	
1000	DAY 2	30 MIN POST DOSE	2020-02-25 T09:37:00	0	2.53		2020-02-25 T13:08:00	
1000	DAY 2	4 HRS POST DOSE	2020-02-25 T13:08:00	0	0.66		.	
1000	DAY 3	PREDOSE	2020-02-26 T08:52:00	0	0.06		2020-02-26 T09:02:00	70mg
1000	DAY 3		2020-02-26 T09:02:00	1		70mg	. 2020-02-26 T09:31:00	
1000	DAY 3	30 MIN POST DOSE	2020-02-26 T09:31:00	0	2.72		2020-02-26 T13:24:00	
1000	DAY 3	4 HRS POST DOSE	2020-02-26 T13:24:00	0	0.63		.	

PKRES data with lead times and dose amounts after using the LEADN macro

CONCLUSION

This paper has introduced and discussed three techniques to simulate a LEAD function, corresponding to the LAG function. The three methods described are not exhaustive but are methods that can be conveniently incorporated into an application program. Programmers should consider which approach is most appropriate for the data being handled and the given programming situation. Other methods not mentioned in this paper are also available to simulate a LEAD function. The 'Recommended Further Reading' section below contains references to methods both similar to and different from this paper content.

ACKNOWLEDGMENTS

Navitas data Sciences, Inc.

Pharmasug 2020 paper Selection Committee.

RECOMMENDED READING

- *Techniques to Perform the Lead Function Through Data Manipulation Kruti Pandya, Independent Consultant Niraj J. Pandya,eClinical Solutions, A Division of Eliassen Group*

  https://www.lexjansen.com/nesug/nesug09/cc/CC26.pdf

- *Paper 3699-2019Calculating Leads (and Lags) inSAS®: One Problem, Many SolutionsAndrew Gannon, The Financial Risk Group, Cary NC*

  https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3699-2019.pdf

- *LEADS AND LAGS IN SAS®Mark Keintz, Wharton Research Data Services, University of Pennsylvania*

  https://www.lexjansen.com/nesug/nesug13/113_Final_Paper.pdf

- *USER GROUP 2019 Leads and Lags:Static and Dynamic Queues in the SAS® DATA STEP, 2nd edMark KeintzWharton Research Data ServicesTA S S: 08FEB2019. Copyright © SAS Institute Inc. All rights reserved.*

  http://torsas.ca/attachments/File/20190208/1_m_k_leads_and_lags.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author at:

Timothy J. Harrington
Professional Services,
Navitas Data Sciences, Inc, 1610 medical Drive, Suite 300, Pottstown, PA 19464
610-970-2333

timothy.harrington@navitaslifesciences.com

www.navitaslifesciences.com