Techniques to Perform the Lead Function Through Data Manipulation

Kruti Pandya, Independent Consultant
Niraj J. Pandya, eClinical Solutions, A Division of Eliassen Group

ABSTRACT

There are numerous amounts of functions available in SAS® which makes data imputations, data comparison and data manipulation easier for a programmer. Though there are some tasks for which SAS does not have functions readily available for use. One such task is to read next record while working on the current record and at the first thought one can imagine an exact opposite processing of data to LAG function, which can be named as LEAD function. Unfortunately SAS does not offer any such function. But fortunately there are some tricks and techniques which can simulate opposite processing of LAG function and can read next couple of records while still processing current record in the dataset. By little addition of extra piece of code, some of these techniques can also handle cases where processing has to be done by grouping variables. This paper is an effort to list such techniques with examples which are proved to work successfully and can be utilized easily by SAS programmers based on various requirements of their day to day work.

KEYWORDS

LEAD FUNCTION, LAG, POINT

INTRODUCTION

If you look at the basic structure of DATA step in SAS and how it works, you will know that this is how it works: one observation comes in; some operation is done on it; it is written out to an output dataset; next observation comes in and same processing is done for all future observations. In this framework, it is possible for SAS to recall what was the value of variable when last time it passed through program data vector (PDV), but it is impossible to know future values of the same variable and this is the reason why SAS does not have inbuilt LEAD function which can simulate exact opposite processing of LAG function. However there are some patterns of code which can look for the information in the future and allows you to work around based on these future values of the variables. We will look in detail for some of these techniques in this compilation.

METHOD 1

One simplest method to look in to future value of the variable is to sort the dataset in descending order, then apply LAG function on it, and then again sort the dataset in ascending order of the values of the variable. This method is very simple to understand and with the addition of little extra statements in the code, it can handle by groups processing also.
METHOD 2:

This solution is based on one-on-one merge of dataset with itself. In this method, the second dataset on the merge statement is the same as first one but this time it is controlled by FIRSTOBS data option.

FIRSTOBS = 2 will start the dataset from the second observation and in this way you will be looking at current observation and a next observation at the same time. One thing you have to make sure that you rename the common variable or the current values of common variables will be overwritten by values of next observation.

This method can also handle by group processing up to some extent. You have to rename group variable also and then have to compare the values of original group variable and renamed group variable and have to set the value of variable of interest to missing when two group variables don’t have same values. Following example is doing this by group processing.

CODE 1:

```
Data a;
  Input id x grp;
  datalines;
  1 10 1
  2 20 1
  3 30 1
  4 40 1
  5 50 1
  1 10 2
  2 20 2
  3 30 2
  4 40 2
  5 50 2
Run;
```

```
Proc sort data = a;
  by grp descending id;
Run;
```

```
Data a;
  Set a;
  by grp descending id;
  lagid = lag(id);
  lagx = lag(x);
  if first.grp then do;
    lagid = .;
    lagx = .;
  end;
Run;
```

```
Proc sort data = a;
  by grp id;
Run;
```

OUTPUT OF CODE 1:

```
<table>
<thead>
<tr>
<th>Id</th>
<th>x</th>
<th>grp</th>
<th>lagid</th>
<th>lagx</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>1</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>1</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>1</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>2</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>2</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>2</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>2</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
```

In this example, we have to keep grouping variable GRP first in the proc sort before sorting the dataset in descending order of values of ID variable. After applying LAG function on ID and X variables and before sorting the dataset back in to ascending order of ID variable we need to set the values of LAGID and LAGX equal to missing for first observation of each value of GRP. We need to do this because we do not want to cross the boundary of GRP variable and values from second group are not being read in first group.
METHOD 3:

Proc SQL also provides an excellent solution to read future observations and it works perfectly fine even if by group processing is required. In this method we need to have a variable in the dataset which keeps the track of number of observation within each group, its value increments as we proceed in the dataset by each record. If any such variable is not present, it can be generated easily and then proc sql can read the future record with the help of this incrementing variable. Following example will show how it works.

CODE 3:

``` SAS 
Data one;
   Input x grp @@;
   datalines;
   1 1 2 1 3 1 4 2 5 2
Run;

Data next;
   Merge old old(firstobs = 2 rename=(x=nextx grp=nextgrp));
   If grp ne nextgrp then nextx = .;
Run;

Proc sql;
   Select one.*, onenext.x as nextx
   from one left join one as onenext
   on one.grp = onenext.grp
   and one.count + 1 = onenext.count;
Quit;
```

CODE 2:

``` SAS 
Data old;
   Input x grp @@;
   datalines;
   1 1 2 1 3 1 4 2 5 2
Run;

Data next;
   Merge old old(firstobs = 2 rename=(x=nextx grp=nextgrp));
   If grp ne nextgrp then nextx = .;
Run;
```

OUTPUT OF CODE 2:

<table>
<thead>
<tr>
<th>x</th>
<th>grp</th>
<th>nextx</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>.</td>
</tr>
</tbody>
</table>

OUTPUT OF CODE 3:

<table>
<thead>
<tr>
<th>x</th>
<th>grp</th>
<th>count</th>
<th>nextx</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>2</td>
<td>.</td>
</tr>
</tbody>
</table>
As showed in above example, SQL makes it very easy to do self join with different name of common variable which is not going to work same way when by statement is used with one-on-one merge of the dataset with itself.

**METHOD 4:**

This solution was suggested by Paul Dorfman. It is a very efficient and more elegant solution. This solution uses the POINT feature along with system variable _N_ which can point to a specified observation number and read the values from it. In doing so this might be consuming more resources but it is worth using little extra resources for what we are trying to achieve here. Following example shows how this works.

**CODE 4:**

```plaintext
Data one;
   Input Id x grp @@;
datalines;
1 10 1 2 20 1 3 30 1
4 40 1 5 50 1 1 10 2
2 20 2 3 30 2 4 40 2
5 50 2
; Run;
DATA Leads;
   _N_ ++ 1;
   IF _N_ <= N THEN DO;
      Set one POINT=_N_;
      y = x;
   END;
   ELSE y = .;
   Set one nobs = n;
RUN;
```

**OUTPUT OF CODE 4:**

<table>
<thead>
<tr>
<th>Id</th>
<th>x</th>
<th>grp</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>2</td>
<td>50</td>
</tr>
</tbody>
</table>

This method can handle by group processing also very efficiently. In this example we saw that while creating Y variable, we crossed the group boundary. Following example (CODE 5) shows how we can handle such by groups processing without crossing group boundary.

**CODE 5:**

```plaintext
Data one;
   Set one;
   laggrp = lag(grp);
Run;
Data Leads;
   _N_ ++ 1;
   IF _N_ <= N THEN DO;
      Set one POINT=_N_; 
      If laggrp = grp then do;
         y = x;
      End;
   END;
   ELSE y = .;
   Set one nobs = n;
   by grp;
Run;
```

**OUTPUT OF CODE 5:**

<table>
<thead>
<tr>
<th>Id</th>
<th>x</th>
<th>grp</th>
<th>laggrp</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>1</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>1</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>2</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>2</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>2</td>
<td>2</td>
<td>50</td>
</tr>
</tbody>
</table>
In above example (CODE 5), we applied LAG first on grouping variable. Then at the time of subsetting the dataset with the use of POINT, we added one extra conditional statement to check if original and lagged group variables have same value before creating Y variable from X (\( y=x \) statement). Make a note of an additional BY statement (by grp) after SET statement in above example. This BY statement is necessary as we are applying by group processing.

By using this technique, you can jump a bit higher and even look values of two, three or any numbers of observations in advance by advancing the value of automatic variable _N_ by appropriate number (in general _N_ ++ _n_ ). Only thing you have to keep in mind while doing so is before applying the POINT processing, you have to apply appropriate lag (lagn) to grouping variable.

Following example (CODE 6) shows how to read the values in advance by three observations while still reading current observation.

### CODE 6:
```plaintext
Data one;
  Set one;
    laggrp = lag3(grp);
  Run;

Data Leads;
  _N_ ++ 3;
  IF _N_ <= N THEN DO;
    Set one POINT=_N_;
    If laggrp = grp then do;
      y = x;
    End;
  END;
  ELSE y = .;
  Set one nobs = n;
  by grp;
Run;
```

### OUTPUT OF CODE 6:
```
<table>
<thead>
<tr>
<th>Id</th>
<th>x</th>
<th>grp</th>
<th>laggrp</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>.</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>1</td>
<td>.</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>1</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>2</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>2</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>2</td>
<td>2</td>
<td>.</td>
</tr>
</tbody>
</table>
```

### CONCLUSION

Now so that we have looked in to few techniques to simulate the opposite of LAG function, you can have a look in to the future. Methods explained in this paper have their own advantages and disadvantages and they will dominate when you use them. Before applying or selecting any of these methods, you have to understand the structure of your data and the result that you are trying to achieve. There are some considerations which will affect what method you choose. Techniques mentioned in this paper are proposed by well known industry experts and have been put through testing for many years. This is just an effort to bring all these techniques under one compilation.
REFERENCES

- http://www.sconsig.com/sastips/tip00070.htm

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

Kruti Pandya
Email: kruti_pandya83@yahoo.com

Niraj J. Pandya
Email: npandya@egistar.com or niraj_mech@yahoo.com

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.