Learning SAS: Understanding and Applying the LAG Function with Practical Examples

Authored by

Mohammed looti

October 31, 2025

The **SAS LAG function** is an indispensable tool for analysts working with sequential or time series data. It is primarily utilized to retrieve previously recorded values of a variable within the current Data Step iteration. This capability allows users to perform critical tasks such as calculating differences over time, comparing current observations to prior periods, or constructing models that rely on lagged values for forecasting.

Understanding how the **LAG function** operates is crucial, as it fundamentally relies on a queue mechanism. When SAS processes data sequentially, the LAG function holds the value of the specified variable in a queue and releases the value from the previous iteration when called. This means that for the very first observation processed, the lagged value will be missing, as there is no prior value to retrieve. This behavior is standard and expected when initiating any lag calculation.

The basic syntax for calculating a simple 1-period lag is straightforward, requiring only the function name and the variable of interest. If you need to retrieve values from further back in the sequence, SAS provides convenient shorthand notation by appending the lag depth (e.g., 2, 3, or N) directly to the function name.

The core syntax is presented below:

lag1_value = lag(value);

While `LAG(value)` defaults to finding the immediate preceding value (a 1-period lag), you have the flexibility to specify deeper lags. For instance, using `LAG2`, `LAG3`, or `LAGN` allows you to efficiently calculate the 2-lagged, 3-lagged, or N-lagged values of a variable, respectively. This feature is particularly useful when analyzing seasonal data or patterns that span multiple periods. The following examples demonstrate how to apply these concepts in practical SAS scenarios.

Example 1: Calculating Lagged Values for a Single Variable

In many business and scientific applications, it is necessary to compare the current metric against the performance of previous periods. Consider a scenario where we track daily sales figures for a single retail store. To analyze momentum or predict future sales, we need to easily access the sales figures from the day prior, two days prior, and so on. This requirement makes the **LAG function** perfectly suited for this type of sequential data manipulation.

The following SAS code establishes a sample dataset representing the total sales achieved on twelve consecutive days. This dataset will serve as the input for our lag calculations.

```
/*create dataset*/
data original_data;
input day $ sales;
```

```
datalines;
1 14
2 19
3 22
4 20
5 16
6 26
7 40
8 43
9 29
10 30
11 35
12 33
;
run;

/*view dataset*/
proc print data=original_data;
```

Executing the code above produces a simple, ordered dataset showing the daily progression of sales:

Obs	day	sales
1	1	14
2	2	19
3	3	22
4	4	20
5	5	16
6	6	26
7	7	40
8	8	43
9	9	29
10	10	30
11	11	35
12	12	33

Step-by-Step Implementation of Simple Lags

To calculate the lagged values, we initiate a new Data Step and use the `SET` statement to read the existing data. Within this step, we introduce three new variables: `lag1_sales`, `lag2_sales`, and `lag3_sales`. Each new variable utilizes the appropriate **LAG function** variant to capture sales from one, two, and three days prior, respectively.

It is important to remember that the first few observations in the lagged columns will automatically contain missing values. For instance, `lag3_sales` will be missing for the first three observations because there are no preceding records available in the data queue to satisfy the 3-day lookback requirement.

The subsequent code shows the required syntax for computing sales lagged by one, two, and three days:

```
/*create new dataset that shows lagged values of sales*/
data new_data;
set original_data;
lag1_sales = lag(sales);
lag2_sales = lag2(sales);
lag3_sales = lag3(sales);
run;

/*view new dataset*/
proc print data=new_data;
```

The output clearly illustrates the result of the lag operations. The newly created columns--`lag1_sales`, `lag2_sales`, and `lag3_sales`--accurately reflect the sales figures shifted down by one, two, and three periods, respectively. This allows for direct comparison and calculation of period-over-period changes.

Obs	day	sales	lag1_sales	lag2_sales	lag3_sales
1	1	14	.	.	.
2	2	19	14	.	.
3	3	22	19	14	.
4	4	20	22	19	14
5	5	16	20	22	19
6	6	26	16	20	22
7	7	40	26	16	20
8	8	43	40	26	16
9	9	29	43	40	26
10	10	30	29	43	40
11	11	35	30	29	43
12	12	33	35	30	29

Example 2: Calculating Lagged Values by Group

A common complexity in data analysis arises when applying lag calculations to datasets containing multiple distinct groups, such as sales data for several different stores or regions. If we simply apply the **LAG function** without accounting for these groups, the calculation will incorrectly carry over the last observation of one group to the first observation of the next group. This unintended result, often referred to as "data leakage," can severely skew subsequent analyses.

To demonstrate this, consider a dataset containing sales records for two separate stores, Store A and Store B, tracked over consecutive days. We aim to calculate the 1-day lagged sales value for each store independently, ensuring the lag calculation resets when transitioning from Store A to Store B.

The initial dataset setup is as follows:

```
/*create dataset*/
data original_data;
input store $ sales;
datalines;
A 14
A 19
A 22
A 20
```

A 16
A 26
B 40
B 43
B 29
B 30
B 35
B 33
;
run;

/*view dataset*/
proc print data=original_data;

This dataset shows the sales figures organized first by Store A, and then sequentially by Store B.

Obs	store	sales
1	A	14
2	A	19
3	A	22
4	A	20
5	A	16
6	A	26
7	B	40
8	B	43
9	B	29
10	B	30
11	B	35
12	B	33

Handling Group Boundaries with the BY Statement

To correctly calculate lagged values for each store independently, we must incorporate the **BY statement** into the Data Step. The **BY statement** instructs SAS to process the data in groups defined by the specified variable (in this case, `store`). Crucially, the **BY statement** generates temporary variables, `FIRST.store` and `LAST.store`, which flag the first and last observation within each group.

We leverage the `FIRST.store` indicator to reset the lag calculation. When SAS encounters the first observation of a new group (i.e., when `FIRST.store` is true), we explicitly set the lagged sales value (`lag1_sales`) to a missing value (`.`). This action ensures that the lag queue is effectively cleared at the beginning of each store's data, preventing the last sales figure from Store A from appearing as the lagged value for the first observation of Store B.

The following code implements the group-wise lag calculation:

```
/*create new dataset that shows lagged values of sales by store*/
data new_data;
set original_data;
by store;
lag1_sales = lag(sales);
if first.store then lag1_sales = .;
run;

/*view new dataset*/
proc print data=new_data;
```

Upon reviewing the output, observe the resulting `lag1_sales` column. The values now correctly represent the 1-day lagged sales strictly within each store category. Crucially, in row 7, which marks the transition from Store A to Store B, the `lag1_sales` value is correctly set to missing, ensuring the integrity of the group-wise time series analysis. If the **BY statement** and conditional logic were omitted, row 7 would erroneously display the last sales figure of Store A (26) as the lagged value for Store B.

Obs	store	sales	lag1_sales
1	A	14	.
2	A	19	14
3	A	22	19
4	A	20	22
5	A	16	20
6	A	26	16
7	B	40	.
8	B	43	40
9	B	29	43
10	B	30	29
11	B	35	30
12	B	33	35

Additional Resources

Mastering the **LAG function** opens up numerous possibilities for advanced data manipulation and statistical modeling in SAS. For users seeking to deepen their expertise, the following tutorials provide guidance on other common and powerful tasks available within the SAS environment:

How to perform other common tasks in SAS (Link placeholder)
Advanced techniques for data manipulation (Link placeholder)
Introduction to time-series modeling in SAS (Link placeholder)