

## Supplement: The LAG Statement

This supplement is offered in response to a question a student raised with regard to the `lag` statement. Please refer to the course notes/textbook for more details on this statement and its purpose.

A rather unexpected problem occurs when we use `lag` along with a conditional statement such as `if/then`. As you'll see from online searches (i.e. Google Groups), you'll find that the problem of `lag` combined with `if/then` can lead to a multitude of headaches. Let's see a couple examples to understand what is happening.

---

---

**Example:** In this example, we want to create a program which will compute the moving average (of the closing stock price `close`) over the periods  $t$ ,  $t - 1$ , and  $t - 2$ . The `mean()` function will only return a value of `.` when all of its arguments are missing. For example, `mean(5,.,.)` will return a value of 5 and `mean(5,6,.)` will return a value of 5.5. We want to avoid this problem by assigning a missing value to the moving average variable if any of  $t$ ,  $t - 1$ , or  $t - 2$  leads to a corresponding missing entry for `close`.

To make things simple, we can just focus on the  $t - 2$  lagged closing price. If that value is missing, then we'll make the moving average missing. If that value is not missing, then the  $t - 1$  and  $t$  closing prices are automatically not missing.

Note that I have offered two methods in the SAS code. The first method offers a straightforward algorithm on establishing the moving average according the scheme mentioned above. The second method seems to do the exact same thing but there's a definite difference (see the output for the comparison between `MA.Yes` (from the first method) and `MA.No` (from the second method)).

Here is a simple explanation about why there is a difference:

- The second method uses the `lag` functions **implicitly** whereas the first method uses the functions **explicitly** through `y` and `z`.
- **VERY IMPORTANT:** Since the implicit `lag` functions are *activated* only when the `else` condition is invoked, SAS will *start* keeping track of lags only when the `else` condition is applied.
- For our code, the first time the `else` condition is invoked is when `close` is 30 (by then, `z` is non-missing). When SAS encounters `lagclose` under the *second method*, it is encountering this for the **first time** ... and so it begins to officially keep track of `close`.

At this point you never asked SAS to keep track of `close` so it has no recollection of a previous value of `close`. Thus, as far as SAS is concerned, the previous value of `close` was simply `.` (that is, it was missing). This of course means the value

for the previous-previous value of `close` was `.` as well. So, when `close` is 30, the expression `MA_No = mean(close,lag(close),lag2(close))` is rendered as `MA_No = mean(30,.,.)` which is simply 30. That is why the first value of `MA_No` is 30 and not 20 as it should be (look under the corresponding value for `MA_Yes`).

- The second time the `else` condition is invoked is when `close` is 40. Under the *second method*, keep in mind that at this point SAS has been keeping track of `close`.

Thus, as far as SAS is concerned, the previous value of `close` was 30. But the value for the previous-previous value of `close` is missing! This should make sense if you keep in mind that SAS only started started keeping track of the `close` variable when it was a value of 30.

So, when `close` is 40, the expression `MA_No = mean(close,lag(close),lag2(close))` is rendered as `MA_No = mean(40,30,.)` which is simply 35. That is why the second value of `MA_No` is 35 and not 40 as it should be (look under the corresponding value for `MA_Yes`).

- The third time the `else` condition is invoked is when `close` is 50.

From the perspective of SAS, the previous value of `close` was 40. The value for the previous-previous value of `close` is 30.

So, when `close` is 50, the expression `MA_No = mean(close,lag(close),lag2(close))` is rendered as `MA_No = mean(50,40,30)` which is 40. **Now, we see that the lags have *caught up* and so the remaining values of `MA_Yes` and `MA_No` will match.**

SAS Code

```

data MovAvg; input close;
y = lag(close); z = lag2(close); ** Explicit definition of lags;

** FIRST METHOD: The following lines lead to the correct result;
if z=. then MA_Yes=.; else MA_Yes = mean(close,y,z);

** SECOND METHOD: The following lines lead to some unexpected results;
if z=. then MA_No=.; else MA_No = mean(close,lag(close),lag2(close));

datalines;
10
20
30
40
50
60
70
;
run;
proc print data=MovAvg; run;

```

SAS Code

SAS Output

Obs	close	y	z	MA_Yes	MA_No
1	10	.	.	.	.
2	20	10	.	.	.
3	30	20	10	20	30
4	40	30	20	30	35
5	50	40	30	40	40
6	60	50	40	50	50
7	70	60	50	60	60

SAS Output

---



---

### Example:

Take a look at the following code. There seems to be no difference between MY CODE and STUDENT'S CODE but the output shows that there IS a difference.

Based on the structure of IF [condition] THEN [execute], the lags ARE indeed referenced in the [condition] ... but when the [execute] part is invoked, it is as if SAS forgets it was keeping track of the lags. The act of "starting all over again" seems to be the only reasonable explanation. According the SAS information available online, it is safe to put LAGs in the [condition] but it can be very dangerous to put LAGs in the [execute] part since the behavior can be unexpected.

Since SAS seems to keep track of LAGS in a completely separate way between the [condition] realm and the [execute] realm, my advice is to use the methodology I have shown in the previous example (under *FIRST METHOD*) and in this example (under *MY CODE*). **That is, always explicitly define a surrogate variable that always keeps track of the lag at every point of the data step ... if you do this, you cannot go wrong!**

SAS Code

```

data MovingAverage; input close;

*** MY CODE ... leads to correct answer;
y = lag(close); z = lag2(close); ** Explicit definition of lags;

if y^=. and z^=. then MA_Correct=mean(close,y,z);
else MA_Correct=.;

*** STUDENT'S CODE ... leads to strange behavior;
if lag(close)^=. and lag2(close)^=.
  then MA_Weird=mean(close,lag(close),lag2(close));
else MA_Weird=.;

datalines;
10
20
30
40
50
60
70
;
run;

proc print data=MovingAverage;run;

```

SAS Code

SAS Output

Obs	close	y	z	MA_Correct	MA_Weird
1	10	.	.	.	.
2	20	10	.	.	.
3	30	20	10	20	30
4	40	30	20	30	35
5	50	40	30	40	40
6	60	50	40	50	50
7	70	60	50	60	60

SAS Output