

The Data Step Crystal Ball: Observing Future Rows

Michael Allie



4th-7th November
KAP Europa
Frankfurt, Germany

EU 2018

The Clinical Data
Science Conference



Introduction

- I encountered an analysis request that required reading future values in a data step
- The analysis involved comparison of different events within the same dataset
 - Flagging instances when 2 or more consecutive lab results were below the lower normal limit
- A creative solution was needed because data steps read data from input datasets one record at a time
- Only looking backwards would not flag all the records of interest

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	6.7630592416	3.8857	
2	1	GLUC	3.3916498243	3.8857	Y ←
3	1	GLUC	3.5584941976	3.8857	Y
4	1	GLUC	2.5299413569	3.8857	Y
5	1	GLUC	4.7909099201	3.8857	
6	1	GLUC	5.2967146997	3.8857	
7	1	GLUC	6.3083869808	3.8857	
8	1	GLUC	5.3322204975	3.8857	
9	1	GLUC	6.1226277479	3.8857	
10	1	GLUC	4.5676621504	3.8857	
11	1	GLUC	3.7288569046	3.8857	
12	1	GLUC	4.5463340997	3.8857	



Method 1: Sort data in reverse and utilise lag operator

- The lag operator allows you retrieve a value from a previous record

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO
1	1	GLUC	6.7630592416	3.8857
2	1	GLUC	3.3916498243	3.8857
3	1	GLUC	3.5584941976	3.8857
4	1	GLUC	2.5299413569	3.8857
5	1	GLUC	4.7909099201	3.8857
6	1	GLUC	5.2967146997	3.8857
7	1	GLUC	6.3083869808	3.8857
8	1	GLUC	5.3322204975	3.8857
9	1	GLUC	6.1226277479	3.8857
10	1	GLUC	4.5676621504	3.8857
11	1	GLUC	3.7288569046	3.8857
12	1	GLUC	4.5463340997	3.8857



Method 1: Sort data in reverse and utilise lag operator

- The lag operator allows you retrieve a value from a previous record

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO
1	1	GLUC	6.7630592416	3.8857
2	1	GLUC	3.3916498243	3.8857
3	1	GLUC	3.5584941976	3.8857
4	1	GLUC	2.5299413569	3.8857
5	1	GLUC	4.7909099201	3.8857
6	1	GLUC	5.2967146997	3.8857
7	1	GLUC	6.3083869808	3.8857
8	1	GLUC	5.3322204975	3.8857
9	1	GLUC	6.1226277479	3.8857
10	1	GLUC	4.5676621504	3.8857
11	1	GLUC	3.7288569046	3.8857
12	1	GLUC	4.5463340997	3.8857



Method 1: Sort data in reverse and utilise lag operator

- The lag operator allows you retrieve a value from a previous record

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO
1	1	GLUC	6.7630592416	3.8857
2	1	GLUC	3.3916498243	3.8857
3	1	GLUC	3.5584941976	3.8857
4	1	GLUC	2.5299413569	3.8857
5	1	GLUC	4.7909099201	3.8857
6	1	GLUC	5.2967146997	3.8857
7	1	GLUC	6.3083869808	3.8857
8	1	GLUC	5.3322204975	3.8857
9	1	GLUC	6.1226277479	3.8857
10	1	GLUC	4.5676621504	3.8857
11	1	GLUC	3.7288569046	3.8857
12	1	GLUC	4.5463340997	3.8857



Method 1: Sort data in reverse and utilise lag operator

- The lag operator allows you retrieve a value from a previous record

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO
1	1	GLUC	6.7630592416	3.8857
2	1	GLUC	3.3916498243	3.8857
3	1	GLUC	3.5584941976	3.8857
4	1	GLUC	2.5299413569	3.8857
5	1	GLUC	4.7909099201	3.8857
6	1	GLUC	5.2967146997	3.8857
7	1	GLUC	6.3083869808	3.8857
8	1	GLUC	5.3322204975	3.8857
9	1	GLUC	6.1226277479	3.8857
10	1	GLUC	4.5676621504	3.8857
11	1	GLUC	3.7288569046	3.8857
12	1	GLUC	4.5463340997	3.8857



Method 1: Sort data in reverse and utilise lag operator

- The lag operator allows you retrieve a value from a previous record
- This does not the flag the first in a series of consecutive low events so more processing is required
- The dataset can be sorted into reverse chronological order and the same logic applied again to flag the records that were missed in the first pass

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	6.7630592416	3.8857	
2	1	GLUC	3.3916498243	3.8857	
3	1	GLUC	3.5584941976	3.8857	Y
4	1	GLUC	2.5299413569	3.8857	Y
5	1	GLUC	4.7909099201	3.8857	
6	1	GLUC	5.2967146997	3.8857	
7	1	GLUC	6.3083869808	3.8857	
8	1	GLUC	5.3322204975	3.8857	
9	1	GLUC	6.1226277479	3.8857	
10	1	GLUC	4.5676621504	3.8857	
11	1	GLUC	3.7288569046	3.8857	
12	1	GLUC	4.5463340997	3.8857	

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	4.5463340997	3.8857	
2	1	GLUC	3.7288569046	3.8857	
3	1	GLUC	4.5676621504	3.8857	
4	1	GLUC	6.1226277479	3.8857	
5	1	GLUC	5.3322204975	3.8857	
6	1	GLUC	6.3083869808	3.8857	
7	1	GLUC	5.2967146997	3.8857	
8	1	GLUC	4.7909099201	3.8857	
9	1	GLUC	2.5299413569	3.8857	Y
10	1	GLUC	3.5584941976	3.8857	Y
11	1	GLUC	3.3916498243	3.8857	
12	1	GLUC	6.7630592416	3.8857	



Method 1: Sort data in reverse and utilise lag operator

- The lag operator allows you retrieve a value from a previous record
- This does not the flag the first in a series of consecutive low events so more processing is required
- The dataset can be sorted into reverse chronological order and the same logic applied again to flag the records that were missed in the first pass

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	6.7630592416	3.8857	
2	1	GLUC	3.3916498243	3.8857	
3	1	GLUC	3.5584941976	3.8857	Y
4	1	GLUC	2.5299413569	3.8857	Y
5	1	GLUC	4.7909099201	3.8857	
6	1	GLUC	5.2967146997	3.8857	
7	1	GLUC	6.3083869808	3.8857	
8	1	GLUC	5.3322204975	3.8857	
9	1	GLUC	6.1226277479	3.8857	
10	1	GLUC	4.5676621504	3.8857	
11	1	GLUC	3.7288569046	3.8857	
12	1	GLUC	4.5463340997	3.8857	

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	4.5463340997	3.8857	
2	1	GLUC	3.7288569046	3.8857	
3	1	GLUC	4.5676621504	3.8857	
4	1	GLUC	6.1226277479	3.8857	
5	1	GLUC	5.3322204975	3.8857	
6	1	GLUC	6.3083869808	3.8857	
7	1	GLUC	5.2967146997	3.8857	
8	1	GLUC	4.7909099201	3.8857	
9	1	GLUC	2.5299413569	3.8857	Y
10	1	GLUC	3.5584941976	3.8857	Y
11	1	GLUC	3.3916498243	3.8857	Y
12	1	GLUC	6.7630592416	3.8857	



Method 1: Sort data in reverse and utilise lag operator

- The lag operator allows you retrieve a value from a previous record
- This does not the flag the first in a series of consecutive low events so more processing is required
- The dataset can be sorted into reverse chronological order and the same logic applied again to flag the records that were missed in the first pass
- The dataset is then re-sorted into chronological order to produce the final result dataset

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	6.7630592416	3.8857	
2	1	GLUC	3.3916498243	3.8857	Y
3	1	GLUC	3.5584941976	3.8857	Y
4	1	GLUC	2.5299413569	3.8857	Y
5	1	GLUC	4.7909099201	3.8857	
6	1	GLUC	5.2967146997	3.8857	
7	1	GLUC	6.3083869808	3.8857	
8	1	GLUC	5.3322204975	3.8857	
9	1	GLUC	6.1226277479	3.8857	
10	1	GLUC	4.5676621504	3.8857	
11	1	GLUC	3.7288569046	3.8857	
12	1	GLUC	4.5463340997	3.8857	



Method 2: Reflexive offset merge combined with lag operator

- Using the automatic variable `_n_`, variables are created for the current and previous row numbers
- The variables in the offset dataset are renamed during the match merge to produce a wider dataset with the future value as an additional variable
- Future values are read horizontally in the `futureLbstresn` variable and the lag operator can be used to read previous records

	⚠ SUBJECT	⚠ LBTESTCD	🕒 LBSTRESN	🕒 LBSTNRLO	🕒 currentRow	🕒 mergeIndex
1	1	GLUC	6.7630592416	3.8857	1	0
2	1	GLUC	3.3916498243	3.8857	2	1
3	1	GLUC	3.5584941976	3.8857	3	2
4	1	GLUC	2.5299413569	3.8857	4	3
5	1	GLUC	4.7909099201	3.8857	5	4
6	1	GLUC	5.2967146997	3.8857	6	5
7	1	GLUC	6.3083869808	3.8857	7	6
8	1	GLUC	5.3322204975	3.8857	8	7
9	1	GLUC	6.1226277479	3.8857	9	8
10	1	GLUC	4.5676621504	3.8857	10	9
11	1	GLUC	3.7288569046	3.8857	11	10
12	1	GLUC	4.5463340997	3.8857	12	11

	⚠ SUBJECT	⚠ LBTESTCD	🕒 LBSTRESN	🕒 LBSTNRLO	🕒 futureLbstresn	⚠ consecutiveLowFlag	🕒 currentRow
1	1	GLUC	6.7630592416	3.8857	3.3916498243		1
2	1	GLUC	3.3916498243	3.8857	3.5584941976	Y	2
3	1	GLUC	3.5584941976	3.8857	2.5299413569	Y	3
4	1	GLUC	2.5299413569	3.8857	4.7909099201	Y	4
5	1	GLUC	4.7909099201	3.8857	5.2967146997		5
6	1	GLUC	5.2967146997	3.8857	6.3083869808		6
7	1	GLUC	6.3083869808	3.8857	5.3322204975		7
8	1	GLUC	5.3322204975	3.8857	6.1226277479		8
9	1	GLUC	6.1226277479	3.8857	4.5676621504		9
10	1	GLUC	4.5676621504	3.8857	3.7288569046		10
11	1	GLUC	3.7288569046	3.8857	4.5463340997		11
12	1	GLUC	4.5463340997	3.8857	8.2936704699		12



Method 2: Reflexive offset merge combined with lag operator

- Using the automatic variable `_n_`, variables are created for the current and previous row numbers
- The variables in the offset dataset are renamed during the match merge to produce a wider dataset with the future value as an additional variable
- Future values are read horizontally in the `futureLbstresn` variable and the lag operator can be used to read previous records

	△ SUBJECT	△ LBTESTCD	LBSTRESN	LBSTNRLO	currentRow	mergeIndex
1	1	GLUC	6.7630592416	3.8857	1	0
2	1	GLUC	3.3916498243	3.8857	2	1
3	1	GLUC	3.5584941976	3.8857	3	2
4	1	GLUC	2.5299413569	3.8857	4	3
5	1	GLUC	4.7909099201	3.8857	5	4
6	1	GLUC	5.2967146997	3.8857	6	5
7	1	GLUC	6.3083869808	3.8857	7	6
8	1	GLUC	5.3322204975	3.8857	8	7
9	1	GLUC	6.1226277479	3.8857	9	8
10	1	GLUC	4.5676621504	3.8857	10	9
11	1	GLUC	3.7288569046	3.8857	11	10
12	1	GLUC	4.5463340997	3.8857	12	11

	△ SUBJECT	△ LBTESTCD	LBSTRESN	LBSTNRLO	futureLbstresn	consecutiveLowFlag	currentRow
1	1	GLUC	6.7630592416	3.8857	3.3916498243		1
2	1	GLUC	3.3916498243	3.8857	3.5584941976	Y	2
3	1	GLUC	3.5584941976	3.8857	2.5299413569	Y	3
4	1	GLUC	2.5299413569	3.8857	4.7909099201	Y	4
5	1	GLUC	4.7909099201	3.8857	5.2967146997		5
6	1	GLUC	5.2967146997	3.8857	6.3083869808		6
7	1	GLUC	6.3083869808	3.8857	5.3322204975		7
8	1	GLUC	5.3322204975	3.8857	6.1226277479		8
9	1	GLUC	6.1226277479	3.8857	4.5676621504		9
10	1	GLUC	4.5676621504	3.8857	3.7288569046		10
11	1	GLUC	3.7288569046	3.8857	4.5463340997		11
12	1	GLUC	4.5463340997	3.8857	8.2936704699		12



Method 2: Reflexive offset merge combined with lag operator

- Using the automatic variable `_n_`, variables are created for the current and previous row numbers
- The variables in the offset dataset are renamed during the match merge to produce a wider dataset with the future value as an additional variable
- Future values are read horizontally in the `futureLbstresn` variable and the lag operator can be used to read previous records

	△ SUBJECT	△ LBTESTCD	🕒 LBSTRESN	🕒 LBSTNRLO	🕒 currentRow	🕒 mergeIndex
1	1	GLUC	6.7630592416	3.8857	1	0
2	1	GLUC	3.3916498243	3.8857	2	1
3	1	GLUC	3.5584941976	3.8857	3	2
4	1	GLUC	2.5299413569	3.8857	4	3
5	1	GLUC	4.7909099201	3.8857	5	4
6	1	GLUC	5.2967146997	3.8857	6	5
7	1	GLUC	6.3083869808	3.8857	7	6
8	1	GLUC	5.3322204975	3.8857	8	7
9	1	GLUC	6.1226277479	3.8857	9	8
10	1	GLUC	4.5676621504	3.8857	10	9
11	1	GLUC	3.7288569046	3.8857	11	10
12	1	GLUC	4.5463340997	3.8857	12	11

	△ SUBJECT	△ LBTESTCD	🕒 LBSTRESN	🕒 LBSTNRLO	🕒 futureLbstresn	△ consecutiveLowFlag	🕒 currentRow
1	1	GLUC	6.7630592416	3.8857	3.3916498243		1
2	1	GLUC	3.3916498243	3.8857	3.5584941976	Y	2
3	1	GLUC	3.5584941976	3.8857	2.5299413569	Y	3
4	1	GLUC	2.5299413569	3.8857	4.7909099201	Y	4
5	1	GLUC	4.7909099201	3.8857	5.2967146997		5
6	1	GLUC	5.2967146997	3.8857	6.3083869808		6
7	1	GLUC	6.3083869808	3.8857	5.3322204975		7
8	1	GLUC	5.3322204975	3.8857	6.1226277479		8
9	1	GLUC	6.1226277479	3.8857	4.5676621504		9
10	1	GLUC	4.5676621504	3.8857	3.7288569046		10
11	1	GLUC	3.7288569046	3.8857	4.5463340997		11
12	1	GLUC	4.5463340997	3.8857	8.2936704699		12



Method 3: Static hash object with all records

- A hash object is a table that exists in memory for the duration of the data step
- All values for the result variable are loaded into a hash object
- A **hash iterator** is used to cycle through the hash object to read the previous and future values
- This creates the required result in just a single data step

Read from disk						Read from memory
	⚠ SUBJECT	⚠ LBTESTCD	🔍 LBSTRESN	🔍 LBSTNRLO	⚠ consecutiveLowFlag	🔍 dynamicLbstrsn
1	1	GLUC	6.7630592416	3.8857		6.7630592416
2	1	GLUC	3.3916498243	3.8857	Y	3.3916498243
3	1	GLUC	3.5584941976	3.8857	Y	3.5584941976
4	1	GLUC	2.5299413569	3.8857	Y	2.5299413569
5	1	GLUC	4.7909099201	3.8857		4.7909099201
6	1	GLUC	5.2967146997	3.8857		5.2967146997
7	1	GLUC	6.3083869808	3.8857		6.3083869808
8	1	GLUC	5.3322204975	3.8857		5.3322204975
9	1	GLUC	6.1226277479	3.8857		6.1226277479
10	1	GLUC	4.5676621504	3.8857		4.5676621504
11	1	GLUC	3.7288569046	3.8857		3.7288569046
12	1	GLUC	4.5463340997	3.8857		4.5463340997



Method 4: Dynamic hash object containing 50,000 records at a time

- The automatic variable `_n_` is used to create a dynamic `where=` option in the hash object to load data from 50,000 records
- The **hash iterator** object is used to observe previous and future values as per Method 3
- When the end of the hash object is reached, the hash object is deleted and re-declared with the next 50,000 records

Read from disk

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO
1	1	GLUC	6.7630592416	3.8857
2	1	GLUC	3.3916498243	3.8857
3	1	GLUC	3.5584941976	3.8857
4	1	GLUC	2.5299413569	3.8857
5	1	GLUC	4.7909099201	3.8857
6	1	GLUC	5.2967146997	3.8857
7	1	GLUC	6.3083869808	3.8857
8	1	GLUC	5.3322204975	3.8857
9	1	GLUC	6.1226277479	3.8857
10	1	GLUC	4.5676621504	3.8857
11	1	GLUC	3.7288569046	3.8857
12	1	GLUC	4.5463340997	3.8857

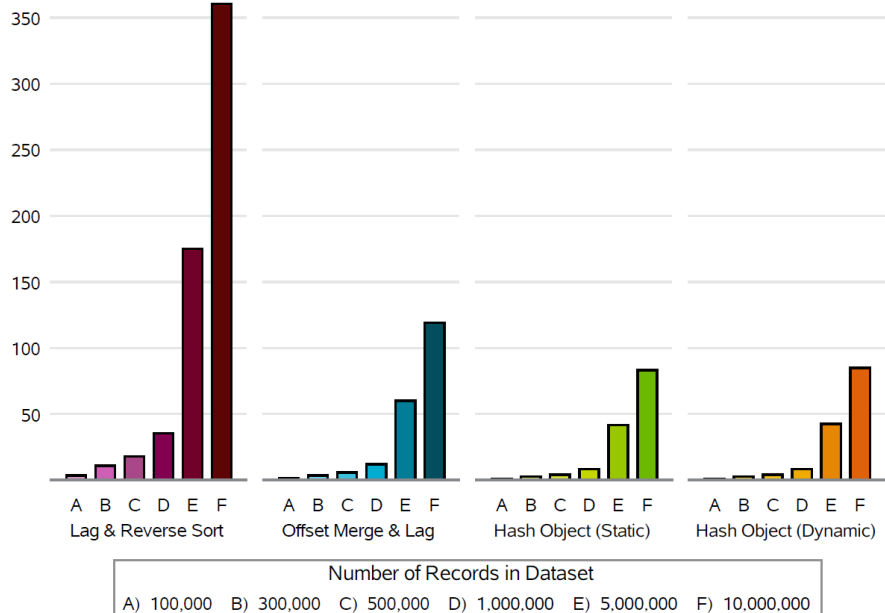
Read from memory

 dynamicLbstresn
6.7630592416
3.3916498243
3.5584941976
2.5299413569
4.7909099201
5.2967146997

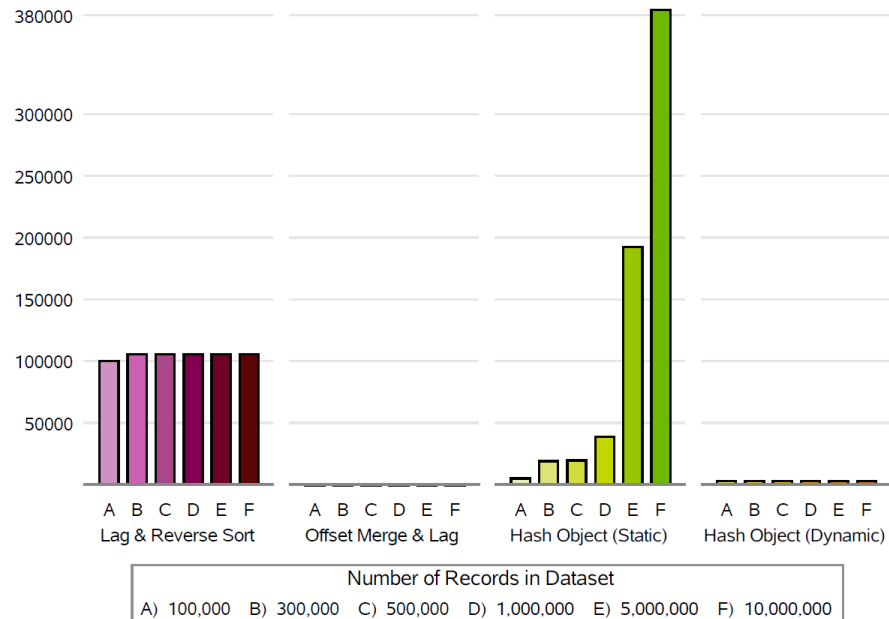


Performance Comparison

Comparison of Execution Time (Seconds)



Comparison of Peak Memory Usage (Megabytes)



Conclusion

Ranking				
Best ↑ ↓ Worst	Complexity (Least complex first)	Execution Time (Fastest first)		Memory Usage (Lowest memory first)
	Reverse Sort	Static Hash Object	Dynamic Hash Object	Offset Merge
	Offset Merge			Dynamic Hash Object
	Static Hash Object	Offset Merge		Reverse Sort
	Dynamic Hash Object	Reverse Sort		Static Hash Object



Questions?



Method 1 (Code): Sort data in reverse and utilise lag operator

* Using lag operator

```
lagUsubjid = lag(usubjid);  
lagLbtestcd = lag(lbtestcd);  
lagLbstresn = lag(lbstresn);
```

* Checking future obs is within usubjid*lbtestcd group;

if usubjid = lagUsubjid and lbtestcd = lagLbtestcd then do;

if . < lbstresn < lbstnrlo then do;

if . < lagLbstresn < lbstnrlo then
consecutiveLowFlag = "Y";
end;

end;

* Sorting data in reverse chronological order;

```
proc sort data = labFirstPass out = labFirstPassRevSort;  
by usubjid lbtestcd descending lbdtc;  
run;
```

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	6.7630592416	3.8857	
2	1	GLUC	3.3916498243	3.8857	
3	1	GLUC	3.5584941976	3.8857	Y
4	1	GLUC	2.5299413569	3.8857	Y
5	1	GLUC	4.7909099201	3.8857	
6	1	GLUC	5.2967146997	3.8857	
7	1	GLUC	6.3083869808	3.8857	
8	1	GLUC	5.3322204975	3.8857	
9	1	GLUC	6.1226277479	3.8857	
10	1	GLUC	4.5676621504	3.8857	
11	1	GLUC	3.7288569046	3.8857	
12	1	GLUC	4.5463340997	3.8857	

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	4.5463340997	3.8857	
2	1	GLUC	3.7288569046	3.8857	
3	1	GLUC	4.5676621504	3.8857	
4	1	GLUC	6.1226277479	3.8857	
5	1	GLUC	5.3322204975	3.8857	
6	1	GLUC	6.3083869808	3.8857	
7	1	GLUC	5.2967146997	3.8857	
8	1	GLUC	4.7909099201	3.8857	
9	1	GLUC	2.5299413569	3.8857	Y
10	1	GLUC	3.5584941976	3.8857	Y
11	1	GLUC	3.3916498243	3.8857	
12	1	GLUC	6.7630592416	3.8857	



Method 2 (Code): Reflexive offset merge combined with lag operator

* Create row number variable to be used for merging;

```
data labRowData;
  set originalLab;
  currentRow = _n_;
  mergeIndex = _n_ - 1;
run;
```

* Merging next row onto current row;

```
data finalLabDataMethod2 (drop=future: lag:)
  merge labRowData (drop=mergeIndex in=a)
    labRowData (keep = usubjid lbtestcd lbstresn
      mergeIndex
      rename = (usubjid=futureUsubjid
        lbtestcd=futureLbtestcd
        lbstresn=futureLbstresn
        mergeIndex=currentRow)
    );
by currentRow;
```

	⚠ SUBJECT	⚠ LBTESTCD	🔍 LBSTRESN	🔍 LBSTNRLO	🔍 currentRow	🔍 mergeIndex
1	1	GLUC	6.7630592416	3.8857	1	0
2	1	GLUC	3.3916498243	3.8857	2	1
3	1	GLUC	3.5584941976	3.8857	3	2
4	1	GLUC	2.5299413569	3.8857	4	3
5	1	GLUC	4.7909099201	3.8857	5	4
6	1	GLUC	5.2967146997	3.8857	6	5
7	1	GLUC	6.3083869808	3.8857	7	6
8	1	GLUC	5.3322204975	3.8857	8	7
9	1	GLUC	6.1226277479	3.8857	9	8
10	1	GLUC	4.5676621504	3.8857	10	9
11	1	GLUC	3.7288569046	3.8857	11	10
12	1	GLUC	4.5463340997	3.8857	12	11

	⚠ SUBJECT	⚠ LBTESTCD	🔍 LBSTRESN	🔍 LBSTNRLO	🔍 futureLbstresn	⚠ consecutiveLowFlag	🔍 currentRow
1	1	GLUC	6.7630592416	3.8857	3.3916498243		1
2	1	GLUC	3.3916498243	3.8857	3.5584941976	Y	2
3	1	GLUC	3.5584941976	3.8857	2.5299413569	Y	3
4	1	GLUC	2.5299413569	3.8857	4.7909099201	Y	4
5	1	GLUC	4.7909099201	3.8857	5.2967146997		5
6	1	GLUC	5.2967146997	3.8857	6.3083869808		6
7	1	GLUC	6.3083869808	3.8857	5.3322204975		7
8	1	GLUC	5.3322204975	3.8857	6.1226277479		8
9	1	GLUC	6.1226277479	3.8857	4.5676621504		9
10	1	GLUC	4.5676621504	3.8857	3.7288569046		10
11	1	GLUC	3.7288569046	3.8857	4.5463340997		11
12	1	GLUC	4.5463340997	3.8857	8.2936704699		12



Method 3 (Code): Static hash object with all records

* Creating hash object;

if _n_ = 1 then do;

```
dcl hash crystalBall(dataset:catt("originalLab (keep = usubjid lbtestcd lbstresn lbdtc"
    , "rename = (usubjid=dynamicUsubjid lbtestcd=dynamicLbtestcd"
    , "lbstresn=dynamicLbstresn lbdtc=dynamicLbdtc))"
    )
    ,ordered:"a"
);
crystalBall.defineKey("dynamicUsubjid","dynamicLbtestcd","dynamicLbdtc");
crystalBall.defineData(all:"yes");
crystalBall.defineDone();
```

* Creating hash iterator object to traverse multiple observations;

```
dcl hiter labTraverse("crystalBall");
```

end;

Read from disk

	 SUBJECT	 LBTESTCD	 LBSTRESN	 LBSTNRLO	 consecutiveLowFlag
1	1	GLUC	6.7630592416	3.8857	
2	1	GLUC	3.3916498243	3.8857	Y
3	1	GLUC	3.5584941976	3.8857	Y
4	1	GLUC	2.5299413569	3.8857	Y
5	1	GLUC	4.7909099201	3.8857	
6	1	GLUC	5.2967146997	3.8857	
7	1	GLUC	6.3083869808	3.8857	
8	1	GLUC	5.3322204975	3.8857	
9	1	GLUC	6.1226277479	3.8857	
10	1	GLUC	4.5676621504	3.8857	
11	1	GLUC	3.7288569046	3.8857	
12	1	GLUC	4.5463340997	3.8857	

Read from
memory

 dynamicLbstresn
6.7630592416
3.3916498243
3.5584941976
2.5299413569
4.7909099201
5.2967146997
6.3083869808
5.3322204975
6.1226277479
4.5676621504
3.7288569046
4.5463340997



Method 4 (Code): Dynamic hash object containing 50,000 records at a time

* Creating hash object of next 50000 records;
if `_n_ = 1` or `newHashRequired = "Y"` then do;

```
dcl hash crystalBall(dataset:catt("originalLab (keep = usubjid lbtestcd lbstresn lbdtc"
    , " rename = (usubjid=dynamicUsubjid lbtestcd=dynamicLbtestcd"
    , " lbstresn=dynamicLbstresn lbdtc=dynamicLbdtc)"
    , " firstObs=", _n_, " obs=", _n_ + 50000, ")")
    ,ordered:"a"
);
crystalBall.defineKey("dynamicUsubjid","dynamicLbtestcd","dynamicLbdtc");
crystalBall.defineData(all:"yes");
crystalBall.defineDone();
```

* Creating hash iterator object to traverse multiple observations;
dcl hiter labTraverse("crystalBall");

* Resetting flag requesting next hash object to be created;
newHashRequired = "";

end;

* Deleting main hash object prior to recreating it with the next 50000 records;
if `newHashRequired = "Y"` then do;
rc = crystalBall.delete();
rc = labTraverse.delete();
end;

Read from disk

	⚠ SUBJECT	⚠ LBTESTCD	🔍 LBSTRESN	🔍 LBSTNRLO
1	1	GLUC	6.7630592416	3.8857
2	1	GLUC	3.3916498243	3.8857
3	1	GLUC	3.5584941976	3.8857
4	1	GLUC	2.5299413569	3.8857
5	1	GLUC	4.7909099201	3.8857
6	1	GLUC	5.2967146997	3.8857
7	1	GLUC	6.3083869808	3.8857
8	1	GLUC	5.3322204975	3.8857
9	1	GLUC	6.1226277479	3.8857
10	1	GLUC	4.5676621504	3.8857
11	1	GLUC	3.7288569046	3.8857
12	1	GLUC	4.5463340997	3.8857

First hash object

🔍 dynamicLbstresn
6.7630592416
3.3916498243
3.5584941976
2.5299413569
4.7909099201
5.2967146997

Next hash object

6.3083869808
5.3322204975
6.1226277479
4.5676621504
3.7288569046
4.5463340997

