Paper 79-27

Taking Advantage of Missing Values in Proc SQL

Ya Huang
Pfizer Global R&D, La Jolla Laboratories, San Diego, CA

ABSTRACT

PROC SQL provides many numerical summary functions, such as SUM, MEAN, MAX, MIN etc. Most summary functions do not count missing values. I consider it a good feature, because it makes 'Comparison Between Observations' possible. In other words, it makes 'Longitudinal Data Processing' using PROC SQL possible.

INTRODUCTION

Most people know how to use PROC SQL summary function. But not all of them may be aware of how missing values are handled in summary functions. This paper gives a few examples and discusses how a typical summary function works, how it handles a missing value, and most of all why the way it handles missing value can be taken advantage. It also briefly discusses other features, such as "Remerging" and how it can save steps.

WHAT DOES A SUMMARY FUNCTION DO?

A summary function in PROC SQL is different from its counterpart in data step. A data step function summarizes columns (or variables) in one row (observation). It usually has multiple arguments, such as MEAN(A, B ,C) or MEAN(OF A1-A10). A summary function in PROC SQL, on the contrary, summarizes rows (or observations) in one column (variable). It has only one argument. A summary function is usually used with a 'GROUP BY' clause. It tells the summary function what observations are in the same group for summary. If no 'GROUP BY' clause is present, a summary function treats all observation as one 'By group'.

```
DATA XX;
INPUT A B C;
CARDS;
1 . 3
2 . 6
. . 7
2 5 2
5 . 3
4 . 9
;

PROC SQL;
SELECT *, MEAN(A) AS MEANA, MAX(B) AS MAXB,
 A-CALCULATED MEANA AS DIFF
FROM XX
;

A  B  C  MEANA  MAXB  DIFF

1  .  3   2.8     5   -1.8
2  .  6   2.8     5   -0.8
.  .  7   2.8     5     .
2  5  2   2.8     5   -0.8
5  .  3   2.8     5    2.2
4  .  9   2.8     5    1.2
```

Because no 'By group' is present in this example, summaries are done for all observations. Here `MEANA` and `MAXB` are generated by `MEAN(A)` and `MAX(B)` respectively. Note that their values are consistent across the observations. It is a result of 'Remerging'.

WHAT IS "REMERGING"?

Remerging is a two-pass process: The first pass, Proc SQL calculates and returns the result of a summary function. In the second pass it merges the result back to original data and creates a new variable. When the new variable is merged back to original data, its value is retained within the 'By group'. In the above example, the values of `MEANA` and `MAXB` are retained across all observations. Once you get the remerged result, you can further derive variables based on that result. In the above example, variable `DIFF` is a subtraction of `A` and `MEANA`. To use a summary result, you need the key word 'CALCULATED'.

Instead of using Proc sort then Proc means to get the summary result and merge the data set back to original data, I used one PROC SQL step, saved myself two steps. In fact, more steps can be saved when things get complicated.

HOW A MISSING VALUE IS HANDLED?

Summary functions do not count missing values. This is shown in the above example: Note that `MEANA=2.8=(1+2+2+5+4)/5`, rather than `MEANA=2.33=(1+2+2+5+4)/6`. The reason it is divided by `5` rather than `6`, the number of observations, is that one missing value is excluded from the calculation.

When a variable has only one non-missing value, the summary function returns that value. In the above example variable `B` is deliberately set to have only one non-missing value `5`. As you can see, `MAXB=5`.

In some cases, you may need to get the statistics for a portion of the variable's values within a 'By group'. The standard way of using summary function won't work because it is for the whole 'By group'. You can do it with the help of missing values. The idea is to keep the values you want and force the values you don't want to missing.

CASE EXPRESSION TO SELECT RESULT

A CASE expression selects values if certain conditions are met. It can be used alone to create a variable or as the argument in a summary function. ELSE can be used to assign values when conditions are not met. I use it to force those unwanted values to missing.

```
PROC SQL;
SELECT A,
CASE WHEN MOD(A,2)=0 THEN A ELSE . END AS
EVENA,
MEAN(CALCULATED EVENA) AS M1,
MEAN(CASE WHEN MOD(A,2)=0 THEN A ELSE . END)
AS M2
FROM XX
;

A  EVENA     M1       M2

1    .    2.66667  2.66667
2    2    2.66667  2.66667
.    .    2.66667  2.66667
2    2    2.66667  2.66667
5    .    2.66667  2.66667
4    4    2.66667  2.66667
```

In this example, two CASE expressions are used. One is used alone to create a variable called `EVENA`. As you can see, `EVENA` is a subset of `A`. It contains even number from `A`. `EVENA` is later passed as an argument to function MEAN to get `M1`. An identical CASE expression is embedded in another function `MEAN` and the result is `M2`. As you can see, `M1=M2`. Obviously, embedding a CASE expression in a summary function is better, since it dose not generate an intermediate variable.

LOCATING OBSERVATIONS FOR SUMMARY

The above example calculates the average of all even values from variable `A`. It shows how you can pick values within a 'By group', or in other words, how you can locate those observations whose variable `A`'s values are even number. In general, you

might not be interested in variable A itself, but rather something related to it. For example, you might be interested in the average value of variable C, when A is an even number. In this situation, variable A can serve as an index variable to locate the observations and CASE expression will return variable C's value to function MEAN.

```
PROC SQL;
SELECT A, SUM(CASE WHEN MOD(A,2)=0 THEN C ELSE
. END) AS SUMC
FROM XX
;

A    C    SUMC

1    3     17
2    6     17
.    7     17
2    2     17
5    3     17
4    9     17
```

WHY NOT USE A WHERE CLAUSE?
You may think you can get the same result by simply using a WHERE clause such as:

```
PROC SQL;
SELECT A, SUM(C) AS SUMC
FROM XX
WHERE MOD(A,2)=0
;
```

There are two reasons the WHERE clause is not recommended:
1. The WHERE clause subsets the data, so that the resulting table only contains the observations meet the WHERE clause condition. A lot of time, you may need to keep all the observations, so that they can be compared with the summary result.
2. If you need to get the summary from two different observations groups and compares with each other, one WHERE clause wouldn't be able to distinguish them.

EXAMPLE 1: VARIABLE HAS A SPECIFIC VALUE THAT CAN BE USED TO LOCATE AN OBSERVATION
The following is an example of patient vital signs data. In the data, each patient has many observations of blood pressure recorded on different visits. I want to compare the blood pressure at each visit with the blood pressure at baseline. Here the baseline observation is the one that variable VISIT has a value of 'baseline'. Each ID has only one baseline observation.

```
DATA VITALS;
INPUT ID VISIT $ BP;
CARDS;
1 BASELINE 140
1 WK1      120
1 WK2      130
1 WK3      110
1 WK4      120
2 BASELINE 145
2 WK1      115
2 WK2      125
2 WK3      135
2 WK4      120
;

PROC SQL;
SELECT *,
MAX(CASE WHEN COURSE='BASELINE' THEN BP ELSE .
END) AS BASEBP,
BP-CALCULATED BASEBP AS DIFF
FROM VITALS
GROUP BY ID
ORDER BY ID, COURSE
;

ID  VISIT     BP  BASEBP   DIFF

 1  BASELINE  140     140      0
 1  WK1       120     140    -20
 1  WK2       130     140    -10
 1  WK3       110     140    -30
 1  WK4       120     140    -20
 2  BASELINE  145     145      0
 2  WK1       115     145    -30
 2  WK2       125     145    -20
 2  WK3       135     145    -10
 2  WK4       120     145    -25
```

As I mentioned before, if a variable has only one non-missing value, the summary function returns that value. In this example, each ID has only one observation that meets the condition of COURSE='BASELINE'. The CASE expression locates that observation and MAX returns the baseline value of BP. In this case, MAX is not meant to get the maximum. It is actually a tool to fetch a specific value. For this purpose, you can also use SUM, MIN or even MEAN.

WHY CASE EXPRESSION?
The code above could be even shorter:

```
PROC SQL;
SELECT *,
MAX((COURSE='BASELINE')*BP) AS BASEBP,
BP-CALCULATED BASEBP AS DIFF
FROM XX
GROUP BY ID
ORDER BY ID, COURSE
;
```

COURSE='BASELINE' as a Boolean test returns either 0 or 1. Therefore, (COURSE='BAELINE')*BP resolves to either zero (not baseline) or BP value (baseline). Since blood pressure is always a positive number, MAX returns it after compared with all those zeros. In general, I don't recommend using this technique. Because if a variable has negative value, this technique will fail to pick up the negative value, in stead it returns a zero!

EXAMPLE 2: VARIABLE DOES NOT HAVE A SPECIFIC VALUE THAT CAN BE USED TO LOCATE OBSERVATION
The following example is similar to the previous one, but it has no variable like VISIT that explicitly defines the baseline observation. In this example, the index variable is perform date, PERFDT. Baseline is defined as the last perform date prior to the first dose date, FSDOSDT.

```
DATA VITALS;
INPUT ID BP PERFDT :DATE. FSTDOSDT :DATE.;
FORMAT PERFDT FSTDOSDT DATE.;
CARDS;
01 122 16AUG01 05SEP01
01 133 04SEP01 05SEP01
01 119 05SEP01 05SEP01
01 115 13SEP01 05SEP01
01 113 18SEP01 05SEP01
02 121 17AUG01 06SEP01
02 122 05SEP01 06SEP01
02 121 06SEP01 06SEP01
02 110 14SEP01 06SEP01
02 108 19SEP01 06SEP01
;

PROC SQL;
CREATE TABLE VITALS AS
SELECT *,
BP-MAX(CASE WHEN PERFDT=PRED THEN BP ELSE .
END) AS DIFF
FROM (SELECT *,
      MAX(CASE WHEN PERFDT < FSTDOSDT THEN
      PERFDT ELSE . END) AS PRED FORMAT=DATE.
      FROM VITALS
      GROUP BY ID)
GROUP BY ID
ORDER BY ID,PERFDT;

ID   BP  PERFDT   FSTDOSDT   PRED   DIFF

 1  122  16AUG01  05SEP01   04SEP01  -11
 1  133  04SEP01  05SEP01   04SEP01    0
 1  119  05SEP01  05SEP01   04SEP01  -14
 1  115  13SEP01  05SEP01   04SEP01  -18
```

```
 1  113  18SEP01  05SEP01   04SEP01  -20
 2  121  17AUG01  06SEP01   05SEP01   -1
 2  122  05SEP01  06SEP01   05SEP01    0
 2  121  06SEP01  06SEP01   05SEP01   -1
 2  110  14SEP01  06SEP01   05SEP01  -12
 2  108  19SEP01  06SEP01   05SEP01  -14
```

In the example, a subquery returns all the original data, and adds a new variables PRED, which has the value of the last date before FSTDOSDT. The result of the subquery is then used as the source data for another query. In the second query, PREDT and PERFDT are compared to locate baseline observations. Finally MAX function fetches the BP value at baseline. A subtraction of BP and the result of MAX function is called DIFF.

EXAMPLE 3: LOCATING OBSERVATIONS BEYOND THE 'BY GROUP'

In the very first example, I mentioned that if no 'By group' is involved, a summary function treats all observations as one group. In some situations, you may need to treat all observations as one group, even if there is some 'By group' variables, because you need to get summary across a few 'By groups'.

The following example has three variables: WDATE, WTIME and WCONSUME. What I want to do is, for each day:

- Get the maximum value of WCONSUME from 9:00pm of the previous day to 8:00am of that day as A.
- Get the minimum value of WCONSUME from 8:00am to 9:00pm of that day as B.
- Get the maximum value of WCONSUME from 9:01pm of that day to 8:00 am of the day after as C.
- Derive a new variable R=(C-B)/16.
- Derive a new variable WATERUSE =(A-B)+16*R.

```
DATA XX;
INPUT WDATE :MMDDYY8. WTIME :TIME. WCONSUME @@;
CARDS;
4/11/01 23:00   2.84214  4/12/01 0:00    2.85152
4/12/01 1:00    2.85621  4/12/01 2:00    2.8609
4/12/01 3:00    2.8609   4/12/01 4:00    2.85621
4/12/01 5:00    2.87028  4/12/01 6:00    2.87497
4/12/01 7:00    2.87966  4/12/01 8:00    2.87966
4/12/01 9:00    2.88435  4/12/01 10:00   2.88904
4/12/01 11:00   2.88904  4/12/01 12:00   2.88904
4/12/01 13:00   2.87966  4/12/01 14:00   2.88904
4/12/01 15:00   2.88904  4/12/01 16:00   2.89842
4/12/01 17:00   2.90311  4/12/01 18:00   2.9078
4/12/01 19:00   2.90311  4/12/01 20:00   2.91249
4/12/01 21:00   2.91249  4/12/01 22:00   2.91249
4/12/01 23:00   2.91718  4/13/01 0:00    2.92187
4/13/01 1:00    2.92656  4/13/01 2:00    2.92656
4/13/01 3:00    2.93125  4/13/01 4:00    2.93594
4/13/01 5:00    2.94063  4/13/01 6:00    2.94532
4/13/01 7:00    2.94532  4/13/01 8:00    2.95001
4/13/01 9:00    2.9547   4/13/01 10:00   2.9547
4/13/01 11:00   2.9547   4/13/01 12:00   2.9547
4/13/01 13:00   2.9547   4/13/01 14:00   2.9547
4/13/01 15:00   2.95939  4/13/01 16:00   2.96408
4/13/01 17:00   2.96408  4/13/01 18:00   2.97346
4/13/01 19:00   2.97346  4/13/01 20:00   2.98284
4/13/01 21:00   2.98284  4/13/01 22:00   2.99222
4/13/01 23:00   2.99222  4/14/01 0:00    3.0016
4/14/01 1:00    2.99222  4/14/01 2:00    3.01098
4/14/01 3:00    3.02036  4/14/01 4:00    3.02974
4/14/01 5:00    3.03443  4/14/01 6:00    3.03912
4/14/01 7:00    3.0485   4/14/01 8:00    3.05788
4/14/01 9:00    3.06257  4/14/01 10:00   3.06726
4/14/01 11:00   3.07664  4/14/01 12:00   3.07664
4/14/01 13:00   3.07664  4/14/01 14:00   3.08602
4/14/01 15:00   3.08133  4/14/01 16:00   3.0954
4/14/01 17:00   3.10009  4/14/01 18:00   3.10947
4/14/01 19:00   3.11416  4/14/01 20:00   3.12354
4/14/01 21:00   3.12823  4/14/01 22:00   3.13761
4/14/01 23:00   3.13292  4/15/01 0:00    3.13292
;

PROC SQL;
 SELECT DISTINCT
   DAYZERO FORMAT=DATE.,
   MAX(CASE WHEN DAYZERO=WDATE+1 AND
                 WTIME >='21:00'T OR
                 DAYZERO=WDATE AND
                 WTIME <'8:00'T
            THEN WCONSUME
            ELSE . END) AS A,
   MIN(CASE WHEN DAYZERO=WDATE AND
                 '8:00'T <=WTIME <'21:00'T
            THEN WCONSUME
            ELSE . END) AS B,
   MAX(CASE WHEN DAYZERO=WDATE AND
                 WTIME >='21:00'T OR
                 DAYZERO=WDATE-1 AND
                 WTIME <'8:00'T
            THEN WCONSUME
            ELSE . END) AS C,
  (CALCULATED C - CALCULATED B)/16 AS R,
  (CALCULATED A - CALCULATED B)+ 16*(CALCULATED
      R) AS WATERUSE
  FROM (SELECT DISTINCT WDATE AS DAYZERO FROM
       XX), XX
  GROUP BY DAYZERO
  ORDER BY DAYZERO
;

DAYZERO       A        B       C       R   WATERUSE

11APR01    .        .     2.880    .        .
12APR01  2.880    2.880  2.945  0.004    0.066
13APR01  2.945    2.950  3.049  0.006    0.094
14APR01  3.049    3.058  3.138  0.005    0.070
15APR01  3.138     .       .       .        .
```

In this example, a SQL Cartesian join is used so that each unique day, DAYZERO can be compared with all the other days. For each DAYZERO, the summary functions go through all the observations, and CASE expression picks up the two consecutive days and times within a specific period. This example shows how you can do both 'Backward looking' and 'Forward looking' in PROC SQL.

Now a seemingly very complicated problem is resolved by a simple PROC SQL step, it otherwise may take several data steps and PROC steps.

CONCLUSION

Missing values usually are troublemakers, but in Proc SQL, they can be beneficial. By introducing missing values in summary function, you can do longitudinal data processing in much shorter SAS programs.

ACKNOWLEDGMENTS

I would like to thank my fellow SAS programmer Julian Mota and Karen Convery for reviewing this paper.

CONTACT INFORMATION

Ya Huang
Pfizer PGRD, La Jolla Laboratories
11085 Torreyana Road
San Diego, CA 92121-1104
Tel: 858 622-7588
Ya.Huang@pfizer.com