PharmaSUG 2018 - Paper QT-17

IF there is a Better Way than IF-THEN

Bob Tian, Anni Weng, KMK Consulting Inc.

ABSTRACT

In this paper, the author compares different methods for implementing piecewise constant functions (step functions) in SAS®. The author uses a simulated approach to measure the efficiencies of different methods in terms of CPU resource usage.

INTRODUCTION

Many SAS® programmers routinely find themselves facing the task of implementing piecewise functions such as classifying BMI scores, or grouping blood sugar levels.

While adopting an IF-THEN logic seems intuitive and effortless, as the size of real world data multitudes and the complexity of assignment increases, will this be the most efficient method? Is there a better way to achieve such tasks in SAS® environment? While there are a few papers discussing different methods of implementing piecewise classification [1] [2], little reference can be found evaluating the efficiencies of different approaches.

To find the best approach, we evaluate the performance of 2 different IF-THEN implementations, an implicit IF logic application, an IFC/IFN function, 2 different WHEN implementations, and a bespoke format method, in terms of computation time. A large simulated dataset was generated to approximate real word data. Each method was applied to the dataset for hundreds of cycles under similar CPU load. The average statistics will be used for the comparison.

The authors were genuinely surprised by their findings, and would like to share the results with the readers.

INPUT DATA

To simulate real world data, we have randomly generated two beta-distributions from 0 to 100 as the input datasets. We have chosen a bell shaped distribution ($\alpha = 2$, $\beta = 3$), as well as a bimodal distribution ($\alpha = 0.5$, $\beta = 0.5$). The histogram plot for the two input datasets are shown below in Figure 1 and Figure 2.

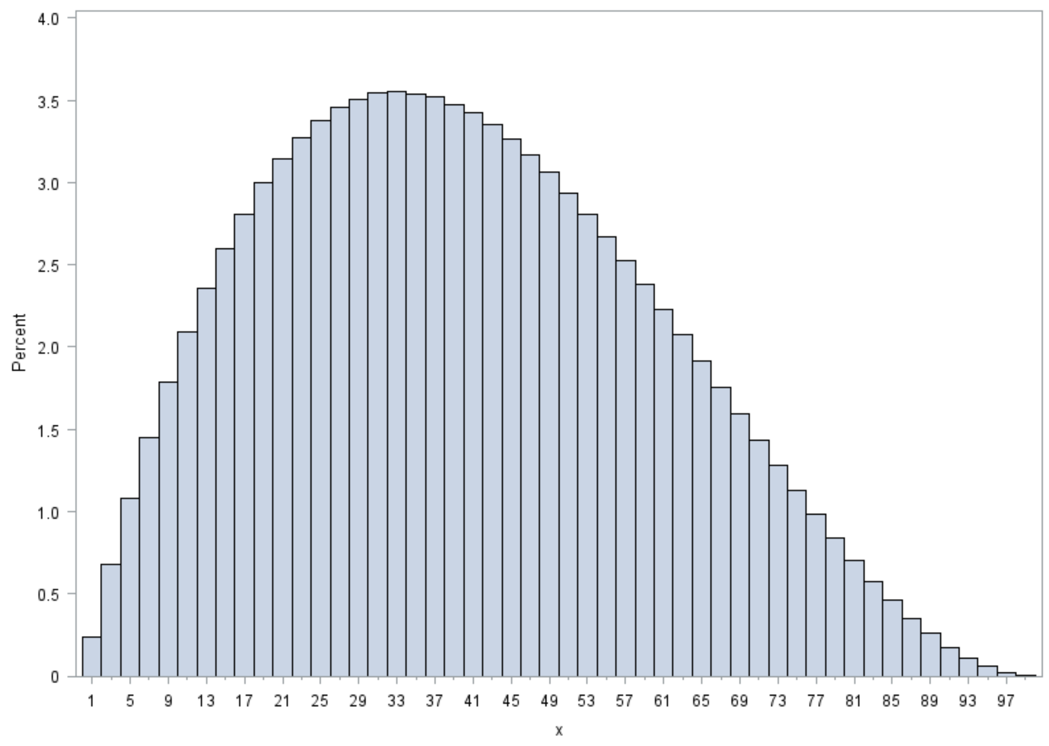

Figure 1. Histogram Plot of Input Distribution 1 Beta(2, 3)

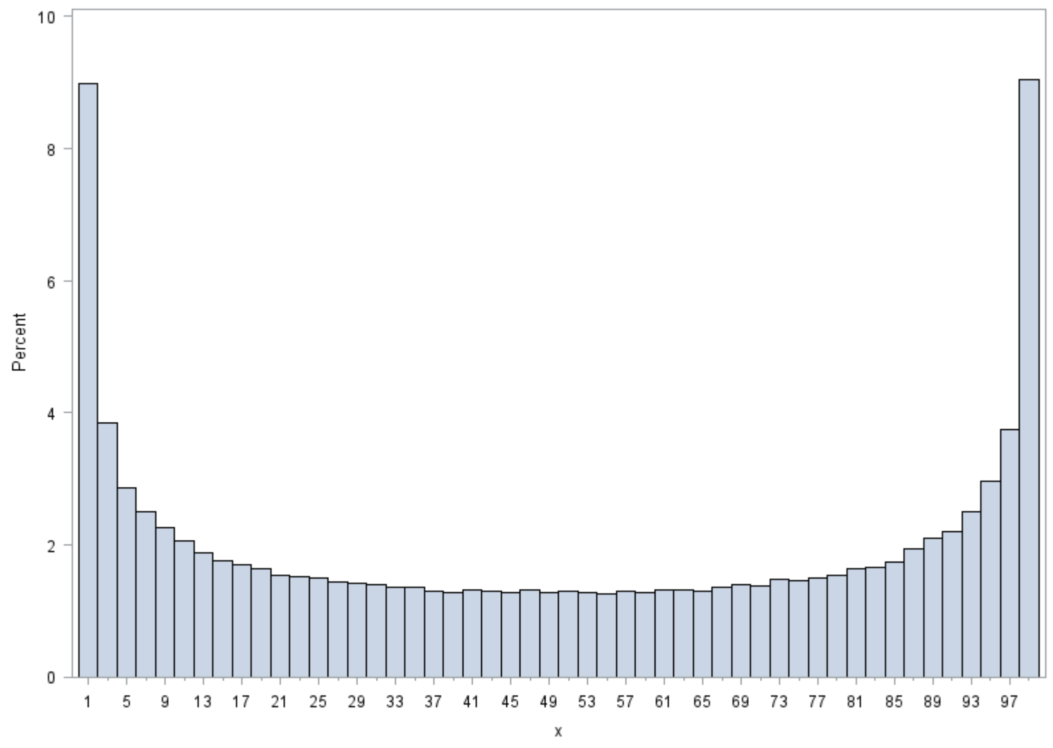

Figure 2. Histogram Plot of Input Distribution 2 Beta(0.5, 0.5)

These two datasets closely mirror many of the natural occurring data the authors have to deal with. We have chosen a sample size big enough for each calculation to take approximately 30 seconds.

METHODS

We used two different piecewise functions to compare the performance of different methods. One is a simple binary classification; the other has 6 mutually exclusive categories. These two classifications resemble some of the most common classifications the authors have to implement on a daily basis.

1. Function 1, 2 categories

$$y = \begin{cases} 0, & x < 50 \\ 1, & x \geq 50 \end{cases}$$

2. Function 2, 6 categories

$$y = \begin{cases} 1, & x < 10 \\ 2, & 10 \leq x < 20 \\ 3, & 20 \leq x < 50 \\ 4, & 50 \leq x < 80 \\ 5, & 80 \leq x < 90 \\ 6, & x \geq 90 \end{cases}$$

A total of 7 different methods were tested by the author. We will illustrate them using the second classification.

METHOD 1, A SIMPLE IF-THEN LOGIC

This is the most straight forward logic, most likely one of the first programs anyone learned to write:

```
DATA Method_1;
    SET inData;
          IF       x < 10 THEN y = 1;
          IF 10 <= x < 20 THEN y = 2;
          IF 20 <= x < 50 THEN y = 3;
          IF 50 <= x < 80 THEN y = 4;
          IF 80 <= x < 90 THEN y = 5;
          IF 90 <= x      THEN y = 6;
RUN;
```

METHOD 2, IF-THEN, ELSE -THEN LOGIC

Textbooks taught us using ELSE IF for mutually exclusive situations would be fast than only IF-THEN. We should now test this statement:

```
DATA Method_2;
    SET inData;
```

```
          IF              x < 10 THEN y = 1;
          ELSE IF 10  <= x < 20 THEN y = 2;
          ELSE IF 20  <= x < 50 THEN y = 3;
          ELSE IF 50  <= x < 80 THEN y = 4;
          ELSE IF 80  <= x < 90 THEN y = 5;
          ELSE y = 6;
RUN;
```

METHOD 3, AN IMPLICIT IF LOGIC

We can also use an implicit function to achieve the same IF logic. In fact, this is the author’s favorite method, as the code is very neat:

```
DATA Method_3;
    SET inData;
          y =   1 * (x < 10)
            + 2 * (10 <= x < 20)
            + 3 * (20 <= x < 50)
            + 4 * (50 <= x < 80)
            + 5 * (80 <= x < 90)
            + 6 * (90 <= x );
RUN;
```

METHOD 4, NESTED IFN/IFC FUNCTIONS

A nested IFN/IFC function can also be used. However, it can be sometimes difficult to read:

```
DATA Method_4;
    SET inData;
          y = ifn(x < 10, 1
            , ifn(10 <= x < 20, 2
            , ifn(20 <= x < 50, 3
            , ifn(50 <= x < 80, 4
            , ifn(80 <= x < 90, 5
            , 6 )))));
RUN;
```

METHOD 5, SELECT-WHEN

SELECT-WHEN method is another favorite of the author’s:

```
DATA Method_5;
    SET inData;
    SELECT;
          when (x < 10)        y = 1;
          when (10 <= x < 20)  y = 2;
          when (20 <= x < 50)  y = 3;
          when (50 <= x < 80)  y = 4;
          when (80 <= x < 90)  y = 5;
          when (90 <= x)       y = 6;
          otherwise            y =.;
    END;
RUN;
```

METHOD 6, CASE-WHEN IN PROC SQL®

A similar logic can be implemented in PROC SQL®:

```
PROC SQL;
    CREATE table Method_6 as
    SELECT *, case
                 when (x<10)         then 1
                 when (10 <= x < 20) then 2
                 when (20 <= x < 50) then 3
                 when (50 <= x < 80) then 4
                 when (80 <= x < 90) then 5
                 when (90 <= x)      then 6
                 else .
               end as y
    from inData;
QUIT;
```

METHOD 7, USING A BESPOKE FORMAT

Using a customized format will results in some very neat coding:

```
PROC FORMAT;
    VALUE pwfmt
          low - <10 = 1
          10 - <20  = 2
          20 - <50  = 3
          50 - <80  = 4
          80 - <90  = 5
          90 - high = 6
             other  =.;
RUN;

DATA Method_7;
    SET inData;
    y = put(x, pwfmt.);
RUN;
```

To minimize the variation due to global CPU load and memory usage of our SAS® server, we packaged each method into a macro, and have them executed sequentially in a loop. This loop was then executed 100 times on each input dataset. For the format method, the format was only created once. We used the method described by McCartney and Hu[3] to exact the run time information from the log. We will discuss the results in the following section.

RESULTS DISCUSSION

The results were not quite what the author had expected. For the bell shaped distribution (α = 2, β = 3), we have the results summarized in the following two tables.

Method	Note	Mean (s)	Std_Dev
Method 1	IF THEN	28.71	0.45
Method 2	IF ELSE THEN	28.41	0.41
Method 3	Implicit	30.16	0.59
Method 4	Ifn()	28.12	0.53
Method 5	SELECT WHEN	28.25	0.50
Method 6	SQL CASE	38.97	0.58
Method 7	Format	36.66	0.51

Table 2. Results on Beta(2, 3) distribution with 2 categories

Method	Note	Mean (s)	Std_Dev
Method 1	IF THEN	30.28	0.60
Method 2	IF ELSE THEN	29.43	0.60
Method 3	Implicit	36.74	0.79
Method 4	Ifn()	39.21	0.87
Method 5	SELECT WHEN	31.55	0.68
Method 6	SQL CASE	42.14	0.85
Method 7	Format	38.15	0.48

Table 2. Results on Beta(2, 3) distribution with 6 categories

The first thing we did not anticipate was that the format method did not perform very well. Using a customized format is a very efficient way to merge two datasets, and the code is very elegant. It also can potentially save a lot code space when the same categorizing has to be applied multiple times. However, it simply lags behind the other methods in terms of performance in this case.

Despite of being a very neat 'looking' method, the author's favorite (Method 3) did not perform either. This is due to the fact that, when coded as an arithmetic operation with implicit IF operations, all cases have to be evaluated before a final value can be assigned. This will hinder the performance especially when a large number of categories are applied. The reason holds true for Method 4.

The biggest surprise is that, Method 1 and Method 2 are on par in terms of performance with the other methods. When the complexity of the categorization increases, they even outperform the others. Even more surprisingly, using mutually exclusively logic ELSE IF does not improve the calculation speed by a huge margin. This is probably due to some internal code optimization at the compiler step.

Lastly, it should not be a surprise that the SQL method does not perform as well as any of the DATA STEP methods.

Similar results were observed for the bimodal distribution ($\alpha = 0.5$, $\beta = 0.5$), and are presented in Table 3 and Table 4.

Method	Note	Mean (s)	Std_Dev
Method 1	IF THEN	28.31	0.48
Method 2	IF ELSE THEN	28.44	0.50
Method 3	Implicit	28.58	0.53
Method 4	Ifn()	28.38	0.48
Method 5	SELECT WHEN	28.48	0.46
Method 6	SQL CASE	39.84	0.51
Method 7	Format	37.14	0.62

Table 3. Results on Beta(0.5, 0.5) distribution with 2 categories

Method	Note	Mean (s)	Std_Dev
Method 1	IF THEN	31.29	0.79
Method 2	IF ELSE THEN	30.53	0.66
Method 3	Implicit	36.02	0.77
Method 4	Ifn()	38.49	0.86
Method 5	SELECT WHEN	32.65	0.83
Method 6	SQL CASE	42.74	0.76
Method 7	Format	39.49	0.79

Table 4. Results on Beta(0.5, 0.5) distribution with 6 categories

CONCLUSION

In doing this experiment, the authors have learned that SAS® has a highly optimized compiler. When dealing with a categorizing problem, it is hard to beat the good old IF-THEN logic, although other methods might appear to be more elegant.

REFERENCES

1. Hortsman, Joshua M., “Beyond IF THEN ELSE: Techniques for Conditional Execution of SAS® Code”, Proceedings of PharmaSUG 2016, Paper TT16, http://support.sas.com/resources/papers/proceedings17/0326-2017.pdf

2. Billings, Thomas E., “IFC and IFN Functions: Alternatives to Simple DATA Step IF-THEN-ELSE, SELECT-END Code and PROC SQL CASE Statements”, Proceedings of Western Users of SAS Software 2012, https://www.lexjansen.com/wuss/2012/28.pdf

3. McCartney, Jeff; Hu, Raymond, “SAS® Shorts: Valuable Tips for Everyday Programming”, Proceedings of NorthEast SAS Users Group 2011, https://www.lexjansen.com/nesug/nesug01/at/at1013.pdf

ACKNOWLEDGMENTS

The authors would like to thank Huanxue Zhou of KMK Consulting Inc, for valuable suggestions and discussion of this paper. Any mistakes herein are the responsibility of the author.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Anni Weng
KMK Consulting Inc.
anni.weng@kmkconsultinginc.com