SESUG Paper 6-2021

Converting Remeasurement Data into Percentile Ranks Based on Baseline Data Using PROC SQL: Patient Experience Measures for CMS Value-Based Purchasing Program

Jenhao Jacob Cheng, Children's National Hospital

ABSTRACT

As postulated by the Centers for Medicare and Medicaid Services (CMS) for hospital quality incentive built upon the inpatient quality measurement and reporting, Value-Based Purchasing (VBP) program started its first payment adjustment in FY 2013 by evaluating the achievement and improvement among hospitals in both clinical quality and patient experience domains. To further evaluate the consistency across different measures, patient experience data were required to be converted into percentile ranks based on the baseline distribution. This paper presents two SQL based methods with various options for converting the remeasurement data points into standardized percentile ranks according to the distribution in baseline measurement, with features and comparisons discussed. These rank based distribution-free methods are implemented mainly by Proc SQL as well as Proc Univariate and Data Step through SAS® 9.4 Software.

INTRODUCTION

In this study, we analyzed the patient experience data used to implement the first VBP program in FY 2013, including the data collected in both baseline period (FY 2010) and performance period (FY 2012). The hospital sample size (N) in the national comparison group (Hospital Compare website) is 3,765. Patient experience data has 8 measures: (1) nurse communication, (2) doctor communication, (3) communication about medications, (4) responsiveness of staff, (5) pain management, (6) cleanliness and quietness, (7) discharge information, and (8) overall rating of hospital. Each measure was scored by either achievement points or improvements points (which larger). Additional consistent points were awarded to the entire patient experience domain by checking the percentile rankings across 8 measures, benchmarked against the baseline period. Therefore, each of the original proportional measures (% of satisfaction) in performance period should be converted into percentile ranks based on the baseline distribution. Note that percentile "ranks" (always between 0 and 100) are different from the percentile "values" which are usually used to describe a data distribution in original data scale at just a few points such as 25th, 50th and 75th percentiles.

There are several approaches to address this problem, but we decided to employ methods that have a straightforward concept and could easily be explained to users and implemented in a reporting tool. In this sense we wish to propose two distribution-free methods that do not rely on the assumption or estimation of the underlying distribution model since they are not easily explainable or understood by many people. The first cross ranking method is to repeatedly rank each remeasurement data point within the entire baseline distribution and then obtain the percentiles ranks by dividing these ranks by the sample size. Such repeated ranking can be naturally handled by the Cartesian join (or cross join) concept through Proc SQL to process all possible combinations of the data points from two tables. Building upon such Cartesian product, additional built-in functions and calculated fields used together with the "select" and "group by" statements are applied to summarize the ranking results and report for N=3765 hospitals. The second percentile mapping method involves two major steps: (1) create a percentile lookup table based on the baseline distribution using Proc Univariate, and (2) map the remeasurement data to the lookup table to get the corresponding percentile ranks defined by baseline data through Proc SQL. Depending on which method to be used, various SQL join options are available including full join, unequal join, and nearest join.

Both methods should be accurate if the sample size is sufficient and there are not many tied values (e.g., more continuously distributed) but the first method can be less computationally efficient if the sample size is too large with N x N comparisons. Second method seems to be able to greatly improve the efficiency

when N is large with only N x 100 comparisons and alleviate the smoothness issue to some extent when N is small (say less than 100) by choosing the pctldef= option of Proc Univariate to be 1 or 4 to create smoother results with interpolation between data points. Ranking accuracy can be improved for both methods with “nearest join” which selects the closest percentile ranking from two unequal join results in different directions, more apparent when N is only moderate, say between 100 and 200. With the built-in selecting, sorting, and grouping capabilities, Proc SQL is a naturally powerful tool for rank-based analysis such as calculating percentile ranking in this study. To better illustrate the concept and methodology, all the code samples are based on the last patient experience measure, the satisfaction rate of the overall hospital ranking. The distributions of both the baseline and remeasurement data are illustrated in Figure 1. The remeasurement data has a higher average and a slightly smaller variance in comparison to the baseline data, and both distributions are quite symmetric

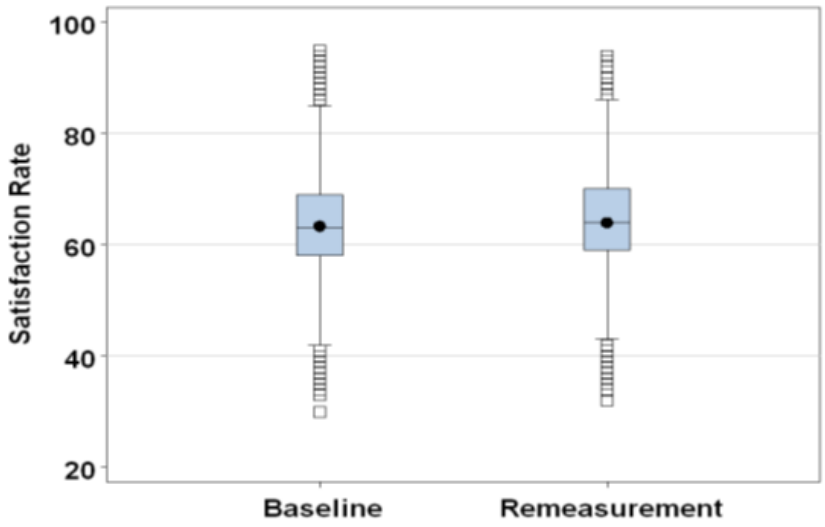

Figure 1: Box Plot for Overall Hospital Rating Measure

METHOD 1: CROSS RANKING

For the first method, once we are given a set of continuous numbers (remeasurement) we want to find out their position in another set of numbers in the similar scale (baseline measurement). In the case of patient experience data, let’s assume a hospital’s satisfaction rate for a particular measure is 76% and we would like to know what this hospital’s percentile rank will be according to the distribution of baseline satisfaction rates from the entire pool of national hospitals. Our approach for the first method is to rank each remeasurement data point repeatedly within the baseline distribution, and then standardize these ranks by the comparison group size to obtain the percentile ranks (e.g., dividing the ranking by N in each loop). Such repeated comparisons can be implemented by Proc SQL with three options.

OPTION 1: FULL JOIN

It is most intuitive to process such repeated rankings by comparing all possible combinations of data points from two sets. The full Cartesian join is the most fundamental join without explicitly specifying the join variable in “where” statement to form the full cross product. This allows us to compare all remeasurement data to all baseline data in N x N combination of rows. Building upon such Cartesian product, additional built-in functions and calculated fields used together with the “select” and “group by” statements are applied to summarize the ranking results and reduce the dimension back to N. With each remeasurement rate as the grouping variable, the comparison is conducted by the “case when” logic within the “select” statement and the percentile is calculated by summarizing the percentage of how many baseline rates are below the current remeasurement rate of its group. The SAS code is shown below:

```
proc sql;
create table pctl_m11 as
select c.hospital_id, c.measure_rate,
```

```
        sum(case when c.measure_rate>=b.measure_rate
                 then 1 else 0 end) as num,
        count(c.hospital_id) as den,
        round(100*(calculated num/calculated den),1) as pctl
from current c, baseline b
group by c.hospital_id, c.measure_rate;
quit;
```

To avoid the tied values to form a pooled group across multiple hospitals (data points), we reinforced the "group by" statement by including the hospital identification number in addition to the remeasurement data itself. We call this enriched query as Cartesian based join in order to distinguish it from the genuine Cartesian join.

OPTION 2: UNEQUAL JOIN

The same idea can be implemented by unequal join where the equivalency is not necessary on the join variable. The repeated comparison is explicitly specified in the "where" statement as the one-way inequality of current rate ≥ baseline rate, which results in only half of the full combinations from two sets of data points (N x N /2). After grouping by each remeasurement data point, the number of rows filtered in by the inequality indicates the ranked position of this remeasurement rate compared to the baseline distribution. The unequal join is more computationally efficient than full Cartesian based join due to the reduced data dimension. The SAS code is shown below:

```
proc sql;
create table pctl_m12 as
select c.hospital_id, c.measure_rate, round(100*count(*)/3765,1)as pctl
from current c, baseline b
where c.measure_rate>=b.measure_rate
group by c.hospital_id, c.measure_rate;
quit;
```

However, one potential problem with this (>=) unequal join is that the remeasurement rates below the minimum of baseline distribution will not be included in the query results because the inequality condition cannot be met. An additional step (code not shown here) must be used to recap these unmatched values and assign their percentile ranks to lowest rank (0 or 1). On the other hand, it is not an issue if the remeasurement rates are higher than the maximum baseline distribution as their percentile ranks will all be assigned to 100 correctly. The situation will be reversed if the different direction (<) is used. If the data is discrete, the unequal join may tend to inflate the percentile ranks since all tied values will be included.

OPTION 3: NEAREST JOIN

Another potential drawback of unequal join is that it may not find the closest points and result in less accurate results since the comparison is based on one-way comparison. For example, if we want to compare a remeasurement value of 7.9 to 10 baseline values from 1 to 10 in the (>=) direction, the calculated percentile rank will be 70 (7/10 x 100%) rather than 80, the closest number. The solution is to combine two unequal joins in different comparison directions together by "union" operator then select the nearest one from the union. The SAS code is shown below:

```
proc sql;
create table pctl_m13 as select hospital_id, measure_rate, pctl from
(
select c.hospital_id, c.measure_rate, round(100*count(*)/3765,1) as pctl,
       min(c.measure_rate-b.measure_rate) as diff
from current c, baseline b
where c.measure_rate>=b.measure_rate
group by c.hospital_id, c.measure_rate

union
```

```
select c.hospital_id, c.measure_rate, round(100*(3765-count(*))/3765,1) as
pctl,
       min(b.measure_rate-c.measure_rate) as diff
from current c, baseline b
where c.measure_rate<b.measure_rate
group by c.hospital_id, c.measure_rate
)
group by hospital_id
having diff=min(diff);
quit;
```

Two counterparts of unequal joins are complementary to each other so the unmatched data in one inequality will be caught by the other inequality. Therefore, the nearest join is a complete procedure without the need of an additional step to make up the unmatched data. The derived variable "diff" is used to measure which direction will give the nearest value. However, the cost of improved accuracy via this complete procedure is the increased data dimension with two half-full combinations (N x N). Note that the nearest join does not improve the accuracy over the unequal join for discrete data since a remeasurement value will not reside between two adjacent baseline values, but the unmatched issue is addressed here.

The percentile ranks of the "baseline" data can also be obtained with exactly the same methods mentioned above by querying from the two same baseline tables in the "from" statement, which is referred to as "self join" or "reflexive join". Due to the space limitations, we have only displayed the percentile ranks calculated by option 1 for the 20 sample hospitals as illustrated in Table 1. Note that the sample hospitals are randomly selected from the national population and the data is ordered by the baseline rate

Hospital	Baseline Rate	Remeasurement Rate	Baseline Percentile Rank	Remeasurement Percentile Rank
1	44	44	3	3
2	50	51	8	10
3	53	55	13	17
4	56	54	19	15
5	58	60	26	34
6	59	59	29	29
7	60	61	34	38
8	62	63	43	47
9	63	62	47	43
10	64	63	52	47
11	65	65	56	56
12	66	68	61	69
13	67	69	65	72
14	68	69	69	72
15	70	70	76	76
16	71	75	79	90
17	73	73	85	85
18	75	74	90	88
19	78	75	94	90
20	88	90	98	99

Table 1: Percentile Ranks Using Method 1 (Option 1)

METHOD 2: PERCENTILE MAPPING

The second method involves using intuitive thinking to solve the problem. The goal is to first create a percentile lookup table based on the baseline data, and then map the remeasurement data to the lookup table to obtain the corresponding percentile ranks. Creating the lookup tables was implemented by using

Proc Univariate with the default pctldef=5 option. Note that we need to transpose the percentile data from 100 columns to 100 rows and remove the prefix character “p” to make the percentile value as a numerical variable to be joined with the remeasurement rate. The SAS code is shown below:

```
proc univariate data=baseline noprint pctldef=5;
var measure_rate;
output out=pctldist pctlpre=p
pctlpts=1 2 3 4 5 ...95 96 97 98 99 100;
run;

proc transpose data=pctldist
               out=pctldist_t(drop=_label_ rename=(col1=pctl_value));
run;

data pctl_lookup; set pctldist_t;
drop _name_;
pctl_rank=input(compress(_name_, 'p'), 3.);
run;
```

As mentioned previously, this method can deal with the sample size and discrete problems to some extent. It will control the comparison size only up to N x 100. On the other hand, if the sample size is not large enough (N < 100), we can choose the pctldef= option of Proc Univariate to be 1 or 4 to make the percentile estimation smoother by applying interpolations (or weighted averages) among existing data points. Table 2 shows the percentile lookup table based on baseline distribution (e.g., percentile distribution of baseline data) with a few selected percentile ranks by using PROC Univariate with the default option for percentile calculation.

Percentile Rank	Percentile Value based on Baseline Data
0	24
1	38
5	47
10	52
20	57
25	58
40	62
50	64
60	66
75	70
80	72
90	76
95	80
99	90
100	98

Table 2: Percentile Distribution of Baseline Data

Once the lookup table is available, two options of the first method, unequal join and nearest join, are perfect tools to map the remeasurement rates to the percentile values from the lookup table to retrieve the corresponding percentile ranks. Cartesian based join is not valuable here since we don’t need to process the full possible combinations to calculate the percentile ranks. Instead, the percentile ranks are available already and need to be pulled out from the best matches between the remeasurement rates and percentile values.

OPTION 1: UNEQUAL JOIN

The concept of this option is identical to method 1-2 but with an unequal join between the remeasurement rates and the baseline-based percentile values to merge the corresponding percentile ranks into the remeasurement data. The percentile rank with its percentile value next smaller to the remeasurement

value will be merged with the comparison of one-way inequality. This implies that the unequal join may lead to more conservative (lower) percentile ranks as all joins happen on the left boundary. The SAS code is shown below:

```
proc sql;
create table pctl_m21 as
select c.hospital_id, c.measure_rate, max(p.pctl_rank) as pctl
from current c, pctl_lookup p
where c.measure_rate>=p.pctl_value
group by c.hospital_id, c.measure_rate;
quit;
```

As the unequal join of the first method, an additional step (code not shown here) must be used to recap the unmatched numbers and assign their percentile ranks to 0 for those remeasurement rates below the baseline scope.

OPTION 2: NEAREST JOIN

The concept of this option is identical to method 1-3 but with a nearest join between the remeasurement rates and the baseline percentile values to merge the corresponding percentile ranks into the remeasurement data. With the purpose to improve the accuracy over the one-way comparison, nearest join also resolves the unmatched problem inherited from the simple unequal join by combining two inequalities with “union” operator. The SAS code is shown below:

```
proc sql;
create table pctl_m22 as select hospital_id, measure_rate, pctl from
(
select c.hospital_id, c.measure_rate, max(p.pctl_rank) as pctl,
       min(c.measure_rate-p.pctl_value) as diff
from current c, pctl_lookup p
where c.measure_rate>=p.pctl_value
group by c.hospital_id, c.measure_rate

union

select c.hospital_id, c.measure_rate, min(p.pctl_rank) as pctl,
       min(p.pctl_value-c.measure_rate) as diff
from current c, pctl_lookup p
where c.measure_rate<p.pctl_value
group by c.hospital_id, c.measure_rate
)
group hospital_id
having diff=min(diff);
quit;
```

Again, the percentile ranks of the “baseline” data can be obtained with exactly the same methods mentioned above by querying from two same baseline tables in the “from” statement, which is referred to as “self join” or “reflexive join”. Due to the space limitation, we do not display the percentile ranks calculated by the second method.

CONCLUSION

This paper presents two simple methods for converting remeasurement data points into percentile ranks based on the baseline distribution. However, they can be extended to any two different distributions that share similar characteristics or scale. Both methods reflect intuitive distribution-free approaches that could be explained easily to users, as opposed to the more complicated approaches where we may have to assume or estimate the underlining distribution. We found the first method the overall efficient and most accurate if the sample size is adequately large enough and the data is relatively smooth. When the first

method lacks smoothness due to moderate sample size or discrete behavior, or when it suffers from the computational inefficiency due to extremely large sample size, the second method is a viable alternative.

Table 3 summarizes the properties of the two methods with different options. There is a mild concern in this study that the downloaded satisfaction rate data is rounded to the nearest 2 significant figures, resulting in several tied values. Two methods lead to very similar results except that some hospitals have one position lower for percentile ranks by method 2 due to the mild rounding issue. Recall that for discrete data method 1 may inflate the percentile ranks while method 2 is more conservative. The core concept of both methods is that the unequal join technique, which is not often used for daily and regular table joins, could be very powerful for conducting any rank-based statistics. As shown in this study, Proc SQL is far more valuable for data management and manipulation than most people realize.

Method	Code Complexity	Data Dimension	Comparison Direction	Sample Size Range	Rounding Bias
1-1	Simple query + additional functions	N x N	One-way	Normal	Rank Up
1-2	Simple query + post procedure	N x N / 2	One-way	Normal	Rank Up
1-3	Composite query	N x N	Two-way	Normal	Robust
2-1	Simple query + prior/post procedures	N x 100 / 2	One-way	Wider	Rank Down
2-2	Composite query + prior procedure	N x 100	Two-way	Wider	Robust

Table 3: Summary of Two Methods with Different Options

REFERENCES

Report to Congress. 2007. “Plan to Implement a Medicare Hospital Value-Based Purchasing Program”. *Department of Health and Human Services*. November 21, 2007.

Hospital Downloadable Database. “Patient survey (HCAHPS) – Hospital”. *Department of Health and Human Services*. FY 2013 Data. Available at https://data.cms.gov/provider-data/dataset.

Mendez, L. and T. Dunn. 2009. “A Propaedeutics for PROC SQL Joins”. *SAS Global Forum 2009*. Washington, DC.

Fairfield-Carter, B. and D. Carr. 2008. “Labor-saving SQL Constructs”. *SAS Global Forum 2008*. San Antonio, TX.

Ronk, K. M. 2004, “Introduction to Proc SQL”. *SUGI 29*, Montréal, CANADA.

Izrael, D. and M. P. Battaglia. 2015. “Two Useful Macros to Nudge SAS to Serve You”. *NESUG 2002*. Buffalo, NY.

CONTACT INFORMATION

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

Jenhao Jacob Cheng, PhD, MS, PStat
Children’s National Hospital
111 Michigan Ave NW, Washington, DC 20010
jcheng@childrensnational.org
https://childrensnational.org