PharmaSUG 2014 - Paper SP09

Same Data, Separate MEANS – 'SORT' of Magic or Logic?

Naina Pandurangi, inVentiv Health Clinical, Mumbai, India

Seeja Shetty, inVentiv Health Clinical, Mumbai, India

ABSTRACT

Sample Mean is the most fundamental element of any statistical analysis. It is also considered to be one of the simplest descriptive statistics with a straightforward formula i.e. sum of individual sample values divided by the number of values. Quite reasonably, one would expect to get a unique mean value for a given set of results irrespective of the order of the individual data points. But does this notion hold true forever or can there be some infrequent cases to challenge this apparent assumption of uniqueness of mean for a pre-defined sample in SAS®? Can we really get different mean values on the same set of data, in two different situations – once when data is sorted by some variable and second when it isn't? Sounds a bit absurd and unthinkable but yes, we are talking about more than one mean value on the same data in SAS®!

This paper discusses some peculiar and uncommon examples of data in SAS® which have the capacity to yield more than one value of sample mean when pulled in, in a sorted and un-sorted format.

INTRODUCTION

In our SAS® world of statistical analysis, the most frequently used statistic is the sample mean. As taught in our basic classes the formula for sample mean of a random sample of size *n* is -

$$\text{MEAN:} \qquad \bar{x} = \frac{1}{n}\sum x_i$$

Where: x = random variable

$i = 1, 2 \ldots n$

When broken down, this is nothing but addition of the n sample values divided by *n*. This indicates that the order in which the individual sample values appear should not be important and make no difference to the mean value $\bar{x}$ because the order in which we add the numbers doesn't change the summed value $\sum x_i$(n being a constant). Does this notion hold true in every platform and in any given application? How does the order of the input values matter in SAS®? Technically what we want to test is, as per the above definition of $\bar{x}$, if the mean for the same sample is calculated as $[(x_3 + x_5 + x_2 + x_{1\ldots} + x_n + x_{n-3})/n]$ in place of $[(x_1 + x_2 + x_3 + x_{4\ldots} + x_{n-1} + x_n)/n]$, then is there any difference between the two mean values.

Let us explore the behavior of SAS® in different situations and procedures executed on the same input values but ordered differently.

EXAMPLE S ILLUSTRATING THE CASE

1] CALCULATING MEAN USING PROC MEANS:

Consider dataset ONE with variables SUBJECT and RES showing the sample results for each subject in random order:

DATA ONE:

SUBJECT	RES
4	6.81
1	-1.51
3	0.67
2	-1.26

Step A: Mean RES using PROC MEANS on unsorted DATA ONE:

```
proc means data=one;
      var res;
      output out=stat1_ mean=mean;
run;
```

As a conventional rule of precision used for displaying the statistics in study reports, we format the MEAN value up to 1 decimal more than the precision of the raw results. In this case the precision for display of mean would be 3 decimal places. So formatting the mean as follows:

```
*format the MEAN to display the decimals up to 3 decimals*;
data stat1;
       set stat1_;

       meanc=put(mean, 8.3);
put 'Mean1 = ' meanc;
run;
```

Log window shows:

```
Mean1 = 1.178
```

Step B: Mean RES using PROC MEANS on DATA ONE sorted by SUBJECT:

```
proc sort data=one out=one_sorted;
       by subject;
run;
```

DATA ONE_SORTED:

SUBJECT	RES
1	-1.51
2	-1.26
3	0.67
4	6.81

```
proc means data=one_sorted;
       var res;
       output out=stat2_ mean=mean;
run;

data stat2;
       set stat2_;

       meanc=put(mean, 8.3);
put 'Mean2 = ' meanc;
run;
```

Log window shows:

```
Mean2 = 1.177
```

Step C: Observation:

We see from the log messages that the mean values coming from the datasets ONE and ONE_SORTED are not exactly equal and when formatted up to 3 decimals the two values i.e. 1.178 and 1.177 differ in the 3rd decimal place.

Note: We get the same output if the mean value is calculated using other SAS® procedures like PROC SQL, PROC UNIVARIATE, PROC TABULATE, PROC SUMMARY…

2] CALCULATING MEAN IN A DATA STEP:

Step A: Calculate the MEAN value using the same input numbers as above in a data step:

```
data check;
       mean1 = put((6.81-1.51+0.67-1.26)/4, 8.3);
       mean2 = put((-1.51-1.26+0.67+6.81)/4, 8.3);

       put 'Mean1 = ' mean1 ', Mean2 = ' mean2;
run;
```

Log window shows:

```
Mean1 = 1.178 , Mean2 = 1.177
```

Step B: Observation:

As per the message in the log window, again, as seen in example 1 we see that the two mean values MEAN1 and MEAN2 calculated on the same set of input results are not exactly equal!

UNDERLYING HITCH

As discussed in the Introduction section, mean is basically a measure of addition of the values. We just saw that this addition produced unequal results in two different situations. So let's see what the core values of this addition are in the dataset CHECK used in example 2 above, and if they are really equal.

```
data check;

       sum1 = 6.81-1.51+0.67-1.26;
       sum2 = -1.51-1.26+0.67+6.81;

       put 'Sum1 = ' sum1 ', Sum2 = ' sum2;

       if sum1=sum2 then put 'Sum1 is equal to Sum2';
       else if sum1 ne sum2 then put 'Sum1 is NOT equal to Sum2';
run;
```

Log window shows:

```
Sum1 = 4.71 , Sum2 = 4.71
Sum1 is NOT equal to Sum2
```

So here it is! It is obvious from the log message saying 'SUM1 is not equal to SUM2', that the difference in mean is due to the difference in the two values of sum that look same but are not identical. This means one of the two values displayed as 4.71 is not originally 4.71. Let us see further whether that's SUM1 or SUM2.

```
data check1;
       set check;

       diff1=4.71-sum1;
       diff2=4.71-sum2;
run;
```

DATA CHECK1

sum1	sum2	diff1	diff2
4.71	4.71	0	8.881784E-16

The variable DIFF2 in the dataset CHECK1 is not 0, which confirms that the incorrect or non-exact value displayed as 4.71is coming from the variable SUM2. And the real value of SUM2 is 8.881784×10^{-16} less than 4.71.

By now, we must agree that the same set of input values may have the capacity to produce two different values when added in two different sequences. We will now see the reason of this difference in the following section.

ROOT CAUSE OF NUMERIC IMPRECISION

Seldom do we notice that the sum of two numbers A and B which is shown as A+B in SAS® dataset is not exactly A+B but is only close to A+B.

For example, logically the three numbers 1.2, 1.4 and 2.7 should add up to 5.3.

```
data now;
       a=1.2+2.7+1.4;
       b=1.2+1.4+2.7;

       diff_a=5.3-a;
       diff_b=5.3-b;
run;
```

DATA NOW

a	b	diff_a	diff_b
5.3	5.3	-8.88178E-16	0

The sequence used in variable *a* fails to reach the exact value of 5.3 in SAS® whereas the one in *b* does succeed. But the value we see in both cases is 5.3 which means that the SAS® storage value here is not 5.3. Then how does SAS® store the value that is seen as 5.3 to us?

Internally any computer uses a binary number format to store all numeric values. Floating point method is one such method that SAS® uses for all numeric values. This method has a limitation of no being able to accurately and fully represent some decimal numbers. Just like how we cannot reach a finite value for the division of 1/3 and settle for 0.33 or 0.3333 based on our requirement. A similar judgment is applicable in this method as well. In other words, using this method, when our code is executed some of the values like 0.1 are already rounded to the nearest number in that format which indirectly brings in some rounding error known as floating point error.

Our next concern would be if SAS® has such a shortcoming then how the variable *b* in DATA does NOW above produces the exact number?

What we must understand is that floating representation is always present but a key factor affecting the authenticity of the final result is the order in which each number (for SAS® it's the binary numeric representation) is entered. Another key factor is how the input values are originally derived. There is more chance of inexactness on derived and re-derived numbers. These two factors are important because when values are accumulated in a different order, it introduces the possibility of some slight numeric imprecision due to floating point differences.

For more details please see references 1, 2, and 3.

HOW TO IDENTIFY ANY EXISTANCE OF FLOATING ERROR

This scientific representation error is not caught easily because it is not directly visible unlike any other log message like a WARNING, ERROR…

Moreover in case of large data when two people get two different mean values the catch is to recognize the accurate one between the two. In many cases where the difference is very tiny it is the investigator's take whether to ignore it or trace down further. For e.g. the *n*th decimal place value may not be important for a garment dealer but it would be of utmost concern for a money lender.

In our case above, we saw that the difference between the two real sum values is extremely minute but the disparity this causes in the mean is significant enough and we would want to trace the real one between 1.177 and 1.178.

Following are a few easy checks we can make in SAS® that should work in majority of the cases.

Check1: Convert the numeric value to numeric value again.

Let us see the original values of MEAN in the two datasets STAT1 and STAT2 referred in the example 1above.

DATA STAT1

mean	meanc
1.1775	1.178

DATA STAT2

mean	meanc
1.1775	1.177

Both the mean values are displayed as 1.1775. Let's convert this to a new numeric value again and verify the accuracy in both the datasets by adding the following lines of codes in both the datasets:

```
"newmean=input(mean, best.);

 diff=newmean-mean;"
```

DATA STAT1

mean	meanc	newmean	diff
1.1775	1.178	1.1775	0

DATA STAT2

mean	meanc	newmean	diff
1.1775	1.177	1.1775	2.220446E-16

The DIFF variable is 0 in STAT1 dataset but not in STAT2 dataset. This tells us that there is a floating point error in the value 1.1775 of DATA STAT2 and hence we should go with 1.178 as the correct mean from DATA STAT1.

Check2: Using the format BINARY64.

We have two values displayed as one particular number. This is like two figures having one image, of which one figure is real and other is close to real. We can spot the 'close to real' one by comparing the binary representations of the original underlying figures with that of the real image. In our case both the mean values have the image 1.1775. Let us check the raw computer representation of these numbers by adding the following lines of codes in both the datasets –

```
data stat1;
       set stat1;

       orig=1.1775;
       put orig binary64. ' Original' orig;
       put mean binary64. ' Mean form STAT1 dataset ' mean;
run; *add similar patch in the STAT2 dataset*;
```

Log window shows:

```
0011111111110010110101110000101000111101011100001010001111010111 Original representation of 1.1775
0011111111110010110101110000101000111101011100001010001111010111 Mean from STAT1 dataset 1.1775

0011111111110010110101110000101000111101011100001010001111010110 Mean from STAT2 dataset 1.1775
```

From the log, the raw binary representation of the actual mean in STAT1 matches exactly with the raw binary representation of its image but the mean in STAT2 dataset doesn't. Hence we can infer that STAT1 contains the accurate mean value.

WHY MEAN?

Mathematically, the reason that MEAN is affected due to this numeric error is that it is a dependant of SUM. Statistically, this floating point error has the capacity to affect the MEAN (and its dependants like standard deviation, variance, confidence limits...) because it is one statistic that depends upon each individual data point and is affected by extreme values, unlike median and mode. This is why there is a lesser chance that statistics like median and

mode are altered due to the order of the data points. When the underlying error has an effect on other statistics along with the MEAN we need to check if the methods discussed below can be used on them as well.

MEASURES OF HANDLING

Once we are aware of the fact that there is some precision error in our numbers, next we would want to eliminate the floating error effect from all our values. Let us explore a few methods that can help us in this direction in most cases.

Method1: Convert the numeric value to numeric value again using 'INPUT BEST.'

As seen in Check1 above, creating a new numeric variable using the "INPUT BEST." function returns the value that is displayed in SAS® (in place of the underlying incorrect value). We can do so either on the input results before calculating the mean (step a below) or on the derived mean value in datasets STAT1 and STAT2 (step b below):

a)
```
data one;
       set one;

       newres=input(res, best.);
  run;
```
(Rest of the steps same as in Example1)

OR

b)
```
data stat2;
       set stat2_;

       meanc=put(input(mean, best.), 8.3);
  put 'Mean2 = ' meanc;
 run;
```
(Rest of the steps same as in Example1)

In both the cases a) and b), we would get the same mean value and the log message for the STAT2 datasets is now correct as follows:

```
Mean2 = 1.178
```

Method2: Multiply and divide by a constant with base 10.

"If all the input values are multiplied (or divided) by a constant, the descriptive statistics like SUM and MEAN get multiplied (divided) by the same constant." This is one property of these statistics based on which is this method.

With regards to our problem, the idea is to remove the floating decimal effect i.e. convert the fractions to integers. In terms of floating point language, move the desired number of precision decimals to the left of the decimal. This is because often working with integers avoids any numeric imprecision.

How do we decide on this constant which will be multiplied with all the individual results and then divided from the resultant mean value?

A safe approach here would be to depend upon the maximum no of decimal places in the data.

In example 1, the data is captured up to 2 decimals, so we will multiply all the individual results by 10^2.

```
data one;
       set one;

       newres=res*100;
run;
```
(Rest of the steps same as in Example1)

After calculating the mean, we then divide it back from the MEAN value.

```
data stat2;
       set stat2_;

       meanc=put((mean/100), 8.3);
put 'Mean2 = ' meanc;
run;
```

This will give the following log message showing the expected MEAN value:

```
Mean2 = 1.178
```

Note: This logic of 'multiplying the data points by 10^x where x is maximum number of decimal places in the sample' may not work smoothly after x ≥ 5. But conventionally, displaying statistics up to 5 decimals or more is a not a common scenario also. Even if we do, there is high chance that the difference in two mean values produced from two separate sequences is minute enough to ignore. And in the rarest case, if we cannot ignore, then we may explore other methods mentioned in this paper.

Method3: Using FUZZ factor through PROC FORMAT.

As per the SAS® definition, the FUZZ function returns the nearest integer value if the argument is within 1E-12 (i.e.10^{-12}) of the integer (that is, if the absolute difference between the integer and argument is less than 1E-12). Otherwise, the argument is returned.

Basically this function is used when we would want one value to be displayed same as other if the distance between the two is within a set limit. For e.g. if we want any value which is within 0.1 (on both sides) of the number 2.1 to be displayed as 2.1 itself then we would use the fuzz format as follows:

```
proc format;
       value fuz (fuzz=0.1) 2.1 = 2.1;
run;

data chk;
       do a=2.1 to 2.5 by 0.1;
              a_fuzz=a;
              format a_fuzz fuz.;
              output;
       end;
run;
```

DATA CHK

a	a_fuzz
2.1	2.1
2.2	2.1
2.3	2.3
2.4	2.4
2.5	2.5

Observe that the value 2.2 which is within 0.1 units of 2.1 is displayed as 2.1.

This gives us an idea that fuzz format can be used to remove the distance between two individual numbers and hence may have the ability to overcome the fuzz caused due to floating error.

But in case of large data we would not know the possible data values or the mean values unlike in the example above where we knew the number 2.1.

So we can let SAS® decide the formatting on its own on the variable we pass as follows:

```
proc format;
       value fuz (fuzz=1E-12); /*Default fuzz factor used by SAS® is 1E-12*/
run;
```

In the STAT2 dataset from example 1, we would add the following lines:

```
"newmean=put(mean, fuz.)*1;
meanc=put(newmean, 8.3);"
```

(Rest of the steps same as in Example1)

We would get the expected mean value and the log message as follows:

```
Mean2 = 1.178
```

Note: Setting fuzz to a value as small as 1E-12 means the original value, say A, will be rounded to the nearest integer, say B, only if B-A ≤ 0.000000000001. In other words this eliminates any distance less than 1E-12 (And in our case the distance between the underlying mean and the displayed mean is 2.220446E-16 (DATA STAT2 on page 7) which is much lesser than 1E-12). So this way we can take help of fuzz format to remove the floating point errors as long as the distance increased by this error is ≤1E-12. This fuzz limit can be altered depending upon the rounding we want to allow. We have used the default value used by SAS® which is wide enough to eliminate differences up to 1E-12. Please note that this option may not produce the desired results if our data contains too many recurring decimal numbers or with fractions with as many as 7 or 8 (or more) decimal places.

CONCLUSION

Numeric imprecision is a common characteristic in data coming from numerous entries and calculations, not only in SAS® but any digital computing. This is not limited to only negative or only positive fractions or a combination of both but yes, integers are safe from this elusiveness. Also, it is not only the MEAN but also other statistics (especially those depending on the MEAN) will be affected by this. We may not even be aware of such subsistence in our reports. It is when two people come across distinct values due to dealing with different order of the same data points or some W.D kind of a note in the SAS® log that we start digging in the roots. But again, this divergence is a relative term and many times may not be of caution. Also, the methods used to tackle such inaccuracy may fail to work in extremely unclean or diverse data because after a certain level of decimal representations it is not possible for SAS® to reach the literal digit.

So it is a matter of researcher's assessment to decide the severity of the inaccuracy and apply the pertinent measures.

REFERENCES

1. *"What Every Computer Scientist Should Know About Floating-Point Arithmetic"*, by David Goldberg, published in the March, 1991 issue of Computing Surveys. Copyright 1991, Association for Computing Machinery, Inc.
2. Numeric Length: Concepts and Consequences, by Paul Gorrell, Social & Scientific Systems, Inc., Silver Spring, MD, NESUG 2007
3. "*What Every Programmer Should Know About Floating-Point Arithmetic*", available at http://floating-point-gui.de/.

ACKNOWLEDGMENTS

I thank all my colleagues who have helped me in understanding these statistical concepts and reviewed and provided useful comments on my paper. I am really thankful to SAS® support to help me with all my queries in a timely fashion and am grateful my colleague Sandeep Sawant who has truly inspired me to make this happen.

RECOMMENDED READING

- [PDF] **TS-DOC: TS-230 - Dealing with Numeric Representation Error in SAS® Applications**

CONTACT INFORMATION

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

Name: Naina Pandurangi
Enterprise: inVentiv Health Clinical
Address: Marwah Centre , Ground Floor, B Wing, Krishnalal Marwah Marg, Andheri (E), Mumbai – 400 072.
City, State ZIP: Mumbai, MH, India
Work Phone: +91-9833915538
E-mail: nainap9@gmail.com