sanofi pasteur
The vaccines business of sanofi-aventis Group

SAS Programming Tips

Y. B. Shekar

Deputy Director, Statistical Programming
Clinical Department
Sanofi Pasteur
Swiftwater, PA 18370

Version 2.2 **11 Nov 2007**

Table of Contents

List of Abbreviations

ASCII – American Standard Code for Information Interchange

ANSI – American National Standards Institute(1)

eBLA – Electronic Biologics License Application

CBER – Center for Biologics Evaluation and Research

CDER – Center for Drug Evaluation and Research

eCTD – Electronic Common Technical Document

EDR – Electronic Document Room

EMEA – European Agency for the Evaluation of Medicinal Products

FDA – Food and Drug Administration (US)

iCSR – integrated Clinical Study Report

IEEE – The Institute of Electrical and Electronic Engineers

IND – Investigational New Drug Applications

ISO – International Organization for Standardization

NDA – New Drug Applications

PDF – Portable Document Format

RTF – Rich Text Format

SAS – Software from SAS, Inc.(2); de facto standard software used to analyze clinical data in biostatistics departments of most pharmaceutical companies

XML – Extensible Markup Language

1 Introduction

This document provides some programming tips to statistical programmers and statisticians on some intricacies or anomalies within SAS(3). Please note the tips provided in this document while writing SAS programs to generate tables, listings and graphs. Validators should be cognizant of these tips too. These tips should be considered as guidance. Not following the guidance could result in incorrect results being sent to FDA(4), EMEA and other regulatory agencies.

This document will be updated as and when new issues are identified. Programmers and statisticians are encouraged to provide any new issue or tip they have noted to their immediate manager. There may be more than one solution to any issue, but only one solution may be provided in this document.

2 Objective

The main objective is to ensure that the output provided by Biostatistics department is of high quality and accurate.

3 Numeric Precision

When comparing variables to constants the resulting outcome may not be as expected or desired. Floating-point representation can account for anomalies in SAS program behavior. In most situations, the way that SAS stores numeric values does not affect the resulting outcome. However, in certain situations it does affect the result, and some representative examples are provided in the following sub-sections where it may have an impact. Please note that examples included in the following sections are not exhaustive, and hence, the concept should be kept in mind while programming.

3.1 Floating-Point Representation

SAS stores all numeric values using floating-point, or real binary, representation. Floating-point representation is a form of scientific notation in which values are represented as numbers between 0 and 1 times a power of 10. For example, a number 56789 could be represented as follows:

$$.56789 \times 10^5$$

Numbers in scientific notation are comprised of the following parts:

- The ***base*** is the number raised to a power; in this example, the base is 10.
- The ***mantissa*** is the number multiplied by the base; in this example, the mantissa is .56789

- The ***exponent*** is the power to which the base is raised; in this example, the exponent is 5

On most operating systems the base is not 10, but is either 2 or 16. The following table summarizes various representations of floating-point numbers that are stored in 8 bytes.

Summary of Floating-Point Numbers Stored in 8 Bytes

Representation	**Base**	**Exponent Bits**	**Maximum Mantissa Bits**
IBM mainframe	16	7	56
Open VMS VAX	2	8	56
IEEE	2	11	52

The Institute of Electrical and Electronic Engineers (IEEE)(5) representation is used by many operating systems, such as, Microsoft Windows (2000, 2003, XP), UNIX and OS/2.

The IEEE representation uses an 11-bit exponent with a base of 2 and bias of 1023. The ***bias*** is an offset used to allow for both negative and positive exponents with the bias representing 0. If a bias is not used, an additional sign bit for the exponent must be allocated. A ***characteristic*** represents a signed exponent and is obtained by adding the bias to the actual exponent. For example, in IEEE standard representation, a characteristic with the value 1022 represents an exponent of –1, while a characteristic of 1026 represents an exponent of +3.

In most operating systems, the value of 1 represented by the IEEE standard is as follows:

3F F0 00 00 00 00 00 00

3.2 Computational Considerations of Fractions

Regardless of how much precision is available, some numbers cannot be represented exactly. In the decimal number system, fractions such as, 0.1, 1/3 cannot be represented exactly in base 2 or base 16 numbering systems. This is the principle reason for difficulty in storing fractional numbers in floating-point representation.

The following DATA step illustrates the aforementioned problem.

```
data one ;
     do i = 1 to 2 by .1 ;
         x= (5*i - 4*i)/i ;  *** x = 1  *** ;
         y=round(x,.001) ;   *** y = 1  *** ;
         a=0 ; b=0 ;
         if x=1 then a=1 ;
         if y=1 then b=1 ;
         OUTPUT ;
```

```
          end ;
     run ;
     proc print ;
     run ;
```

When executed the output obtained is as follows:

```
Obs      i      x      y      a      b

  1     1.0     1      1      1      1
  2     1.1     1      1      0      1
  3     1.2     1      1      1      1
  4     1.3     1      1      0      1
  5     1.4     1      1      1      1
  6     1.5     1      1      0      1
  7     1.6     1      1      0      1
  8     1.7     1      1      0      1
  9     1.8     1      1      1      1
 10     1.9     1      1      0      1
 11     2.0     1      1      0      1
```

In the above output, variable *a* is not always assigned 1 because the accumulation of the imprecise number introduces enough error that the exact value of 1 is not always encountered by *x*. The number is close and prints as 1 (see output above), but not always exactly equal to 1. This problem is easily resolved by explicitly rounding as assigned for *y*, which results in b taking the value 1 - always.

3.3 Numeric Comparison Considerations

Imprecisions can also cause problems with comparisons. In the following sub-sections various situations where a ROUND function would be useful is given. Consider the following example where fever rates are determined after converting the temperature from Fahrenheit to Celcius:

3.3.1 Temperature Conversion

```
     **  classify fever  ** ;
     data temp(keep=tempf) ;
          do i = 100 to 103.4 by .1 ;
             tempf=i ;
             output ;
          end ;
     run ;

     data local ;
           set temp;
           tempc=(tempf - 32 )/1.8;
           tempcr=round((tempf - 32)/1.8,.0001) ;
           if tempc < 38.0 then severity = 0;
```

```
            else if 38.0 <= tempc < 38.5 then severity = 1;
            else if 38.5 <= tempc < 39.5 then severity = 2;
            else if tempc >= 39.5 then severity = 3;

            if tempcr < 38.0 then severtyr = 0;
            else if 38.0 <= tempcr < 38.5 then severtyr = 1;
            else if 38.5 <= tempcr < 39.5 then severtyr = 2;
            else if tempcr >= 39.5 then severtyr = 3;
      run;
      proc print ;
      run ;
```

Execution of the above code provides the output as given in Appendix 1. It is clear that the variables SEVERITY and SEVERTYR do not match when the temperature is 38.0°C, 38.5°C or 39.5°C. Even though both TEMPC and TEMPCR display the same values for the aforementioned values of temperature, the one that uses the ROUND function, namely, TEMPCR, is the only one that assigns severity correctly. This anomaly is due to floating-point representation of numeric values. Using ROUND function appropriately resolves this problem.

Please note that the variable should be rounded to at least two decimal points more (x+2) than the comparison constant. In the above example, the variable *tempcr* is rounded to 4 decimals as the comparison constant has one decimal point.

3.3.2 X-fold rise

An example where ROUND function would be useful is in calculating four-fold or two-fold rise. However, using the ROUND function in the appropriate place is critical. For example, consider the following code:

```
      * bleed 1 is baseline, bleed 2 is post-vaccination *;

      data serologi;
         subject=1; bleed=1; test1=40; test2=80; output;
         subject=1; bleed=2; test1=160; test2=320; output;
      run;

      * get GMT for individual subject/antigen/bleed     *;

      data serology;
         set serologi;
         result = round(10**mean(log10(test1),log10(test2)));
      run;

      data baseline(drop=bleed rename=(result=baseline))
           postbase(drop=bleed);
         set serology(keep=subject bleed result);
         if bleed=1 then output baseline ;
         else if bleed=2 then output postbase ;
      run;

      data final;
```

```
         merge postbase baseline;
            by subject;
         fold = result/baseline;
         fourfold = (fold>=4);
      run;

      proc print;
      run;
```

Executing the above code generates the following result:

```
   Obs    subject    result    baseline     fold     fourfold

     1       1         226        57       3.96491       0
```

As variables RESULT and BASELINE are rounded prior to performing the ratio, variable FOLD is not close to 4 as it should be. The above output clearly demonstrates that rounding too early, whether it is done in Biostatistics, Data Management or Clinical Immunology, may result in incorrect results. Please inform your manager and/or study statistician, before database lock, if raw data is not available but only calculated values are available.

Alternatively, execution of the code below

```
      * bleed 1 is baseline, bleed 2 is post-vaccination *;

      data serologi;
         subject=1; bleed=1; test1=40; test2=80; output;
         subject=1; bleed=2; test1=160; test2=320; output;
      run;

      * get GMT for individual subject/antigen/bleed     *;

      data serology;
         set serologi;
         result = 10**mean(log10(test1),log10(test2));
      run;

      data baseline(drop=bleed rename=(result=baseline))
           postbase(drop=bleed);
         set serology(keep=subject bleed result);
         if bleed=1 then output baseline;
         else if bleed=2 then output postbase;
      run;

      data final;
         merge postbase baseline;
            by subject;
         fold = result/baseline;
         frfold = (fold>=4);
         fourfold = (round(fold,.01)>=4);
      run;

      proc print;
      run;
```

generates the output

```
      Obs    subject    result    baseline    fold    frfold    fourfold

       1        1       226.274    56.5685      4        0           1
```

This output clearly demonstrates that not using ROUND function results in variable FRFOLD taking the value 0, where as using ROUND function results in variable FOURFOLD taking the value 1. Please note that in the above case, even though the variable FOLD is displayed as 4, it is close to 4 but not exactly equal to 4, and hence, the need to use ROUND function for comparison purpose is essential.

3.3.3 GMT Calculation

Another example where ROUND function would be useful is in calculating seroprotection rates as given below:

```
      **  Possible values  ** ;

      data serology;
         do result1=., 10, 20, 40, 80, 160;
            do result2=., 10, 20, 40, 80, 160;
               do result3=., 10, 20, 40, 80, 160;
                  output;
               end;
            end;
         end;
      run;

      * GMT Base 10 *;

      data sero;
         set serology;
         gmt = 10**mean(log10(result1), log10(result2), log10(result3));
         gmtr = round(gmt,.01) ;
         seroprot = (gmt >= 40);
         seropro2 = (gmtr >= 40);
         if 39.9 <= gmt <= 40.1;
      run;

      proc print;
         title "Geometric Mean using Base 10";
      run;
```

The above code generates a result as shown in Appendix 2. In the output, SEROPROT is equal to zero even though GMT is listed as 40. However, SEROPRO2 is 1 whenever GMTR, which is obtained from GMT by ROUNDing, is 40. It is clear that the ROUND function is essential to obtain correct results.

3.3.4 Precision issues with output from SAS Procedures

Results generated by certain procedures may not be what they they appear on screen. For example, consider PROC FREQ used in the code below:

```
data TestData(Keep = Times Subject);
     Subject  = 1;
     Do Times = 1 To 100 ;
        Do Clone = 1 To Times ;
           Output;
        End;
     End;
run;

Proc Freq Data = TestData noprint;
     By Times;
     Table Subject / Out = Freq_Out;
run;

Data Final;
     Set Freq_Out;
     Format Delta e32.8 ;
     /** The statement below must NOT BE TRUE for PROC FREQ to work **/
     If Percent ^ = 100.;
     Delta  = Percent - 100. ;
     if delta = 0.0 then true = 1 ;
     else true=0 ;
     Label Delta   = 'Percent*Minus 100'
           Times   = 'Freq*Count'
           true    = 'does not*work'
           Count   = 'Numerator*and*Denominator'
           Percent = 'Percent*Computed By*Proc Freq' ;
run;

Proc Print Data = Final Label N Split = '*';
      Var Count Percent Delta true;
      Title1 "ERROR IN PROC FREQ PROCEDURE";
      Title2 "Certain percentages computed by PROC FREQ as 100% and
      stored in a SAS dataset";
      Title3 "DO NOT compare to be 100%.";
run;
```

Executing the above code generates the follwing output:

```
                       ERROR IN PROC FREQ PROCEDURE
            Certain percentages computed by PROC FREQ as 100% and
              stored in a SAS dataset DO NOT compare to be 100%.

         Numerator      Percent
            and       Computed By        Percent                    does not
  Obs   Denominator    Proc Freq        Minus 100                     work

   1        11           100        1.42108547152020000000000E-14      0
```

```
    2          22            100         1.4210854715202000000000000E-14        0
    3          39            100         1.4210854715202000000000000E-14        0
    4          44            100         1.4210854715202000000000000E-14        0
    5          78            100         1.4210854715202000000000000E-14        0
    6          83            100         1.4210854715202000000000000E-14        0
    7          88            100         1.4210854715202000000000000E-14        0
    8          91            100         1.4210854715202000000000000E-14        0
    9          97            100        -1.4210854715202000000000000E-14        0

                                                N = 9
```

Please note that the variable DELTA, which is PERCENT – 100, is not equal to zero in several instances due to numeric precision issue even though PERCENT is displayed as 100. Using ROUND function resolves this issue as given in the code below:

```
Data Final;
     Set Freq_Out;
     Format Delta e32.8 ;
     If Percent ^ = 100.;
     Delta = round(percent,.01) - 100. ;
     if round(percent,.01)=100.0 then true=1 ;
     else true = 0 ;
     if delta = 0.0 then trueb = 1 ;
     else trueb=0 ;
     Label Delta   = 'round(percent)*Minus 100'
          Times   = 'Freq*Count'
          true    = 'round*works'
          trueb   = 'round*works2'
          Count   = 'Numerator*and*Denominator'
          Percent = 'Percent*Computed By*Proc Freq' ;
run;

Proc Print Data = Final Label N Split = '*';
     Var Count Percent Delta true trueb;
     Title1 "Resolve ERROR by using ROUND function";
run;
```

Executing this code generates the following output:

```
          Numerator       Percent
             and        Computed By        round(percent)              round      round
  Obs    Denominator     Proc Freq           Minus 100                 works      works2

   1          11            100        0.0000000000000000000000000E+00     1          1
   2          22            100        0.0000000000000000000000000E+00     1          1
   3          39            100        0.0000000000000000000000000E+00     1          1
   4          44            100        0.0000000000000000000000000E+00     1          1
   5          78            100        0.0000000000000000000000000E+00     1          1
   6          83            100        0.0000000000000000000000000E+00     1          1
   7          88            100        0.0000000000000000000000000E+00     1          1
   8          91            100        0.0000000000000000000000000E+00     1          1
   9          97            100        0.0000000000000000000000000E+00     1          1

                                           N = 9
```

This output clearly shows that DELTA, which is ROUND(PERCENT) – 100, is equal to 0. Further checks also show that the comparison works as displayed in the last two columns above.

3.3.5 Precision issue while performing a MERGE

Sometimes the floating-point issue causes MERGE to fail when the merging is done by numeric variable(s). For example, consider the code

```
** precision issue while merging ** ;

data ds1(keep=pat pt x1) ;
     do i = 0 to 1 by .1 ;
         pat=10*(i +.1) ;
         pt=round(pat,.01) ;
         if pat=pt then x1=1 ; else x1=0 ;
         OUTPUT ;
     end ;
run ;
proc print ;
run ;

data ds2(keep=pat pt x2);
     set ds1(keep=pt) ;
     pat=_n_ ;
     if pat=pt then x2=1 ; else x2=0 ;
run ;
proc print ; run ;

data check ;
     merge ds1 ds2(drop=pt);
           by pat ;
run ;
proc print ; run ;
```

Executing the above code generates the output as shown in Appendix 3. It is clear from the output that both datasets DS1 and DS2 have PAT and PT numbers from 1 through 11. In DS1 dataset, the variable X1 takes the value 0 for record numbers 4 and 11 indicating that PAT is not equal to PT for these two records. However, in dataset DS2, X2 takes the value 1 for all records implying PAT equals PT for all records. Therefore, merging the datasets by PAT, as done in dataset CHECK, results in 13 records. The PAT numbers 4 and 11 do not match, and as a consequence two records are created in the merged file for these two patients.

For small datasets one can print and identify the *merge* problem visually. However, for large datasets it would be better to use PROC FREQ, to identify problems in merging. One could also create some new flags in the merged dataset and use PROC FREQ to count them. For the case mentioned above, consider the code

```
proc freq data=check;
     tables pat ;
run ;
```

Executing the code generates the following output:

```
                                                Cumulative    Cumulative
               pat     Frequency     Percent     Frequency      Percent

                 1             1        7.69             1         7.69
                 2             1        7.69             2        15.38
                 3             1        7.69             3        23.08
                 4             2       15.38             5        38.46
                 5             1        7.69             6        46.15
                 6             1        7.69             7        53.85
                 7             1        7.69             8        61.54
                 8             1        7.69             9        69.23
                 9             1        7.69            10        76.92
                10             1        7.69            11        84.62
                11             2       15.38            13       100.00
```

From the above output it is clear that PAT numbers 4 and 11 have 2 records each. Also, it is obvious that they both have one record in the initial datasets DS1 and DS2, and therefore, the merged dataset CHECK should have only one record each for these two subjects as well. In the case of large datasets, sometimes it may be necessary to perform a PROC FREQ on the individual datasets prior to merging to properly estimate the expected number of records in the merged dataset.

To resolve the aforementioned *merge*, consider the code

```
data checkr ;
     merge ds1 ds2(drop=pat);
           by pt ;
run ;
proc print ; run ;
```

Execution of this code generates the output

```
                         Obs    pat    pt    x1    x2

                           1      1     1     1     1
                           2      2     2     1     1
                           3      3     3     1     1
                           4      4     4     0     1
                           5      5     5     1     1
                           6      6     6     1     1
                           7      7     7     1     1
                           8      8     8     1     1
                           9      9     9     1     1
                          10     10    10     1     1
                          11     11    11     0     1
```

As the output shows that there are 11 records in total with each unique value of PT having only one record, it is clear that the merge works correctly. The variable PT is obtained from PAT after ROUNDing and therefore, results in correct merging of the datasets DS1 and DS2.

3.4 Truncating Numbers and Making Comparisons

The TRUNC function truncates a number to a requested length and then expands the number back to full length. The following example illustrates where a ROUND function does not work but TRUNC works.

Consider a dataset ONE having one record with three variables as follows:

```
     obs          pvalue           x1              x2

      1            0.05         0.049950        0.04995
```

executing the datastep TWO below

```
**   ROUND function  ** ;
data two ;
     set one ;
     y1 = 0 ;
     y2 = 0 ;
     x1r = round(x1,.0001);
     x2r = round(x2,.0001);
     if x1r < pvalue then y1 = 1 ;
     if x2r < pvalue then y2 = 1 ;
run ;
proc print ;
run ;
```

gives the result

```
Obs   pvalue      x1          x2         y1      y2      x1r        x2r

  1     0.05   0.049950    0.04995       1       0     0.0499      0.05
```

Please note that even though both x1 and x2 appear to be same, they are 0.0499495 and 0.04995, respectively. A similar assignment to y1 and y2 does not result in same result. That is the consequence of using ROUND function as shown in variables x1r and x2r.

However, executing the data step THREE below

```
**   TRUNC function  ** ;
data three ;
     set one ;
     z1 = 0 ;
     z2 = 0 ;
     x1t = trunc(x1,3) ;
     x2t = trunc(x2,3) ;
     if trunc(x1,4) < pvalue then z1 = 1 ;
     if trunc(x2,4) < pvalue then z2 = 1 ;
run ;
proc print ;
run ;
```

where TRUNC function is used instead of ROUND function, the follwing result is obtained

```
  Obs    pvalue      x1           x2        z1     z2       x1t          x2t

   1     0.05     0.049950     0.04995      1      1     0.049950     0.049950
```

clearly demonstrating TRUNC function is a better choice in such situations, that is when working with small numbers.

3.5 Rounding Vs. Truncating Numbers

In example 3 of Section 3.4, the TRUNC function is not appropriate as shown below.

```
data serology;
   do result1=., 10, 20, 40, 80, 160;
      do result2=., 10, 20, 40, 80, 160;
         do result3=., 10, 20, 40, 80, 160;
            output;
         end;
      end;
   end;
run;

* GMT Base 10 *;

data sero;
   set serology;
   gmt = 10**mean(log10(result1), log10(result2), log10(result3));
   gmtr = round(gmt,.01) ;
   gmtt = trunc(gmt,3) ;
   seroprot = (gmt >= 40);
   seropro2 = (gmtr >= 40);
   seropro3 = (gmtt >= 40) ;
   if 39.9 <= gmt <= 40.1;
run;

proc print;
   title "Geometric Mean using Base 10";
run;
```

When the above code is executed the result is given in Appendix 4. The output clearly shows incorrect results when a TRUNC function is used.

3.6 Transfering Data between Operating Systems

The problems of precision and magnitude when using floating-point numbers are not confined to a single operating system. Additional problems can arise when data is transferred from one operating system to another. In section 3.1, the maximum number of digits of the base, exponent and mantissa on various operating systems is mentioned. Because there are differences in the maximum values that can be stored in different operating environments, there might be problems in transferring floating-point data from one machine to another.

Transferring data from an IBM mainframe to a PC: the number of bits for the mantissa is 4 less than that for an IBM mainframe, resulting in loss of 4 bits when moving to a PC.

Caution: **Transfer of data between machines can affect numeric precision.**

3.7 Storing Numbers with Less Precision

The SAS system allows numeric variables to be stored on disk with less than full precision. LENGTH statement can be used to dictate the number of bytes that are used to store the floating-point number. The default LENGTH of a numeric variable is 8.

For example, consider the number 123456789, which would be 0.123456789 to the 9th power of 10 in base 10. If six digits of precision are used, the number becomes 123457000 (or 0.123457 to the 9th power of 10) as it rounds up.

The **only reason** to truncate length by using the LENGTH statement is to save disk space. All values are expanded to full size to perform computations in DATA and PROC steps. Extreme caution should be used when choosing a length of less than 8 for numeric variables. Do not use the LENGTH statement to truncate, if the variable values are not integers. **Fractional numbers lose precision if truncated.** Consider the following program.

```
** precision issue due to length of a numeric variable being less than 8  **;

      data issloc (drop=i j) ;
         do i= 1 to 2 ;
            patno=i;
              do j= 1 to 4 ;
               dose=j ;
               tempc1=37.2 + i/10 + j/10 ;
               tempc2=37.2 + i/10 + 2*(j/10) ;
               tempc3=37.2 + i/10 + 3*(j/10) ;
               tmpc1=tempc1 ;
               tmpc2=tempc2 ;
               tmpc3=tempc3 ;
               output ;
              end ;
         end ;
      run ;

      data loc (drop=i) ;
         do i= 1 to 2 ;
            patno=i;
              dose=6 ;
               tempc1=37.2  + 3*(i/10) ;
               tempc2=37.2  + 2*(i/10) ;
               tempc3=37.2  + i/2 ;
               tmpc1=tempc1 ;
               tmpc2=tempc2 ;
               tmpc3=tempc3 ;
               output ;
         end ;
      run ;
```

```
data issloc3 ;
     length tempc1 3 tempc2 4 tempc3 7;
     set issloc ;
run ;

data local ;
     set issloc loc ;
run ;
data local3 ;
     set issloc3 loc ;
run ;

proc sort data=local ;
     by patno dose;
run ;
proc sort data=local3 ;
     by patno dose;
run ;

data chk ;
     set local ;
       if tmpc1=tempc1 then a=1 ;
       else a=0 ;
       if tmpc2=tempc2 then b=1 ;
       else b=0 ;
       if tmpc3=tempc3 then c=1 ;
       else c=0 ;
run ;

data chk3 ;
     set local3 ;
       if tmpc1=tempc1 then a=1 ;
       else a=0 ;
       if tmpc2=tempc2 then b=1 ;
       else b=0 ;
       if tmpc3=tempc3 then c=1 ;
       else c=0 ;
run ;

title5 'Table 1 - Numeric variables with length = 8' ; title7 ' ' ;
proc print data=chk;
     var patno dose tempc1 tmpc1 a tempc2 tmpc2 b tempc3 tmpc3 c ;
run ;

title5 'Table 2 - Numeric TEMPC variables with length < 8'; title7 ' ';
proc print data=chk3 ;
     format tempc1-tempc3 tmpc1-tmpc3 4.1 ;
     var patno dose tempc1 tmpc1 a tempc2 tmpc2 b tempc3 tmpc3 c ;
run ;
```

When the above code is executed the result is given in Appendix 5. The flag **'a'** in Table 1 of the output clearly shows that the numeric variables TEMPC1 and TMPC1 are identical when they have a default length of 8, and they do not always match, in Table 2, when a length of 3 (less than

8) is used for TEMPC1. Similarly, flags 'b' and 'c' demonstrate the loss of precision when a length of 4 or 7 is used.

Therefore, it is highly recommended to use the default length of 8 for all numeric variables.

4 Logic Errors

Sometimes the programming style may introduce logic errors, which may not be easy to identify. Consider the example below which is used to calculate change from baseline for some lab analytes, such as, CHOLesterol.

```
**   Logic Errors     ** ;
data base;
input pat base ;
cards ;
   1  21.9
   2  22.9
   3  23.9
   4  24.9
   ;
run ;

data postbase;
input pat result ;
cards ;
  1  21.1
  2  22.2
  2  23.2
  2  24.2
  3  23.3
  3  24.3
  4  24.4
  4  25.4
  4  .
  4  26.4
  4  27.4
;
run ;

data all ;
     merge base postbase ;
           by pat ;
     base=base*10 ;     **  convert from UK to US units  ** ;
     result=result*10 ; **  convert from UK to US units  ** ;
run ;

proc print ;
run ;
```

Please note that variables BASE and RESULT are multiplied by a constant to convert them to US units as most clinical personnel in the US prefer to review clinical data in units that are commonly used in the US rather than in some foreign units, such as, weight in stones. Executing the above code generates the following output

```
obs    pat    result     base

  1     1       211       219
  2     2       222       229
  3     2       232      2290
  4     2       242     22900
  5     3       233       239
  6     3       243      2390
  7     4       244       249
  8     4       254      2490
  9     4         .     24900
 10     4       264    249000
 11     4       274   2490000
```

The variable BASE is exponentially increasing whenever a patient has more than one post-baseline record. This problem is due to the fact that this is a one-to-many merge, and the values of the variables in the one dataset, BASE, is retained for patients with multiple records in the many, which is POSTBASE dataset. However, the assignment *base=base*10* is intentionally changing it in each record and therefore, shows no error in the log.

To avoid this type of logic errors, please use a new name to the variable, such as, *base2=base*10*. Alternative ways to avoid this logic error are (1) perform the convertion in the individual datasets prior to merging, (2) perform the conversion in a separate dataset after merging, (3) use PROC SQL, (4) use *if first.pat then base=base*10*. The actual output that was desired is as follows:

```
obs   pat    result    base

  1     1      211       219
  2     2      222       229
  3     2      232       229
  4     2      242       229
  5     3      233       239
  6     3      243       239
  7     4      244       249
  8     4      254       249
  9     4        .       249
 10     4      264       249
 11     4      274       249
```

It is best not to use same names for variables and datasets, repeatedly, to avoid this type of logic errors. Also, while programming, it is a good habit to verify if the program is generating the expected output in each datastep and procedure.

5 Merge Considerations

One of the most powerful aspects in SAS programming is MERGE. It is useful to put together variables from two or more datasets into one dataset. Typical merges are one-to-one, many-to-one and one-to-many. Many-to-many merge does not always yield expected results, and therefore, should be avoided. Sometimes it may be necessary to merge without a by statement

5.1 Common Error

One frequently observed merge problem appears to be *common variables* that are not used in the *by* statement in the datasets being merged. Keeping numerous common variables, in two or more datasets that are being merged, and not using them in the *by* statement causes unexpected anomalies. For example, consider the following code:

```
data lab ;
   input pat vis result ;
   cards ;
    1  0  19.1
    1  1  21.1
    2  0  19.2
    2  1  22.2
    2  2  .
    2  3  24.2
    3  0  19.3
    3  1  23.3
    3  2  24.3
    4  0  19.4
    4  1  .
    4  2  .
    4  3  25.4
    4  4  26.4
    4  5  .
;
run ;

data base(drop=vis) postbase ;
     set lab ;
       if vis=0 then output base ;
       else output postbase ;
run ;

data base ;
     set base ;
     base=result ;
run ;

data final1 ;
     merge postbase base ;
           by pat ;
run ;
proc print ; run ;
```

Executing the above code generates the following output:

```
                 Obs     pat    vis      result     base

                  1       1      1        19.1     19.1
                  2       2      1        19.2     19.2
```

```
               3       2      2         .      19.2
               4       2      3      24.2      19.2
               5       3      1      19.3      19.3
               6       3      2      24.3      19.3
               7       4      1      19.4      19.4
               8       4      2         .      19.4
               9       4      3      25.4      19.4
              10       4      4      26.4      19.4
              11       4      5         .      19.4
```

From the above output it is clear that the value of RESULT when VIS=1 has been replaced with the value of BASE. This problem may be more pronounced if additional calculations are performed, such as, LOCF, as shown in the following code

```
      data final2 ;
           merge postbase base ;
                 by pat ;
           retain temp ;
           if first.pat then temp=. ;
           if result^=. then temp=result ;
           if ^first.pat then do ;
              if result=. then result=temp ;
           end ;
      run ;
      proc print ; run ;
```

execution of which generates the output

```
               Obs    pat    vis    result    base    temp

                 1     1      1      19.1     19.1    19.1
                 2     2      1      19.2     19.2    19.2
                 3     2      2      19.2     19.2    19.2
                 4     2      3      24.2     19.2    24.2
                 5     3      1      19.3     19.3    19.3
                 6     3      2      24.3     19.3    24.3
                 7     4      1      19.4     19.4    19.4
                 8     4      2      19.4     19.4    19.4
                 9     4      3      25.4     19.4    25.4
                10     4      4      26.4     19.4    26.4
                11     4      5      26.4     19.4    26.4
```

To avoid such errors, it is best to keep only the variables that are necessary in each dataset and not have any common variables that are not part of a *by* statement. To obtain the desired output, consider the code

```
      data final3 ;
           merge postbase base(keep=pat base) ;
                 by pat ;
           retain temp ;
           if first.pat then temp=. ;
           if result^=. then temp=result ;
           if ^first.pat then do ;
              if result=. then result=temp ;
           end ;
      run ;
      proc print ; run ;
```

execution of the above code generates the output

```
Obs    pat    vis    result    base    temp

  1     1      1      21.1     19.1    21.1
  2     2      1      22.2     19.2    22.2
  3     2      2      22.2     19.2    22.2
  4     2      3      24.2     19.2    24.2
  5     3      1      23.3     19.3    23.3
  6     3      2      24.3     19.3    24.3
  7     4      1        .      19.4      .
  8     4      2        .      19.4      .
  9     4      3      25.4     19.4    25.4
 10     4      4      26.4     19.4    26.4
 11     4      5      26.4     19.4    26.4
```

5.2 Many-to-Many Merge

In some situation many-to-many merge may useful. However, a merge statement within a datastep will not always yield the expected result. In this case, it is best to use PROC SQL. Consider the following example:

```
data ae ;
   input pat aeseq ae $ 11-30 ;
   cards ;
    1  1   NECK PAIN
    1  2   HEADACHE AND NAUSEA
    1  3   TACHYCARDIA
    1  4   HEADACHE AND NAUSEA
    1  5   URI
    2  1   URI
    2  2   EAR INFECTION
    2  3   TACHYCARDIA
    3  1   CHEST PAIN
    3  2   FEVER
;
run ;
proc sort data=ae ;
     by ae ;
run ;

data split ;
     input ae $ 6-25 newae $ 26-37 ;
       cards ;
       HEADACHE AND NAUSEA  HEADACHE
       HEADACHE AND NAUSEA  NAUSEA
       URI                  RUNNY NOSE
       URI                  FEVER
       URI                  CONGESTION
;
run ;
proc sort data=split ;
     by ae ;
```

```
run ;

data merg ;
     merge ae split ;
               by ae ;
run ;
proc print ; run ;

proc sql ;
     create table new1 as
     select a.*, (case
                        when b.newae=' ' then a.ae
                                 else b.newae
                             end) as newae
         from ae as a left join split as b
         on a.ae=b.ae
         order by 1, 2, 4 ;

proc print ; run ;
```

In this situation, a clinician has decided to split selected adverse events into multiple events. To ensure that these splits are completed efficiently, it is best to use the information provided by the clinician (in an Excel spreadsheet) directly in the SAS program. Alternatively, one could write "if then ...; else if ...;" statement, but that program will require modification whenever a new term is split.

6 Duration from dates

One of the most common durations calculated using dates is age of a subject on a specific visit date at the beginning of a clinical trail. On the flip side, dates could be estimated depending on the specified increment in days, weeks, months, quarters, years, etc. Such estimation would be useful in determining when a subject is eligible to move into next stage of the study.

6.1 Age Calculation

At the outset, age calculation (in years) appears to be simple. However, there could be a problem in age categorization depending on how it is calculated. For example, consider child A born on 28Jun1996, and child B born on 29Jun1996, as given in the code below.

```
data tmp ;
     childa=mdy(06,28,96) ;
     childb=mdy(06,29,96) ;
     dt=mdy(06,28,99) ;
     agea=(dt-childa)/365.25 ;
     ageb=(dt-childb)/365.25 ;
     agea2r=round(agea,.01) ;
     ageb2r=round(ageb,.01) ;
     agea3r=round(agea,.001) ;
     ageb3r=round(ageb,.001) ;
     format childa childb dt date9. ;
```

```
      run ;
      proc print ; run ;
```

Please note that on the visit date 28Jun1999, child A is 3 years old and child B is a day short of 3. But execution of the above code generates the following result:

```
 childa      childb        dt          agea       ageb    agea2r  ageb2r  agea3r   ageb3r

28JUN1996   29JUN1996   28JUN1999   2.99795    2.99521    3       3     2.998    2.995
```

This problem is due to the fact that when age is not a multiple of 4, an incorrect age is obtained using this formula. From the values of variables AGEA and AGEB, it is clear that both A and B are not 3 years old. Next, consider rounding function. Rounding to two decimals does not work as child B is incorrectly noted as 3 years old, and rounding to three decimals also does not work as child A is incorrectly noted as <3 years old. To ensure that correct age is calculated please use the TRUEAGE macro given below:

```
  ***  calculate true age of any subject  *** ;

  %MACRO TRUEAGE(INDS,DOBDT,VISDT,OUTDS,LEAP=NO) ;
      DATA &OUTDS ;
           SET &INDS ;
           IF NMISS(&DOBDT,&VISDT)=0 THEN DO;
              DOBDT = DATEPART(&DOBDT);
              VISDT = DATEPART(&VISDT);
              IF      DOBDT in (0,-1) AND VISDT in (0,-1)  THEN DO ;
                      DOBDT=&DOBDT;  VISDT=&VISDT;  END ;
              ELSE IF DOBDT in (0,-1) AND VISDT ^in (0,-1) THEN DO ;
                      DOBDT=&DOBDT;  END ;
              ELSE IF DOBDT ^in(0,-1) AND VISDT in (0,-1)  THEN DO ;
                      VISDT=&VISDT;  END ;

              IF MONTH(DOBDT) = MONTH(VISDT) AND DAY(DOBDT) = DAY(VISDT) THEN
                      AGEDER = YEAR(VISDT) - YEAR(DOBDT);
              ELSE    AGEDER = YRDIF(DOBDT,VISDT,'ACT/ACT');

              %IF %UPCASE(&LEAP)=YES  %THEN %DO ;
                 IF MONTH(DOBDT)=2 AND DAY(DOBDT)=29 THEN DO ;
                    LEAPYEAR = ROUND(YEAR(VISDT)/4 - INT(YEAR(VISDT)/4),.01);
                    IF LEAPYEAR>0 AND MONTH(DOBDT)=MONTH(VISDT)
                                  AND DAY(VISDT)=28
                                 THEN AGEDER = YEAR(VISDT) - YEAR(DOBDT);
                    IF LEAPYEAR=0 AND MONTH(DOBDT)=MONTH(VISDT)
                                  AND DAY(VISDT)=29
                                 THEN AGEDER = YEAR(VISDT) - YEAR(DOBDT);
                 END ;
              %END ;
           END;
           FORMAT DOBDT VISDT DATE9. ;
      RUN;
  %MEND ;
```

Using the above macro one can determine that the age of child A is 3, as follows:

```
      %trueage(tmp,childa,dt,tmpa) ;
      proc print ; run ;

                  Obs     DOBDT         VISDT        AGEDER
                   1    28JUN1996     28JUN1999        3
```

Using the same macro one can see that child B is less than 3 years old (in fact, 2.99587)

```
      %trueage(tmp,childb,dt,tmpb) ;
      proc print ; run ;

                  Obs     DOBDT         VISDT        AGEDER
                   1    29JUN1996     28JUN1999     2.99587
```

In this macro, using the optional 5th parameter LEAP=YES, one can define the last day of February as the birthday for anyone born on Feb 29 in a leap year. For example, consider the code

```
      data tmp ;
           childa=mdy(02,29,96) ;
           dt=mdy(02,28,03) ;
           format childa dt date9. ;
      run ;

      %trueage(tmp,childa,dt,tmpa,leap=yes) ;
      proc print ; run ;
```

execution of the above code yields the result

```
   Obs     childa          dt           DOBDT          VISDT       AGEDER       LEAPYEAR

    1     29FEB1996     28FEB2003     29FEB1996     28FEB2003       7            0.75
```

while execution of the code below

```
      %trueage(tmp,childa,dt,tmpa) ;
      proc print ; run ;
```

yields the result

```
     Obs         childa            dt          DOBDT          VISDT       AGEDER

      1      29FEB1996     28FEB2003     29FEB1996     28FEB2003     6.99770
```

6.2 INTNX function(6)

The data step function INTNX returns a SAS date value incremented by a specified number of intervals (days, weeks, months, quarters, years, etc.). INTNX's optional fourth argument determines how the SAS date is aligned before it is incremented. Possible values of the fourth argument are “beginning”, “middle”, “end”, and (new in Version 9) “sameday”, with the default value being “beginning”.

In version 8 of SAS, unexpected results are obtained when INTNX function is used to determine the intervals other than day. However, in version 9, a new alignment value, “sameday”, was added. “sameday” preserves the SAS date value's alignment within the interval before it is incremented, generating the expected results, as in the following examples. Note that 2000 and 2004 are leap years but not 2003.

SAS Statement	Description	Result	SAS date value
intnx('day','20nov2002'd,-7,"sameday");	7 days before 11/20/02	15657	November 13, 2002
intnx('month','20nov2002'd,1,"sameday");	1 month after 11/20/02	15694	December 20, 2002
intnx('year','20nov2002'd,-1,"sameday");	1 year before 11/20/02	15299	November 20, 2001
intnx('year','29feb2000'd,1,"sameday");	1 year after 2/29/00	15034	February 28, 2001
intnx('year','29feb2000'd,4,"sameday");	4 years after 2/29/00	16130	February 29, 2004
intnx('month','31mar2003'd,-1,"sameday");	1 month before 3/31/03	15764	February 28, 2003
intnx('month','31mar2004'd,-1,"sameday");	1 month before 3/31/04	16130	February 29, 2004

7 New Problems

New problems, with examples and solutions, will be added in the future as and when programmers and statisticians identify them.

References List

1 www.ansi.org

2 www.sas.com

3 SAS Language Reference: Concepts: Version 8

4 www.fda.gov

5 www.ieee.org

6 The INTNX Function Alignment Value SAMEDAY, Proceedings of SUGI 31

Appendix 1 : Output for Section Temperature Conversion.

```
Obs      tempf       tempc         tempcr    severity     severtyr

  1      100.0      37.7778       37.7778       0             0
  2      100.1      37.8333       37.8333       0             0
  3      100.2      37.8889       37.8889       0             0
  4      100.3      37.9444       37.9444       0             0
  5      100.4      38.0000       38.0000       0             1
  6      100.5      38.0556       38.0556       1             1
  7      100.6      38.1111       38.1111       1             1
  8      100.7      38.1667       38.1667       1             1
  9      100.8      38.2222       38.2222       1             1
 10      100.9      38.2778       38.2778       1             1
 11      101.0      38.3333       38.3333       1             1
 12      101.1      38.3889       38.3889       1             1
 13      101.2      38.4444       38.4444       1             1
 14      101.3      38.5000       38.5000       1             2
 15      101.4      38.5556       38.5556       2             2
 16      101.5      38.6111       38.6111       2             2
 17      101.6      38.6667       38.6667       2             2
 18      101.7      38.7222       38.7222       2             2
 19      101.8      38.7778       38.7778       2             2
 20      101.9      38.8333       38.8333       2             2
 21      102.0      38.8889       38.8889       2             2
 22      102.1      38.9444       38.9444       2             2
 23      102.2      39.0000       39.0000       2             2
 24      102.3      39.0556       39.0556       2             2
 25      102.4      39.1111       39.1111       2             2
 26      102.5      39.1667       39.1667       2             2
 27      102.6      39.2222       39.2222       2             2
 28      102.7      39.2778       39.2778       2             2
 29      102.8      39.3333       39.3333       2             2
 30      102.9      39.3889       39.3889       2             2
 31      103.0      39.4444       39.4444       2             2
 32      103.1      39.5000       39.5000       2             3
 33      103.2      39.5556       39.5556       3             3
 34      103.3      39.6111       39.6111       3             3
 35      103.4      39.6667       39.6667       3             3
```

Appendix 2 : Output for Section GMT Calculation.

```
 Obs     result1     result2     result3       gmt     gmtr  seroprot    seropro2

   1           .           .          40        40       40         0           1
   2           .          10         160        40       40         0           1
   3           .          20          80        40       40         1           1
   4           .          40           .        40       40         0           1
   5           .          40          40        40       40         0           1
   6           .          80          20        40       40         1           1
   7           .         160          10        40       40         0           1
   8          10           .         160        40       40         0           1
   9          10          40         160        40       40         0           1
  10          10          80          80        40       40         0           1
  11          10         160           .        40       40         0           1
  12          10         160          40        40       40         0           1
  13          20           .          80        40       40         1           1
  14          20          20         160        40       40         0           1
  15          20          40          80        40       40         0           1
  16          20          80           .        40       40         1           1
  17          20          80          40        40       40         0           1
  18          20         160          20        40       40         0           1
  19          40           .           .        40       40         0           1
  20          40           .          40        40       40         0           1
  21          40          10         160        40       40         0           1
  22          40          20          80        40       40         0           1
  23          40          40           .        40       40         0           1
  24          40          40          40        40       40         0           1
  25          40          80          20        40       40         0           1
  26          40         160          10        40       40         0           1
  27          80           .          20        40       40         1           1
  28          80          10          80        40       40         0           1
  29          80          20           .        40       40         1           1
  30          80          20          40        40       40         0           1
  31          80          40          20        40       40         0           1
  32          80          80          10        40       40         0           1
  33         160           .          10        40       40         0           1
  34         160          10           .        40       40         0           1
  35         160          10          40        40       40         0           1
  36         160          20          20        40       40         0           1
  37         160          40          10        40       40         0           1
```

Appendix 3 : Output for Section Precision issue while performing a MERGE.

```
                    Dataset DS1


           Obs     pat     pt     x1

             1       1      1      1
             2       2      2      1
             3       3      3      1
             4       4      4      0
             5       5      5      1
             6       6      6      1
             7       7      7      1
             8       8      8      1
             9       9      9      1
            10      10     10      1
            11      11     11      0


                    Dataset DS2


           Obs     pt     pat     x2

             1      1       1      1
             2      2       2      1
             3      3       3      1
             4      4       4      1
             5      5       5      1
             6      6       6      1
             7      7       7      1
             8      8       8      1
             9      9       9      1
            10     10      10      1
            11     11      11      1
```

```
                 Dataset CHECK

        Obs     pat     pt     x1     x2

          1       1      1      1      1
          2       2      2      1      1
          3       3      3      1      1
          4       4      .      .      1
          5       4      4      0      .
          6       5      5      1      1
          7       6      6      1      1
          8       7      7      1      1
          9       8      8      1      1
         10       9      9      1      1
         11      10     10      1      1
         12      11     11      0      .
         13      11      .      .      1
```

Appendix 4 : Output for Section Rounding Vs. Truncating Numbers.

```
Obs    result1     result2    result3    gmt    gmtr       gmtt     seroprot    seropro2    seropro3

  1        .           .         40       40     40      39.9922        0           1           0
  2        .          10        160       40     40      39.9922        0           1           0
  3        .          20         80       40     40      40.0000        1           1           1
  4        .          40          .       40     40      39.9922        0           1           0
  5        .          40         40       40     40      39.9922        0           1           0
  6        .          80         20       40     40      40.0000        1           1           1
  7        .         160         10       40     40      39.9922        0           1           0
  8       10           .        160       40     40      39.9922        0           1           0
  9       10          40        160       40     40      39.9922        0           1           0
 10       10          80         80       40     40      39.9922        0           1           0
 11       10         160          .       40     40      39.9922        0           1           0
 12       10         160         40       40     40      39.9922        0           1           0
 13       20           .         80       40     40      40.0000        1           1           1
 14       20          20        160       40     40      39.9922        0           1           0
 15       20          40         80       40     40      39.9922        0           1           0
 16       20          80          .       40     40      40.0000        1           1           1
 17       20          80         40       40     40      39.9922        0           1           0
 18       20         160         20       40     40      39.9922        0           1           0
 19       40           .          .       40     40      39.9922        0           1           0
 20       40           .         40       40     40      39.9922        0           1           0
 21       40          10        160       40     40      39.9922        0           1           0
 22       40          20         80       40     40      39.9922        0           1           0
 23       40          40          .       40     40      39.9922        0           1           0
 24       40          40         40       40     40      39.9922        0           1           0
 25       40          80         20       40     40      39.9922        0           1           0
 26       40         160         10       40     40      39.9922        0           1           0
 27       80           .         20       40     40      40.0000        1           1           1
 28       80          10         80       40     40      39.9922        0           1           0
 29       80          20          .       40     40      40.0000        1           1           1
 30       80          20         40       40     40      39.9922        0           1           0
 31       80          40         20       40     40      39.9922        0           1           0
 32       80          80         10       40     40      39.9922        0           1           0
 33      160           .         10       40     40      39.9922        0           1           0
 34      160          10          .       40     40      39.9922        0           1           0
 35      160          10         40       40     40      39.9922        0           1           0
 36      160          20         20       40     40      39.9922        0           1           0
 37      160          40         10       40     40      39.9922        0           1           0
```

Appendix 5 : Output for Section Storing Numbers with Less Precision

```
                 Table 1 - Numeric variables with length = 8

 Obs   patno   dose   tempc1   tmpc1   a   tempc2   tmpc2   b   tempc3   tmpc3   c

   1     1       1     37.4     37.4   1    37.5     37.5   1    37.6     37.6   1
   2     1       2     37.5     37.5   1    37.7     37.7   1    37.9     37.9   1
   3     1       3     37.6     37.6   1    37.9     37.9   1    38.2     38.2   1
   4     1       4     37.7     37.7   1    38.1     38.1   1    38.5     38.5   1
   5     1       6     37.5     37.5   1    37.4     37.4   1    37.7     37.7   1
   6     2       1     37.5     37.5   1    37.6     37.6   1    37.7     37.7   1
   7     2       2     37.6     37.6   1    37.8     37.8   1    38.0     38.0   1
   8     2       3     37.7     37.7   1    38.0     38.0   1    38.3     38.3   1
   9     2       4     37.8     37.8   1    38.2     38.2   1    38.6     38.6   1
  10     2       6     37.8     37.8   1    37.6     37.6   1    38.2     38.2   1
```

```
              Table 2 - Numeric TEMPC variables with length < 8

 Obs   patno   dose   tempc1   tmpc1   a   tempc2   tmpc2   b   tempc3   tmpc3   c

   1     1       1     37.4     37.4   0    37.5     37.5   0    37.6     37.6   0
   2     1       2     37.5     37.5   0    37.7     37.7   0    37.9     37.9   0
   3     1       3     37.6     37.6   0    37.9     37.9   0    38.2     38.2   0
   4     1       4     37.7     37.7   0    38.1     38.1   0    38.5     38.5   0
   5     1       6     37.5     37.5   1    37.4     37.4   0    37.7     37.7   0
   6     2       1     37.5     37.5   0    37.6     37.6   0    37.7     37.7   0
   7     2       2     37.6     37.6   0    37.8     37.8   0    38.0     38.0   0
   8     2       3     37.7     37.7   0    38.0     38.0   0    38.3     38.3   0
   9     2       4     37.8     37.8   0    38.2     38.2   0    38.6     38.6   0
  10     2       6     37.8     37.8   0    37.6     37.6   0    38.2     38.2   0
```