PharmaSUG 2023 - Paper QT-222

A SAS® System Macro to Quickly Display Discrepant Values that are too Long for the COMPARE Procedure Output

Kevin R. Viel, Ph.D., Navitas Data Sciences; Histonis, Incorporated

ABSTRACT

At times, values being compared by the SAS® System COMPARE procedure are truncated in the output (.lst) because they are too long and the differences are not shown. While the COMPARE procedure can produce various data sets, that approach can be cumbersome. Instead, the programmer might want to display the discrepant values in the log or redirect the log to a permanent file for a quick view. The **goal** of this paper is to introduce a SAS macro that quickly displays these overly long values in the log for quick comparison.

INTRODUCTION

Independent Validation usually involves a 100% match on the data set of interest, whether it is a **CDISC** (Clinical Data Interchange Standards Consortium) **SDTM** (Study Data Tabulation Model) domain, a CDISC **ADaM** (Analysis Data Model) data set, or a **VALDS** (Validation Data Sets from Tables, Figures, and Listings programs, also known as **TFL**s). For Validation programming using the SAS® System, the COMPARE procedure is nearly ubiquitous and one may not infrequently see the abbreviation **DP-PC** (Double Programming-Proc COMPARE) in SOPs (standard operating procedures) and WIs (work instructions).

Depending on the Validation Plan and applicable SOPs/WIs, the Validation programmer may need to exactly match:

- Variable names
- Variable attributes (type, label, and length)
- Variable Order within the data set (essential for SDTM domains)
- Variable Values

This paper pertains to matching of *values* in the Validation process. With an exact match of the Main and Validation data sets, the results, i.e. the COMPARE output in the .lst file or Output window of an interactive SAS session, are the minimal possible and easy to interpret. When discrepancies occur, investigating the origins of them can be complex and challenging, i.e., time consuming, especially if the Main and Validation data sets are not correctly sorted on the key variables. A further complication occurs when values are too long to be fully displayed in the COMPARE output, in which case they may appear to match exactly to the point of truncation. The **goal** of this paper is to introduce a SAS macro that quickly displays these overly long values fully in the log or a file for quick comparison, with flexibility in the format of the display, and the option to show a byte-by-byte comparison, including showing the hexadecimal (the SAS HEX. format) valued to emphasize discrepant non-printable bytes. The second **goal** of this paper is to introduce another SAS macro to display values with leading spaces, an issue that is notoriously hard to spot in COMPARE procedure output. The third **goal** of this paper is to discuss additions to values solely to create formatting, modifications such as leading spaces, versus using logic and styles to achieve selective formatting with the least complex value possible, i.e., to expedite Validation and minimize the data and comparisons required.

COMPARE RESULTS

The following data steps and COMPARE procedure generates the output in **Display 1**:

```
data one ;
  x   = 8 ;
  id1 = 1 ;
  id2 = "LONG INDENTIFICATION A" ;
run ;

data two ;
  set one ;
  x   = 9 ;
run ;

proc compare
   data = one
   comp = one
   listall
   ;
run ;
```

The desired scenario of an exact match is assured because the base data set and the comparison data set are the same (WORK.ONE).

```
 1  The COMPARE Procedure
 2  Comparison of WORK.ONE with WORK.ONE
 3  (Method=EXACT)
 4
 5  Data Set Summary
 6
 7  Dataset            Created          Modified  NVar    NObs
 8
 9  WORK.ONE  05FEB23:14:46:00  05FEB23:14:46:00     3       1
10  WORK.ONE  05FEB23:14:46:00  05FEB23:14:46:00     3       1
11
12
13  Variables Summary
14
15  Number of Variables in Common: 3.
16  Number of ID Variables: 2.
17
18
19  Observation Summary
20
21  Observation      Base  Compare  ID
22
23  First Obs           1        1  id1=1 id2=LONG INDENTIFICATION A
24  Last  Obs           1        1  id1=1 id2=LONG INDENTIFICATION A
25
26  Number of Observations in Common: 1.
27  Total Number of Observations Read from WORK.ONE: 1.
28  Total Number of Observations Read from WORK.ONE: 1.
29
30  Number of Observations with Some Compared Variables Unequal: 0.
31  Number of Observations with All Compared Variables Equal: 1.
32
33  NOTE: No unequal values were found. All values compared are exactly
34  equal.
```

Display 1. COMPARE procedure output for exact matches.

Note that values of the ID2 variables (lines 23 and 24) show the entire length: “LONG INDENTIFICATION A”. Contrast this with **Display 2**, specifically, line 60, which displays only the truncated value: “LONG INDENTIFICATION” for ID2.

```
 1  The COMPARE Procedure
 2  Comparison of WORK.ONE with WORK.TWO
 3  (Method=EXACT)
 4
 5  Data Set Summary
 6
 7  Dataset            Created          Modified  NVar    NObs
 8
 9  WORK.ONE  05FEB23:14:46:00  05FEB23:14:46:00     3       1
10  WORK.TWO  05FEB23:14:46:00  05FEB23:14:46:00     3       1
11
12
13  Variables Summary
14
15  Number of Variables in Common: 3.
16  Number of ID Variables: 2.
17
18
19  Observation Summary
20
21  Observation      Base  Compare  ID
22
23  First Obs           1        1  id1=1 id2=LONG INDENTIFICATION A
24  First Unequal       1        1  id1=1 id2=LONG INDENTIFICATION A
25  Last  Unequal       1        1  id1=1 id2=LONG INDENTIFICATION A
26  Last  Obs           1        1  id1=1 id2=LONG INDENTIFICATION A
27
28  Number of Observations in Common: 1.
29  Total Number of Observations Read from WORK.ONE: 1.
30  Total Number of Observations Read from WORK.TWO: 1.
31
32  Number of Observations with Some Compared Variables Unequal: 1.
33  Number of Observations with All Compared Variables Equal: 0.
34
35
36  Values Comparison Summary
37
38  Number of Variables Compared with All Observations Equal: 0.
39  Number of Variables Compared with Some Observations Unequal: 1.
40  Total Number of Values which Compare Unequal: 1.
41  Maximum Difference: 1.
42
43  The COMPARE Procedure
44  Comparison of WORK.ONE with WORK.TWO
45  (Method=EXACT)
46
47  All Variables Compared have Unequal Values
48
49  Variable  Type  Len  Ndif   MaxDif
50
51  x         NUM     8     1    1.000
52
53  Value Comparison Results for Variables
54
55  ____________________________________________________________________________
56                                  ||       Base    Compare
57       id1  id2                   ||          x          x      Diff.     % Diff
58   _______  ____________________  ||  _________  _________  _________  _________
59                                  ||
60         1  LONG INDENTIFICATION  ||     8.0000     9.0000     1.0000    12.5000
61  ____________________________________________________________________________
62
```

Display 2. COMPARE procedure with a mismatch and truncation of an ID variable value.

When one searches for the observations of interest, one might not find the observations, but instead received:

```
NOTE: No observations were selected from data set WORK.ONE.
NOTE: There were 0 observations read from the data set WORK.ONE.
      WHERE (id1=1) and (id2='LONG INDENTIFICATION');
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.05 seconds
      cpu time            0.00 seconds
```

Depending on the data, one might not easily detect which (group of) observations to investigate. For instance, a possible DUT (Device Under Test), or UAT (User Acceptance Testing), program might include:

```
%let usub =   "LONG INDENTIFICATION A1"
            , "LONG INDENTIFICATION A2"
            ;

proc print
   data = adam.adlb
            ( keep  = usubjid
                      visitnum
                      paramcd
                      lbtestcd
                      aval
              where = (     lbtestcd = "ALB"
                        and usubjid in
                                ( &usub. )
                      )
            )
   ;
run ;

proc print
   data = v_adlb
            ( keep  = usubjid
                      visitnum
                      paramcd
                      lbtestcd
                      aval
              where = (     lbtestcd = "ALB"
                        and usubjid in
                                ( &usub. )
                      )
            )
   ;
run ;

proc print
   data = sdtm.lb
            ( keep  = usubjid
                      visitnum
                      lbtestcd
                      lbstresn
              where = (     lbtestcd = "ALB"
                        and usubjid in
                                ( &usub. )
                      )
            )
   ;
run ;
```

In this case, suspicion is immediate on ID2 since ID1 is numeric variable. We know both values match, so we might switch to the “in :” operator instead of “in”, but that could result in voluminous output. Discerning the exact value and being able to copy and paste is preferrable.

MAC_R_LONG_COMPARE RESULTS

One could not copy the correct value of ID2 from the output in Display 2 for an observation of interest because the value is truncated, perhaps undetectably unless one is rather familiar with the data; which of the values is truncated might not be obvious if the ID variables include several “long” variables. The macro, MAC_R_LONG_COMPARE, overcomes this issue by rapidly displaying the values in either the log or directing the results to a file. The latter options does not pertain to most aspects of programming in clinical trials that follow CDISC guidance since the maximum length of variables is 200 bytes, which can be fully displayed in a SAS log. For issues concerning long DNA, RNA, or amino acid sequences, the reader may wish to consider BLAST®[1,2] and the SAS macro previously presented[3,4] noting that an updated version of MAC_U_BLAST is available upon request.

Appendix 1 presents MAC_R_LONG_COMPARE and invoking the HELP option of MAC_R_LONG_COMPARE results in a short description of the macro in the log:

```
%mac_r_long_compare
          ( help = Y ) ;
_____________________________________________________________________________________
Purpose of program:        This macro displays values of mismatching variables that cannot be
                           displayed fully by the COMPARE procedure output due to truncation.

Macro Parameter             Description
_________________          _____________________________________________________________
data                     = Name of the base data set.
comp                     = Name of the compare data set.
out                      = Name of the output data set.
                               Default: __lc
var                      = Variable to compare.
id                       = ID variables.
id_rows                  = Whether to put one ID variable per row.
                               Default: N
where                    = Where clause.
                               Default: 1
differences              = Whether to use a bar/hyphen to emphasize differences.
                               Default: N
transpose                = The threshold of the length to transpose the values.
                               Default: 256
format                   = The format to display the values when not transposed.
                               Default: %str()
byte_format              = The format to display the individual bytes of the values when transposed.
                               Default: $char1.
both_formats             = When transposed and a HEXw. format is used, whether to display both the
                           values with a HEXw. format and with a $CHAR1. format.
                               Default: Y
file                     = The path and filename to FILE (write) the results (discrepancies).
                               Default: %str()

_____________________________________________________________________________________
End of help
```

Display 3 presents two calls of MAC_R_LONG_COMPARE and their respective output for what should be the two most frequent uses of it. Note that the entire value of ID2 is displayed, i.e. no truncation, in contrast to the COMPARE results in Display 2 (line 60). Given that the most likely ID variables will be the keys of the SDTM domain or ADaM data sets, and thus could be five or more variables, some with long values, i.e., USUBJID, PARAM, VISIT, displaying each variable-value pair on its own row might be preferred (right panels of Display 3).

%mac_r_long_compare (data = one , comp = two , var = x , id = id1 id2) ;	%mac_r_long_compare (data = one , comp = two , var = x , id = id1 id2 , id_rows = Y) ;
id1=1 id2=LONG INDENTIFICATION A 8 9	id1=1 id2=LONG INDENTIFICATION A 8 9

Display 3. Two typical uses of MAC_R_LONG_COMPARE.

Another use of MAC_R_LONG_COMPARE is to elucidate values that appear to match in the output but one or both have one or more leading spaces. The presence of leading spaces may be obvious, but the number of leading spaces is not readily discernable. The following two data steps and COMPARE procedure produces the subset of the output in **Display 4**:

```
data three ;
  length x $ 30 ;
  x   = "    9 (10)" ;
  id1 = 1 ;
  id2 = "LONG INDENTIFICATION A" ;
  output ;
  x   = "   18 (20)" ;
  id1 = 2 ;
  id2 = "LONG INDENTIFICATION B" ;
  output ;
run ;

data four ;
  length x $ 30 ;
  x   = "   9 (10)" ;
  id1 = 1 ;
  id2 = "LONG INDENTIFICATION A" ;
  output ;
  x   = "  18 (20)" ;
  id1 = 2 ;
  id2 = "LONG INDENTIFICATION B" ;
  output ;
run ;

proc compare
   data = three
   comp = four
   listall
   ;
  id id1
     id2
     ;
run ;
```

```
Value Comparison Results for Variables
_____________________________________________________________________
                                ||  Base Value          Compare Value
    id1  id2                    ||  x                    x
 _______ ____________________   ||  ___________________+ ___________________+
                                ||
       1  LONG INDENTIFICATION  ||      9 (10)              9 (10)
       2  LONG INDENTIFICATION  ||     18 (20)             18 (20)
_____________________________________________________________________
```

Display 4. Partial COMPARE output demonstrating leading spaces mismatches.

Depending on where the mismatching observations occur in the list, one may not discern a leading space or two. With some experience, when one cannot see the mismatching characters, one suspects leading spaces, especially for VALDS's in which a programmer may use leading spaces to format the output (PDF/RTF) or fail to use LEFT() or STRIP() when converting a numeric variable to its character version. One manual approach would be to open the data set in the SAS Explorer in an interactive session, find the observation(s) of interest, copy the value and paste it to an editor. One may need to manually increase the width of the variable in the SAS Explorer to see the entire value (and potential leading spaces); the next section discusses this is in more detail. **Display 5** demonstrates a quicker approach using MAC_R_LONG_COMPARE.

```
%mac_r_long_compare
   ( data         = three
   , comp         = four
   , var          = x
   , id           = id1
                    id2
   ) ;
```

```
id1=1 id2=LONG INDENTIFICATION A
    9 (10)
   9 (10)


id1=2 id2=LONG INDENTIFICATION B
   18 (20)
  18 (20)
```

```
%mac_r_long_compare
   ( data         = three
   , comp         = four
   , var          = x
   , id           = id1
                    id2
   , differences = Y
   ) ;
```

```
id1=1 id2=LONG INDENTIFICATION A
    9 (10)
   |||||||
   9 (10)

id1=2 id2=LONG INDENTIFICATION B
   18 (20)
  ||||||||
  18 (20)
```

Display 5. MAC_R_LONG_COMARE output demonstrating leading spaces mismatches.

The limitation of this approach is that one may still need to count the number of spaces by placing the cursor in column one and moving it using the right arrow key. That these values differ by one leading space is readily obvious from this output, however. Using the value of 0 (zero) for the macro parameter TRANSPOSE forces the display to be transposed, that is, to be displayed byte-by-byte across the rows of the log (or file). **Display 6** demonstrates this effect with the number of leading spaces being easily read from the row number.

```
%mac_r_long_compare
   ( data         = three
   , comp         = four
   , var          = x
   , id           = id1
                    id2
   , transpose    = 0
   ) ;
```

```
id1=1 id2=LONG INDENTIFICATION A
1
2
3
4           9
5         9
6           (
7         ( 1
8         1 0
9         0 )
10        )


id1=2 id2=LONG INDENTIFICATION B
1
2
3           1
4         1 8
5         8
6           (
7         ( 2
8         2 0
9         0 )
10        )
```

```
%mac_r_long_compare
   ( data         = three
   , comp         = four
   , var          = x
   , id           = id1
                    id2
   , differences  = Y
   , transpose    = 0
   ) ;
```

```
id1=1 id2=LONG INDENTIFICATION A
1
2
3
4           - 9
5         9 -
6           - (
7         ( - 1
8         1 - 0
9         0 - )
10        ) -


id1=2 id2=LONG INDENTIFICATION B
1
2
3           - 1
4         1 - 8
5         8 -
6           - (
7         ( - 2
8         2 - 0
9         0 - )
10        ) -
```

Display 6. MAC_R_LONG_COMPARE output demonstrating leading spaces mismatches.

Another scourge of validation using the COMPARE procedure is the presence of non-printable characters. The following two data steps and COMPARE procedure produces the subset of the output in **Display 7**:

```
data five ;
length x $ 4 ;
  x   = cat( "ab"
           , "0A"x
           , "c"
           ) ;
  id1 = 1 ;
  id2 = "LONG INDENTIFICATION A" ;
run ;

data six ;
  x   = "abc" ;
  id1 = 1 ;
  id2 = "LONG INDENTIFICATION A" ;
run ;

proc compare
   data = five
   comp = six
   listall
   ;
run ;
```

```
Variables with Unequal Values

Variable  Type  Len1 Len2  Ndif   MaxDif

x         CHAR     4    3     1


Value Comparison Results for Variables

_____________________________________________________________
            ||  Base Value           Compare Value
       Obs  ||  x                     x
 ________   ||  ____                  ___
            ||
        1   ||  abc                   abc
_____________________________________________________________
```

Display 7. Partial COMPARE output demonstrating a mismatch due to non-printable characters.

A keen eye might have noted the LENGTH statement in the data step that created data set FIVE *and* the difference between LEN1 and LEN2 in the COMPARE results, but that is likely an artifact of this contrived example; also including that LENGTH statement in the data step that created data set SIX would not affect this example. Further in independent Validation, one does not view the code or log of the peer program. While the Validation Plan and/or applicable SOP/WI might require a match on the *lengths* of variables, one can conclude that other *values* of X in data set FIVE could not have been length four because that would obviously create more mismatches since no value of X in data set SIX could have been length four, ignoring the fact that both data sets contain but one observation for this example. Those who have experienced this extreme consternation (and the pressure it creates with regards to deadlines), may have resorted to examining the hexadecimal representation of the values, and reasonably display them byte-by-byte using a DO-LOOP and SUBSTR(). This is the approach used to create the output in **Display 8**.

```
%mac_r_long_compare
   ( data         = five
   , comp         = six
   , var          = x
   , id           = id1
                    id2
   , differences  = Y
   , transpose    = 0
   , byte_format  = hex.
   , both_formats = N
   ) ;
```

```
id1=1 id2=LONG INDENTIFICATION A
1        61   61
2        62   62
3        0A - 63
4        63 - 20
```

```
%mac_r_long_compare
   ( data         = five
   , comp         = six
   , var          = x
   , id           = id1
                    id2
   , differences  = Y
   , transpose    = 0
   , byte_format  = hex.

   ) ;
```

```
id1=1 id2=LONG INDENTIFICATION A
1        a  61   61  a
2        b  62   62  b
3          0A - 63  c
4        c  63 - 20
```

Display 8. A side-by-side comparison of a byte-by-byte display of the hexadecimal representation of each character.

From this output, it is easy to see that the values mismatches starting at byte 3. The mismatch is even more obvious in the bottom right panel of Display 8 because of the “deformity” caused by the spacing when the original values are also displayed. "0A"x is, again, “0 length”. The values in Display 7 both appear to be “abc”, but note that the Compare Value is offset one byte to the left of the variable x and the underline (both of the latter start under the “o” of Compare, but “a” of “abc” starts under the “C” of

“Compare”. Note that if one copies the value from the log or from the data set opened in the SAS Explorer and pastes it to the SAS Enhanced Editor, the “c” of “abc” will be on the next line:

```
ab
c
```

which is expected since "0A"x is the line feed byte. One could (re)write the macro to COMPRESS(), for instance, the non-printable character using the third argument “KW” (Keep Written), to avoid the deformity, but the emphasis is appealing to the author and his eyes and one can copy the value origin byte from the log, which has some other utilities.

Another issue with the output in the bottom panels of Display 8 might catch a keen eye. The length of the variable X is 3 (three) bytes in data set SIX. Why does the sequence of bytes show that the fourth value is "20"x, a space (a byte, that does not exist in the value)? Would not displaying no value be a better representation? Indeed, but this is supposed to be a “quick and dirty” macro, a service macro. The discussion of the code below will elucidate the cause and provide a justification of the reason. Under no circumstances would this macro be used in Production (formal) execution or be part of a final Validation program, but it *should* be retained in a (dated) DUT/UAT program, especially for the astute Validation programmer who retains such programs as alternative or supplemental documentation to the Validation process beyond an issue tracker. Note that upon correction of the issue(s) that created a value with non-printable characters, Validation becomes more rapid, but the DUT/UAT program/code investigating mismatches can be “versioned” to document the original issue and it can be re-used for the next tasks.

MAC_R_LEAD_SPACES

Appendix 2 presents the macro, MAC_R_LEAD_SPACES, which displays any value with a leading space after replacing spaces with underscores and, optionally, removes those leading spaces with the STRIP() function. The following call demonstrates a macro using data set THREE above:

```
%mac_r_lead_spaces
  ( ds     = three ) ;
```

The SAS log shows that two values of the variable X have leading spaces:

```
x                                    ____9_(10)_____________________
x                                    ___18_(20)_____________________
```

For the purposes of investigation, one might wished to COMPARE the values of two data sets without leading spaces. In that case, one can use the macro parameter REMOVE= and OUT= as follows:

```
%mac_r_lead_spaces
  ( ds     = three
  , out    = tres
  , remove = Y
  , list   = N
  ) ;

proc compare
   data = three
   comp = tres
   listall
   ;
  id id1
     id2
     ;
run ;
```

			Base Value	Compare Value
id1	id2		x	x
1	LONG INDENTIFICATION		9 (10)	9 (10)
2	LONG INDENTIFICATION		18 (20)	18 (20)

The Validation programmer certainly would *not* want to modify a permanent data set or file created by the Main programmer but using the macro to output a new (temporary) data set for investigational purposes in a non-production program could be informative. In this case, following whatever processes the Validation plan, SOPs, and WIs instruct, the Validation programmer could update the issue tracker and send an email to the Production programmer that *informally*, i.e., in a non-production (unofficial) run, a match occurs on the values without leading spaces and request either that the Production programmer report the algorithm that decides how many leading spaces a value should have or that the leading spaces be removed and to re-submit the Production program for another cycle of Validation.

INCLUDING FORMATTING IN VALUES

To increase the accuracy of Validation and to expedite the process, creating values of variables that are the minimal complexity with the maximum clarity is essential. The author presented a poster[5] describing the structure (variable names, order, labels, types, and formats) of a standardized VALDS for Table and Listings programs. The code in **Appendix 3** presents a data step and REPORT procedure to achieve indentation, for instance, the row labels, like “Female”, under section headers, like “Sex n(%)”, are typically indented. Using actual leading spaces can lead to problems and not only in Validation, but also in formatting of the output (PDF/RTF file), for instance, if the width of the column needs to be shorted or the value wraps due to the width of the column.

Display 9 presents the data set, INDENT, created by the data step in lines 1-64 of Appendix 3. Even though more involved than manually type leading spaces, if one may prefer to use the SAS function REPEAT(), on line 38, since one can explicitly see how many spaces were inserted. In this data set, one can see that the value of COLUMN_1 in observation 3 has leading spaces, if one expands the length of the column in the SAS Explorer (as the author did for Display 9).

	row_1	column_1	row_order_1
1	Plain	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9	1
2	Indent: style = {indent = 4%}	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9	2
3	5 Leading spaces in value	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9	3
4	^{style [textdecoration = underline]Left Margin}: ^{style [fontweight = medium color = black]style = ^{unicode 007B} leftmargin = 4% ^{unicode 007D}}	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9	4

Display 9. The data set INDENT created by the data step in Appendix 3.

The value of ROW_1 of observation 4 demonstrates the complexity of adding inline formatting to a value and that the length of the variable needs to be much greater than its intended display length. Note that the lengths of character variables in the proposed standardized VALDS[5] are 200 by default, since this is the CDISC and Version 5 SAS Transport file (.xpt) maximum.

Display 10 present the PDF output of this code. Note that both the Style INDENT and leading spaces approach will not likely create acceptable wrapping of long text. The Style LEFTMARGIN, however, insures that any text that wraps is also properly indented. Both values subject to formatting by the REPORT procedure do not have leading spaces. Instead, the code in the COMPUTE block on lines 118-131 achieves the selective formatting and was demonstrated by Cynthia Zender[6] on SAS Communities.

row_1	column_1
Plain	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9
Indent: style = { indent = 4% }	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9
5 Leading spaces in value	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9
Left Margin: style ={ leftmargin = 4%}	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9

Display 10. The PDF output created by the REPORT procedure in Appendix 3.

MAC_R_LONG_COMPARE CODE

The macro is standard fare. Lines 99-141 create a SQL procedure that creates a data set containing observations with mismatching values in the variable of interest via an inner join of observations that match on the ID variables. If this data set contains observations, then the macro determines the type (char or num) of the variable from the SASHELP.VCOLUMN view (lines 147-161). If that variable is character, it then determines the maximum length of the values (lines 166-172). The remainder of the macro creates a data step to display the mismatching values for the given ID variables values.

Of note, instead of using column pointers, i.e. "@10" for each item, the columns are determined by the +n pointer. With slight effort, if desired, that can be changed so that the "deformity" discussed above can be rectified. Reading the output programmatically might be one reason to make the change.

The use of regular expressions is worth explaining, for example on line 221:

```
%sysfunc( prxchange( s/([_a-z][_a-z0-9]{0,31})/$1=/i , -1 , &id. ))
```

The first argument is a substitution with the "/" delimiter: "s///". The pattern "[_a-z][_a-z0-9]{0,31}" is within a grouping (). If a string in the value matches that pattern, then that string is retained in memory. The value in the second part of the substitution is what is substituted. In this case $1 refer to the first (and only) grouping. So the result returned by that code is that string followed by an equal sign. The pattern is by bytes, "[_a-z]" matches one byte that is either an underscore or a letter. In this case the "i" following "s///" directs that match to be case insensitive, so "a" or "A". The "[_a-z0-9]" extends this match to include the characters 0-9 (*character* versions of digits). The "{0,31}" following that pattern instructs the regular expression engine to match 0 to 31 bytes following the first byte. Take in total, these are the rules for a SAS variable as invoked by the SAS System Option VALIDVARNAME = V7. Note that the need for quoting is obviated since it is macro code. In SAS code, this would be:

```
prxchange( "s/([_a-z][_a-z0-9]{0,31})/$1=/i"
         , -1
         , "&id."
         )
```

Such formatting of one argument per line allows one to more easily see the arguments, but it is also very good for 1) debugging or experimenting since the arguments can be more easily commented out using a block comment (Ctrl-/ and Shift-Ctrl-/) and 2) written by a program (that will be subsequently assigned to a human programmer for fine tuning and approval). The second argument, "-1", instructs the regular expression engine to repeat the pattern substitution indefinitely.

Another viable way to write this is using the "#" delimiter: "s###". When the target or substitution text contains "/", this may be easier to code (and read):

```
1     data _null_ ;
2
3       x = "1/2" ;
4
5       y = prxchange( "s/\// divided by /" , 1 , x ) ;
6       z = prxchange( "s#/# divided by #"  , 1 , x ) ;
7
8       put x / y / z ;
9
10    run ;

1/2
1 divided by 2
1 divided by 2
NOTE: DATA statement used (Total process time):
      real time           0.20 seconds
      cpu time            0.01 seconds
```

Note the need to escape the "/" when it is the delimiter of the substitution or match, i.e., "s///", but not when the delimiter is "#", i.e., "s###".

Finally, on lines 222-226, the PRXCHANGE() is worth explaining. The third argument is also a to this PRXCHANGE() call is also a PRXCHANGE() call. The following code demonstrates:

```
%let id = id1
          id2
          ;

%put %sysfunc( prxchange( s/([_a-z][_a-z0-9]{0,31})/$1=\//i , -1 , &id. )) ;
id1=/           id2=/

%put %sysfunc( prxchange( s/\/$//
                        , 1
                        , %sysfunc( prxchange( s/([_a-z][_a-z0-9]{0,31})/$1=\//i , -1 , &id. ))
                        )
             ) ;
id1=/           id2=
```

The inner PRXCHANGE() appends an "=/" to each variable in the list of variables provided in the ID= macro parameter variable. The outer PRXCHANGE() removes the "/" if it is the last byte in the string, indicated by "$" in the pattern. Effectively, substitution removes the "/" from the last variable in that list and only from that variable.

CONCLUSION

Validation is a central, indispensable task in clinical trial programming that can be a bottleneck in the delivery process. The SAS COMPARE procedure is an essential tool in Validation when the SAS System is used. When values of the ID variables, variables being compared, or both are long, their values in the COMPARE output (.lst) may be truncated, complicating either the identification of the observation (in the case of ID variables) or the nature of the mismatch (in the case of the compared variable) and hampering subsequent investigations into the origins of the mismatch. In fact, despite being in the COMPARE out because the value of compared variable mismatched, the values may appear to match when the mismatch occurs "downstream" of the point of truncation.

This paper contributes to the accuracy and expeditiousness of Validation by presenting two macros to aid the programmers in identification of observations that mismatch on the variable of concern. MAC_R_LONG_COMPARE matches observations from two data sets on ID variables, but mismatch on the variable of interest and displays the full values of those variables rapidly, with minimal coding. The

paper demonstrated how to use the macro when the long values of the variables were truncated in the COMPARE results; such results can either show the pattern or can be used for further investigations. The paper demonstrated mismatches due to varying numbers of leading spaces, often used to achieve desired formatting in the TFL output (PDF/RTF) files and showed how to use the macros to investigate and determine the number of leading spaces without manually counting. In contrast to a method such as

```
Length( X ) - Length( Left( X ))
```

the macro quantitates *and* visualizes the leading spaces. The paper demonstrated how to MAC_R_LONG_COMPARE if non-printable characters were "embedded", but hard to see or identify, allowing the user to see the hexadecimal and original value in a byte-by-byte display.

The macro MAC_R_LEAD_SPACES displays values with leading spaces and the variables to which they pertain from a single data set. Seeing multiple variable/value pairs in one frame with a clearly defined starting point may assist the programmer in inferring a pattern, if any. Optionally, MAC_R_LEAD_SPACES can remove leading spaces so that an informal comparison of values without leading spaces can inform the programmers or guide further investigations if leading spaces are not the cause of mismatches. Usually, once the programmers know that the values without leading spaces match, arriving at the same algorithm to independently insert the same number of leading spaces occurs rapidly.

Finally, the paper contributes to an important discussion of whether values (in VALDS's) should contain modifications to achieve formatting in TFL output or whether a more parsimonious approach would be to use programming logic in the REPORT procedure to achieve the desired formatting. The paper demonstrated indentation or changes to the "left margin", but this could easily be adopted to other attributes or paging, for instance, changing font or background color or font emphasis (bold or italics, for instance). Substantially, some modifications may be distinct to a given programming language, package, or output destination, but "raw" values should be invariant, i.e., programming language-agnostic, for instance.

REFERENCES

[1] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990 Oct 5;215(3):403-10. doi: 10.1016/S0022-2836(05)80360-2. PMID: 2231712.

[2] Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, Bealer K, Madden TL. BLAST+: architecture and applications. BMC Bioinformatics. 2009 Dec 15;10:421. doi: 10.1186/1471-2105-10-421. PMID: 20003500; PMCID: PMC2803857.

[3] Viel, K. 2012. "Using the SAS System as a bioinformatics tool: a macro that calls the standalone Basic Local Alignment Search Tool (BLAST) setup." Proceedings of the PharmaSUG 2012 Conference, San Francisco, CA: PharmaSUG. https://www.pharmasug.org/proceedings/2012/HO/PharmaSUG-2012-HO07.pdf

[4] Viel, K. 2022. "Using the SAS System® with the National Center for Biotechnology Information resources and Immune Epitope Database to explore the realm of potential SARS-CoV-2 variants." Proceedings of the PharmaSUG 2022 Conference, Austin, TX: PharmaSUG. https://www.pharmasug.org/proceedings/2022/AD/PharmaSUG-2022-AD-167.pdf

[5] KRV Holder, PHUSE Connect 2023, Orlando, FL.

[6] SAS Communities. Zender, Cynthia (Cynthia_sas). "Re: ODS PDF/RTF and the REPORT procedure: break on space not word." 01-29-2017 04:14 PM. Available at https://communities.sas.com/t5/ODS-and-Base-Reporting/ODS-PDF-RTF-and-the-REPORT-procedure-break-on-space-not-word/m-p/328300#M17819.

RECOMMENDED READING

SAS® 9– Perl Regular Expressions Tip Sheet

https://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf

CONTACT INFORMATION

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

Kevin R. Viel, Ph.D.

Navitas Data Sciences
kevin.viel@navitaslifesciences.com
www.navitasdatasciences.com

Histonis, Incorporated
kviel@histonis.org

APPENDIX

Appendix 1

```
%macro mac_r_long_compare
   ( data          =
   , comp          =
   , out           = __lc
   , var           =
   , id            =
   , id_rows       = N
   , where         = 1
   , differences   = N
   , transpose     = 256
   , format        = %str()
   , byte_format   = $char1.
   , both_formats  = Y
   , file          = %str()
   , help          = N
   ) ;

  %if &help. = Y
  %then
     %do ;
         %put ____________________________________________________________________________________________________ ;
         %put Purpose of program:        This macro displays values of mismatching variables that cannot be        ;
         %put %str(                           ) displayed fully by the COMPARE procedure output due to truncation. ;
         skip ;
         %put Macro Parameter             Description                                                              ;
         %put ____________________        _________________________________________________________________________ ;
         %put data                    = Name of the base data set.                                                 ;
         %put comp                    = Name of the compare data set.                                              ;
         %put out                     = Name of the output data set.                                               ;
         %put %str(                             )Default: __lc                                                     ;
         %put var                     = Variable to compare.                                                       ;
         %put id                      = ID variables.                                                              ;
         %put id_rows                 = Whether to put one ID variable per row.                                    ;
         %put %str(                             )Default: N                                                        ;
         %put where                   = Where clause.                                                              ;
         %put %str(                             )Default: 1                                                        ;
         %put differences             = Whether to use a bar/hyphen to emphasize differences.                      ;
         %put %str(                             )Default: N                                                        ;
         %put transpose               = The threshold of the length to transpose the values.                       ;
         %put %str(                             )Default: 256                                                      ;
         %put format                  = The format to display the values when not transposed.                      ;
         %put %str(                             )Default: %nrstr(%%)str()                                          ;
         %put byte_format             = The format to display the individual bytes of the values when transposed.  ;
         %put %str(                             )Default: $char1.                                                  ;
         %put both_formats            = When transposed and a HEXw. format is used, whether to display both the    ;
         %put %str(                           )values with a HEXw. format and with a $CHAR1. format.               ;
         %put %str(                             )Default: Y                                                        ;
```

```
48          %put file                       = The path and filename to FILE (write) the results (discrepancies).        ;
49          %put %str(                              )Default: %nrstr(%%)str()                                          ;
50          skip ;
51          %put ____________________________________________________________________________________________________  ;
52          %put End of help                                                                                            ;
53
54          %goto __END ;
55       %end ;
56
57    proc sql ;
58      create table &out. as
59      select a.&var. as v1
60           , b.&var. as v2
61
62             %let i = 1 ;
63             %let d = %scan( &id. , &i. , %str( )) ;
64
65             %do %while ( &d. ne ) ;
66
67                , a.&d.
68
69                %let i = %eval( &i. + 1 ) ;
70                %let d = %scan( &id. , &i. , %str( )) ;
71             %end ;
72      from &data. as a
73         , &comp. as b
74      where  %let i = 1 ;
75             %let d = %scan( &id. , &i. , %str( )) ;
76
77             %do %while ( &d. ne ) ;
78
79                %if &i. > 1 %then and ;
80                a.&d. = b.&d.
81
82                %let i = %eval( &i. + 1 ) ;
83                %let d = %scan( &id. , &i. , %str( )) ;
84             %end ;
85             and a.&var. ne b.&var.
86             and &where.
87      order by %let i = 1 ;
88               %let d = %scan( &id. , &i. , %str( )) ;
89
90              %do %while ( &d. ne ) ;
91
92                 %if &i. > 1 %then , ;
93                 a.&d.
94
95                 %let i = %eval( &i. + 1 ) ;
96                 %let d = %scan( &id. , &i. , %str( )) ;
97              %end ;
98      ;
99    quit ;
100
101   %if &sqlobs. > 0
102   %then
103      %do ;
104
105          proc sql
106             noprint ;
107            select type
108                      into : type
109            from sashelp.vcolumn
110            where     %if %sysfunc( index( &out. , . ))
111                      %then     upcase( libname ) = "%sysfunc( upcase( %sysfunc( scan( &out. , 1 , . ))))"
112                            and upcase( memname ) = "%sysfunc( upcase( %sysfunc( scan( &out. , 2 , . ))))"
113                      ;
114                       %else     upcase( libname ) = "WORK"
115                            and upcase( memname )  = "%sysfunc( upcase( &out. ))"
116                      ;
117                   and upcase( name ) = "V1"
118            ;
119          quit ;
120
121          %if &type. = char
122          %then
123             %do ;
124                 proc sql
125                    noprint ;
126                   select strip( put( max( length( v1 )) max max( length( v2 )) , 8.0 ))
127                             into : length separated by ""
128                   from &out.
129                   ;
130                 quit ;
131
132                 %if &format. = %str()
133                 %then %let format = $char&length.. ;
134             %end ;
135           %else %let length = ;
136
137          data _null_ ;
138
139            %if %nrbquote(&file.) ne %str()
140            %then file "&file." %str(;) ;
141
142            set &out. ;
```

```
143
144             if _n_ > 1 then put / ;
145
146             %if     &differences. = Y
147                 and &type.        = char
148             %then
149                %do ;
150
151                    length diff $ &length.
152                           byte $ 1
153                           ;
154
155                    diff = " " ;
156
157                    l1 = length( v1 ) ;
158                    l2 = length( v2 ) ;
159
160                    min = min( l1 , l2 ) ;
161                    max = max( l1 , l2 ) ;
162
163                    do _n_ = 1 to min ;
164                      if substr( v1 , _n_ , 1 ) = substr( v2 , _n_ , 1 )
165                      then byte = " " ;
166                       else byte = "|" ;
167                      substr( diff , _n_ , 1 ) = byte ;
168                    end ;
169
170                    if l1 ne l2
171                    then substr( diff , min + 1 , max - min ) = repeat( "|" , max - min - 1 ) ;
172
173                %end ;
174
175             %if     &type.        ne char
176                 or  &length.      <= &transpose.
177             %then
178                %do ;
179                    put %if &id_rows. = N %then %sysfunc( prxchange( s/([_a-z][_a-z0-9]{0,31})/$1=/i , -1 , &id. )) ;
180                         %else %sysfunc( prxchange( s/\/$//
181                                                  , 1
182                                                  , %sysfunc( prxchange( s/([_a-z][_a-z0-9]{0,31})/$1=\//i , -1 , &id. ))
183                                                  )
184                                       ) ;
185
186                      / v1 %if &type. = char %then &format. ;
187                      %if     &differences. = Y
188                          and &type.        = char
189                      %then / diff &format. ;
190                      / v2 %if &type. = char %then &format. ;
191                      ;
192                %end ;
193
194              %else
195                 %do ;
196                     length v1_byte
197
198                            %if     &differences. = Y
199                                and &type.        = char
200                            %then diff_byte ;
201
202                            v2_byte $ 1
203                            ;
204
205                     put %sysfunc( prxchange( s/([_a-z][_a-z0-9]{0,31})/$1=/i , -1 , &id. )) ;
206
207                     do _n_ = 1 to &length. ;
208
209                       if _n_ <= length( v1 ) then v1_byte = substr( v1 , _n_ , 1 ) ;
210                        else v1_byte = " " ;
211
212                       %if     &differences. = Y
213                           and &type.        = char
214                       %then
215                          %do ;
216                              diff_byte = substr( diff , _n_ , 1 ) ;
217                              if diff_byte = "|" then diff_byte = "-" ;
218                          %end ;
219
220                       if _n_ <= length( v2 ) then v2_byte = substr( v2 , _n_ , 1 ) ;
221                        else v2_byte = " " ;
222
223                       put _n_
224                           %if     %sysfunc( prxmatch( /^hex/i , &byte_format. ))
225                               and &both_formats. = Y
226                           %then @10 v1_byte $char1. +2 v1_byte &byte_format. ;
227                            %else @10 v1_byte &byte_format. ;
228
229                           %if     %sysfunc( prxmatch( /^\$char1\./i , &byte_format. ))
230                               and &differences. = N
231                           %then +1 ;
232
233                           %if     &differences. = Y
234                               and &type.        = char
235                           %then
236                              %do ;
237                                  +1 diff_byte
```

```
                            %end ;

                          %if     %sysfunc( prxmatch( /^hex/i , &byte_format. ))
                              and &both_formats. = Y
                          %then v2_byte &byte_format. +2 v2_byte $char1. ;
                           %else v2_byte &byte_format. ;
                          ;
                    end ;

                %end ;
          run ;

      %end ; /* END OF sqlobs > 0 */

    %else %put A%str(LERT: ) No qualifying observations. ;

  %__END:

%mend  mac_r_long_compare ;
```

Appendix 2

```
%macro mac_r_lead_spaces
  ( ds     =
  , out    = _null_
  , remove = N
  , list   = Y
  ) ;

  data &out. ;
    set &ds. ;
    array __c( * ) $ _character_ ;
    do _n_ = 1 to dim( __c ) ;

      %if     %sysfunc( prxmatch( /Y|YES/i , &list. )) > 0
      %then
         %do ;
             if __c( _n_ ) ne strip( __c( _n_ ))
             then
               do ;
                  __t = translate( __c( _n_ ) , "_" , " " ) ;
                  __v = vname( __c( _n_ )) ;
                  put @1  __v
                      @35 __t
                      ;
                  %if %sysfunc( prxmatch( /Y|YES/i , &remove.   )) > 0 %then __c( _n_ ) = strip( __c( _n_ )) %str(;) ;
               end ;
         %end ;

       %else %if %sysfunc( prxmatch( /Y|YES/i , &remove. )) > 0
       %then __c( _n_ ) = strip( __c( _n_ )) %str(;) ;

    end ;
  run ;

%mend  mac_r_lead_spaces ;
```

Appendix 3

```
data indent ;

  length row_1
         column_1     $ 200
         ;

  /********/
  row_order_1 = 1 ;
  row_1       = "Plain" ;
  column_1    = " " ;
  do _n_ = 1 to 9 ;
    column_1 = cats( column_1
                   , "----+----"
                   , put( _n_ , 1. )
                   ) ;
  end ;
  output ;

  /********/
  row_order_1 + 1 ;
  row_1    = "Indent:   style = { indent = 4% }" ;
  column_1 = " " ;
  do _n_ = 1 to 9 ;
    column_1 = cats( column_1
                   , "----+----"
                   , put( _n_ , 1. )
                   ) ;
  end ;
  output ;

  /********/
  row_order_1 + 1 ;
  row_1    = "5 Leading spaces in value" ;
                 /* The REPEAT function, though cumbersome, makes the number obvious:
                    note that it creates one more replicate than the (non-negative
                    integer) second argument.
                 */
  column_1 = cat( repeat( " "
                        , 4
                        )
                , "----+----1"
                ) ;
  do _n_ = 2 to 9 ;
    column_1 = cat( trim( column_1 )
                   , "----+----"
                   , put( _n_ , 1. )
                   ) ;
  end ;
  output ;

  /********/
  row_order_1 + 1 ;
  row_1    = "`{style [ textdecoration = underline ]Left Margin}:   `{style [ fontweight = medium color = black ]style =
`{unicode 007B} leftmargin = 4% `{unicode 007D}}" ;
  column_1 = " " ;
  do _n_ = 1 to 9 ;
    column_1 = cats( column_1
                   , "----+----"
                   , put( _n_ , 1. )
                   ) ;
  end ;
  output ;

run ;

%m_r_lead_spaces
  ( ds       = indent ) ;

ods escapechar = "`" ;

/***********************************/
ods pdf
    file = "%sysfunc( getoption( SASUSER ))\%trim( %scan( %sysget( sas_execfilename ) , 1 , . )).pdf"
    ;
ods listing close ;
ods noresults ;

title1 'Only row_1 = "5 Leading spaces in value" affects the value of ROW_1' ;
title2 'Left margin is preferable, if the text might wrap, since the wrapped text is "properly" aligned' ;

proc report
   data = indent ;

  column row_order_1
         row_1
         column_1
         ;

  define row_order_1
       / order
         noprint
         ;
```

```
 93
 94    define row_1
 95         / order
 96           style ( column ) = { cellwidth = 45% }
 97           ;
 98
 99    define column_1
100         / order
101           style ( column ) = { cellwidth = 50%
102                                asis      = on
103                              }
104           ;
105
106    compute row_1 ;
107
108        if row_order_1 = 4
109        then call define ( _col_
110                         , "style"
111                         , "style = { color          = green
112                                      fontweight     = bold
113                                    }"
114                         ) ;
115
116    endcomp ;
117
118    compute column_1 ;
119
120       if row_order_1 = 2
121       then call define ( _col_
122                        , "style"
123                        , "style = { indent = 4% }"
124                        ) ;
125        else if row_order_1 = 4
126        then call define ( _col_
127                         , "style"
128                         , "style = { leftmargin = 4% }"
129                         ) ;
130
131    endcomp ;
132
133
134  run ;
135
136  ods pdf close ;
137  ods listing ;
138  ods results ;
```