Paper PP17

A Standard Data Set Format for the Validation of Tables and Listings in Regulatory Submissions for Clinical Trials

Kevin R. Viel, Ph.D., Navitas Data Science; Histonis, Incorporated, Manchester, USA

ABSTRACT

The community of sponsors, contract research organizations, regulatory agencies, support or advisory organizations, investigators, and academics rightly focus on the structure and content of data sets that result from clinical trials and are included in submissions for regulatory approval. Many regulatory agencies, including the Food and Drug Agency (FDA) of the United States, require that submissions adhere to the implementation guides (IG) created by the Clinical Data Interchange Standards Consortium (CDISC), for example the SDTM-IG or ADaM-IG. Programmers use these standardized data sets to create the clinical study report (CSR) comprising tables, figures, and listings (TFLs). Typically, TFLs are double programmed with 100% independence. Whereas the code to create the TFL will be included in the Analysis Results Metadata (ARM) of the define.xml, the Validation Data Sets (VALDS) used to validate and approve the TFLs are not and are not otherwise part of a submission in contrast to the SDTM and ADaM data sets. Although data sets are typically rows and columns, considering them as a matrix, with cells indexed by page, section, row, and column (PSRC), standardizes them and may automate their creation (programming) based on the shells typically provided in a Statistical Analysis Plan (SAP). Including a variable for the type of observation (header, footnote, body) enables the inclusion of column headers and page footnotes, data that are often derived, but rarely programmatically evaluated. The **goal** of this paper is to describe a standardized structure and content of the VALDS that are used the create of tables and listings and enable their Validation, thus aligning with the spirit of an interchange standard, increasing accuracy and efficiency.

INTRODUCTION

Consider some of the issues that a programmer who is tasked with validating a TFL program and its output (PDF/RTF) may face: what observations (rows) and variables (fields) will the VALDS contain, what are the orders of the variables and observations, what are the names of the variables, and what are the types (character or numeric) of the variable. With enough experience in TFL Validation, one may encounter programs that name the variables for columns as letters (“A”, “B”, “C”,…) or as a variation of “COLUMN” and an index (number): “COLUMN1”, “COL1”, or “COL_1”. Usually, the collation order corresponds to the position of the column in a shell, but some of us will see that correspondence violated when the biostatistician adds a new column and the Main programmer just assigned it to the next available column name instead of reorganizing the new version to maintain correspondence. Such issues decrease the accuracy, independence, and speed of Validation.

Standardization improves data exchange and the use of data. The implementation guides[1,2] published by CDISC include details on the names of variables or the naming system for them, the type (character or numeric), the length of the names of the variables and their values, the labels of the variables, what variables are valid to include in a given data set, the order of the variables, among other issues. When a programmer or biostatistician is new to a project, such standardization allow her or him to begin work immediately without studying the data dictionary of the project or learning the conventions of the team. Given that the number of TFLs on a Phase 3 trial may surpass 500, standardization of the Validation Data Sets is appropriate and will have meaningful returns. The **goal** of this paper is to describe a standardized structure and content of the VALDS that are used to create of tables and listings and enable their Validation, to discuss advantages of using a standardized VALDS, and to provide some examples.

VALDS STRUCTURE AND VARIABLES

Table 1 presents a typical shell for a demographics table for the Safety Population. “Table 14.1.X Demographics” might be a usual title for such a table. The shell provides the orders of the pages (Safety Population), the sections, the rows, and the columns. The programmer solely derives the values represented with X’s and decimal points, possibly indicating precision.

Table 1. Demographics
Safety Population

Characteristic	Arm 1 (N=XXX)	Arm 2 (N=XXX)	Total (N=XXX)
Sex, n (%)			
Female	xx (xx.x)	xx (xx.x)	xx (xx.x)
Male	xx (xx.x)	xx (xx.x)	xx (xx.x)
Missing	xx (xx.x)	xx (xx.x)	xx (xx.x)
Age (years)			
n	xx	xx	xx
Mean (SD)	xx.x (xx.xx)	xx.x (xx.xx)	xx.x (xx.xx)
Median	xx.x	xx.x	xx.x
Min, Max	xx, xx	xx, xx	xx, xx
Missing (%)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Age Group, n (%)			
< 65 years	xx (xx.x)	xx (xx.x)	xx (xx.x)
>= 65 years	xx (xx.x)	xx (xx.x)	xx (xx.x)
Missing	xx (xx.x)	xx (xx.x)	xx (xx.x)
Race, n (%)			
White	xx (xx.x)	xx (xx.x)	xx (xx.x)
Black or African American	xx (xx.x)	xx (xx.x)	xx (xx.x)
Asian	xx (xx.x)	xx (xx.x)	xx (xx.x)
American Indian or Alaska Native	xx (xx.x)	xx (xx.x)	xx (xx.x)
Native Hawaiian or Other Pacific Islander	xx (xx.x)	xx (xx.x)	xx (xx.x)
Other	xx (xx.x)	xx (xx.x)	xx (xx.x)
Missing	xx (xx.x)	xx (xx.x)	xx (xx.x)

Table 2 presents the proposed VALDS variables, including their types, controlled terminology, whether they are required (Req), conditionally required (Cond), or permissible (Perm), and notes. Following section 3.1.1(c) of CDISC ADaM Implementation Guide Version 1.1[2], the "y" indicates an integer from 1-99 without zero padding. The four dimension variables are Page, Section, Row, and Column and are listed by level of hierarchy. Column is nested in Row, Row is nested in Section, and Section is nested in Page. Multiple variables may exist for a given dimension, for instance, one may see PAGE_1 and PAGE_2 with PAGE_2 devised to address paging, for example, to limit the number of rows per physical page in an attempt to avoid splitting sections across two pages. The only numeric variables are the ORDER variables (ROW_ORDER_y for example. The variables PAGE_Y, SECTION_Y, and ROW_Y are labels and COL_Y pertains to the derived values. Selecting a default length of 200, the CDISC maximum, seems reasonable. This length will, obviously, be inefficient with regard to computational resources, but VALDS are typically small and not submitted to regulatory agencies, so a default length eliminates the need to determine length to achieve a match between the Main and Validation data sets. Finally, TYPE indicates whether the observation pertains to a header, appears in the body, or is a footnote.

The ORDER variables are the primary keys of the data sets. They should be used to sort the observations of the data set: PAGE_ORDER_1… PAGE_ORDER_n, SECTION_ORDER_1… SECTION_ORDER_n, ROW_ORDER_1… ROW_ORDER_n. The variables should appear in the data set in the order of this table. No labels exist for these variables. The purpose is to facilitate a manual "spot checking" of the output (PDF/RTF) against the data sets. Both programmers may have included a given observation, but that does not mean that it also appears in the output or that the value displayed is correct (for instance, truncation might have occurred). Having the data set "appear like" the output facilitates such reviews.

Table 2. VALDS Variables

Variable Name	Type	Codelist/Controlled Terms	Core	Notes
PAGE_y	Char		Perm	Higher level PAGE variables are nested in the lower-level PAGE variables.
PAGE_ORDER_y	Num	Integer	Cond	Key variable(s) for the dataset.
SECTION_y	Char		Perm	Nested in the PAGE dimension. Higher level SECTION variables are nested in the lower-level SECTION variables.
SECTION _ORDER_y	Num	Integer	Cond	Key variable(s) for the dataset.
ROW_y	Char		Req	Nested in the SECTION dimension. Higher level ROW variables are nested in the lower-level ROW variables.
ROW_ORDER_y	Num	Integer	Req	Key variable(s) for the dataset.
COL_y	Char		Req	Nested in the ROW dimension.
TYPE	Char	B,H,F	Req	Indicator of whether the observation pertains to the Body, Header, or Footnote

Table 3 presents the example shell annotated using the variables in Table 2. Often, one TFL program creates one output, even if related, such as Demographics tables for the Safety and Intention-to-Treatment (ITT) Populations. Besides allowing an audit of programs and output prior to delivery, for instance, to make sure every program has output and that the output file has a datetime stamp latter than the program and input data sets, having a one-to-one requirement for programs and output also reduces the Validation and review requirements if only one of the output were affected by an update, such as an update to a title, but not the input data set.

Table 3. The example shell annotated using the standardized VALDS

Table 14.1.X Demographics Safety Population (Page_1)							
Section_1	Section_Order_1	Row_1	Row_order_1	Col_1	Col_2	Col_3	Type
		Characteristic		Arm 1 (N=XXX)	Arm 2 (N=XXX)	Total (N=XXX)	H
Sex, n (%)	1	Sex, n (%)	1				B
Sex, n (%)	1	Female	2	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Sex, n (%)	1	Male	3	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Sex, n (%)	1	Missing	4	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
							B
Age (years)	2	Age (years)	1				B
Age (years)	2	n	2	xx	xx	xx	B
Age (years)	2	Mean (SD)	3	xx.x (xx.xx)	xx.x (xx.xx)	xx.x (xx.xx)	B
Age (years)	2	Median	4	xx.x	xx.x	xx.x	B
Age (years)	2	Min, Max	5	xx, xx	xx, xx	xx, xx	B
Age (years)	2	Missing (%)	6	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
							B
Age Group, n (%)	3	Age Group, n (%)	1				B
Age Group, n (%)	3	< 65 years	2	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Age Group, n (%)	3	>= 65 years	3	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Age Group, n (%)	3	Missing	4	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
							B
Race, n (%)	4	Race, n (%)	1				B
Race, n (%)	4	White	2	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Race, n (%)	4	Black or African American	3	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Race, n (%)	4	Asian	4	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Race, n (%)	4	American Indian or Alaska Native	5	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Race, n (%)	4	Native Hawaiian or Other Pacific Islander	6	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Race, n (%)	4	Other	7	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Race, n (%)	4	Missing	8	xx (xx.x)	xx (xx.x)	xx (xx.x)	B

While choice of the names of columns is set by their order in the shell, the structure of the VALDS is flexible with regard to PAGE and SECTION. For instance, a programmer could just use a ROW_1 variable for the shell in Table 1. This does reduce the complexity of the data set, but the Validation programmer would need to be informed

of the structure. Further, addition utility is gained including section, such as the ability to use programming logic to indent a ROW_1 (row label) without the need to include leading spaces in it (discussed below).

In this case, the variable SECTION_1 is not displayed in the output. Note that it is populated for each observation of a section in the VALDS. Including SECTION_1 and SECTION_ORDER_1 has distinct programming advantages. Including PAGE_1 and PAGE_ORDER_1 creates uniformity between TFL programs and between the same program in different projects and between updates or deliveries. Note that a change to the shell or an update to a data set could make PAGE and SECTION required. Parsimony increases the familiarity of structure and contents of the VALDS, while the added complexity and size is negligible.

Table 4 present a schematic of a data set, for instance, a data set in the SAS System®. Actual numbers were not substituted in this example, but in a real data set, they would be. Note that for TYPE = "H" (header), SECTION_ORDER_1 and ROW_ORDER_1 are not populated. In this shell, the Section Label, for instance, "Sex, n(%)", is incorporated in ROW_1, so for TYPE = "H", SECTION_1 is not populated. Importantly, the column headers include the "Big N", i.e., the "N=XXX", the column total (denominator). This derived value is often omitted from the VALDS even though it is subject to independent Validation. Further, this value is often audited across multiple tables (and updates). Having it in the VALDS facilitate such an audit because it can be performed programmatically. Finally, that the VALDS does not address appearance issues like blank lines between sections or indentation of certain row labels (discussed below).

Table 4. An example of a data set that follows the VALDS standards, for instance a SAS data set.

Page_1	Page _Order_1	Section_1	Section_Order_1	Row_1	Row_order_1	Col_1	Col_2	Col_3	Type
Safety Population	1			Characteristic		Arm 1 (N=XXX)	Arm 2 (N=XXX)	Total (N=XXX)	H
Safety Population	1	Sex, n (%)	1	Sex, n (%)	1				B
Safety Population	1	Sex, n (%)	1	Female	2	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Sex, n (%)	1	Male	3	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Sex, n (%)	1	Missing	4	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Age (years)	2	Age (years)	1				B
Safety Population	1	Age (years)	2	n	2	xx	xx	xx	B
Safety Population	1	Age (years)	2	Mean (SD)	3	xx.x (xx.xx)	xx.x (xx.xx)	xx.x (xx.xx)	B
Safety Population	1	Age (years)	2	Median	4	xx.x	xx.x	xx.x	B
Safety Population	1	Age (years)	2	Min, Max	5	xx, xx	xx, xx	xx, xx	B
Safety Population	1	Age (years)	2	Missing (%)	6	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Age Group, n (%)	3	Age Group, n (%)	1				B
Safety Population	1	Age Group, n (%)	3	< 65 years	2	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Age Group, n (%)	3	>= 65 years	3	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Age Group, n (%)	3	Missing	4	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Race, n (%)	4	Race, n (%)	1				B
Safety Population	1	Race, n (%)	4	White	2	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Race, n (%)	4	Black or African American	3	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Race, n (%)	4	Asian	4	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Race, n (%)	4	American Indian or Alaska Native	5	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Race, n (%)	4	Native Hawaiian or Other Pacific Islander	6	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Race, n (%)	4	Other	7	xx (xx.x)	xx (xx.x)	xx (xx.x)	B
Safety Population	1	Race, n (%)	4	Missing	8	xx (xx.x)	xx (xx.x)	xx (xx.x)	B

Note that this example does not have an observation with TYPE = "F". The purpose of such an observation is to make a physical page-specific footnote. For example, a list of subjects who did not have a lab measurement for a given analyte at a given visit is usually included as a footnote to explain why the "little n", that is the value for the row "n" in the section "Age (years)" of Table 4 is less than the Big N. This list is a derived value that should be subject to 100% independent validation. Obviously, if the list is a value of a variable in VALDS, then it would automatically be included in the comparison performed by the SAS COMPARE procedure, for instance. Further, changing the (superscript) symbol in the value to meet FDA specifications for using multiple symbols reduces to a programmatic issue and updates automatically with changing data and shell formats. It is also available for a programmatic audit of the TLFs, by virtue of being a value in a data set.

VALDS AS A MATRIX

The VALDS can be considered as a matrix with a minimum of four dimensions: PAGE, SECTION, ROW, and COLUMN. Each "cell" is indexed by the PAGE_ORDER_1, SECTION_ORDER_1, ROW_ORDER_1, and column number. The programmer derives the value for a given cell, then uses the labels to obtain the ORDER variable. This approach reduces the changes between tables and listing programs since each data set is a matrix. Contrast this with a usual approach:

```
if arm = "Group 1", then col_1 = "XXX"
if arm = "Group 2", then col_2 = "XXX"
if arm = "Group 3", then col_3 = "XXX"
if arm = "Group 4", then col_4 = "XXX"
if arm = "TOTAL",   then col_5 = "XXX"
```

At times, the biostatistician may request that the program add two new columns that groups “Group 1” and “Group 2” and “Group 3” and “Group 4”. Following the logic above, this would result in additional programming:

```
if arm = "Group 1",   then col_1 = "XXX"
if arm = "Group 2",   then col_2 = "XXX"
if arm = "Group 1/2", then col_3 = "XXX"
if arm = "Group 3",   then col_4 = "XXX"
if arm = "Group 4",   then col_5 = "XXX"
if arm = "Group 3/4", then col_6 = "XXX"
if arm = "Group 4",   then col_7 = "XXX"
```

If instead, the label was linked to the column number, this simplifies the program. Consider the following SAS code:

```
data __columns ;
   length column_label $ 200 ;
   column_label = "Group 1"   ;
   column + 1 ;
   output ;
   column_label = "Group 2"   ;
   column + 1 ;
   output ;
   column_label = "Group 1/2" ;
   column + 1 ;
   output ;
   column_label = "Group 3"   ;
   column + 1 ;
   output ;
   column_label = "Group 4"   ;
   column + 1 ;
   output ;
   column_label = "Group 3/4" ;
   column + 1 ;
   output ;
   column_label = "Group 4"   ;
   column + 1 ;
   output ;
run ;
```

Assuming that one linked the value of ARM to COLUMN_LABEL to obtain COLUMN, for instance, using a HASH object (associative array) in SAS (or other language), then the “IF-THEN” statements above reduce to the following in SAS:

```
array col_( &num_of_cols. )
      $ 200
      ;

col_( column ) = value ;
```

The code does NOT need to be structured around the data or shell. Whether we have 20 or three columns, the code does not change. If the Biostatistician adds or remove rows, the code does not change. The only thing that changes is the data set linking column labels to column numbers. The same is true for pages, sections, and rows. This is conducive to “turning the data” and using one procedure to derive that values of the cell. Consider the following SAS code:

```
proc format ;
   value $ sect_fmt
      "sex"     = "Sex, n(%)"
      "race"    = "Race, n(%)"
      "agrgr1"  = "Age Group, n(%)"
      ;
run ;


%let dimension = section_1
                 row_1
                 column_label
                 ;

%let dimension_number = %eval( %sysfunc( countc( %nrbquote(%sysfunc( compbl( &dimensions.
))) , %str( ))) + 1 ) ;

%let variables = sex
                 agegr1
                 race
                 ;

/* "Turn the data" */
data ads_cat
        ( keep   = &dimensions.
                   value
        )
     ;

  length &dimensions.
         value        $ 200
         ;

  set &in_ds.
         ( keep   = usubjid
                    &variables.
                    saffl
           where  = ( saffl = "Y" )
         )
      ;

  array __v ( * )
        &variables.
        ;

  do _n_ = 1 to dim( __v ) ;
    row_1     = __v( _n_ ) ;
    section_1 = put( lowcase( vname( __v( _n_ ))) , $sect_fmt. ) ;
    value     = "Y" ;
    output ;
  end ;

run ;

data ads_cat ;
  set ads_cat ;
  output ;
  column_label = "Total" ;
  output ;
run ;

ods listing close ;

%let freq_type = %sysfunc( repeat( 1 , &dimension_number.)) ;
```

```
/* Obtain the frequencies for the turned data */
proc freq
   data = ads_cat ;

  ods output CrossTabFreqs = ctf
                                ( keep   = &dimensions.
                                           value
                                           _type_
                                           frequency
                                  where  = (     _type_ = "&freq_type."
                                             and value  = "Y"
                                           )
                                )
            ;

  tables %sysfunc( prxchange( s/\s+/%str(*)/
                            , -1
                            , &dimensions.
                            )
                 )
       * value
       / nocol
         norow
         nopercent
         ;
 run ;

ods output close ;
ods listing ;
```

This code truly exemplifies "analysis-ready" in that after turning the data, only one total SAS FREQ procedure was called for the entire table. Using the labels to obtain the index for a given dimension simplifies the code while making it more flexible. Often, one sees a call to a macro for each variable to obtain a frequency listing. Note that the same technique can be used for numeric (quantitative) data for which summary statistics like mean, standard deviation, median, minimum, and maximum are required (in which case the MEANS procedure is appropriate).

MINIMAL VALUES VERSUS MODIFICATION FOR FORMATTING

The most common type of modification of a value to achieve formatting in the output is probably leading spaces. Leading spaces can be a scourge to Validation for several reasons. The author covers techniques for discovering them, discerning their numbers, and the potential implications for Validation in another paper[3]. Other types of modification include destination-specific code, such as RTF, or language-specific items, like the SAS inline style below. As the values of ROW_1 in Table 4 suggest, the author does not favor the approach of modification of values in most cases.

The SAS REPORT procedure is a computational procedure, and the COMPUTE BLOCK can be used to conditionally implement a LEFTMARGIN. The following SAS code demonstrates:

```
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 ;

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
         ;

  define row_1
       / order
         style ( column ) = { cellwidth = 45% }
         ;

  define column_1
       / order
         style ( column ) = { cellwidth = 50%
                              asis      = on
                            }
         ;

  compute row_1 ;

      if row_order_1 = 4
      then call define ( _col_
                       , "style"
                       , "style = { color          = green
                                    fontweight     = bold
                                  }"
                       ) ;

  endcomp ;

  compute column_1 ;

     if row_order_1 = 2
     then call define ( _col_
                      , "style"
                      , "style = { indent = 4% }"
                      ) ;
      else if row_order_1 = 4
      then call define ( _col_
                       , "style"
                       , "style = { leftmargin = 4% }"
                       ) ;

  endcomp ;


run ;

ods pdf close ;
ods listing ;
ods results ;
```

Figure 1 presents the output from code above. Clearly, using programming logic is superior than modifying the value since either leading spaces fail if the string wraps and the indentation is a set amount instead of a relative amount, which matters if the widths of columns change. Note that Cynthia Zender[4] proposed the use of the SAS style attribute LEFTMARGIN for this purpose.

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

Figure 1. The effects of attempts to indent using the SAS REPORT procedure or leading spaces.

CONCLUSION

This paper proposes a standardized Validation Data Set (VALDS) for TFL programs that decreases the reliance on conventions, which should increase the speed and accuracy of the 100% independent Validation process. Knowing the structure, including sort and variable order, and the variable attributes will increase the ability of programmers to adapt a program from update to update and from project to project. In fact, with a standardized VALDS, programmers can write SAS macros that will further reduce the need for code changes between project or for updates. Finally, this paper discusses modifications of values to achieve formatting of the output with the recommendation to minimize any such modifications, including leading spaces. The result will be a data set that is more agnostic to the output destination (PDF, RTF, or HTML) and programming language.

REFERENCES

[1] CDISC. 2023. "ADaM." Accessed February 20, 2023. https://www.cdisc.org/standards/foundational/adam.

[2] CDISC. 2023. "SDTM." Accessed February 20, 2023. https://www.cdisc.org/standards/foundational/sdtm.

[3] Viel, K. 2023. " A SAS System® Macro to Quickly Display Discrepant Values that are too Long for the COMPARE Procedure Output." Proceedings of the PharmaSUG 2012 Conference, San Francisco, CA: PharmaSUG. In Press.

[4] 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.

CONTACT INFORMATION

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

Kevin R. Viel, Ph.D.
Navitas Data Sciences
Histonis, Incorporated
Manchester / 01304
Email: kevin.viel@navitaslifesciences.com
kviel@histonis.org
Web: www.navitasdatasciences.com