Paper SM04

A standard data set format in regulatory submissions for clinical trials: automatic creation of headers, headers of headers, and footnotes.

Kevin R. Viel, Navitas Data Sciences and Histonis, Incorporated, Manchester, NH, USA

ABSTRACT

Accuracy, efficiency, reproducibility, and traceability are major goals of the programming of tables and listings, but automation and reusability have emerged as important targets. Within this framework, standardization is a foundation to achieve these goals. Busa and LeRoy introduced the Analysis Results Standards model to drive automation, especially with re-arrangement of a table or the reproduction of it with various languages, including open-source tools. Viel described a standardized structure and content of a data set to aid in Validation and automatic creation of a table. The **goal** of this paper is to expand on the latter, emphasizing headers and footnotes and the ease with arranging them, and to introduce a SAS® System macro that automates their creation. Importantly, headers and footnotes contain derived data that are subject to Validation, but are often not included in the programmatic approaches, i.e., the COMPARE procedure in SAS, because the data set omits them.

INTRODUCTION

Tables, figures, and listings (TFL) predominate the tasks of statistical programmers in clinical trials. While SDTM domains and ADaM data sets are supremely important, the number and type of TFL programs far exceeds the number of programs that might create their input data sets. For tables and listings, whether the output is similar or divergent does not change the basic structure, exemplified by a SAS System® dataset: observations (rows) with variables (columns). Although, the SAS .sas7bdat and .xpt files are 2 dimensional (RxC), we can conceptualize this to include page and section dimensions (PxSxRxC), with the right most variables nested in the dimension to their left. This does not preclude multiple variables at a given dimension: PAGE_2 nested in PAGE_1. Viel[1] describes a standard data set to be used as the input data set to create output and for Validation, the **VALDS** (**VAL**idation **D**ata **S**et).

Busa, Marshall, and LeRoy[2] detailed the **Analysis Result Standards (ARS)** to enable the automation of producing tables and listings in a programming language agnostic manner to improve the navigation of and reusability of results and analysis. Emphasizing the potential for automation, Busa and Dedhia[3] introduce a TFL Designer that would aid in the creation of shells and, thus, ARS.

TFL shells are designed to guide programmers, that is, they are the *blueprints* to follow. A programmer does not study the shell and decide to add a page, section, row, or column without contacting the author and/or approver of the shells, usually a biostatistician who conferred internally with medical and safety scientists and regulatory teams and externally with the regulatory agencies. In doing so, the programmer is usually attempting to address pragmatic issues like fitting the number of columns on a single (physical page). The shells provide a framework to present the data derived by the statistical programmers. The table and listings shells can be considered a matrix presented in the two dimensions of a physical page, but indexed by page, section, row, and column dimensions. The programmer derives the datum (or the collations of derived data, for example Mean and Standard Deviation) and “pokes” or “pigeon holes”[4] the value into the table or listing using dimensions: page 1, section 2, row 5, column 7. Additional variables may be added to the VALDS, such as, in the jargon of Viel[1], PAGE_2. In this case, PAGE_2 is nested in PAGE_1 and PAGE_2 might be an enumeration to control the number of physical rows on a physical page to stop sections, for example, for breaking across the physical pages. In less frequent occasions, PAGE_1 might be various populations (SAFFL, ITTFL, FASFL) and PAGE_2 might be AVISIT or PARAM, but while this illustrates the nature of nesting within a dimension, we might usually see a separate program for each population. The closer the programs are to a one-to-one correspondence with their output, the easier it is to track progress and to correct issues pertaining to one specific sub-dimension without re-Validating/re-checking numerous outputs.

Such structures are not purely academic or for job security but enable automation of the creation of output using SAS (and quickly and easily expandable), while allowing manual (spot) checking of the output against the VALDS, and very important job in both the creation of output by the Main programmer and its Validation. Converting between a VALDS and the ARS seems possible, but creating the ARS while deriving the VALDS might be a better approach, either way, that task is beyond the scope of this paper. The shell drives the creation of the VALDS and the ARS since it contains inherent information on both the desired derivations and the format of the output, including the order of the pages, sections, rows, and columns, but it also the margins,

row heights, columns widths, font families, font sizes, et cetera that are required to create the output, the code of which may be part of the DEFINE.XML as Analysis Results Metadata. The **goals** of this paper include the introduction, demonstration, and brief code description of a SAS macro, for which the REPORT procedure is fundamental, and some discussion of the implications of a shell within the framework of a data sets that is indispensable to (manual) Validation (spot checking) of the output against it. Importantly, this macro allows the creation of dynamic and *auditable* features to aid in Validation or readability with minimal or no changes to the macro call by the user, enabling fewer changes in programs between updates or projects.

SAS USAGE NOTE 24542

Acknowledging that the REPORT procedure is a computational procedure that can greatly alter (process) the input data set, typically the VALDS[1] in this paper, is paramount to creating the output of interest as desired. Despite this power, the VALDS should *look as much like the output* as possible, apart from certain structures like blank lines between rows of interest, such as before the first row of a section, which the REPORT procedure creates with no more than moderate programming. One major reason is that the VALDS is used to manually (spot) check the output. If the programmer prefers to achieve the VALDS of interest using the REPORT procedure, then one way to achieve this is to use a first REPORT procedure with an *OUT*= option of the **REPORT** statement, then to use a second REPORT procedure to display those results. Often, a table consists of data from multiple different SAS procedures, such as one that displays statistics from both the LIFETEST and the PHREG procedures, so using the computational capabilities of the REPORT procedure is naturally restricted. Occasionally, the output (statistics) from the same SAS procedure with different conditions might also be needed, such as different TYPE= (covariance-structure) options to the RANDOM statement of the MIXED procedure. One does see, usually as a macro, numerous calls to the FREQ or MEANS procedure, but that can typically be achieved by "turning" the data set and running just one FREQ and/or one MEANS procedure per program. In clinical trial programming, the REPORT procedure is often used *mainly* to display the data and offers excellent control for formatting it.

Even if the sorted order of the intended output is the same as the VALDS, one might still use the *ORDER* and *ORDED=DATA* options to a **DEFINE** statement. The results can be frustratingly surprising and may not be obvious for several physical pages into the output, that is, if one does not select these observations in the VALDS to spot check in the output, then a Validation programmer, biostatistician, or, worse, reviewer may find them. SAS addresses this exact order issue in Usage Note 24542[5] and the following input data set and output illustrates the issue within the framework of the VALDS that is developing to automate the creation of tables and listings. Consider reviewing output that is "in order", then you encounter a section in which the order differs. You might wonder if it is intentional (misdirecting or fraud) or if the "error" represents a loss of information or mislabeling but, either way, you increase your vigilance, causing the review to be more prolonged and, potentially, sent to additional reviewers. Pragmatically, we programmers should not allow this to occur and, because the program and output represents our work and name, and perhaps that of our sponsor, we cannot allow it to remain. We learn; after years of using the REPORT procedure, I only read this usage note a month prior to writing this paper.

Consider a shell for a table showing the frequency of adverse events (AEs) listed by visit (section) and ordered by overall descending frequency (row order within a section). **Table 1A** shows the data with the original approach of using the overall descending frequency in all subjects (col_3) displayed in Table 1B:

Table 1: AE data by Visit ordered by descending maximum frequency in all subjects.

A

section_1	row_1	col_1	col_2	col_3	section_order_1	r_o_1		row_order_1
Visit 2	AE B	5 (50)	4 (40)	9 (90)	1	1		1
Visit 2	AE C	3 (30)	3 (30)	6 (60)	1	3	≠	2
Visit 10	AE B	2 (20)	1 (10)	3 (30)	2	1		1
Visit 10	AE A	1 (10)	6 (60)	7 (70)	2	2		2
Visit 20	AE B	1 (10)	1 (10)	2 (20)	3	1		1
Visit 20	AE A	0	1 (10)	1 (10)	3	2		2
Visit 20	AE C	2 (20)	1 (10)	3 (30)	3	3		3
Visit 20	AE D	2 (20)	2 (20)	4 (40)	3	4		4

B

row_1	col_3	r_o_1
AE B	9 (90)	1
AE A	7 (70)	2
AE C	6 (60)	3
AE D	4 (40)	4

Table 1A shows the sort order of the input data set. That order is SECTION_ORDER_1, R_O_1 (as determined in Table 1B), and ROW_1. Including the last variable will achieve a *consistent sort order* between the Main and Validation VALDS's and its exact nature should be dictated by the shell, but here it is not consequential because we have no ties in Table 1B, but that could change with data updates and should be included so that programs do not need to be unnecessarily updated between deliveries or projects. Using the variable ROW_1, that order is alphabetical by preferred term, but the shell could require it to be by a given order of SOC (System Organ Class) then alphabetically by preferred term within SOC, which is not shown and not part of this output. Those details are not required here to demonstrate the concept in the usage note (they are important otherwise). Following the initial design of the VALDS[1], the PAGE_1 and PAGE_ORDER_1 variables were omitted for brevity. In fact, R_O_1 and ROW_ORDER_1 differ only in the second observation. This is a consequence of the contrived data and that

ROW_ORDER_1 is an enumeration within SECTION_1 (VISIT) of the observations sorted R_O_1. AE C is the third most frequent AE in the entire data (Table 1B), but the subjects only experienced AE B and AE C during VISIT 2. I emphasize that I created an example that is intended to demonstrate the issue in Usage Note 24542 and not be guidance for protocols, statistical analysis plans (SAPs), or shells. The guiding principles of those documents are the best statistical tools to which the regulatory agency agreed. Understanding this concept using the *ORDER=DATA* option to the **DEFINE** statement of the REPORT procedure in SAS will help create an accurate table or listing**. Table 2A** demonstrates the order of the output using R_O_1 (the original row order variables while **Table 2B** demonstrates using the new ROW_ORDER_1 variable. The code in panel Table 2C generates the code in Table 2A and 2B with the respective row order variable.

The *working* explanation that follows for this is not certain, enough tests have not been run to assign any confidence to the accuracy of the explanation. The Usage Note reveals two gems: 1) the REPORT procedure establishes the *order pattern* across *all observations* and 2) the established order pattern does not change for each group. In the first approach in Table 2A, the value 3 of R_O_1 (the ordered occurrence of AE C, see Table 1B) appears "before" the value 2 of R_O_1 (the ordered occurrence of AE A, see Table 1B). Even though 2 is less than 3, 2 is not encountered until VISIT 10 (the second "group"), whereas 3 occurs in VISIT 2 (the first "group"). Using the alternative ROW_ORDER_1, the value 2 corresponds to AE C in VISIT 2, but to AE A if VISIT 20, thus the "global" order that REPORT preserves across all "groups" created the *intended order* of rows (AE A before AE C in VISIT 20) in the output.

As with many things, multiple approaches can achieve this "correction". Parsimony might guide the selection of variables to use (or overwrite, as ROW_ORDER_1 replaced R_O_1 since R_O_1, despite having inherent information, is not a standard variable of the VALDS and was only retained to demonstrate the Usage Note), but team/enterprise conventions and other issues like creating PAGE_ORDER/PAGE_ORDER_2 variable to impose a maximum number of rows on a physical page may dictate the choices. Obviously, the algorithm should be as accurate, efficient, and simple as possible so that two more programmers independently using it are not confused and do not require clarification or adjudications. A final consideration about whether to drop or overwrite a variable with inherent information, but one that is not used in the output, including to create it is that that variable *may assist complicated Validation*. That is, some "transparency" is retained that might help investigations into discrepancies. One could create and include ROW_ORDER_2, which *is* a VALDS variable, but less verbose and more efficient ways to create transparency and/or methods of auditing and reconciliation might be found and could be separated from a delivery or submission. The VALDS are small and are usually not included in submission or deliveries, so that programmers may have latitude, depending on the SOPs or Working Instructions (WIs).

Table 2. The REPORT output when only the R_O_1 and ROW_ORDER_1 variables are used as the rightmost ORDER variables.

A

Visit	PT	Group 1	Group 2	Total
Visit 2	AE B	5 (50)	4 (40)	9 (90)
	AE C	3 (30)	3 (30)	6 (60)
Visit 10	AE B	2 (20)	1 (10)	3 (30)
	AE A	1 (10)	6 (60)	7 (70)
Visit 20	AE B	1 (10)	1 (10)	2 (20)
	AE C	2 (20)	1 (10)	3 (30)
	AE A	0	1 (10)	1 (10)
	AE D	2 (20)	2 (20)	4 (40)

B

Visit	PT	Group 1	Group 2	Total
Visit 2	AE B	5 (50)	4 (40)	9 (90)
	AE C	3 (30)	3 (30)	6 (60)
Visit 10	AE B	2 (20)	1 (10)	3 (30)
	AE A	1 (10)	6 (60)	7 (70)
Visit 20	AE B	1 (10)	1 (10)	2 (20)
	AE A	0	1 (10)	1 (10)
	AE C	2 (20)	1 (10)	3 (30)
	AE D	2 (20)	2 (20)	4 (40)

C

```
proc report

  data = usage note 24542
  style( report ) = { width = 45% }
  ;
 column section order 1 section 1   r o 1   /* row order 1 */   row_1 col_1 - col_3 ;
 define section order 1 / noprint order order = data ;
 define section 1       / "Visit" order order = data ;
 define r o 1           / noprint order order = data ;
 /* define row order 1           / noprint order order = data ; */
 define row 1           / "PT" ;
 define col 1 / "Group 1" ;
 define col 2 / "Group 2" ;
 define col_3 / "Total" ;

 compute before section order 1
       / style = { height = 5pt }
       ;
    line " " ;
 endcomp ;

run ;
```

MAC_R_REPORT

As originally designed, the SAS Macro MAC_R_REPORT (see **Appendix 1**) was intended to use a set of the standardized variables of the VALDS and a data set of DEFINE statement options to create REPORT procedure code. Obviously, we have no short cuts in the SAS code that this macro ultimately generates, down to style elements and column names, but the intention was to create flexibility with a minimal amount of code initially required by the programmer and that might require minimal code changes to update the program. That includes rearrangements and additions (such as sections or columns) of an updated shell. When this goal is accomplished, table and listing programs become more standardized, which means fewer resources and time are required to update or adapt the programs for corrections, new deliveries, or new shells. For instance, changing the order of sections of a table is just a matter of changing the numbers in the relevant data sets that are associated with the section labels, for example, SECTION_1, and not the code of the program (in contrast to, for instance, IF-THEN-ELSE statements to create the VALDS) and no updates to the macro call (parameter values) or REPORT procedure statements or options by the programmer (the macro performs this). By request or users then managers in response, criteria on the structure of the input data set have been relaxed to incorporate, for instance, variables named COL1-COLn, some of which may or may not have associated COLn_ORDER variables, to adhere to team conventions or SOPs/WIs. Note that one could use the macro with the VALDS and change the names of the variables after the output is produced, but the macro call would need to be updated to re-create the output, unless the variable names were renamed to their original names.

A plateau in this developing automation is ***programs writing programs*** with input from the shells (the blueprints). While it is possible, the intention is not to create code (a program) of which a subsequent program assumes ownership and fine tunes or executes as is. The shells in the vast majority of cases provides a structure (rows, columns, widths, heights, margins, font families, font sizes, et cetera) that one can represent as a matrix indexed by page, section, row, and column dimensions (for instance, page 2, section 2, row 7, column 8)[1]. The programmer derives a datum and "pigeon holes"[4] it into its cell. One exception to this rule is when the rows have to ordered by (descending frequency), which is still accomplished by dynamic logic and not by hard-coding. One might hard code the order of the System Organ Class (SOC), which is generally not alphabetically or by frequency, and then order the Preferred Terms (PT) within a SOC by descending frequency of occurrence (the actual data drives this step, and that order may change with updates to the data). While important to understand, creating the VALDS is not within the scope of this paper. The macro M_R_REPORT takes the instructions in a data set instead of the programmer coding them into statements of the REPORT procedure. The programmer may not even see that REPORT code created (although the regulatory agencies may require code, not calls of externally defined macros). IMPORTANTLY, all of that information: labels, order, widths, et cetera is *inherent in a properly written shell*, but its availability at the moment is through manual translation, not a programmatically reading the shell. An output that has varying number of columns or varying column headers on different physical pages, for instance, which in SAS requires separate REPORT procedures, illustrates the utility of MAC_R_REPORT. Corrections or updates to such output further emphasize the utility and ease of use MAC_R_REPORT as the rest of the paper will demonstrate. To obtain a brief description of the macro and its parameter, one uses HELP = Y, which ignores all other parameters and exits the macro after writing the help section to the log (not shown).

REPORT_DEFINE_OPTIONS

MAC_R_REPORT requires a second data set beyond the input data that provides code and information to the REPORT procedure. Without this data set, the macro issues an ERROR message and terminates. One of the challenging aspects of creating output is finessing the widths of columns given the constraints of the margins, number of columns, the font size, and the font family. Having multiple physical pages with varying numbers of columns and widths of columns complicates this issue. MAC_R_REPORT allows the programmer to display the column widths and their running totals established in the REPORT_DEFINE_OPTIONS dataset and then exit. The macro assumes that the units of CELLWIDTH will be percent (%) and does not account for additional "null" columns computed by the REPORT procedures to "space" the column headers. **Table 3** shows the output created for the data set created in **Appendix 2**.

```
%mac_r_report
   ( page_by_vars              = page_1
   , report_column_widths      = Y
   ) ;
```

If needed, one can "fine tune" the cellwidths and keep track of the running total (and ultimate total) without investing the time into repeated running the code and waiting for ODS to write to the destination to see the effects. Generally, keeping the grand total under 99% and lower, if "null" columns to space headers spanning multiple columns might be used, reduces the occurrence of columns being pushed to another physical page of the output.

Table 3. The display of column widths and their running total produced by MAC_R_REPORT.

```
page_1=ALT

type             __var1     __var2     __var3     __var4     __var5     __var6     __var7

variable         row_1      col1       col2       col3       col4       col5       col6
cellwidth        15         10         10         10         10         10         10
running total    15         25         35         45         55         65         75
```

ZERO OBSERVATIONS

Occasionally, a requested table or listing cannot be produced because the data does not exist. For example, no subject experienced treatment-emergent, serious adverse events in the trial. Occasionally, this poses problems since many programs do not plan for zero observations. The emergence of %IF-%THEN statements in *open SAS code* has facilitated ways of accounting for this issue, but two issues continue for the programmers. First, what does the Validation programmer do with a data set with zero observations (or no observations in the VALDS with TYPE = "B", that is, observations in the **B**ody of the report)? Even performing a log check might not assure the Validation programmer that the Main program ran as intended. Second, informative output may still be required. Depending on what the SOPs/WIs prescribe, MAC_R_REPORT covers these situations by creating a VALDS with a single observation and a single variable and creates the output with the text in that variable with the expected titles and footnotes making explicit that no qualifying data exists. **Table 4** illustrates a VALDS with no resulting observations and the use of custom text using the ZERO_OBSERVATION_TEXT= parameter.

Table 4. Zero observation with custom text.

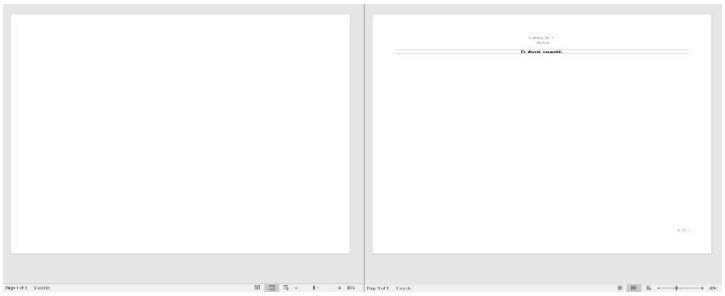
A

```
data l16_1_1 ;
  length page_1
         page_order_1
         section_1
         section_order_1
         row_1
         section_order_1
         col1 - col6 $ 200
         ;
  call missing( page_1
              , page_order_1
              , section_1
              , section_order_1
              , row_1
              , section_order_1
              , of col1 - col6
              ) ;
  stop ;
run ;
```

B

```
%mac_r_report
   ( in_ds                    = l16_1_1
   , page_by_vars             = page_1
   , num_of_cols              = 5
   , code_after_define        = %str()
   , zero_observation_text    = No deaths occurred.
   , make_zero_obs_ds         = Y
   ) ;
```

Figure 1A demonstrates the resulting output when code does not account for a VALDS with zero observations is the input data set to a typical REPORT procedure. Notably, it lacks titles, footnotes, and, of course, a report body; it is blank potentially causing confusion. Further, the summary provided by MS Word shows Page 1 of 1 and 0 words indicating that a valid .rtf file is open but blank. Figure 1B demonstrates the results of using the code in Table 4.



1A **1B**

Figure 1. Output created for no qualifying observations.

The resulting VALDS created by the code in Table 4 is:

```
Obs            col0

 1      No deaths occurred.
```

Note that in audit of the directories[6] (including a directory read), the Lead, Manager, or Project Manager, will find COMPARE procedure results and Last Modified datetime stamps for programs that have no qualifying observations with either situation in Table 4. The SOPs/WIs may direct the programmers, but when the programmers/managers otherwise have latitude, the approach by MAC_R_REPORT may reduce confusion by making the situation explicit.

DEMONSTRATION VALDS

Appendix 3 displays an image of the VALDS used to demonstrate MAC_R_REPORT in this paper. **Appendix 4** includes the data in CSV format and a suggested data set to INFILE it. As displayed in Appendix 4, the lines wrap and when copied to a text file, the wrapping must be eliminated by deleting to create the SAS data set in Appendix 3. When viewing the resulting datalines, the first field should be left justified (no leading spaces) and be “ALT” or “AST”. The creation of this data set from ADSL and ADLB is out of scope for this paper.

DEMONSTRATION OF MAC_R_REPORT

For the purposes of this paper, no assumptions of an environment or macros pertaining thereof are needed. Usually, a macro might handle the titles and footnotes (**Lines** 1-16 below), another might handle the ODS destination, and a template might be available to format the output. The following code is, thus, verbose compared to a macro call in a typical program:

```
 1  title1 j = c "Table 14.2.1" ;
 2  title2 j = c "Laboratory Results" ;
 3  title3 j = c "Safety Population" ;
 4  title5 j = l "Parameter = #byval( page_1 )" ;
 5  footnote1 "Very important text" ;
 6  footnote3 "#byval( footer_3 )" ;
 7
 8  /* footnote3 "`{super A} The following subject(s) have missing values: 001-01-0001`n`{super B} The following subject(s) have missing values:
 9  001-01-0001`n`{super C} The following subject(s) have missing values: 001-01-0001`n`{super D} The following subject(s) have missing values: 001-
10  01-0001" ;
11  */
12
13  footnote5 j = r "`{pageof}" ;
14
15  options orientation             = landscape
16          topmargin               = 0.75in
17          bottommargin            = 0.75in
18          rightmargin             = 0.75in
19          leftmargin              = 0.75in
20          ;
21
22  ods listing close ;
23  ods results ;
24
25  ods rtf
26      file  = "&path.\phuse_main_1.rtf"
27      ;
28
29  %mac_r_report
30     ( in ds                      = tfl.tfl_outsas
31     , page by vars               = page 1
32     , num of cols                = &num of cols.
33     , report define options ds   = report_define_options
34     , report column widths       = N
35     , report options             = style( report ) = { width             = 100%
36                                                        frame             = hsides
37                                                        rules             = groups
38                                                        cellpadding       = 1pt
39                                                        cellspacing       = 0pt
40                                                        borderwidth       = 1
41                                                        fontfamily        = "Courier New"
42                                                        fontsize          = 9pt
43                                                        backgroundcolor   = white
44                                                      }
45                                    style( header ) = { backgroundcolor   = white
46                                                        fontweight        = medium
47                                                      }
48                                    style( column ) = { just              = c
49                                                        vjust             = m
50                                                        backgroundcolor   = white
51                                                      }
52                                    missing
53
54     , code_after_define          = compute before section_order_1
55                                           / style = { height = 18pt }
56                                           ;
57                                       line " " ;
58                                    endcomp ;
59
60                                    compute row_1 ;
61
62                                       if row order 1 > 0 then
63                                       call define( _col_
64                                                  %str(,) "style"
65                                                  %str(,) "style = { leftmargin = 1% }"
66                                                  ) ;
67
68                                    endcomp ;
69     , report_by                  = page_1
70     ) ;
71
72  ods rtf close ;
73  ods listing ;
74
```

Note that this code includes two versions of FOOTNOTE3, the first a text substitution (**Line** 8) and the second with hard-coded text (**Lines** 10-14, in a block comment). The purpose is to contrast the global nature of FOOTNOTE3, highlighting the ability to add footnotes only to specific pages. In this example, the macro generates the code and executes two REPORT procedures, one for each value of PAGE_1. When **Line 8** is executed, the macro may produce a value for the variable FOOTER_3 or it may remain missing (blank). With planning, FOOTNOTE_3 (or its equivalent) could be populated only for certain combinations, for instance, PAGE_1 = “AST” and PAGE_2 = “2”, where PAGE_2 controls the number of rows per physical page (and is not displayed). When **Line** 8 is commented out and **Lines** 10-13 are executed, the text is displayed on each physical page of the output. **Figure 2** displays the output using the first version, which is only visible when PAGE_1 = “ALT” (Parameter = ALT, top panels) since MAC_R_REPORT creates FOOTER_3 as blank (missing) for PAGE_1 = “AST” (see Appendix 3). Note the FOOTNOTE3 statement executes, even with a blank text string, which is a questionable, but potentially acceptable use of space in the output.

Several issues pertaining to titles and footnotes that are out of scope of this paper are worth observing. The footnotes occupy too much “real estate”, reducing the number of rows in body of the table. The number (output area required) of the footnotes may (will) vary, complicating an attempt to standardize the number of rows per physical page. Other methods are available instead of GLOBAL FOOTNOTE statement, such as an “inline” footnote via a COMPUTE BLOCK in the REPORT procedure. Short of re-executing TITLE and FOOTNOTE statements, one may not substitute a title or footnote if higher statements exist, for example, FOOTNOTEn substitutes FOOTNOTEn *and* erases FOOTNOTE(n+1) to FOOTNOTE10. SASHELP.VTITLE contains the text of the titles and footnotes, but not the justification or any (inline style) “decorations”. Dynamic footnotes of this type are DERIVED data and, thus, should be Validated. The listing of subjects omitted from a section may be very important to a reviewer, especially, but not exclusively, if subjects are not missing at random (MAR) and a concise footnote will make that review more efficient and accurate than referencing a supporting listing.

Titles and footnotes can affect the interpretation of a table or listing, providing important information, like a key for abbreviations, definitions, or citations to the SAP. Frequently, the titles and footnotes for an entire delivery are maintained in a separate, versioned file, the datetime stamp of which may be subject to audits for versioning control purposes. Dynamically writing to that file may not be good practice and would affect versioning and/or datetime stamps. This approach allows *programmatic Validation* in the data set, which explicitly includes the custom (derived) text of the titles and footnotes and MAC_R_REPORT programmatically displays such dynamic text and data.

Since MAC_R_REPORT creates a REPORT procedure for each group defined in the PAGE_BY_VARS parameter (here PAGE_1), it could create a global title or footnote specific to that group, obviating the need for #BYVAR or #BYVAL, but these substitution make the title or footnote obvious in the (versioned) title and footnote (TF) file, which is presumably reviewed and approved by key personnel, like the biostatistician and (medical) scientists. Obviously, the resolved values are not available in the TF file but using a footnote like the first version acts as placeholder in the TF file so that key personnel is aware that additional footnotes may appear and reduces the risk of erasing titles or footnotes with an errant calculation of n (1-10, as in FOOTNOTEn).

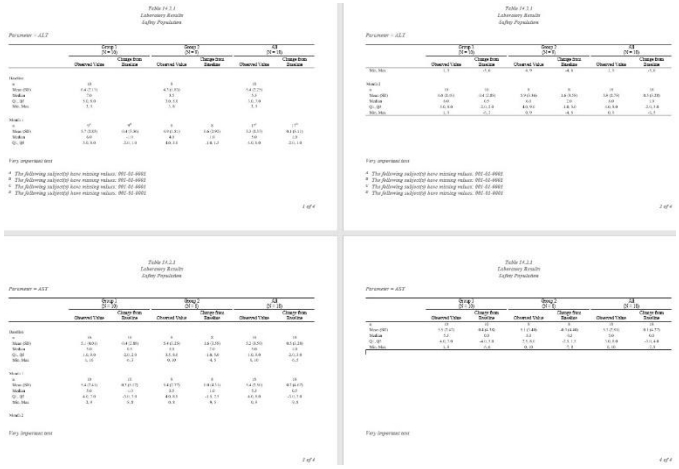


Figure 2. RTF output created by the main example code of this paper.

Lines 18 -30 are also usually omitted from the execution. A macro usually executes these statements. The ODS RTF statement usually includes a STYLE= option. If so, then **Lines** 38-56 might be omitted completely; **Lines** 16-17 of **Appendix 1** provides a default value of the macro parameter REPORT_OPTIONS:

```
report_options              = style( report ) = { width = 100% }
                              missing
```

The macro parameter CODE_AFTER_DEFINE provides the opportunity of free text (code). **Lines** 58-62 will insert a blank line in the output before each new value of the SECTION_ORDER_1, which is thus required to be an "ORDER" variable in the REPORT_DEFINE_OPTIONS data sets, i.e. the ORDER (or GROUP) option to the DEFINE statement for SECTION_ORDER_1. These lines match the default value, reflecting the fact that this is the standard for the current work of the author. The user can update the default value or the code of the macro. Note that the macro parameter value includes both equal signs (=) and semi-colons (;). These need not be masked since SAS tokenizes this code without issues. This contrasts with the commas (,) in **Lines** 68-69, which are masked by the %STR() macro function.

Lines 66-72 indent the "row labels" under their "section label". We can achieve this by adopting conventions of numbering ROW_ORDER_1 appropriately. The "indentation" is achieved using the LEFTMARGIN style element, instead of the INDENT style element. Viel[1] demonstrates the advantage, which is relevant when the label is "too long" and wraps.

Line 77 closes the ODS RTF destination (by virtue of ODS RESULTS in PC SAS, the Windows operating system open the RTF file automatically). **Line** 78 returns the interactive output to the result window. **Lines** 80-81 reset the TITLE and FOOTNOTE statements, which is relevant to an interactive environment, for instance, during development.

UPDATE OF SHELL

Updates to the shell may occur within a delivery, between deliveries, or between projects. Some common updates include omission of column (for readability, for example), addition of columns (for example, to combine two groups totals), or two add, delete, or reorder sections or rows. To achieve this in the VALDS is beyond the scope of this paper. Consider, the following reordering of the pages and omission of the "All" columns for this particular example:

```
data reorder ;
  set tfl.tfl_outsas ;
  page_order_1 = mod( page_order_1 , 2 ) ;

  array __c ( * )
        col1 - col&num_of_cols.
        ;

  if page_1 = "AST"
  then call missing ( col5
                    , col6
                    ) ;
run ;
```

Re-running the code above with IN_DS = REORDER results in a major change in the output (not shown), with no change in the macro call or other code not associated with the creation of the IN_DS data set. Of course, the SAS code generated by the macro call will differ (greatly), but not at the "expense" of the programmer calling MAC_R_REPORT, that part of the program remains remarkably similar.

CAPTURING THE GENERATED SAS CODE (SUBMISSION CONCERNS)

Four major issues might concern a lead programmer and/or managers about a call to macro that accomplishes so much. Admittedly, one usually abandons a project with one major issue, but *four* major issues usually persuade readers to continue just to see to where this might go. First, how to assure that it is accurate (and efficient, including code development/correction time)? Second, what are the courses of action when it is not working as intended? Third, how does a programmer or, perhaps more importantly, a legacy programmer maintain and adopt or update the code? Fourth, are these programs *submission-ready*?

The only person, in general, who wants a macro programmer who programs for job security is the macro programmer (that is until she or he also must update or change the program). Since the fourth issue is the meaningful bottleneck of the whole process, obtaining and preserving the SAS code generated by a macro is paramount. In this regard, one can obtain the SAS code using MFILE, but this can be captured using the MAC_DEBUG_MFILE macro[7]. Importantly, in the Windows (PC SAS) environment one might be able to interactively run the macro then paste the code copied to the Windows clipboard. It can also be directed to a file, which one might then manual or programmatic format for readability. The latter is beyond the scope of this paper.

For example, wrapping the code above with the MAC_DEBUG_MFILE:

```
%mac_debug_mfile( mcall = mac_r_report
   ( in_ds                    = tfl.tfl_outsas
   , page_by_vars             = page_1
   <SNIPPED>
   ));
```

generates the code (partially shown):

```
proc sql noprint ;
select count( * ) into : num_obs from tfl.tfl_outsas ;
quit ;
proc sql noprint ;
select distinct page_1 , page_order_1 into : __d1_1 - , : null1 - from tfl.tfl_outsas order by
page_order_1 ;
quit ;
<SNIPPED>
proc report data = __mrr split = "#" style( report ) = { width = 100% frame = hsides
<SNIPPED>
```

Some of the code is quite superfluous to producing the output, but instead is used to direct the macro. An experienced user, capable macro programmer, or programmer with an inclination to experiment might parse this code and pare into to the bare code necessary. Otherwise, the code is far from too verbose and will run as intended. At any rate, most programmers should be able to understand and adopt the code instead of using MAC_R_REPORT: a program writing programs, but in this case, one in which the programmer who becomes the owner of the code may need to format it, for example, add blank lines between boundaries of data steps and procedures or indentations to format. All of that is purely for readability and does not affect the results of the code. Some (if not all) of this might be accomplished programmatically, but that is beyond the scope of this paper.

MACRO EXPLAINED

Besides the help section, the macro (**Appendix 1**) has three parts. The first only reports the column widths and their running totals (**Lines 145-300**). The second deals with the situation of no qualifying observations resulting in zero observation (**Lines 303-355**). The third (**Lines 358-1124**) processes the data and instructions, including providing instructions to the macro for logic (decisions), and reporting.

Some macro code might require brief explanation, but some of the reason is not because it is macro code, but because it is more advanced BASE SAS code. For example (on one line here, but **Lines 152-156**):

```
%sysfunc( prxchange( s/ / %str(,) / , -1 , &page_by_vars. ))
```

This is a macro version of a data step function for regular expression (regex). It substitutes (s///) a single space with " , " (a spaced delimited comma). Multiple spaces have already been removed from PAGE_BY_VARS. If the value was "PAGE_1 PAGE_2", the result of this function would be "PAGE_1 , PAGE_2", which is the required format for the SELECT clause of the SQL procedure. Of note is that absence of quotation marks that would be required in the data step version:

```
prxchange( "s/ / , /" , -1 , "&page_by_vars." )
```

Note that the comma in the macro version had to be masked using the %STR() macro function. The SQL code in **Lines 363-395** obtains the page_by_variables and their respective variables. For instance, in this example, &__d1_1. = ALT. An important task is to determine the number of columns in the current (physical) page (current "level" of the "group" defined by the variables listed in the macro parameter PAGE_BY_VARS) or the entire data set if PAGE_BY_VARS is null (**Lines 405-461**). To emphasize, in the design of this macro, column headers determine if a column is included or omitted because null values may be valid (derived) values of a column, but missing column headers are not. An alternative is to include a data set that specifies the columns per page-by group, but the reader would then have to adapt the macro to such requirements. Column headers are processed in **Lines 464-730**. One assumption is that a non-missing column header is "carried" left. Two TRANSPOSE procedures accomplish this, including making headers of (column) headers. If groups of columns have a common header (header of headers), depending on the style elements, null columns might be desired to create a small break for readability (**Lines 598-620**). The macro code from **Line 732** pertains to the REPORT procedure. Recalling from above that &__d1_1. = ALT, the following macro code might require explanation:

```
&&__d&__pb._&__level.
```

Assume that __pb = 1 and __level = 1 (see the %DO loops). The first resolution is:

```
&__d1_1
```

With one remaining ampersand (&), another resolution occurs to the value ALT. Besides token masking (and unmasking), mastering such macro variables is a milestone of SAS macro programming. **Lines 755-847** use the REPORT_DEFINE_OPTIONS_DS data set (by default, REPORT_DEFINE_OPTIONS) to obtain the variables and the options for their respective DEFINE statements. **Lines 857-1061** creates the ultimate input data set, including variables for dynamics footnotes, if applicable. **Lines 1064-1112** pertain to REPORT procedure(s). Exploring the various circumstances and using MAC_DEBUG_MFILE provides opportunities to experiment, including with user updates to the macro.

CONCLUSION

This paper contributes to ways to automate creation of tables and listing while increasing accuracy and traceability in a manner that aids in reusability with minimal changes to the generating program code. Importantly, this approach facilitates manual (spot) checking of the out against the Validation Data Set, a penultimate task in approving and promoting a program. It addresses parsimony with a focus on audit/reconciliation of potential thousands of files in a delivery, including situations in which SAS data sets with zero observations and output with a single blank page might otherwise persist and confuse reviewers not familiar with the process or data. Whether as a stand-alone product or a program generating programs, the macro MAC_R_REPORT enables programmers to create more standardized programs, resulting in fewer overall changes between updates (deliveries) or projects.

REFERENCES

[1] Viel, KR. “A Standard Data Set Format for the Validation of Tables and Listings in Regulatory Submissions for Clinical Trials.” Conference Proceedings: PHUSE US Connect 2023. PP17. Available at https://phuse.s3.eu-central-1.amazonaws.com/Archive/2023/Connect/US/Florida/PAP_PP17.pdf and https://phuse.s3.eu-central-1.amazonaws.com/Archive/2023/Connect/US/Florida/POS_PP17.pdf.

[2] Busa B, Marshall R, and LeRoy B. 2023. "All You Need to Know about the New CDISC Analysis Result Standards!" Proceedings of the PharmaSUG 2023 Conference, San Francisco, CA: PharmaSUG. https://www.pharmasug.org/proceedings/2023/MM/PharmaSUG-2023-MM-327.pdf

[3] Busa B and Dedhia N. 2023. "Introducing TFL Designer: Community Solution to Automate Development of Mock-up Shells and Analysis Results Metadata." Proceedings of the PharmaSUG 2023 Conference, San Francisco, CA: PharmaSUG. https://www.pharmasug.org/proceedings/2023/HT/PharmaSUG-2023-HT-355.pdf

[4] Viel, K. 2024. " Standardizing Validation Data Sets (VALDS) as matrices indexed by Page, Section, Row, and Columns (PSRC) to improve Validation and output creation and revisions." Proceedings of the PharmaSUG 2024 Conference, Baltimore, MD: PharmaSUG. (*Submitted*)

[5] SAS Institute Inc. Unknown. “Usage Note 24542: How to get the exact order of your input data set in PROC REPORT output when the ORDER=DATA option is specified in a DEFINE statement.” 2007-12-17 10:47:13. Available at https://support.sas.com/kb/24/542.html

[6] Viel, K. 2023. "SAS® System Macros to Summarize the COMPARE Procedure Results and SAS Logs for a Directory or Single File." Proceedings of the PharmaSUG 2023 Conference, San Francisco, CA: PharmaSUG. https://www.pharmasug.org/proceedings/2023/SD/PharmaSUG-2023-SD-221.pdf

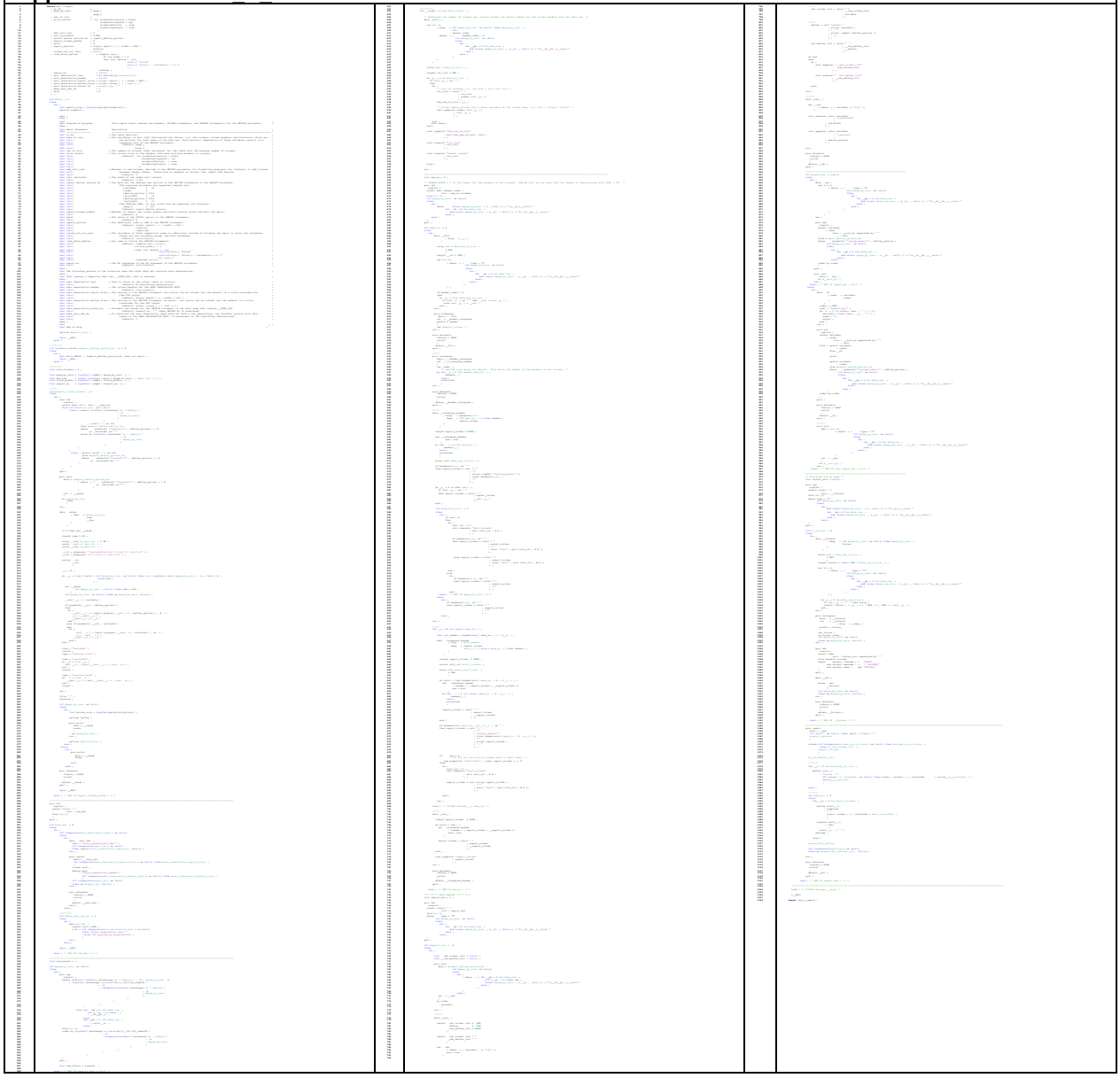
[7] Viel, K. 2016. "Capturing Macro Code when Debugging in the Windows Environment: The Power of MFILE and the Simplicity of Pasting." Proceedings of the PharmaSUG 2016 Conference, Denver, CO: PharmaSUG. https://www.pharmasug.org/proceedings/2016/TT/PharmaSUG-2016-TT17.pdf

CONTACT INFORMATION

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

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

Appendix 1. The MAC_R_REPORT macro.



Appendix 2. The example REPORT_DEFINE_OPTIONS data set.

	page_1	variable	define_options	cellwidth	order
1	ALT	page_order_1	order order = data noprint		1
2	ALT	page_1	order order = data noprint		2
3	ALT	section_order_1	order order = data noprint		3
4	ALT	row_order_1	order order = data noprint		4
5	ALT	row_1	" " order style (column) = { cellwidth = 15% just = l vjust = top } style (header) = { just = l }		5
6	ALT	col1		10%	6
7	ALT	col2		10%	7
8	ALT	col3		10%	8
9	ALT	col4		10%	9
10	ALT	col5		10%	10
11	ALT	col6		10%	11
12	AST	page_order_1	order order = data noprint		1
13	AST	page_1	order order = data noprint		2
14	AST	section_order_1	order order = data noprint		3
15	AST	row_order_1	order order = data noprint		4
16	AST	row_1	" " order style (column) = { cellwidth = 15% just = l vjust = top } style (header) = { just = l }		5
17	AST	col1		10%	6
18	AST	col2		10%	7
19	AST	col3		10%	8
20	AST	col4		10%	9
21	AST	col5		10%	10
22	AST	col6		10%	11

Appendix 3. The example IN_DS data set.

	page_1	page_order_1	section_1	section_order_1	row_1	row_order_1	col1	col2	col3	col4	col5	col6	header_order	type	type_order	footer_order
1	ALT		1	Header	0	Header	0	Group 1'n(N = 10)		Group 2'n(N = 8)		All'n(N = 18)		1	H	1	
2	ALT		1	Header	0	Header	0	Observed Value	Change from'nBaseline	Observed Value	Change from'nBaseline	Observed Value	Change from'nBaseline	0	H	1	
3	ALT		1	Baseline	1	Baseline	0								B	2	
4	ALT		1	Baseline	1	n	1	10		8		18			B	2	
5	ALT		1	Baseline	1	Mean (SD)	2	6.4 (2.17)		4.3 (1.83)		5.4 (2.25)			B	2	
6	ALT		1	Baseline	1	Median	3	7.0		3.5		5.5			B	2	
7	ALT		1	Baseline	1	Q1, Q3	4	5.0, 8.0		3.0, 5.0		3.0, 7.0			B	2	
8	ALT		1	Baseline	1	Min, Max	5	3, 9		3, 8		3, 9			B	2	
9	ALT		1	Month 1	2	Month 1	0								B	2	
10	ALT		1	Month 1	2	n	1	9'(super A)		8		17'(super C)	17'(super D)		B	2	
11	ALT		1	Month 1	2	Mean (SD)	2	5.7 (2.83)		-0.4 (3.36)		4.9 (1.81)	0.6 (2.92)		B	2	
12	ALT		1	Month 1	2	Median	3	6.0		-1.0		4.0	1.0		B	2	
13	ALT		1	Month 1	2	Q1, Q3	4	5.0, 8.0		-2.0, 1.0		4.0, 5.0	-1.0, 1.5		B	2	
14	ALT		1	Month 1	2	Min, Max	5	1, 9		-5, 6		-4, 6	-5, 9		B	2	
15	ALT		1	Month 2	3	Month 2	0								B	2	
16	ALT		1	Month 2	3	n	1	10		8		18			B	2	
17	ALT		1	Month 2	3	Mean (SD)	2	6.0 (2.40)		-0.4 (2.88)		5.9 (3.36)	1.6 (3.58)		B	2	
18	ALT		1	Month 2	3	Median	3	6.0		0.5		6.0	2.0		B	2	
19	ALT		1	Month 2	3	Q1, Q3	4	5.0, 8.0		-2.0, 2.0		4.0, 9.0	-1.0, 5.0		B	2	
20	ALT		1	Month 2	3	Min, Max	5	1, 9		-6, 3		0, 9	-4, 5		B	2	
21	ALT		1	Footer	0	Footer	0	'(super A) The following subject(s) have missing values: 001-01-0001'n '(super B) The following subject(s) have missing values: 001-01-0001'n '(super C) The following subject(s) have missing values:	001-01-0001'n '(super D) The following subject(s) have missing values: 001-01-0001					F	3		3
22	AST		2	Header	0	Header	0	Group 1'n(N = 10)		Group 2'n(N = 8)		All'n(N = 18)		1	H	1	
23	AST		2	Header	0	Header	0	Observed Value	Change from'nBaseline	Observed Value	Change from'nBaseline	Observed Value	Change from'nBaseline	0	H	1	
24	AST		2	Baseline	1	Baseline	0								B	2	
25	AST		2	Baseline	1	n	1	10		8		18			B	2	
26	AST		2	Baseline	1	Mean (SD)	2	5.1 (4.01)		-0.4 (2.88)		5.4 (3.25)	1.6 (3.58)		B	2	
27	AST		2	Baseline	1	Median	3	5.0		0.5		5.0	2.0		B	2	
28	AST		2	Baseline	1	Q1, Q3	4	1.0, 9.0		-2.0, 2.0		3.5, 8.0	-1.0, 5.0		B	2	
29	AST		2	Baseline	1	Min, Max	5	1, 10		-6, 3		0, 10	-4, 5		B	2	
30	AST		2	Month 1	2	Month 1	0								B	2	
31	AST		2	Month 1	2	n	1	10		8		18			B	2	
32	AST		2	Month 1	2	Mean (SD)	2	5.4 (2.41)		0.3 (5.17)		5.4 (2.77)	0.0 (4.31)		B	2	
33	AST		2	Month 1	2	Median	3	5.0		-1.0		5.5	1.0		B	2	
34	AST		2	Month 1	2	Q1, Q3	4	4.0, 7.0		-3.0, 3.0		4.0, 8.0	-1.5, 2.5		B	2	
35	AST		2	Month 1	2	Min, Max	5	2, 9		-8, 8		0, 9	-5, 5		B	2	
36	AST		2	Month 2	3	Month 2	0								B	2	
37	AST		2	Month 2	3	n	1	10		8		18			B	2	
38	AST		2	Month 2	3	Mean (SD)	2	5.5 (2.42)		0.4 (4.38)		5.1 (3.40)	-0.3 (4.40)		B	2	
39	AST		2	Month 2	3	Median	3	5.5		0.0		5.0	-0.5		B	2	
40	AST		2	Month 2	3	Q1, Q3	4	4.0, 7.0		-4.0, 5.0		2.5, 8.0	-2.5, 1.5		B	2	
41	AST		2	Month 2	3	Min, Max	5	1, 9		-6, 6		0, 10	-7, 8		B	2	

Appendix 4. The CSV version of the IN_DS of Appendix 3.