PharmaSUG 2022 - Paper RW-92

Automation of Variable Name and Label Mapping from Raw Data to Analysis Data Standards

Bo Zheng, Li Ma, and Xingshu Zhu, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

Raw data files with no standard variable naming and labeling conventions are a common occurrence in non-interventional primary data collection studies. As real-world data gains more relevance in the pharmaceutical industry, it becomes important to have a repeatable and consistent process to map raw variable names and labels to standardized versions for analysis and reporting. This paper shows two methods for importing and mapping variables in SAS to reduce the time spent on data management. One method leverages the built-in features of SAS PROC IMPORT as a simple direct solution for importing and mapping raw data to predefined standards. The second method uses SAS to extract the relevant metadata from a raw file to create an Excel mapping template and facilitates collaboration with non-SAS users by making it easier for them to access and comment on the mapping process.

INTRODUCTION

The general lack of standardization for variable names, file formats, documentation, and analysis processes continue to be long-term challenge for statistical programmers working in Real-world Evidence (RWE). Microsoft® Excel spreadsheets are the most common format for raw data distribution in non-interventional primary data collection studies (PDCS). The flexibility and ease-of-use makes Excel a popular tool for entering and saving data from primary sources. Potential challenges arise when a researcher tries to import these files in SAS® and finds data quality issues such as long descriptors, Unicode, or other unsupported formats instead of standardized variable names and labels. A manual variable mapping process in the Excel data file is inefficient and may introduce additional user errors when used on large complex raw data. A repeatable and consistent process to map raw variable names and labels to standards at the research group or organization level is needed to ensure high-quality deliverables in analysis and reporting (A&R). A well-defined process also helps to accelerate the overall development of RWE and PDCS standards when shared across the pharmaceutical research industry.

In this paper, two examples for standardizing the variable name and label mapping process are presented. The first example uses the built-in features of SAS PROC IMPORT and the second method uses SAS to create an Excel spreadsheet mapping template from relevant metadata. Both processes can be described as having these main components:

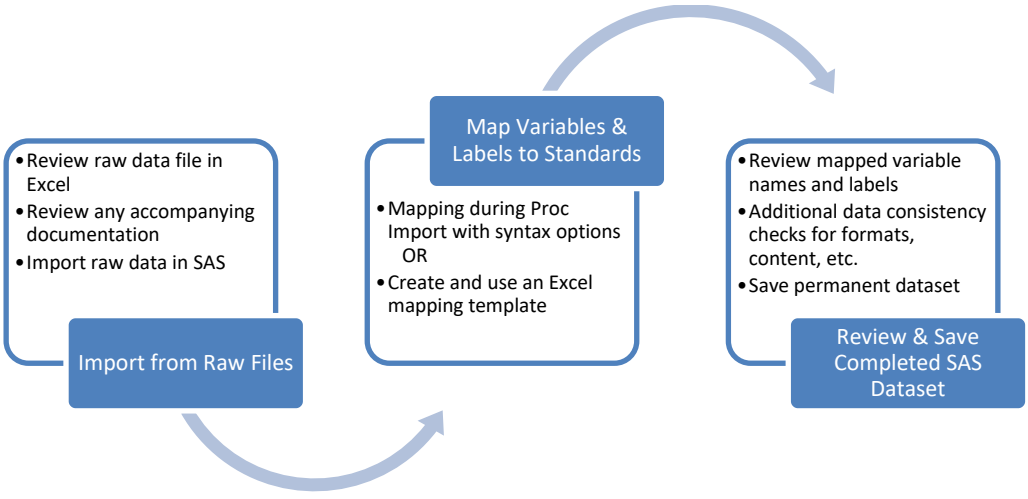


Figure 1. Generalized data mapping process

METHOD

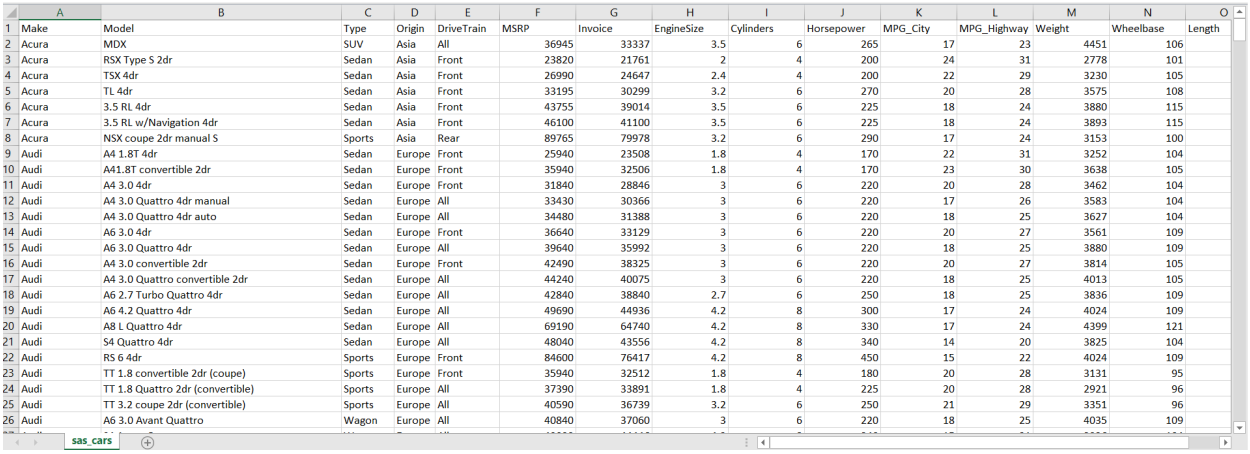
Before selecting a preferred mapping process, review the raw data file in Excel and any accompanying documentation. Pay special attention to the number of columns and rows in the raw file that contain data and whether the file has variable names and labels stored in other rows.

EXAMPLE 1: MAPPING WITH SAS PROC IMPORT SYNTAX OPTIONS

For this example, the sashelp.cars dataset is used, start by exporting it to an Excel file:

```
proc export data    = sashelp.cars
            dbms    = xlsx replace
            outfile = "C:\Users\Documents\sas_cars.xlsx" ;
run ;
```

The output of sashelp.cars as an Excel file is a good example of what an ideal raw datafile would look like; it's clean data, includes variable names in the first row, and has consistent formatting. It also contains no Unicode or other special characters and obeys SAS rules in its general structure:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
2	Acura	MDX	SUV	Asia	All	36945	33337	3.5	6	265	17	23	4451	106	
3	Acura	RSX Type S 2dr	Sedan	Asia	Front	23820	21761	2	4	200	24	31	2778	101	
4	Acura	TSX 4dr	Sedan	Asia	Front	26990	24647	2.4	4	200	22	29	3230	105	
5	Acura	TL 4dr	Sedan	Asia	Front	33195	30299	3.2	6	270	20	28	3575	108	
6	Acura	3.5 RL 4dr	Sedan	Asia	Front	43755	39014	3.5	6	225	18	24	3880	115	
7	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	46100	41100	3.5	6	225	18	24	3893	115	
8	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	89765	79978	3.2	6	290	17	24	3153	100	
9	Audi	A4 1.8T 4dr	Sedan	Europe	Front	25940	23508	1.8	4	170	22	31	3252	104	
10	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	35940	32506	1.8	4	170	23	30	3638	105	
11	Audi	A4 3.0 4dr	Sedan	Europe	Front	31840	28846	3	6	220	20	28	3462	104	
12	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	33430	30366	3	6	220	17	26	3583	104	
13	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	34480	31388	3	6	220	18	25	3627	104	
14	Audi	A6 3.0 4dr	Sedan	Europe	Front	36640	33129	3	6	220	20	27	3561	109	
15	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	39640	35992	3	6	220	18	25	3880	109	
16	Audi	A4 3.0 convertible 2dr	Sedan	Europe	Front	42490	38325	3	6	220	20	27	3814	105	
17	Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	All	44240	40075	3	6	220	18	25	4013	105	
18	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	All	42840	38840	2.7	6	250	18	25	3836	109	
19	Audi	A6 4.2 Quattro 4dr	Sedan	Europe	All	49690	44936	4.2	8	300	17	24	4024	109	
20	Audi	A8 L Quattro 4dr	Sedan	Europe	All	69190	64740	4.2	8	330	17	24	4399	121	
21	Audi	S4 Quattro 4dr	Sedan	Europe	All	48040	43556	4.2	8	340	14	20	3825	104	
22	Audi	RS 6 4dr	Sports	Europe	Front	84600	76417	4.2	8	450	15	22	4024	109	
23	Audi	TT 1.8 convertible 2dr (coupe)	Sports	Europe	Front	35940	32512	1.8	4	180	20	28	3131	95	
24	Audi	TT 1.8 Quattro 2dr (convertible)	Sports	Europe	All	37390	33891	1.8	4	225	20	28	2921	96	
25	Audi	TT 3.2 coupe 2dr (convertible)	Sports	Europe	All	40590	36739	3.2	6	250	21	29	3351	96	
26	Audi	A6 3.0 Avant Quattro	Wagon	Europe	All	40840	37060	3	6	220	18	25	4035	109	

sas_cars

Display 1. Contents of SASHELP.CARS as an Excel file

The first method is best suited for smaller datasets where there are no major problems that SAS can't automatically correct. For this example, let's ignore row 1 and assume that the variable names and labels for sashelp.cars are not already well-defined and will need to be mapped to standards. After opening and reviewing the file in Excel, note the name of the sheet, the number of columns, and the number of rows that data is stored in. In the example the name of the sheet is "sas_cars" and the relevant data is stored in A2 through O429. In SAS, the PROC IMPORT step alone can handle reading in the raw Excel file, mapping the raw variable names to standard with the RENAME option, and assigning labels and formats. The GETNAMES option is set to 'no', with DBMS=XLSX and SAS 9.4, the default variable naming is based on the letter of the column in Excel. The syntax for the example looks like this:

```
proc import datafile="C:\Users\Documents\sas_cars.xlsx"
            out=work.sas_cars
            /* ASSIGN STANDARD VARIABLE NAMES HERE */
            (rename=(A = Make
                     B = Model
                     C = Type
                     D = Origin
                     E = DriveTrain
                     F = MSRP
                     G = Invoice
                     H = EngineSize
                     I = Cylinders
                     J = Horsepower
```

```
                K = MPG_City
                L = MPG_Highway
                M = Weight
                N = Wheelbase
                O = Length))
        /* PROC IMPORT OPTIONS */
        dbms      = xlsx replace ;
        range     ="sas_cars$A2:O429" ;
        getnames  = no ;
        /* ASSIGN LABELS HERE */
        label Make         = "Make"
              Model        = "Model"
              Type         = "Type"
              Origin       = "Origin"
              DriveTrain   = "Drivetrain"
              MSRP         = "MSRP ($)"
              Invoice      = "Invoice ($)"
              EngineSize   = "Engine Size (L)"
              Cylinders    = "Cylinders"
              Horsepower   = "Horsepower"
              MPG_City     = "MPG (City)"
              MPG_Highway  = "MPG (Highway)"
              Weight       = "Weight (LBS)"
              Wheelbase    = "Wheelbase (IN)"
              Length       = "Length (IN)" ;
        /* ASSIGN FORMATS HERE */
        format MSRP Invoice DOLLAR8. ;
run ;
```

The overall code is simple but still gives the SAS user full control of the incoming dataset for defining the naming, labeling, and formatting in a single import step. The output dataset after import looks like this:

Variables in Creation Order						
#	Variable	Type	Len	Format	Informat	Label
1	Make	Char	13	$13.	$13.	Make
2	Model	Char	40	$40.	$40.	Model
3	Type	Char	6	$6.	$6.	Type
4	Origin	Char	6	$6.	$6.	Origin
5	DriveTrain	Char	10	$10.	$10.	Drivetrain
6	MSRP	Num	8	DOLLAR8.		MSRP ($)
7	Invoice	Num	8	DOLLAR8.		Invoice ($)
8	EngineSize	Num	8	BEST.		Engine Size (L)
9	Cylinders	Num	8	BEST.		Cylinders
10	Horsepower	Num	8	BEST.		Horsepower
11	MPG_City	Num	8	BEST.		MPG (City)
12	MPG_Highway	Num	8	BEST.		MPG (Highway)
13	Weight	Num	8	BEST.		Weight (LBS)
14	Wheelbase	Num	8	BEST.		Wheelbase (IN)
15	Length	Num	8	BEST.		Length (IN)

Display 2. Contents of dataset after mapping with example 1

The main limitation of method 1 is still requiring a significant amount of programmatic preparation for each raw file and it can become too complicated and inefficient for large datasets. It can be a tedious process to do everything within the SAS environment with code and it's also difficult to collaborate with non-SAS users on A&R specifications for variable names and labels. The advantage is that all work is completed within the SAS programming window with simple code and only relies on the flexibility of PROC IMPORT to complete the mapping process for names, labels, formats, and additional attributes.

EXAMPLE 2: MAPPING WITH AN INTERMEDIATE EXCEL TEMPLATE

For the second example, the sashelp.cars dataset is used again with the same assumptions. Start by importing the unformatted raw Excel file (sas_cars.xlsx), extracting the relevant metadata, and saving it as a new Excel file (sas_cars_contents.xlsx). Example open code is provided below, though for added efficiency and repeatability these or similar steps should be written into a SAS macro:

```
** EXTRACT VARIABLE NAMES, LABELS, AND RELEVANT METADATA **;
proc import OUT      = in_xlsx
            DATAFILE = "C:\Users\Documents\sas_cars.xlsx"
            DBMS     = XLSX REPLACE ;
            RANGE    = "sas_cars$A2:O429" ;
            GETNAMES = NO ;
run ;

** ODS FOR PROC CONTENTS **;
ods output position=odsout ;
title "CONTENTS: OF INPUT XLSX FILE" ;
proc contents data=in_xlsx varnum ;
run ;
title ;

data odsout ;
      length VARIABLE_NAME COMMENTS LABEL $300 ;
      set odsout (rename=(VARIABLE = RAW_VARIABLE_NAME
                          LEN      = LENGTH)) ;
      ** CREATE BLANK COLUMNS **;
      VARIABLE_NAME = "" ;
      COMMENTS      = "" ;
      drop MEMBER NUM INFORMAT ;
run ;

** ODS OUTPUT EXCEL TEMPLATE FILE FOR MAPPING **;
ods excel file="C:\Users\Documents\sas_cars_contents.xlsx"
    options(sheet_name="sas_cars_contents");
proc print data=odsout noobs ;
      var VARIABLE_NAME LABEL TYPE LENGTH FORMAT
          RAW_VARIABLE_NAME COMMENTS ;
run ;
ods excel close ;
```

The PROC IMPORT GETNAMES option remains set to "NO" for this example. PROC CONTENTS is used to store the existing raw variable name, label (if the data has them), and other select metadata into a dataset before being saved into an external Excel file through ODS. Additional rows for documentation can be added to the output template as needed.

In this example mapping template, the COMMENTS variable is included for non-SAS users to add any additional input or to assist with the renaming and relabeling:

	A	B	C	D	E	F	G
1	VARIABLE_NAME	LABEL	Type	LENGTH	Format	RAW_VARIABLE_NAME	COMMENTS
2	Make	Make of car	Char	13	$13.	A	The
3	Model	Model of car	Char	40	$40.	B	power
4	Type	Type of car	Char	6	$6.	C	to
5	Origin	Country of origin	Char	6	$6.	D	know
6	DriveTrain	Drivetrain configuration	Char	5	$5.	E	
7	MSRP	Manufacturer's suggested retail price ($)	Num	8	BEST.	F	
8	Invoice	Invoice price ($0	Num	8	BEST.	G	
9	EngineSize	Engine size in liters	Num	8	BEST.	H	
10	Cylinders	Number of cylinders	Num	8	BEST.	I	
11	Horsepower	Metric horsepower	Num	8	BEST.	J	
12	MPG_City	Miles per gallon in the city	Num	8	BEST.	K	
13	MPG_Highway	Miles per gallon on the highway	Num	8	BEST.	L	
14	Weight	Weight in pounds (LBS)	Num	8	BEST.	M	
15	Wheelbase	Wheelbase in inches (IN)	Num	8	BEST.	N	
16	Length	Length in inches (IN)	Num	8	BEST.	O	

Display 3. Example of Excel output template for mapping variable names

The original variable name is saved in column F (Display 3) as RAW_VARIABLE_NAME; its values will default to the Excel column as letters when the GETNAMES=NO option is used during import. The two columns highlighted in yellow for VARIABLE_NAME and LABEL must still be filled in or updated manually by the user. This step is the ideal place for non-SAS users to add additional requirements in the COMMENTS field or to assist with the mapping process by defining the standard variable names and labels needed. This mapping template acts as a "define.xlsx" type of file; its functionality is like the define.xml metadata file that is already used in other areas of statistical programming. The next step is to import the completed Excel mapping template back into SAS. The raw dataset for sashelp.cars is also needed; it should already exist as work.in_xlsx if the provided example code was previously used.

A PROC IMPORT step is used to bring the mapping template into SAS, the main difference this time is that the GETNAMES=YES option is used instead of NO:

```
** IMPORT MAPPING TEMPLATE **;
proc import OUT      = in_xlsx_contents
            DATAFILE = "C:\Users\Documents\sas_cars_contents.xlsx"
            DBMS     = XLSX REPLACE ;
            GETNAMES = YES ;
run ;
```

With GETNAMES=YES, the imported example dataset will use the values of row 1 for variable names instead of the column letters. After the template is successfully imported into SAS, a macro that automatically reads in the raw dataset and the mapping dataset is needed to rename and relabel the raw variables to the defined standards. The parameters of a basic macro to handle this task includes:

Parameter Name	Description	Default Value	Required (Y/N)	Valid Values	Comments
DSRAW	Name of raw dataset	NA	Y		Example: work.in_xlsx
DSTMPLT	Name of mapping dataset	NA	Y		Example: work.in_xlsx_contents
DSRDY	Name of output dataset	NA	N		Example: work.in_xlsx_mapped
DEBUG	Prints values of generated macro variables for debugging.	Default is NO	N		Anything other than NO will print the stored values of macro variable &RENAME and &RELABEL.

Table 1. Macro parameters for map0var0names0labels.sas

The example macro below can be used to map raw variables to the standards defined in the Excel template. The main components consist of a PROC SQL step to generate the macro variables and a PROC DATASETS step to modify the raw variables in the chosen dataset to defined names and labels. The PROC SQL SELECT INTO clause creates the local macro variables &RENAME and &RELABEL by using the NLITERAL function to read in the RAW_VARIABLE_NAME and the standardized VARIABLE_NAME and the QUOTE and TRIM functions to read in the LABEL variable. The CATX function is used to add an "=" in between the RAW_VARIABLE_NAME, VARIABLE_NAME, and LABEL and completes the syntax requirements for using the PROC DATASETS MODIFY statement to finish the mapping process. For this example, the &RENAME macro variable resolves to: A=Make B=Model C=Type D=Origin E=DriveTrain F=MSRP G=Invoice H=EngineSize I=Cylinders J=Horsepower K=MPG_City L=MPG_Highway M=Weight N=Wheelbase O=Length.

It's recommended to make modifications to this example macro for specific project needs because it is quite barebones and may not cover all raw dataset variants:

```
***** MACRO TEMPLATE FOR VARIABLE NAME AND LABEL MAPPING *****;
%macro map0var0names0labels(DSRAW=
                           ,DSTMPLT=
                           ,DSRDY=
                           ,DEBUG=NO
                           ) ;
      /* DEFINE MACRO VARS */
      %local RENAME RELABEL ;
      /* CREATE DSRDY AS A COPY OF DSRAW */
      data &DSRDY ;
            set &DSRAW ;
      run ;
      /* CREATE MACRO VARS */
      proc sql noprint ;
            /* GET RENAME AND RELABEL MACRO VARS */
            Select
               catx('=',nliteral(RAW_VARIABLE_NAME),nliteral(VARIABLE_NAME))
              ,catx('=',nliteral(RAW_VARIABLE_NAME),quote(trim(LABEL)))
                  into :RENAME  separated by ' '
                      ,:RELABEL separated by ' '
            from &DSTMPLT ;
      quit ;
      /* RENAME AND RELABEL INPUT DATASET PER TEMPLATE */
      %if (&SQLOBS) %then
            %do ;
                  proc datasets nolist ;
                        modify &DSRDY ;
                        label  &RELABEL ;
                        rename &RENAME ;
                  run ;
                  quit ;
            %end ;
      /* DEBUGGING MESSAGES */
      %if %upcase(&DEBUG) ne NO %then
            %do ;
                  %put ----------------------------------;
                  %put Contents of RENAME Macro Variable: ;
                  %put ----------------------------------;
                  %put &RENAME                           ;
                  %put                                   ;
                  %put ----------------------------------;
                  %put Contents of RELABEL Macro Variable:;
                  %put ----------------------------------;
                  %put &RELABEL                          ;
                  %put ----------------------------------;
            %end ;
%mend map0var0names0labels ;
```

```
** MAPPING MACRO CALL EXAMPLE **;
%map0var0names0labels(DSRAW   = In_xlsx
                     ,DSTMPLT = In_xlsx_contents
                     ,DSRDY   = In_xlsx_mapped
                      ,DEBUG   = YES
                     ) ;
```

After the macro is called, the output dataset work.In_xlsx_mapped is created. The default A, B, C, D naming of variables in the raw file (In_xlsx) has been replaced by the user-defined standard names and labels from the template file (In_xlsx_contents). A screenshot of the output dataset's (In_xlsx_mapped) contents is presented below:

Variables in Creation Order

#	Variable	Type	Len	Format	Informat	Label
1	Make	Char	13	$13.	$13.	Make of car
2	Model	Char	40	$40.	$40.	Model of car
3	Type	Char	6	$6.	$6.	Type of car
4	Origin	Char	6	$6.	$6.	Country of origin
5	DriveTrain	Char	10	$10.	$10.	Drivetrain configuration
6	MSRP	Num	8	BEST.		Manufacturer's suggested retail price ($)
7	Invoice	Num	8	BEST.		Invoice price ($0
8	EngineSize	Num	8	BEST.		Engine size in liters
9	Cylinders	Num	8	BEST.		Number of cylinders
10	Horsepower	Num	8	BEST.		Metric horsepower
11	MPG_City	Num	8	BEST.		Miles per gallon in the city
12	MPG_Highway	Num	8	BEST.		Miles per gallon on the highway
13	Weight	Num	8	BEST.		Weight in pounds (LBS)
14	Wheelbase	Num	8	BEST.		Wheelbase in inches (IN)
15	Length	Num	8	BEST.		Length in inches (IN)

Display 4. Contents of dataset after mapping with example 2

The limitations of this method are 1) a more complicated setup and 2) requires the user to also make edits in Excel instead of just using SAS. The main benefits over example 1 are having a cleaner graphical interface to do the manual portions of the mapping work (Excel data cells versus SAS coding) and allowing non-SAS users to readily access the file, to add their input in the COMMENTS column, and to assist with the mapping process. There is no option for changing or defining variable formats with the example macro, but this feature can be easily added with additional SAS macro parameters along with any other project or organization-specific needs.

DISCUSSION

Methodology	Usage Summary
Proc Import	The first example is best suited for data with a manageable number of variables or for ad-hoc projects with significant time constraints. Its main advantage is having the flexibility of SAS data step processing within PROC IMPORT but it is limited by requiring significant manual work to rename and relabel variables using SAS code.
"Define.xlsx" Excel Mapping Template	The second example is suitable for more complex projects and allows for greater automation and easier collaboration with non-SAS users. It is limited by requiring customized macro development but the long-term benefits of being able to add more flexibility and modularity to the mapping process outweigh the cost of the initial effort invested.

Table 2. Summary of usage cases for the examples provided

The sashelp.cars example has been simplified down to just the basics to make presenting the concepts and logical flow of this paper's processes easier. The examples provided may not fully cover the specific needs of a given project because it does not delve into data quality review and data management steps. Typically, when Excel data is read in for real projects, the GETNAMES=YES option is used unless there is Unicode or other unsupported special characters in the Excel data row that contains the variable names. Both methods described in this paper will still work with the GETNAMES=YES option but the raw variables name will change from the Excel column name A,B,C,D format to the names specified in the raw data.

The first method can be used to quickly map raw data for ad-hoc project work when time constraint is the primary concern. Otherwise, it's recommended to invest the time to customize and standardize a formal process for variable name and label mapping using the method 2 macro that covers user-group or organization needs. The steps to read in the raw Excel data and the define.xlsx mapping template can be integrated into the mapping macro code and additional functionality for automatic formatting, data quality review, or deriving new variables can also be added. A master define template that contains all the raw to standard variable name and label definitions can also be created and continuously updated over multiple projects to expand the mapping process's overall coverage for additional usage cases. A consistent mapping process with multiple master define.xlsx templates that cover each major project type would be an ideal endgame goal to achieve at the user-group or organization level.

CONCLUSION

The challenges described in this paper can be traced back to the general lack of standardization within PDCS and the broader RWE work area. Standardizing variable names and labels is a good starting point for many statistical programming user-group and organization level strategic initiatives because these actions also facilitate the development of modular A&R standards and tools. Encouraging organizational and industry level adoption of the newly defined variable names, labels, and processes is the next major hurdle. The methodology described in this paper is meant to offer a couple of suggestions to help make data analysis in PDCS and RWE more consistent and repeatable on the journey towards standardization. There's no easy path forward but the unique challenges of standardizing RWE research are great opportunities for collaboration and skill-sharing within statistical programming and beyond.

ACKNOWLEDGMENTS

The authors would like to thank their real-world evidence and epidemiology colleagues at Merck & Co., Inc., Upper Gwynedd, PA, USA, for their partnership in support in this work.

CONTACT INFORMATION

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

Bo Zheng
Merck & Co. Inc.
351 N Sumneytown Pike
North Wales, PA 19454, USA
bo.zheng1@merck.com

Li Ma
Merck & Co. Inc.
351 N Sumneytown Pike
North Wales, PA 19454, USA
li.ma2@merck.com

Xingshu Zhu
Merck & Co. Inc.
351 N Sumneytown Pike
North Wales, PA 19454, USA
xingshu_zhu@merck.com

TRADEMARK