PharmaSUG 2021 - Paper QT-189

Management of Metadata and Documentation When Your Data Base Structure is Fluid: What to do if Your Data Dictionary has a Varying Number of Variables

Louise S. Hadden, Abt Associates Inc.

ABSTRACT

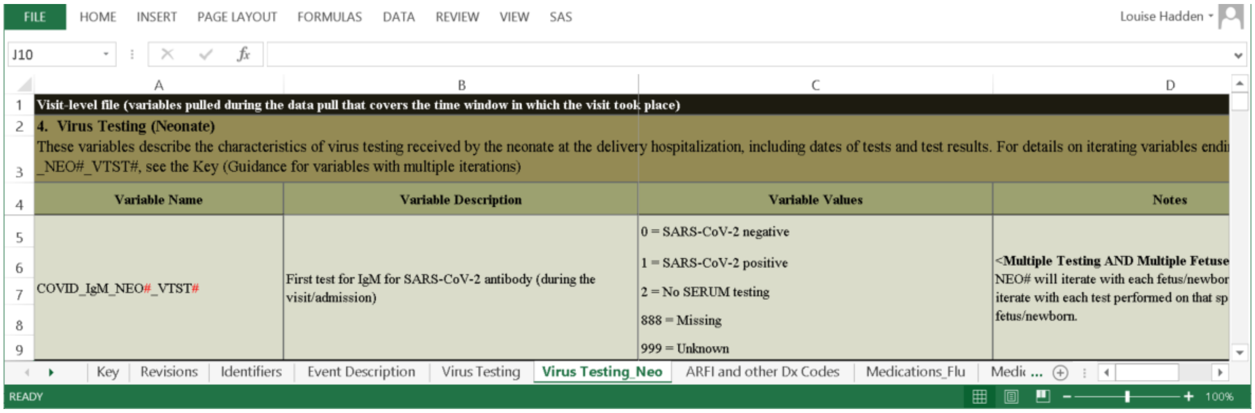
A data dictionary for a file based on Electronic Medical Records (EMR) contains variables which represent an unknown number of COVID-19 tests for an unknown number of infants – there is no way to know in advance how many iterations of the COVID test variable will exist in the actual data file from medical entities. In addition, variables in this file may exist for three different groups (pregnant women, postpartum women, and infants), with PR, PP and IN prefixes, respectively. This presentation demonstrates how to process such variables in a data dictionary to drive label (and value label) description creation for iterated (and other) labels using SAS functions, as well as other utilities.

INTRODUCTION

Documentation is a key product of any programming task. We use data dictionaries to drive much of our processing, using the metadata contained therein to drive the provision of variable and value labels, assign variable prefixes and split out subsets of files, as well as drive the creation of format assignment statements and reporting for file subsets. Data dictionaries are provided to study sites, who return extracted electronic medical record (EMR) data to us for processing.

DATA DICTIONARY READ IN

We read in the separate tabs in the workbook in the data dictionary and collect information to be used for variable names, labels and value label, as well as other information used solely for data processing. Figure 1 below shows a screenshot of entries in one of the workbook tabs.



Visit-level file (variables pulled during the data pull that covers the time window in which the visit took place)

4. Virus Testing (Neonate)

These variables describe the characteristics of virus testing received by the neonate at the delivery hospitalization, including dates of tests and test results. For details on iterating variables endi _NEO#_VTST#, see the Key (Guidance for variables with multiple iterations)

Variable Name	Variable Description	Variable Values	Notes
COVID_IgM_NEO#_VTST#	First test for IgM for SARS-CoV-2 antibody (during the visit/admission)	0 = SARS-CoV-2 negative 1 = SARS-CoV-2 positive 2 = No SERUM testing 888 = Missing 999 = Unknown	<**Multiple Testing AND Multiple Fetuse** NEO# will iterate with each fetus/newbor iterate with each test performed on that sp fetus/newborn.

Figure 1. Sample Data Dictionary

Since multiple tabs are read in with the same structure on each tab, we take advantage of macro processing to read in our file metadata. We use ranges in our PROC IMPORT, storing valuable metadata about our incoming data.

```
*************************************************************;
*** Import Personal Data Dictionary one tab at a time    ***;
*************************************************************;

%macro imptabs(tabn=1, tabnm=identifiers, intab=Identifiers, startrow=10, endcol=H);
```

```
       proc import dbms=xlsx out = temp datafile = " \file.xlsx" replace;
             RANGE="&intab.$A&startrow.:&endcol.999";
             getnames=YES;
run;

. . .

data labels&tabn.;
         length label labelstr $ 300 variable_type $ 8;
         set &tabnm (keep=variable_: pw_preg pw_pp inf
         where=(variable_name ne '' and variable_description ne ''));
             label=catx(": ","&tabnm.",variable_description);
             labelstr=cats(variable_name,'="',label,'"');

             variable_length=length(variable_name);
             length_flag=(variable_length+7 GT 32);
             label variable_length="Length of Variable"
             length_flag="Current Variable Length + 7 exceeds 32";

              /* find out the # of iterations within a variable name */
             iteration_flag=(indexc(variable_name,'#') gt 0);
             iteration_count=countc(variable_name,'#');

             label iteration_flag="Binary: Variable iterations"
             iteration_count="# of iteration points within variable name";
run;

%mend;

%imptabs(tabn=1, tabnm=Identifiers, intab=Identifiers, startrow=4, endcol=H);
```

First, the LENGTH function is used to calculate the length of “variable”. The length of variable names is limited to 32 columns, and the name an of iterated variable may exceed the limits. The data dictionary is a living document, and if any overlong variables that exist once prefixes and iterated counts are added to the base, the spelling is adjusted. Identification variables are exempt from prefixes. When a file is first processed, variables, with the exception of ID variables, have prefixes added, using the CATS function. Additionally, label strings are created by concatenating prefixes and existing variable descriptions using the CATX function.

ITERATION

It is relatively simple to replace a single iterator, in this case, a #, in a variable name. It is more complicated to replace two or more iterators, especially if you do not know how many iterations there are. SAS functions process one variable transformation at a time – that is, they stop after completing a single operation on a string. We look for # in a variable and flag using the INDEX function. The INDEX function returns the position of the first pound sign. We then use the SUBSTR function to replace the # using a do loop, outputting additional label records for each iteration.

We use the COUNTC function to discover how many #s exist in a variable name. In practice, this is done in a multidimensional array with a dimension for each # sign in a variable, with the maximum number of possible iterations. You can use functions to discover the number of iterations needed as well in the actual data – including the REVERSE and ANYNUM functions – in the actual data. Additionally, the iteration numbers are added to the label strings using CAT functions. Multiple supplemental label records are created until no more # signs appear in the variable names.

We have thousands of variables, and multiple occurrences of iteration and the need to replace (via the SUBSTR function) items of different lengths. We quickly realized we would need to employ macro loops to handle the different requirements for a number of situations (number of iterations, the “base” of the variables needing to be iterated, one or two iteration symbols, and substr length.) Sample code for a simple loop and more complex loop follow below.

Simple loop

```
%macro do_list1(maxiter=1,suffix=neo);

%do i=1 %to &maxiter;

data iter&suffix.1_&i (drop=loc);
    length variable $ 50 labelstr $ 300;
    set formats0 (where=(count(variable,"#")=1 and
index(variable,"IDENTIFIER#")>0));

    *get the first indexed # location;
    loc=index(variable,"#");

    substr(variable,loc,1)="&i";
    labelstr=catt(labelstr," #&i");

run;

proc print data=iter&suffix.1_&i (obs=5) noobs;
    var variable labelstr;
run;

%END;

%MEND DO_LIST1;
```

Complex loop

```
%macro do_list2(maxiter=20,suffix=vtst);

%if &maxiter le 9 %then %do i=1 %to &maxiter;

data iter&suffix.1_&i (drop=loc);
    length variable $ 50 labelstr $ 300;
    set formats0 (where=(count(variable,"#")=1 and
index(variable,"VTST#")>0));

    *get the first indexed # location;
    loc=index(variable,"#");

    substr(variable,loc,1)="&i";
    labelstr=catt(labelstr," #&i");

run;

proc print data=iter&suffix.1_&i (obs=5) noobs;
    var variable labelstr;
run;

%END;

%if &maxiter gt 9 %then %do;

%do i=1 %to 9;
```

```
data iter&suffix.1_&i (drop=loc);
    length variable $ 50 labelstr $ 300;
    set formats0 (where=(count(variable,"#")=1 and
index(variable,"VTST#")>0));

    *get the first indexed # location;
    loc=index(variable,"#");

    substr(variable,loc,1)="&i";
    labelstr=catt(labelstr," #&i");

run;

proc print data=iter&suffix.1_&i (obs=5) noobs;
    var variable labelstr;
run;

%END;

%do i=10 %to &maxiter;

data iter&suffix.1_&i (drop=loc);
    length variable $ 50 labelstr $ 300;
    set formats0 (where=(count(variable,"#")=1 and
index(variable,"VTST#")>0));

    *get the first indexed # location;
    loc=index(variable,"#");

    substr(variable,loc,2)="&i";
    labelstr=catt(labelstr," #&i");

run;

proc print data=iter&suffix.1_&i (obs=5) noobs;
    var variable labelstr;
run;

%END;

%END;

%MEND DO_LIST2;
```

PRACTICAL APPLICATIONS

The iteration techniques discussed above are employed in several different scenarios: data quality checks, creating variable labels, creating format assignment statements, driving range checks, and producing missingness reports. Below follow snippets of code to create a data driven variable label statement.

Assign a filename for the label statement:

```
  filename label1 ".\&short._Labels.txt";
```

Create iterations of variables with # signs using macro loops described above:

```
%do_list1(maxiter=3,suffix=id);
%do_list2(maxiter=4,suffix=vtst); . . .
```

Add iterated records created by the do loops together:

```
data expand_labels;
    set iterid: itervtst: . . ._ ;
run;
```

Add iterated records to the records that did not require iteration:

```
data labels;
    length variable $ 32;
    set labels0 (where=(index(variable,"#")=0))
        expand_labels (where=(index(variable,"#")=0))
        ;
run;
```

Output the label statement:

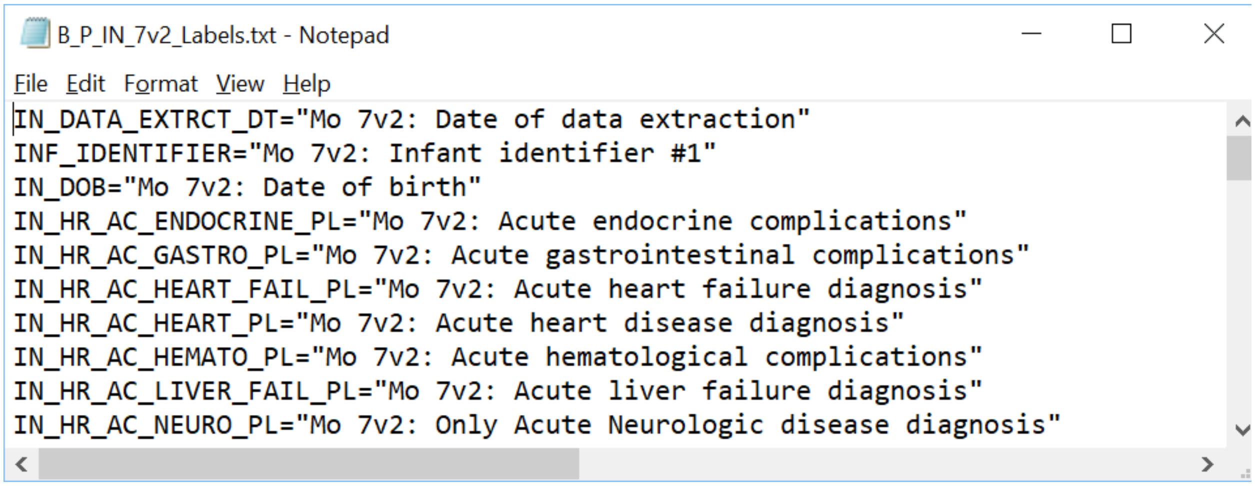
```
data tolabel;
    retain VARIABLE_CATEGORY VARIABLE LABELSTR
           VARIABLE_TYPE VARIABLE_LENGTH
           PW_PREG PW_PP INF ITERATION_COUNT INLABELS INPOS NUM ;
    file label1 lrecl=400;
    set matchtest ( keep= VARIABLE_CATEGORY VARIABLE LABELSTR
                          VARIABLE_TYPE
                          VARIABLE_LENGTH PW_PREG PW_PP INF
                          PRIORITY_VARIABLE
                          MISSING_NOT_OK ITERATION_COUNT
                          INLABELS INPOS NUM DD_ORDER);
    by NUM;
    STATEMENT=compbl(cats(variable,'="',labelstr,'"'));
    if inlabels=1 and inpos=1 then put statement;
run;
```

Include the label statement:

```
filename label1 ".\&short._Labels.txt";
filename retain1 ".\&short._retain.txt";
run;

data &outfi. (label="Labeled &short");
    retain
        %include retain1;
    ;
    set &infi.;
    label
        %include label1;
    ;
run;
```

Figure 2 is a snippet of the text file included to produce variable labels.



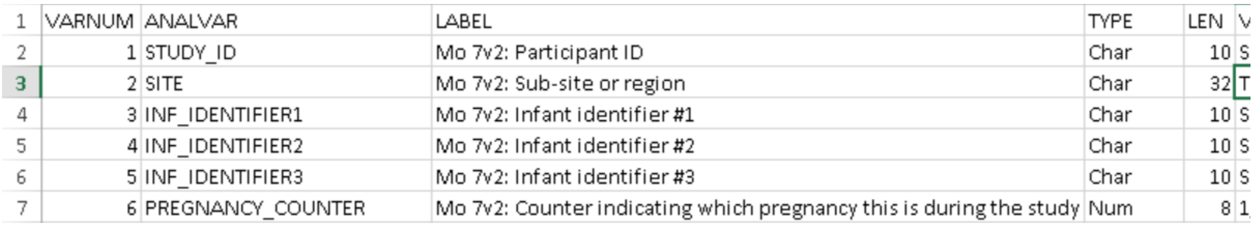
```
IN_DATA_EXTRCT_DT="Mo 7v2: Date of data extraction"
INF_IDENTIFIER="Mo 7v2: Infant identifier #1"
IN_DOB="Mo 7v2: Date of birth"
IN_HR_AC_ENDOCRINE_PL="Mo 7v2: Acute endocrine complications"
IN_HR_AC_GASTRO_PL="Mo 7v2: Acute gastrointestinal complications"
IN_HR_AC_HEART_FAIL_PL="Mo 7v2: Acute heart failure diagnosis"
IN_HR_AC_HEART_PL="Mo 7v2: Acute heart disease diagnosis"
IN_HR_AC_HEMATO_PL="Mo 7v2: Acute hematological complications"
IN_HR_AC_LIVER_FAIL_PL="Mo 7v2: Acute liver failure diagnosis"
IN_HR_AC_NEURO_PL="Mo 7v2: Only Acute Neurologic disease diagnosis"
```

Figure 2. Sample Label Statement Text File

CONCLUSION

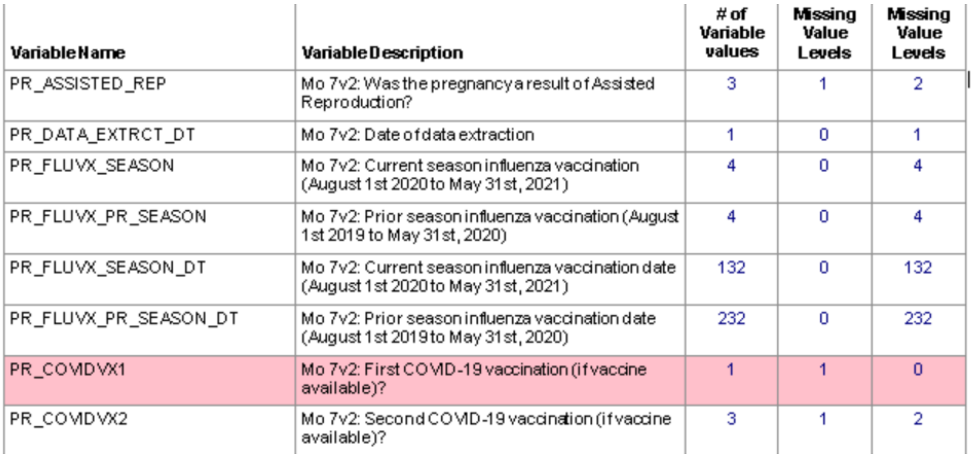
The same process of iteration and concatenation based on metadata elements is used to create macro calls to create a codebook, a range report and a "missingness" report. We hope you'll have some fun iterations with functions with your metadata as well!

Figure 3 shows a range check report, and Figure 4 shows a missingness report, all products of the processes shown above.



1	VARNUM	ANALVAR	LABEL	TYPE	LEN	V
2	1	STUDY_ID	Mo 7v2: Participant ID	Char	10	S
3	2	SITE	Mo 7v2: Sub-site or region	Char	32	T
4	3	INF_IDENTIFIER1	Mo 7v2: Infant identifier #1	Char	10	S
5	4	INF_IDENTIFIER2	Mo 7v2: Infant identifier #2	Char	10	S
6	5	INF_IDENTIFIER3	Mo 7v2: Infant identifier #3	Char	10	S
7	6	PREGNANCY_COUNTER	Mo 7v2: Counter indicating which pregnancy this is during the study	Num	8	1

Figure 3. Sample Range Check Report



Variable Name	Variable Description	# of Variable values	Missing Value Levels	Missing Value Levels
PR_ASSISTED_REP	Mo 7v2: Was the pregnancy a result of Assisted Reproduction?	3	1	2
PR_DATA_EXTRCT_DT	Mo 7v2: Date of data extraction	1	0	1
PR_FLUVX_SEASON	Mo 7v2: Current season influenza vaccination (August 1st 2020 to May 31st, 2021)	4	0	4
PR_FLUVX_PR_SEASON	Mo 7v2: Prior season influenza vaccination (August 1st 2019 to May 31st, 2020)	4	0	4
PR_FLUVX_SEASON_DT	Mo 7v2: Current season influenza vaccination date (August 1st 2020 to May 31st, 2021)	132	0	132
PR_FLUVX_PR_SEASON_DT	Mo 7v2: Prior season influenza vaccination date (August 1st 2019 to May 31st, 2020)	232	0	232
PR_COVIDVX1	Mo 7v2: First COVID-19 vaccination (if vaccine available)?	1	1	0
PR_COVIDVX2	Mo 7v2: Second COVID-19 vaccination (if vaccine available)?	3	1	2

Figure 4. Sample Missingness Report

ACKNOWLEDGEMENTS

This type of complex programming and processing is a team sport. I could not have created and implemented these techniques on my own. A very heartfelt thank you to team members Mary Juergens, Jenna Spirt, Nickolas Ferguson, Michael Duckworth and Peiyi Zhang.

CONTACT INFORMATION

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

Louise S. Hadden
Abt Associates Inc.
Louise_hadden@abtassoc.com