MWSUG 2016 - Paper PH01

Pre-Data Checks for SDTM Development

Abhinav Srivastva, PaxVax Inc., Redwood City, CA

ABSTRACT

In clinical trials SDTM development is a critical step in transforming raw data coming from CRFs into a standard format for FDA submission and building the foundation for downstream analysis and reporting (ADaM, TLFs, Define.xml etc). OpenCDISC (or Pinnacle 21) is a common tool used to validate SDTM datasets, but not enough effort is dedicated towards pre-processing the data coming from CRFs. The paper is targeted towards identifying basic data integrity checks that can be done at an early stage to get a full awareness of the data quality before proceeding with SDTM development. These checks are driven by CDISC compliance standards and are meant as pointers for SDTM programmers to initiate the investigation by raising a query with cross functional teams including Clinical data management, Vendors and Site representatives. Doing early quality checks on raw data provides an in-depth overview of how the final SDTM datasets will perform against the CDISC compliance tests. Additionally, it provides an extra layer of edit checks from those which may already be in place within the Electronic Data Capture (EDC) system used for the Clinical study like Medidata RAVE, InForm etc.

INTRODUCTION

Quality of data is the basis of accurate analysis and reporting, and making informed decisions. In Clinical trials, data is used to assess the safety and efficacy of drug or biologics. The checks defined in the paper are basic data integrity checks which can be expanded as per study design and protocol. The checks are grouped into four main categories as below and will be discussed in great detail in following sections.

- Metadata-level checks
- Value-level checks
- Logical checks
- Cross-reference checks

CASE STUDY

The data collected in CRFs is mapped into SDTM domains as per CDISC guidelines. Below Figure 1 and Table 1 is an example showing the mapping process.

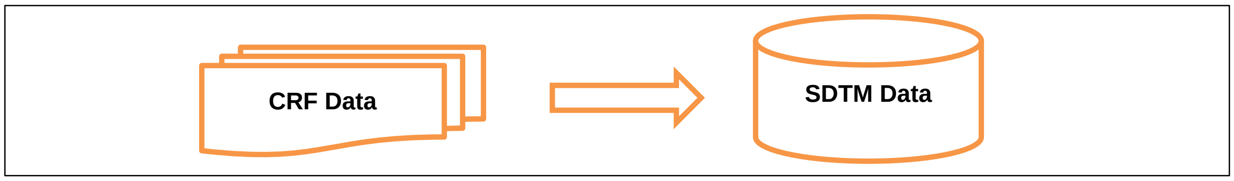


Figure 1: Data flow

CRF data	Variables	Mapped To SDTM Domain	SDTM Domain Label
demog	SubjectID, Age, Race…<more vars>	DM	Demographics
visit	SubjectID, Visit, Visitnum…<more vars>	SV	Subject Visits
ae	SubjectID, Aeterm, Aestdat, Aeendat...<more vars>	AE	Adverse Events
dose	SubjectID, Dose, Exstdat, Exendat…<more vars>	EX	Exposure
lab	SubjectID, Test_name, Result, Lbdat…<more vars>	LB	Laboratory Test Results

Table 1: SDTM Mapping

METADATA-LEVEL CHECKS

At metadata-level these checks can be performed to assess the raw CRF data that will be transformed into SDTM data. Caution should be taken especially if the variables are directly copied into SDTM datasets as they may not comply with the FDA submission guidelines for SAS® transport files.

- Length of any variable name >8 characters
- Length of any variable label >40 characters
- Length of any variable value >200 characters
- Variables types other than character or numeric
- Presence of special characters in variable names

The length of any variable in SDTM datasets cannot exceed 200 characters. The only exceptions are Comments (CO) and Trial Summary (TS) domains where if length > 200 characters then they are spilt into multiple variables with relevant suffix as COVAL, COVAL1, COVAL2 and so on for long text of Comments. All variable types must be ‘character’ or ‘numeric’ only. Additionally, variable names cannot contain special characters. Variable names are limited to alphabets, numbers and underscores. Please refer to SDTM Implementation Guide v3.2 (see REFERENCES section) for further details.

SAS I/O functions OPEN, CLOSE, VARNAME, VARNUM, VARLABEL, VARTYPE and VARLEN can be used to read the metadata from a dataset.

```
<more lines of code>
...
%let dsid  = %sysfunc(open(&dsn.));

%let nvars = %sysfunc(attrn(&dsid.,nvars));

%do i=1 %to &nvars.;

    %let varname&i = %sysfunc(varname(&dsid.,&i.));
    %let varnum&i  = %sysfunc(varnum(&dsid.,&&varname&i.));
    %let varlabel&i= %sysfunc(varlabel(&dsid.,&i.));
    %let vartype&i = %sysfunc(vartype(&dsid.,&i.));
    %let varlen&i  = %sysfunc(varlen(&dsid.,&i.));

%end;

%let rc  = %sysfunc(close(&dsid.));
....
<more lines of code>
```

To perform the length restriction check LENGTHN function can be used. For special character check in variable names a PERL regular expression is used.

```
if lengthn(VAR_NAME)>8 then flag = 'Variable Name >8 chars';
if prxmatch("/\W/",strip(VAR_NAME))>0 then
   flag = 'Variable Name has special chars'; * Note the wildcard character
                                               '\W' in PRXMATCH;
```

Please refer to APPENDIX for the complete SAS program.

VALUE-LEVEL CHECKS

In value-level check a few critical variable values can be examined and investigated further before developing SDTM:

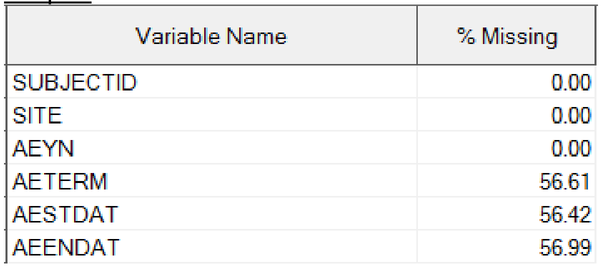
- Missing values check
- Variables containing special characters

Missing values are prohibited in Identifier variables in SDTM like Subject ID, Site ID etc. Also, variables of interest (TOPIC) cannot contain missing values like Reported Term for the Adverse Event (AETERM) in the Adverse Events dataset. Variables which are completely missing tend to be PERMISSIBLE in SDTM but partial missing will likely get flagged as a warning for EXPECTED or REQUIRED SDTM variables.

The FREQ procedure provides an easy solution to check for missing values in a variable.

```
proc format;
  value num_f
      .     = '0' low-high = '1' ;
  value $char_f
      ' '   = '0'    other = '1' ;
run;

* One-way Frequency table ;
ods listing close;
ods output onewayfreqs=tables (keep = table frequency percent);

proc freq data= <dataset>;
  tables _all_ / missing;
  format _numeric_ num_f. _character_ $char_f.;
run;

ods output close;
ods listing;

* Format the dataset from ODS to produce a report ;
data Missing_report (Keep = VARIABLE M_PERCENT);
  length VARIABLE $100;
  set tables;
    by table notsorted;
       VARIABLE=scan(Table,2,' '); * get variable name ;

         if 'F_'||strip(VARIABLE) = '1' and percent=100 then M_PERCENT=0;
                                                  * calculate missing % ;
            else M_PERCENT=percent;

         if first.table then output;

  attrib M_PERCENT label='% Missing' format=6.2
         VARIABLE  label='Variable Name';
run;
```

Output:



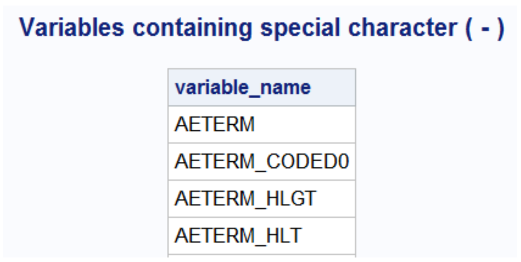
Variable Name	% Missing
SUBJECTID	0.00
SITE	0.00
AEYN	0.00
AETERM	56.61
AESTDAT	56.42
AEENDAT	56.99

Display 1: Missing value Report

Non-Printable and Special Characters (NPSC) are often not desired and they can be easily identified using a number of SAS functions like INDEXC, FINDC or PRX. Below code uses PRXMATCH to identify a special character ‘**-**‘ in the data. For more details please see REFERENCES section.

```
options noquotelenmax;
%let sp = -; * special character '-' to be identified ;

data char_check (drop=__:);
     set <dataset>;
     array char_arr{*} _character_; * look only at character variables ;
     array __varnam{100} $100;
      do i=1 to dim(char_arr);
          if prxmatch("/\&sp./",char_arr{i})>0 then
          do;
        __varnam{i} = strip(vname(char_arr{i})); * ‘vname’ will get the
                                                    variable name ;
         end;
      end;
      varlist=catx('--',of __varnam:);
run;

proc sql noprint;
    select distinct(varlist) into:varnames separated by '--' /* any
                                                            delimiter */
      from char_check;
quit;

data special_char_check;
   length variable_name $100;
    i=1;
      do while(scan("&varnames.",i,'--') ne ' ');
        variable_name=scan("&varnames.",i,'--'); * get variable names ;
        output;
        i=i+1;
     end;
    drop i;
run;

proc sort nodupkey; by variable_name; run;

proc print noobs;
   title "Variables containing special character ( &sp. )";
run;
```

Output:


Variables containing special character (-)

variable_name
AETERM
AETERM_CODED0
AETERM_HLGT
AETERM_HLT

Display 2: Special Character Summary

Another edit check can be established when certain variables are expected to be in a specific format only. Studies typically assign Subject ID with a fixed pattern like ABCDE-12345, ABC-12-3456 etc. PRX function can easily identify any non-conformity to this pattern as below:

```
data pattern_check;
     set <dataset>;
     if _n_ = 1 then pattern=prxparse("/[a-zA-Z]{5}-\d{5}/"); * will match
                                                            ABCDE-12345;

     retain pattern;
     if prxmatch(pattern,SubjectID)= 0 then output; * output if no match;
run;
```

LOGICAL CHECKS

Several logical checks can be performed for data integrity. This list can be expanded based on the study protocol requirements.

- Flag Start date greater than End date
- Flag exact duplicate records
- Flag duplicates in Demographic data
- Flag inconsistent pairs of values
- Flag any other inconsistency across a group of variables

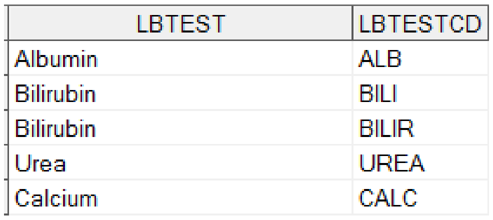
The FREQ procedure comes handy in combination with a data step to flag erroneous dates entered in the database.

```
<more lines of code>
..
if nmiss(start_dt,end_dt)=0 and start_dt > end_dt then
           DT_FLAG='Start Dt. > End Dt.';
   else    DT_FLAG='Start Dt. <= End Dt.';
..

proc freq data=<dataset>;
   table DT_FLAG / out=date_error;
run;
```

Some duplicates are worth reviewing before discarding them. The SORT procedure NODUPRECS option along with DUPOUT=<dataset> is very useful in that aspect. Demographic data is expected to contain one record only per subject. If there are more than one record per subject then it can be examined and queried accordingly.

```
* Check for exact duplicates ;
proc sort data = <dataset> out=<out> noduprecs dupout=<dup_dataset>;
  by _all_;
run;

* Check for duplicates in Demographic dataset ;
proc freq data= <dataset> noprint;
  table SubjectID / out=<out>(where=(count>1) drop=percent);
run;
```

Some pair of variables are expected to have a 1-1 relationship between their values and any inconsistency between them should be investigated. Consider a sample case (Display 3) where multiple Lab Test codes (LBTESTCD) exist for the same Lab Test Name (LBTEST = ‘Bilirubin’).

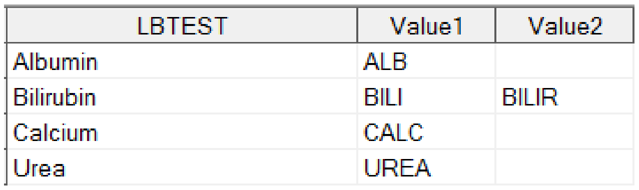


LBTEST	LBTESTCD
Albumin	ALB
Bilirubin	BILI
Bilirubin	BILIR
Urea	UREA
Calcium	CALC

Display 3: Sample Data

```
  proc sort data = <dataset> nodupkey;
     by lbtest lbtestcd;
  run;

  data check;
    set <dataset>;
      by lbtest lbtestcd;
        if first.lbtest then i=1;
           else i+1; * more than one value ;
  run;

  * Transpose for review ;
  proc transpose data = check (drop=_NAME_) prefix=Value;
    by lbtest;
    var lbtestcd;
    id i;
 run;
```

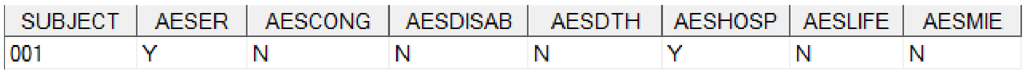
<u>Output:</u>



LBTEST	Value1	Value2
Albumin	ALB	
Bilirubin	BILI	BILIR
Calcium	CALC	
Urea	UREA	

Display 4: Inconsistent value Report

There are some specific consistency check performed by OpenCDISC (or Pinnacle 21). One such check is for an adverse event dataset when an event is classified as ‘Serious Adverse Event (SAE)’ and additional variables need to be flagged in support of the serious adverse event. In the below example (Display 5), Subject – 001 reported a serious adverse event (AESER=Y) which requires atleast one of the associated variables (AESCONG – AESMIE) to be flagged as ‘Y’ or vice-versa. Here AESHOSP (Hospitalized) is flagged ‘Y’ which seems consistent with the serious adverse event (AESER=Y).



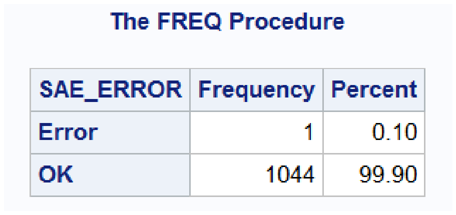
SUBJECT	AESER	AESCONG	AESDISAB	AESDTH	AESHOSP	AESLIFE	AESMIE
001	Y	N	N	N	Y	N	N

Display 5: SAE Data

This type of consistency check can be implemented through the use of ARRAYS as demonstrated below:

```
data sae_check (drop=i);
   set <dataset>;
     array sae{*} aescong aesdisab aesdth aeshosp aeslife aesmie;

       do i=1 to dim(sae);
          if sae{i} = 'Y' then flag=1;
       end;

        if (aeser='Y' and flag ne 1) or (flag=1 and aeser ne 'Y') then
             SAE_ERROR='Error';
        else SAE_ERROR='OK';

run;

proc freq data=sae_check;
  table sae_error / out=sae_error;
run;
```

Output:



The FREQ Procedure

SAE_ERROR	Frequency	Percent
Error	1	0.10
OK	1044	99.90

Display 6: SAE Inconsistency Summary

CROSS-REFERENCE CHECKS

Cross-reference checks include comparing consistency across datasets. In SDTM, two of the basic checks are:

- Subject ID needs to be consistent between Demographic (DM) and all other datasets
- Subject visits need to be consistent between Visit (SV) and relevant visit-based datasets

To illustrate this scenario let's consider some CRF datasets as below. (Also refer Table 1: SDTM Mapping)

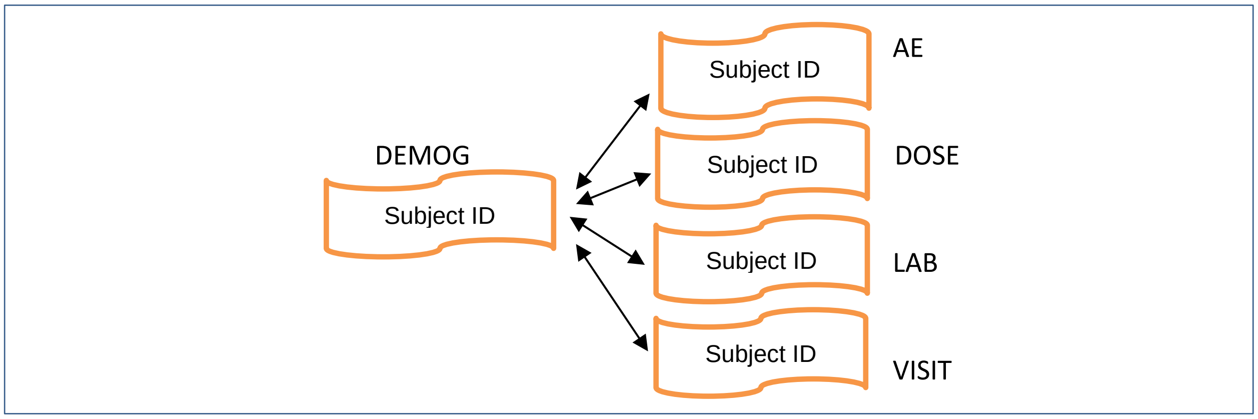


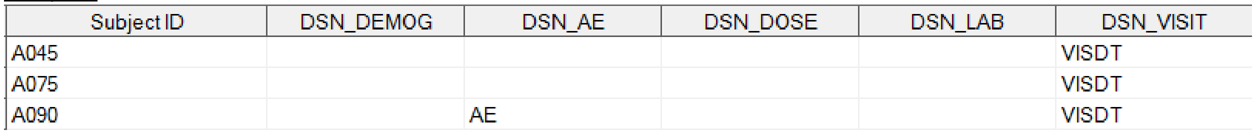
Figure 2: Demographic Subject IDs linked with Non-Demographic Subject IDs

All Subject IDs in various forms like AE, DOSE etc need to be consistent with the Demographic dataset. For example, if a Subject with ID=’A01’ is recorded in AE (adverse event) but the subject is not present in demographics it should trigger a query of this unknown subject missing in Demographic but present in AE. A similar check can be designed for consistent Subject ID and Visit.

Data step MERGE along with the SQL procedure can be used to implement this check as below:

```
libname RAW '</location/>' ; * library where all datasets are located ;

proc sql;
  * Get a list of all datasets ;
  create table datasets as
    select memname from dictionary.members
    where libname='RAW';

  * Store all members (except DEMOG) in a macro variable ;
    select strip(memname) into:merge_var separated by ' '
     from datasets
     where memname not in ('DEMOG');
quit;

* Get Unique Subject IDs from all datasets ;
data _null_;
 set datasets;
 call execute('proc sql;');
 call execute('create table '|| strip(memname) || ' as');
 call execute('select distinct SubjectID from '||'RAW.'||strip(memname)
              ||' ;');
 call execute('quit;');

 call execute('data '||strip(memname)||';');
 call execute('set '||strip(memname)||';');
 call execute('length DSN_'||strip(memname)||' $10 ;');
 call execute('DSN_'||strip(memname)||'='||"'"||strip(memname)||"'"||';');
 call execute('run;');
run;

* Combine all Non-Demographic datasets ;
data non_demog;
  merge &merge_var.;
    by SubjectID;
run;

* Compare Subject ID in Demographic with All Non-Demographic datasets ;
data Subject_check;
   merge demog(in=a) non_demog(in=b);
      by SubjectID;
        if b and not a then do; * Check: Missing in DEMOG but present in
                                           Non-DEMOG ;
         output;
        end;
run;
```

Output:



Subject ID	DSN_DEMOG	DSN_AE	DSN_DOSE	DSN_LAB	DSN_VISIT
A045					VISDT
A075					VISDT
A090		AE			VISDT

Display 7: Subject ID Inconsistency Report

From the above Output (Display 7), Subjects A045, A075, A090 are missing in Demographic but present in Visit dataset. Similarly, Subject A090 is missing in Demographic but present in Adverse Event dataset. All these anomalous records need to be queried for clarification.

CONCLUSION

The data checks that are described in this paper can be used as a first step in reviewing clinical data. In addition several custom checks can be established based on the study design and protocol. Though these checks are especially designed as a precursor to SDTM development, they can also be used in many edit checks to assess the data quality and raise queries as needed. Attacking the data issues at an early stage will result in cleaner and more compliant SDTM datasets for downstream analysis and reporting.

REFERENCES

Cody, Ron. 2004. “An Introduction to Perl Regular Expressions in SAS 9” SUGI 29. Paper 265-29.

Hortsman, Joshua M & Muller, Roger D. 2011. “Dealing with Duplicates in Your Data”. MWSUG 2011 - Paper S111.

Zdeb, Mike. SAS Presentation “An Easy Route to a Missing Data Report with ODS+PROC FREQ+A Data Step”.

CDISC, SDTM Implementation Guide v3.2: Human Clinical Trials, 2013-11-26. Prepared by: CDISC Submission Data Standards Team.

Dodlapati, Sridhar & Lakkaraju, Praveen & Tulluru, Naresh & Zeng, Zemin. “Non Printable & Special Characters: Problems and how to overcome them” NESUG 2010.

SAS Support, SAS(R) 9.2 Language Reference: Dictionary, Fourth Edition, “Function and Call Routines”. http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000245852.htm

ACKNOWLEDGMENTS

I would like to thank Michelle Pallas (PaxVax Inc.), Trudi White (PaxVax, Inc.) and Frank Liu (PaxVax, Inc.) for the review and feedback of the paper.

CONTACT INFORMATION

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

Name: Abhinav Srivastva
Enterprise: PaxVax, Inc.
E-mail: asrivastva@paxvax.com

APPENDIX

```
/* Perform Metadata Check */
%macro get_attrs (dsn=);

%let dsid  = %sysfunc(open(&dsn.));
%let crdate= %sysfunc(attrn(&dsid.,modte));
%if  &crdate.>0 %then %let crdate=%qsysfunc(putn(&crdate.,dateampm17.0));
                 %else %let crdate = UNK;
%let nvars = %sysfunc(attrn(&dsid.,nvars));
%let nobs  = %sysfunc(attrn(&dsid.,nobs));
%if  &nobs. ne -1 %then %let nobs=%sysfunc(putn(&nobs.,comma20.));
              %else %let nobs=UNK;

 %do i=1 %to &nvars.;
   %let varname&i =%sysfunc(varname(&dsid.,&i.));
   %let varnum&i  =%sysfunc(varnum(&dsid.,&&varname&i.));
   %let varlabel&i=%sysfunc(varlabel(&dsid.,&i.));
   %let vartype&i =%sysfunc(vartype(&dsid.,&i.));
   %let varlen&i  =%sysfunc(varlen(&dsid.,&i.));

 %end;
 %let rc  =%sysfunc(close(&dsid.));

data report (drop = j);
  length DSN_NAME DSN_OBS DSN_MODDT $20 VAR_NAME $100 VAR_TYPE $20
         VAR_LABEL VAR_LEN $1000;
  DSN_NAME = "&dsn.";
  DSN_OBS  = "&nobs.";
  DSN_MODDT= "&crdate.";

  do j=1 to &nvars.;
       VAR_NAME = symget('varname'||strip(put(j,best.)));
       VAR_TYPE = symget('vartype'||strip(put(j,best.)));
       VAR_LABEL= symget('varlabel'||strip(put(j,best.)));
       VAR_LEN  = symget('varlen'||strip(put(j,best.)));

       if lengthn(VAR_NAME)>8 or prxmatch("/\W/",strip(VAR_NAME))>0 then
             COMMENT1='ALERT: Variable Name (>8 chars) or special chars';
           else COMMENT1=' ';
       if VAR_TYPE NOT IN ('N', 'C') then COMMENT2='ALERT: Variable Type
                                                  (N or C only)';
           else COMMENT2=' ';
       if lengthn(VAR_LABEL)>40 then COMMENT3='ALERT: Variable Label (>40
                                              chars)';
           else COMMENT3=' ';
       if lengthn(VAR_LEN)>200 then COMMENT4='ALERT: Variable Length
                                                   (>200 chars)';
           else COMMENT4=' ';
       output;
    end;
run;

%mend;

%get_attrs(dsn=AE);
```