PharmaSUG2014 – Paper AD13

Let SAS© Improve Your CDISC Data Quality

Wayne Zhong, Accenture, Berwyn, PA

ABSTRACT

How often is “but this data passed OpenCDISC” used as a defense for poor quality data? Adoption of data standards benefits reviewers and analysts by keeping data structures consistent. Ensuring data integrity however remains a manual labor with edit checks or eyeballing. This paper presents a data quality control application for CDISC data: using the metadata of CDISC datasets to trigger SAS code modules, running relevant checks for each dataset and presenting issues for review. This tool can be run with the availability of any CDISC data, not waiting for late statistical analysis to report strange results and later investigations to identify data issues, making this application an important component of a good CDISC data creation process.

INTRODUCTION

Data collected from a clinical study is organized and presented using the Study Data Tabulation Model (SDTM), a data structure standard created by the Clinical Data Interchange Standards Consortium (CDISC). Clinical data reviewers and analysts benefit from SDTM because it allows different companies to present their data the same way. Analysis Data Model (ADaM) datasets, which support the safety and efficacy analysis of clinical studies, are created from SDTM data. ADaM data is used to produce tables, listings, and figures, which together with SDTM and ADaM datasets are submitted to regulatory bodies, such as the FDA, for use in evaluating new clinical treatments.

To ensure the quality of CDISC datasets, a free application called OpenCDISC Validator is typically used to generate a report. This report focuses on the implementation side of CDISC standards, such as whether the dataset contains SDTM compliant variable names, types, labels, and code lists. It will not check laboratory results for outliers, adverse events for miscoded terms, or sites for reporting in the wrong unit. For these and other checks, detection still requires manual effort unless we automate it, which we will do using SAS. This paper assumes basic knowledge of CDISC dataset structure standards, SAS programming, and SAS macros.

FRAMEWORK

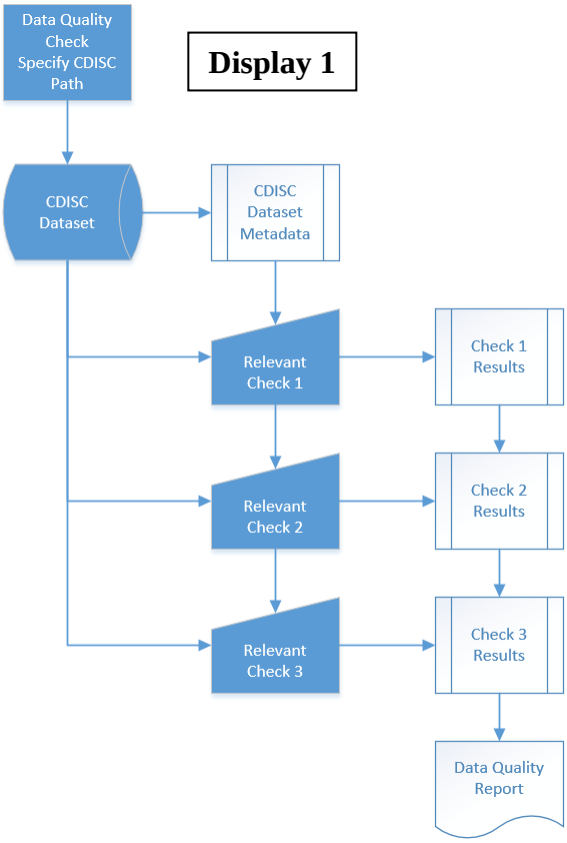
The requirements of a basic data quality assurance application are:

1. A good library of checks
2. Data driven check execution
3. Informatively presented issue details

The flow chart in Display 1 summarizes the design of the application. When a data quality check is requested, the provided path locates the CDISC datasets. Metadata is extracted from them, and used to identify and execute relevant checks from the library. Results from the checks are summarized in the final data quality report.

The framework of this paper is organized into sections for extracting metadata, using the metadata to execute checks, creating checks, and displaying results. Examples will use SDTM throughout this paper, but there are no differences compared with approaching ADaM datasets.

Display 1



METADATA

The metadata of CDISC datasets will be used to select the checks to execute for each dataset. SAS provides a variety

of methods to retrieve dataset metadata, the code below uses SASHELP. The metadata is processed into a form that is easy to work with, which will be demonstrated in the next section.

```
libname sdtmlib 'x:\path to sdtm datasets\';

data meta1(keep=memname name) meta2(keep=memname allvars);
    set sashelp.vcolumn;
    where libname='SDTMLIB' & memtype='DATA';
    by memname;
    if type='num' then name=cats(name,'#');
    else name=cats(name,'$');

    output meta1;

    length allvars $10000;
    retain allvars;
    if first.memname then allvars=' '||name;
    else allvars=trim(allvars)||' '||name;

    if last.memname then output meta2;
run;

data meta3;
    merge meta1 meta2;
    by memname;
run;
```

The first DATA STEP processes the metadata source, SASHELP.VCOLUMN, into datasets META1 and META2. META1 contains a list of variables names, with variable META1.NAME adding a $ suffix to character variable names and # suffix to numeric variable names. META2 contains one record per dataset, with variable META2.ALLVARS concatenating all variables names in the dataset together.

The second DATA STEP creates dataset META3 by merging META1 and META2, a sample is shown in Display 2. The next section will use this to demonstrate the conditional execution of checks.

Display 2

MEMNAME	NAME	ALLVARS
AE	AESTDTC$	STUDYID$ DOMAIN$ USUBJID$ AESEQ# AEGRPID$ AEREFID$ AETERM$ AEMODIFY$ AEDECOD$ AEBODSYS$ AESER$ AEACN$ AEREL$ AEPATT$ AEOUT$ AESDISAB$ AESDTH$ AESHOSP$ AESLIFE$ AESMIE$ AETOXGR$ AESTDTC$ AEENDTC$ AESTDY# AEENDY#
AE	AEENDTC$	STUDYID$ DOMAIN$ USUBJID$ AESEQ# AEGRPID$ AEREFID$ AETERM$ AEMODIFY$ AEDECOD$ AEBODSYS$ AESER$ AEACN$ AEREL$ AEPATT$ AEOUT$ AESDISAB$ AESDTH$ AESHOSP$ AESLIFE$ AESMIE$ AETOXGR$ AESTDTC$ AEENDTC$ AESTDY# AEENDY#
AE	AESTDY#	STUDYID$ DOMAIN$ USUBJID$ AESEQ# AEGRPID$ AEREFID$ AETERM$ AEMODIFY$ AEDECOD$ AEBODSYS$ AESER$ AEACN$ AEREL$ AEPATT$ AEOUT$ AESDISAB$ AESDTH$ AESHOSP$ AESLIFE$ AESMIE$ AETOXGR$ AESTDTC$ AEENDTC$ AESTDY# AEENDY#
AE	AEENDY#	STUDYID$ DOMAIN$ USUBJID$ AESEQ# AEGRPID$ AEREFID$ AETERM$ AEMODIFY$ AEDECOD$ AEBODSYS$ AESER$ AEACN$ AEREL$ AEPATT$ AEOUT$ AESDISAB$ AESDTH$ AESHOSP$ AESLIFE$ AESMIE$ AETOXGR$ AESTDTC$ AEENDTC$ AESTDY# AEENDY#
CM	STUDYID$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	DOMAIN$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	USUBJID$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	CMSEQ#	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	CMGRPID$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	CMTRT$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$

DATA DRIVEN

Each data quality check is designed for specific datasets or variables, so it makes sense to let the data decide if a check should run. Below is an example using the metadata (as presented in Display 2) to execute checks in a data driven application format. Take macro **%subset**:

```
%macro subset(data=, var1=, var2=, condition= );
    data finding;
        set &data(keep=&var1 &var2 rename=(&var1=v1 &var2=v2));
        if &condition;
    run;
%mend subset;
```

This macro performs one DATA STEP, creating dataset FINDING. Depending on the input parameters to this macro, this dataset will either be empty or non-empty. The following _NULL_ DATA STEP takes the metadata contained in META3 and conditionally uses CALL EXECUTE to generate lines of codes:

```
data _null_;
    set meta3;
    if index(name,'STDY#') and index(allvars,' '||tranwrd(strip(name),'STDY#','ENDY#'))
        then call execute(
            cats(
            '%subset(data=',memname,
                  ', var1=',compress(name,'#'),
                  ', var2=',tranwrd(name,'STDY#','ENDY'),
                  ', condition= v1>v2>.z );'
            )
        );
run;
```

The IF statement first searches variable META3.NAME for the value ‘STDY#’, the # sign helpfully marks the last character of the variable name and identifies the variable type. If found, another search is made in variable META3.ALLVARS for the value created from replacing ‘STDY#’ with ‘ENDY#’ in META3.NAME. If both searches succeed, the dataset and variables names are placed into a CALL EXECUTE statement. CALL EXECUTE creates lines of code that SAS will subsequently run, and after processing META3 the following lines of code are generated and executed:

```
%subset(data=AE, var1=AESTDY, var2=AEENDY, condition= v1>v2>.z );
%subset(data=CM, var1=CMSTDY, var2=CMENDY, condition= v1>v2>.z );
%subset(data=EX, var1=EXSTDY, var2=EXENDY, condition= v1>v2>.z );
```

This code performs SDTM compliance check SD0012: Study Day of Start of Event, Exposure or Observation (--STDY) must be less or equal to Study Day of End of Event, Exposure or Observation (--ENDY). When the macro calls run, the dataset FINDING will be non-empty if the condition is met (STDY variable greater than ENDY), indicating a failure of the check and providing records at the same time. Please note the current code does not collect details for reporting.

Using metadata to execute checks allows the assumption that correct parameters will be passed into check macros, which simplifies the code in the macros. One _NULL_ DATA STEP allows the conditional execution of any number of macros, any number of times as demanded by the input data. The next section examines the first data issue addressed in this paper.

DATA ISSUE #1

Numeric outliers are one of the most easily recognizable data issues. They appear in data such as laboratory tests, vital signs, and ECGs as values too large or small to be plausible. It is not obvious how to define what constitutes “too large or too small”, hence the defining feature of this and any data quality check is the algorithm making this judgment. Consider the following macro **%chk1**:

```
%macro chk1(data=, numvar=, class1=, class2=, class3= );

proc means noprint nway missing data=&data(where=(&numvar>.z));
    var &numvar;
    class &class1 &class2 &class3;
    output out=quartile1 median=median min=min max=max q1=q1 q3=q3;
run;

data quartile2(where=(n(low,high)>0));
    set quartile1;
    dif1=median-q1;
    dif2=q3-median;
    if dif1>0 and min<(median-20*dif1) then low=(median-20*dif1);
    if dif2>0 and max>(median+20*dif2) then high=(median+20*dif2);
run;

proc sql noprint;
    create table finding as
        select distinct a.low, a.high, a.q1, a.median, a.q3, b.*
```

```
            from quartile2 a left join
                &data(where=(&numvar>.z) keep=&numvar &class1 &class2 &class3) b
            on a.&class1=b.&class1 and a.&class2=b.&class2 and a.&class3=b.&class3
                and (.z<a.high<b.&numvar or a.low>b.&numvar);
quit;

%mend chk1;
```

The first step, a PROC MEANS, creates dataset QUARTILE1 containing the median, 1st & 3rd quartile, minimum and maximum of the specified numeric variable stratified by 3 class variables. The second step, a DATA STEP, processes QUARTILE1 by determining if the minimum or maximum is more than 20x the distance away from the median as compared to the 1st or 3rd quartile. If so, a record is output to dataset QUARTILE2 with LOW and HIGH limits set to the 20x mark. The third step, a PROC SQL, outputs values found outside the limits defined in QUARTILE2 to the dataset FINDING.

This algorithm defines an allowable value range of twenty times the interquartile range (distance between Q3 and Q1) and considers values outside of it outliers. Using the data driven _NULL_ DATA STEP presented above, suppose the following line of code is generated by CALL EXECUTE:

```
%chk1(data=LB, numvar=LBSTRESN, class1=LBCAT, class2=LBTEST, class3=LBSTRESU );
```

This applies the algorithm in **%chk1** to LB, generating findings such as the high bilirubin values below. The difference between the median and Q3 for bilirubin is 0.2, hence a value of 4.6 is 20.5x the distance away from the median of 0.5 and may be an outlier. 96000 however probably is one.

DATASET	DETAILS	DETAILS
LB	LBCAT=CHEMISTRY LBTEST=Bilirubin LBSTRESU=mg/dl	High Value=96000 Median=0.5 Q3=0.7
LB	LBCAT=CHEMISTRY LBTEST=Bilirubin LBSTRESU=mg/dl	High Value=4.6 Median=0.5 Q3=0.7
LB	LBCAT=CHEMISTRY LBTEST=Bilirubin LBSTRESU=mg/dl	High Value=4.508 Median=0.5 Q3=0.7

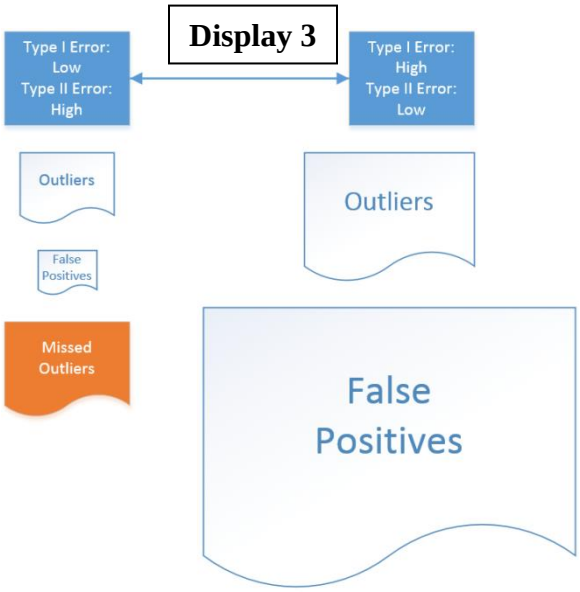
The collection and presentation style of findings will be discussed in a later section. Before that, there is hopefully valid concern regarding the arbitrary nature of the detection algorithm. The values 4.6 and 4.508 are borderline findings. If the acceptable range was extended to 21x the interquartile range, these values would be eliminated. If the range was reduced to 19x, more outliers would be identified. How should it be determined which limits are valid?

TYPE I & TYPE II ERROR

Compared with CDISC compliance checks, which are black and white in their evaluation to a pass or fail result, data quality algorithms adhere to the concept of Type I and Type II errors. A Type I error is also known as a false positive; using outlier detection as an example, it means a value was reported to be an outlier when it is not. A Type II error is also known as a false negative, a real outlier in the data that the check did not detect and report.

While it is given that we want both error types to be low, reducing one type of error may increase the other. Consider bilirubin values of 4.6 and 4.508 that are considered outliers under the 20x interquartile range rule. If an investigation reveals they are not outliers and in response the rule was adjusted to 21x, these false positives would be gone reducing Type I error. For another lab test however, conceivably real outliers will be missed because of this adjustment, increasing Type II error.

How then should Type I and Type II errors be balanced? Take the case the outlier detection algorithm is tweaked to have very low Type I error by setting the acceptable range to 100x the interquartile range. Every value detected and reported is likely to be an outlier, but a great many real outliers will be missed. Suppose instead the algorithm was tweaked to have very low type II error by setting the acceptable range to 5x the interquartile range, most outliers will be detected but be hidden in the report by an enormous number of false positive values. This is illustrated in Display 3.



The deciding factor to select where to dial in an algorithm is time: while perfect data quality would be nice, in the real world there is a finite amount of effort to dedicate to data quality. A report containing more valid issues will also contain a larger proportion of false positives; the size of the report and the time to process it will increase faster than the number of valid issues it contains. The best report allows the detection and resolution of a maximum number of data issues in the time available. For most purposes, if 25-50% of a check's findings are real issues, each is worth the time to investigate. The severity of valid findings will vary by check, so acceptable Type I and II errors may be decided individually for each check.

DATA ISSUE #2

A variation on the outlier detection algorithm is the site outlier detection algorithm. While individual values may be too high or low compared to a majority of the data, there are also sites that as a whole report values outside the expected range. Reasons may be statistical randomness, wrong units, incorrect labelling, or other causes that an investigation would reveal. Consider the following macro **%chk2**:

```
%macro chk2(data=, numvar=, class1=, class2=, class3= );

proc means noprint nway missing data=&data(where=(&numvar>.z));
    var &numvar;
    class &class1 &class2 &class3;
    output out=allsite median=median q1=q1 q3=q3;
run;

proc sql noprint;
    create table inputsite(where=(&numvar>.z)) as
        select a.*, b.siteid
        from &data a left join dm b
        on a.usubjid=b.usubjid and a.studyid=b.studyid;
quit;

proc means noprint nway missing data=inputsite;
    var &numvar;
    class &class1 &class2 &class3 siteid;
    output out=site1 median=sitemedian;
run;

data finding(where=(flag ne ' '));
    merge allsite(drop=_:) site1;
    by &class1 &class2 &class3;
    dif1=median-q1;
    dif2=q3-median;
    if dif1>0 and sitemedian<(median-5*dif1) then flag='L';
    if dif2>0 and sitemedian>(median+5*dif2) then flag='H';
run;

%mend chk2;
```

The first step, a PROC MEANS, creates dataset ALLSITE containing the median, 1st & 3rd quartile for all sites as in data issue #1. The second step, a PROC SQL, adds variable DM.SITEID to the input dataset, creating temporary dataset INPUTSITE. The third step, another PROC MEANS, calculates the median for each site with INPUTSITE. In the fourth step, the median by site is checked against 5x the interquartile range from all sites, and site medians outside that range are flagged for review in dataset FINDING. Code to count the number of subjects in each site and for presentation was omitted for simplicity.

The following CALL EXECUTE generated macro call finds and reports the possible outlier sites below:

```
%chk2(data=LB, numvar=LBSTRESN, class1=LBCAT, class2=LBTEST, class3=LBSTRESU );
```

DATASET	DETAILS	DETAILS	
LB	LBCAT=CHEMISTRY LBTEST=Bilirubin LBSTRESU=mg/dl	High Site=038	Site N=14 Site Median=9 Median=0.5 Q3=0.7
LB	LBCAT=CHEMISTRY LBTEST=Bilirubin LBSTRESU=mg/dl	High Site=040	Site N=16 Site Median=3.5 Median=0.5 Q3=0.7
LB	LBCAT=CHEMISTRY LBTEST=Bilirubin LBSTRESU=mg/dl	High Site=039	Site N=13 Site Median=11 Median=0.5 Q3=0.7

The median of the bilirubin results reported by these sites, each with more than 10 subjects, are an order of magnitude higher than the median across all sites. It may be the case that the data is valid, however seeing such a result is usually cause for further investigations.

The algorithms used in the previous two checks both use the median and quartiles rather than the mean and standard deviation. This is because an egregious outlier, such as the 96000 bilirubin value, could have a significant impact in shifting the arithmetic mean and standard deviation but not the quartiles. Nonetheless, there are many valid algorithms for each check with strengths and weaknesses that may be debated. Next is an example of a check with many possible implementation variations.

DATA ISSUE #3

In data where verbatim terms are coded for tabulation purposes, such as adverse events, concomitant medications and disposition events, the same verbatim term is expected to be coded the same way. It is simple to check that the exact text string is not coded to different values, however what about a slightly different text string? Is it Ok if “MOOD DEPRESSION” is coded differently than “MOOD-DEPRESSION” or “LOW WBC” is coded differently than “LOW WBC’S”? The answer is usually no as the accuracy of tabulations would suffer.

To create a check that finds similar terms coded to different values, a string pattern matching algorithm is necessary. Please see macro **%chk3** below:

```
%macro chk3(data=, termvar=, codevar= );

proc sort nodupkey out=match1
    data=&data(keep=&termvar &codevar rename=(&termvar=v1 &codevar=v2));
    by v1 v2;
    where v1 ne ' ';
run;

data match2;
    set match1;
    v1c=compress(v1,,'adk');
    loop=_n_;
    len=length(v1c);
run;

data finding;
    set match2(rename=(v1=v1_ v2=v2_ v1c=v1c_ len=len_)) nobs=num_obs;
    do i= loop+1 to num_obs;
        set match2(drop=loop) point=i;
        if .7<=len/len_<=1.42 and v2 ne v2_ then do;
            limit=(len+len_)*5;
            score=min(compged(v1c,v1c_,1000,'iL'), compged(v1c_,v1c,1000,'iL'));
            if score<min(limit,1000) then output;
        end;
    end;
run;

%mend chk3;
```

The first step, a PROC SORT, removes duplicates and renames the verbatim variable to MATCH1.V1 and coded variable to MATCH1.V2. The second step, a DATA STEP, creates variable MATCH2.V1C by removing characters that are not alphabetical letters or digits from verbatim variable MATCH1.V1 using the COMPRESS function. The position of each record in the data is stored in variable MATCH2.LOOP and the length of MATCH2.V1C is stored in variable MATCH2.LEN.

The third step, a DATA STEP, sets each record of dataset MATCH2 side-by-side with every other record of dataset MATCH2, allowing a pattern matching algorithm to operate on unique pairs of modified verbatim values. If a pair of verbatim values is coded to different values, the SAS function COMPGED is used to generate a numeric score representing the level of differences between the two strings. If the score is less than an arbitrary threshold of five times the length of the verbatim terms, the pair of values is output to the dataset FINDING for review.

The third step also includes optimizations to increase computing performance. The LOOP variable is used in conjunction with the POINT option to reduce the pairwise comparisons to unique pairs; if record 1 is compared with

record 2, record 2 is not compared with record 1 again. A length condition ensures each pair of verbatim terms differs in length by no more than 30%, this also reduces computational time.

A CALL EXECUTE step generates the following line for dataset AE and identified some coding issues:

```
%chk3(data=AE, termvar=AETERM, codevar=AEDECOD );
```

DATASET	DETAILS	DETAILS
AE	AETERM=MOOD / DEPRESSION	AEDECOD=DEPRESSION
AE	AETERM=MOOD DEPRESSION	AEDECOD=DEPRESSED MOOD
AE	AETERM=MOOD-DEPRESSION	AEDECOD=DEPRESSION
AE	AETERM=THROMBOSIS (VASCULAR ACCESS RELATED)	AEDECOD=THROMBOSIS IN DEVICE
AE	AETERM=THROMBOSIS VASCULAR ACCESS RELATED	AEDECOD=THROMBOSIS
AE	AETERM=TRANSAMINASES INCREASE	AEDECOD=HYPERTRANSAMINASAEMIA
AE	AETERM=TRANSAMINASES INCREASED	AEDECOD=TRANSAMINASES INCREASED
AE	AETERM=CIRCULATION DISORDER	AEDECOD=ANGIOPATHY
AE	AETERM=CIRCULATION DISORDERS	AEDECOD=CARDIOVASCULAR DISORDER
AE	AETERM=E. COLI, URINARY TRACT INFECTION	AEDECOD=URINARY TRACT INFECTION BACTERIAL
AE	AETERM=E.COLI URINARY TRACT INFECTION	AEDECOD=ESCHERICHIA URINARY TRACT INFECTION
AE	AETERM=INCREASE CHOLESTEROL	AEDECOD=BLOOD CHOLESTEROL INCREASED
AE	AETERM=INCREASE CHOLESTEROLO	AEDECOD=HYPERCHOLESTEROLAEMIA
AE	AETERM=LOW WBC	AEDECOD=WHITE BLOOD CELL COUNT DECREASED
AE	AETERM=LOW WBC'S	AEDECOD=LEUKOPENIA

The results above are a selected set of likely miscoded terms detected by this algorithm. Many false positives are also identified; terms naturally similar to each other such as “ELEVATED ALT” versus “ELEVATED ALP” would be flagged using this code. By controlling the score limit threshold, over half of detected coding issues are valid; however as a whole this algorithm has substantial Type I & II error.

Luckily, the code presented is intended to be a starting point; there are many steps one can take to improve this check. The COMPGED function has scoring rules that can be adjusted, completely different pattern matching algorithms are available, a white-list to manually rule out similar strings as false positives can be created, and a better check with smaller Type I and II errors can be built. This challenge will be left to the reader.

PRESENTATION

As the number of checks increase and the types of detected issues become more varied, it is worthwhile to create a standard presentation style that helps reviewers grasp the report more easily. OpenCDISC Validator reports are freely available and could be used as a reference: A list of checks with details on the rules, and separately a list of issues identified with each check.

The code below shows an expansion of the example in the data driven section to collect check results.

```
data results;
    length check $4 dataset $20 detail1 detail2 $1000;
    stop;
run;

%macro subset(data=, var1=, var2=, condition= );
    data finding;
        set sdtmlib.&data(keep=&var1 &var2 rename=(&var1=v1 &var2=v2));
        if &condition;
    run;

    data results;
        set results finding(in=a obs=5);
        if a then do;
            check='0001';
            dataset="&data";
            detail1=cats("&var1=",vvalue(v1),", &var2=",vvalue(v2));
        end;
        keep check dataset detail1 detail2;
    run;
%mend subset;
```

An empty dataset RESULTS is initialized first. For each call of the check macro, records from the FINDING dataset are added to the RESULTS dataset with processing to include the check number, dataset name, variable names and values. This information is sufficient to allow a reviewer to see the issue is valid and track it down in the data. The OBS=5 option was used to limit the number of findings to 5 per macro call. For some checks, an error may be repeated thousands of times, and if allowed to can needlessly clutter the report with redundant findings.

Included at the end of the paper in APPENDIX A is expanded code for each of the 3 data issues discussed, with finding collection code omitted earlier. Each check was adapted to present issues using different criteria such as number of records and sorting order. While the final output is a SAS dataset, it is recommended to output to another document such as an EXCEL spreadsheet for a professional appearance.

CONCLUSION

This paper presented three examples of data issues that could be detected with a data quality application. The application uses the metadata of input CDISC datasets to select check macros to execute, and collects results into a final report. Check macros use relatively simple SAS code and are calibrated for each data issue. In APPENDIX A, complete code to run these three checks is provided so readers can try them on real data.

To build a good data quality assurance application, the final remaining task is to add more checks: make use of experience with data issues to create SAS algorithms that detect and prevent them. If 100 checks pave the road to quality CDISC compliant data, 3 are down and only 97 left to go.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact Wayne at:

Author Name:	Wayne Zhong
Company:	Accenture
Address:	1160 W. Swedesford Rd. Bldg. One, Berwyn PA
Work Phone:	(610) 407-7584
Email:	wayne.zhong@accenture.com

APPENDIX A

```
** fill in the path the SDTM datasets below **;
** run the entire code interactively in SAS **;
** open final dataset RESULTS **;
** can also right click RESULTS dataset and open in EXCEL **;

libname sdtmlib 'x:\path to sdtm datasets\' access=readonly;


** get metadata for specified library **;
data meta1(keep=memname name) meta2(keep=memname allvars);
    set sashelp.vcolumn;
    where libname='SDTMLIB' & memtype='DATA';
    by memname;
    if type='num' then name=cats(name,'#');
    else name=cats(name,'$');

    output meta1;

    length allvars $10000;
    retain allvars;
    if first.memname then allvars=' '||name;
    else allvars=trim(allvars)||' '||name;

    if last.memname then output meta2;
run;

data meta3;
    merge meta1 meta2;
    by memname;
run;

data results;
    length check $4 dataset $20 detail1 detail2 $1000;
    stop;
run;

** checks **;
%macro subset(data=, var1=, var2=, condition= );

data finding;
    set sdtmlib.&data(keep=&var1 &var2 rename=(&var1=v1 &var2=v2));
    if &condition;
run;

data results;
    set results finding(in=a obs=5);
    if a then do;
        check='0001';
        dataset="&data";
        detail1=cats("&var1=",vvalue(v1),", &var2=",vvalue(v2));
    end;
    keep check dataset detail1 detail2;
run;

%mend subset;


%macro chk1(data=, numvar=, class1=, class2=, class3= );

proc means noprint nway missing data=sdtmlib.&data(where=(&numvar>.z));
    var &numvar;
    class &class1 &class2 &class3;
    output out=quartile1 median=median min=min max=max q1=q1 q3=q3;
run;

data quartile2(where=(n(low,high)>0));
    set quartile1;
    dif1=median-q1;
    dif2=q3-median;
    if dif1>0 and min<(median-20*dif1) then low=(median-20*dif1);
    if dif2>0 and max>(median+20*dif2) then high=(median+20*dif2);
run;

proc sql noprint;
    create table finding as
        select distinct a.low, a.high, a.q1, a.median, a.q3, b.*
        from quartile2 a left join
            sdtmlib.&data(where=(&numvar>.z) keep=&numvar &class1 &class2 &class3) b
```

```
        on a.&class1=b.&class1 and a.&class2=b.&class2 and a.&class3=b.&class3
            and (.z<a.high<b.&numvar or a.low>b.&numvar)
        order by &class1, &class2, &class3, &numvar desc;
quit;

data results;
    set results finding(in=a);
    if a then do;
        check='1001';
        dataset="&data";
        detail1=cats("&class1=",vvalue(&class1),", &class2=",vvalue(&class2),", &class3=",vvalue(&class3));
        if &numvar>high>.z then detail2=cats("Median=",vvalue(median),", Q3=",vvalue(q3),", High
Value=",vvalue(&numvar));
        if &numvar<low     then detail2=cats("Median=",vvalue(median),", Q1=",vvalue(q1),", Low
Value=",vvalue(&numvar));
    end;
    keep check dataset detail1 detail2;
run;

%mend chk1;


%macro chk2(data=, numvar=, class1=, class2=, class3= );

proc means noprint nway missing data=sdtmlib.&data(where=(&numvar>.z));
    var &numvar;
    class &class1 &class2 &class3;
    output out=allsite median=median q1=q1 q3=q3;
run;

proc sql noprint;
    create table inputsite(where=(&numvar>.z)) as
        select a.*, b.siteid
        from sdtmlib.&data a left join sdtmlib.dm b
        on a.usubjid=b.usubjid and a.studyid=b.studyid;
quit;

proc means noprint nway missing data=inputsite;
    var &numvar;
    class &class1 &class2 &class3 siteid;
    output out=site1 median=sitemedian;
run;

data finding(where=(flag ne ' '));
    merge allsite(drop=_:) site1;
    by &class1 &class2 &class3;
    dif1=median-q1;
    dif2=q3-median;
    if dif1>0 and sitemedian<(median-5*dif1) then flag='L';
    if dif2>0 and sitemedian>(median+5*dif2) then flag='H';
run;

proc sort data=inputsite nodupkey out=siten;
    by &class1 &class2 &class3 siteid usubjid;
run;

proc freq data=siten noprint;
    table &class1*&class2*&class3*siteid/out=siten2;
run;

proc sort data=finding;
    by &class1 &class2 &class3 siteid;
run;

data finding2;
    merge finding(in=a) siten2;
    by &class1 &class2 &class3 siteid;
    if a;
run;

proc sort data=finding2;
    by &class1 &class2 &class3 descending count siteid;
run;

data results;
    set results finding2(in=a);
    if a then do;
        if count>4;
        check='1002';
        dataset="&data";
        detail1=cats("&class1=",vvalue(&class1),", &class2=",vvalue(&class2),", &class3=",vvalue(&class3));
```

```
        if flag='H' then detail2=cats("Median=",vvalue(median),", Q3=",vvalue(q3),", Site N=",vvalue(count),",
High Site Median=",vvalue(sitemedian),", Site ID=",vvalue(siteid));
        if flag='L' then detail2=cats("Median=",vvalue(median),", Q1=",vvalue(q1),", Site N=",vvalue(count),",
Low Site Median=",vvalue(sitemedian),", Site ID=",vvalue(siteid));
    end;
    keep check dataset detail1 detail2;
run;

%mend chk2;


%macro chk3(data=, termvar=, codevar= );

proc sort nodupkey out=match1
    data=sdtmlib.&data(keep=&termvar &codevar rename=(&termvar=v1 &codevar=v2));
    by v1 v2;
    where v1 ne ' ';
run;

data match2;
    set match1;
    v1c=compress(v1,,'adk');
    loop=_n_;
    len=length(v1c);
run;

data finding;
    set match2(rename=(v1=v1_ v2=v2_ v1c=v1c_ len=len_)) nobs=num_obs;
    do i= loop+1 to num_obs;
        set match2(drop=loop) point=i;
        if .7<=len/len_<=1.42 and v2 ne v2_ then do;
            limit=(len+len_)*5;
            score=min(compged(v1c,v1c_,1000,'iL'), compged(v1c_,v1c,1000,'iL'));
            sort=score/limit;
            if score<min(limit,1000) then output;
        end;
    end;
run;

data finding2;
    length detail1 detail2 $1000;
    set finding;
    order=_n_;
    detail1=cats("&termvar=",vvalue(v1));
    detail2=cats("&codevar=",vvalue(v2));
    output;
    detail1=cats("&termvar=",vvalue(v1_));
    detail2=cats("&codevar=",vvalue(v2_));
    output;
run;

proc sort data=finding2;
    by sort order;
run;

data results;
    set results finding2(in=a);
    if a then do;
        check='1003';
        dataset="&data";
    end;
    keep check dataset detail1 detail2;
run;

%mend chk3;


** check whether DM.SITEID exists for chk3 **;
data _null_;
    set meta3 end=eof;
    retain check 0;
    if memname='DM' and name='SITEID$' then do;
        call symput('dmsiteid','"Y"');
        check=1;
    end;
    if eof and check=0 then call symput('dmsiteid','" "');
run;


** call execute checks based on metadata **;
data _null_;
```

```
    set meta3;
    if index(name,'STDY#') and index(allvars,' '||tranwrd(strip(name),'STDY#','ENDY#'))
        then call execute(
            cats(
            '%subset(data=',memname,
                  ', var1=',compress(name,'#$'),
                  ', var2=',tranwrd(name,'STDY#','ENDY'),
                  ', condition= v1>v2>.z );'
            )
        );

    if index(name,'STRESN#') and index(allvars,' '||tranwrd(strip(name),'STRESN#','CAT$'))
        and index(allvars,' '||tranwrd(strip(name),'STRESN#','TEST$'))
        and index(allvars,' '||tranwrd(strip(name),'STRESN#','STRESU$'))
        then call execute(
            cats(
            '%chk1(data=',memname,
                ', numvar=',compress(name,'#$'),
                ', class1=',tranwrd(name,'STRESN#','CAT'),
                ', class2=',tranwrd(name,'STRESN#','TEST'),
                ', class3=',tranwrd(name,'STRESN#','STRESU'),
                ');'
            )
        );

    if &dmsiteid="Y" and index(name,'STRESN#') and index(allvars,' '||tranwrd(strip(name),'STRESN#','CAT$'))
        and index(allvars,' '||tranwrd(strip(name),'STRESN#','TEST$'))
        and index(allvars,' '||tranwrd(strip(name),'STRESN#','STRESU$'))
        then call execute(
            cats(
            '%chk2(data=',memname,
                ', numvar=',compress(name,'#$'),
                ', class1=',tranwrd(name,'STRESN#','CAT'),
                ', class2=',tranwrd(name,'STRESN#','TEST'),
                ', class3=',tranwrd(name,'STRESN#','STRESU'),
                ');'
            )
        );

    if index(name,'TERM$') and index(allvars,' '||tranwrd(strip(name),'TERM$','DECOD$'))
        then call execute(
            cats(
            '%chk3(data=',memname,
                ', termvar=',compress(name,'#$'),
                ', codevar=',tranwrd(name,'TERM$','DECOD'),
                ');'
            )
        );

run;

proc sort data=results;
    by dataset check;
run;
```