PhUSE 2016

Paper DH02

Data De-Identification Made Simple

Jørgen Mangor Iversen, LEO Pharma A/S, Ballerup, Denmark

ABSTRACT

This paper is a presentation of a small collection of macros to de-identify a complete set of well-formed SDTM data using the de-identification specification itself as metadata. The macros will handle any set of domains and variables, as long as they comply with SDTM in any version compatible with the specification. The macros demonstrate a robust way of handling metadata, variations of SDTM data, SAS code generation and dynamic execution, macro parameter validation, and reporting of any findings. The presentation is though focused on interpretation and implementation of various de-identification requirements as they are presented in the specification.

INTRODUCTION

The main idea for this solution came from a quick study of the **Rules** and **SDTMIG** sheets of the spreadsheet containing the De-Identification standard from SDTM data from PhUSE [1]. The **Rules** sheet specifies in sufficient detail algorithms for de-identifying various types of data points, and the **SDTMIG** sheet consists of a full set of metadata for applying the algorithms to each data point in a set of SDTM data. Knowing that LEO Pharma A/S (LEO) SDTM data generally are in very close adherence with SDTM specifications, and also knowing that sufficient metadata can easily be extracted from any set of SAS™ datasets, it quickly dawned upon me that the process of de-identification according to the specification ought to be possible to automate with a reasonable effort. The obvious prerequisites are of course that the de-identification specification is complete enough to cover its objectives safely, and that clinical trial SDTM data is structured in a way compatible with the metadata in the **SDTMIG** sheet.

OVERALL STRUCTURE

The structure of the solution thus becomes these simple steps (leaving out trivial housekeeping details):

1. Set up the metadata.
2. Calculate the base of date offsetting.
3. Obtain a working copy of the complete set of domain datasets.
4. Perform trial specific operations (established iteratively by running the program several times).
5. Execute all operations on all variables in prioritized order.
6. Write the de-identified data to their destination.
7. Report on the process.

METADATA

The whole process is metadata driven, and metadata is collected from several sources.

OBTAIN DE-IDENTIFICATION METADATA

The **SDTMIG** sheet is imported from the **phuse-deid-standard---sdtm-3.2---v1.01.xls** excel file using simple PROC IMPORT, keeping domain- and variable names as well as a definition of which operation to perform on each data point. This solution only reads the primary rules, as secondary rules are dealt with as trial specific extra processing.

MAPPING COUNTRIES TO CONTINENTS

A thorough search of the internet did not reveal any simple, reliable and free source of geographical data for mapping countries to continents. It was quite surprising to learn that such seemingly simple information is not readily available without parsing complex web-pages. And yet it is available right under our noses, but found elsewhere. All SAS installations to my knowledge come with map data out of the box, and continents are indeed part of those data.

However, be aware that these data contains 2 errors (in SAS 9.3 and 9.4): Russia is reported to be part of both Europe and Asia, which is geographically true, but I choose to place all of it in Asia to avoid many-to many relations in the mapping. Mexico is reported to be placed in both South America as well as in Europe, which is a bit more mysterious. I have placed Mexico in South (Latin) America, again primarily for mapping reasons.

The **maps.names** and **maps.metamaps** datasets are joined by both **name** and **isoname** variables to increase the number of hits. The results are joined together, the errors corrected and duplicates removed.

MAPPING OF ALGORITHMS TO OPERATIONS

The **Rules** sheet of the **PHUSE_STDM_redaction.xlsx** Excel file define a quite simple set of operations, which can be mapped to macro calls 1:1 through a simple PROC FORMAT:

```
value $rules
  'Remove dataset'                                              = '%di_remove_dsn'
  'Derive Age'                                                  = '%di_derive_age'
  'Offset'                                                      = '%di_offset'
  'Elevate to continent'                                        = '%di_continent'
  'Recode subject ID'                                           = '%di_recode_subject'
  'Recode ID variable'                                          = '%di_recode_id'
  'Remove'                                                      = '%di_remove_var'
  'Keep'                                                        = '%di_keep'
  'No further de-identification'                                = '%di_no_further'
  'Review and only redact values with personal information' = '%di_manual'
  other                                                         = '';
```

PRIORITIZING OPERATIONS

Each macro is given a priority. Again, a simple PROC FORMAT does the trick:

```
invalue rules
  '%di_remove_dsn'     = 1
  '%di_derive_age'     = 2
  '%di_offset'         = 3
  '%di_continent'      = 4
  '%di_recode_subject' = 5
  '%di_recode_id'      = 6
  '%di_remove_var'     = 7
  '%di_keep'           = 9
  '%di_no_further'     = 8
  '%di_manual'         = 10
  other                = 99;
```

The only real trick here is to determine the priority for each macro operation. Removing entire datasets first is done from a performance perspective, thus avoiding unnecessary operations. Age is derived before offsetting dates, to avoid any complications and debugging from doing those operations in the reverse order. Dates are then offset, as age has been derived. Countries are then elevated to continents, as this is a very independent operation. The recoding of subjects implies a resorting of data by the new recoded values. Other id's can be recoded as another very independent operation. Variables are removed as the last data management operation, as other operations may be dependent on variables to be removed. The remaining operations are variations of do nothing, or do something at a later iteration.

CONTROL PROGRAM

Apart from the global metadata, metadata have to be extracted from the SAS datasets holding the clinical trial data themselves. This is guided by the global metadata, which have to be reconciled with the actual trial data.

CLASSES OF VARIABLES

The variable level metadata found in The **SDTMIG** sheet of the **PHUSE_STDM_redaction.xlsx** Excel file defines 3 classes of variables:

- Global variables identified by having no domain prefix and no double dash prefix.
- Generic variables identified by having a domain prefix, but no double dash prefix.
- Domain specific variables identified by having a domain prefix and a double dash prefix.

Each class of variable names are compared to trial data variables, double dash prefixes replaced by all (but not SUPPQUAL) domain prefixes, using **dictionary.columns** and **dictionary.tables** as the metadata source for the trial data. The classes of variables are collected as a total of variables present in the trial data, and the operation in form of the name of a macro call is added to each variable from the **SDTMIG** metadata table.

COLLECTING MESSAGES

A dataset is set up to collect messages as the operations occur. Trial specific operations are done before the main body of operations in prioritized order. Hence reporting of operations will not be in the same order as they are performed. Each macro documents its operation in the dataset as it is performed.

CALCULATING THE BASE FOR OFFSETTING DATES

Three main dates are considered as candidate for being the initial date of any subject: The reference start date RFSTDTC in DM, the visit date of visit 1 as recorded in SVSTDTC in SV, and the date of informed consent DSSTDTC in DS. These dates are examined across subjects, and the earliest non-missing date is considered the

base date for offsetting all subjects. An offset is calculated for each subject, and stored in a dataset.

SPECIAL CONSIDERATIONS FOR USUBJID

USUBJID is an identifier just like any other identifier, with the significant exception that most domains are sorted by USUBJID as one of several variables. Apart from the sorting, USUBJID is recoded in the same manner as any other identifier to be recoded.

TRIAL SPECIFIC OPERATIONS

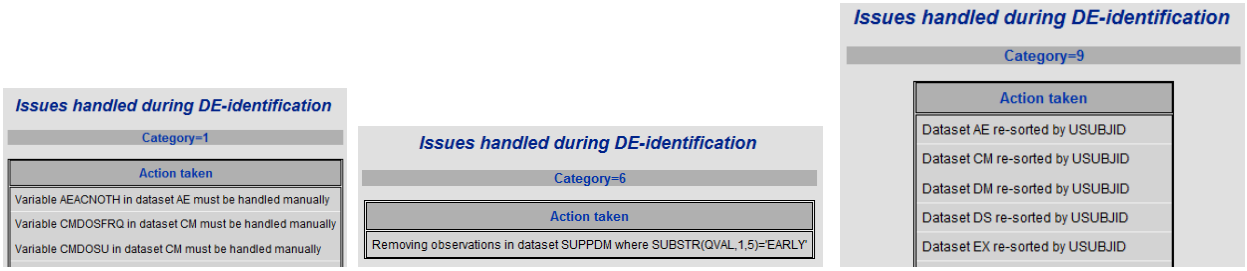
In some cases, trial data contains dates and identifiers in variables normally not identified as such. Examples include dates in QVAL in SUPPQUAL datasets, and SUBJID or even USUBJID in IDVARVAL also in SUPPQUAL datasets. Furthermore, a macro exists to remove certain rows of any dataset, identified by a suitable WHERE clause. Typically, the decision to remove an entire dataset will be a trial specific operation.

RUNNING OPERATIONS DYNAMICALLY

For each variable in each dataset with an associated operation, a macro performing that operation is called with the dataset, variable, and WHERE clause as appropriate, in strict prioritized order. This is the main operation of the program, but really the fewest lines of code.

REPORTING MESSAGES

The dataset containing the messages regarding the performed operations is sorted in order of importance, specified by each macro. The actual reporting is a simple PROC REPORT surrounded by ODS statements, A PROC TEMPLATE specifies a style tweaking the generated HTML output through a custom style.



Issues handled during DE-identification

Category=1

Action taken
Variable AEACNOTH in dataset AE must be handled manually
Variable CMDOSFRQ in dataset CM must be handled manually
Variable CMDOSU in dataset CM must be handled manually

Issues handled during DE-identification

Category=6

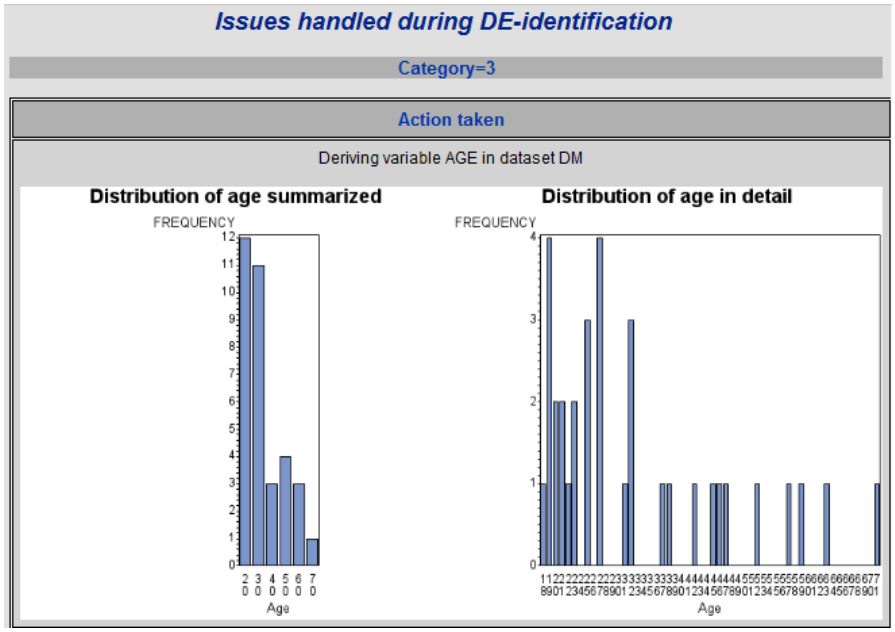
Action taken
Removing observations in dataset SUPPDM where SUBSTR(QVAL,1,5)='EARLY'

Issues handled during DE-identification

Category=9

Action taken
Dataset AE re-sorted by USUBJID
Dataset CM re-sorted by USUBJID
Dataset DM re-sorted by USUBJID
Dataset DS re-sorted by USUBJID
Dataset EX re-sorted by USUBJID

The resulting HTML document simply contains a list of all the operations performed, including a couple of graphs illustrating the distribution of the AGE variable.



OPERATIONS TO DE-IDENTIFY

All the individual operations are completely independent, conflicts resolved through the prioritization.

REMOVING DATASETS

Apart from the obvious, this operation also removes the dataset and all its variables from the metadata, to avoid further processing in vain.

DERIVE AGE

This is by far the longest and most complicated macro, as it does several things. The main procedure is to group together all subjects older than 89 into one group, not that complicated. The macro takes age unit into account, and is able to calculate the tender age of 89 in hours, days, weeks, months and years. Furthermore, a couple of PROC GCHART procedures display the distribution of age in the final report for assessment of further grouping of age.

OFFSET DATES

Offsetting a date is a rather simple operation where the offset is subtracted from the date, per subject. A thing of relatively minor complication is handling of incomplete dates. These are imputed to either the middle of the month or the middle of the year, depending on which parts of the date is missing. Incomplete dates are first imputed, then offset, then stripped of the parts added by the imputation. The result is then an offset imputed date.

ELEVATE COUNTRY TO CONTINENT

The dataset containing the mapping from country to continent is converted into a format, observing the length of continent names. Using this format, the conversion is a straight forward **put()** function.

RECODE SUBJECT ID

The detailed recording of subject identifiers (USUBJID) is no different from the procedure mentioned above for the control program. There really is only one recoding algorithm, common to all recoding operations. Only the additional sorting is added for subject identifiers. The main difference is the naming of the operations in the report.

RECODE OTHER VARIABLES

The recoding of variables in a robust and irreversible manner is really at the heart of de-identification. Luckily, this can be achieved without too much effort, simply by unsorting and renumbering. The recoding will result in a pseudo-random assignment of increasing numbers starting from the number of values rounded up to the nearest multiple of 10. If there are 47 distinct values, the recoded values will range from 101 to 148. This way overflow problems have been avoided, as variable length is adjusted to accommodate for the increased length. Both numeric and character values are handled in the same way taking the data type into account. The variable is then simply unsorted by sorting it according to a pseudo-random function on the variable value itself. I have chosen to use the **md5()** function rather than any of the random number functions, to avoid any issues with seed values when performing recoding of primary and foreign keys at different times of program execution. This way I can assure that the recoding is uniform at each invocation in the same program run, guaranteeing that keys always match up.

The relevant code segment is the following few SQL steps:

```
/* Round up to next multiple of 10 from number of distinct values */
select distinct 10 ** length(strip(put(count(distinct &var), &len..)))
  into :offset
  from &lib..&mem;

/* Obfuscate ID variable */
create table __&mem._&var (drop=order) as select distinct
       &var.,
       %if &type = C %then md5(&var.);
       %else          left(md5(put(&var., &len..)));
       as order
  from &lib..&mem
 order by order;

/* Assign new values as dull but quazi random sequence */
create table __&mem._&var._new as select distinct
       &var,
       %if &type = C %then left(put(monotonic() + &offset, 32.));
       %else                        monotonic() + &offset;
       as new&var
  from __&mem._&var
 order by &var;

/* Update values in situ */
update &lib..&mem a
   set &var = (select distinct
                  %if &type = C %then put(new&var, $&len..);
                  %else new&var;
                  from __&mem._&var._new b
                 where a.&var = b.&var)
 where &var = (select distinct &var
                  from __&mem._&var._new c
                 where a.&var = c.&var)
   and &cond;
```

REMOVE COLUMNS

Removing a column is another one of those very trivial operations, achieved by a simple SQL ALTER TABLE. The prioritizing of operations guarantees that this is always a safe operation to perform.

NON-INTERVENTIONAL OPERATIONS

The operation to keep, do nothing further, and handle manually all implies that de-identification is not necessary or to be assessed individually. This implies that in particular for manual operations; the most important aspect is reporting that the issue should be dealt with.

GENERAL ASSUMPTIONS ACROSS MACROS

All macros performing de-identification operation are forged from the same mold. They take more or less the same parameters in the same order, they all have the same basic structure, and they all report and do one well defined thing each. At the end, they all have a few lines of comments outlining code to test their primary function. This is added for debugging and refinement.

VALIDATION OF ARGUMENTS

The total superset of macro arguments are libname, member name, variable name and where clause. All macros have 2-4 of these arguments. Each macro validates each argument, both for presence and for validity. This is to ensure that no macro crashes when called with an argument for a value that doesn't exist. Even within a closed suite of code such as this, validation of arguments becomes a tool to be exploited when having metadata controlling code execution.

UNIFORMITY OF CODE

The common structure for macros enhances readability and debugging. All macros first validate their arguments, insert a message documenting the operation, and finally perform the operation it is supposed to. One macro performs two operations; recoding and sorting. The structure is maintained as reporting and operation is executed in the same sequence for each operation.

PROCESS OF DE-IDENTIFICATION

The process of de-identification using this solution is iterative by nature, and always comprises of a few manual steps.

BROWSE THE DATA

The first step is to get an overview of the study data. This is done simply by browsing the actual data, scanning for unexpected dates, initials, names, addresses etc. The main purpose is to identify unwanted variables (i.e. INVNAME) or unwanted values in the more verbose text variables. Pay particular attention to the SUPPQUAL domain(s). It might seem like a daunting task to browse through the clinical data of an entire study, but keep in mind that it is not a deep read of every data point, but a scan for out-of-place data values. An average (LEO) study of 800 subjects in 30 domains ought not to take more than one hour to browse.

INITIAL RUN

The control program is modified (lines added) with operations not expected to be caught by the specification. This includes lines of data to be removed in their entirety, columns containing identifiers to be recoded, etc.

EVALUATE OUTPUT

The report from the de-identification is examined to see if all is de-identified as expected. Particularly, the graphs showing the distribution of age are used to determine if further redacting of age is necessary. Also the section of manual handling needs careful evaluation. Finally the de-identified data needs to be browsed again, to catch anything overlooked.

ITERATE

Several iterations of adding (or correcting) the section of manual operations in the control program may be necessary for a complete de-identification.

CONCLUSION

Using the software and the processes specified, it is my best estimate that an entire clinical trial can be de-identified in just a few days, where most of that time is spent estimating the quality of the output from the software, and making decisions and parameter adjustments to further de-identify beyond the suggested level. The main prerequisite for this timeframe is of course that the trial data is well formed in adherence to the SDTM standard. If this is not the case, it may be necessary to do a possibly time consuming conversion of data. The exact version of SDTM version is less important as the structure of data is determined dynamically, as long as there is a good fit between the SDTM version and the data de-identification specification. Should the specification evolve as may well be expected, the suite of macros must naturally evolve with it, particularly in case of new rules emerging.

REFERENCES

1. PhUSE: De-Identification Standard for CDISC SDTM 3.2 version 1.01 date 20-May-2015 at http://www.phuse.eu/Data_Transparency_download.aspx (requires registration).

ACKNOWLEDGMENTS

This paper, and the entire solution, would not be possible without the splendid work done by the PhUSE De-Identification Working Group.

CONTACT INFORMATION

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

Jørgen Mangor Iversen
LEO Pharma A/S
Industriparken 55
DK-2750 Ballerup
Work Phone: +45 72 26 31 16
Email: jmidk@leo-pharma.com
Web: www.leo-pharma.com