PharmaSUG 2021 - Paper AP-103

Developing SAS® Macro to Find Subject's Latest Available Date from Entire Database

Raghava Pamulapati, Merck & Co., Inc., Kenilworth, NJ, USA;
Sandeep Meesala, Merck & Co., Inc., Kenilworth, NJ, USA

ABSTRACT

In Data Analysis and Reporting, programmers often need to find the subject's latest available date for analysis. To find this date, we need to check every possible date variable from the entire study database.

In general, to find the latest available date from multiple date variables, a programmer needs to either sort the data by the date variable and take the latest date by FIRST.VARIABLE or LAST.VARIABLE or using GROUP BY Statement and MAX Function in PROC SQL. This process is time-consuming and hard to debug. The programing challenge is writing simple code to execute each data variable's above steps in all datasets in the study database efficiently and accurately.

INTRODUCTION

This paper mainly focuses on a step-by-step process to develop a macro to find the subject's latest available date using SAS® variable level metadata with efficient techniques like PROC SQL and CALL EXECUTE statements to reduce the length of code and SAS® execution time. This macro will output the dataset with the subject's ID, latest date, dataset names, and variable names containing the latest available date. This macro provides flexibility to users to handle scenarios like datasets with multiple date variables, a selective option to choose only required datasets or/and date variables, and select the date from the subset of records.

PREPARING METADATA

First, we need to prepare a variable-level metadata dataset containing all date variables in a library, with corresponding dataset names, variable types, and variable formats.

We can use SASHELP.VCOLUMN or PROC Contents out option to generate a metadata dataset that contains variables attributes of each dataset in a library except format information for character date variables. This information should be derived using user-defined input. The dataset below is a subset of SASHELP.VCOLUMN filtered by library name, SDTM.

libname	memname	memtype	name	type	length	npos	varnum	label
SDTM	AE	DATA	STUDYID	char	255	104	1	Study Identifier
SDTM	AE	DATA	DOMAIN	char	2	359	2	Domain Abbreviation
SDTM	AE	DATA	USUBJID	char	255	361	3	Unique Subject Identifier
SDTM	AE	DATA	AESEQ	num	8	0	4	Sequence Number
SDTM	AE	DATA	AEGRPID	char	255	616	5	Group ID
SDTM	AE	DATA	AESPID	char	255	871	6	Sponsor-Defined Identifier
SDTM	AE	DATA	AETERM	char	255	1126	7	Reported Term for the Adverse Event
SDTM	AE	DATA	AEDECOD	char	255	1381	8	Dictionary-Derived Term
SDTM	AE	DATA	AECAT	char	255	1636	9	Category for Adverse Event

SELECTING DATE VARIABLES:

Based on the SDTM IG requirement, most of the date variables end with *DTC*, so the dataset example below uses the suffix *DTC* as the filter.

libname	memname	memtype	name	type	length	npos	varnum	label
SDTM	AE	DATA	AESTDTC	char	19	3505	33	Start Date/Time of Adverse Event
SDTM	AE	DATA	AEENDTC	char	19	3524	34	End Date/Time of Adverse Event
SDTM	AE	DATA	AEDTHDTC	char	19	7013	56	Date of Death
SDTM	AE	DATA	AEHODDTC	char	19	7632	61	Hospitalization Discharge Date
SDTM	AE	DATA	AEHOSDTC	char	19	7651	62	Date of Hospitalization
SDTM	BS	DATA	BSDTC	char	19	4983	28	Date/Time of Specimen Collection
SDTM	BS	DATA	BSSTDTC	char	19	5002	29	Start Date/Time of Specimen Collection
SDTM	BS	DATA	BSENDTC	char	19	5021	30	End Date/Time of Specimen Collection
SDTM	BS	DATA	BSRFTDTC	char	19	5295	36	Date/Time of Reference Point
SDTM	BS	DATA	BSSTPDTC	char	19	6842	49	Stop Date and Time of Sample
SDTM	CE	DATA	CEDTC	char	19	4171	28	Date/Time of Event Collection

The programmer can have a parameter in macro to specify the variable suffix as the filter.

All the date variables with the same suffix should have a consistent format, either all in character or all in numeric format. If there is a need to consider variables in different formats, the programmer can set up multiple suffixes as the filter, separated by "|". The filter of each suffix variable type is independent.

Example: `varsuffix = %str(dtc|dti)`

SPECIFYING DATE FORMAT:

To calculate duration, we need to convert character date variables into numerical variables first. To do the conversion, we need to know the date format of the character variables. In our macro, we build in a macro variable, "varfmt" for that.

In the case of the need for multiple date suffixes, the user needs to specify each set of date variable format. Just like the above example, the user can pass in "is8601da. |is8601da." which is corresponding to "DTC" | "DTI" respectively.

Example: `varcfmt = %str(is8601da|is8601da)`

Please note, the value of varsuffix and varcfmt parameters should be consistent. It's required to pass format for numerical variables as well.

SELECTING SPECIFIC DATE VARIABLES:

Users can select specific date variables that come with different options. In this macro, we have four parameters to allow the users to either keep datasets/ variables or drop datasets/ variables from the metadata dataset.

Users can pass multiple datasets and variables into any of these parameters separated by a comma.

Example for keep:

```
dskeep  = %str(ds,ae)
varkeep = %str(dsstdtc,aestdtc)
```

Example for drop:

```
dsdrop = %str(dm,ae)

vardrop =
```

Please note using both keep and drop parameters might give wrong results.

ADDING FILTER CONDITION:

We also built-in filter clause parameters to filter out certain records in specific datasets. The example below demonstrates a scenario of filtering out visit two or screening epoch records from VS and AE

datasets.

Example:
```
fillterds = %str(vs | ae)
filltercond = %str(vsgrpid="VS02" | epoch ^="SCREENING")
```

ID VARIABLE:

This macro has an ID variable for the study participation identification variable.

Example: `idvars = usubjid`

OTHER PARAMETERS:

Few other parameters are defined to allow users to control the output dataset, output variable name, and variable format.

Example of final variable level metadata:

libname	memname	name	type	where_clause	varfmt
SDTMS	AE	AESTDTC	char	and ^missing(epoch)	is8601da
SDTMS	AE	AEENDTC	char	and ^missing(epoch)	is8601da
SDTMS	AE	AEDTHDTC	char	and ^missing(epoch)	is8601da
SDTMS	AE	AEHODDTC	char	and ^missing(epoch)	is8601da
SDTMS	AE	AEHOSDTC	char	and ^missing(epoch)	is8601da
SDTMS	CM	CMDTC	char		is8601da
SDTMS	CM	CMSTDTC	char		is8601da
SDTMS	CM	CMENDTC	char		is8601da
SDTMS	CM	CMENDTI	char		is8601da
SDTMS	CM	CMSTDTI	char		is8601da
SDTMS	CM	STINMDTI	char		is8601da
SDTMS	CM	ENINMDTI	char		is8601da
SDTMS	DM	RFSTDTC	char		is8601da
SDTMS	DM	RFENDTC	char		is8601da
SDTMS	DM	BRTHDTC	char		is8601da
SDTMS	DM	DMDTC	char		is8601da
SDTMS	DM	BRTHDTI	char		is8601da
SDTMS	DM	DISCNDTC	char		is8601da
SDTMS	DS	DSDTC	char		is8601da
SDTMS	DS	DSSTDTC	char		is8601da

FINDING LATEST DATE PER SUBJECT IN EACH DATE VARIABLES:

Date variables are in multiple datasets, and each dataset may have more than one date variable. The subject's latest available date may be present in more than one dataset and variables.

This program will find the subject's latest date from each date variable separately as a first step. Selecting the latest date per date variable is done using PROC SQL Max function and Group BY operator. PROC SQL code is more efficient to write and execute than the data step approach.

Since we need to repeat selecting the latest date per subject of each date variable for every data set, DATA SET CALL EXECUTE statement can significantly simplify the code and reduce the running time.

CALL EXECUTE:

The CALL EXECUTE routine accepts a single argument that is a character string or character expression. The character expression is usually a concatenation of strings containing SAS code elements for execution after they are resolved. The argument expression components can be character constants, data step variables, macro variable reference, and macro references. CALL EXECUTE dynamically builds SAS code during DATA step iterations; that code executes after the DATA step's

completion outside its boundary. It makes a DATA step iterating through a driver table an effective SAS code generator similar to SAS macro loops.

Suppose an argument string to the CALL EXECUTE contains SAS code without any macro or macro variable references. In that case, that code is pushed out (of the current DATA step) and appended to a queue after the current DATA step. As the DATA step iterates, the code is appended to the queue as often as the iterations. After the DATA step completes, the code gets executed in the order of its creation in the queue.

In the below example, we are executing the SQL step on each date column to find the latest date per subject per date column. Using this step, we are limiting the length of the code and reducing the processing time.

Example:

```
call execute
       ('proc sql ;
             create table &_var._&_dsn.
             as  select
             &idvar.
             ,"&_dsn." as memname length = 2
             ,"&_var." as name length = 8
             ,max(input('|| strip(name) || ',?' || strip(varfmt)||
             '. )) as _dt
             from '  || strip(libname) || '.' || strip(memname) ||
             '
             where 1=1 ' || strip(where_clause) || '
             group by &idvar. ;
        quit;'

       );
```

SELECTING LATEST DATE FROM ALL DATASETS:

Now we have all date variables with the latest date per subject present in "summary_data". We will use this dataset and create one record per subject with the latest date.

```
proc sql;
      create table maxdate0
            as select *
                  ,max(_dt) as &outvarname.
            from __summary
                  group by &idvar
                        order by &idvar, _dt, memname,name;
quit;
```

CREATING VARIABLES WITH LATEST DATE INFORMATION:

After finding the latest date, the next step is to find the variable having the latest date value across detests. If multiple variables in different datasets all have the value of the latest date, the macro will concatenate these datasets and variables with "." as a limiter.

The final dataset contains a variable with the latest date variable and source variable.

Unique Subject Identifier	lkadt	source
1234-000_000100014	2019-01-23	DS.DSDTC, DS.DSSTDTC
1234-000_000200001	2017-08-28	AE.AESTDTC, DS.DSDTC, DS.DSSTDTC
1234-000_000200002	2018-03-06	AE.AEENDTC, DS.DSDTC, DS.DSSTDTC
1234-000_000200003	2017-10-10	AE.AESTDTC, DS.DSDTC, DS.DSSTDTC
1234-000_000200004	2017-11-08	DS.DSDTC, DS.DSSTDTC
1234-000_000200005	2018-01-30	AE.AEENDTC, DS.DSDTC, DS.DSSTDTC
1234-000_000200006	2019-11-01	DS.DSDTC, DS.DSSTDTC
1234-000_000200007	2017-11-30	DS.DSDTC, DS.DSSTDTC
1234-000_000200008	2018-04-10	DS.DSDTC, DS.DSSTDTC
1234-000_000200009	2018-04-27	DS.DSDTC, DS.DSSTDTC
1234-000_000200010	2018-04-29	AE.AEENDTC, CM.CMENDTC, CM.CMENDTI, CM.CMSTDTC, CM.CMSTDTI, DS.DSDTC, DS.DSSTDTC
1234-000_000200011	2018-03-02	DS.DSDTC, DS.DSSTDTC
1234-000_000200012	2018-05-15	DS.DSDTC, DS.DSSTDTC
1234-000_000200013	2020-10-13	CM.CMSTDTC, CM.CMSTDTI, CM.STINMDTI

USE CASES:

Here are some of the use cases for deriving the latest date

- For oncology studies, the last alive date will be useful for overall survival analysis.
- Analysis/date imputations that require the subject's latest date.
- To identify data issues like if any last alive date is greater than the death date.
- Identify records beyond the cutoff date.

MACRO TEMPLATE AND CALLING CODE EXAMPLE:

Below macro template and calling code is a step by step approach to build a standard macro. Please note, user needs to update few areas based on the logic mentioned in above sections.

```
%macro finding0latest0date
(libname =
,idvar =
,varsuffix =
,varcfmt =
,dskeep =
,varkeep =
,dsdrop =
,vardrop =
,outdsn =
,outvarname =
,outvarfmt =
,fillterds =
,filltercond =
);

%*-------------------------------------------------------------------------
-------------------------*;
%*- Step 01: Creating variable level metadata with list of required date
variables and its attributes -*;
```

```
%*-------------------------------------------------------------------------
--------------------------*;

data date_metadata;
set sashelp.vcolumn;
where upcase("&libname.") = libname ;
i1 = 1;

%* Deriving where_clause variable with user defined where clause to
corresponding dataset and variables;

do until (scan("&fillterds.",i1, "|") = " ");
if scan(upcase(compress("&fillterds.")),i1,"|") = compress(memname) then
where_clause = "and " || strip(scan(compress("&filltercond."),i1,"|"));
i1+1;
end;

%* Dropping user specified dataset and variable from variable level meta
data;

if indexw(upcase("&dsdrop"),memname,"|") > 0 then
delete;

if indexw(upcase("&vardrop"),name,"|") > 0 then
delete;
i2 = 1;

%* Deriving var_format variable with user defined format to corresponding
date variables;

do until (scan("&varsuffix.",i2, "|") = " ");
if
index(strip(reverse(name)),strip(reverse(scan(upcase(compress("&varsuffix."))
,i2,"|")))) = 1 then
varfmt = scan(compress("&varcfmt."),i2,"|");
i2+1;
end;

%* Keeping user specified dataset and variable from variable level meta data;

if ^missing(varfmt) then
do;
if ^missing("&dskeep") & ^missing("&varkeep") then
do;
if indexw(compress(upcase("&dskeep")),compress(memname),",") > 0
& indexw(compress(upcase("&varkeep")),compress(name),",") > 0 then
output;
end;
else if ^missing("&dskeep") and missing("&varkeep") then
do;
if indexw(compress(upcase("&dskeep")),compress(memname),",") > 0 then
output;
end;
else if missing("&dskeep") and ^missing("&varkeep") then
do;
if indexw(compress(upcase("&varkeep")),compress(name),",") > 0 then
```

```
output;
end;
else
do;
output;
end;
end;
keep libname memname name type varfmt where_clause;
run;

%*-------------------------------------------------------------------------
------------------*;
%*- Step 02: Creating latest date from each date variable and outputting into
separate dataset-*;
%*-------------------------------------------------------------------------
------------------*;

data _null_;
set date_metadata;
call symputx ("_dsn",memname);
call symputx ("_var", name);
call symputx ("_type", type);

%* If data is in Character then convert into number using user defined
format;
if lowcase(type) = "char" then
call execute(
< Use CALL EXECUTE statement to run set of code that creates latest date from
each date variable. Finally add below append code at the end of CALL EXECUTE
block. For more details please see CALL EXECUTE section above. >

<proc append base = __summary  data = &_var._&_dsn. force; >
<run;>
)
%* If date is in numeric then no conversion is required;

else
call execute (
< Use CALL EXECUTE statement to run set of code that creates latest date from
each date variable. Finally add below append code at the end of CALL EXECUTE
block. For more details please see CALL EXECUTE section above.>

<proc append base = __summary  data = &_var._&_dsn. force; >
<run;>
)
%*-------------------------------------------------------------------------
-----------------------------------------------------------------*;
%*- Step 03: Finding latest date from all intermediate datasets created at
step02 and generating single dataset with subject level latest date -*;
%*-------------------------------------------------------------------------
-----------------------------------------------------------------*;

< Write a data step or proc step and create maxdate0 dataset with subject
latest date. For more details please check SELECTING LATEST DATE FROM ALL
DATASETS section above.>
```

```
%*-------------------------------------------------------------------------
-----------------------------------------*;
%*- Step 04: Creating final dataset with subject latest date and dataset and
variables names that contains latest date -*;
%*-------------------------------------------------------------------------
-----------------------------------------*;

data &outdsn.;
set maxdate0;

< Create source variable with list of variables names these contains latest
date. >
< If multiple datasets and variables contains subject latest date then
concatenating them with delimiter >

run;

%mend finding0latest0date;

%*----------------------*;
%*- Calling Code example *;
%*----------------------*;

%finding0latest0date
(libname = sdtms
,idvar =  usubjid
,varsuffix =  dtc|dti
,varcfmt = is8601da|is8601da
,dskeep = %str(ae,dm,cm,ds)
,varkeep =
,dsdrop =
,vardrop =
,outdsn = max0date
,outvarname = lkadt
,outvarfmt = is8601da
,fillterds = ae
,filltercond = %str( ^missing(epoch))
);
```

CONCLUSION:

For each date variable, this macro/template code defines the algorithm once. It applies to all date variables using call execute and data step to reduce the length of code and increase efficiency.

REFERENCES

CALL EXECUTE made easy for SAS data-driven programming

https://blogs.sas.com/content/sgf/2017/08/02/call-execute-for-sas-data-driven-programming/

ACKNOWLEDGMENTS

The authors would like to thank the management teams for their advice on this paper/presentation.

CONTACT INFORMATION

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

Raghava Pamulapati
Merck & Co., Inc.
126 Lincoln Avenue
Rahway, NJ 07065, USA
Phone: 732-594-5819

e-mail: raghava.pamulapati@merck.com

Sandeep Meesala
Merck & Co., Inc.
126 Lincoln Avenue
Rahway, NJ 07065, USA
Phone: 732-594-7298

e-mail: sandeep.meesala@merck.com

TRADEMARK