A Practical and Efficient Approach in Generating AE (Adverse Events) Tables within a Clinical Study Environment

Jiannan Hu Vertex Pharmaceuticals, Inc.

Abstract

When a clinical trial is at the stage of the submission to regulatory authorities, or at the investigation of the interim analysis, most of the cases, safety analysis plays a key part in deciding whether the trial will be ongoing and drug is approved or not according to whether the drug is safe or not. Adverse event analysis is a pivotal piece in the safety analysis, and it is common in almost every trial, and every clinical study report. The adverse events related to study drug will be summarized, as well as by severity, by relationship. They are tabulated by MedDRA system organ class (SOC) and preferred term (PT), and each subject is counted only once, and the sorting is by descending frequency of SOC and PT. And the AE analysis is different from other safety analysis and efficacy analysis. However, there are similarities among the adverse event analysis within a study. There are challenges for a statistical programmer to provide high quality AE reports with a limited time frame. One approach is to use macros to modularize repeated work, and thus save the development time. This paper will discuss this approach in detail and share the code in getting the work done in an efficient way.

Introduction

In the drug development within pharmaceutical industry, clinical reports are created during the trials and provided to medical monitors or medical committee for trial monitoring, or after the clinical database is locked for regulatory agencies for approval. To generate accurate tables which explain the clinical data, and summarize it right, a good approach is conducted in the programming efforts to provide high quality AE tables for final CSR.

Step 1. Analyze AE tables

Some clinical trials may have different phases, and AE tables may be required to be summarized for each phase. Not surprisingly, there are tables by relationship, by severity, by seriousness across different treatment groups, and there are incidence and prevalence tables. The tables could be classified into different groups based on the similarities. This will be the base to design macros.

Step 2. Modularize Macros

In generating tables, there are some practices using open macro code to create a table, the macro statements are not encapsulated in one block to have a macro which likes a function to call. In this study, the macros are developed and for each production program, only one statement calling the macro by specifying different condition flag and timing
flag in verifying all the production tables. This way, high quality is achievable and efficiency is manageable. Since after one table matches the production table, then the same logic should work for the rest tables with the same shell and layout. On quality control side, this approach did catch some errors in some production tables.

After classifying the tables, the macro modules are designed to meet the table requirements to streamline the table creation process. Some common function for different table types may be put in macros. The final macro for one table type may contain multiple macros.

**Step 3 . Design the Flow Chart or Algorithm of Macros**

Flow chart has been widely used in the large complex software development. For some complex macros, flow chart will help to illustrate the workflow of the macro and will provide guideline in implementing and debugging the code. This will help to add comments in the program to document the macro, then make it easier to maintain and use.

**Step 4 . Develop and Test Macros**

At the stage of macro development, it is very useful to turn on some macro options, such as

```plaintext
Options symbolgen mprint;
```

After macro is released and is put in production, turning off these two options eliminate the log about macro and macro variables.

```plaintext
Options nosymbolgen nomprint;
```

**Algorithm**

There are different approaches to creating the summary data for a table with total counts at the last row and last column. Different programmer could have different preference and style. Below, the algorithm is demonstrated in two cases. For each case, multiple macros are created, which can be reused for multiple table sets with almost no modifications. You can read the flow of the approach by reading the program comments.

**Case 1. Common Macros**

```plaintext
%macro getPopulationDs(inDs=, outDs=) ;

***************Created data for the column subtotal and total;
data &outDs;
   set &inDs;
   output;
   trtan = 5;
   trta = "Total";
```
output;
run;

%mend getPopulation;

The following macro is to calculate the denominator for each column. First, it gets the number of treatment groups. Then statement

%let colNum = &colNum;

will remove the leading space in macro variable colNum, which will be used to create other macro variables storing the denominators. Please be aware the data here is ADSL, subject level analysis dataset, where each subject only has one record.

**************get denominators -- number for each group;
%macro getNum(inDs=);
proc sql;
  select count(distinct trtan) into: colNum from &inDs where trtan ne .;
  %let colNum = &colNum;
  select trta into: col1 - : col&colNum from (select distinct trta, trtan from &inDs) order by trtan;
  select count(usubjid) into: dcol1 - : dcol&colNum from &inDs
     group by trtan;
quit;

**************create the column header;
%do i = 1 %to &colNum;
  %global count&i;
  %let count&i = %str((N=&&dcol&i));
%end;
%mend getNum;

Case 2. Frequency Table by Most Severe AE Macro

One set of adverse event tables is the count and percentage of different adverse event characteristics distribution of combinations of treatment group, adverse events and study phases. The worst AE is counted for each subject. The following macro gets the numbers ready for reporting. The input parameters inDs is the input data, which should be a subject level data. Every subject is counted only once in this kind of table. inVar is the variable indicating the adverse event characteristics a table will describe. Different clinical trials could have different adverse events if the therapeutic area is different. Three different arrays are used here, a numeric array arr1, a character array arr2, and an array arr3 which holds the values, not the variables as the other two.

%macro getCountPerc(inDs=, inVar=);
proc freq data = &inDs noprint;
  table trtan*&inVar/out=sevCount;
run;
```sas
proc sort data = sevCount;
    by trtan &inVar;
run;

proc transpose data=sevCount out=sevCountT prefix=col_;
    by &inVar;
    var count;
    id trtan;
run;

data final;
    set sevCountT;
    array arr1(*) col_1 col_2 col_3 col_3_1 col_4 col_5;
    array arr2(*) $12 col_1c col_2c col_3c col_3_1c col_4c col_5c;
    array arr3(6) _temporary_ (&dcol1 &dcol2 &dcol3 &dcol4 &dcol5 &dcol6);
    do i = 1 to 6;
        if arr1(i) = . then arr2(i) = "-";
        else
            arr2(i) = cat(arr1(i), 3.), " (", put(arr1(i)/arr3(i)*100, 5.1), ")");
    end;
run;

%mend getCountPerc;

Case 3. Frequency Table by SOC, Preferred Term, and Treatment Group Macro

Another set of adverse event tables is the count and percentage of adverse events by System Organ Class, Preferred Term, and Treatment Group. The rules in counting the different adverse event characteristics is that if a subject has multiple events within a system organ class or preferred term, the subject is counted once. And the table is sorted in descending frequency of the total number for each system organ class and preferred term.

%macro aeRepData(adsl=, inDs=, eventCond=, aeDesc=, aeOutcome=, outDs=report);

**********************create pooled data for counting;

%if &eventCond ne %then %let eventCond = %str(and &eventCond);

data aeFin(keep = usubjid aedecod aebodsys &aeOutcome trta trtan);
    set &inDs;
    where aedecod ne "" &eventCond;
    if missing(aebodsys) then aebodsys="Uncoded";
run;

**********************************count statistics;

proc sql ;
**************************This data is to count the unique subjects;
**********It is required in the table;

    create table aeSubj as select distinct usubjid, trtan, trta,
                max(&aeOutcome) as maxVal from aeFin group by usubjid;
```

Coders' Corner
NESUG 2011
create table aeSoc as select usubjid, trtan, trta, aebodsys, 
     max(&aeOutcome) as maxVal from aeFin group by trtan, 
     trta, usubjid, aebodsys;

create table aePt as select usubjid, trtan, trta, aebodsys, 
     aedecod, max(&aeOutcome) as maxVal from aeFin 
     group by trtan, trta, usubjid, aebodsys, aedecod;

quit;

 Count number of subjects;

proc freq data = aeSubj noprint;
     table trtan*trta*maxVal/out=subjCount;
run;

proc transpose data=subjCount out=subjCountT prefix=col_;;
     by trtan trta;
     var count;
     id maxVal ;
run;

 Count SOC;
 ***get total;
 ***99 is used to represent the total category;

data aeSocFin;
     set aeSoc;
     output;
     maxVal = 99;
     output;
run;

proc freq data = aeSocFin noprint;
     table trtan*trta*aebodsys*maxVal /out=socCount;
run;

proc sort data = socCount;
     by trtan trta aebodsys;
run;

proc transpose data=socCount out=socCountT prefix=col_;;
     by trtan trta aebodsys;
     var count;
     id maxVal ;
run;

 sort SOC;

proc sort data = socCountT;
     by trtan trta descending col_99;
run;

data socCountT;
     set socCountT;
     by trtan trta descending col_99;
retain ind 0;
    ind + 1;
    run;

******************************************************************************
***get total;
data aePtFin;
    set aePt;
    output;
    maxVal = 99;
    output;
    run;

proc freq data = aePtFin noprint;
    table trtan*trta*aebodsys*aedecod*maxVal /out=ptCount;
    run;

proc transpose data=ptCount out=ptCountT prefix=col_;;
    by trtan trta aebodsys aedecod;
    var count;
    id maxVal;
    run;

*************Sort the preferred term;

proc sort data = ptCountT;
    by trtan trta aebodsys descending col_99;
    run;

data ptCountT;
    set ptCountT;
    by trtan trta aebodsys;
    retain indPt;
    if first.aebodsys then indPt = 1;
    else indPt + 1;
    run;

******************************************************************************
*********The three parts in the table;
data ae4Count;
    set subjCountT(in=a) socCountT(in=b) ptCountT(in=c);
    length cat $200;
    if a then
        do;
            ind = 0;
            cat = "&aeDesc";
            aebodsys = "";
        end;
        if b then cat = aebodsys;
        else if c then cat = cat(" ", strip(aedecod));
    run;

******Populated the sorting key;

proc sql;
    create table countFin as select * from
(select *, max(ind) as indSoc from ae4Count group by trtan, aebodsyst) order by trtan, indSoc, indPt;

quit;

******************************get final report data;
******get denominators for each column;

proc sql;
create table test as
    (select distinct trta, trtan from &adsl) order by trtan;
quit;

data test2;
    set test;
    length trtaq $200;
    trtaq = quote(strip(trta));
run;

proc sql;
    select trtaq into: trtArr separated by " " from test2;
    select count(usubjid) into: denomArr separated by " " from &adsl group by trtan;
quit;

data &outDs;
    set countFin;
    array trtArr(4) _temporary_ (&trtArr);
    array denomArr(4) _temporary_ (&denomArr);
    array arr1(*) col_1 col_2 col_3 col_4;
    array arr2(*) $12 col_1c col_2c col_3c col_4c;
    drop i j _label_ _name_; do j = 1 to dim(trtArr);
        if trta = trtArr(j) then do;
            do i = 1 to 4;
                if arr1(i) = . then arr2(i) = "-";
                else
                    arr2(i) = cat(put(arr1(i), 3.), ", ", put(arr1(i)/denomArr(j)*100, 5.1), " )");
            end;
            leave;
        end;
    end;
run;
%mend aeRepData;

******make the report;

%macro report(inDs=, width=);
******************************report part;
%do i = 1 to 6;
    title9 "&col&i (N=&dcol&i)";
    proc report list nowd missing headline headskip split = "*"
        data=&inDs(where = (trta = "&col&i"));
        column ("_____", indSoc indPt catalyst "Relationship"
define indSoc /order noprint;
define indPt /order noprint;
define cat /display width=&width "System Organ Class*Preferred Term" left flow;
define col_1c /display width=14 center "Not Related*n (%)";
define col_2c /display width=14 center "Unlikely*Related*n (%)";
define col_3c /display width=14 center "Possibly*Related*n (%)";
define col_4c /display width=14 center "Related*n (%)";
break after indSoc/skip;
run;
%end;
%mend report;

Acknowledgements

The author would like to acknowledge the encouragement and support from Xiangchen (Bob) Cui, associate director of Statistical Programming, Vertex Pharmaceuticals, Inc.

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries.

References


Author Contact

Jiannan Hu, Ph.D.

Biometrics
Vertex Pharmaceuticals, Inc.
88 Sidney St.
Cambridge, MA 02139

Tel: (857)919-0220
Email: hujiannan@gmail.com