PharmaSUG 2024 - Paper AP-145

Integrity, Please: Three Techniques for One-Step Solution in Pharmaceutical Programming

Jason Su, SAS Institute

ABSTRACT

In pharmaceutical world, data are frequently reorganized or summarized before the final tabulation with tables or display with listings. SAS programmers typically use a straightforward method composed of three (3) steps: split the data set into different data sets, process them individually, and merge these data sets back, which I called Split-and-combine method. Despite its intuitive structure, the route comes with computational efficiency issues and requires extra efforts to make sure the data structures are compatible in these data sets for avoidance of undetected erroneous results. Here, I summarized three (3) field-tested methods including Do-loop of Whitlock (DoW), self-interleaving techniques, and direct outputting, to circumvent this method with primarily one (1) DATA step. The new programs usually contain much less steps and more importantly, less or no intermediate data sets, which greatly saves system resources and reduces the later maintenance efforts. The principle is demonstrated with real-world examples and common scenarios for the three techniques summarized.

INTRODUCTION

In clinical trials, all data need to be reorganized or summarized before the final data tabulation with tables or display with listings, and in many cases the data manipulation is self-based, instead of other data being needed from extra sources. In the healthcare or pharmaceutical field, the popular examples are:

- Creating new variable values such as an extra artificial treatment group and combing all groups into one file;
- Within a BY-group, deriving the summary data such as baseline-based calculation results, to *the last record* or *the last subset of records*;
- Within a BY-group, adding the summary data such as a flag variable, to other than the last record;
- Inserting a new record per BY-group.

In dealing with these tasks, programmers typically resort to the following three (3) steps:

- Firstly, split the data set into different data sets, each with same characteristics, such as treatment numbers;
- Secondly, if needed, process them individually, such as adding the flag variable;
- Lastly, merge these data sets back usually with SET statements or match-merge in DATA step.

I call this commonly used algorithm “split-&-combine”, which is not efficient, albeit intuitive, due to the creation, maintenance, and merging of these intermediate data sets. Additionally, the seemingly easy method of splitting may negatively impact the next step derivation when some other information is needed from other observations in the original data set, which will be shown in the following examples. Ideally, the good solution would be one-step operation, namely, finishing all requirements in as little as one (1) step. The paper summarized three (3) straightforward techniques that can handily deal with these jobs as in the following sections.

DO-LOOP-OF-WHITLOCK (DOW)

The DoW technique was proposed and later explored by Henderson, Whitlock, Dorfman, and others (Dorfman, 2008). It is essentially a BY-group processing with the SET statement placed within a DO-UNTIL-loop. Here the break-event is the start or the end of the BY-group which is inherently checked in the loop for each observation, neatly rendering superfluous the explicit RETAIN statements and

reinitialization to zero for these new variables. Due to these great features, DoW is extremely powerful to handle BY-group based summarization tasks and I have found that DoW-based one-step solution can be handily dished out to three (3) tasks, which are commonly seen in the clinical trial field.

PERFORMING MULTIPLE CALCULATIONS IN ONE STEP AND OUTPUTTING ONLY ONE RECORD PER BY-GROUP

One common job is to summarize all data from a certain BY group, for example, subject identifier (SUBJID), treatment arm (TRTA) , or adverse event coding variables (AEBODSYS/AEDECOD) and it may come in various forms including counting the event, finding the average, collecting all entry values, etc. It is well known that when GROUP BY clause is used, PROC SQL can summarize data based on BY-groups, such as averaging, summing etc. However, the function is rather limited. For example, PROC SQL cannot collect all values from individual records into one record, as far as I know. In this point, DATA step is all advantageous and capable of any complex derivations. Being a special form of the DATA step, DoW has the ability to summarize the records in various ways as long as summarization is for the same BY-group, and output only one (1) row of data with the inherent OUTPUT statement at the end of the DATA step.

Job example: Suppose that we are to 1) count the number of events, and 2) collect the list of unique subject IDs (USUBJIDs) from data set ADAE per BY-group of TRTA, AEBODSYS & AEDECOD.

Analysis: It is well known to many programmers to take advantage of the GROUP BY clause in PROC SQL to individually deal with each task, followed by match-merge. However, there are many steps involved whereas DoW can finish all these in one simple step, as follows.

```
data ae2;
    length subj_list $2000;

    **SUBSTEP 1;
    if _n_=1 then
        do;
            declare hash h();
            h.definekey('usubjid');
            h.definedone();
        end;

    **SUBSTEP 2;
    do until (last.aedecod);
        set ae;
        by trta aebodsys aedecod;

        event_num=sum(event_num,1);
        _rc=h.ref();
    end;

    **SUBSTEP 3;
    do while (h.do_over()=0);
        subj_list=catx(', ', subj_list,usubjid);
    end;

    h.clear();

    keep aebodsys aedecod trt: event_num subj_list;
run;
```

Brief explanations:

- To collect the unique USUBJIDs, hash object is used with DoW technique.
- In SUBSTEP 1, it is the pre-DoW stage and hash object was created. For efficiency, it was created only once in the very beginning but got cleared after each BY-group. Additionally, since KEY and DATA are same, DEFINEDATA was performed inherently and there is no need to explicit write it out;
- In SUBSTEP 2, a regular DoW construct was set and in each BY-group, each record/event was counted and the USUBJID was loaded into the hash table via REF() method. Due to the nature of the hash object, only unique USUBJID was stored;
- In SUBSTEP 3, it is the post-DoW stage where the hash table is scanned with DO_OVER() method and all the entries are dumped into the variable SUBJ_LIST with inherent alphabetic order. Lastly a CLEAR() method is used to deplete the hash object and reset it for the next BY-group. The current record was written to the data containing both the number of event (EVENT_NUM) and the list of unique USUBJIDs (SUBJ_LIST) for the current BY-group.

TRANSPOSING DATA IN A BY-GROUP BASED ON ONE VARIABLE

In clinical trial, another common job is to recombine data in a BY-group for juxtaposing or displaying them side-by-side in one table. Skipping the explicit RETAIN & reinitialization statements, DoW has the natural ability to inherently retain and reinitialize the new variable values around the DO-loop, which is perfect for flatten the data, i.e. recombining multiple rows of data.

Job example: Suppose that we want to display per USUBJID:

1. When PARAMCD='TRTC', the value of one variable (EXTRT) and
2. When PARAMCD='EXDOSE', the multiple pairs of values from two other variables (EXSTDTC & AVALC).

Analysis:

- DATA-step instead of TRANSPOSE PROC has to be used because the 2nd PARAMCD value ('EXDOSE') have many instances of the same BY variable (USUBJID).
- With the usual split-&-combine method, two separate data sets must be created individually based on the two PARAMCD values (SPLIT), and then merged into one with the key variable (USUBJID) following 1-to-many matching strategy (COMBINE). The disadvantage is that too many steps and intermediate data sets are involved. A quick analysis shows that the split is totally dispensable, and with DoW technique, the whole job can be completed in only one (1) data step after a quick sorting step as follows.

```
proc sort data=adb.adex out=adex;
    by usubjid descending paramcd;
    where paramcd in ('EXDOSE' 'TRTC');
run;

Data final;
    length col1 - col4 $300;

    do until (last.usubjid);
        set adex;
        by usubjid;

        if paramcd='TRTC' then
                do;
                    col1=usubjid;
                    col2=left(extrt);
                end;
            else
```

```
                do;
                    col3=exstdtc;
                    col4=avalc;
                    output;
                end;
    end;
run;
```

Brief explanations:

1. The implementation is taking advantage of DoW's ability to naturally retain the values of the new variables (COL1-2) within each BY-group.
2. Due to the sorting step with DESCENDING keyword, the two variables (USUBJID & EXTRT) for PARAMCD="TRTC" are always showing up first in each BY-group but will stay in the Program Data Vector (PDV). Only in the 2nd do-loop after ELSE for PARAMCD='EXDOSE', all four (4) new variables COL1-4 are output in one row per USUBJID, which is like the right join in PROC SQL.
3. If the full join is desired, an additional OUTPUT statement can be easily inserted right before the END statement of the outer do-loop. To avoid the potential duplication, I usually employ a temporary flag in the 2nd do-loop and only when it is null output the result with this additional OUTPUT statement.

INSERTING A NEW RECORD PER BY-GROUP

Another common job is to insert a new record at the end of BY-group. DoW construct is simply to put two (2) explicit OUTPUT statements: one inside of the do-loop and the other right outside of the do-loop after data manipulation.

Job example: In each BY-group (TRTN & AVISITN) we are to have two variable values (CUM_PCT & AVAL) in the new record and combine the new record with the original records.

Analysis: Usually the split-&-combine algorithm must contain these three (3) steps: 1) Sorting and outputting the records having unique TRTN & AVISITN; 2) Setting CUM_PCT & AVAL to certain values; 3) Adding back these records to the original file with the BY-group. All these can be replaced with one DATA step with DoW construct as follows,

```
data ds_2;
    **SUBSTEP 1;
    do until (last.avisitn);
        set ds_1;
        by trtn avisitn;

        output;
    end;

    **SUBSTEP 2;
    cum_pct = 0;
    aval = 10;
    output;
run;
```

Brief explanations:

- In SUBSTEP 1, each observation in the BY-group is simply output without any modification;.
- In SUBSTEP 2, the two variables (CUM_PUT & AVAL) are reset while other variables including the BY-group variables (TRTN & AVISITN) remain intact. All these variables are written as the new records in the output data set. Please note that since the new record retains the values of other variables from the last record in the PDV, you can set them as missing right before the 2nd OUTPUT statement.
- Unlike in this simplified example, in many cases the new record is the summary result from the other records in the BY-group, such as the derivation of DTYPE variable but the idea remains the same.

In sum, DoW technique is extremely powerful in dealing with the popular group summarization jobs including: 1) Performing multiple calculations in one step and outputting only one record per BY-group 2) transposing data in a BY-group based on one variable; 3) Inserting a new record per by-group.

Please note that this technique requires the BY-group to be consistent, i.e. if different BY-group variables are used during split/combine, these one-step solutions might not get the same results, and the split-&-combine algorithm might be used. Additionally, since DoW essentially is a BY-group processing with SET statement, all these jobs can be accomplished with the non-DoW form as well, albeit usually in a clumsy and less efficient manner, as benchmarked in a previous paper against the non-DoW version (Su, 2020). Therefore, the bottom line is that when the manipulation is based upon a common BY-group, the split-&-combine algorithm is usually lengthy and inefficient, and should be easily replaced with DoW techniques.

SELF-INTERLEAVING TECHNIQUE

Sometimes we need to add the group summarization data back to all or a subset of the group records and use other techniques such as self-interleaving since the single-passing one-step DoW technique is lacking the “looking backward” capacity. Self-interleaving technique was first proposed by Howard Schreier (Schreier, 2003) and its ability is achieved via two passings of the same BY-group:

1. In the 1st passing, data are analyzed but not output. Typically, group summarization is performed, and new variables are calculated and retained for the next passing such as flag variables. Additionally, if needed, a temporary ID variable can be set to “remember” some records and retained for the next passing.
2. In the 2nd passing of the same BY-group, data are finally output. Since the calculated value of this new variable from the whole group is still kept in the PDV, it is simply output with other variables. If the ID variable is in place for identifying the specific records for the new summarization variables, it will be employed to select these records.

In pharmaceutical programming I found the technique is particularly good at these two jobs.

DERIVING & ADDING THE GROUP SUMMARIZATION DATA TO ALL RECORDS

Job example: Suppose that we need to collect assessment (AVALC) from the BY-group records where parameter category variable (PARCAT1) is equal to the string "FINDING" and add the final collection (FIND) into all records where two flag variables (SAFFL & ANL01FL) are flagged. ID variables are not needed since it applies to all appropriate records.

Analysis: The familiar split-&-combine algorithm might have these steps: First, subsetting the data into a new data set with the condition PARCAT1= "FINDING"; Second, collecting the AVALC values for each BY-group into another data set; Third, merging back the data and outputting the appropriate records. The self-interleaving technique only has one step as follows,

```
data adeg2;
    length find $400;
    retain find;

    set adeg (in=a)
        adeg;
    by trt01p siteid usubjid atpt egdtc;
```

```
    **SUBSTEP 1;
    if first.egdtc then call missing(find);

    **SUBSTEP 2;
    if a then
        do;
            if parcat1 = "FINDING" then find = catx("; ", find,avalc);
        end;
    **SUBSTEP 3;
        else if saffl='Y' & anl01fl='Y' then output;

run;
```

Brief explanations:

- In SUBSTEP 1, the retained variable must be re-initiated upon a new BY-group;
- In SUBSTEP 2, this is the 1st passing and all the appropriate AVALC values are collected;
- In SUBSTEP 3, this is the 2nd passing of the same BY-group, the new variable FIND is retained and added to all records but only a subset of records are output with flagged SAFFL & ANL01FL.

DERIVING & ADDING THE GROUP SUMMARIZATION DATA TO SPECIFIC RECORDS

Job example: Sometimes we desire only specific records to be added with the new variable. In this popular example we are to: First derive a new flag variable (ANL01FL) from the appropriate records, where assessment (AVALC) is not missing & treatment start date (TRTSDT) is not greater than assessment date (ADT); Then add it only to the last non-missing assessment record in BY-group of universe subject ID (USUBJID), parameter code (PARAMCD) & assessment visit (AVISIT), that is, with the greatest assessment day (ADY).

Analysis: The split-&-combine algorithm is commonly provided as 1) subsetting into a new data set, 2) processing the data set, 3) merging back after sorting, as in the following program:

```
**SUBSETTING INTO A NEW DATA SET;
proc sort data=adLB_1(keep=usubjid paramcd avisit adt avalc trtsdt) out=
anl01fl_1(drop=avalc trtsdt) nodupkey;
  by usubjid paramcd avisit adt;
  where avalc ne '' and .z<trtsdt<=adt;
run;

**PROCESSING THE DATA: OBTAINING LAST NON-MISSING OBS;
data anl01fl;
  set anl01fl_1;
  by usubjid paramcd avisit;

  if last.avisit;
run;

**MERGING BACK;
proc sort data= adlb_1 out=adlb_2;
  by usubjid paramcd avisit adt;
run;

data adlb_3;
  merge adlb_2(in=a) anl01fl(in=b);
  by usubjid paramcd avisit adt;

  if b then anl01fl='Y';
```

```
run;
```

The program with self-interleaving technique will have one-step processing after a simple sorting, as follows:

```
**SUBSTEP 1;
proc sort data=adlb_1;
by usubjid paramcd avisit adt;
run;

data adlb;
    length _last_assess_day 6;
    retain _last_assess_day;

    **SUBSTEP 2;
    set adlb_1 (in=a)
        adlb_1;
    by usubjid paramcd avisit;

    **SUBSTEP 3;
    if first.avisit then call missing(_last_assess_day);

    **SUBSTEP 4;
    if a then
            do;
                if .<trtsdt<=adt & ^missing(avalc) then
    _last_assess_day=adt;
            end;

        **SUBSTEP 5;
        else
            do;
                if _last_assess_day=adt & .<trtsdt<=adt & ^missing(avalc)
    then anl01fl='Y';
                output;
            end;

    drop _last_assess_day;
run;
```

Brief explanations:

- In SUBSTEP 1, sorting the data with the ascending order of the date (ADT) in the by-group (USUBJID PARAMCD AVISIT), which will ensure that in the SUBSTEP 4 the temporary ID variable _LAST_ASSESS_DAY is always replaced by the higher valid ADT values;
- In SUBSTEP 2, self-interleaving step with data set option IN for labelling the 1st passing;
- In SUBSTEP 3, re-initiating the retained ID variable _LAST_ASSESS_DAY for each BY-group;
- In SUBSTEP 4, it is the 1st passing, the ID variable _LAST_ASSESS_DAY always is replaced by the higher valid ADT values and the final value should be from the record containing the highest ADT. There is no output in this passing;
- In SUBSTEP 5, it is the 2nd passing, the ID variable _LAST_ASSESS_DAY is retained from the 1st passing for identifying the correct record via matching with ADT, followed by ANL01FL flagging and output. Please note that the program assumes that the flag variable ANL01FL already exists in the source dataset. Otherwise, an additional statement must be added after IF-THEN logic like “else call

missing(anl01fl);" for us to avoid the unintentional retaining of this newly derived variable in the PDV.

- Multiple records will be flagged within the BY-group that have the same date with the largest ADT value; If only one such record is desired, additional filter has to be used.

Self-interleaving technique can also be replaced by double DoW technique. However, it has the advantage of presenting only one BY statement, therefore avoiding the error-prone element of two BY statements which might be erroneously different in the double DoW technique on top of the DO-loops. Additionally, in some cases the self-interleaving technique is more efficient as in the above example since one of the sorting steps is removed. However, due to the double passing of the same data sets, the self-interleaving technique might not be significantly more efficient than the split-&-combine algorithm. Even though, it does enjoy a much more succinct program and might substantially alleviate the later debugging or maintenance burden in the future.

Please note that the BY-group must be ready and remain unchanged for the one-step solutions supported by DoW or self-interleaving techniques as above. Otherwise, additional preparatory steps must be performed before the summarization.

DIRECT OUTPUTTING

OUTPUT statements cause the current data in the PDV to be copied into the designated datasets in the DATA statement. Every programmer seems to have known this, however, as in the following example where the OUTPUT statements are used to split the original data into the intermediate data sets, direct outputting is good enough and there seems no point to first create these data sets and then combine all of them together.

Job example: Suppose that we need to exclude all original records with two parameter code (PARAMCD) values ('HEIGHT' & 'BMI') and replicate the proper records with conditions (LASFL="Y" or ABLFL="Y") after resetting assessment visit (AVISIT) & assessment visit number (AVISITN) values for the next step analysis. The original program splits these records into two separate data sets (ADVS_2 & ADVS_L) and then combines them together, as follows,

```
data advs_2
      advs_L;

    set advs_1;
    output advs_2;

    if lasfl="Y" then do;
       avisitn = 99;
       avisit = 'End of Study';
      output advs_L;
    end;

    if ablfl="Y" then do;
       avisitn = -4.5;
       avisit = 'Baseline';
      output advs_L;
    end;
run;

data advs;
    set advs_2 advs_L;

     if PARAMCD in ('HEIGHT', 'BMI') then delete;
run;
```

A quick analysis indicates that the two intermediate data sets (ADVS_2 & ADVS_L) are not required and all these transformations/filtering can happen in one (1) step simply with IF-THEN logic and multiple OUTPUT statements, as follows:

```
data advs;
    set advs_1;

    if ^(PARAMCD in ('HEIGHT', 'BMI') then
        do;
            output;

           if lasfl="Y" then
                 do;
                     avisitn = 99;
                     avisit = 'End of Study [a]';
                     output;
                 end;

          if ablfl="Y" then
                 do;
                     avisitn = -4.5;
                     avisit = 'Baseline';
                     output;
                 end;
   end;

run;
```

The modified program creates the exactly same data set while has fewer steps and eliminates the intermediate data in the memory, which significantly increased program efficiency and reduced the system resource usage by 100% or more. It goes without saying that the benefit of direct outputting becomes much more significant with the size of the source data jumps to millions or more.

To further highlight the prevalence of the split-&-combine method and the easiness of one-step replacement, we provide another example. Suppose that we are to extract different groups of records, assign respective treatment numbers (TRTX) to them based on different variable values (RFASFL = 'Y', NRFASFL = 'Y', PFASFL = 'Y' & TRTA = 'BSN') and create another data set for analysis. The original program is using the “split-&-combine” algorithm as follows,

```
data vs12 vs3 vs4 vs5;
    set adb.advs;

    if RFASFL = 'Y' then do;
          if trtan = 1 then TRTX = 2;
          else if trtan = 2 then TRTX = 1;
          output vs12;
        end;

    if NRFASFL = 'Y' and trtan ne . then do;
      TRTX = 3;
      output vs3;
    end;

    if PFASFL = 'Y' and trtan ne . then do;
      TRTX = 4;
      output vs4;
    end;
```

```
    if trta = 'BSN' then do;
      TRTX = 5;
      output vs5;
    end;
run;

data advs_1 (where = (trtx ne .));
  set vs12 vs3 vs4 vs5;
run;
```

Like the analysis in the above example, these four intermediate data sets are not required at all. The program of direct outputting solution is simply implemented as follows,

```
data advs_1;
    set adb.advs;

    if rfasfl = 'Y' then
          do;
                trtx = ifn(trtan=1,2,ifn(trtan=2,1,.));
                if ^missing(trtx) then output;
          end;

    if ^missing(trtan) then
        do;
            if nrfasfl = 'Y' then
                do;
                      trtx = 3;
                      output;
                end;
            if pfasfl = 'Y'  then
                do;
                      trtx = 4;
                      output;
                end;
        end;

    if trta = 'BSN' then
                do;
                      trtx = 5;
                      output;
                end;
run;
```

The new solution is more straightforward, easy to follow and more efficient, while cutting in half system resource usages.

A MORE COMPLEX EXAMPLE: INSERTING RECORDS IN THE MIDDLE, AND MODIFY THE REST OF RECORDS IN THE GROUP

Direct outputting with its simple nature is not usually used for BY-group processing like DoW and self-interleaving techniques. Sometimes, however, direct outputting can deal with complex issues quite neatly and we don't have to resort to the multi-step split-&-combine algorithm.

Job example: Suppose that we have a data set where the records are not sorted but grouped with group name variable (GROUPNAME) instead, as created in this DATA step.

```
%let new_entry =# Trt A = Placebo/Placebo
                # Trt B = Construct A /Construct B
```

```
                # Trt E = Construct A /Construct C
                ;

data test;
    input groupname $ series :$3. wholeline $200.;
    infile datalines truncover dlm=', ';

    datalines;
    T140102, T01, Table 14.1.2
    T140102, T02, Important Protocol Deviations
    T140102, T03, Intent-to-Treat Analysis Set
    T140102, F01, Important protocol deviations were defined as leading to study discontinuation.
    T140102, F02, Percentages were based on ITT Analysis Set within each treatment and overall.
    T140103, T01, Table 14.1.3
    T140103, T02, Demographics
    T140103, T03, Safety Analysis Set
    T140103, F01, Body Mass Index (BMI) was calculated as weight (kg) divided by squared height (m).
    T140103, F02, Percentages were based on Safety Analysis Set within each treatment and overall.
    F140105, T01, Table 14.1.5.2
    F140105, T02, Medical or Surgical Treatment Procedures Conducted During the Study Period
    F140105, T03, Safety Analysis Set
    F140105, F01, Subjects may have had more than one medical or surgical treatment procedure.
    ;
run;
```

We are supposed to insert three records stored in macro variable (&NEW_ENTRY) into variable WHOLELINE after the record containing the string “Safety Analysis Set” as in the rows ‘T140103’ ‘T03’ and ‘T140105’ ‘T03’, and populate another two variables (GROUPNAME & SERIES) in this way: GROUPNAME is retained from the previous record; SERIES set as “F01”, “F02” & “F03”, respectively. Additionally, the “F”-starting values in variable SERIES in the later records in the same group are to be adjusted accordingly, e.g. F01 -> F04, F02 -> F05, etc. since three (3) new records are inserted ahead of them. The split-&-combine algorithm would go this way:

```
**SPLITTING THE TARGET GROUPS & INSERTING INTO NEW DATA;
data test_new;
    *PARSING &new_entry;
    retain _replacelist "&new_entry.";

    set test;

    if prxmatch('/^Safety Analysis Set/i',wholeline) then
                do _i=1 to countw(_replacelist,'#');
                    wholeline = scan(_replacelist,_i,'#');
                    series = cats("F0",_i);
                    output;
                end;
   drop _:;
run;

**SELECTING THESE TARGET GROUPS INTO A NEW DATA SET;
proc sql noprint;
    create table test_1 as
    select *
    from test
    where prxmatch('/^Safety Analysis Set/i',wholeline)
    ;
quit;

**LABELING THE TARGET GROUPS WITH GROUPNAME2, TO ADJUST THE SERIES VALUE;
proc sql;
```

```
    create table test_2 as
    select *
    from test natural left join
        (select distinct groupname,groupname as groupname2
         from test_1
        )
    order by groupname,series
    ;
quit;

**ADJUSTING THE SERIES VALUE AFTER 3 NEW OBS IN THE LATER RECORDS IN THE
SAME GROUP;
data test_orig;
    length _new_numc $2;
    retain _num 3;

    set test_2;

    if groupname=groupname2 & index(series,'F') then
        do;
            _new_num = _num +input(compress(series,,'kd'),??best.);
            _new_numc=ifc(_new_num<10,cats('0',_new_num),put(_new_num,best.
-l));
            series = cats("F",_new_numc);
        end;
   drop _: ;
run;


*FINALLY COMBINING BOTH;
proc sort data=test_new;
    by groupname series;
run;

data new_final;
    set test_orig
        test_new;
    by groupname series;
run;
```

The regular “split-&-combine” program first creates a new data set (TEST_NEW) which only contains the newly inserted records based on the split condition (prxmatch('/^Safety Analysis Set/i',wholeline)), then splits into another data set in order to selectively modify the SERIES values. In the end, the two data sets are combined. The whole process takes even more steps since the additional information is needed to find the target by matching GROUPNAME & GROUPNAME2 and adjust SERIES values.

Analysis: There are no external data needed for the manipulation. The original data set (TEST) is not sorted but it is grouped by GROUPNAME. We can take the advantage of this organized data structure as well as the record of containing the string “Safety Analysis Set” for being both the inserting point and always the last record in the SERIES “T” records. The direct outputting strategy can be this: first output all records up to that inserting point and then output the three new records if it is the target group. Since GROUPNAME is already in the records there is no need to derive GROUPNAME2. The renovated program is fully implementing the whole job in one step, as follows:

```
data final;
        length groupname _retain_groupname $10 _new_numc $2;
```

```
        retain _replacelist "&globalfoot." _retain_groupname '' _num 3;

        set test;

        **SUBSTEP 1;
        if groupname=_retain_groupname then
                    do;
                        _new_num = _num + input(compress(series,,'kd'),
??best.);
                        _new_numc=ifc(_new_num<10,cats('0',_new_num),
put(_new_num,best. -l));
                        series = cats("F",_new_numc);
                    end;
        **SUBSTEP 2;
        output;

        **SUBSTEP 3;
        **INSERTING NEW ENTRY;
        if prxmatch('/^Safety Analysis Set/i',wholeline) then
                    do _i=1 to countw(_replacelist,'#');
                        wholeline = scan(_replacelist,_i,'#');
                        series = cats("F0",_i);
                        _retain_groupname = groupname;
                        output;
                    end;

        drop _:;
run;
```

Brief explanations:

- In SUBSTEP 1, modifying “F” SERIES values in the group records only after the SUBSTEP 3. The trick of the timing is to check the retained GROUPNAME (_RETAIN_GROUPNAME), which is only refreshed after the three new records are created in SUBSTEP3. For other groups or the front-running “T” series in the same group, since _RETAIN_GROUPNAME still carries the value from the old group, the IF-THEN logic would be false and the values of SERIES remain intact;
- In SUBSTEP 2, outputting the original record with or without modified SERIES values;
- In SUBSTEP 3, inserting and outputting the three new records in the target group where GROUPNAME contains the string “Safety Analysis Set”. For being used in the SUBSTEP1, the current value of GROUPNAME is retained in _RETAIN_GROUPNAME. Because the data is grouped with GROUPNAME, _RETAIN_GROUPNAME is always reset in the target group at the end of “T” series.

Clearly, direct outputting technique is a much more straightforward and powerful tool for the scenarios like this.

CONCLUSION

Modern data can be easily massive, containing billions of records, which inevitably imposes an efficiency challenge to SAS programmers. Fortunately, over the years SAS institute as well as talented SAS users has successfully developed many powerful techniques which can be readily tapped into by the SAS programmers in the clinical trial field. Here I summarized these three powerful techniques to deal with the common pharmaceutical jobs including DoW, self-interleaving techniques, and direct outputting, especially when the manipulation does not need external data and is based on ordered/grouped data.

Specifically, DoW is powerful to handle these 3 tasks: Performing multiple calculations in one step and outputting only one record per BY-group, transposing data in a BY-group based on one variable, and inserting a new record per BY-group. Self-interleaving technique is great for these 2 looking-backward tasks: deriving & adding the group summarization data to all records, and to specific records. Direct outputting technique is for grouped data or other creative applications.

Please note there may be other methods to achieve the same goal. For example, PROC SQL may seem to have such capacity with subqueries (Vemuri, 2013) but its application is rather limited, and the program may quickly become unwieldy especially when multiple derivations are required. Further, PROC SQL might become intolerably slow when millions or billions of records are to be processed especially with subqueries.

As might be recognized by many seasoned programmers, these techniques have been around for a few years and are technically mature for any data processing. What the paper really is promoting is that we programmers should not settle with a less ideal solution easily. Rather, we need to have our eyes on the most efficient methods to complete the tasks at hand in all circumstances, if allowed. Although it is indeed a topic beyond the scope of such a short paper, it would be the everlasting topic for every ardent programmer, wouldn't it?

REFERENCES

Brucken, Nancy. 2008. "*One-Step Change from Baseline Calculations, and Other DOWLoop Tricks*" Proceedings of the MWSUG 2008 Conference. Available at https://www.mwsug.org/proceedings/2008/pharma/MWSUG-2008-P01.pdf

Dorfman, Paul. 2008. "*The DOW-Loop Unrolled*". PharmaSUG 2008 Conference Proceedings. Available at https://www.lexjansen.com/pharmasug/2008/tu/TU02.pdf

Schreier, Howard. 2003. "*Interleaving a Dataset with Itself: How and Why"* Proceedings of the NESUG 16 Conference, Arlington VA.
Available at https://www.lexjansen.com/nesug/nesug03/cc/cc002.pdf

Su, Jason. 2020. "*A Game Plan for Beating the IF-THEN-ELSE Overhead in DATA Steps"* Proceedings of the SESUG 28th Conference.
Available at https://www.lexjansen.com/sesug/2020/SESUG2020_Paper_152_Final_PDF.pdf

Vemuri, Pavan. 2013. "SQL SUBQUERIES: Usage in Clinical Programming". PharmaSUG 2013 Conference Proceedings.

Available at https://www.pharmasug.org/proceedings/2013/PO/PharmaSUG-2013-PO15.pdf

ACKNOWLEDGMENTS

The author is sincerely indebted to Ken Borowiak (PPD Inc.), Gary Greenfield (PPD Inc.) and Jim Box (SAS Institute) for their constructive suggestions.

RECOMMENDED READING

- Lafler Kirk P. 2013. *PROC SQL: Beyond the Basics Using SAS*, 2nd Ed. Cary, NC: SAS Institute

CONTACT INFORMATION

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

Jason J. Su
Daiichi Sankyo, Inc.
211 Mt. Airy Rd
Basking Ridge, NJ 07920
(919) 260 5649

jsu@dsi.com