PharmaSUG 2026 - Paper PK-276

SAS and R for Expanding Data for Pharmacometrics Analysis (PMx) Analysis Data Sets

Jeff Rathbun and Rebecca Humphrey, Simulations Plus

ABSTRACT

Pharmacometrics (PMx) Analysis is performed with a time-ordered data set. The data set contains observation records (Pharmacokinetic (PK) concentrations and/or Pharmacodynamic (PD) endpoint values), dosing records, any demographic, vital sign, laboratory test, or additional classifying / qualifying data of interest. Typically, this data is contained in SDTM and/or ADaM data domains that are assembled into the PMx Analysis data set. SAS and R are programming languages that are used to accomplish this.

This paper focuses on expanding data in PMx Analysis data set builds and how that is handled by SAS and R. Data is often provided on specific dates or over a range of dates instead of daily records. Dosing information, subject data (e.g. weight), and simulated patient data are examples where expanding data is needed. R methods examined will include use of the **tidyverse** package and the combination of several functions. SAS methods examined will include macro programming, retain statements, and use of the Program Data Vector (PDV). There are advantages and disadvantages to using each language.

INTRODUCTION

Preparing data for PMx analysis uses many data assembly techniques. When daily records are required and are not provided in the source data, they need to be created. Both SAS and R have methods to generate daily data from data provided as points or ranges. This paper compares the methods that SAS and R use for expanding data with examples for dosing data, subject data, and simulated patient data. SAS and R are structurally different but will generate similar results.

The code presented is based on typical SDTM/ADaM data domains, not any specific data. R code utilizes the **tidyverse** package and the *dplyr::rowwise()* function. SAS code utilizes macro functions, retain statements and the Program Data Vector (PDV).

DOSING DATA

Dosing data for a participant is typically in EX or EC SDTM domains. The ADaM ADEX data set may contain detailed dosing data as well. A trial can last for one day, weeks, months, or even years. It is not always possible to have a record of every dose that was taken. Instead, the data is often presented as a range of time the patient took a given dose. In EX, this would be captured in the variables EXDOSE, EXDOSU, EXSTDTC, and EXENDTC as seen in Table 1

SUBJID	EXDOSE	EXDOSU	EXSTDTC	EXENDTC
1	5	mg	2023-07-07	2023-07-28
1	5	mg	2023-07-29	2023-08-15
1	0	mg	2023-08-16	2023-08-18
1	5	mg	2023-08-19	2023-08-31
1	10	mg	2023-09-01	2023-09-22
1	20	mg	2023-09-23	2023-10-10

Table 1. Example dosing data from EX

The subject is dosed daily (QD) and records for each day need to be generated.

R Implementation

R can expand the above dosing data into daily records with the **tidyverse** package and the *rowwise()* function as seen in Program 1.

```
library(tidyverse)

# Covert character date variables into date variables
# Expand each range into daily dose records
ex_expanded <- ex %>%
     mutate(SDATE = lubridate::as_date(EXSTDTC),
      EDATE = lubridate::as_date(EXENDTC)) %>%
      rowwise() %>%
      mutate(DATE = list(seq(from = SDATE, to = EDATE, by = 1))) %>%
      unnest(cols = DATE) %>%
      arrange(SUBJID, DATE)
```

Program 1. R code example for creating daily dosing records

Separate daily dosing records for each day in the range of SDATE to EDATE are created. The function *group_by()* defines the individual row keys to be expanded, The function *rowwise()* allows computation on a data frame one row-at-a-time, similar to SAS PDV. The function *mutate()* defines the new variable, DATE. The functions *list()* and *seq()*_used together create a list of the individual dates from the start and end dates. The function *unnest()* creates the separate records per DATE from the list of dates. The function *arrange()* is then used to sort the data. The use of *rowwise()* and *group_by()* of the key variables for the record will yield the same result.

Start and end dates, SDATE and EDATE, need to have the class of 'Date' in order to increment by 1 day in the *seq()* function. Without the *unnest()* function, the DATE is stored as a list of the numeric value of the expanded dates, c(19545, 19546, 19547, 19548, 19549, 19550, 19551, [...]. and additional rows are not created. If the desired outcome is a summary by the grouped variables rather than the entire data fram,for example, if the total dose between the start and end date ranges was the desired outcome, then add after arrange(), *group_by(SUBJID, SDATE, EDATE) %>% mutate(TOTDOSE = sum(EXDOSE)) %>% ungroup()*

SAS Implementation

SAS can expand the dosing data in Table 1 into daily records with use of an output statement as seen in Program 2

```
/*Convert character date variable into numeric date variable*/
data ex;
   set ex;
   sdate=input(exstdtc,yymmdd10.);
   edate=input(exendtc,yymmdd10.);
run;

/*Expand each range into daily dose records*/
data ex_expanded;
    set ex;
     do date=sdate to edate;
        output;
    end;
    format date date8.;
```

```
run;

/*Sort data*/
proc sort data=ex;
 by subjid date;
run;
```

Program 2. SAS code example for creating daily dosing records

With a data step, do loop, and output statement, a record is created for each day specified in the do=sdate to edate range (a by is not specified here, the do loop will assume an increment of 1 if no by condition is specified). A proc sort orders the data by subject and date.

Both R and SAS are straightforward in their implementations. Each row is expanded to create a daily dose record. With **tidyverse's** piping feature, the whole process can be completed in one code block. SAS requires separate data and proc steps to both expand and sort the data.

SUBJECT DATA

Subject data is typically provided at each visit, rather than every day. In the PMx data set, this needs to be applied to the daily records. The example below uses weight and body mass index (BMI) as typically provided in VS/ADVS, but could be applied to any time-vary subject data (laboratory values, concomitant medications, etc). Weight and BMI data as seen in VS is shown in Table 2

SUBJID	VSTESTCD	VSDTC	VSSTRESN
1	WEIGHT	2023-07-06	82.4
1	WEIGHT	2023-07-22	81.2
1	WEIGHT	2023-08-02	80.7
1	WEIGHT	2023-08-19	81.1
1	BMI	2023-07-06	23.7
1	BMI	2023-07-22	23.1
1	BMI	2023-08-02	23.0
1	BMI	2023-08-19	23.2

Table 2. Example weight data from VS

Subject data can be expanded by interleaving with the analysis data by using Last Observation Carried Forward (LOCF) and/or Last Observation Carried Backwards (LOCB). This example does not consider date time. In the case when there is time associated with subject data, all records on that date should have the subject data on that day.

R Implementation

R can expand subject data by use of **tidyverse's** *fill()* function as seen in Program 3

```
# Time-Varying Subject Data
vs_wide   <-
  pivot_wider(vs,
              id_cols = c(SUBJID, VSDTC),
              names_from = VSTESTCD,
              values_from = VSSTRESN) %>%
  mutate(DATE = lubridate::as_date(VSDTC)) %>%
  select(SUBJID, DATE, WTKGT = WEIGHT, BMIT = BMI)

# Combine Subject Data and Analysis Data
dat <- full_join(dat %>% mutate(ANRCD = 1),
                 vs_wide %>% mutate(TVREC=1),
```

```
                by = c("SUBJID", "DATE")) %>%
      arrange(SUBJID, DATE, desc(TVREC)) %>%
      group_by(SUBJID) %>%
      fill(WTKGT, BMIT, .direction = "downup") %>%
      ungroup() %>%
      filter(ANREC == 1) %>%
      select( -c(ANREC, TVREC))
```

Program 3. R code example for expanding subject data

After creating the subject data with variables for both weight and BMI, the data is then merged to the analysis dataset with a *full_join()* statement. In order to be able to identify the source of records, ANREC is used to identify records that are in the analysis dataset and TVREC is used to identify records that are subject data records.

Following the merge, the data is sorted so when there is time-varying data on a day, the subject data will come first. The *group_by()* statement is used to prevent a previous or subsequent subject's data populated on another subject. The f*ill()* function with .direction="downup" specified will carry the subject information forward then backwards. Group processing is no longer required, so *ungroup()* is used. The time-varying records are no longer required, so filtering to the analysis records can be applied. The record flags, TVREC and ANREC can be dropped after the subset to the analysis records.

SAS Implementation

SAS can expand subject data by using retain functionality as seen in Program 4

```
/*Time-Varying Subject Data*/
proc sql;
    create table vs_wide as
    select case
            when not missing(a.subjid) then a.subjid
            when not missing(b.subjid) then b.subjid
           end as subjid,
           case
            when not missing(a.date) then a.date
            when not missing(b.date) then b.date
           end as date,
           a.wtkg, b.bmit
    from (select subjid, input(vsdtc, ddmmyy10.) as date, vsstresn as wtkgt
          from vs
          where vstestcd="WEIGHT") a full join
         (select subjid, input(vsdtc, ddmmyy10.) as date, vsstresn as bmit
          from vs
          where vstestcd="BMI") b
    on a.subjid=b.subjid and a.date=b.date
    order by calculated subjid, calculated date;
    quit;
run;

/*Combine Subject Data and Analysis Data
data dat;
   merge dat(in=a)
         vs_wide(in=b);
   by subjid date;
   if a=1 then anrec=1;
   if b=1 then tvrec=1;
run;


/*Carry Forward then backward*/
%macro carry(var1);

proc sort data=dat;
    by subjid date descending tvrec;
run;
```

```
data dat;
    drop t&var1;
    set dat;
    by subjid date;
    retain t&var1;
    if first.subjid then t&var1=.;
    if &var1 ne . then t&var1=&var1;
    else if &var1=. then &var1=t&var1;
run;

proc sort data=dat;
    by subjid descending date tvrec;
run;


data dat;
    drop t&var1;
    set dsat;
    by subjid descending date;
    retain t&var1;
    if first.subjid then t&var1=.;
    if &var1 ne . then t&var1=&var1;
    else if &var1=. then &var1=t&var1;
run;


%mend carry;

%carry(wtkgt);
%carry(bmit);
```

Program 4. SAS code example for expanding subject data

After creating the subject data with variables for both weight and BMI, the data is then merged to the analysis dataset with a data step merge. In order to be able to identify the source of records, ANREC is used to identify records that are in the analysis dataset and TVREC is used to identify records that are subject data records.

Following the merge, the data is sorted so when there is time-varying data on a day, the subject data will come first. The subject data is first carried forward with a data step retain statement. The created macro *%carry* allows for the same code to be used without repeating it. In the data step, the by statement combines with the use of first. processing prevents data from one patient being set on another statement. A temporary variable is used for carrying the data forward. The variable will resolve based on the variable we are setting (WTKGT will use TWTKGT for example) to fill in missing values. After this is completed, the data sequence is reversed and the same process will be used to carry the values backwards.

Both R and SAS use the same framework for setting subject data of carrying forwards and/or backwards. R has the advantage of being able to do multiple variables in both directions with one code statement. SAS requires each variable/direction to specified separately. The SAS code assumes that there is numeric data.

SIMUALTION SUBJECT DATA

Generating exposures for exposure response values generated with a deterministic simulation in NONMEM can require a data set that is comprised of typical patients. A typical structure is defined with dosing and observation records. An example scenario is a patient dosed daily for 1 week (168 hours). An observation record is inserted prior to every dose starting with the second dose. There will also be observation records inserted every 4 hours for 24 hours after the last dose. Variables TYPE (-1 for the first dose, 0 for all other doses, 2 for observation records), RATE (-2 on dose records, missing on sample records), and EVID (1 on dose records, 0 on observation records) will be set for the simulation. DV will be missing on all records. A typical patient will resemble the data in Table 3

ID	EVID	TYPE	RATE	DOSE	TIME	TSPD	DV
1	1	-1	-2	30	0	0	.
1	0	2	.	30	23.999	23.999	.
1	1	0	-2	30	24	0	.
1	0	2	.	30	47.999	23.999	.
1	1	0	-2	30	48	0	.
1	0	2	.	30	71.999	23.999	.
1	1	0	-2	30	72	0	.
1	0	2	.	30	95.999	23.999	.
1	1	0	-2	30	96	0	.
1	0	2	.	30	119.999	23.999	.
1	1	0	-2	30	120	0	.
1	0	2	.	30	143.999	23.999	.
1	1	0	-2	30	144	0	.
1	0	2	.	30	167.999	23.999	.
1	1	0	-2	30	168	0	.
1	0	2	.	30	172	4	.
1	0	2	.	30	176	8	.
1	0	2	.	30	180	12	.
1	0	2	.	30	184	16	.
1	0	2	.	30	188	20	.
1	0	2	.	30	192	24	

Table 3. Sample subject data for simulation dataset

The data set will also contain subject data that is not being presented here.

R Implementation

R can create patient data with use of **tidyverse** and *rowwise()* processing as seen in Program 5. It is assumed that PTS is a data set that is one record per patient and has all necessary subject level data (ID, dose information, subject data, etc.)

```
# Dose Records
dose <- pts %>%
    group_by(ID) %>%
    rowwise() %>%
    mutate(TIME = list(seq(from = 0, to = 336, by = 24))) %>%
    ungroup() %>%
    unnest(cols = c(TIME)) %>%
    arrange(ID, TIME) %>%
    mutate(TYPE = ifelse(TIME > 0, 0,-1),
         DV = NA_real_,
         TSPD = 0,
         RATE = -2,
         EVID = 1)
```

```
# Sample Time Points
samptm <- c(seq(23.999,167.999,by=24),seq(172, 192, by = 4))

# Observation Records
obs <- pts %>%
   group_by(ID) %>%
   rowwise() %>%
   mutate(TIME = list(samptm)) %>%
   ungroup() %>%
   unnest(cols = c(TIME)) %>%
   arrange(ID, TIME) %>%
   mutate(TYPE = 2,
          DV = NA_real_,
          TSPD = ifelse(TIME < 168, 23.999, TIME - 168),
          RATE = NA_real_,
          EVID = 0)

# Combine
dat <- bind_rows(dose, obs) %>%
   arrange(ID, TIME)
```

Program 5. R code example for simulated patient data

Dose records and observation records are created separately based on specified time points and then combined. Similar to the dose data implementation above, **tidverse** and *rowwise()* function are used to expand the data. After combining the data, the data is then sorted in subject and time sequence.

SAS Implementation

SAS can create subject data by using a data step, do loop, and output statement as seen in Program 6. It is assumed that PTS is a data set that is one record per patient and has all necessary subject level data (ID, dosing information, subject data, etc.)

```
data dat(drop=_h);
    set pts;
   dv=.;
   /*Do loop for outputing doses every 24 hours up to 168 hours*/
    do _h=0 to 168 by 24;
     /*For the last record, we are outputting additional samples,
       so need to handle separately*/
     if _h=168 then do;
           type=0;
           time=_h;
           tspd=0;
           evid=1;
           rate=-2;
           output;
           /*Samples after last dose*/
           do _k=4 to 24 by 4;
               type=2;
               time=_h+_k;
               tspd-_k;
               evid=0;
               rate=.;
               output;
           end;
       end;
       /*All other doses*/
       else do;
           /*Dose*/
           if _h=0 then type=-1;
           else type=0;
           time=_h;
           tspd=0;
           evid=1;
           rate=-2;
           output;
```

```
            /*sample*/
            type=2;
            time=_h+23.999;
            tspd-23.999;
            evid=0;
            rate=.;
            output;
        end;
run;
```

Program 6. SAS code example for simulated patient data

With the PDV, SAS processes each row individually and is able to create multiple records based on each row. The do loop is based on the dose records that need to be created. Conditional statements are used to create the necessary sample records. Since the data is being created row by row, sorting the data is not needed as the records are created in time-order sequence.

The framework for creating simulated subject data is similar in R and SAS. SAS has the advantage of being able to create all records at once and in subject and time order.

CONCLUSION

When generating a PMx analysis data set, it is often necessary to expand data. SAS and R have similar frameworks, but different implementations to accomplish this. In some cases, there is little difference in the process. In other cases, one language has an advantage over the other. Both are power tools in data assembly and can be used to create PMx analysis data sets.

REFERENCES

R Documentation

SAS Documentation

ACKNOWLEDGMENTS

The authors wish to acknowledge their colleagues at Simulations Plus for many years of discussion and sharing of techniques.

CONTACT INFORMATION

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

Jeffrey Rathbun
Simulations Plus
jeff.rathbun@simulations-plus.com

Rebecca Humphrey
Simulations Plus
rebecca-humphrey@simulations-plus.com