PharmaSUG China 2024 - Paper AP-10077

Create Mock Datasets for Earlier Developing Data Validation and Deviation Program

Wanchun ZANG, Weina JIA, Fei GUO, Sanofi Corporation

ABSTRACT

Programmers follow pre-defined rules to develop programs to perform data validation or deviation detection. Typically, programmers use real data from the study or manually enter dummy data to develop programs. To start the programming process earlier, we develop a SAS program to simulate datasets for the study. The realistic like mock datasets are created based on the study case report form (CRF) design, and the data used to create the mock datasets are from real studies with similar designs. By using the program, the study programmer can set simple parameters and the run the program to generate mock datasets. The use of mock datasets can help identify potential issues or errors in the programming before real data is used, which can save time and resources in the long run. Overall, this approach has been shown to speed up the programming process, and ultimately ensures that data validation, data cleaning, and deviation detection are addressed in a timely manner.

INTRODUCTION

Performing data validation, cleaning, and deviation detection in clinical trials is essential for ensuring the accuracy, completeness, and reliability of the data collected during the trail. This is important for making reliable conclusions about the safety and efficacy of a treatment, as well as for meeting regulatory requirements. Using program can perform data validation and deviation detection more quickly and accurately. The program designed to follow pre-defined rules can help ensure the data validation and deviation detection process is consistent and reliable.

Programmers typically do not wait for real data to start programming. Instead, we typically start by developing program using the test data entered by data manager (DM) in the electronic data capture (EDC) test env. The data in data management system is extracted and integrated into a data hub, the datasets per domain are eventually available in a data mart that can be used by programmers. While this approach allows for an earlier start to the programming process, it can be low efficiency when manually entered for multiple forms and folders. Furthermore, due to the long process, the database in the data mart may be unavailable even if the data has created in data management system.

To address this issue, we developed a SAS program that creates mock datasets based on CRF design and real data from studies with similar design. The program can accurately simulate the types of data that will be collected in the real study, reflect the variability or complexity of the real data, and represent the real-world context in which the data validation and deviation program will be used. The mock datasets allow programmers to start the programming process earlier and more efficiently.

LOGIC & DESIGN

The mock dataset should contain data that is similar in structure and content to the data that will be collected in the actual study. To create mock datasets that are more like real study data, we first create empty datasets for the study, then insert value into the datasets from a mock data library.

CREATE MOCK DATA LIBRARY

Logic to create mock data library

1. For a specific therapeutic area (TA), there is a set of expected domains
2. Each domain is a set of records from the same domain data of real studies in the TA
3. Records in each dataset should randomized mixed/sorted, to enable records extracted from mock data library are randomized from real studies in the TA.

Inclusion criteria of real study

The study database has been locked or the database should be stable and have enough records. In total, we selected 22 studies from 5 TAs to create mock data library.

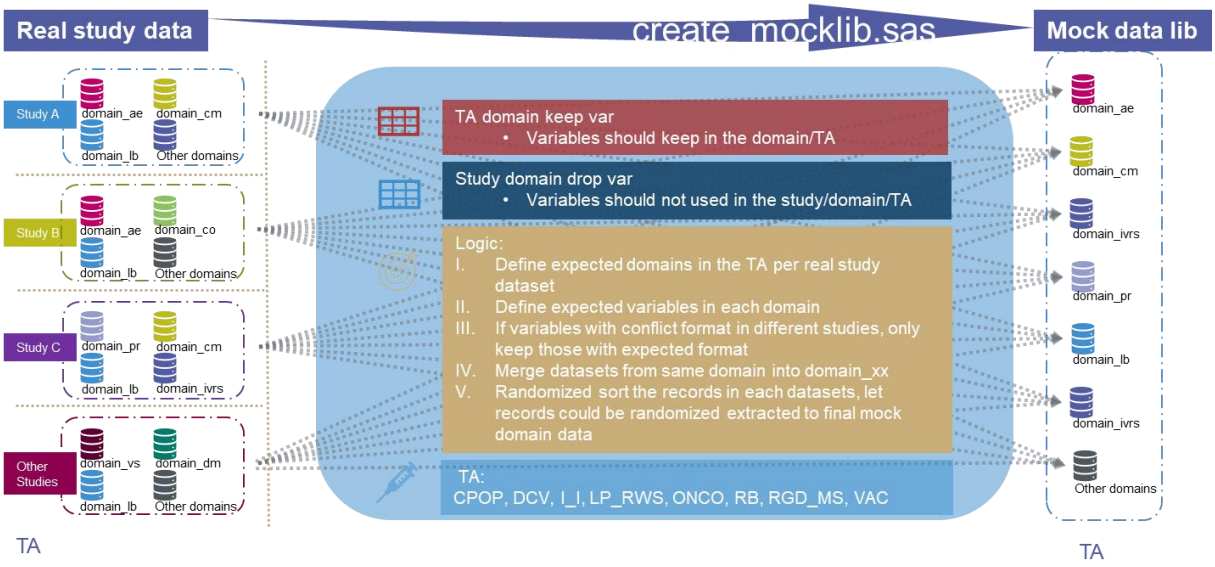


Figure 1. Logic to create mock data library.

Program to create mock data library

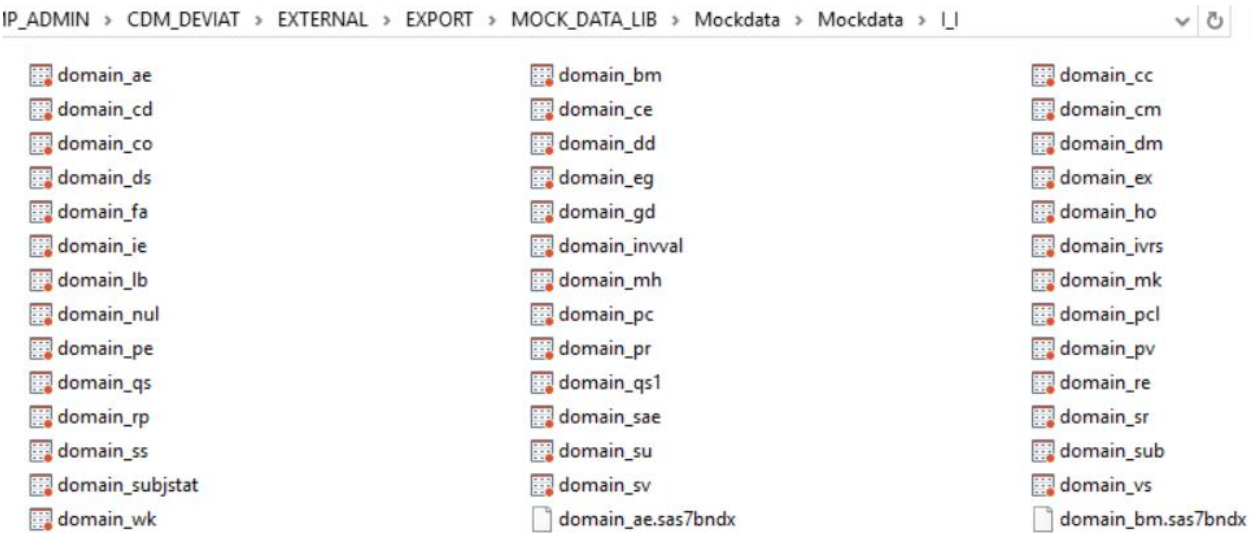
We prepare a study list file which defines the TA, compound and study name, then use program to loop all the study data according to the study list file, to determine domains expected in each TA, variables expected in each domain, and data type and format expected for each variable. The program creates domain datasets for each TA and create an index file based on the variable “FORM_OID” for each dataset as well.

```
%macro create_lib();
    %do i = 1 %to &nb.;
          %sysexec mkdir -p &out_path./Mockdata/&&TA&I.;
          libname &&TA&I. "&out_path./Mockdata/&&TA&I.";
          data _null_;
                set ta_domain(where=(ta="&&TA&I."));
                call symputx("domain_num",_N_);
                call symputx("domain"||left(trim(_N_)),domain);
          run;
          %put In &&TA&I., there are &domain_num. domain;
          %do n= 1 %to &domain_num.;
                data _null_;
                      set ta_domain_study(where=(ta="&&TA&I." and
domain="&&domain&n." ));
                      call symputx("study_num",_N_);
                      call symputx("study"||left(trim(_N_)),study);
                      call symputx("dataset"||left(trim(_N_)),dataset);
                run;

/*              domain keep var*/
                data _null_;
                      set domain_keepvar(where=(ta="&&TA&I." and
domain="&&domain&n." ));
                      call symputx("keepvar_num",keepvarnum);
```

```
                run;
                data _null_;
                      set domain_keepvar(where=(ta="&&TA&I." and
domain="&&domain&n." ));
                      call symput('keepvar', catx(" ",of KEEPVAR1-
KEEPVAR&keepvar_num.));
                run;
                %put In &&TA&I. &&domain&n., &keepvar. will keep.;
/*              study drop var*/
                data _null_;
                      set study_dataset_dropvar(where=(ta="&&TA&I." and
domain="&&domain&n." ));
                      call symputx("dropdataset_num",_N_);
                      call
symputx("dropvar_num"||left(trim(_N_)),dropvarnum);
                      call symputx("dropdataset"||left(trim(_N_)),dataset);
                run;

                %do d=1 %to &dropdataset_num.;
                      %local dropvar_&&dropdataset&d.;
                      %if  &&dropvar_num&d.>0 %then %do;
                            proc sql noprint;
                                  select
                                        %if &&dropvar_num&d.>1 %then %do;
                                              catx(' '
                                              %do v=1 %to &&dropvar_num&d.;
                                                    ,dropvar&v.
                                              %end;
                                              )
                                        %end;%else %do;
                                              dropvar1
                                        %end;
                                              into: dropvar_&&dropdataset&d.
                                  from study_dropvar
                                  where dataset="&&dropdataset&d."
                            ;quit;
                      %end;%else %do;
                            data _null_; call
symput("dropvar_&&dropdataset&d.",'');run;
                      %end;
                %end;

/*              create merged domain*/
                data DOMAIN_&&domain&n.;
                      retain STUDYID SITEID SUBJID USUBJID STUDY_EVENT_OID
VISIT FORM_OID;
                      set
                            %do m= 1 %to &study_num.;

                                  %put In &&TA&I., &&domain&n., study
&&study&m. with dataset &&dataset&m. will set into mocklib for this TA;
                                  %put &&&&dropvar_&&dataset&m. will drop;

    &&study&m...&&dataset&m.(drop=&&&&dropvar_&&dataset&m.)
                            %end;
                      ;
                      STUDYID='XXX99999';
```

```
                    USUBJID=cat('099999-',substr(USUBJID,prxmatch('/-
/',usubjid)+1,LENGTH(USUBJID)-prxmatch('/-/',usubjid)));
                    keep &keepvar.;
                run;
/*              mix the records in merged datasets*/
                %let dset=DOMAIN_&&domain&n.;
                %let dsid = %sysfunc(open(&dset));
                %let nobs =%sysfunc(attrn(&dsid,NOBS));
                %let rc = %sysfunc(close(&dsid));

                data DOMAIN_&&domain&n.;
                    set DOMAIN_&&domain&n.;
                    x=rand('Integer',1,&nobs.);
                run;
                proc sort;by x;run;
                data &&TA&I...DOMAIN_&&domain&n.;
                    set DOMAIN_&&domain&n.;
                    drop x;
                run;
/*              create index for each domain dataset*/
                proc datasets library=&&TA&I..;
                modify domain_&&domain&n.;
                index create form_oid;
                quit;
          %end;
    %end;
%mend create_lib;
```

Output 1. shows an example of the mock library created.



Output 1. Mock data library datasets.

CREATE MOCK DATA FOR STUDY

The creation of mock data can begin based on either the Study Data Setup (SDS) file or existing datasets. This is because the SDS file is typically ready earlier than the database, as it involves the creation of the physical database structure and the loading of the study data into the database. By using the SDS file as a starting point, we can generate the mock data earlier because the necessary data elements have been defined. Using existing datasets with or without records is safer because the variables and their types are

stable, so the program just needs to insert mock data. Overall, the choice of starting point will depend on the study status and specific needs.

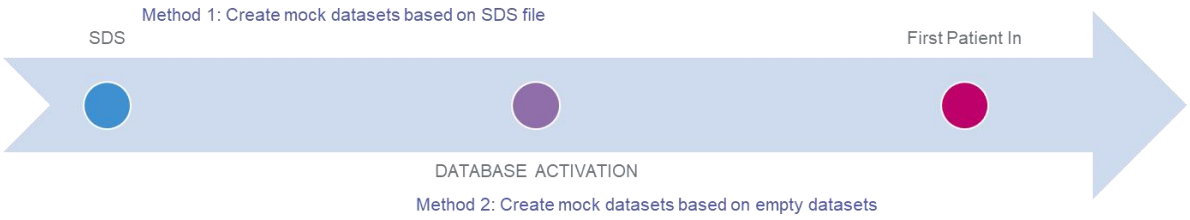


Figure 2. Select method depends on study status.

Create mock datasets based on SDS file

We first create empty datasets based on the forms, fields, and their types defined in the SDS file. Then, we insert records from the mock data library to fill the empty datasets.

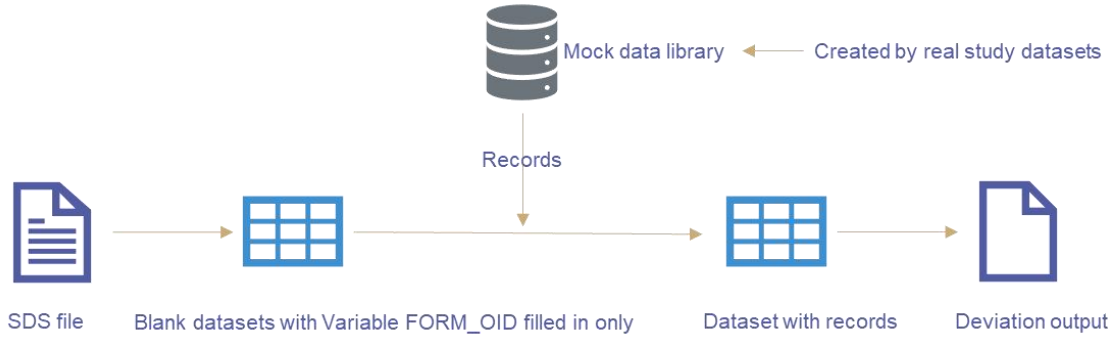


Figure 3.. Create mock datasets based on SDS file.

The program has below functions:

1. Import SDS file to obtain the forms, fields, and data format
2. Simulate forms datasets per the form, variable, variable format, and variable length from step 1
3. Set the form level datasets generated in step 2 into domain level datasets per the rule from step 1, e.g. variable order
4. Add default variables into the datasets generated in step 3 to make them as empty datasets like in real study, e.g. STUDYID, SITEID, SUBJID, etc.
5. Insert data from mock data library

Create mock datasets based on empty datasets

When the datasets are created in the data mart following the normal process, we duplicate the datasets and clear them. Then, we insert an empty dataset with records from the mock data library.

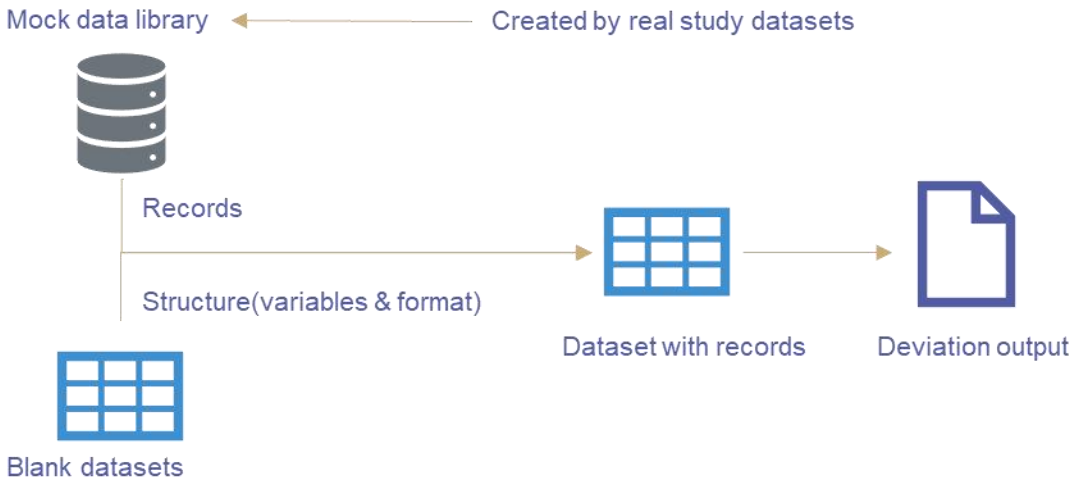


Figure 4. Create mock datasets based on empty datasets.

As we create mock datasets based on the existing datasets, the first step is to duplicate the datasets as well as clear the datasets.

```
%macro empty_dataset();
          data deviat;
                set source.DEVIAT_&w_study.;
                %do p = 1 %to &var_nb.;
                      if &&type&p. = 1 then &&variable&p.=.;
                            else if &&type&p. = 2 then &&variable&p.='';
                %end;
          run;
    data RDFIX_dm.deviat_study;
          set deviat;
          if _n_=1;
    run;
%mend empty_dataset;
```

Then insert the data from mock data library into the empty datasets.

```
    %do i = 1 %to &targetdomain_nb.;
          %if %eval(not(%upcase(&&targetdomain&i.) in
&missing_domain.)) %then %do;
/*        vars to keep in final output*/
                proc sql noprint;
                select distinct   targetname into: keep_var separated by "
"
                from domain_var
                where targetmemname="&&targetdomain&i."
                ;quit;

                /*check if with inconsistent type between target and mock
lib*/
                proc sort nodupkey
data=domain_var(where=(targetmemname="&&targetdomain&i." and targettype ne
mocklibtype and mocklibmemname ne ''))
                                                    out=vartype_dif;
                by targetname targettype;
                run;
```

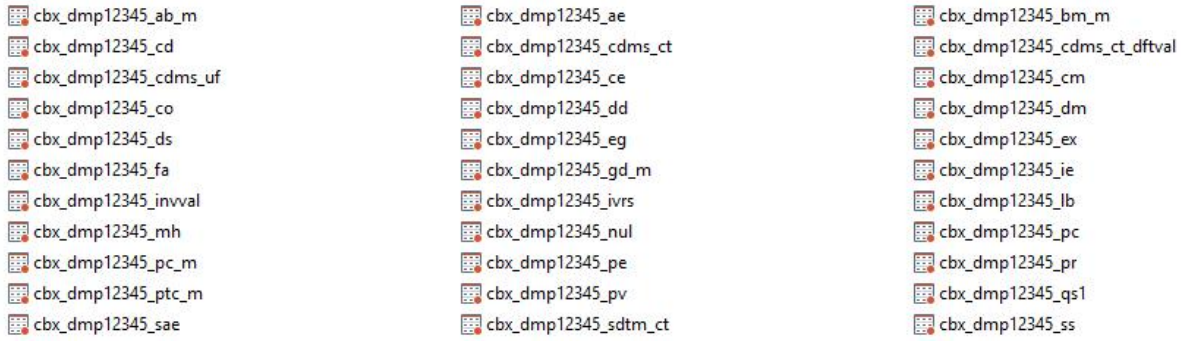
```
                %let dset=vartype_dif;
                %let dsid = %sysfunc(open(&dset));
              %let vartypedif_nb =%sysfunc(attrn(&dsid,NOBS));
              %let rc = %sysfunc(close(&dsid));
                %put &vartypedif_nb varriables with inconsistent type
between target and mocklib;

                %if &vartypedif_nb. >0 %then %do;
                      data _null_;
                            set vartype_dif;
                            call symputx("dropvarnb",_N_);
                            call
symputx("drop_var"||left(trim(_N_)),strip(targetname));
                            call
symputx("dropvartype"||left(trim(_N_)),targettype);
                      run;
                      %put dropvarnb is &dropvarnb.;
                      data out.&&targetdomain&i.;
                            set RDFIX_dm.&&targetdomain&i.(obs=0 drop= %do
x=1 %to &dropvarnb.; &&drop_var&x. %end;)
                                  mocklib.&&mocklibdomain&i.(rename= (%do
x=1 %to &dropvarnb.;

                &&drop_var&x.=_&&drop_var&x.

          %end;)

    %if %eval(&record_num. <1) %then %do;

    obs=2000

    %end;%else %do;

    obs=&record_num.

    %end;)
                            ;
                            STUDYID=strip("&w_study.");

    USUBJID=cat("0",compress("&w_study.",'/','A'),substr(usubjid,6,length(u
subjid)-5));
                            %do y=1 %to &dropvarnb.;
                                  if &&dropvartype&y.=1 then
&&drop_var&y.=input(_&&drop_var&y.,??best.);
                                  else &&drop_var&y.=strip(_&&drop_var&y.);
                            %end;/*%do y=1 %to &vartypedif_nb.;*/
                            keep &keep_var.;
                      run;
                %end;%else %do;
                      data out.&&targetdomain&i.;
                            set RDFIX_dm.&&targetdomain&i.(obs=0)
mocklib.&&mocklibdomain&i.(%if %eval(&record_num. <1) %then %do;

                                                obs=2000

                                          %end;%else %do;
```

```
                                                    obs=&record_num.

                                                %end;);
                                keep &keep_var;
                                STUDYID=strip("&w_study.");

    USUBJID=cat("0",compress("&w_study.",'/','A'),substr(usubjid,6,length(u
subjid)-5));
                          run;
                       %end;/*%if &vartypedif_nb. >0 %then %do;*/
                %end;%else %do;
                       data out.&&targetdomain&i.;
                             set RDFIX_dm.&&targetdomain&i.;
                       run;
                %end; /*%if %eval(not(%upcase(&&targetdomain&i.) in
&missing_domain.)) %then %do;*/
    /*SUBJID is all pt extracted in all domains*/
                %data subjid;
                       set subjid out.&&targetdomain&i.(keep=SUBJID);
                %run;
                %proc sort data=subjid(where=(not missing(subjid))) nodupkey;by
subjid;run;
    %end;/*%do I = 1 %to &domain_nb.;*/
```

The above programs can be packed into a macro program, which the study programmer can run by only needing to define the path to save datasets, the record number expected in the mock datasets, and the TA that links to the mock data library.

```
/*Main program*/
%mock_data_dev
                (work_path  =
&W_ROOT./&W_MODWORK.OPS/&W_COMPOUND./&W_STUDY./CDM_DEVIAT/EXTERNAL/EXPORT
               ,record_num = 5000
               ,TA   = I_I
               ,data_out = &work_path./&w_study._MOCK_DATA_&sysdate.
               ,create_deviat_file     = Y
               );
```

Output 2 shows an example of the mock datasets created.



Output 2. Mock datasets for study.

QUALITY CHECK

We focus on below to perform quality checks on the mock dataset library and mock dataset for study.

1. Check for missing domains in mock data library
    - Check for missing domains: Compare datasets in mock data library with standard SDS file to detect domains that are not included.
    - Compare with real study datasets: Compare datasets in mock data library with datasets in real study to check for:
        - Missing domains or variables
        - Different formats
2. Check feasibility of using mock datasets to develop deviations in real study
    - Apply the real deviation programs in the mock datasets, to validate if we can program using the mock datasets
    - Check if the program can work when using mock data.

QC FINDINGS & SOLUTION

1. Incomplete library - domain missing in mock data library
    - The library may not have been fully populated with all the necessary domains. The reason is that the studies chose to create mock data library failed to cover all cases, moreover, we found some domains only exists in very specific study. To address the issue, we add the specific study into study list and update the mock data library.
2. Domain/ variables failed to create in study mock data
    - The reason is that some variables are derived when data is integrated into the data warehouse, which means that the variables are not present in the SDS files. We solve this issue by updating the program or manually adding the variables to the SDS files. Additionally, we found that the data types created based on the SDS files are different from the real datasets, which is due to the SDS files being incorrect.

CONCLUSION

Our mock data creation program has been demonstrated to be useful for programmers to create mock datasets as soon as the CRF design is close to being completed. By using this program, programmers can initiate programming activities more quickly and efficiently. Furthermore, using mock data can help identify potential issues or errors in the programming before real data is used, which can save time and resources in the long run. Overall, the use of mock data is an effective way to streamline the programming process, enable the study team to validate, clean the data, and review and address the deviation list in a timely manner.

REFERENCES

SAS® 9.4 Macro LanguageReference Fourth Edition
http://support.sas.com/documentation/cdl/en/mcrolref/67912/PDF/default/mcrolref.pdf

ACKNOWLEDGMENTS

It's team effort when it comes to validating logic, testing programs, and providing feedback.

CONTACT INFORMATION

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

Name: Wanchun ZANG (April ZANG)
Enterprise: Sanofi
Email: April.Zang@Sanofi.com