PharmaSUG 2016 - Paper IB12

Handling Interim and Incomplete Data in a Clinical Trials Setting

Paul Stutzman, Axio Research LLC, Seattle, WA

ABSTRACT

This paper discusses potential problems that can arise when working with interim and incomplete data. It identifies ways to mitigate these issues through preparation and ongoing reassessment. Techniques and programmatic tools for performing these preparation and reassessment tasks are also presented.

Programmers and statisticians are frequently presented with interim and incomplete data. This is often the case when preparing reports for Data Monitoring Committees (DMCs) or Data Safety Monitoring Boards (DSMBs), performing interim analyses, or developing standards-compliant data set programs. Thus, it is important to understand the constraints and potential pitfalls of working with preliminary data, to write programs that handle both anticipated and unanticipated values, and to develop tools and techniques for understanding how data change over time.

INTRODUCTION

Within the conduct of a clinical trial, there are many programming tasks that must be completed before data have been finalized and databases are locked. This paper explores the implications of working with this data, and it presents tools and processes that can reduce their impact.

There are a number of instances when this is necessary. Reports to DMCs and DSMBs and interim analysis reports are, by definition, prepared on data that have not been finalized. Similarly, the data used for medical monitoring reports are often not complete.

A somewhat newer area where interim and incomplete data can be problematic is in the preparation of standards-compliant data sets – specifically, data sets that comply with the Clinical Data Interchange Standards Consortium (CDISC) Study Data Tabulation Model (SDTM) and Analysis Data Model (ADaM) standards. The preferred method of programming a Final Statistical Report (FSR) is to produce CDISC SDTM compliant domains, and to use those domains to produce ADaM compliant analysis data sets, from which the FSR's tables, listings and figures are produced. When data are sparse and incomplete, the generation of SDTM specifications, metadata, and the SDTM domains themselves can be difficult. However, delaying the specification and programming of SDTM domains, means the analysis data sets needed for interim analyses might have to come from source (raw) data, and the programming of SDTM domains and ADaM data sets for the FSR could end up being pushed off until the last minute.

Note that the term "data snapshot" (or just "snapshot) is used in this paper to refer to a collection of clinical trial study data, extracted from data collection databases and made available periodically for programming and analyses throughout the course of a clinical trial.

Also, note that the terms "clinical trial" and "study" are used interchangeably throughout this paper.

MOTIVATION

It is a given that a variety of statistical and programming tasks are going to take place before database lock. This can present problems, given the nature of the data that are available early in a trial.

One of the goals in finding effective ways to deal with these early data is to maximize the quality and accuracy of reports that are based upon interim and incomplete data. Another goal is to minimize the cost and effort associated with ongoing programming efforts that occur throughout the conduct of the trial and once databases have been locked. Finally, deliverables throughout the trial should be produced in a timely manner. These goals can be met by putting procedures and tools in place to deal with interim and incomplete data thoughtfully, and by diligently assessing newer data as they are made available.

INTERIM AND INCOMPLETE DATA

For the purposes of this paper, interim and incomplete data can be defined as follows.

Interim data: Interim data is data that are made available before study databases have been locked and before the data have been thoroughly cleaned. Data are often provided in multiple snapshots over time. It is not unusual for the actual structure of the data change from snapshot to snapshot: data sets and variables might be added or removed, and variable attributes might change. Sometimes the meaning and mappings of values might change, or values themselves might change.

Incomplete data: Data incompleteness usually manifests itself in two ways: missing records and missing values. The number of subject-related records will increase as enrollment accrues. Observational and visit-related record counts will increase as a study progresses. Value-wise, variables might not be populated with all their eventual values, and those values (for categorical variables) or value ranges (for continuous variables) might not be known.

There is usually a tradeoff between good, fast and cheap when it comes to obtaining interim data. Project planning texts often describe this as in Figure 1 – Good, Fast, Cheap: Pick any Two. There is a balancing act that has to happen between getting the most accurate interim data possible, ensuring the data are current, while keeping costs and the amount of effort expended reasonable.

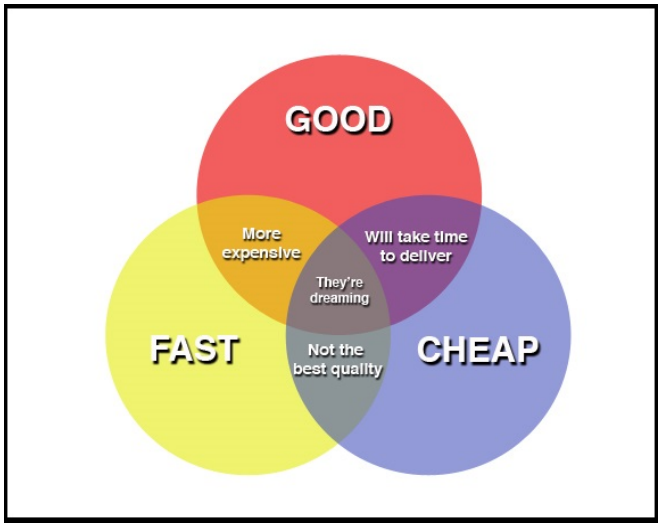

Figure 1. Good, Fast Cheap: Pick any Two[1]

POTENTIAL PITFALLS OF INTERIM AND INCOMPLETE DATA

There are a number of potential pitfalls that can arise when working with interim and incomplete data, and they need to be considered so they can be avoided or addressed. These issues can be grouped into three broad categories: instances where all data are missing, instances where not all values are present in the data, and instances where data are invalid or incorrect.

ALL MISSING

In some cases all data necessary for a certain criteria to be meet will be missing from the data. An example of this might be a study where no deaths have occurred on a trial yet, but deaths are expected at some point. It might be unclear how deaths will be reported. Will they come from records in a disposition data set? Will they show up as adverse events? Will there be a separate deaths data set? Will it be necessary to reconcile these disparate sources of information as the trial progresses?

The more questions of this type that can be answered ahead of time, the better. Programs can be written to handle any issue when the issues are known. If the issues are not completely known, defensive programming techniques should be used, and the user should be notified (often through the SAS® log) that this issue must be addressed as more data accumulate.

MISSING VALUES

The second type of pitfall has to do with not all values being present in the data yet. This happens a lot, particularly when the data are quite immature. It is important to plan for all possible values, even if they are not in the actual data yet. Unexpected values need to be anticipated and handled. Minimally, the user should be notified (again, often through the SAS log) that an unexpected value has been encountered.

One particular issue that can arise from not all values being present is not knowing the values that make up related pairs or groups of variables' values. An example of this is the mapping of visits to visit numbers. Another is the relationship between laboratory test names, test codes and their associated measurement units.

Laboratory data can be particularly problematic, as an individual test's results could come in varying and unexpected measurement units over the course of a trial. It is important that this is handled programmatically, that unit conversions are made correctly, and the user is notified when unanticipated measurement units are encountered.

INVALID AND INCORRECT VALUES

Finally, invalid and incorrect values might be introduced into the snapshot data, particularly when snapshots are made before data can be thoroughly cleaned. It is important to look for values that are out of range, that invalid categorical values are handled, and that mismatched related variables' values are identified. In all of these cases, invalid or incorrect values should be identified, the user should be notified, and decisions need to be made regarding how to proceed with these values. Often a data management organization needs to be notified so these issues can be resolved in future snapshots.

For each of these potential pitfalls, it is important to prepare and plan for them. It is also necessary to put tools and process into place, in order to continually reassess them as each snapshot of new data arrives.

PREPARATION AND PLANNING

It is important to review relevant study documents as early as possible in the planning process, in order to anticipate issues that might arise when working with interim and incomplete data. These documents include the study protocol, annotated Case Report Forms (CRFs), data dictionaries, the Statistical Analysis Plan (SAP), Interim Analysis Plans (IAPs), DMC or DSMB charters, and any other documents that pertain to the structure of the trial and its data. Consideration also needs to be given to the analyses that will be performed, and the information (and its structure) that will be required to support them.

One important consideration is visit structures and numbering. Often, the study protocol and SAP/IAP can be particularly helpful with this. It is critical, though, that an effort is made to handle unanticipated visits and visit relationships. For example, how are unscheduled visits going to be handled? Or, in an oncology trial, is visit windowing necessary?

Flexibility should be built into visit numbering schemes. One study protocol, for example, specified something like,

> Randomized patients will have 21-day chemotherapy cycles until disease progression, unacceptable toxicity, withdrawal of consent, or protocol specified parameters to stop treatment.

For this study, each week on study was numbered sequentially from 1, so Cycle 1 was visit number 1, Cycle 2 was visit number 4 (since it occurs three weeks later), etc. End of Study was arbitrarily assigned visit number 65. That meant subjects could have up to 22 cycles of treatment on study. Unfortunately for the study design (fortunately for the subjects), some subjects ended up with far more than 22 cycles of treatment, necessitating a complete redesign of the study's visit number structure half way through the trial. A more flexible visit numbering scheme could have avoided this problem.

Another consideration relates to categorical variables. Programs, code lists and metadata should be designed up front to handle all possible values of categorical values. Annotated CRFs can provide a great deal of information regarding the values that will ultimately populate these variables.

Continuous variables should also be considered. Anticipated ranges of values should be identified, and code should be put into place to flag out-of-range values.

Regardless of the amount of research that can be done by reviewing study documentation, and how thorough that documentation is, there are always certain aspects of the data that will not be known until the data are actually seen. Therefore, it is important to get a data snapshot as early as practical. At this point, it is unlikely that all potential data values will be represented in the data, but the structure of data sets and their variables should be somewhat stable.

PREPARATION AND PLANNING FOR STANDARDS COMPLIANCE

The increasing importance of generating CDISC standard compliant data sets further increases the importance of planning for all possible values and for handling the unanticipated. Historically, many organizations chose to process raw data into analysis data sets for interim analyses, DMC/DSMB reports, etc. without regard for CDISC standards. This was expedient, but it meant that the production of SDTM and ADaM compliant data sets occurred in parallel, often toward the very end of a trial. This resulted in back-loaded, intensive programming efforts, and it made reconciling interim reporting with final reports difficult.

Now, these same organizations are finding value in converting raw data to SDTM-compliant domains, which are used to create ADaM data sets that are ultimately used to produce interim reports. This can front-load a substantial portion of the SDTM specification and development effort. It can also improve the timeliness of final deliverables, and it can facilitate consistency between reports. In order for this to work effectively, metadata for these standards compliant data sets needs to be drafted as early in the study, and as completely, as possible. Preparation and planning based on addressing and handling the aforementioned issues can help this process greatly.

INITIAL PROGRAMMING

The initial programming effort must consider the potential pitfalls mentioned above. Specifications for the programs and their metadata must be created thoughtfully and thoroughly. The programs themselves should be written in ways that address these potential problems.

There are a number of defensive programming techniques that can be employed to accomplish this. These will be discussed in detail in the Tools and Processes section of this paper.

ONGOING REASSEMENTS

It is certain that data will change over the course of a clinical trial. Therefore it is imperative that reassessments of data structures and content occur regularly throughout the trial – ideally at each data snapshot.

STRUCTURAL CHANGES

Structurally, there are a number of changes that could occur between data snapshots:

- New data sets might be introduced, or previously existing data sets might no longer be included. The information in new data sets needs to be incorporated. For data sets that are omitted, it is important to know if this is by design, and if so, if (and where) the information they contained can be found in the new snapshot.
- The number of records in individual data sets normally increases as more subjects are enrolled and previously enrolled subjects have more study visits. If the number of records in a data set decreases, it is important to understand why. Disproportionate increases in the number of records in one or more data sets should be investigated too.
- New variables might be introduced, or previously existing variables might no longer be included. As with new or omitted data sets, new or omitted variables must be scrutinized.
- Variable attributes might change. It is particularly important to recognize when variable types or lengths change.

CHANGES IN VALUES

There are also a number of changes that could occur within the data itself:

- Categorical variables might contain new values that were not anticipated. Programs should be written to handle these contingencies, but it is not always possible to plan for all changes.
- Continuous variables might have new ranges, and there might be unexpected outliers. This can be particularly problematic in laboratory data.
- There might be changes in the relationships between sets of variables' values. For example, there might be new or different mappings between laboratory tests and their associated test codes or between visit names and visit numbers.

PARTICULAR AREAS OF CONCERN

Laboratory data seem to be particularly associated with changes in values over time. As mentioned above, outliers, variances in test to test code mappings, and changes in ranges have to be checked regularly.

Equally important is the assessment of test result values and the measurement units in which these values are expressed. New units might be introduced for existing tests, and it is not uncommon to find that incorrect units have been specified for individual test results. These issues can be hard to detect, but there are tools and processes that can help.

Another common area of inconsistencies is in the mapping of visit names to visit numbers. As in the example given in the Preparation and Planning section of this paper, it is not uncommon for changes to be made to the mapping of visit names to visit numbers due to unplanned events in a trial's conduct.

Sometimes there is even inconsistency in visit naming and numbering between data sets within a single data snapshot. This seems to happen most often when test results are gathered at local laboratories. It is important that these inconsistencies are recognized and reconciled.

A subtle difference that can have a large impact involves changes in record uniqueness. For example, specifying a subject identifier, an adverse event name, and that adverse event's start date might be sufficient to uniquely identify records in an adverse events data set initially. However, that would no longer be sufficient if a subject were to have two occurrences of the same event (perhaps two headaches) on the same day later in a trial. This can lead to problems with merging data, and if undetected, it can lead to specious results.

Any of these structural or value-related changes could necessitate changes to programs and their specifications. It is also critical that data set, variable and value level metadata are kept up-to-date. Fortunately, there are tools and processes that can identify and address many of these issues.

TOOLS AND PROCESSES

There are a number of tools and processes that can be employed to identify, mitigate and notify users about the issues that can arise from using interim and incomplete data. Some pertain to initial specifications and programming, and others assist with ongoing assessments of data snapshots that happen throughout the course of a trail.

DEFENSIVE PROGRAMMING

Defensive programming is one of the more powerful tools for dealing with interim and incomplete data. It can address many of the pitfalls that can occur.

Defensive programming is a term that comes from traditional computer science. In that context:

> "Defensive programming is a practice where you anticipate failures in your code, then add supporting code to detect, isolate, and in some cases, recover from the anticipated failure."[2]

In the context of this paper, however, the term "defensive programming" is used to mean something a little different:

> Defensive programming involves writing programs that handle all input data (both anticipated and unanticipated) gracefully, and notify the user/programmer when data or processing issues arise.

Programs should be written in such a way that they handle all possible known values, not just values that are represented in the data. These values can often be identified by looking at annotated CRFs prior to programming. All of these values should be present in code lists and the study's metadata. This is particularly important when dealing with data snapshots from early in a study.

Also, the user should be made aware of unanticipated values and outliers, should any occur. This is often accomplished by writing messages to the SAS log. Another possible tactic is to have the program end without producing output (while still writing informative messages to the SAS log). This can be useful if unanticipated values or outliers are encountered in particularly important variables.

Code should be put into place to attempt to calculate all needed variables, even in instances when some or all of the input data are not available yet. Again, messages written to the SAS log can be helpful in ensuring such variables are reassessed when subsequent data snapshots occur.

Programmatically, tests for unanticipated and outlier data can be accomplished in SAS programs by using:

- a final ELSE in IF/ELSE constructs
- OTHERWISE with SELECT statements
- a final ELSE on SQL CASE statements
- OTHER statements in the SAS FORMAT procedure

The following is an example of handling unanticipated values for the variable SEX:

```
if SEX = "M" then MALES + 1;
else if SEX = "F" then FEMALES + 1;
else put "!!" "! War" "ning: missing or unexpected value for SEX: " USUBJID= SEX=;
```

It is recommended that user-written SAS log messages are distinguishable from SAS-generated log messages by using a standard message prefix. For example:

"`!!! Warning: message text`" could designate user-generated warning messages

"`### Note: message text`" could designate user-generated notes

It is also recommended that code that generates these log messages be invisible to automated log searching programs. This could be accomplished for the previous examples with:

```
put "!!" "! War" "ning: message text";
put "##" "# Note: message text";
```

Splitting up the exclamation marks, pound (or hash) signs, and the word "warning" ensures that log search programs do not find the program code itself in the SAS log.

Through these and other techniques, it is possible to handle and identify unanticipated and outlier values. Defensive programming should be incorporated liberally during initial and ongoing program development.

ASSESSING RECORD UNIQUENESS

As mentioned in the Particular Areas of Concern section of this paper, changes in the criteria for record uniqueness can be hard to detect, yet they can be quite important. To that end, a macro could be written to check if a particular sort order provides uniqueness within one or more data sets. An example of such a macro (called FindDups) is provided as Appendix A of this document.

The parameters for this macro include the names of one or more data sets to be checked, and a list of variables that make up a sort order that is expected to identify records in these data set(s) uniquely. For example, if the following code snippet is used to merge three data sets (ds1, ds2 and ds2) by the variable USUBJID:

```
data combined;
     merge ds1 ds2 ds3;
     by USUBJID;
```

The following macro invocation could check for the necessary record uniqueness:

```
%finddups(InDS=ds1 ds2 ds3, SortOrd=USUBJID);
```

Messages are written to the SAS log for each data set that was checked, indicating either that no duplicates were found, or showing the duplicate sort key values. The log messages from this invocation would look something like the following:

```
### Note: No duplicates found in ds1 with SortOrd=USUBJID
!!! Warning: Duplicates in ds2: USUBJID=111-123-0001
!!! Warning: Duplicates in ds2: USUBJID=111-345-0001
### Note: No duplicates found in ds3 with SortOrd=USUBJID
```

It is also necessary to ensure final output data sets have record uniqueness as described in their metadata. The following is an example of testing for record uniqueness in an output data set:

```
%finddups(InDS=Analysis.ADLB, SortOrd=USUBJID PARAMCD AVISITN ADT);
```

The log messages from this invocation would look something like the following (assuming no duplicates were found):

```
### Note: No duplicates found in Analysis.ADLB with SortOrd=USUBJID PARAMCD AVISITN
ADT
```

NEW DATA ASSESSMENT PROCESS

It is important that new data are assessed when each new snapshot arrives. Tools can be created to test for and handle many of the issues mentioned above, but having a defined process can make these assessments easier.

Here is one recommended process:

1. Existing data should be archived before new data are loaded and any changes are made. These data should also be made accessible in an “Archive” or “Previous” folder, to facilitate comparison with the new snapshot.
2. The previous data should be deleted, and the new snapshot’s data should be loaded in its place. By deleting the previous data, data sets that were in a previous snapshot, but are missing from the current snapshot, can be identified easily.
3. The new snapshot should be compared to the data that were archived in the Step 1 above. Discrepancies should be identified, and plans can be formulated for dealing with them. At this point, changes can be made to specifications, metadata and programs as required.

COMPARING OLD AND NEW DATA SNAPSHOTS

An automated tool for comparing old and new data snapshots can be developed. The function of a SAS macro that performs this function is described here, and sample output is shown. A detailed description of how this macro was designed and written is provided in Appendix B of this paper.

The first step in designing such a tool is establishing a set of comparison criteria. These are the criteria that were chosen:

- Identify new or missing data sets
- Show how the number of records increased or decreased in each data set
- Identify new or missing variables in each data set
- Highlight any variable attributes that have changed

- Check for new values in categorical variables
- Provide simple, descriptive statistics for numeric variables in order to identify potential outliers

Thought should be given to selecting an informative way to present the results of these comparisons. In this case the ODS EXCELXP tagset is used to create a Microsoft EXCEL workbook that contains comparison results in several spreadsheets.

The SAS CONTENTS procedure is used to get data set and variable information from the old and new snapshots. This information is compared and spreadsheets are generated based on these results.

First, a data set summary spreadsheet is created. It shows one row for each data set in either snapshot. An example can be seen as Figure 2. This spreadsheet also shows the record counts for old and new versions of each data set, and it shows absolute and percentage changes in record counts. Data sets that only appear in one snapshot, and data sets whose record counts decreased are shown in red for easy identification.

		Record Counts			
				Change	
Dataset	Status	Old	New	Count	Percent
AE		14095	14298	203	1.40%
AM		6913	6962	49	0.70%
CM		26015	24532	-1483	(5.7%)
DM		797	795	-2	(0.3%)
DS		10312	5542	-4770	(46.3%)
DV		4242	4717	475	11.20%
EX		11087	11090	3	0.00%
...					
PD		7650	7742	92	1.20%
PR	Not in Old		2093		
QS		371242	380121	8879	2.40%

Figure 2. Data Set Summary Spreadsheet

Next, a data set level spreadsheet is created for each data set in the new snapshot. An example can be seen as Figure 3. All variables that appear in either snapshot are listed, along with their types and lengths. Variables that only appear in one snapshot and variables that have differing attributes are shown in red for easy identification.

		Old Transfer			New Transfer		
Dataset	Variable	Label	Type	Length	Label	Type	Length
AE	AEACN	Action Taken with Study Treatment	Char	200	Action Taken with Study Treatment	Char	40
	AEACNOTH	Other Action Taken	Char	200	Other Action Taken	Char	200
	AEBDSYCD	Body System or Organ Class Code	Num	8	Body System or Organ Class Code	Num	8
	AEBODSYS	Body System or Organ Class	Char	80	Body System or Organ Class	Char	80
	AECAT	Category for Adverse Event	Char	40	Category for Adverse Event	Char	40
	AECONTRT	Concomitant or Additional Trtmnt Given	Char	1	Concomitant or Additional Trtmnt Given	Char	2
	AEDECOD	Dictionary-Derived Term	Char	200	Dictionary-Derived Term	Char	200
	AEENDTC	End Date/Time of Adverse Event	Char	64	End Date/Time of Adverse Event	Char	64
	AEENDY	Study Day of End of Adverse Event	Num	8	Study Day of End of Adverse Event	Num	8
	AEENRF	End Relative to Reference Period	Char	20			
	AEENRTPT	End Relative to Reference Time Point	Char	40	End Relative to Reference Time Point	Char	40

Figure 3. Data Set Level Spreadsheet

Two additional spreadsheets are created in a similar format. One lists all variables (across all data sets) that only appear in the old snapshot; the other shows all variables that only appear in the new snapshot.

The values of categorical variables can be compared. This step is optional, as this process can be somewhat time consuming and resource intensive, but the information it provides can be quite valuable.

Categorical variables are identified programmatically by using the SAS FREQ procedure to determine the number of discrete values in each variable. Variables that have fewer discrete values than a pre-specified threshold are deemed to be categorical for these purposes. By default, 20 or fewer discrete values indicate a categorical variable.

Once the categorical variables are identified, a categorical values spreadsheet is created for each data set that has categorical variables in the new snapshot. An example can be seen as Figure 4. For each categorical variable, its values are listed, the frequency and percentage that each value represents are shown – along with differences between old and new. Values that only appear in one snapshot are shown in red for easy identification.

		Counts			Percentages		
Variable	Values	Old	New	Difference	Old	New	Difference
DOMAIN	QS	371242	380121	8879	100.00%	100.00%	0.00%
EPOCH	FOLLOW-UP	37731	45953	8222	10.00%	12.00%	1.90%
	SCREENING	26117	28354	2237	7.00%	7.50%	0.40%
	TREATMENT	307394	305645	-1749	83.00%	80.00%	(2.4%)
	Missing		169			0.00%	
QSCAT	BPI SHORT FORM	105732	104697	-1035	28.00%	28.00%	(0.9%)
	FACT-P	265510	268639	3129	72.00%	71.00%	(0.8%)
	KPS Scale		6785			1.80%	
QSDTC	More than 20 discrete values						
QSDY	More than 20 discrete values						

Figure 4. Categorical Values Spreadsheet

Finally, a numeric variables' summary statistics spreadsheet is created for each data set in the new snapshot that contains numeric variables. An example can be seen as Figure 5. A row of simple, descriptive statistics is shown for all numeric variables that appear in either snapshot.

	N		Minimum		Maximum		Mean		STD	
Variable	Old	New	Old	New	Old	New	Old	New	Old	New
QSDY	369651	378165	-742	-742	716	778	105.0908	108.6304	107.7306	112.5284
QSSEQ	371242	380121	1	1	1479	1507	359.2554	370.0487	254.9425	264.1602
QSSTRESN	353946	364065	0	0	155	155	7.86212	9.207232	19.8587	22.10026
VISITNUM	371122	379889	-1	-1	600120	600140	132786.2	138146.6	201805.8	206520.2

Figure 5. Numeric Variables' Summary Statistics Spreadsheet

These spreadsheets can be quite useful for finding structural and value-related issues that arise as data accumulate over numerous data snapshots, throughout the course of a clinical trial. They do not, however, highlight the way related variables' values might change over time.

CHANGES IN RELATED VARIABLES' VALUES

Tools can be developed to investigate the way related variables' values change over time. These tools could show how, for example, the matching of visit names to visit numbers change over time – or even across data sets within a single snapshot.

PROC CONTENTS or SAS Dictionary Tables can be used to identify data sets that have specific variable pairs, such as VISIT/VISITNUM or all xxTEST/xxTESTCD pairs. They can also be used to identify data sets that have measurement result and measurement unit pairs. Once identified, this information can feed automated executions of PROC FREQ to see how the variable pairs' values relate and how these relationships change over time.

Complex comparisons of more than two variables can be set up in a similar fashion. The key is to identify which relationships are important, and which variables' value relationships could change over time.

CONCLUSION

The statistical analyses and programming tasks associated with clinical trials often have to deal with interim and incomplete data. With thorough planning, ongoing reassessments, and the implementation of effective tools and processes, the issues that arise from using this data can be mitigated.

REFERENCES

1. Pyragraph.com. Remington, Jennifer Kes. Good, Fast, Cheap: You Can Only Pick Two! Los Angeles, CA. May 2, 2013. Available at http://www.pyragraph.com/2013/05/good-fast-cheap-you-can-only-pick-two/
2. Dr. Dobb's. Jones, M. Tim. Defensive Programming. February 01, 2005. Available at http://www.drdobbs.com/defensive-programming/184401915

CONTACT INFORMATION

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

Paul Stutzman
Axio Research, LLC
2601 - 4th Avenue, Suite 200
Seattle, WA 98121
pauls@axioresearch.com

APPENDIX A

This is a sample macro that looks for duplicate records in one or more data sets, based on a specified sort order. This macro can be used to see if that sort order provides record uniqueness within these data sets.

```
%macro FindDups(InDS=, SortOrd=);
/*************************************************************************
* Program: FindDups.sas
* Purpose: Find duplicate records in one or more SAS data sets based on
*          SortOrd.  Duplicate values of SortOrd will appear in Warning
*          messages in the SAS log.  A Note will be written to the SAS
*          log if no duplicates are found.
* Written by: Paul Stutzman, Axio Research LLC
*************************************************************************
* Input Data sets:  &InDS
*************************************************************************
* Output:  Messages in the SAS log
*************************************************************************
*  Parameters:
*    InDS    - Name of the data set(s) to be checked.  If multiple data sets
*              are specified, they should be separated by spaces.
*              Optionally, libname(s) may be specified, as in InData.AE.
*              (required no default)
*    SortOrd - Name of one or more variables that should provide a unique
*              sort order.  If multiple variables are specified, they
*              should be separated by spaces.  (required no default)
*************************************************************************
* Sample Invocations:
*    %FindDups(InDS=DM DV, SortOrd=USUBJID);
*        Data sets DM and DV in the Work library will be checked to see if
*        duplicate values of USUBJID are present in either data set.  If so,
*        Warnings that contain the duplicate values will be written to the
*        SAS log.  If no duplicates are found, a Note will appear in the
*        SAS log.
*
*    %FindDups(InDS=InData.LB, SortOrd=USUBJID VISITNUM LBTESTCD);
*        Data set LB in the InData library will be checked to see if USUBJID
*        VISITNUM and LBTESTCD uniquely describe its records.  If not,
*        Warnings that contain the duplicate values will be written to the
*        SAS log.  If no dups are found, a Note will appear in the SAS log.
*************************************************************************/
%local _SO _i _n;

*  Make sure both parameters were specified  *;
%if (&InDS eq) or (&SortOrd eq) %then
    %put !!! Warning: InDS and SortOrd are required parameters: InDS=&InDS
SortOrd=&SortOrd;

%else %do;
    data _null_;
         *  Get rid of extra spaces in SortOrd  *;
         call symput("SortOrd", strip(compbl("&SortOrd")));
         *  Create a version of SortOrd with equals signs after the variable names  *;
         call symput("_SO", tranwrd(strip(compbl("&SortOrd")), " ", "= ") || "=");
    run;

    options nonotes;
    %let _i = 1;

    *  Loop through all the input datasets  *;
    %do %while (%scan(&InDS, &_i, " ") ne);
        %local DS&_i;
        %let DS&_i = %scan(&InDS, &_i, " ");
```

```
        *  Make sure &&DS&_i exists  *;
        %if not %sysfunc(exist(&&DS&_i)) %then %do;
           %put !!! Warning: &&DS&_i dose not exist.;
           %put !!! FindDups ending.;
           %let _n = %eval(&_i - 1);
           %goto exit;
        %end;
        %else %do;
            *  put duplicate records, if any, in _dups&_i  *;
            proc sort data=&&DS&_i(keep=&SortOrd) out=_tmp nodupkey dupout=_dups&_i;
                 by &SortOrd;
            run;
            %let _i = %eval(&_i + 1);
        %end;
    %end;
    %let _n = %eval(&_i - 1);

    *  write messages to the log for each input dataset  *;
    %do _i = 1 %to &_n;
        data _null_;
             *  open _dups&_i.  nobs will be 0 if there are no duplicates  *;
             dsid = open("_dups&_i");
             if (attrn(dsid, "nobs") < 1) then
                put "##" "# Note: No duplicates found in &&DS&_i with
SortOrd=&SortOrd";
             rc = close(dsid);

             *  write duplicates to the log  *;
             set _dups&_i;
             put "!!" "! War" "ning: Duplicates in &&DS&_i: " &_SO;
        run;
    %end;
    options notes;

%exit:
    *  Clean up  *;
    proc data sets library=work nolist;
         delete _tmp _dups1%if (&_n>1) %then %str(-_dups&_n);;;
    quit;
    run;
%end;
%mend FindDups;
```

APPENDIX B

The tool mentioned in the Comparing Old and New Data Snapshots section of this document is described here. The macro itself is not provided due to size constraints.

This is a SAS macro that invokes several smaller internal macros. It reads an entire study folder and its associated \Previous folder, and compares the data sets within. It is expected that the base folder contains the newest data snapshot, and its subordinate \Previous folder contains the previous data snapshot.

The macro has four parameters:

- Folder (required, no default): The name of the folder to compare with its \Previous folder.
- Values (not required): Determines the data sets, if any, whose values will be compared. Values= can take on the following values:
  - missing (the default value): Values will not be compared.
  - an integer: Variables in data sets that have this number of observations or fewer will be compared.
  - a list of data set names, separated by spaces: The values of the variables in the specified data sets will be compared.
  - _all_: Variables in all data sets will be compared. This provides the most comprehensive information, but it could cause the program to run for a long time.
- MaxCats (not required): Maximum number of discrete values that will be shown for categorical variables, if values are being compared. The default is 20 (i.e. if a variable has 20 or fewer discrete values, those values will be compared and shown).
- Debug (not required): Set to "yes" if all messages are to be written to the SAS log and temporary data sets are to be kept. The default is "no", which means messages from the variable values comparison section of the program will be suppressed, and all temporary data sets will be deleted.

Here are some sample invocations of the macro:

```
%mLibComp(Folder=RawData);
```

This invocation compares all the data sets in the \RawData folder (the current snapshot) with the data sets in \RawData\Previous (the old snapshot). It does not compare the values of categorical variables.

```
%mLibComp(Folder=Analysis, Values=_all_);
```

This invocation compares all the data sets in the \Analysis folder (the current snapshot) with the data sets in \Analysis\Previous (the old snapshot). It compares the values of all categorical variables. Categorical variables are those variables that have 20 (the default) or fewer discrete values.

```
%mLibComp(Folder=SDTM, Values=DM EX, MaxCats=25);
```

This invocation compares the DM and EX data sets in the \STDM folder (the current snapshot) with their corresponding data sets in \SDTM\Previous (the old snapshot). It compares the values of all categorical variables in these data sets. Categorical variables are those variables that have 25 or fewer discrete values.

The macro writes the output from its comparisons to a number of spreadsheets contained in a Microsoft EXCEL workbook. It uses the ODS EXCELXP tagset to accomplish this. The Comparing Old and New Data Snapshots section of this paper shows examples of each type of comparison spreadsheet.

Here is the flow of the macro:

1. Input parameters are checked.
2. PROC CONTENTS is used to get data set and variable information for both the old and new snapshots.
3. The SAS SQL procedure creates two pairs of data sets: lists of data sets (and their record counts) in the old and new, and lists of variables (and their attributes) in the old and new.
4. A set of DATA steps look for and flags mismatches between data sets and variables in the old and new, and they create subset data sets for subsequent reporting.
5. The ODS EXCELXT tagset environment is set up, and the output EXCEL workbook (in XML format) is opened.

6. The SAS REPORT procedure is invoked to create a data sets level spreadsheet. This spreadsheet and the ones created in the next three steps are based on data sets created in Step 4 above.
7. PROC REPORT creates the two spreadsheets: one that shows all variables that only appear in the old snapshot, and one that shows all variables that only appear in the new snapshot.
8. PROC REPORT creates a spreadsheet that shows all variables that have differing attributes between snapshots.
9. PROC REPORT creates a spreadsheet for each data set in the new snapshot that shows all variables (from either snapshot) along with their attributes.
10. If value comparisons are to be done, the following occurs:

    A DATA _NULL_ step reads the list of all variables created in Step 4 above. It uses BY processing to perform the following tasks for each data set in the new snapshot.

    a. A macro is called for each variable in the data set. It uses invocations of PROC FREQ to find the frequency of each value in the variable, and to determine the number of discrete values for that variable in the data set, in both the old and new snapshots. This information gets appended a cumulative results data set for this snapshot data set.

    b. A macro variable that contains a list of all numeric variables is created.

    c. After the last variable record for a data set has been processed, two macros are called: one that produces a categorical values report that includes values for variables that meet the criteria for being considered a categorical variable, and one that produces a simple descriptive statistics report for all numeric variables.
11. General cleanup occurs, and the XML workbook is opened in Microsoft EXCEL, so it can be saved as an XLSX workbook.