Paper 3309-2015

Snapshot SNAFU: Preventative Measures to Safeguard Deliveries

Spencer Childress, Rho®; Alexandra Buck, Rho

ABSTRACT

Little did you know that your last delivery ran on incomplete data. To make matters worse, the client realized the issue first. Sounds like a horror story, no? A few preventative measures can go a long way in ensuring that your data are up-to-date and progressing normally. At the data set level, metadata comparisons between the current and previous data cuts will help identify observation and variable discrepancies. Comparisons will also uncover attribute differences at the variable level. At the subject level they will identify missing subjects. By compiling these comparison results into a comprehensive scheduled e-mail, a data facilitator need only skim the report to confirm that the data is good to go--or in need of some corrective action. This paper introduces a suite of checks contained in a macro that will compare data cuts at the dataset, variable, and subject levels and produce an e-mail report. The wide use of this macro will help all SAS® users create better deliveries while avoiding rework.

INTRODUCTION

Three basic components comprise a database: tables, columns, and rows. Various types of metadata describe each component and comparisons of those metadata between data snapshots help identify potential issues. For the purposes of this topic 'tables' and 'datasets' are synonymous as are 'columns' and 'variables'. However, rather than consider each row we consider ID values which link rows among datasets. For instance in a clinical trial a database typically contains a subject-level dataset in addition to longitudinal, parameter-wise, and event-wise datasets which may contain any number of records per subject.

This article focuses on the application and utility of **%SnapshotCompare**, a SAS macro which compiles metadata and generates an email report summarizing comparisons at the dataset, variable, and ID levels. This tool allows SAS users keep to tabs on the progression of their data at each snapshot or 'cut' of data. The macro contains code specific to the Windows operating system and may need to be modified for a UNIX environment. **%SnapshotCompare** requires only that the user know how to call a SAS macro. The internals of the macro however require an understanding of macro programming, operating system statements, PROC SQL, data step programming, and HTML formatting.

THE MACRO

We need a few inputs in order to compare two snapshots and send the report to pertinent recipients:

- Required:
  - Physical locations of the snapshots
  - Recipients' email addresses
  - The ID-level variable name
- Optional:
  - The level of detail of the report
    - Dataset, variable, and/or ID level
  - The name of the project to print in the subject of the email

To this end the macro call is as follows:

```
%SnapshotCompare(Path1   = ,
                 Emails  = ,
                 Path2   = ,
                 IDVar   = ID,
                 Detail  = ALL,
                 Project = );
```

Detailed logic allows for a number of snapshot directory setups. For example if C:\Project\Data contains all snapshots in individual subfolders, the call can be as simple as:

```
%SnapshotCompare(Path1    = C:\Project\Data,
                 Emails   = jane_doe@gmail.com,
                 Project = Example Project);
```

In this example **%SnapshotCompare** would determine and compare the two most recently created folders in C:\Project\Data and the subject line of the email report would read 'Example Project Snapshot Comparison – <today's date>'.

METADATA COMPARISONS

Data snapshots break down into three levels: the dataset level, the variable level, and the ID level. Below we discuss how to capture metadata of interest at each level.

DATASET-LEVEL COMPARISON

Dataset-level metadata includes existence, size, number of variables and observations, and file creation and modification date times. SAS provides a number of ways to capture and compare these attributes and we cover two complementary methods.

In order to peer into each snapshot we assign a LIBNAME to each, '_PATH1_' to the current snapshot and '_PATH2_' to the previous one. The PROC SQL DICTIONARY view TABLES can then provide a simple yet thorough summary of number of observations and number of variables among other attributes:

```
proc sql;
  %*&Path1 datasets;
    create table Path1Datasets as
        select *
            from dictionary.tables
              where libname = '_PATH1_';
  %*&Path2 datasets;
    create table Path2Datasets as
        select *
            from dictionary.tables
              where libname = '_PATH2_';
quit;
```

By generating and merging two datasets, one for each snapshot, we can easily compare metadata and additionally determine new and missing datasets based on datasets present in one library but not in the other.

We can capture file size, file creation date time, and file modification date time from the operating system itself with the external file suite of functions:

```
filename1 = filename('Path1DS', cats("&Path1\", memname, '.sas7bdat'));

fileopen1  = fopen('Path1DS');
    size1     = input(finfo(fileopen1, 'File Size (bytes)'), best.);
    created1  = input(finfo(fileopen1, 'Create Time'), datetime19.);
    modified1 = input(finfo(fileopen1, 'Last Modified'), datetime19.);
fileclose1 = fclose(fileopen1);
```

We opt out of DICTIONARY.TABLES file size, creation date time, and modification date time metadata because the operating system keeps more up-to-date file metadata than do SAS datasets. For example suppose a user copies a SAS dataset from one location to another. Its operating system creation date time will update but its SAS creation date time will not.

Finally with all dataset-level metadata compiled and compared, we apply a little HTML formatting to produce the report:

Dataset-level comparison:

--> New datasets: AHSTMSTR, DOGMSTR
--> Missing datasets: LATXMSTR, VASWMSTR

Dataset	Number of Variables	Number of Observations	File Size (bytes)	Created	Modified
AEPGMSTR	148	2	33792	09OCT2014:10:12:49	09OCT2014:03:11:25
AEXPCODE	119	3850	3621888	24OCT2014:15:08:32	24OCT2014:15:08:33
AEXPMSTR	100	3605	19710976	09OCT2014:10:12:49	09OCT2014:03:11:29

Display 1. Example of Dataset-level Comparison Output

Traffic lighting highlights potential issues as well as expected file progressions. Expected changes appear green while unexpected changes appear red. We expect the later data cut to have at least the same number of observations and variables and later creation and modification date times. PUT() statements print the metadata comparisons out and provide structure, text formatting, and table formatting. For instance we define the general table format like so:

```
put '<head>';
put '<style>';
put '  table, th, td {';
put '    border:          1px solid black;';
put '    border-collapse: collapse;';
put '  }';
put '  th, td {';
put '    padding: 5px;';
put '  }';
put '  th {';
put '    background-color: lightblue;';
put '  }';
put '</head>';
put '</style>';
```

Dataset indicators of first and last records in the data step like the automatic variable _N_ and the SET statement option END allow us to instantiate and close out an HTML table (and instantiate its headers) once while the HTML table data prints iteratively with each record in the SAS dataset. In order to populate the table data we simply sandwich every column value between the HTML table row tag '<tr>':

```
put '  <tr>';
put '  ' dataset;
put '  ' nvar;
put '  ' nobs;
put '  ' size;
put '  ' created;
put '  ' modified;
put '  </tr>';
```

VARIABLE-LEVEL COMPARISON

Variable-level metadata includes existence, type, length, format, informat, and label. The PROC SQL DICTIONARY view COLUMNS provides these attributes and much like the dataset-level metadata we can generate a dataset of each snapshot's variable-level metadata:

```
proc sql noprint;
  %*&Path1 variables;
    create table Path1Variables as
        select *, upcase(memname) as dataset, upcase(name) as variable
            from dictionary.columns
              where libname = '_PATH1_'
    order by dataset, variable;
```

```
  %*&Path2 variables;
    create table Path2Variables as
        select *, upcase(memname) as dataset, upcase(name) as variable
            from dictionary.columns
              where libname = '_PATH2_'
    order by dataset, variable;
quit;
```

Aggregating our results, we output a vertical table with a row for each attribute within each dataset along with a list of variables with mismatches for a given attribute:

Variable-level comparison:

Dataset	Attribute	Variables with Mismatches
RANDMSTR	Type	RANDYN
	Informat	RANDYN
	Label	RANDYN
	Length	RANDYN
RQLQMSTR	Missing	ID, INSTANCEID, INSTANCENAME, INSTANCEREPEATNUMBER

Display 2. Example of Variable-level Comparison Output

New and missing variables as well as variables with differing type highlight red because these attributes are more likely to cause issues in downstream programs than are format, informat, label, and length. We present problem variables in a comma-delimited list.

ID-LEVEL COMPARISON

ID-level metadata comprises a short list: new IDs, missing IDs, and IDs with fewer records than the previous snapshot. In order to aggregate datasets, IDs, and number of records per dataset per ID we stack and count in a PROC SQL CREATE TABLE statement:

```
proc sql;
  %*Collapse datasets to one record per dataset per ID value,
    count number of records per dataset per ID value,
    and stack resulting datasets.;
    create table Path1IDs as
        %do i = 1 %to %sysfunc(countw(&Datasets));
            %let Dataset = %scan(&Datasets, &i);

                select "&Dataset" as Dataset,
                                    &IDVar,
                       count(1)   as nID1
                    from _path1_.&Dataset
                group by &IDVar

            %if &i lt %sysfunc(countw(&Datasets))
                %then      outer union corr;
                %else %str(order by Dataset, &IDVar;);
        %end;
quit;
```

&Datasets is a space-delimited list of datasets in the snapshot which we iterate over to generate a SELECT query for each dataset and stack the resulting output together with OUTER UNION CORR operators. Like the dataset- and variable-level comparisons, once we manipulate both snapshots' metadata into like form, i.e. one record per dataset per ID, we merge, compare, and format the metadata into a simple report:

ID-level comparison:

Dataset	SUBJECT Status	SUBJECTs
INSCMSTR	Missing	12324
MOLDMSTR	New	11192
SANCMSTR	Missing	12050
VITLMSTR	Missing	10674, 11670, 11840, 12000
	Fewer Records	10124 (-1), 10141 (-1), 10251 (-1), 10290 (-1), 10372 (-1), 10454 (-1), 10471 (-1), 10531 (-1), 10592 (-1), 10850 (-2), 10954 (-1), 11043 (-1), 11054 (-1), 11164 (-1), 11202 (-2), 11241 (-1), 11373 (-1), 11532 (-2), 11593 (-1), 11603 (-1), 11692 (-1), 1179 1 (-1), 11801 (-1), 11950 (-1), 12110 (-1), 12121 (-1), 12154 (-1), 12193 (-1), 12270 (-1)

Display 3. Example of ID-level Comparison Output

In this example SUBJECT is the ID variable and we highlight statuses of missing and fewer records red. Note that ID values with fewer records print with the change in number of records to give an idea of the scale of the issue. Like the variable-level table's attributes we present statuses vertically and problem IDs in a comma-delimited list.

EMAIL REPORT

With the metadata and HTML formatting in place, all that remains is to sandwich everything in a FILENAME statement with the EMAIL (SMTP) access method. The FILENAME syntax is straight-forward:

```
%*Create email file.;
  filename snapshot
           email(%do i = 1 %to %sysfunc(countw(&Emails, %str( ))));
                    %let Email = %scan(&Emails, &i, %str( ));

                    "&Email"
                 %end;)
           ct       = 'text/html'
           subject = "&Project Snapshot Comparison - &Today";
```

The %DO loop on the EMAIL line iterates through the list of emails in the &Emails argument to **%SnapshotCompare** to enclose each email address in quotes. The CT option is an alias for content type and we specify 'text/html' to indicate the output file is text-based with HTML formatting. As one might expect the SUBJECT option defines the subject line in the outgoing email. To the fileref named SNAPSHOT we print our metadata dataset in a data step with a FILE statement:

```
%*Write snapshot comparison report to email.;
  data _null_;
      file snapshot notitles;
      set Metadata;

     *Not HTML formatting, abridged from full macro code;
      put '<Greeting>';
      put '<Description of Comparison with Hyperlink Snapshots>';
      put '<Comparison Tables>';
  run;
```

And that's the last step. Readers will find the full code in the supplemental text file.

SUMMARY

In summary the final emailed report requires three primary steps:

- Assign LIBNAMEs to each snapshot
- Compare metadata at the:
    - Dataset level
    - Variable level
    - ID level
- Print metadata comparisons to an email

Several intermediate steps round out the macro including code which checks input arguments, generates hyperlinks to local and network drives, and customizes the report.

CONCLUSION

SAS provides a number of tools to capture and track metadata. One desire across all disciplines and industries is trustworthy data. A few backend database checks provide a sense of integrity in a snapshot with a concise summary of metadata metrics. Data facilitators can set up a scheduled task to run **%SnapshotCompare** periodically or run the macro manually for intermittent deliveries. We encourage SAS users to test the tool out and send us feedback.

RECOMMENDED READING

- *http://www.w3schools.com/html/*
- *http://www.regular-expressions.info/*

CONTACT INFORMATION

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

Alexandra Buck	Spencer Childress
alexandra_buck@rhoworld.com	spencer_childress@rhoworld.com
Rho	Rho