Paper PH01

Automated or Manual Validation: Which One is for You?

Richann Watson, i3, Batavia, OH
Patty Johnson, i3, San Diego, CA

ABSTRACT

Validation is an essential part of the derived data set, summary table and listing development process. Validation can require that an independent program be created that confirms the results of the original program are accurate. It is common practice that derived data sets are validated using a more automated method, but it is not as common that table and listing outputs are validated using an automated method. A validation programmer often manually compares the results of her independent programming to the original production output or results. This entails checking not only the titles and footnotes and other cosmetic aspects of the output but also the informational aspect of the output. The manual method of checking the informational aspects of the output is a time consuming process and is subject to human error. This paper presents an automated approach for performing a 100% validation of the informational portion of summary table and listing outputs.

INTRODUCTION

The motivating factor for developing this process was the requirement of having to generate and validate over 400 adverse event tables within a short time frame and limited budget. All of the outputs, however, were programmed using only 2 SAS® programs by utilization of SAS macro programming. Manually comparing the production output to the validation results would not have been feasible in the allotted time without additional resources. Additionally, since only 2 programs generated 400 outputs, it made sense to have only 2 programs generate the validation results which automatically limited the number of resources that could be used anyway. How to accurately and efficiently validate over 400 tables within the allotted timeframe and budget was the main concern. As a result of this concern we developed an automated process of validation. The automated process has a 3 step approach.

STEP 1: PRODUCING TABLE AND LISTING OUTPUT DATA SETS

In order to automate the validation of the results, the production programmer (i.e. the programmer responsible for producing the production table or listing) will create a permanent SAS data set containing the data used to generate the summary table or listing. We will refer to this data set throughout this paper as the production data set. The production data set can be produced in several ways:

- Create a permanent SAS data set after the data is formatted according to the table or listing specifications. This data set will subsequently be read into the printing/reporting tool that is used for generating the production output. Only the variables required for creating the output should be retained in the data set.
- If the REPORT procedure is used to produce the output, then the OUT= option can be used to produce a permanent SAS data set. This paper will focus on using OUT= option in PROC REPORT to create the production data set.

PROC REPORT OUT= OPTION

The OUT= option of PROC REPORT will produce a production data set that contains one observation for each report row and one observation for each unique summary line. It also contains one variable for each column of the report. PROC REPORT tries to use the name of the variable specified in the COLUMN statement in the output data set. Therefore, it is important that the columns have unique names otherwise SAS will create variable names based on the column number (i.e. _C1_, _C2_, etc.) In addition to the variables from the COLUMN statement, the production data set will contain a character variable named _BREAK_ which is automatically generated by PROC REPORT. The value of _BREAK_ is based on the type of record in the production data set.

- If the record is a detail row, then _BREAK_ is missing or blank.
- If the record is summary line, then _BREAK_ is the name of the break variable that is associated with the summary line, or _RBREAK _ .
- If the record is derived from a COMPUTE BEFORE/AFTER _PAGE_ statement, then _BREAK_ is _PAGE_. Note that this record only appears in the data set and does not appear on the actual output.

The production data set will contain a record for each row generated from the 'break after …' statement and from the compute block. In order for the data set to only contain the records with data that are to be checked (since the validation programmer most likely will not be using PROC REPORT to generate their validation data set), a where

clause is necessary so the extra rows that are produced from the 'break after …' and the compute block are removed. The where clause should be in the form of "(where=(_BREAK_ not in ("*break_variable"* "_PAGE_))", where *break_variable* represents the variable specified in the "break after …" statement.

SAMPLE CODE

```
libname PROD "directory where SAS data set from PROC REPORT is stored";

PROC REPORT DATA=indsn SPLIT='~' SPACING=1 MISSING NOWINDOWS HEADLINE
            OUT=PROD.OUTDSN (where=(_BREAK_ not in ('SUBJID' '_PAGE_')));
      COLUMN (ord1 TRTDOSE ord2 DISCAT USUBJID DOSESD diagdt C_STAGE);

  DEFINE ord1    / ORDER   NOPRINT;
  DEFINE TRTDOSE / ORDER   WIDTH=20  LEFT  FLOW "Treatment~Dose";
  DEFINE ord2    / ORDER   WIDTH;
  DEFINE DISCAT  / ORDER   WIDTH=30  LEFT  FLOW "Disease~Type";
  DEFINE USUBJID / ORDER   WIDTH=20  LEFT  FLOW "Subject";
  DEFINE DOSESD  / DISPLAY WIDTH=12  RIGHT      "First Dose~Date";
  DEFINE diagdt  / DISPLAY WIDTH=20  RIGHT      "Date of~Diagnosis";
  DEFINE C_STAGE / DISPLAY WIDTH=22  LEFT  FLOW "Disease~Entry";
  BREAK AFTER SUBJID/SKIP;

  COMPUTE BEFORE _PAGE_ / LEFT;
    LINE @1 &ls * '-';
  ENDCOMP;
run;
```

STEP 2: VALIDATING TABLE AND LISTING OUTPUTS

The validation programmer can perform 100% validation on the content of the output (from the example in Step 1, PROD.OUTDSN – the production data set) using the following method.

1. Create a program that will re-produce the results of the table or listing and store the results in a SAS data set. This data set will be referred to throughout this paper as the validation data set.
2. The production programmer and validation programmer should agree on variable names, formats, and sort orders for the data ahead of time. Typically it makes sense for the production programmer to make the decision and then share that information with the validation programmer.
3. Use the COMPARE procedure to compare the production data set generated in Step 1 with the validation data set.

In order for the validation of the results to be fully automated, the following options should be used in PROC COMPARE when performing the final comparison of the production data set and the validation data set:

- **OUT=** Creates a SAS data set of the results from the PROC COMPARE.
- **OUTNOEQUAL** Suppresses the writing of observations to the SAS data set specified in OUT= when all values are equal.
- **OUTBASE** Writes an observation for each observation in the BASE= data set .
- **OUTCOMP** Writes an observation for each observation in the COMPARE= data set.
- **OUTDIF** Writes an observation that contains the differences for each pair of matching observations.

The data set produced from PROC COMPARE OUT= should be stored as a permanent SAS data set for future use. This data set will be referred to throughout this paper as the compare data set.

The compare data set will either

- be an empty data set if there are no discrepancies, or
- it will contain the following types of records (_TYPE_)
    - **BASE** Record from the BASE= data set that is different from the corresponding COMPARE= data set.
    - **COMPARE** Record from the corresponding COMPARE= data set that is different from the corresponding BASE= data set.
    - **DIF** Record that shows the discrepancies between the previous BASE and COMPARE records.
        - For character variables the discrepancy will be shown with an 'X'
        - For numeric variables the discrepancy will be the difference of BASE and COMPARE

SAMPLE CODE

```
libname PROD "directory where the production data set is stored";
libname VAL  "directory where the validation data set is stored";
libname COMP "directory where the compare data set is stored";

PROC COMPARE    BASE=PROD.OUTDSN (DROP=_BREAK_)
             COMPARE=VAL.v_outdsn LISTALL
                 out=COMP.OUTDSN outnoequal outbase outcomp outdif;
  ID ord1 TRTDOSE ord2 DISCAT USUBJID;
run;
```

It is recommended that the compare data set be stored in a separate location from the production data set and that the files have the same name. Requiring the two types of SAS data sets have the same file names helps to identify which compare data set corresponds with which production data set.

SAMPLE COMPARE DATA SET

TYPE	_OBS_	ord1	TRTDOSE	ord2	DISCAT	USUBJID	DOSESD	diagdt	C_STAGE
BASE	2	1	TRT A	1	DISEASE A	10001	16-Jul-08	4-Aug-08	STAGE 1
COMPARE	2	1	TRT A	1	DISEASE A	10001	16-Jul-08	4-Aug-08	Stage 1
DIF	2	E	……..	E	……..	10001	E	E	.XXXX..
BASE	4	1	TRT A	2	DISEASE B	10002	3-Oct-07	3-Oct-07	STAGE 2
BASE	5	2	TRT B	3	DISEASE C	10003	24-Oct-07	14-Nov-07	STAGE 2
COMPARE	5	2	TRT B	3	DISEASE C	10003	24-Oct-07	4-Nov-07	Stage 2
DIF	5	E	…..	E	……..	10003	E	-10	.XXXX..
COMPARE	6	1	TRT A	2	DISEASE B	10004	24-Oct-07	14-Nov-07	Stage 3

If the data sets do not match exactly, with the specified options used, a compare data set containing the discrepancies will be produced. This data set shows that for observations 2 and 5, a match was found on the ID variables between the production and validation data sets but the last variable C_STAGE had different values. In addition, observation 5 shows that the values of variable DIAGDT are different between BASE and COMPARE. The data set also shows that for observation 4 there are no corresponding records (note the absence of records where _TYPE_ is equal to COMPARE or DIF). This indicates that there was no matching record with the same ID variables in the validation data set when compared to the production data set. A similar situation occurred for observation 6. There is no matching record in the production data set with the same ID variables as the validation data set.

STEP 3: SUMMARIZING THE VALIDATION RESULTS

In practice, we can individually open up each compare data set and inspect the number of observations. If the data set has zero observations we can quickly ascertain that the production and validation data sets are equal therefore confirming the content of the output is validated by independent programming. If there are in fact differences, we can further inspect the compare data set to see what those differences are. The time it would take to do this, however, escalates quickly with the number of outputs.

If the compare data sets are permanently stored in a separate folder, a SAS program can be used to automatically check the data sets and quickly summarize which outputs passed and which outputs failed the validation. Only the outputs which failed the validation would need to have their compare data sets opened and inspected for the reason for the failure.

The theory behind the program is to read in all of the compare data sets and check for the number of observations. If the data set has zero observations we can quickly ascertain that these outputs have passed the validation. If however, the data set has any number of observations greater than zero, it would be necessary to look further at that data set to determine the reason for the failure. Keep in mind that, due to limitations discussed below, a failure does not necessarily mean the production output or the validation output are incorrect or do not match. It might purely be that variable names or formats are not matching which would require further collaboration between the validation and the production programmers.

The format of the report should include at a minimum the name of the data set being compared (relating to the name of the program or output being validated), the number of observations in the compare data set, and a Pass/Fail indicator. Below is a sample of what the summary report of the validation results of 5 outputs might look like.

SAMPLE COMPARE REPORT

Compare Data set Name	Number of Observations in Compare Data set	Validation Pass/Fail
AESUM1_S.sas7bdat	6	FAIL
AESUM2_S.sas7bdat	0	PASS
AESUM3_S.sas7bdat	0	PASS
AESUM4_S.sas7bdat	0	PASS
AESUM5_S.sas7bdat	0	PASS

SAMPLE, BASIC CODE TO CREATE A REPORT SUCH AS THE ONE ABOVE FOLLOWS:

```
%Macro createrpt (include any macro parameters such as Directory and the report
name and destination);

Filename source pipe "Directory";

** Read each compare data set for the number of observations. **;
    Data one;
       Infile source lrecl=2048 pad truncover scanover end=eof;
       Input dsname $1-50;
       Call symput('ds'||trim(left(_n_)),trim(left(dsname)));
       If eof Then call symput('numds',trim(left(_n_)));
     Run;

** Start scanning outputs by reading in each compare data set one by one **;
%Do j=1 %To &numds;
      Data two;
        Set "Directory\&&ds&j" ;
     Run;
     Data _null_;
        If 0 Then Set two nobs=final;
        Call symput("fnlobs",left(put(final,8.)));
        Stop;
      Run;

** If the number of observations (&fnlobs) is 0 the data set has passed the **;
** validation. If the number of observations in a data set is not 0, the    **;
** data set failed the validation.                                          **;
      %If &fnlobs^=0 %Then %Do;
         Data dsck (keep=dsname numobs valcom);
            Set two;
        Length  dsname $50 numobs $25 valcom $25;
        dsname=%upcase(scan("&&ds&j",1,'.'))||".sas7bdat";
        numobs="&fnlobs";
           valcom='FAIL';
        Run;
        Proc sort Data=dsck nodupkey;
       By _all_;
        Run;
      %End;
      %Else %Do;
       Data dsck;
          Length  dsname $50 numobs $25 valcom $25;
          dsname=%upcase(scan("&&ds&j",1,'.'))||".sas7bdat";
          numobs='0';
              valcom='PASS';
           Run;
       %End;
```

```
** Append results of checking all the data sets together into one data set **;
      %If &j=1 %Then %Do;
        Data allds;
       Set dsck;
      Run;
     %End;
     %Else %Do;
       Proc append base=allds
         new = dsck;
       Run;
      %End;
    %End;

%Mend createrpt;
```

… Sort and print the report either using Proc Print or Proc Report

LIMITATIONS

This type of validation does have some limitations. If the production and validation data sets do not have the same structure (e.g. variable names, variable lengths, formats) or are otherwise generated using a different method (e.g. using PROC SORT instead of PROC RANK to order records) the data sets will not match which does not necessarily mean the production output or the validation data is not valid. More upfront collaboration between the production programmer and the validation programmer is required.

The validation programmer must thoroughly check the results of the Proc Compare to make sure they are not missing any variables in their validation data set. If a variable exists in the production data set but not the validation data set, the compare data set can still have 0 observations, but the particular variable that was not created in the validation data set will not have been checked.

In addition, this method only checks the results or content of the output. It does not check for cosmetic issues. Therefore, after the production data set and the validation data set matches exactly, the output still needs to be reviewed for any cosmetic issues.

CONCLUSION

Regardless of the limitations of this process, it is still an efficient way to check numerous amounts of output in a short time frame. In addition, this process allows the lead programmer to run all the tables and listings production programs and the tables and listings validation programs when new data has been loaded and check to see if all the tables and listings still pass validation with the new data.

REFERENCES

SAS Help and Documentation on-line

ACKNOWLEDGMENTS

Thanks to our colleagues at i3 Statprobe, who reviewed this paper and supported us while we worked through this process.

CONTACT INFORMATION

Richann Watson
i3
richann.watson@i3statprobe.com

Patty Johnson
i3
patricia.johnson@i3statprobe.com