SESUG Paper 153-2018

Obtain better data accuracy using reference tables

Kiran Venna, SMACT Works, Inc
Suryakiran Pothuraju, Unify solutions Inc

Abstract

Data accuracy can be improved tremendously by using reference tables, especially when data is loaded from external files into target tables. Checking the data accuracy of source tables can be easily done with help of reference tables. By using SAS® macro, data accuracy for many files can be performed with a single reference table. Step by step approach to build reference tables and a way to automate the data accuracy checks will be discussed in this paper.

Keywords

Metadata check, data accuracy, reference table, macro variable, PROC SQL, Dictionary.columns, PROC REPORT, SAS Macro.

Introduction

Data accuracy is very important component of data quality. Inaccurate data often leads to very ambiguous insights, hampering decision making ability of large companies. Data accuracy consists of two main components, having data in right form with right values.

Data in right form can be maintained by comparing source metadata with reference table metadata. Right values in target table can be managed by allowing right values in the target table with help of reference table values. Different ways to check inaccurate data in external files by using reference tables is topic of this paper.

Topics covered in this paper are

1. Metadata check for each table using appropriate reference table.
2. Data check for many tables using a single reference table.

Metadata check for each table using appropriate reference table

Checking the accuracy of metadata requires building of a reference table and checking metadata of source tables and consists of four steps.

a) Create reference table with appropriate metadata.
b) Compare column names of source tables with reference table.
c) Validate length, position and data type of columns in source table with reference table.
d) Reporting results of metadata check.

a) Create reference table with appropriate metadata.

First step in metadata check is to create a reference table, example of the same is shown below.

```
proc sql;
   create table ref_paylist
       (IdNum char(4),
        Gender char(1),
        Jobcode char(3),
        Salary num,
        Birth num,
        Hired num informat=date7. format=date7.);
```

b) Compare column names of external file with reference table

In second step, column names of external file are compared with that of reference tables. This is done by using dictionary.columns on external file after importing as a SAS Dataset. In the code for column name comparison (presented below) when all column names match then macro variable value will be set up as “ALL COLUMN NAMES MATCHED” else mismatched column names are captured. Information about matched or unmatched columns is stored in a macro variable using PROC SQL into clause.

For purpose of this paper, test_playlist1 is the sample SAS Dataset, against which metadata checks will be done. Code for creating test_playlist1 SAS Dataset is provided in the appendix. Code for column name matching is shown below.

```
proc sql noprint;
select
       case when name is missing
            then "ALL COLUMN NAMES MATCHED"
             else name
       end into :COLUMN_NAME_MATCHORNOMATCH SEPARATED by ','
       from
           (select coalesce(a.name, b.name) as name, count(*)
                   from (Select name
                         from   Dictionary.columns
                         where upcase(libname)= upcase('work')
                           and upcase(memname) = upcase('ref_paylist'))a
     full join
            (Select name
                         from   Dictionary.columns
                         where upcase(libname)= upcase('work')
                           and upcase(memname) = upcase('test_paylist1'))b
      on upcase(a.name) =upcase(b.name)
      where a.name is missing
      or b.name is missing);
```

c) Validate length, position and data type of columns of source table with reference table

Third step is to validate length, position and type of the columns matched in above step. This is done using an inner join with dictionary.columns of reference table with that of external file (after importing into a SAS Dataset). When columns attributes are matched, the macro variable is set up as “ALL COLUMNS Type position length MATCHED”, else mismatched column names are captured. Matched or unmatched columns information depending on length position or type of column are stored in macro variable using PROC SQL into clause. Code for checking length, position and type of the columns is shown below.

```
proc sql noprint;
select
       case when name is missing
            then "ALL COLUMN Type position length MATCHED"
             else name
       end into :COLUMN_Values_MATCHorNOMATCH SEPARATED by ','
       from
           (select coalesce(a.name, b.name) as name, count(*)
                  from (Select name, varnum, type, length
                        from   Dictionary.columns
                        where upcase(libname)= upcase('work')
                          and upcase(memname) = upcase('ref_paylist'))a
inner join
            (Select name, varnum, type, length
                        from   Dictionary.columns
                          where upcase(libname)= upcase('work')
                          and upcase(memname) = upcase('test_paylist1'))b
      on upcase(a.name) =upcase(b.name)
      where a.varnum ne b.varnum
or a.type ne b.type
or a.length ne b.length);
```

d) Reporting results of metadata check

Reporting metadata check is further sub divided into 2 steps, firstly above macro variables are captured in a work table as shown below.

```
Data Temp;
  Length column $500.;
   If index(strip("&COLUMN_NAME_MATCHORNOMATCH" ), "ALL COLUMN")>0
   then column ="&COLUMN_NAME_MATCHORNOMATCH";
   else column = catx(' ', "Unmatched columns is/are", "&COLUMN_NAME_MATCHORNOMATCH"
                      );
    output;
    If index(strip("&COLUMN_Values_MATCHorNOMATCH " ), "ALL COLUMN")>0
    then column ="&COLUMN_Values_MATCHorNOMATCH ";
    else column =catx(' ', "Unmatched columns for type or length or position
                is/are","&COLUMN_Values_MATCHorNOMATCH ");
 output;
run;
```

Finally, a report is created using PROC REPORT, if the metadata is matched then report will have green background and else the back ground is red. Sample Report is presented in Appendix section. Code for PROC Report is shown below.

```
Title "Final info about metadata for test_paylist1 table";

Proc report data =temp nowd;
   column column;
   define column/display "Column";
   compute column;
   if index(column,  "ALL COLUMN") = 0 then call define
      (_col_, "style", "style={background = red}");
    else  call define
      (_col_, "style", "style={background = green}");
     endcomp;
 run;
```

This concludes the first topic of this paper “Metadata check of tables using appropriate reference table”.

Data check for many tables using a single reference table.

For metadata check, a reference table is needed for each table, but for data check a single reference table can be used to check many external files or source tables. First step in data check is to create a reference table with all columns of interest. An example of reference table creation is shown below.

```
proc sql;
create table referencetable
  (column_name char(55),
  column_value char(20),
  column_flag char(1),
  column_bucket num);
 quit;
```

In the above reference table, column_name indicates name of the column which needs to be compared with source table. Column_values are allowed values for a particular column and can be used to compare external file/source tables column records. Column_flag usually is “Y” indicating the values are correct. Column_bucket is used to analyze various variables that fall in simple groups. Bucketing gives higher flexibility to analyze multiple variables in a similar fashion.

Second step is to populate the reference table. An example to populate reference table is shown below.

```
Proc sql;
Insert into referencetable values('Gender', 'F', 'Y',1);
Insert into referencetable values('Gender', 'M', 'Y',1);
```

```
Insert into referencetable values('Jobcode', '123', 'Y',1);
Insert into referencetable values('Jobcode', '111', 'Y',1);
Insert into referencetable values('Jobcode', '121', 'Y',1);
Insert into referencetable  values('Salary', '1000', 'Y',2);
Insert into referencetable  values('Birth', '/[0-9]{8}/', 'Y',3);
```

Data accuracy checks are done on external file after importing as a SAS Dataset. For purpose of this paper, test_playlist is the SAS Dataset, against which data checks will be done. Sample code for creating test_playlist SAS Dataset is given in the appendix.

For data check reference table, three buckets have been described for the purpose of this paper. First bucket is used for direct comparison and can check/compare any variable, where a direct match (equal to) is needed. All inaccurate records are captured and printed. Below is the Code, which can be used to check variables that fall into first bucket.

```
title "code for checking first bucket(equal to comparision) to capture inaccurate
records";
proc sql;
select distinct idnum
       ,Gender
       , case when  Gender = column_value
       then column_flag
       else "N"
       end as gender_code_flag
from test_playlist a
 left join referencetable b
 on Gender = column_value
 where calculated Gender_code_flag = "N";
```

Second bucket is to compare greater than or equal to values of source table and code for the same is shown below. All inaccurate records are captured and printed.

```
title " code for checking Second bucket (greater than equal to) to capture inaccurate
records ";

proc sql;
select idnum
       ,salary
       , case when Salary ge column_Value then column_flag
       else "N"
       end as Salary_flag from
   (select idnum, Salary from test_playlist)a
   cross join
   (select input(column_value,8.) as column_value, column_flag from
     referencetable
   where column_name = "Salary")b
   where calculated salary_flag = "N";
```

The third bucket is on matching length and number of characters in numeric variable and code for same is shown below. All inaccurate records are captured and printed.

```
title " code for checking third bucket (pattern match) to capture inaccurate
records";

proc sql;
select idnum
       ,Birth
       ,column_value
       , case when prxmatch(column_value, trim(put(Birth,8.))) > 0
       then column_flag
       else "N"
       end as Birth_flag
       from test_playlist a
       left join
        referencetable b
   on  trim(b.column_name) = "Birth"
   where calculated Birth_flag = 'N';
```

All three buckets can be easily placed into the macro, to analyze multiple variables that fall into each bucket. Below is the partial view of macro, which only shows how to test various variables that fall into bucket 1 and complete macro is available in appendix section.

```
%macro data_check(ref=,test=);
Proc sql noprint;
   select distinct cats('"', column_name, '"') into :ref_col1 separated by ','
    from &ref
    where column_bucket = 1;
quit;
proc sql noprint;
   select distinct name into :curr_col1 separated by ' '
    from dictionary.columns
    where catx('.', libname, memname)  =upcase("&test")
    and name in (&ref_col1) ;
quit;

%do i = 1 %to %sysfunc(countw(&curr_col1,' '));
    %let val%left(&i) = %scan(&curr_col1, &i);
    %let forflag = &&&val&i;
    %let forflag1 = _flag;
    %let final_flag = &forflag&forflag1;
    %put &final_flag;

    title " Inaccurate record captured for variable &&&val&i in bucket 1";
    proc sql;
      select idnum
            ,&&&val&i
           , case when &&&val&i = column_Value
                  then column_flag
              else "N"
           end as &final_flag
       from &test a
       left join   &ref b
       on   &&&val&i = column_value
      where calculated &final_flag = "N";
```

```
 %end;
%mend;
```

Above macro can be executed by below code to find records that fail in data accuracy checks.

```
%data_check(ref=work.referencetable, test=work.test_playlist);
```

This concludes the second and the last topic of this paper “Data check for many tables using a single reference table”.

CONCLUSIONS

Reference tables can be very helpful in finding data inaccuracies before they make their way into target table. Manually checking data inaccuracies is often time consuming and ineffective and can be easily avoided by using reference tables and SAS macro. Reference table approach to handle data accuracy is very flexible compared to others as they can be easily rebuilt or enhanced as per requirement without making any changes to target tables.

ACKNOWLEDGEMENTS

I would like to thank, Vinay Srikanth Marapaka, Vikas Chhabra, Charannag Devarapalli, Srirama Reddy, Ravi Ganesana, Nagaraju Dande, Sarika Elisetti, Anvesh Reddy Perati, for helping me with proof reading and giving their valuable suggestions.

REFERENCES

1. SAS 9.4 Product Documentation. http://support.sas.com/documentation/94/

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the authors for more information:

Kiran Venna
SMACT Works Inc.
Dublin, OH – 4301
kiranvenna@gmail.com

Suryakiran Pothuraju
Unify Solution Inc.
Catonsville, MD - 21228
Suryakiran.pothuraju@analyticscurator.com
www.analyticscurator.com

Appendix

Sample test table to check metadata accuracy (reference table creation is given in main article) can be created as shown below.

```
proc sql;
   create table test_paylist1
       (IdNum char(4),
        Gender char(1),
        Jobcode char(3),
        Salary num,
        Hired num informat=date7.
                  format=date7.);
```

Sample output for metadata check is shown below.

Column
Unmatched columns is/are Birth
Unmatched columns for type or length or position is/are Hired

Sample test table to check data accuracy (reference table creation is given in main article) can be created and populated as shown below.

```
 proc sql;
   create table test_playlist
       (IdNum char(4),
        Gender char(1),
        Jobcode char(3),
        Salary num,
        Birth num,
       Hired num informat=date7.
                  format=date7.);

proc sql;
insert into test_playlist
values('123','M','678',1200,10091998,'09AUG1987'd);
insert into test_playlist
values('111','K','668',1700,10091998,'08AUG1987'd);
insert into test_playlist
values('121','F','150',600,10091998,'08AUG1987'd);
insert into test_playlist
values('141','J','157',900,1001998,'08AUG1987'd);
```

Full Macro code to check data accuracy is shown below.

```
%macro data_check_final (ref=,test=);

/* for Bucket one*/
```

```
Proc sql noprint;
   select distinct cats('"', column_name, '"') into :ref_col1 separated by ','
    from &ref
    where column_bucket = 1;
quit;
proc sql noprint;
   select distinct name into :curr_col1 separated by ' '
    from dictionary.columns
    where catx('.', libname, memname)  =upcase("&test")
    and name in (&ref_col1) ;
quit;

%do i = 1 %to %sysfunc(countw(&curr_col1,' '));
    %let val%left(&i) = %scan(&curr_col1, &i);
    %let forflag = &&&val&i;
    %let forflag1 = _flag;
    %let final_flag = &forflag&forflag1;
    %put &final_flag;

    title " Inaccurate record captured for variable &&&val&i in bucket 1";
    proc sql;
      select idnum
            ,&&&val&i
           , case when &&&val&i = column_Value
                  then column_flag
              else "N"
           end as &final_flag
       from &test a
       left join   &ref b
       on   &&&val&i = column_value
      where calculated &final_flag = "N";
 %end;
 /* bucket 2 greater than equal to 0 with only value*/

Proc sql noprint;
   select distinct cats('"', column_name, '"') into :ref_col2 separated by ','
    from &ref
    where column_bucket = 2;
quit;

proc sql noprint;
   select distinct name into :curr_col2 separated by ' '
    from dictionary.columns
    where catx('.', libname, memname)  =upcase("&test")
    and name in (&ref_col2) ;
quit;

%do i = 1 %to %sysfunc(countw(&curr_col2,' '));
    %let val%left(&i) = %scan(&curr_col2, &i);
    %let forflag = &&&val&i;
    %let forflag2 = _flag;
    %let final_flag = &forflag&forflag2;
    %put &final_flag;

    title " Inaccurate record captured for variable &&&val&i in bucket 2";
```

```
   proc sql;
    select
         idnum
       ,&&&val&i
       , case when &&&val&i ge column_Value then column_flag
       else "N"
       end as &final_flag from
                  (select idnum, &&&val&i from &test)a
                 cross join
               (select input(column_value,8.) as column_value, column_flag from &ref
                 where trim(column_name) = "&&&val&i")b
             where calculated &final_flag = "N";
   quit;
   %end;
/*bucket3 pattern match*/
 title " Inaccurate record captured for variable &&&val&i in bucket 2";

Proc sql noprint;
   select distinct cats('"', column_name, '"') into :ref_col3 separated by ','
    from &ref
    where column_bucket = 3;
quit;
proc sql noprint;
   select distinct name into :curr_col3 separated by ' '
    from dictionary.columns
    where catx('.', libname, memname)  =upcase("&test")
    and name in (&ref_col3) ;
quit;

%do i = 1 %to %sysfunc(countw(&curr_col3,' '));
    %let val%left(&i) = %scan(&curr_col3, &i);
    %let forflag = &&&val&i;
    %let forflag3 = _flag;
    %let final_flag = &forflag&forflag3;
    %put &final_flag;

 proc sql;
select idnum
       ,&&&val&i
       , case when prxmatch(column_value, trim(put(&&&val&i,8.))) > 0
              then column_flag
            else "N"
         end as  &final_flag
   from &test a
       left join
             &ref b
   on  trim(b.column_name) = "&&&val&i"
   where calculated &final_flag = 'N';
quit;
%end;
%mend;
```

Execution code for full macro is shown below.

```
%data_check_final(ref=work.referencetable, test=work.test_playlist);
```