Paper S1-03-2013

Data Cleaning 101: An Analyst's Perspective

Anca Tilea, University of Michigan, Ann Arbor, MI
Deanna Chyn, University of Michigan, Ann Arbor, MI

ABSTRACT

On a daily basis, we are faced with data, both clean and dirty. SAS® offers a multitude of ways to clean and maintain data. It is up to us, the analysts, to choose the best way. Often times the choice we make depends on the analysis needed further down the road. When you start as a novice analyst, you are familiar with the basics of data steps and procedures: SET statement, PROC MEANS, PROC FREQ, PROC SORT. We hope this paper will help a SAS® user who has this basic knowledge feel more comfortable moving beyond the basics to code more efficiently. We will introduce you to SAS® procedures (PROC TRANSPOSE, PROC SQL) and SAS® tricks (using CALL SYMPUT, DO-LOOP, and PRX functions) that are beyond a simple data step, but not too complicated to be understood by someone with basic SAS® skills. This paper is intended for the novice SAS® user, with basic to intermediate skills and SAS® 9.1 or above.

INTRODUCTION

Congratulations! You just started your job as a data analyst. You open your data set and wait a minute! You have 1 million more rows than expected! This is not what was it like in SAS® class. This is not what *SASHELP.CLASS* looks like. These are REAL DATA. The data are bigger, more complex and often not what you expect (a patient that is 723 years old); add on top of that a particular boss with particular needs. So, how do you check your data? How do you clean your data? How do you re-structure your data? Here, let us help.

GETTING STARTED

1. **CALL SYMPUT**
   A typical thing you will be asked to look at is the distribution (average ± standard deviation) of given variables. There are many ways one can accomplish this (PROC SUMMARY, PROC RANK, PROC MEANS, PROC UNIVARIATE…), all offering the options to output the results in a SAS® data set (using ODS statements or an output statement) or in the listing file. *CALL SYMPUT* provides the ability to save values into macro variables that later can be *recalled.*

CALL SYMPUT	Copying/pasting from the .LST file to the editor

`PROC MEANS DATA = SASHELP.CLASS;`
`  VAR AGE;`
`OUTPUT OUT = MY_MEANS;`
`RUN;`
`DATA _NULL_;`
`  SET MY_MEANS(WHERE = (_STAT_ = "MEAN"));`
`  CALL SYMPUT ("AGE_AVG",INPUT(AGE, 8.0));`
`RUN;`
`DATA WANT;`
`  SET SASHELP.CLASS;`
`  AGE_LT_MEAN = (. < AGE < &AGE_AVG.);`
`RUN;`
	`PROC MEANS DATA = SASHELP.CLASS;`
	`  VAR AGE;`
	`OUTPUT OUT = MY_MEANS;`
	`RUN;`
	
	`DATA WANT;`
	`  SET SASHELP.CLASS;`
	`  AGE_LT_MEAN = (. < AGE < 13.31578);`
	`RUN;`

2. **PROC SQL SELECT INTO:**

Sometimes you may need to subset your data set. Or you may need to rename specific variables in a data set. In the exploratory phase it is often the case that you run PROC MEANS, PROC UNIVARIATE, PROC SUMMARY and PROC FREQs on your data. In any of these scenarios, typically, you would manually list all the variables of interest. A quicker way would be to select the variables into a macro variable, and reference it later. Using this method also reduces the risk of manual error when listing the variable names.

PROC SQL SELECT INTO:	**Manually write the variables**

`PROC SQL;`
`  SELECT CATS("'",NAME, "'") INTO:`
`NAMES_A SEPARATED BY " , "`
`  FROM SASHELP.CLASS`
`  WHERE NAME LIKE "A%";`
`QUIT;`
`%PUT &NAMES_A.;`
 	
`DATA NAMES_A;`
`  SET SASHELP.CLASS`
`    (WHERE = (NAME IN (&NAMES_A.)));`
`RUN;`
	`PROC FREQ DATA = SASHELP.CLASS;`
	`  TABLES NAME;`
	`RUN;`
	
	`DATA NAMES_A;`
	`  SET SASHELP.CLASS(WHERE = (NAME IN`
	`('Alfred','Alice')));`
	`RUN;`

3. **PROC TRANSPOSE**

Sometimes you need to perform calculations on the rows of a data set. It is not complicated to do so (using PROC SQL, for example), but it may be easier to re-arrange the data set from *long* to *wide* and perform calculations on the columns. One way to re-arrange the data is using simple *array* statement(s) in a data step. An alternative solution is to use PROC TRANSPOSE.

PROC TRANSPOSE	**ARRAY STATEMENT**

`PROC SORT DATA=SASHELP.CLASS`
`OUT=CLASS_SORTED;`
`  BY SEX;`
`RUN;`
 	
`PROC TRANSPOSE DATA = CLASS_SORTED`
`  OUT = CLASS_TRANSPOSE(DROP = _NAME_)`
`  PREFIX = AGE_;`
`  BY SEX;`
`  VAR AGE;`
`RUN;`
	`PROC SORT DATA=SASHELP.CLASS`
	`OUT=CLASS_LONG;`
	`  BY SEX;`
	`RUN;`
	
	`DATA CLASS_WIDE(KEEP = SEX AGE_:);`
	`ARRAY AGE_ARRAY{10} AGE_1-AGE_10;`
	`  DO I = 1 BY 1 UNTIL (LAST.SEX);`
	`    SET CLASS_LONG;`
	`    BY SEX;`
	`      AGE_ARRAY{I} = AGE;`
	`  END;`
	`RUN;`

4. **%MACRO %DO 1 %TO N**

Say you need to read-in yearly data that are stored in individual EXCEL files (e.g., year1, year2, year3…yearN). The most straightforward way is to use PROC IMPORT for each file. Another way would be to create a macro and invoke it *n* number of times. And yet another way would be to use a macro array.

Macro Arrays	PROC IMPORT

%MACRO QUICK();
  %DO I = 1 %TO 3;
  PROC IMPORT DATAFILE =
  "&PATH.\SASHELP_CLASS&I..XLSX"
  OUT=CLASS&I. DBMS=EXCEL REPLACE;
  RUN;
  %END;
%MEND;
%QUICK();
	PROC IMPORT DATAFILE =
	  "&PATH.\SASHELP_CLASS1.XLSX"
	  OUT=CLASS1 DBMS=EXCEL REPLACE;
	RUN;
	PROC IMPORT DATAFILE =
	  "&PATH.\SASHELP_CLASS2.XLSX"
	  OUT=CLASS2 DBMS=EXCEL REPLACE;
	RUN;
	PROC IMPORT DATAFILE =
	  "&PATH.\SASHELP_CLASS3.XLSX"
	  OUT=CLASS3 DBMS=EXCEL REPLACE;
	RUN;
	
	*----OR-----;
	%MACRO IMPORT(YEAR =);
	  PROC IMPORT DATAFILE =
	  "&PATH.\SASHELP_CLASS&YEAR..XLSX"
	  OUT=CLASS&YEAR. DBMS=EXCEL REPLACE;
	  RUN;
	%MEND IMPORT;
	%IMPORT(YEAR = 1);
	%IMPORT(YEAR = 2);
	%IMPORT(YEAR = 3);

5. PROC SQL JOIN: DATE RANGES

Often times you may have two data sets, with different data elements that you want to merge together based on the date. But the dates are not always the same. One way to tackle this issue is by creating several data steps, merging and sorting. Another way is to use PROC SQL, as shown below.

PROC SQL	(many) Data Steps

PROC SQL;
  CREATE TABLE BUYS_STOCKS AS
  SELECT A.DATE AS A_DATE, A.AMOUNT, B.DATE
  AS B_DATE, B.STOCK, ABS(A.DATE-
  B.DATE) AS DIFF_DATE
  FROM SASHELP.BUY AS A JOIN
  SASHELP.STOCKS (WHERE =
  (YEAR(DATE) > 2000)) AS B
  ON YEAR(A.DATE) = YEAR(B.DATE)
  HAVING DIFF_DATE LE MIN(DIFF_DATE);
QUIT;
	DATA STEP;
	
	PROC SORT;
	DATA STEP2;
	  MERGE;
	RUN;
	PROC SORT;
	DATA STEP;
	  IF FIRST.VAR;
	RUN;

6. SAS® CHARACTER FUNCTIONS

Sometimes you want to subset an existing SAS® data set based on some specific rule. Let's say you need to only keep records for students whose first name starts with either the letter P or J. An obvious way to accomplish this is to manually code all values that meet this criterion. Luckily, there are several alternatives, three of which are shown below.

Using SAS® character functions	Manually list values

*PRXMATCH;
DATA CLASS_pj;
  SET SASHELP.CLASS
	PROC FREQ DATA = SASHELP.CLASS;
	  TABLES NAME;
	RUN;

```
 (WHERE=(PRXMATCH("M/^[PJ]/i", NAME) > 0));
  /*prxmatch matches (M) any value that
  starts (^)with either P or J [PJ], case
  insensitive (i),in the variable NAME.*/
RUN;

*----OR----;
*SQL %LIKE;
PROC SQL;
  CREATE TABLE CLASS_pj AS
  SELECT *
  FROM SASHELP.CLASS
  WHERE UPCASE(NAME) LIKE 'P%' OR
UPCASE(NAME) LIKE 'J%';
QUIT;

*----OR----;
*SUBSTR;
DATA CLASS_PJ;
  SET SASHELP.CLASS
  (WHERE = (UPCASE(SUBSTR(NAME,1,1))="P"
          OR
          UPCASE(SUBSTR(NAME,1,1))="J"));
RUN;
```

```
DATA CLASS_pj;
  SET SASHELP.CLASS
  (WHERE = (NAME = “Philip” OR
             NAME =  “James” OR
             NAME = “Jane” OR
             NAME = “Janet” OR
             NAME = “Jeffrey” OR
             NAME = “John” OR
             NAME = “Joyce” OR
             NAME = “Judy”
   );
RUN;
```

7. IF-ELSE STATEMENT(s) - with the largest group in the first if-statement

As an analyst, it is important to strive for efficiency not only in the length of the code, but also in the way the code processes the data. You don’t want just shorter code, but smarter code. One simple example is when creating a group variable.

```
IF INCOME = “<20K” THEN GRP_INCOME = 1;
ELSE IF INCOME = “20-45K” THEN GRP_INCOME = 2;
ELSE IF INCOME = “45-75K” THEN GRP_INCOME = 3;
 ELSE IF INCOME = “>75K” THEN GRP_INCOME = 4;
      ELSE GRP_INCOME = .;
```

Depending on the distribution of your INCOME variable, SAS® may be passing through data unnecessarily. If the distribution of your INCOME variable is:

income	Frequency	Cumulative Percent	Cumulative Frequency	Percent
<20K	4	20.00	20	100.00
20-45K	4	20.00	4	20.0
45-75K	12	60.00	16	80.00

SAS® will process 60% of the data (the “45-75K” group) two times before actually using it. In the first “IF” statement, 100% of the data is being processed. In the second “IF” statement, 80% of the data is being processed (20% of “<20K” has been identified). And, lastly, SAS® processes the remaining 60% of the data corresponding to “45-75K” records.

In order to avoid this unnecessary process time, it may be useful to run a PROC FREQ beforehand and start the IF-ELSE statements with the group that has the largest frequency (i.e., the “45-75K” in this example).

CONCLUSION

It is often the case that much of the analyst's time is spent on cleaning data. Certain tasks could be accomplished more efficiently using simple PROC SQL and SAS® character functions. We present a limited number of useful commands that will speed up the process of cleaning data.

CONTACT INFORMATION

Anca M Tilea
Senior Research Analyst
University Of Michigan
1415 Washington Height 3645A
Ann Arbor MI, 48104
atilea@med.umich.edu