Paper RF-08-2014

Burst Reporting With the Help of PROC SQL

Dan Sturgeon, Priority Health, Grand Rapids, Michigan
Erica Goodrich, Priority Health, Grand Rapids, Michigan

ABSTRACT

Many SAS® programmers need a solution for creating a large amount of reports for multiple customers in an automated fashion using one piece of code. While there are many different ways to accomplish this, I will be talking about one method using PROC SQL and the SAS® macro language. This method creates a custom SAS® macro that needs two data sets, one for metadata and one with the actual data. The macro loops through the procedures written within and plugs in the information that is needed, creating many customized reports in a short amount of time.

INTRODUCTION

For those of us who create reports for multiple customers the idea of a quick, automated solution is an important one. In my job there are times where I need to output multiple pages of reports for hundreds of different groups (Which can consist of provider groups, networks of care, hospitals, members or other groups). Doing this one at a time would not be feasible and standard output options like class or by statements do not provide flexibility. The solution is a macro that can loop through each group and create a report for those groups individually.

While there are many different ways to accomplish this in SAS®, I will be looking at one way I came up with that takes advantage of PROC SQL. This method was then applied to a macro that can be adjusted based on the situation you are in.

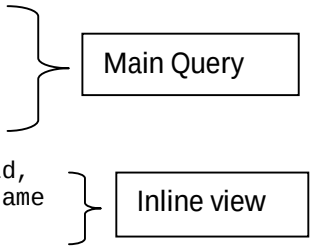
MACRO

The first step for this method is to define the inputs that are needed for this macro. A dataset with a column that identifies each group in question and a name to go with each group is necessary. These are identified in the code as 'unique_id' and 'group_name' while the dataset itself is identified as 'data.' From unique_id and group_name the macro then outputs two variables during the looping process discussed later, those variables are 'ID' and 'Name' and these variables need to be leveraged within the code that the user has written. This code needs to be wrapped within a new macro so that it can be used within this macro. It is identified as "code."

Input Variable	Use	Output from column
Unique_id	Column that has unique identifier for group	ID
Group_name	Column that has assigned name	Name
Data	Name of the dataset being used	N/A
Code	Macro where the code is stored	N/A

The macro itself is split up into two parts, the initial table creation and the loop. The initial table creation consists of a PROC SQL statement with two parts in it. The first takes the dataset used in the data input and creates a metadata table of only the unique ID's and names plus we will add a row number for our counter that comes up later. For the traditional data step programmers this is done by using something like N = _N_, but this example leverages PROC SQL so the monotonic function is used instead. The following code is used:

```
PROC SQL NOPRINT;
  CREATE TABLE meta AS
  SELECT distinct unique_id,
                  group_name,
                  monotonic() AS rank
    FROM
        (SELECT distinct &unique_id AS unique_id,
                         &group_name AS group_name
           FROM &data);
```



It should be noted that the code above is creating a table from an inline view. The inline view (the lower select clause in parenthesis) pulls the unique ID's and names specified from the dataset specified and names them 'unique_id' and 'group_name.' Then the main query queries that and creates a table from it and adds rank numbers using the monotonic function. The new table 'meta' is now a normal SAS® dataset in the work folder.

From here, but still within the same PROC SQL statement we have the second piece of code:

```
SELECT count(*)
  INTO :group_cnt  /*Works similarly to CALL SYMPUT in the data step*/
  FROM meta;
QUIT;
```

Some more notes: The quit at the end goes with the initial PROC SQL statement. This statement is just pulling the number of lines are in the dataset meta and putting them into a macro variable (group_cnt) using the INTO statement. Now we have a variable that represents how many groups we have to create reports for.

For the loop section one line from the meta data is pulled at a time. Each line has those pieces put into macro variables.

The code for this method looks like this:

```
%DO I = 1 %TO &group_cnt;
  PROC SQL NOPRINT;
  SELECT unique_id,
         group_name
    INTO :id,  /*Variable for unique ID*/
         :name /*Variable for grouping name. This does not need to be unique*/
    FROM meta
   WHERE rank = &I;
  QUIT;
```

This first part counts from 1 to the number of groups in the set. Each loop grabs the unique ID and the name of each group and puts them into the macro variables id and name. These will be used in any where statements used in the code that you use and in any title statements. After pulling these variables the loop continues:

```
     %LET Name = %SYSFUNC(strip(&name));
     %LET id   = %SYSFUNC(strip(&id));

&Code

%END;
```

The two LET statements are removing leading and trailing spaces in case we want to use the variable in a title. The &Code is where the code will be placed within the loop. So to review the loop will pull a unique ID and group name for each group, one at a time and apply them to whatever method you choose to use below.

EXAMPLE

For our example I will use de-identified data from our provider incentive program to show exactly how this works. The data contains five randomly chosen groups, labeled as GROUP1 through GROUP5. The set contains 219 rows,

covering multiple measures and products (HMO, ASO, Medicaid, and Medicare). The unique identifier is 'PIP_LEV_ID' and the group name is 'GROUP.'

	PIP_LEV_ID	PIP_MEAS_DESC	GROUP	SCORE	PRODUCT
1	129	Mammography	GROUP4	55%	HMO
2	112	Mammography	GROUP2	71%	HMO
3	102	Mammography	GROUP3	78%	HMO
4	10	Mammography	GROUP5	68%	HMO
5	1	Mammography	GROUP1	80%	HMO
6	129	Mammography	GROUP4	67%	Medicare
7	102	Mammography	GROUP3	81%	Medicare
8	10	Mammography	GROUP5	72%	Medicare
9	1	Mammography	GROUP1	80%	Medicare
10	102	Mammography	GROUP3	72%	Medicaid
11	10	Mammography	GROUP5	58%	Medicaid
12	129	Adolescent Immunizat...	GROUP4	50%	HMO
13	102	Adolescent Immunizat...	GROUP3	88%	HMO
14	10	Adolescent Immunizat...	GROUP5	67%	HMO
15	1	Adolescent Immunizat...	GROUP1	80%	HMO
16	102	Adolescent Immunizat...	GROUP3	87%	Medicaid
17	10	Adolescent Immunizat...	GROUP5	80%	Medicaid
18	129	Cervical Cancer Scre...	GROUP4	71%	HMO
19	112	Cervical Cancer Scre...	GROUP2	70%	HMO
20	102	Cervical Cancer Scre...	GROUP3	82%	HMO
21	10	Cervical Cancer Scre...	GROUP5	70%	HMO
22	1	Cervical Cancer Scre...	GROUP1	80%	HMO
23	129	Diabetes Care: Contr...	GROUP4	15%	HMO
24	102	Diabetes Care: Contr...	GROUP3	37%	HMO
25	10	Diabetes Care: Contr...	GROUP5	37%	HMO
26	1	Diabetes Care: Contr...	GROUP1	54%	HMO
27	129	Diabetes Care: Contr...	GROUP4	29%	PPO
28	112	Diabetes Care: Contr...	GROUP2	0%	PPO
29	102	Diabetes Care: Contr...	GROUP3	41%	PPO
30	10	Diabetes Care: Contr...	GROUP5	42%	PPO
31	1	Diabetes Care: Contr...	GROUP1	50%	PPO
32	102	Diabetes Care: Contr...	GROUP3	47%	Medicaid
33	10	Diabetes Care: Contr...	GROUP5	33%	Medicaid
34	129	Diabetes Care: Contr...	GROUP4	17%	HMO

Figure 1. Partial Dataset

PROC MEANS is used in the example below:

```
%MACRO lean_mean();
ODS PDF file="&personal\MWSUG\&Name._Means.pdf" style=journal2;
TITLE1 "Testing Variables: ID = &ID. Group = &Name.";
PROC MEANS DATA=personal.dummy;
  WHERE pip_lev_id = &ID;
  VAR score;
  CLASS pip_meas_desc;
RUN;
ODS PDF CLOSE;

%MEND;
```

The macro above outputs the contents of a PROC MEANS into PDF files into my MWSUG folder .The macro needs the output information from the main macro (ID and Name) to work. The “Name” macro variable in the filename is used in this example. If the “Name” macro is not unique then an additional unique ID is necessary (e.g. the counter used) to set it apart from the others. The Name macro variable is also used in the title. The ID macro variable is used in both the title and, more importantly, in the where clause where it is used to differentiate the data.

The MACRO statement needs to be in this form:

```
%burst(unique_id,group,dataset,code);
```

Which in this example becomes:

```
%burst(pip_lev_id,group,personal.dummy,%lean_mean());
```

The results are as follows:

Here is what the meta dataset looks like this:

	unique_id	group_name	rank
1	1	GROUP1	1
2	10	GROUP5	2
3	102	GROUP3	3
4	112	GROUP2	4
5	129	GROUP4	5

Figure 2. 'Meta' Dataset

This information is then fed into the code provided by the user. An example of how it looks the first time through is as follows:

```
ODS PDF file="&personal\MWSUG\GROUP1_Means.pdf" style=journal2;
TITLE1 "Testing Variables: ID = 1 Group = GROUP1";
PROC MEANS DATA=personal.dummy;
  WHERE pip_lev_id = 1;
  VAR score;
  CLASS pip_meas_desc;
RUN;
ODS PDF CLOSE;
```

Which produces the below files:

Figure 3. View of Created Files

And creates and output that looks like this:

Testing Variables: ID = 1 Group = GROUP1

The MEANS Procedure

Analysis Variable : SCORE

PIP_MEAS_DESC	*N Obs*	*N*	*Mean*	*Std Dev*	*Minimum*	*Maximum*
Adolescent Immunizations	1	1	0.8048780	.	0.8048780	0.8048780
Cervical Cancer Screenings	1	1	0.8015267	.	0.8015267	0.8015267
Childhood Immunizations: Combination 3	1	1	0.8076923	.	0.8076923	0.8076923
Chlamydia Screenings	3	3	0.5246212	0.1016924	0.4375000	0.6363636
Chronic Kidney Disease	2	2	0.8055095	0.0217591	0.7901235	0.8208955
Diabetes Care: Annual Retinal Eye Exam	3	3	0.7080116	0.1190428	0.5714286	0.7897196
Diabetes Care: Controlled Blood Pressure (<140/80)	3	3	0.5069673	0.0061853	0.5000000	0.5118110
Diabetes Care: Controlled HbA1c Less than 7.0%	2	2	0.5197368	0.0279121	0.5000000	0.5394737

Figure 4. Output Created

CONCLUSION

The macro described within this paper is a useful tool for those of us who need to create many reports for many different groups. While there are many different ways to do this, this method is the preferred method for us in our day to day work activities because of the simplicity of the code and because it is better suited for the data we deal with on a regular basis. The full macro for burst reporting is seen in its entirety in the appendix at the end of this paper.

Contact Information

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

Name: Erica Goodrich
Enterprise: Priority Health
Address: 1231 East Beltline Ave NE
City, State ZIP: Grand Rapids, MI 49525
Work Phone: 616-464-8142
Fax: 616-942-0024
E-mail: erica.goodrich@priorityhealth.com

Name: Daniel Sturgeon
Enterprise: Priority Health
Address: 1231 East Beltline Ave NE
City, State ZIP: Grand Rapids, MI 49525
Work Phone: 616-575-5740
Fax: 616-942-0024
E-mail: daniel.sturgeon@priorityhealth.com

Appendix: Code

```
%MACRO BURST(unique_id, group_name,data,code);
   PROC SQL NOPRINT;
/*Creating a dataset with the unique ID's and Names, plus a row
counter*/           CREATE TABLE meta AS
SELECT distinct unique_id,
                group_name,
                monotonic() AS rank /*similar to using n = _n_ in the datastep*/
 FROM (SELECT distinct &unique_id AS unique_id,
                       &group_name AS group_name
         FROM &data);

/*Creates a macro variable for how many groups are in the meta set*/

 SELECT count(*)
   INTO :group_cnt  /*Works similarly to CALL SYMPUT in the data step*/
   FROM meta;
QUIT;

/*Here is the start of the loop that grabs the information based on the row number*/
%DO I = 1 %TO &group_cnt;
   PROC SQL NOPRINT;
   SELECT unique_id,
          group_name
     INTO :id,    /*Variable for unique ID*/
          :name   /*Variable for grouping name. This does not need to be unique*/
     FROM meta
    WHERE rank = &I;
   QUIT;

/*The Below LET statements remove leading and trailing blanks*/

     %LET Name = %SYSFUNC(strip(&name));

     %LET id   = %SYSFUNC(strip(&id));

/*Here is where the MACRO is put*/

&Code

     %END;

%MEND;
```