Paper SAS2022-2018

Implementing Privacy Protection-Compliant SAS® Aggregate Reports

Leonid Batkhan, PhD, SAS Institute Inc.

ABSTRACT

Personal data protection in published reports is an important human right, and is required by US Federal Law, European Union General Data Protection Regulation, and many other jurisdictions. This paper highlights the importance of producing privacy protection-compliant reports, discusses aspects of potential privacy breaches, and suggests a robust algorithm for producing well-protected aggregate reports. This paper walks through the complex logic of an enhanced complementary suppression process, and demonstrates SAS® coding techniques to implement and automatically generate aggregate tabular reports compliant with privacy protection law. The result is a set of SAS macros ready for use in any reporting organization responsible for compliance with privacy protection.

INTRODUCTION

Personally Identifiable Information (PII) is any data that could potentially identify specific individuals and their sensitive or confidential information. Such sensitive data can include health insurance information, medical and epidemiological records, student education records, demographic information such as age, ethnicity, race, gender, education, political or religious beliefs, financial and credit information, geographical location, criminal history, and so on.

According to the law, Personally Identifiable Information must not be disclosed in any published reports. However, it is easy to overlook an unintended disclosure of PII, especially when reports are generated automatically or semi-automatically via dynamic data queries. Even for aggregate reports there is a high potential for such a disclosure.

Suppose that we produce an aggregate crosstabular report on a small demographic group, and we count distribution by students' grade and gender. It is highly probable that we can get the count to equal 1 for some cells in the report, which will unequivocally identify a person and thus disclose their education record (grade). Even if the count is not equal to 1, but is equal to some other small number, there is still a risk of possible deducing or disaggregating PII from related reports on that small demographic group.

Therefore, in order to protect PII from being figured out, an aggregate public demographic report must have small count numbers somehow concealed, obscured, masked, or suppressed. But how much of the data suppression is good enough in order to protect privacy and at the same time not have it overdone and make the report useless?

GROUPING AS DATA MASKING

One way of obscuring small count numbers to protect people's privacy is to combine them into a larger group and call it OTHER.

For example, when producing pie charts with PROC GCHART, the `other=` option can be used to specify a cutoff number. If a category count (or percent) is less or equal to that of a cutoff number, then that count is put in the OTHER category. However, this might not protect small numbers if only a count of 1 falls into the OTHER category. Also, this approach can be problematic as it does not show all the values of the categorical variable and might encourage inaccurate interpretation.

In addition, while protecting PII, this method distorts the composition of the report group as it can put different demographic characteristics into OTHER category for different report groups, thus making it impossible to compare them side by side.

PRIMARY DATA SUPPRESSION

Primary data suppression is based on the values of the cells in a tabular aggregate report. The criteria for the primary data suppression can be expressed as $c_{ij} \leq s_{max}$ where s_{max} is the suppression cutoff number. All cell values c_{ij} less or equal than s_{max} must be suppressed.

When a row or a column contains just a single suppressed small number *and* in addition to frequencies (or counts), the report also displays totals by rows and/or columns, primary data suppression that is based on the number value itself is not enough for privacy protection as the single suppressed number can be calculated as total minus the sum of the other reported (not suppressed) numbers in that row or column.

The following figures illustrate how easy it is to reveal data that is subjected to the primary suppression *only* (this example uses s_{max} = 5). Let's compare report outputs before and after the primary suppression.

People by Race and Age Group

	People Reported				
	Age Group				
Race	<18	19-64	65-99	100+	Total
Martian	2	12	7	0	**21**
Asian	14	11	0	10	**35**
Black	17	8	5	16	**46**
Hispanic	9	4	24	19	**56**
White	18	13	19	20	**70**
Total	**60**	**48**	**55**	**65**	**228**

Figure 1. Cross-Frequency Report before Suppression

People by Race and Age Group

	People Reported				
	Age Group				
Race	<18	19-64	65-99	100+	Total
Martian	*	12	7	0	**21**
Asian	14	11	0	10	**35**
Black	17	8	*	16	**46**
Hispanic	9	*	24	19	**56**
White	18	13	19	20	**70**
Total	**60**	**48**	**55**	**65**	**228**

Figure 2. Cross-Frequency Report after Primary Suppression

As you can see, it is very easy to unveil the suppressed cells by simply subtracting from the row or column totals all the displayed counts in that row or column.

Different jurisdictions have various guidelines for assigning the value for s_{max} depending on the sensitivity of the reported data. These guidelines range from s_{max} = 2 as the absolute minimum (suppressing only 1s), to s_{max} = 5 for more sensitive PII, and to as high as 15.

However, no matter how large a number is assigned for s_{max}, there is no guarantee that there won't be a single suppressed cell in a row or column that would allow for an easy unraveling of all the suppressed cells.

COMPLEMENTARY DATA SUPPRESSION

To exclude the possibility of deducing suppressed data, primary data suppression needs to be complemented by *secondary data suppression,* when a second small number in a row or column is also suppressed. What further complicates the matter is that as long as we suppress that second number in a *row***,** that suppressed cell might become the only suppressed cell across the *column*. In such a case, data can be derived from the column total and other numbers in that column. To make sure that we truly protect privacy and eliminate the possibility of unraveling suppressed data like we solve a Sudoku puzzle, we need to complement secondary suppression with *tertiary data suppression* of that cell in that column. We refer to those secondary, tertiary, and so on, data suppressions as *complementary data suppressions*.

Depending on the cardinality of classification variables and data combinations, the number of those complementary suppression iterations can vary.

Also, it is worthwhile noting that the complementary suppression might produce different solutions (end results) depending on which classification variable you start with, that is whether you start with rows or columns. If you do it both ways and the results are different, you have a choice of selecting a better solution. However, the criteria of "a better solution" is somewhat nontrivial. It can be the solution that has the least number of suppressed cells; the solution that produces the least sum of all suppressed counts, as it has less "wasteful" count suppression, which makes the report more informative; or it could be something else.

ZEROS: TO SUPPRESS OR NOT TO SUPPRESS?

An interesting question is what to do with cell counts that are equal to zero. While zero technically is considered a "small number" (it's the smallest of all counts), it's a unique number in that it reveals *nobody's* personal information.

Although there might be an interpretation that zeros are not suitable for primary suppression, but are fair game for complementary suppression, I don't think it makes a lot of sense as that would make suppression inconsistent and dependent on the data composition. Besides, suppressing zeros and then counting them as complementary might actually aid in deducing values of other cells in a row or column that it is supposed to protect. Therefore, we will treat zeros as not to be suppressed in either primary or complementary suppression. The reason is that zero counts represent very useful and unique information about the data distribution without compromising anybody's confidentiality, while other "small numbers" present the risk of revealing PII.

With zeros "disqualified" from being suppressed, the primary data suppression criteria can be rewritten as: all cell values c_{ij} between 1 and s_{max}, inclusive, must be suppressed.

ENHANCED COMPLEMENTARY SUPPRESSION

However, as if the iterative complementary suppression is not complex enough, Kristian Lønø (Statistics Norway) brought to my attention a special case that renders that seemingly invincible PII protection algorithm vulnerable.

Imagine a situation where all (two or more) suppressed cells in a row/column of a report have the original (before suppression) counts of 1. Then, even with the complementary suppression applied over all rows and columns, if Total minus all unsuppressed counts in a row/column equals the number of suppressed cells in that row/column, it is obvious to conclude that all the suppressed counts in that row/column are equal to 1.

As long as we decipher that, it becomes easy to unscramble the rest of the suppressed counts in a tabular report.

This is illustrated in the following example. Let's say, our report output before suppression looks like the report shown in Figure 3.

People by Race and Age Group

	People Reported				
	Age Group				
Race	<18	19-64	65-99	100+	Total
Martian	0	20	13	7	**40**
Asian	9	21	8	1	**39**
Black	4	22	10	1	**37**
Hispanic	3	23	15	1	**42**
White	2	24	12	1	**39**
Total	**18**	**110**	**58**	**11**	**197**

Figure 3. Cross-Frequency Report before Suppression

After we apply iterative complementary suppression, we get the following tabular aggregate report in which every row and column has at least two cells suppressed (complementary suppression criteria has been satisfied).

People by Race and Age Group

	People Reported				
	Age Group				
Race	<18	19-64	65-99	100+	Total
Martian	0	20	13	7	**40**
Asian	9	21	*	*	**39**
Black	*	22	*	*	**37**
Hispanic	*	23	15	*	**42**
White	*	24	12	*	**39**
Total	**18**	**110**	**58**	**11**	**197**

Figure 4. Cross-frequency Report after Complementary Suppression

As you can see, the column Age Group 100+ has 4 cells suppressed. However, look at the displayed numbers in that column. If you subtract the remaining unsuppressed cell with the count of 7 from the column total of 11, you get 11-7=4. This means that the sum of 4 suppressed cells is 4, which unequivocally reveals that each suppressed number in this column is equal to 1 (don't forget, we are dealing with counts that are natural numbers, zero excluded). After we figure that out, it becomes quite easy to crack the rest of the suppressed counts by subtracting the sum of the unsuppressed numbers in each row from the row totals.

Hence, we need to augment the complementary suppression criteria with one more constraint: the sum of all unsuppressed cells in each row/column subtracted from the row/column total should not equal the number of the suppressed cells. If it is equal, then one more suppression iteration is warranted for that row/column to conceal the fact that they are all equal to 1.

To simplify implementation of this requirement, we can approach this problem from a different, "insider" perspective since we have access to the pre-suppression data. Instead of relying on surrounding data to determine that all suppressed cells in a given row/column are equal to 1 (as the row/column total minus the sum of the unsuppressed cells equals the number of the suppressed cells), we can derive that from the sum and the number of the suppressed cells *themselves*, by keeping track of them. Here is my reasoning. If suppressed cell counts in a given row/column are all equal to 1, it means that their sum divided by their number is equal to 1. The converse is true as well for the counts. If the sum of the positive integer numbers in a set divided by their cardinality is equal to 1, then all the numbers in the set are equal to 1. But isn't the sum of several quantities divided by the number of these quantities called an *average*? Of course it is. Then we can say that the average of the suppressed cells in a row/column must be greater than 1 in order not to breach privacy protection. Let's call it the *enhanced complementary suppression* criteria.

The enhanced complementary suppression criteria then can be written as follows. For each row and each column of a tabular aggregate report, the averages of the suppressed cells should be greater than 1, otherwise one more cell in that row/column must be suppressed. Here are the conditions that require another suppression iteration.

$$a_i = \frac{1}{n}\sum_{j=1}^{n} s_{ij} \leq 1$$

or

$$a_j = \frac{1}{m}\sum_{i=1}^{m} s_{ij} \leq 1$$

In these formulas, *i* is a row number, *j* is a column number, s_{ij} is as a suppressed cell original (hidden) value, a_i and a_j are the averages of the suppressed counts in each row and each column accordingly, and *n* and *m* are the numbers of the suppressed counts in each row and each column.

An additional benefit of using this criteria that is expressed through averages is that it can be easily strengthened by requiring further suppression iterations when this average is not greater than any real number ≥1. We can modify this criteria to be $a_i \leq a_{max}$, $a_j \leq a_{max}$ where a_{max} is any real number ≥1. For example, a_{max} = 1.5 indicates that if an average of the suppressed cells in a row/column is less or equal to 1.5, then we need to suppress one more cell in that row/column. This is not an unreasonable requirement. If a row has two (and complementary) suppressed cell values 1 and 2, and we are not satisfied with these two values being the only ones suppressed, then a_{max} = 1.5 will do just that. It initiates another suppression iteration to suppress one more cell in that row because the average of 1 and 2 is equal (1+2)/2 = 1.5.

In other words, depending on the sensitivity of reported data, the assigned value for a_{max} could range from a_{max} = 1 as the absolute minimum identifying "all 1s" for additional complementary suppression, to some larger real number (for example, 1.3, 1.5, 2, 2.5, and so on.) to flag a cluster of small counts in a row/column *as a group* for additional complementary suppression.

TOTALS: TO SUPPRESS OR NOT TO SUPPRESS

While totals by rows and columns are inherently different from other report cells, they are just as capable of revealing PII. Therefore, counts representing totals for rows and columns of a tabular report must be suppressed by the same rules as the other report cells. That is, the row and column totals should be suppressed based on all three suppression criteria: primary suppression, complementary suppression, and enhanced complementary suppression.

SUPPRESSION ALGORITHM

Here is the summary algorithm combining the above primary, complementary, and enhanced complementary suppression criteria for implementing small numbers of cell suppression using SAS for a two-dimensional aggregate tabular report:

1. Use PROC MEANS to calculate count sums and output results in a data table. There will be two class variables: *class-var-1* and *class-var-2*. Class variable values for the totals will have blank values.
2. Sort the data table from step 1 *by class-var-1, count*. That will order counts from smallest to largest within each value of the class-var-1 (including blanks for totals).
3. Read through the sorted data table from step 2 by *class-var-1* group and do the following:
    - Replace all count values within [1, s_{max}] interval with a special numeric missing value, **.S**. This is the *primary suppression*. All primary suppression "suspects" are "eliminated" during this first iteration.
    - If there is already a single suppressed value in the group, replace the current count value that is the next largest to the first suppressed value with **.S**. This is the *complementary suppression*.
    - Calculate the average a_i of the original values for the previously suppressed cells. If $a_i \leq a_{max}$, replace the current count value with **.S**. This is *enhanced complementary suppression*.
    - If there are no cells that are eligible for primary suppression, the whole suppression process stops after this step without suppressing any data.
4. Repeat steps 2 and 3 with *class-var-2* variable instead of *class-var-1*. This is an iteration of the complementary suppression.
5. Keep repeating steps 2 and 3 while swapping *class-var-1* and *class-var-2* variables until step 3 produces no suppressed values.
6. Use PROC REPORT to write the results with **.S** values formatted to *****.

SAS CODE TO IMPLEMENT SUPPRESSION

Let's suppose that our original, unsuppressed tabular aggregate report that was produced by PROC REPORT directly out of our data, looks like the one shown in Figure 5.

People by Race and Age Group

	People Reported				
	Age Group				
Race	<18	19-64	65-99	100+	Total
Martian	15	12	7	0	**34**
Asian	14	11	0	10	**35**
Black	17	8	6	16	**47**
Hispanic	9	4	24	19	**56**
White	18	13	19	20	**70**
Total	**73**	**48**	**56**	**65**	**242**

Figure 5. Original Report without Suppression

The full sample of the SAS code implementing primary, complementary, and enhanced complementary suppression algorithms described above is presented in the **Appendix** section at the end of this paper. This code produces the following final report as shown in Figure 6.

People by Race and Age Group

	People Reported				
	Age Group				
Race	<18	19-64	65-99	100+	Total
Martian	15	*	*	0	**34**
Asian	*	11	0	*	**35**
Black	17	*	*	*	**47**
Hispanic	*	*	24	19	**56**
White	18	13	19	20	**70**
Total	**73**	**48**	**56**	**65**	**242**

*) The data is suppressed to protect privacy

Figure 6. Final Report after Suppression

Here are the highlights of the suppression code.

The centerpiece of this code is the `suppress` macro:

```
%macro suppress(dsname=,supclass=);

   proc sort data=&dsname;
```

```
      by &supclass count;
   run;

   %let sup_flag = 0;
   data &dsname (drop=_supn_ _supsum_ avg_flg);
      set &dsname;
      by &supclass count;

      if first.&supclass then
      do; /* initialize number and sum of suppressed cells */
         _supn_ = 0;
         _supsum_ = 0;
      end;

      /* enhanced suppression flag */
      if (_supn_ ne 0) then avg_flg = (_supsum_/_supn_ <= &sup_avg);

      /* apply suppression criteria */
      if (count>0) and (count<=&sup_max or _supn_=1 or avg_flg) then
      do; /* suppress cell */
         count = .S;
         call symputx('sup_flag',1);
      end;

      if (count eq .S) then
      do; /* increment number and sum of suppressed cells */
         _supn_ + 1;
         _supsum_ + cnt_orig;
      end;
   run;

%mend suppress;
```

This macro performs a single iteration of the suppression algorithm. First, the source data table is sorted by `&supclass count`. This arranges the data set in a way where for each value of the `&supclass` counts are sorted from smallest to largest.

Then we loop through the source data set applying all three suppression criteria, and if at least one count has been suppressed, we set flag macro variable sup_flag = 1.

I highlighted (in colors) the pieces of the code that relate to the three suppression criteria:

Blue – primary suppression.

Green – complementary suppression (`_supn_ = 0;` and `_supn_ + 1;` are also used in the enhanced complementary suppression).

Red – enhanced complementary suppression.

The code snippet responsible for the suppression *iterations* across different dimensions is the `iterate_suppression` macro:

```
%macro iterate_suppression (class1=, class2=);

   %let nextclass = &class1;
   %do %while (&sup_flag); /* iterate suppression across dimension */

      %suppress(dsname=people_sup, supclass=&nextclass);
```

```
      /* flip dimension */
      %if &nextclass eq &class1
         %then %let nextclass = &class2;
         %else %let nextclass = &class1;

   %end;

%mend iterate_suppression;
```

It iteratively invokes the `suppress` macro and swaps the class variables with each iteration until the `suppress` macro stops producing any suppressed counts, that is when `sup_flag` macro variable becomes 0.

Suppression constraint parameters s_{max} and a_{max} are assigned as macro variables:

```
%let sup_max = 5; /* suppression constraint based on value   */
%let sup_avg = 1; /* suppression constraint based on average */
```

We also assign class variable names and format names to SAS macro variables:

```
%let tclass1 = race;
%let tclass2 = ageg;
%let f1 = $racef;
%let f2 = $agegf;
```

Then we calculate counts using PROC MEANS:

```
proc means data=people_roster noprint completetypes;
   format &tclass1 &f1.. &tclass2 &f2..;
   class &tclass1 &tclass2 / preloadfmt order=data;
   var count;
   output out=people_sup sum(count)=count sum(count)=cnt_orig;
run;
```

We use the `COMPLETETYPES` option along with the `PRELOADFMT` option to produce all combinations of the class variable values even when they are not present in the source data. That provides consistency of the data composition.

We create two duplicate variables representing counts using `sum` statistic, `count`, and `cnt_orig`, as we will be manipulating the first one (`count`) by assigning **.S** (suppressed value) while preserving the second one (`cnt_orig`) for use in the enhanced suppression.

Note that all SAS statistical procedures are smart enough not to calculate requested identical statistics twice. They calculate them once and just replicate the second variable. It is a more efficient way of creating duplicate columns than having an additional DATA step.

After that, we invoke the `iterate_suppression` macro that drives the whole suppression process:

```
%let sup_flag = 1;
%iterate_suppression (class1=&tclass1, class2=&tclass2);
```

And finally, when all the counts are suppressed according to our algorithm, we produce report using PROC REPORT where **.S** special missing values are formatted as *.

To better understand and visualize the process, I encourage you to visit my *blog* where you can download a variation of this suppression code that produces not just final report, but all the interim data sets and interim report outputs before and after each suppression iteration.

REMAINING CHALLENGES

While I believe that the algorithm and its SAS code implementation reliably protects PII in aggregated tabular reports by suppressing small counts, there are still challenges and unsolved problems waiting to be addressed:

- There is no solid mathematical foundation for optimizing the suppression constraint parameters s_{max} and a_{max} in order to optimize data suppression (minimize over-suppressing report data without sacrificing PII protection). At this point, these parameters are mostly set heuristically, if not arbitrarily.
- The suggested suppression algorithm and SAS implementation might produce different solutions (report outputs) depending on which classification variable you start with. This is another opportunity to optimize the suppression algorithm based on the report's cell values composition and the starting suppression dimension.
- Whether to keep zeros immune from suppression throughout all the suppression iterations or dynamically "sacrifice" zeros by revoking their "special protection status" when complementary suppression encroaches on rather large counts that might significantly reduce report usefulness.

REFERENCES

Committee on Privacy and Confidentiality, American Statistical Association. 2011. "Laws and Regulations about Privacy and Confidentiality." Available at http://community.amstat.org/cpc/lawsregulations.

European Commission. 2016. "Data protection: Rules for the protection of personal data inside and outside the EU." Available at http://ec.europa.eu/justice/data-protection/reform/files/regulation_oj_en.pdf.

Batkhan, Leonid. 2016. "Automatic data suppression in SAS reports." Available at https://blogs.sas.com/content/sgf/2016/04/13/automatic-data-suppression-in-sas-reports/.

ACKNOWLEDGEMENTS

I would like to thank Kristian Lønø of Statistics Norway for being an inspiration for this paper, and for his valuable suggestions and assistance in code testing. I would also like to thank Kita Garrido of SAS Institute Inc. for thoroughly reviewing this paper.

CONTACT INFORMATION

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

Leonid Batkhan
SAS Institute Inc.
919-531-0880
Leonid.Batkhan@sas.com
https://blogs.sas.com/content/author/leonidbatkhan

APPENDIX: SAS CODE TO IMPLEMENT SMALL COUNT SUPPRESSION

```
/* SAS sample program to accompany SGF-2018 paper by Leonid Batkhan,
   "Implementing Privacy Protection-Compliant SAS Aggregate Reports."
   This program illustrates data suppression in SAS tabular reports.
*/

/* Read in actual data, recode age to age group, ageg.                */
/* Pre-aggregated sample. In reality you would use person-level data. */
data people_roster;
   length ageg $1;
   input ageg $ race $ count;
   datalines;
1 A 14
1 B 17
1 H 9
1 M 15
1 W 18
2 A 11
2 B 8
2 H 4
2 M 12
2 W 13
3 M 7
3 B 6
3 H 24
3 W 19
4 A 10
4 B 16
4 H 19
4 W 20
;


%let sup_max = 5; /* suppression constraint based on value   */
%let sup_avg = 1; /* suppression constraint based on average */


/* assign other macro variables */
%let tclass1 = race;
%let tclass2 = ageg;
%let tlabel1 = Race;
%let tlabel2 = Age Group;
%let f1 = $racef;
%let f2 = $agegf;

%let hbcolor = edf2f9; /* header background color */
%let hfcolor = 3872ac; /* header font color       */


/* define user formats */
proc format;
   value $racef (notsorted)
   'M' = 'Martian'
   'A' = 'Asian'
   'B' = 'Black'
   'H' = 'Hispanic'
```

```
    'W' = 'White'
    ' ' = 'Total'
    ;
    value $agegf (notsorted)
    '1' = '<18'
    '2' = '19-64'
    '3' = '65-99'
    '4' = '100+'
    ' ' = 'Total'
    ;
    value cntf
    .S = '*'
    . = '0'
    other = [comma18.0]
    ;
    value $totbgf
    'Total' = "&hbcolor"
    ;
run;


/* suppression macro definition */
%macro suppress(dsname=,supclass=);

   proc sort data=&dsname;
      by &supclass count;
   run;

   %let sup_flag = 0;
   data &dsname (drop=_supn_ _supsum_ avg_flg);
      set &dsname;
      by &supclass count;

      if first.&supclass then
      do; /* initialize number and sum of suppressed cells */
         _supn_ = 0;
         _supsum_ = 0;
      end;

      /* enhanced suppression flag */
      if (_supn_ ne 0) then avg_flg = (_supsum_/_supn_ <= &sup_avg);

      /* apply suppression criteria */
      if (count>0) and (count<=&sup_max or _supn_=1 or avg_flg) then
      do; /* suppress cell */
         count = .S;
         call symputx('sup_flag',1);
      end;

      if (count eq .S) then
      do; /* increment number and sum of suppressed cells */
         _supn_ + 1;
         _supsum_ + cnt_orig;
      end;
   run;

%mend suppress;
```

```
/* suppression iteration macro definition */
%macro iterate_suppression (class1=, class2=);

   %let nextclass = &class1;
   %do %while (&sup_flag); /* iterate suppression across dimension */

      %suppress(dsname=people_sup, supclass=&nextclass);

      /* flip dimension */
      %if &nextclass eq &class1
         %then %let nextclass = &class2;
         %else %let nextclass = &class1;

   %end;

%mend iterate_suppression;


/* ---------------------------------------------- */
/*                START PROCESSING                */
/* ---------------------------------------------- */

/* get count summary by class variables */
proc means data=people_roster noprint completetypes;
   format &tclass1 &f1.. &tclass2 &f2..;
   class &tclass1 &tclass2 / preloadfmt order=data;
   var count;
   output out=people_sup sum(count)=count sum(count)=cnt_orig;
run;


%let sup_flag = 1;
%iterate_suppression (class1=&tclass1, class2=&tclass2);


/* generate output */
options nocenter pagesize=max;
title1 'People by Race and Age Group';
ods html style=styles.seaside path='C:\temp' file='report.html';


 /* generate footnote */
%let ftnote=;
data _null_;
   set people_sup (where=(count eq .S));
   call symput('ftnote',"j=L h=10pt '*) The data is suppressed to protect
privacy'");
   stop;
run;
footnote &ftnote;


/* get number of columns for table report */
data _null_;
   set people_sup (where=(_type_ eq 1)) end=e;
```

```
   totcol + 1;
   if e then call symputx('totcol',totcol + 2); *<-- Column number for Total;
run;
%put ***&totcol***;

/* produce tabular report */
proc report data=people_sup nowd split='*' spanrows missing
            style(header)=[font_size=11pt just=C color=#&hfcolor]
            style(column)=[font_size=10pt just=L];
      column &tclass1 count,&tclass2 count=tot;
      define &tclass1 / group  order=data "&tlabel1" f=&f1.. preloadfmt
style(column)=header[vjust=m cellwidth=1.5in color=#&hfcolor];
      define &tclass2 / across order=data "&tlabel2" f=&f2.. preloadfmt
style(header)=[background=$totbgf.];;
      define count    / display 'People Reported' style(column)=[just=R
cellwidth=1in] f=cntf.;
     define tot      / sum noprint;

     compute &tclass1;
      if &tclass1 eq ' ' then
      do;
         call define(_col_,'style',"style=[just=C font_size=11pt
color=#&hfcolor]");
            call define(_row_,'style',"style=[font_weight=bold
background=#&hbcolor]");
      end;
     endcomp;

     compute &tclass2;
           call define("_c&totcol._",'style',"style=[font_weight=bold just=R
background=#&hbcolor]");
     endcomp;
run;

ods html close;
```