PROC REPORT

PROC REPORT is the SAS System's most flexible reporting procedure. For a simple listing, PROC PRINT is your best choice, and for a summary table with a rectangular structure, PROC TABULATE is your best choice, but for most other reporting purposes, you'll often want to go with PROC REPORT. So why isn't it covered in BIOS 511? PROC REPORT's internal logic isn't as straightforward as that of PROC TABULATE so it's not as easy to explain, and added complexity comes from the ability to compute new variables within the procedure and also to explicitly control report break points (usually for special summarization). This introduction to PROC REPORT is intended to get you started so that you can produce some of your own tables with PROC REPORT and also understand PROC REPORT code that you might encounter.

While PROC REPORT has the ability to compute complicated statistics with grouping variables, typically programmers use it as a glorified PROC PRINT. They compute the values to be displayed in a data set with a useful structure, and then they feed that data set to PROC REPORT for a nice display. These notes include some examples where PROC REPORT is used for both computation and display, and other examples where it is used for display purposes only.

Basic syntax:

```
PROC REPORT DATA=data-set-name NOWD <options>;
        COLUMNS col1 col2 col3;      (more generally:  COLUMNS column-specifications;)
        DEFINE col1 / <usage> <attributes> <options>;       (more generally:  DEFINE report-item / …; )
        DEFINE col2 / <usage> <attributes> <options>;
        DEFINE col3 / <usage> <attributes> <options>;
        COMPUTE report-item </type-specification>;
                appropriate statements
        ENDCOMP;
        BREAK location break-variable </options>;
        RBREAK location </options>;
RUN;
```

The **COLUMNS** statement lists the items to appear in the columns of the report, controls the arrangement of those columns, and defines headings that span multiple columns. Items listed in the COLUMNS statement can be data set variables, statistics to be calculated by PROC REPORT, or variables that you want PROC REPORT to compute.

Each **DEFINE** statement provides details about one of those columns, including how the item should be treated (its role in the report), the text of its column heading (like a label; you can try inserting the default split character / to control where a heading splits, though this does not always work with the RTF destination), the format to use to display its values, and how to deal with missing values.

You should always use the **NOWD** (for NOWINDOWS) option on the PROC REPORT statement to prevent the procedure from bringing up a special report window that you won't want to use.

Plan the layout of your report

Before starting a report, think about what you want to show and what you want the report to look like. What do you want to display in the columns of the report? In what order should the columns appear? What should each row of the report represent: one observation or a group of observations? In what order should the rows appear?

For complex reports, you might even want to sketch the report. In formal report requests, people use what are called *report shells* or *table shells* that show everything about the report except for the cell values: all row and column headings, all report titles and footnotes, and other notes as necessary to describe what should be in each cell of the report.

Once you have the report layout in mind, you can plan the COLUMN and DEFINE statements needed to produce it.

Variable roles for use in a DEFINE statement

Specifying a role is not absolutely required, but I recommend that you specify a role (or usage) in every DEFINE statement.

DISPLAY	The values (possibly formatted) of DISPLAY variables are displayed. A report that contains one or more DISPLAY variables has a row for every observation in the data set. DISPLAY is the default for character variables.
ORDER	ORDER variables, which can be character or numeric, can be used to control the order of rows in a report. As with DISPLAY, a report that contains one or more ORDER variables has a row for every observation in the data set. The default order is ascending, but you can specify DESCENDING to reverse that. Note: If you want to use the values to control the row order but you do not want to actually print the values, you can specify NOPRINT on the DEFINE statement.
GROUP	GROUP variables act like CLASS variables in PROC TABULATE in the row dimension: they cause data to be consolidated into groups defined by the combinations of all unique (formatted) values of the GROUP variables. Each of those unique combinations becomes one row in the report.
ACROSS	An ACROSS variable is like having one of PROC TABULATE's CLASS variables in the column dimension. Each value of an ACROSS variable creates a column in the report. Thus, having both GROUP and ACROSS variables could create a cross-tabulation.
ANALYSIS	An ANALYSIS variable is a numeric variable that is used to calculate a statistic for all the observations represented by a cell in the report. (Which observations a cell represents is determined by the report's ORDER, GROUP, and ACROSS variables.) You associate a statistic with an analysis variable in either the COLUMN statement or in the variable's DEFINE statement. By default, numeric variables are treated as ANALYSIS variables with the statistic SUM.
COMPUTED	COMPUTED variables are not found in the input data set but you define them for the report. You add a computed variable by listing the computed variable in the COLUMN statement, defining the variable's usage as COMPUTED in its DEFINE statement, and using a *compute block* to compute the values of the variable. A compute block is also a great place to specify style information for a certain part of a report, typically by using CALL DEFINE, even when there is not a COMPUTED variable (see example 12a).

Common attributes and options specified in DEFINE statements

'columnheading'	Specify text to use as the column heading. PROC REPORT uses variable labels by default.
FORMAT=	Specify a SAS-supplied format or user-defined format to use to display values in the column.
MISSING	If a grouping-type variable has missing values, consider them as valid values.
STYLE=	Specify style information for use by ODS.
NOPRINT	Don't actually display the column (used in several examples and see the ORDER role above).
CENTER	Center values in the column.
RIGHT	Right-justify values in the column.
LEFT	Left-justify values in the column.

If you look at DEFINE statement documentation, you'll see many other options that can be specified after the / . Likewise, as with any other SAS reporting or analysis procedure, many useful options can be specified on the PROC REPORT statement itself. I leave such research to you!

Sample data set: DEMOG669

The sample data set used for most examples in this chapter is DEMOG669. This data set is similar to but not exactly the same as the DEMOG data set used by Jack Shostak in his book SAS Programming in the Pharmaceutical Industry. It contains 60 observations and the variables below. Note that all variables are numeric.

Variables in Creation Order

#	Variable	Type	Len	Label
1	subjid	Num	8	Subject ID
2	trt	Num	8	Treatment
3	gender	Num	8	Gender
4	race	Num	8	Race
5	age	Num	8	Age

SAS code for creating this data set, along with associated formats:

```
data data669.demog669;
      label subjid='Subject ID'
            trt   ='Treatment'
            gender='Gender'
            race  ='Race'
            age   ='Age';
      input subjid trt gender race age @@;
cards;
101 1 1 3 47 201 1 2 1 49 401 0 2 1 64 601 0 2 1 32 701 1 1 1 45
102 1 1 2 40 202 1 2 1 30 402 0 1 1 60 602 0 2 1 37 702 0 1 1 46
103 0 1 2 35 203 0 2 2 38 403 1 2 1 57 603 1 2 3 40 703 1 1 2 40
104 0 2 1 33 204 1 1 3 31 404 0 2 3 58 604 1 1 1 41 704 0 2 1 38
105 0 2 1 40 205 0 1 3 47 405 0 2 2 47 605 0 1 1 35 705 0 2 3 64
106 1 1 1 44 206 0 1 2 52 406 0 1 1 39 606 1 1 2 33 706 1 1 2 57
107 0 1 1 57 301 0 2 2 63 407 1 1 3 52 607 1 1 2 42 707 1 2 1 58
108 1 2 3 24 302 0 1 1 44 408 1 1 3 50 608 1 1 2 47 708 1 2 1 49
109 1 2 2 61 303 1 2 1 52 409 0 2 2 61 609 0 1 1 40 709 0 2 1 51
110 1 1 1 52 304 1 1 1 58 410 1 1 1 62 610 1 2 1 32 710 1 2 1 30
111 0 1 1 48 305 1 1 . 37 411 1 2 2 52 611 0 2 1 25 711 0 1 1 32
112 0 2 1 26 306 0 2 1 25 412 1 2 1 56 612 0 2 1 29 712 0 1 1 56
;

proc format;
      value trt           1='Active'
                          0='Placebo';
      value gender        1='Male'
                          2='Female';
      value race          1='White'
                          2='Black'
                          3='Other';
run;
```

Syntax-illustrating examples in which PROC REPORT does the calculation and display work

Example 1: Running PROC REPORT on a data set with all numeric variables, no statements or options

```
PROC REPORT DATA=bios669.demog669 NOWD;
RUN;
```

No statements or options, all numeric variables

Subject ID	*Treatment*	*Gender*	*Race*	*Age*
24954	30	90	91	2690

By default, with no COLUMN or DEFINE statement, and with all numeric variables, PROC REPORT displays the variables in their order in the data set, and it displays the sum of each variable. Note that unlike PROC PRINT, PROC REPORT automatically uses variable labels as column headings.

Example 2: Using a DISPLAY variable

When we designate a DISPLAY variable, PROC REPORT lists all observations rather than summarizing. Since we did not specify differently here, variables and observations are listed in their order in the data set.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     DEFINE subjid / DISPLAY;
RUN;
```

Using a DISPLAY variable

Subject ID	*Treatment*	*Gender*	*Race*	*Age*
101	1	1	3	47
201	1	2	1	49
401	0	2	1	64
601	0	2	1	32
701	1	1	1	45
102	1	1	2	40
202	1	2	1	30
402	0	1	1	60
602	0	2	1	37
702	0	1	1	46
103	0	1	2	35
203	0	2	2	38
403	1	2	1	57
603	1	2	3	40
703	1	1	2	40

and so forth…

Example 3: Adding an ORDER variable

```
PROC REPORT DATA=bios669.demog669 NOWD;
     DEFINE subjid / DISPLAY;
     DEFINE age / ORDER;
RUN;
```

Adding an ORDER variable

Subject ID	*Treatment*	*Gender*	*Race*	*Age*
108	1	2	3	24
611	0	2	1	25
306	0	2	1	
112	0	2	1	26
612	0	2	1	29
202	1	2	1	30
710	1	2	1	
204	1	1	3	31
601	0	2	1	32
610	1	2	1	
711	0	1	1	
104	0	2	1	33
606	1	1	2	
103	0	1	2	35
605	0	1	1	

Now PROC REPORT has listed the observations in order by variable AGE. Note that if multiple observations have the same ORDER variable value, the value is omitted after the initial occurrence.

Example 4: Controlling the column order

If we have an ORDER variable, and for many other reasons, we will usually want to control column order in the report, and for that we use the COLUMNS statement.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS age subjid trt gender race;
     DEFINE subjid / DISPLAY;
     DEFINE age / ORDER;
RUN;
```

Controlling the column order

Age	*Subject ID*	*Treatment*	*Gender*	*Race*
24	108	1	2	3
25	611	0	2	1
	306	0	2	1
26	112	0	2	1
29	612	0	2	1
30	202	1	2	1
	710	1	2	1
31	204	1	1	3
32	601	0	2	1
	610	1	2	1
	711	0	1	1
33	104	0	2	1
	606	1	1	2
35	103	0	1	2
	605	0	1	1
37	602	0	2	1
	305	1	1	.
38	203	0	2	2
	704	0	2	1

Example 5: Using additional order controls

The output above shows that the original data set is not sorted by subject ID. (Probably a silly decision by the person taking care of this data!) In detailed reports on this data, we will want it in subject ID order. We can do this universally by sorting and then reporting, or we can do it in a single report by making subjid an ORDER variable. The following two sets of code produce the same result.

```
PROC SORT DATA=bios669.demog669 OUT=SortedBySubjid; BY subjid; RUN;
PROC REPORT DATA=SortedBySubjid NOWD;
     COLUMNS age subjid trt gender race;
     DEFINE subjid / DISPLAY;
     DEFINE age / ORDER;
RUN;

PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS age subjid trt gender race;
     DEFINE subjid / ORDER;
     DEFINE age / ORDER;
RUN;
```

```
                Using additional order controls

         Subject
   Age      ID    Treatment   Gender   Race
    24     108            1        2      3
    25     306            0        2      1
           611            0        2      1
    26     112            0        2      1
    29     612            0        2      1
    30     202            1        2      1
           710            1        2      1
    31     204            1        1      3
    32     601            0        2      1
           610            1        2      1
           711            0        1      1
```

Example 6: Computing statistics and using groups and formats

PROC REPORT can do more than listing observations. If you use a GROUP or ACROSS variable, it does summarization within groups defined by that variable. What's the difference between GROUP and ACROSS? GROUP variable values define rows, and ACROSS variable values define columns. Statistics are computed for the ANALYSIS variables, with the DEFINE statement also specifying the particular statistic desired (N, MEAN, STD, NMISS, etc.)

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS trt age;
     DEFINE trt / GROUP;
     DEFINE age / ANALYSIS MEAN;
RUN;
```

Computing statistics within groups

```
Treatment        Age
        0  44.066667
        1       45.6
```

We can improve this report by adding formats, which we specify in the DEFINE statements. These formats can be either SAS-supplied or user-defined.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS trt age;
     DEFINE trt / GROUP FORMAT=trt.;
     DEFINE age / ANALYSIS MEAN FORMAT=5.2;
RUN;
```

Computing statistics within groups

```
Treatment    Age
   Active  45.60
  Placebo  44.07
```

Example 7: Using aliases to compute multiple statistics for a variable, also a spanning header

Since we can only specify one statistic on a particular DEFINE statement, and since each DEFINE statement only defines one column, we need to use "aliases" to define multiple statistics for a single variable. In this example, agen and agemean are aliases for age, as specified in the COLUMN statement. Once these aliases are established, we use a separate DEFINE statement for each one.

The DEFINE statements for agen and agemean include strings to be used as headers for their columns. You can specify such a header string in any DEFINE statement.

This example also shows that the COLUMN statement can be used to specify what are called "spanning headers". The character string 'Age' is to be used as a header spanning the other items specified within the parentheses.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS trt ('Age' age=agen age=agemean);
     DEFINE trt / GROUP FORMAT=trt.;
     DEFINE agen / ANALYSIS N 'N';
     DEFINE agemean / ANALYSIS MEAN FORMAT=5.2 'Mean';
RUN;
```

Using aliases to compute multiple statistics for a variable

	Age	
Treatment	*N*	*Mean*
Active	30	45.60
Placebo	30	44.07

Example 8: Producing a crosstab with ACROSS and GROUP

If you use both a GROUP variable and an ACROSS variable, you can produce something like a cross-tabulation.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS gender trt;
     DEFINE trt / ACROSS FORMAT=trt.;
     DEFINE gender / GROUP FORMAT=gender.;
RUN;
```

Producing a crosstab with ACROSS and GROUP

	Treatment	
Gender	*Active*	*Placebo*
Female	13	17
Male	17	13

Example 9: Summarizing an entire report using RBREAK

The RBREAK statement can be used to produce a report summary at either the beginning or end of a report (or each BY group, if a BY variable is being used).

This example also starts to show style options that you can use if you are producing RTF output (all output in this chapter is pasted from an RTF file produced with the JOURNAL style and the BODYTITLE option). Style options also work if you are producing PDF or HTML output, in addition to some other ODS destinations. Style options can be specified on DEFINE statements, BREAK and RBREAK statements, in COMPUTE blocks, or on the PROC REPORT statement itself.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS gender trt;
     DEFINE trt / ACROSS FORMAT=trt. STYLE=[JUST=CENTER];
     DEFINE gender / GROUP FORMAT=gender.;
     RBREAK AFTER / SUMMARIZE STYLE=[BACKGROUNDCOLOR=ltgray];
RUN;
```

Summarizing the entire report using RBREAK

	Treatment	
Gender	*Active*	*Placebo*
Female	13	17
Male	17	13
	30	*30*

Example 10: Using multiple grouping variables

If you specify multiple GROUP variables, they are nested and statistics are produced for each combination of values.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS trt gender age;
     DEFINE trt / GROUP FORMAT=trt.;
     DEFINE gender / GROUP FORMAT=gender.;
     DEFINE age / ANALYSIS MEAN FORMAT=5.2;
RUN;
```

Using multiple grouping variables

Treatment	*Gender*	*Age*
Active	Female	45.38
	Male	45.76
Placebo	Female	43.00
	Male	45.46

Example 11: Adding summaries for groups using BREAK

If we use the same code as Example 10 but add a BREAK statement, we get summary statistics every time the value of the BREAK variable changes.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS trt gender age;
     DEFINE trt / GROUP FORMAT=trt.;
     DEFINE gender / GROUP FORMAT=gender.;
     DEFINE age / ANALYSIS MEAN FORMAT=5.2;
     BREAK AFTER trt / SUMMARIZE;
RUN;
```

Adding summaries for groups using BREAK

Treatment	*Gender*	*Age*
Active	Female	45.38
	Male	45.76
Active		*45.60*
Placebo	Female	43.00
	Male	45.46
Placebo		*44.07*

Example 12: Using a COMPUTED variable and a COMPUTE block

PROC REPORT is one of the few SAS procedures that can actually compute new variables, like a DATA step. This is done within a COMPUTE block, which has the syntax used below. Associated with a COMPUTE block will be a variable designated as COMPUTED in its DEFINE statement. In this example, that variable is agemonthsmean. Our goal here is to compute and display the average age in months of the members of each treatment group.

This example also shows use of the NOPRINT option, which is used here with variable age. We don't want age to appear in our report, but we need the mean of age in order to compute agemonthsmean. So we need to list age in our COLUMNS statement and include a DEFINE statement for it which designates it as an ANALYSIS variable with desired statistic MEAN. Specifying NOPRINT means that the mean ages are not printed, but the age mean (age.mean in PROC REPORT shorthand) is available for use in the COMPUTE block.

Note: The order of variables in your COLUMNS statement matters when you are using a computed variable. The component variables must appear before the computed variable. To see that this is the case, you could try the code below with COLUMNS trt agemonthsmean age;

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS trt age agemonthsmean;
     DEFINE trt / GROUP FORMAT=trt.;
     DEFINE age / ANALYSIS MEAN NOPRINT;
     DEFINE agemonthsmean / COMPUTED 'Average Age in Months';
     COMPUTE agemonthsmean;
          agemonthsmean=age.mean*12;
     ENDCOMP;
RUN;
```

Using a COMPUTED variable and a COMPUTE block

Treatment	*Average Age in Months*
Active	547.2
Placebo	528.8

Example 12a: More on using STYLE options and COMPUTE blocks

Besides using STYLE options to control almost any report component, you can also use COMPUTE blocks to highlight rows and columns based on values of variables in the report.

```
TITLE 'More on using STYLE options and COMPUTE blocks';
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS subjid trt ('Demographics' age gender race);
     DEFINE subjid / ORDER STYLE(HEADER)=[FONT_WEIGHT=BOLD];
     DEFINE trt / DISPLAY STYLE=[FONT_WEIGHT=BOLD JUST=CENTER];
     DEFINE race / DISPLAY;
     DEFINE age / DISPLAY STYLE=[CELLWIDTH=1.5cm];
     COMPUTE subjid;
          IF MOD(subjid,2) THEN DO;
               CALL DEFINE(_row_,"STYLE","STYLE=[BACKGROUND=
                    cxDDDDDD]");
          END;
     ENDCOMP;
     COMPUTE race;
          IF race=3 THEN DO;
               CALL DEFINE(_col_,"STYLE","STYLE=[FONT_WEIGHT=Bold]");
          END;
     ENDCOMP;
RUN;
```

More on using STYLE options and COMPUTE blocks

		Demographics		
Subject ID	***Treatment***	*Age*	*Gender*	*Race*
101	**1**	47	1	**3**
102	**1**	40	1	2
103	**0**	35	1	2
104	**0**	33	2	1
105	**0**	40	2	1
106	**1**	44	1	1
107	**0**	57	1	1
108	**1**	24	2	**3**
109	**1**	61	2	2
110	**1**	52	1	1
111	**0**	48	1	1

The following variation of this table is not very appealing, but I include it to show you how to highlight an entire column including the heading (see subjid DEFINE statement) vs. how to highlight just the data area, that is the cells (see gender COMPUTE block).

```
TITLE "More on using STYLE options and COMPUTE blocks";
TITLE2 "Check highlighting of subjid and gender columns";
PROC REPORT NOWD DATA=bios669.demog669;
    COLUMNS subjid trt ('Demographics' age gender race);
    DEFINE subjid / ORDER STYLE(HEADER)=[FONT_WEIGHT=BOLD]
STYLE=[BACKGROUND=cxDDDDDD];
    DEFINE trt / DISPLAY STYLE=[FONT_WEIGHT=BOLD JUST=CENTER];
    DEFINE race / DISPLAY;
    DEFINE age / DISPLAY STYLE=[CELLWIDTH=1.5cm];
    COMPUTE race;
        IF race=3 THEN DO;
            CALL DEFINE(_col_,"STYLE","STYLE=[FONT_WEIGHT=Bold]");
        end;
     ENDCOMP;
     COMPUTE gender;
        CALL DEFINE(_col_,"STYLE","STYLE=[BACKGROUND=cxDDDDDD]");
     ENDCOMP;
RUN;
```

More on using STYLE options and COMPUTE blocks
Check highlighting of subjid and gender columns

		Demographics		
Subject ID	***Treatment***	*Age*	*Gender*	*Race*
101	**1**	47	1	**3**
102	**1**	40	1	2
103	**0**	35	1	2
104	**0**	33	2	1
105	**0**	40	2	1
106	**1**	44	1	1
107	**0**	57	1	1
108	**1**	24	2	**3**
109	**1**	61	2	2
110	**1**	52	1	1
111	**0**	48	1	1

Example 12b: Using COMPUTE blocks to add a row counter to a PROC REPORT listing

What if you are using PROC REPORT to do a listing report, and someone asks you to add a row counter such as PROC PRINT displays by default? Assuming that you can't simply switch your report to PROC PRINT, here is some code to add this "feature" with PROC REPORT. This code also demonstrates more about how COMPUTE blocks work with and without the BEFORE and AFTER keywords.

```
PROC REPORT NOWD DATA=sashelp.class;
        COLUMN obs name;
        DEFINE obs / COMPUTED 'Obs' F=3.;
        COMPUTE BEFORE;
                obsnum=0;
        ENDCOMP;
        COMPUTE obs;
                obsnum+1;
                obs=obsnum;
        ENDCOMP;
RUN;
```

Adding a PROC PRINT-like row counter

```
Obs  Name
  1  Alfred
  2  Alice
  3  Barbara
  4  Carol
  5  Henry
  6  James
  7  Jane
  8  Janet
  9  Jeffrey
 10  John
 11  Joyce
 12  Judy
 13  Louise
 14  Mary
 15  Philip
 16  Robert
 17  Ronald
 18  Thomas
 19  William
```

Example 12c: Printing a special note if no data exists for a report

It would be nice in some situations to have report code that produced a helpful report whether or not any data satisfied the criteria for the report. This would be very robust reporting code! For example, you can imagine preparing an adverse events report for a regular clinical trials report, but sometimes no adverse events might have occurred.

In this example, I'm presenting this situation for a very simple report based on the sashelp.class data set. If there is data to satisfy the report needs, the code below produces this report:

Name	*Age*
Alfred	14
Alice	13
Barbara	13
Carol	14
Henry	14

and so forth…

And here's the report produced by the program below in the situation where no data exist to satisfy the report's needs:

Name	*Age*
************* No data met the criteria for this display *************	

After running the code below and reading its comments and looking at all of the output, you could probably implement this code in your own situation.

```
title; footnote; options nodate nonumber;
%let odsdir=P:\BIOS 669\PROC REPORT;

ODS RTF STYLE=JOURNAL BODYTITLE FILE="&odsdir.\NoDataMetCriteria.rtf";

*** the basic desired report;
proc report nowd data=sashelp.class;
    columns name age;
```

```
run;

*** to illustrate our point, create a table with 0 obs but ;
*** the same basic structure as the data set to be reported on;
proc sql noprint;
    create table work.class
        like sashelp.class;
quit;

*** by default, we simply get no output when reporting on this 0 obs data
set;
proc report nowd data=work.class;
    columns name age;
run;

*** instead, we would like the report to automatically tell us ;
*** if it has been run on a data set with 0 observations;

*** use PROC SQL to count the observations in the data set;
proc sql ;
    select * from work.class;
%put &sqlobs;

*** store that count in an OBS variable;
data blank;
    obs=&sqlobs;
run;

***  add that OBS variable to our report data set;
data classR;
    set class(in=in1) blank;
    if in1 then obs=_N_;
run;

*** now run our report on the augmented report data set, using OBS as a not
printed order variable;
*** if OBS is 0, the compute block prints a message;
options missing=' ';
proc report nowd data=classR;
    columns obs name age;
    define obs / order noprint;

    compute before obs;
        if obs=0 then do;
            text1=" ************* No data met the criteria for this display
************* ";
            line @1 text1 $;
        end;
    endcomp;
run;

*** run parallel code showing that the special message is NOT printed;
*** if our report data set DOES have observations;
```

```
proc sql ;
    select * from sashelp.class;
%put &sqlobs;

data blank;
    obs=&sqlobs;
run;

data classR;
    set sashelp.class(in=in1) blank;
    if in1 then obs=_N_;
run;


options missing=' ';
proc report nowd data=classR;
    columns obs name age;
    define obs / order noprint;

    compute before obs;
        if obs=0 then do;
            text1=" ************* No data met the criteria for this display
************* ";
            line @1 text1 $;
        end;
    endcomp;
run;

ODS RTF CLOSE;
```

Example 13: Can PROC REPORT do percentages?

You might recall that PROC TABULATE is an ace at producing different types of percentages, but not so PROC REPORT. The only built-in percentage statistic available is PCTSUM, which can be used with an ANALYSIS variable and uses as the denominator the sum of the variable for the entire report or BY group. As an alternative, if we have an indicator variable we can compute its mean and display that as a percentage, as in this example. Note that the PERCENT. format makes the values display on a 0-100 scale.

```
PROC REPORT DATA=bios669.demog669 NOWD;
     COLUMNS gender trt;
     DEFINE gender / GROUP FORMAT=gender.;
     DEFINE trt / ANALYSIS MEAN FORMAT=PERCENT7.1 'Percent in Active Group';
RUN;
```

Can PROC REPORT do percentages?

Gender	*Percent in Active Group*
Female	43.3%
Male	56.7%

PROC REPORT's weakness in computing percentages is one reason why people often use other procedures to compute values for reports, producing and combining output data sets until they have a data set structured for nice display with PROC REPORT. You'll see examples of this soon.

Example 14: Displaying long text fields

PROC REPORT is good at displaying text fields, and when writing to the RTF (or PDF or HTML) destination it automatically wraps values as necessary.

```
DATA addnotes;
     MERGE SortedBySubjid data669.demogtext;
     BY subjid;
RUN;

PROC REPORT DATA=addnotes NOWD;
     DEFINE subjid / DISPLAY;
     DEFINE longnote / 'Comment';
RUN;
```

Displaying long text fields

Subject ID	*Treatment*	*Gender*	*Race*	*Age*	*Comment*
101	1	1	3	47	
102	1	1	2	40	
103	0	1	2	35	
104	0	2	1	33	This participant says hi to her friend Mabel in Minnesota.
105	0	2	1	40	
106	1	1	1	44	

```
PROC REPORT DATA=addnotes NOWD;
     DEFINE subjid / DISPLAY;
     DEFINE longnote / 'Comment' STYLE=[CELLWIDTH=2in];
RUN;
```

Displaying long text fields with a controlled column width

Subject ID	*Treatment*	*Gender*	*Race*	*Age*	*Comment*
101	1	1	3	47	
102	1	1	2	40	
103	0	1	2	35	
104	0	2	1	33	This participant says hi to her friend Mabel in Minnesota.
105	0	2	1	40	
106	1	1	1	44	

Example 15: Controlling additional style characteristics

Several examples above show use of the STYLE=[ ] option to control aspects of the visual display of a particular column. I'll add more examples as I have time, or see the references. Support for this option is a fun and powerful feature of PROC REPORT. You can specify multiple style characteristics within one pair of brackets, specifying font color, background color, font, fontsize, cellwidth, etc.

Examples in which PROC REPORT is simply used to display pre-calculated results

In actual practice, PROC REPORT is probably used more to display pre-calculated results than to do all calculations along with the display. In my workplaces I have seen PROC REPORT used for some amazing reports. Below are a couple of typical examples that should teach you some useful tricks.

Fancy Example 1: Displaying counts and percentages

Imagine that you conceive or receive a table shell such as the following for a report on your study's demographic data:

Treatment Group Composition by Race and Gender

	Active (N=#)	Placebo (N=#)
Race		
White	N (%)	
Black		
Other		
Gender		
Male		
Female		

Notes to programmer: The denominator for all percentages should be the N of the treatment group. If a grouping variable has missing values, include a row for Missing as the last value for that variable.

Can we produce an RTF version of this with PROC REPORT? Absolutely!

```
LIBNAME bios669 'C:\Users\Kathy\Documents\669 data';
%LET odsdir=C:\Users\Kathy\Documents\output;
OPTIONS NODATE NONUMBER;
TITLE;

*** set up "hard spaces" to use for indenting in RTF file;
DATA _NULL_  ;
  CALL SYMPUT('b',LEFT(INPUT("A0",$hex2.)))  ;
RUN;
%LET c=&b&b&b;
%LET d=&c&c&c;

*** compute basic counts and percents required by table shell;
```

```
TITLE 'Basic counts and percents';
PROC FREQ DATA=data669.demog669;
     TABLES trt*race / OUTPCT OUT=racecnt MISSING;
     TABLES trt*gender / OUTPCT OUT=gendcnt MISSING;
     TABLES trt / OUT=trtcnt;
RUN;

*** see structure of output data sets - which % variable uses;
*** right denominator?;
TITLE 'FREQ output data set';
PROC PRINT DATA=racecnt; RUN;

*** store treatment group counts in macro variables for later display;
*** in column headings;
DATA _NULL_;
     SET trtcnt;
     IF trt=0 THEN CALL SYMPUT('p',PUT(count,2.));
     IF trt=1 THEN CALL SYMPUT('a',PUT(count,2.));
RUN;
%PUT p=&p a=&a ;

*** create N (%) character values requested in table shell;
*** (simplify data set BEFORE transposing);
DATA racepct;
     SET racecnt(KEEP=trt race count pct_row);
     LENGTH cp $10;
     cp=PUT(count,2.) || ' (' || PUT(pct_row,2.) || '%)';
RUN;
DATA gendpct;
     SET gendcnt(KEEP=trt gender count pct_row);
     LENGTH cp $10;
     cp=PUT(count,2.) || ' (' || PUT(pct_row,2.) || '%)';
RUN;

*** basic transposition by race or gender to get active & placebo values;
*** on same record per table shell;
PROC SORT DATA=racepct; BY race; RUN;
PROC TRANSPOSE DATA=racepct(KEEP=trt race cp) OUT=racetrn PREFIX=trt;
     BY race;
     ID trt;
     VAR cp;
RUN;
```

```
PROC SORT DATA=gendpct; BY gender; RUN;
PROC TRANSPOSE DATA=gendpct(KEEP=trt gender cp) OUT=gendtrn PREFIX=trt;
     BY gender;
     ID trt;
     VAR cp;
RUN;

*** make sure transposed tables are similar to table shell;
TITLE 'TRANSPOSE output data set';
PROC PRINT DATA=racetrn; RUN;

*** stack transposed data sets, adding an order variable and rows for;
*** headers per the table shell;
*** (is there a better way to assign order values?);
DATA racetrn2(DROP=_NAME_ RENAME=(race=variable));
     SET racetrn;
     LENGTH vartext $20;
     IF race=1 THEN DO; order=2; vartext="&c&c.White";   END;
     IF race=2 THEN DO; order=3; vartext="&c&c.Black";   END;
     IF race=3 THEN DO; order=4; vartext="&c&c.Other";   END;
     IF race=. THEN DO; order=5; vartext="&c&c.Missing"; END;
RUN;
PROC SORT DATA=racetrn2; BY order; RUN;

DATA gendtrn2(DROP=_NAME_ RENAME=(gender=variable));
     SET gendtrn;
     LENGTH vartext $20;
     IF gender=1 THEN DO; order=7; vartext="&c&c.Male";   END;
     IF gender=2 THEN DO; order=8; vartext="&c&c.Female"; END;
RUN;

* this step makes rows for headers;
DATA varheads;
     LENGTH vartext $20;
     order=1; vartext='Race';   OUTPUT;
     order=6; vartext='Gender'; OUTPUT;
RUN;

* and finally we stack, ordering by our order variable;
DATA for_report;
     SET racetrn2
          gendtrn2
          varheads;
     BY order;
RUN;

*** finally write the report;
ODS RTF FILE="&odsdir/fancy_report1.rtf" BODYTITLE STYLE=JOURNAL;
TITLE "Treatment Group Composition by Race and Gender";
PROC REPORT DATA=for_report NOWD;
     COLUMNS order vartext trt1 trt0;
     DEFINE order / ORDER NOPRINT;
     DEFINE vartext / DISPLAY ' ';
     DEFINE trt1 / DISPLAY "Active (N=&a)" RIGHT;
     DEFINE trt0 / DISPLAY "Placebo (N=&p)" RIGHT;
     COMPUTE vartext;
          IF order=1 OR order=6 THEN CALL DEFINE(_ROW_,"STYLE",
               "style=[fontweight=bold]");
     ENDCOMP;
RUN;
TITLE;
ODS RTF CLOSE;
```

Treatment Group Composition by Race and Gender

	Active (N=30)	*Placebo (N=30)*
Race		
White	15 (50%)	21 (70%)
Black	8 (27%)	6 (20%)
Other	6 (20%)	3 (10%)
Missing	1 (3%)	
Gender		
Male	17 (57%)	13 (43%)
Female	13 (43%)	17 (57%)

Fancy Example 2: Displaying continuous and categorical variables in one table

The following type of demographics table is commonly called “Table 1” since such a look at a study population is typically the first display in a manuscript or comprehensive report. This table was made using BIOS 511 LRC120, which can be found on Sakai with the other selected BIOS 511 data sets.

Table 1: Demographics

	Active (N=56)	*Placebo (N=64)*	*Overall (N=120)*	*P-value**
Age (years)				0.2651
N	56	64	120	
Mean	49.2	50.1	49.7	
Standard Deviation	5.71	6.77	6.29	
Minimum	37.0	35.0	35.0	
Maximum	58.0	60.0	60.0	
Smoking Status				0.6873
Non-smoker	15 (26.8%)	20 (31.3%)	35 (29.2%)	
Smoker	38 (67.9%)	43 (67.2%)	81 (67.5%)	

**** P-values: Age = Wilcoxon rank-sum, Smoking Status = Pearson's chi-square***

```
/* creating a demographics summary table */

/*******************************************************************
    The outline of this code is taken from Example 5.3 of Jack
    Shostak's SAS Programming in the Pharmaceutical Industry:
    Creating a Typical Summary of Demographics
 *******************************************************************/

/*******************************************************************

    In our table I will use BIOS 511's LRC120 data set, using
    treatment variable Trt, continuous variable Age,
    and categorical variable Smoke.

 *******************************************************************/

*libname bios511 'C:\BIOS 613 data\BIOS511';
libname bios511 'P:\Sakai\Sakai 2016\Data\BIOS511';
%*let odsdir=P:\BIOS 613\PROC REPORT;
%*let odsdir=C:\Users\ucckjr\Documents\BIOS 613\PROC REPORT;
```

```
%let odsdir=P:\Sakai\Sakai 2016\Course units\04 PROC REPORT and reporting in
general;

*** set up "hard spaces" to use for indenting in RTF file;
DATA _NULL_  ;
  CALL SYMPUT('b',LEFT(INPUT("A0",$hex2.)))  ;
RUN;
%LET c=&b&b&b;
%LET d=&c&c&c;


data demog1;
    set bios511.lrc120;

    label Age   = 'Age'
          Smoke = 'Smoking Status';

run;

**** CREATE FORMATS NEEDED FOR TABLE ROWS.;
proc format;
   value smoke
      0 = "&c&c.Non-smoker"
      1 = "&c&c.Smoker";
run;


**** DUPLICATE THE INCOMING DATA SET FOR OVERALL COLUMN
**** CALCULATIONS SO NOW TRT HAS VALUES 0 = PLACEBO, 1 = DRUG,
**** AND 2 = OVERALL.;
data demog2;
   set demog1;
   output;
   trt = 2;
   output;
run;


**** AGE STATISTICS PROGRAMMING ********************************;
**** GET P VALUE FROM NON-PARAMETRIC COMPARISON OF AGE MEANS.;
proc npar1way
   data = demog2
   wilcoxon
   noprint;
      where trt in (0,1);
      class trt;
      var age;
      output out = pvalue wilcoxon;
run;

proc sort
   data = demog2;
      by trt;
run;

***** GET AGE DESCRIPTIVE STATISTICS N, MEAN, STD, MIN, AND MAX.;
proc means
   data = demog2 noprint;
      by trt;

      var age;
      output out = age1
             n = _n mean = _mean std = _std min = _min
             max = _max;
run;

**** FORMAT AGE DESCRIPTIVE STATISTICS FOR THE TABLE.;
data age2;
   set age1;

   format n mean std min max $14.;
   drop _n _mean _std _min _max;

   n = put(_n,3.);
   mean = put(_mean,7.1);
   std = put(_std,8.2);
   min = put(_min,7.1);
   max = put(_max,7.1);
run;

**** TRANSPOSE AGE DESCRIPTIVE STATISTICS INTO COLUMNS.;
proc transpose
   data = age2
   out = age3
   prefix = col;
      var n mean std min max;
      id trt;
run;

**** CREATE AGE FIRST ROW FOR THE TABLE.;
data label1;
   set pvalue(keep = p2_wil rename = (p2_wil = pvalue));
   length label $ 85;
   label = "Age (years)";
```

```
run;

**** APPEND AGE DESCRIPTIVE STATISTICS TO AGE P VALUE ROW AND
**** CREATE AGE DESCRIPTIVE STATISTIC ROW LABELS.;
data age4;
   length label $ 85 col0 col1 col2 $ 25 ;
   set label1 age3;

   keep label col0 col1 col2 pvalue ;
   if _n_ > 1 then
      select;
         when(_NAME_ = 'n')    label = "&c&c.N";
         when(_NAME_ = 'mean') label = "&c&c.Mean";
         when(_NAME_ = 'std')  label = "&c&c.Standard Deviation";
         when(_NAME_ = 'min')  label = "&c&c.Minimum";
         when(_NAME_ = 'max')  label = "&c&c.Maximum";
         otherwise;
      end;
run;
**** END OF AGE STATISTICS PROGRAMMING *************************;


**** SMOKING STATISTICS PROGRAMMING *****************************;
**** GET SIMPLE FREQUENCY COUNTS FOR GENDER.;
proc freq
   data = demog2
   noprint;
      where trt ne .;
      tables trt * smoke / missing outpct out = smoke1;
run;

**** FORMAT Smoke N(%) AS DESIRED.;
data smoke2;
   set smoke1;
      where smoke ne .;
      length value $25;
      value = put(count,4.) || ' (' || put(pct_row,5.1)||'%)';
run;

proc sort
   data = smoke2;
      by smoke;
run;

**** TRANSPOSE THE SMOKE SUMMARY STATISTICS.;
proc transpose
   data = smoke2
   out = smoke3(drop = _name_)
   prefix = col;
      by smoke;
      var value;
      id trt;
run;

**** PERFORM CHI-SQUARE ON SMOKE COMPARING ACTIVE VS PLACEBO.;
proc freq
   data = demog2
   noprint;
      where smoke ne . and trt not in (.,2);
      table smoke * trt / chisq;
      output out = pvalue pchi;
run;

**** CREATE SMOKE FIRST ROW FOR THE TABLE.;
data label2;
   set pvalue(keep = p_pchi rename = (p_pchi = pvalue));
   length label $ 85;
   label = "Smoking Status";
run;

**** APPEND SMOKE DESCRIPTIVE STATISTICS TO SMOKE P VALUE ROW
**** AND CREATE SMOKE DESCRIPTIVE STATISTIC ROW LABELS.;
data smoke4;
   length label $ 85 col0 col1 col2 $ 25 ;
   set label2 smoke3;

   keep label col0 col1 col2 pvalue ;
   if _n_ > 1 then
        label= put(smoke,smoke.);
run;
**** END OF SMOKE STATISTICS PROGRAMMING *********************;



**** CONCATENATE AGE AND SMOKE STATISTICS AND CREATE
**** GROUPING GROUP VARIABLE FOR LINE SKIPPING IN PROC REPORT.;
**** (Unfortunately, this way of line skipping does not work in ODS
destinations.);
data forreport;
   set age4(in = in1)
       smoke4(in = in2)
       ;

       group = sum(in1 * 1, in2 * 2);
```

```
run;


**** DEFINE THREE MACRO VARIABLES &N0, &N1, AND &NT THAT ARE USED
**** IN THE COLUMN HEADERS FOR "PLACEBO," "ACTIVE" AND "OVERALL"
**** THERAPY GROUPS.;
data _null_;
   set demog1 end = eof;

   **** CREATE COUNTER FOR N0 = PLACEBO, N1 = ACTIVE.;
   if trt = 0 then
      n0 + 1;
   else if trt = 1 then
      n1 + 1;

   **** CREATE OVERALL COUNTER NT.;
   nt + 1;

   **** CREATE MACRO VARIABLES &N0, &N1, AND &NT.;
   if eof then
      do;
         call symput("n0",compress('(N='||put(n0,4.) || ')'));
         call symput("n1",compress('(N='||put(n1,4.) || ')'));
         call symput("nt",compress('(N='||put(nt,4.) || ')'));
      end;
run;
%put first:  &n0 &n1 &nt;


**** alternative N calculation technique;
proc sql noprint;
    select n(trt) into :c0 - :c1
    from demog1
    group by trt ;
quit;
%let ct=%eval(&c0+&c1);


**** a different SQL N calculation technique;
proc sql noprint;
    select count(*) into :c0 from demog1 where trt=0;
    select count(*) into :c1 from demog1 where trt=1;
quit;
%let ct=%eval(&c0+&c1);


data _null_;
```

```
     call symput("n0",compress('(N='||put(&c0,4.) || ')'));
     call symput("n1",compress('(N='||put(&c1,4.) || ')'));
     call symput("nt",compress('(N='||put(&ct,4.) || ')'));
run;
%put second: &n0 &n1 &nt;


**** We need MISSING=' ' for this table, but save previous MISSING setting;
**** so that we can return SAS to that state after producing our table;
%let missingSetting = %sysfunc(getoption(MISSING));
options missing=' ';

ods rtf file="&odsdir./fancy2.rtf" bodytitle style=journal;
proc report
   data = forreport
   nowindows
   split = "|";

   columns group label col1 col0 col2 pvalue;

   define group   /order order = internal noprint;
   define label   /display " ";
   define col0    /display center "Placebo|&n0";
   define col1    /display center "Active|&n1";
   define col2    /display center "Overall|&nt";
   define pvalue  /display center " |P-value*"
                   f = pvalue6.4;

   break after group / skip;  /* does not work with ODS destinations */

   title1 "Table 1: Demographics";

   footnote1 "* P-values:  Age = Wilcoxon rank-sum, Smoking Status "
             "= Pearson's chi-square              ";

run;
ods rtf close;

options MISSING=&missingSetting;
```

References

Nice PROC REPORT introductions but write to output window rather than RTF:

Pages 136-147 of The Little SAS Book, fourth edition (Delwiche and Slaughter)

Chapter 15 of Learning SAS by Example (Cody)

Official SAS documentation – provides important details for serious PROC REPORT users

http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#n1dz7jdasx5t56n1rmlx346dyk6n.htm

Chapter 5 of SAS Programming in the Pharmaceutical Industry (Jack Shostak)

Cynthia Zender – Proc REPORT Tutorial (posted on Sakai)
http://www.wuss.org/proceedings10/TUT/3027_1_TUT-Zender.pdf

Frank DiIorio – A Tour of the SAS Reporting Toolbox (more general than PROC REPORT)
http://analytics.ncsu.edu/sesug/2005/IN03_05.PDF

Carter Sevick – “Table 1” in Scientific Manuscripts; Using PROC REPORT and the ODS System
http://www.lexjansen.com/wuss/2006/posters/POS-Sevick.pdf