Review of PROC SQL for use in SAS

Sadie Rhen
Joe Hurst
Daniel DeFoe

ABSTRACT

The following report discusses the uses and benefits of using PROC SQL in SAS code to help in managing and outputting data analysis. This investigation of this particular tool argues that PROC SQL has great potential to minimize code writing, save users from writing extra data steps, and simplify some otherwise complex SAS functions and maneuvers. There is explanation of basic functions in PROC SQL as well as prospective benefit to using them instead of other SAS methods.

INTRODUCTION

SQL is a coding language used for dataset queries and is used in conjunction with various programs and languages including SAS. In SAS, PROC SQL is a tool one can use to attain meaningful SAS output with what will often be less actual code than other SAS statements. The goal of SQL in SAS is often to combine the data and proc steps into a single proc step. PROC SQL has reasonably powerful ability to both manipulate data and logically output data with increased ease.

PROC SQL BASICS

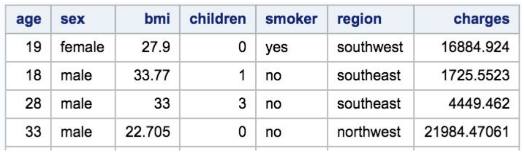
To accomplish its goal of combining the data and proc steps into one, PROC SQL must be able to manipulate data in a way similar to how a data step would. It can easily do this with a variety of commands very similar to data step functions. For PROC SQL to run, the data does not need to be sorted in any special way, but each observation should be on a row and each variable on a column. The basic syntax for PROC SQL is as follows:

```
PROC SQL;
       SELECT <column1>, <column2>, ...
       FROM <data_set1>, <data_set2>, ...
       WHERE condition
       GROUP BY <column1>, ...
       HAVING condition
       ORDER BY <column1>, ...
       ;
QUIT;
```

OUTPUTTING DATA

All data outputting is centered around the PROC SQL clauses “SELECT” and “FROM”, which are necessary for using this proc, all other clauses are optional. "SELECT *" tells PROC SQL to print every column in the data set. "FROM insurance" tells PROC SQL to use the data set named insurance. Setting OUTOBS=4 limits the output to the first four rows of the data set.

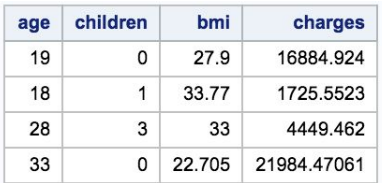
```
PROC SQL OUTOBS=4;
       SELECT *
       FROM insurance;
QUIT;
```



age	sex	bmi	children	smoker	region	charges
19	female	27.9	0	yes	southwest	16884.924
18	male	33.77	1	no	southeast	1725.5523
28	male	33	3	no	southeast	4449.462
33	male	22.705	0	no	northwest	21984.47061

If we only want to print certain columns, we replace the * after the SELECT statement with the columns we want to print, separated by commas.

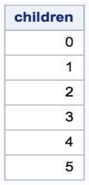
```
PROC SQL OUTOBS=4;
       SELECT age, children, bmi, charges
       FROM insurance;
Quit;
```



age	children	bmi	charges
19	0	27.9	16884.924
18	1	33.77	1725.5523
28	3	33	4449.462
33	0	22.705	21984.47061

Let's see all of the different values there are for the column children. We can do this with the DISTINCT statement. This removes all duplicate values. We see that people in this dataset have 0-5 children.

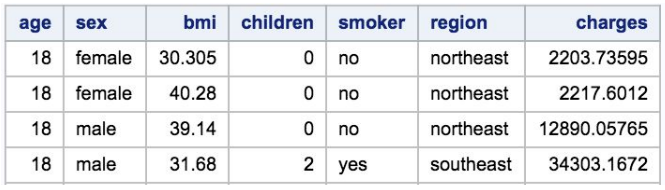
```
PROC SQL;
       SELECT DISTINCT(children)
       FROM insurance;
QUIT;
```



children
0
1
2
3
4
5

Say we want to sort our output. We can do this with the ORDER BY clause. To order by more than one column, column names should be separated by commas.

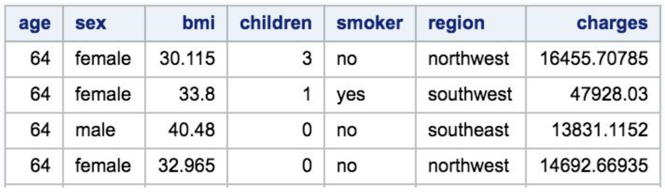
```
PROC SQL OUTOBS=4;
       SELECT *
       FROM insurance
       ORDER BY age;
QUIT;
```



age	sex	bmi	children	smoker	region	charges
18	female	30.305	0	no	northeast	2203.73595
18	female	40.28	0	no	northeast	2217.6012
18	male	39.14	0	no	northeast	12890.05765
18	male	31.68	2	yes	southeast	34303.1672

The default for ORDER BY is to sort in ascending order, but we can use the DESC statement to reverse the order.

```
PROC SQL OUTOBS=4;
       SELECT *
       FROM insurance
       ORDER BY age DESC;
QUIT;
```

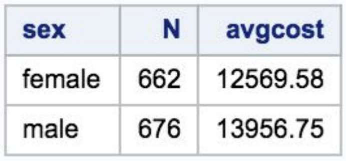


age	sex	bmi	children	smoker	region	charges
64	female	30.115	3	no	northwest	16455.70785
64	female	33.8	1	yes	southwest	47928.03
64	male	40.48	0	no	southeast	13831.1152
64	female	32.965	0	no	northwest	14692.66935

MANIPULATING DATA

PROC SQL is also able to create new columns. To see how difference columns in this data set compare for males and female, use the GROUP BY clause to combine the data into the factors of the column sex. Counting the number of observations in each group is done by applying the COUNT function to the column sex and assigning it to the new column N. Similarly, to find the average cost, take the mean of the charge column and assign that to the new column avgcost.

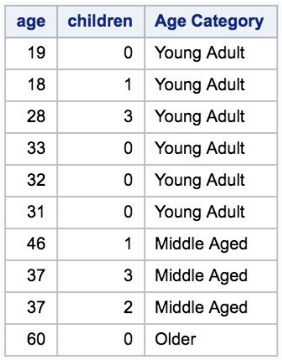
```
      PROC SQL;
             SELECT DISTINCT sex, COUNT(sex) AS N,
                    MEAN(charges) AS avgcost
             FROM insurance
             GROUP BY sex;
      QUIT;
```



sex	N	avgcost
female	662	12569.58
male	676	13956.75

We can also create a new categorical column from an existing column using case and when then statements.

```
PROC SQL OUTOBS=10;
       SELECT age, children,
            CASE WHEN age <  35 THEN "Young Adult"
                 WHEN 35 <= age < 55 THEN "Middle Aged"
                 ELSE "Older"
            END AS agecat LABEL = "Age Category"
        from insurance;
quit;
```



age	children	Age Category
19	0	Young Adult
18	1	Young Adult
28	3	Young Adult
33	0	Young Adult
32	0	Young Adult
31	0	Young Adult
46	1	Middle Aged
37	3	Middle Aged
37	2	Middle Aged
60	0	Older

PROC SQL EXAMPLE

Not only can PROC SQL be used to make quick summary charts with just a few steps, but these clauses can be used in conjunction with one another to create more complex tables and reports.

Say we have three data sets containing information on the salaries of White House staffers, one for each of the years 2015 through 2017. If we wanted to know if the Trump or Obama administration increased salaries more overall, then it would be helpful to combine these data sets into one larger set and create some new columns to answer our question.

To save data in a new table, we start with the CREATE TABLE clause. Then we select our columns that we want in the table; the name of the White House position, the average salary for each of the three years, the change in salary between 2015 and 2016 as well as between 2016 and 2017, the net difference between those salary changes, and a factor column that tells if the yearly salary increase was greater under the Obama or Trump administration. Each of the columns were assigned a label, which shows up as the column name in the output, and all of the columns that were monetary values were assigned a format so they will be outputted with a dollar sign, comma, and decimal. When creating new columns using columns that were also created in the data set, the word CALCULATED must be placed in front of column being referenced, otherwise SAS will try to locate that column in the original data set and will throw an error when the column is not found. Additionally, if columns have the same name in two different datasets they must be referenced as dataset.column in order for them to be referenced correctly.

```
PROC SQL;
      CREATE TABLE white_house_salaries AS
      SELECT obama15.title LABEL = "Position Title",
            AVG(obama15.salary) AS as15 LABEL="Average Salary 2015" FORMAT=DOLLAR14.2,
            AVG(obama16.salary) AS as16 LABEL="Average Salary 2016" FORMAT=DOLLAR14.2,
            AVG(trump.salary) AS as17 LABEL="Average Salary 2017" FORMAT=DOLLAR14.2,
            (calculated as16 - calculated as15) AS ochange
                  LABEL="Obama Salary Change (2016-2015)" FORMAT=DOLLAR14.2,
            (calculated as17 - calculated as16) AS tchange
                  LABEL="Trump Salary Change (2017-2016)" FORMAT=DOLLAR14.2,
            ABS(calculated tchange - calculated ochange) AS netdiff
                  LABEL="Net Difference" FORMAT=DOLLAR14.2,
            CASE WHEN (calculated tchange - calculated ochange) > 0 THEN "Trump"
                  ELSE "Obama" END AS netdifflabel LABEL = "Larger Salary Increase"
```

The FROM clause indicates that the data for the table is coming from three existing data sets, and that they are to be combined by position, leaving out any positions that were not in both administrations. The GROUP clause combines any replicate positions, so that each row is a position rather than an individual employee, and only the positions with a difference of less then $7,500 between how much the salary for that position changed under the Obama administration as compared to the Trump administration are kept, per the HAVING clause. Finally, the data set is ordered by the net difference. Now that the data table is saved for use, the final SELECT clause prints all of the columns in that new data table, the first few observations of which are shown below.

```
      FROM obama15, obama16, trump
            WHERE obama15.title = obama16.title AND obama16.title = trump.title
      GROUP BY obama15.title
      HAVING netdiff < 7500
      ORDER BY netdiff DESC;

      SELECT * FROM white_house_salaries;
QUIT;
```

Position Title	Average Salary 2015	Average Salary 2016	Average Salary 2017	Obama Salary Change (2016-2015)	Trump Salary Change (2017-2016)	Net Difference	Larger Salary Increase
SPECIAL ASSISTANT TO THE PRESIDENT FOR ECONOMIC POLICY	$120,000.00	$117,372.40	$122,142.86	$-2,627.60	$4,770.46	$7,398.06	Trump
EXECUTIVE CLERK	$143,079.00	$145,162.00	$153,730.00	$2,083.00	$8,568.00	$6,485.00	Trump
DIRECTOR OF RECORDS MANAGEMENT	$143,079.00	$145,162.00	$153,730.00	$2,083.00	$8,568.00	$6,485.00	Trump
LEGISLATIVE ASSISTANT	$45,270.00	$45,000.00	$51,000.00	$-270.00	$6,000.00	$6,270.00	Trump
STAFF ASSISTANT	$42,300.57	$42,504.31	$48,611.11	$203.74	$6,106.80	$5,903.07	Trump
LEGAL ASSISTANT	$45,450.00	$45,328.50	$51,000.00	$-121.50	$5,671.50	$5,793.00	Trump
DEPUTY ASSISTANT TO THE PRESIDENT AND DIRECTOR OF INTERGOVERNMENTAL AFFAIRS	$155,035.00	$157,299.00	$165,000.00	$2,264.00	$7,701.00	$5,437.00	Trump
ASSISTANT EXECUTIVE CLERK FOR MESSAGES AND EXECUTIVE ACTIONS	$110,902.00	$112,517.00	$119,489.00	$1,615.00	$6,972.00	$5,357.00	Trump
DEPUTY ASSOCIATE DIRECTOR	$55,550.00	$56,175.00	$62,000.00	$625.00	$5,825.00	$5,200.00	Trump
STENOGRAPHER	$57,570.00	$64,559.00	$66,500.00	$6,989.00	$1,941.00	$5,048.00	Obama

While this is only a small section of the data, it appears that generally, salaries increased more under the Trump administration as compared to during the Obama administration.

CONCLUSION

Many applications of PROC SQL commands have been showcased in the body of the report. The following tables will summarize the overall functions of what some of these various PROC SQL commands will do in a general sense.

PROC SQL Function Name	**Function Operation Description**
OUTOBS	Limits the number of observations
SELECT *	Selects all columns in the dataset
DISTINCT	Eliminates duplicate values
(Function) AS (String)	Will create a new column with label “String” as a product of the provided function
AVG	Takes the average of a selection of numeric data
COUNT	Counts number of observations within a column
CASE	Creates a factor column
TITLE	Creates a title for the output
LABEL	Will create a new label for a column column as a string
FORMAT	Allows user to apply a SAS format in line with statements that alter existing columns or create new ones
FROM	Selects the table from which the data is worked with
WHERE	Specifies which rows should be in the data set
GROUP BY	Organizes observations and primes for aggregation as what commands follow will apply to all observations grouped by the given columns
HAVING	Specifies which rows of data that have been grouped should be in the data set
ORDER BY	Changes the order in which the data will be output in

As exhibited in the report, PROC SQL does have significant power to serve the purpose of both the data and proc processing steps. It has also been shown that the PROC SQL can very effectively format and adjust output. Examples have shown that PROC SQL can subset and output data very few lines of code. It should be noted that many of the functions of the proc can be fulfilled by other SAS operations, however, one of the main advantages is that it can simplify code processes. This often results in overall less code being written, making it easier to work with, add on to, and interpret.

REFERENCES

Ronk, K.M. (2004). Introduction to Proc SQL. *Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference.* Cary, NC: SAS Institute Inc.

"The SQL Procedure." *Base SAS(R) 9.2 Procedures Guide*, SAS Institute, 25 May 2010, support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002294523.htm.

RECOMMENDED READING

- Official SAS PROC SQL Syntax Guide: https://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002294523.htm