A GUIDE TO PROC SQL

Pranav Bhagat, California Polytechnic State University, San Luis Obispo, CA

ABSTRACT

PROC SQL is a very beneficial base SAS procedure that is a pathway to run SQL code in SAS. SQL, or Structured Query Language, is a programming language which is most used for database management. Thus, it has much of the same functionality as the SAS, DATA, and PROC steps, all together. PROC SQL can be used to retrieve and manipulate data using SQL queries, by creating new variables, sorting, merging, or sub-setting them. Additionally, it can also be used to visualize data by printing results and creating tables. In this paper, I will go over the diverse optional statements, the syntax of PROC SQL, various built in functions, and will provide a couple of examples of PROC being utilized using different data sets.

INTRODUCTION

There are many reasons why we should learn about PROC SQL when working with SAS. PROC SQL is useful within SAS due to its ability to quickly and efficiently manipulate and display data in different ways. In many cases, PROC SQL will be able to perform the same tasks as multiple SAS PROC and DATA steps combined which will lead to significantly less computing time. Furthermore, PROC SQL can also be used to avoid complex SAS syntax when programming. If you are more familiar with SQL but are working in SAS, you can run the PROC within SAS code to make use of SQL syntax. This will increase efficiency and lead to less errors when running your code. In addition, learning how to use SQL in SAS will help when using other PROC functions in SAS due to the similarity in syntax and other programming languages which might be better suited for your task at hand.

PROC SQL SYNTAX

PROC SQL requires at least one SELECT statement and at least one FROM statement to build a query. The SELECT statement allows us to name columns in the order in which we would like them to appear in the report. It also lets us create entirely new variables, unlike the VAR function. The FROM statement is used to specify which data set the information will be taken from. The code below would pull the three variables from the named data set. Any amount of other optional SQL statements can follow this statement. This paper is restricted to working with the SELECT assertion of PROC SQL.

```
PROC SQL;
      SELECT price, make, model
      FROM cars;
QUIT;
```

SELECT STATEMENT SYNTAX

A SELECT statement must be written in the following order:

```
PROC SQL;
      SELECT column(s)
      FROM table(s) |
      view(s) WHERE
      expression
      GROUP BY
      column(s) HAVING
      expression
      ORDER BY column(s);
QUIT;
```

PROC SQL is what starts the SQL procedure. The SELECT keyword specifies which columns or variables are retrieved from the data. The FROM statement specifies the tables of datasets that the variables will be drawn from. The WHERE and HAVING statement subset the data based on a condition which is similar to how PROC PRINT works. The GROUP BY statement classifies the data into groups based on the specified columns, and the ORDER BY sorts the resulting rows (observations) by the specified columns. Lastly, the QUIT statement ends the procedure.

1. SYNTAX OF SIMPLE SELECT

In the SELECT statement, adding an asterisk will automatically select all columns from the data set. Similar to a PROC PRINT statement, this procedure displays the data without any modifications. A line will separate the column headings from the data and no observation numbers will appear.

```
PROC SQL; SELECT
      * FROM
      cars;
QUIT;
```

2. SYNTAX OF COMPLEX SELECT

The most basic use of PROC SQL usually has the SELECT and FROM clauses. However, we can also use the other six possible clauses for a more complex example.

```
PROC SQL;
      SELECT make, model, price
      FROM cars
      WHERE make =
      "Pontiac" GROUP BY
      model
      ORDER BY price;
QUIT;
```

3. LIMITING THE INFORMATION DISPLAYED

The SELECT statement can include any variables from the data set specified by the FROM statement. If we want to display specific variables on the report, we can list the column names separated by a comma in the SELECT statement. The NUMBER option prints a column in the report labeled as “ROW” which will contain the observation number. We can relate this to the KEEP statement in PROC PRINT. This code below will only print the make, model and trim variable from the data set.

```
PROC SQL NUMBER;
      SELECT make, model, trim
      FROM cars;
QUIT;
```

This will only limit the variables being displayed. To limit the number of observations that are displayed we specify the OUTBOS = X option in the PROC SQL line and only the first X observations will be displayed.

```
PROC SQL OUTOBS =
      10; SELECT *
      FROM cars;
QUIT;
```

CREATING NEW VARIABLES

In PROC SQL, it is possible to create new variables and fill them with values. You can either manually assign values or use the variables that were input from the data set to fill in the values for each observation. New variables can either be declared and given a name with the format ‘AS variable-name’, or be calculated without a label. The code below will save 7.5% of the value of each observation price as a new variable “tax.”

```
PROC SQL NUMBER;
      SELECT make, price, (price * 0.075) AS tax
      FROM cars;
QUIT;
```

1. CALCULATED OPTION

Within the SELECT statement, you are able to perform arithmetic and store those values as new variables. The CALCULATED statement is used to refer to a previously calculated variable earlier in the select statement. The code below will add the previously calculated tax value to the price and save it as a new total price variable.

```
PROC SQL NUMBER;
      SELECT make, price, (price * 0.075) AS
      tax, (price + CALCULATED tax) AS
      totalprice
      FROM cars;
QUIT;
```

2. **CASE EXPRESSION**

In SQL, case expressions can be used to create new variables based on categories of another. In the example below, we are filling in values of the new variable “pricerating” based on the values taken in from price from our data set. The different WHEN statements inside the overall CASE subsets the different possible values of price. The case statement is ended with an ‘END AS variable-name’ to declare the new variable. This can be related to multiple “if-then” statements in SAS DATA steps. This CASE assigned a value for price rating to each observation based on its price value.

```
PROC SQL;
      SELECT
        price, CASE
           WHEN price BETWEEN 0 and 10000 then 1
           WHEN price BETWEEN 10001 and 20000 then 2
           WHEN price BETWEEN 20001 and 30000 then 3
           WHEN price BETWEEN 30001 and 40000 then 4
           WHEN price BETWEEN 40001 and 50000 then 5
           ELSE 6
        END AS pricerating
      FROM cars;
QUIT;
```

3. **LABELS AND FORMATS**

Similar to other SAS functions for labeling and formatting, you are able to use formats and labels within SELECT statements to either format variables or label columns. This includes titles and footnotes for the final printed product.

```
TITLE "4th Quarter Sales
Data"; FOOTNOTE "SALES
DEPARTMENT"; PROC SQL;
      SELECT liter LABEL = "engine
             displacement", price FORMAT 8.2
      FROM cars;
QUIT;
```

MANIPULATING DATA

Manipulating data in SQL can be done quickly and be a very useful tool for a SAS user. It can either be done within one data set or ever between multiple data sets. Functions like GROUP BY, HAVING, ORDER BY, and WHERE can be used to subset and sort data by inclusion, exclusion, or groups. Also, data can also be manipulated across datasets through the use of JOINs and many more functions in SAS.

1. **GROUP BY STATEMENT**

   A GROUP BY statement can be used to group data based on the summary statistics for the groups and the value of one variable. The code below finds the average total metals for each country presented in the data set. Other grouping functions include SUM(), MAX(), MIN(), COUNT(), STD() and many others.

```
PROC SQL NUMBER;
      SELECT country,
      MEAN(total) FROM
      2012olympics
      GROUP BY country;
QUIT;
```

2. **HAVING STATEMENT**

   The HAVING statement can be used to further subset data that is already being grouped with a GROUP BY statement. Let's say we only wanted to know which countries had an average number of metals per medalist of over one, then we could use HAVING.

```
PROC SQL NUMBER;
      SELECT country, MEAN(total) AS average
      FROM 2012olympics
      GROUP BY country
      HAVING average > 1;
      QUIT;
```

3. **ORDER BY STATEMENT**

   The ORDER BY statement can be used to sort observations by the values of a specific variable. This can be compared to the PROC SORT in SAS. The order can be reversed by adding a DESC statement after the variable name, and the results can be sorted by multiple variables (in different directions), by adding more variables separated by commas. The code below will sort all of the Olympians from the data set by their total number of medals from highest to lowest, and then by name

```
PROC SQL
      NUMBER;
      SELECT *
      FROM 2012olympics
      ORDER BY total;
QUIT;
```

4. **WHERE STATEMENT**

   The WHERE statement can be used after specifying the data set that you are importing from, to subset the data on a condition. The code below only grabs the swimmers from the 2012 Olympics data set. This is similar to the WHERE statement in other SAS PROCs.

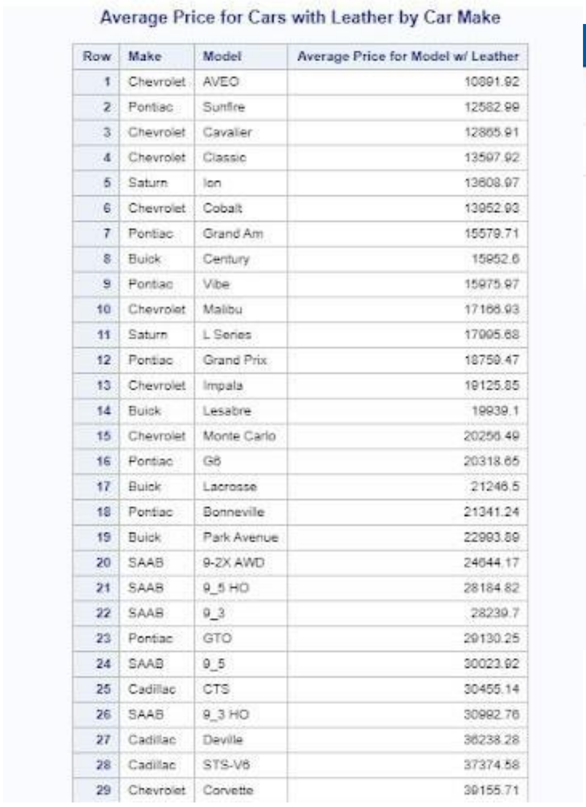
```
PROC SQL
      NUMBER;
      SELECT *
      FROM 2012olympics
      WHERE sport = "Swimming";
QUIT;
```

EXAMPLE 1

Detailing car prices:

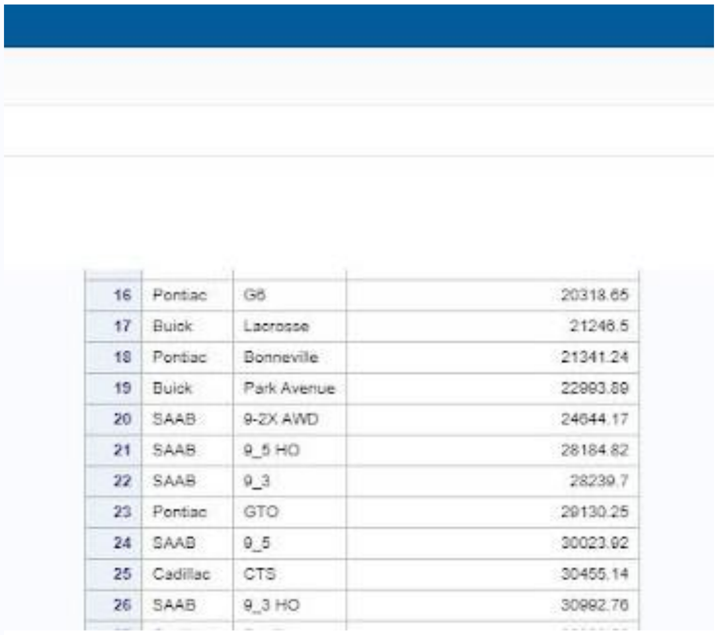
```
TITLE "Average Price for Cars with Leather by Car
Make"; PROC SQL NUMBER;
      SELECT make, model,
      MEAN(price) AS average LABEL = "Average Price for Model w/
      Leather" FROM flash.cars
      WHERE leather = 1
      GROUP BY make, model
      ORDER BY average;
QUIT;


TITLE "Average Price for Cars without Leather by Car
Make"; PROC SQL NUMBER;
      SELECT make, model,
      MEAN(price) AS average LABEL = "Average Price for Model w/o
      Leather" FROM flash.cars
      WHERE leather = 0
      GROUP BY make, model
      ORDER BY average;
QUIT;
```

PROC SQL is being used to compare the average price of every car model in the example above. Both of the tables are split up by the leather value in the data set, indicating whether or not the specific car sold had a leather in it or not. After splitting up the data with the WHERE statement, I used the GROUP BY statement to model and see which model corresponds to each average price value. Finally, we order by the price, since it is easier to visualize when not ordered by groups. When looking at each of the two tables, we are able to see which car models are sold with leather only, without leather only, or both. Also, we are allowed to see how the leather variable affects the average price of that model, if a model is in both tables.



Average Price for Cars with Leather by Car Make

Row	Make	Model	Average Price for Model w/ Leather
1	Chevrolet	AVEO	10891.92
2	Pontiac	Sunfire	12582.99
3	Chevrolet	Cavalier	12865.91
4	Chevrolet	Classic	13597.92
5	Saturn	Ion	13608.97
6	Chevrolet	Cobalt	13952.93
7	Pontiac	Grand Am	15579.71
8	Buick	Century	15952.6
9	Pontiac	Vibe	15975.97
10	Chevrolet	Malibu	17166.93
11	Saturn	L Series	17995.68
12	Pontiac	Grand Prix	18759.47
13	Chevrolet	Impala	19125.85
14	Buick	Lesabre	19939.1
15	Chevrolet	Monte Carlo	20256.49
16	Pontiac	G6	20318.65
17	Buick	Lacrosse	21246.5
18	Pontiac	Bonneville	21341.24
19	Buick	Park Avenue	22993.89
20	SAAB	9-2X AWD	24644.17
21	SAAB	9_5 HO	28184.82
22	SAAB	9_3	28239.7
23	Pontiac	GTO	29130.25
24	SAAB	9_5	30023.92
25	Cadillac	CTS	30455.14
26	SAAB	9_3 HO	30992.76
27	Cadillac	Deville	36238.28
28	Cadillac	STS-V6	37374.58
29	Chevrolet	Corvette	39155.71



16	Pontiac	G6	20318.65
17	Buick	Lacrosse	21246.5
18	Pontiac	Bonneville	21341.24
19	Buick	Park Avenue	22993.89
20	SAAB	9-2X AWD	24644.17
21	SAAB	9_5 HO	28184.82
22	SAAB	9_3	28239.7
23	Pontiac	GTO	29130.25
24	SAAB	9_5	30023.92
25	Cadillac	CTS	30455.14
26	SAAB	9_3 HO	30992.76

EXAMPLE 2

Investigating 2012 Olympic medalist data.

```
TITLE "2012 OLYMPICS FINAL
STANDINGS"; PROC SQL NUMBER;
      SELECT country,
         SUM(bronze) AS bronzescore LABEL = "Bronze Score",
         SUM(2*silver) AS silverscore LABEL = "Silver
         Score", SUM(3*gold) AS goldscore LABEL = "Gold
         Score", SUM(bronze + 2*silver + 3*gold) AS
         totalscore
            LABEL = "Total
      Score" FROM flash.o2012
      WHERE total ne
      . GROUP BY
      country
      ORDER BY totalscore DESC;
QUIT;
```

In this second example of PROC SQL, we are creating a scoring system for the Olympics and then ranking all the countries given in the dataset based on that designed system. As we acquire individual variables for the number of metals of each type, we multiply the number of medals by the multiplier for that medal (1 for bronze, 2 for silver, and 3 for gold). Afterwards, we would total those values for each observation and get each person's total score. In the process, the WHERE statement removes any observations that do not have any medals. Then we would group by the country and find the sum of the total score value for each country. The ORDER BY statement helps us display this data in from greatest to least so that the countries with the highest score are at the top and those with the lowest are at the bottom. The table below would show us the final standings for the 2012 Olympics using this scoring system. Also, by displaying each of the individual medal scores, we can see how the total is broken up for each country.

Row	Country	Bronze Score	Silver Score	Gold Score	Total Score
1	United States of America	20	38	120	178
2	People's Republic of China	13	30	75	118
3	Germany	8	22	63	93
4	Great Britain	20	26	33	79
5	France	11	22	33	66
6	Australia	8	34	21	63
7	Russian Federation	13	32	9	54
8	Republic of Korea	10	4	30	44
9	Canada	7	36	0	43
10	Italy	2	12	27	41
11	Japan	12	22	3	37
12	Netherlands	11	10	6	27
	...				
49	Qatar	1	0	0	1

CONCLUSION

PROC SQL is a useful and versatile tool in SAS which allows you to easily make use of a shorter syntax and allows for more efficient computing. The best part about PROC SQL is its ability to process long and complicated data and data-outputs in SAS code and narrow it to specific variables and data in which we are interested in. We could easily limit the data based on what we are looking for or what we need for certain projects/tasks. The basic syntax, the way we can create new variables/expressions, and the many ways we can manipulate data using the PROC SQL function allows us statisticians to be flexible when presenting our data. This paper is a mere glimpse of how useful the different parts of the PROC SQL function are, such as the SELECT statement. The amount of different ways to utilize it is a very valuable tool for many statisticians.

REFERENCES

The Ultimate Guide To Proc SQL. (n.d.). Retrieved from https://www.sascrunch.com/the-ultimate-guide-to-proc-sql.html

Minten Ronk, Katie. First, Steve. Beam, David. "An Introduction to PROC SQL." Pages 191-27. Madison, WI. Systems Seminar Consultants, Inc.

https://lexjansen.com/