Paper TS05-2011

Data Manipulation with SQL

Mara Werner, HHS/OIG, Chicago, IL

Abstract

SQL was developed to pull together information from several different data tables - use this to your advantage as you organize the results of your analysis. Using SQL, you can make your data more accessible to a new audience by explaining numerical data in words. Then, organize your results by creating a results summary table. As an added bonus, the results summary table acts as an index that can help you find sections of your SAS code later.

Introduction

This paper goes over two ways to make your analysis more approachable using SQL. The first uses the SQL CASE-WHEN statement to translate numerical data into words. This will make it easier for someone looking at your data for the first time to scan through it and find the message. The second uses the SQL UNION statement to make a master table with the results or all your analysis. This will help to organize your code and gives you easy access to your important results.

To demonstrate these techniques I'll use fictional data from site visits to Medicare providers. The data includes variables indicating what site reviewers found at each location – did they find the provider they were looking for or another business. If it was the correct provider, was the office open? How much did Medicare pay to providers that weren't open during the site visit?

Summarize Categorical Data with Words

Some data are easy to understand even without much background. Most people can scan a price list and understand the meaning. Other data, like categorical data, may be harder to scan and understand because the numbers are really standing in for something else. A common example is gender (male=1, female=0). In the site visit data, we have a variable called "found". If the site reviewers found the provider, then found=1, otherwise, found=0. Similarly, if the provider's office was open, then open=1, otherwise open=0. "Different Business" and "No Business" describe what was at the location if not the provider.

Source Table:

Location	Found	Open	Different Business	No Business
1	1	1	0	0
2	0	0	1	0

Without knowing the project or the data well, a table like this may be difficult to interpret. Having a data dictionary would help, but explaining the take away in words will make it much easier. The table below has a summary column that helps readers scan the data and absorb the important information quickly.

Location	Found	Open	Different Business	No Business	Summary
1	1	1	0	0	We found this provider. The office was open.
2	0	0	1	0	We found a different business at this location.

When site reviewers visited location 1 they found the provider there (found=1) and open (open=1). The columns for "different_business" and "no_business" aren't represented in the summary for this site visit because they are only used to describe a location where the provider was not found.

When site reviewers visited location 2, they did not find the provider (found=0). Therefore, it doesn't matter whether or not the office was open. It may be interesting to know what site reviewers did find at that address. In this case, they found a different business (different_business=1).

Creating Text with Conditional Logic

Creating text for the summary column is as easy as putting quotation marks around the text. For example, to create a new variable that is consistently the same string of text, put the text into quotation marks as a new variable in the select statement.

```
proc sql;
create table results as
select location, found, "We found this provider.  " as summary
from source_table
;quit;
```

Results:

Location	Found	Summary
1	1	We found this provider.
2	0	We found this provider.

However, we only found the first provider, not the second. To address this, we can use conditional logic. If found=1 then "we found this provider", otherwise "we did not find this provider". In SQL this is achieved with a CASE-WHEN statement.

```
proc sql;
create table results as
select location, found,
      case when found = 1 then "We found this provider.  "
            else "We did not find this provider.  "
      end as summary
from source_table
;quit;
```

Results:

Location	Found	Summary
1	1	We found this provider.
2	0	We did not find this provider.

Conditional Logic with Multiple Variables

Depending on the situation, you may need a more complicated logic to provide a meaningful summary. In the site visit example, if the provider wasn't found, there is information on what site reviewers did find at that location. Any time found=0, we can categorize what was there instead of the provider. When we found the provider, we can check whether the office was open.

To write this into code, we can string together multiple CASE-WHEN statements. The first one will generate text based on what site reviewers found at the location: the provider, another business, or no business. If site reviewers found the provider, we would like to say "we found this provider." If they didn't find the provider, but we did find a business, then "We found a different business at this location." Similarly, if there was no business at that location, the summary can say, "We did not find a business at this location."

```
proc sql;
create table results as
select *,
      case when found=1 then "We found this provider.  "
            when found=0 and different_business=1 then "We found a different
            business at this location.  "
            when found=0 and no_business=1 then "We did not find a business
            at this location.  "
            else ''
      end as Summary
from source_table
;quit;
```

Results:

Location	Found	Open	Different Business	No Business	Summary
1	1	1	0	0	We found this provider.
2	0	0	1	0	We found a different business at this location.

So, you have generated text that either indicates that we found the place or describes what we found instead – those are all mutually exclusive categories. Now we'll build on that. If found=1 we still want the text "We found this provider. " but in addition, we'd like to say whether or not the office was open. If the place was not found, it doesn't matter whether the office was open or not, so we insert blank text instead of words. In SAS, you can generate blank text by using a set of quotation marks with a space between them (" ").

To string together multiple statements, you can use the train tracks symbol (||). Keep in mind, if you are stringing together multiple chunks of text, it becomes helpful to put a period and spaces after each sentence.

```
proc sql;
create table results as
select *,
      case when found=1 then "We found this provider.  "
            when found=0 and different_business = 1 then "We found a
            different business at this location.  "
            when found=0 and no_business=1 then "We did not find a business
            at this location.  "
            else '' end ||
      case when found=1 and open=1 then "The office was open.  "
            when found=1 and open=0 then "The office wasn't open.  "
            else '' end
            as Summary
from source_table
;quit
```

Results:

Location	Found	Open	Different Business	No business	Summary
1	1	1	0	0	We found this provider. The office was open.
2	0	0	1	0	We found a different business at this location.

Organizing Summary Analysis

Another way that SQL can help you to make your analysis more approachable to others and to your future self is to create a data table combining all the important numbers into a results summary table. For the site visits, we might have the total number of providers that were not at the correct location and the total number of providers that were

not open. We might also break out the total that were not found into those where there was another business at the location and those were there was no business at the location.

Results Summary:

Number	Description
10	Total not found
50	Total not open
8	Different Business
2	No Business

To create the results summary table, the first step is to write code for your analysis. I use Proc SQL rather than some of the other procedures in base SAS to calculate my results. This allows me to have results in the form of a data tables rather than text in the output window. I can then use SQL to query those results and include them in the results summary table. The following code calculates the number of providers that were not found (N1) and then the number that were not open (N2).

```
proc sql;
create table N1 as
select count(*) as num_not_found
from site_visits
where found=0
;quit;

proc sql;
create table N2 as
select count(*) as num_not_open
from site_visits
where found=1 and open=0
;quit;
```

This produces the following two tables:

N1

Num_not_found
10

N2

Num_not_open
50

Two tables aren't so hard to keep track of, but when it gets to be 25 or 50 of them, you may find yourself (as I often do) scrolling through pages of code trying to find the place where you created some particular number so you can check it, update it, or use it. You can save yourself this grief by creating a summary results table. Then, you will have everything you need in one place. The following two options show different ways to achieve the same goal – collecting your important results into a single data table and producing descriptive labels for each number.

Option 1: Proc SQL + Proc Transpose

Once you have your summary analysis, you can use a combination of Proc SQL and Proc Transpose to create the results summary table ("Allresults"). For this to work, each of the source tables should be a summary – only one row per table.

```
proc sql;
create table allresults as
select a.num_not_found, b.num_not_open
from n1 as a, n2 as b
;quit;
```

allresults

Num_not_found	Num_not_open
10	50

And then transpose the allresults table to make it easier to read.

```
proc transpose data=allresults
               out=allresults2
               name=Description
               prefix=Number;
run;
```

allresults2

Description	Number1
Num_not_found	10
Num_not_open	50

Option 2: Proc SQL – UNION statement

You could also create the allresults table at the same time as you write your analysis. This method will use the Proc SQL UNION statement to join together your summary analysis. Instead of using a descriptive variable name, call each new result something generic, like "number" and then include the description in a separate variable. Create each new select statement with the same variables in the same order.

In the code below, the results (the count or summary, etc.) are in the first variable. The second variable is text describing the result ("Total not found").

```
proc sql;
create table N3 as
select count(*) as number,
      'Total not found' as description
from site_visits
where found=0
union
select count(*) as number,
      'Total not open' as description
from site_visits
where found=1 and open=0
;quit;
```

N3

Number	Description
10	Total not found
50	Total not open

To make this more useful, you can add another column that groups the data in some meaningful way. For example, you could add another column indicating your source data or to group the numbers into the different sections of a report.

Formats in the Results Summary

You may have results that make more sense when they are formatted. For example, dollar signs, percent signs, and commas for large numbers all can make your data easier to read. In SAS, all the rows in a column must have the same format. To utilize multiple formats, put the data into different columns. For example, the number of providers that were found can be in one column formatted as a number and the amount that Medicare paid to these facilities can be in another column formatted as currency. In the following sections, I'll give two options for using formats in the results summary table.

Format Results: option 1

The first is to create one table for numbers (N3), another for dollars (N4), and then join them together using a SET statement (allresults).

```
proc sql;
create table N4 as
select avg(dollars2010) as dollars,
      'Average dollars- not open' as description
from site_visits
where found=1 and open=0
;quit;
```

N4

Dollars	Description
$75,000	Average dollars- not open

```
data allresults;
format number 10. dollars dollar15.
length number 10. descr $50.;
set N3 N4;
run;
```

Allresults:

Number	Dollars	Description
10		Total not found
50		Total open
	$75,000	Average dollars- not open

Format Results: option 2

The second is to carry both columns the whole way down. Since “number” and “dollar” are both numerical columns, instead of using quotation marks to generate a blank, use a period.

```
proc sql;
create table allresults2 as
select count(*) as number,
      . as dollar,
      'Total not found' as description
from site_visits
where found=0
union
select count(*) as number,
      . as dollar,
      'Total not open' as description
from site_visits
where found=1 and open=0
union
select . as number,
      avg(dollars2010) as dollars,
      'Average dollars- not open' as description
from site_visits
where found=1 and open=0
;quit;
```

Allresults2:

Number	Dollars	Description
10		Total not found
50		Total door Locked
	$75,000	Average dollars- not open

Benefits of Creating an All-Results table

Creating a table like the all-results tables in the example above can be coding-intensive. But, in my experience, the time up front pays off later. You'll have all of the important results of your analysis together in one place. If you need to alter any of the code or update your numbers with new data, you can find it easily using the text from the "description" column. When you search for that text in your code, it will take your right to the place where that number was created.

Conclusion

Data analysis is often for the purpose of discovery, but also for sharing a message. Using SQL and SAS, you can make the message easier to understand by creating summary text and you can make your summary results easier to find and the code easier to understand by organizing your results into an all-results summary table.

Contact Information

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

Mara Werner
Department of Health and Human Services,
Office of Inspector General
233 N. Michigan ave, suite 1390
Chicago, IL 60601
312.353.9872
mara.werner@oig.hhs.gov
www.oig.hhs.gov