Statistical summary functions

Suman Sahil, Steve Simon

Creation date: 2019-09-24

Statistical summary functions

- SQL can produce a handful of summary statistics.
- Common to all versions of SQL
  - count, avg, sum, min, max
- Standard deviation function is not consistent
  - stddev, stdev, stdevp, stdev_samp
  - Not available in SQLite

It will not replace your favorite statistical package, but sometimes calculating a few of these statistics in SQL can greatly simplify your life.

The count, average, sum, min, and max functions are identical across all versions of SQL, but the standard deviation function is not. It is just not available at all in SQLite. Depending on which version of SQL you are using, you might use stddev with two "d"'s or stdev with only one "d" and you might have two different versions of the standard deviation in the same version of SQL. One would use n-1 in the denominator and the other would use n in the denominator.

I would recommend that you avoid the use of a standard deviation calculation in SQL, if you can.

Min, max, and avg functions

– SQL code

```
select max(age), min(age)
  from  fat
```

– SQL output

```
  max(age) min(age)
1       81       22
```

The min and max functions behave just as you'd suspect.

Be sure to rename your variables

- SQL code

```
select
  max(age) as oldest,
  min(age) as youngest
  from  fat
```

- SQL output

```
  oldest youngest
1     81       22
```

The default names that SQL gives to your statistical functions include parentheses. If you know anything about The min, max, and avg functions behave just as you'd suspect.

Computing a range

– SQL code

```
select
  max(age)-min(age) as age_range
  from  fat
```

– SQL output

```
  age_range
1        59
```

You can combine these functions with a subtraction operator to get a range.

Computing an average

– SQL code

```
select avg(age) as average_age
  from  fat
```

– SQL output

```
  average_age
1    44.88492
```

The average also behaves as you'd suspect. Notice that SQL produces five decimal places for the average age. If you are using the average in some intermediate calculations, then keeping all this precision is a good idea. But if your only reason for computing the average is for displaying it's value in a table, then you should round your results.

Rounding the average

– SQL code

```
select round(avg(age),1) as average_age
  from  fat
```

– SQL output

```
  average_age
1        44.9
```

Here is how you would round the average. Note that in most settings, I would prefer to round an average age to the nearest year. No one really cares if the average age is 44.9 versus 45. Even half a year is irrelevant unless you are talking about children. In geneal, I would encourage you to round aggressively.

When should you use these statistical functions

- All these functions are available in SAS and R
- Use statistical functions in SQL if you want
  - to save time and space
  - to simplify your SAS or R code

All of these functions, of course, are available in SAS or R. An important question if why you would use SQL to calculate these statistics instead?

The biggest reason is time and space. SQL excels at managing massive amounts of data. It uses an efficent storage model and an efficient retrieval model. That's not to say that the storage and retrieval models in SAS and R are substandard. It's just that SQL has been designed from the ground up to handle massive amounts of data. It does a lot of things behind the scenes to speed things up.

There may be advantages from the hardware end as well. When you run a database query, you send a request from your client computer, which may be a humble desktop or laptop computer to a power server computer with multiple processors and massive storage systems that run in parallel.

There are exceptions, but any data management work that you can do in SQL will usually lead to savings in time.

Storage is also a consideration. All the examples that you've seen in this class involve just a few dozen or maybe a few hundred records, but in the real world, your database may contain millions or even billions of records. This amount of data would not fit on your local

computer, and even if it did, you would have trouble downloading it through even a high speed Internet connection. It would be better to ask for your summary statistics that the thousands or millions of records needed to produce those summary statistics.

You might prefer to have SQL do the summarizations if it simplifies the SAS or R code. In general, SQL code is easier for a non-programmer to read than SAS or R code. So using SQL might make your code a bit easier to folow.

When should you use these statistical functions (2/2)

- Use statistical functions in SAS or R if you need
  - to store both the summary statistics AND the raw data
  - to account for variation
  - sophisticated summary statistics

In some applications, you will need SAS or R to have access both the summary statistics and the raw data. This might be for plotting purposes, for example, where you would show both the summary staistics and the individual data points. If you need to account for variation, especially variation across multiple levels of a hierarchy, then you shoud do the summarization in SAS or R. It also goes without saying that if you need a summary statistic that is even just a tiny bit more complicated than a standard deviation, then you need to use SAS or R.

Count funtion

– SQL code

```
select count(case_number) as n_rows
  from fat
```

– SQL output

```
  n_rows
1    252
```

The COUNT function is, in my opinion, one of the most useful summary functions. It allows you to quickly identify the number of records in your query. You can use COUNT on any field in your database and you should get the same answer, with one exception, which you will see in the section on NULL values.

The name chosen by SQL for the result of COUNT is not a legal R name,so you would normally use the AS statement to create a better name.

Counting distinct values

– SQL code

```
select
  count(age) as n_ages,
  count(distinct age) as n_distinct_ages
  from fat
```

– SQL output

```
  n_ages n_distinct_ages
1    252              51
```

You can also count the number of distinct values. You might want to do this with a continuous variable like age to see how many plotting positions you would need.

Use where to count subset

– SQL code

```
select count(case_number) as subset_count
  from fat
  where age >= 65
```

– SQL output

```
  subset_count
1           21
```

Where also computes other statistics on subset

– SQL code

```
select
    round(avg(age)) as avg_senior_age,
    round(avg(bmi)) as avg_senior_bmi
  from fat
  where age >= 65
```

– SQL output

```
  avg_senior_age avg_senior_bmi
1             69             27
```

You can get other summary statistics on a subset. This SQL code produces the average age and average fat percentagle for the 21 senior citizens

Switch to titanic data set

– SQL code

```
select PClass, count(Name) as n
  from titanic_table
  where PClass='1st'
```

– SQL output

```
  PClass   n
1    1st 322
```

This is how you get a count of first class passengers on the Titanic.

Use group by to count all subsets

– SQL code

```
select PClass, count(Name) as n
  from titanic_table
  group by PClass
```

– SQL output

```
  PClass   n
1    1st 322
2    2nd 280
3    3rd 711
```

If you wanted a count for every passenger class, use the group by keyword.

Multiple categories

– SQL code

```
select Sex, PClass, count(Name) as n
  from titanic_table
  group by Sex, PClass
```

– SQL output

```
     Sex PClass   n
1 female    1st 143
2 female    2nd 107
3 female    3rd 212
4   male    1st 179
5   male    2nd 173
6   male    3rd 499
```

You can put two or more categorical variables in the GROUP BY statement.

The having keyword (1/4)

– SQL code

```
select PClass, min(Age) as youngest_female
  from titanic_table
  where Sex='female'
  group by PClass
  having youngest_female < 1
```

– SQL output

```
  PClass youngest_female
1    3rd            0.17
```

If you want to filter on a value of the summary statistic, you need to use the having keyword. This code selects those ages with only one passenger. You cannot use the where keyword because where is run BEFORE the data is summarized. You use the having keyword to filter AFTER the data is summarized.

This code selects only female passengers and it does this before calculating the summary statistic. So the minimum age that you get is computed from the ages of only female passengers. We does this for the three passenger classes. Then we only keep the results if the youngest female passenger in that age group is less than a year old. You can see that only third class has a baby girl under a year old.

The having keyword (2/4)

– SQL code

```
select Age, count(Age) as n
  from titanic_table
  group by Age
  having n >= 30
  order by n desc
```

– SQL output

```
  Age  n
1  22 35
2  21 31
3  30 31
4  18 30
```

Here's a second example. There are 1,313 passengers on the Titanic, so many of them will have the same age. Which ages occur at least 30 times? That's what this query tells you.

There's one problem here. The most common value for age is null, which occurs 756 times. Why didn't this show up when you asked for ages with counts of 30 or more? Let's investigate.

The having keyword (3/4)

– SQL code

```
select Age, count(Age) as n
  from titanic_table
  group by Age
  having Age is null
```

– SQL output

```
   Age n
1  NA 0
```

There are 756 passengers with an unknown age. So why do we get a count of zero?

The count function, like all other statistical functions ignores any missing values. So it refuses to count any of the 756 null values associated with age is null. There's an easy fix here, if you want to count missing values.

The having keyword (4/4)

– SQL code

```
select Age, count(*) as n
  from titanic_table
  group by Age
  having n >= 30
  order by n desc
```

– SQL output

```
  Age   n
1  NA 557
2  22  35
3  21  31
4  30  31
5  18  30
```

If you do a count on a variable that does not have missing values like Name, or if you do a generic count (count(*)) then you will see that the null values for age are counted properly.

This is proof that missing values are the bane of every statistician's existence. If you ever get weird and unexpected results, consider undertaking a hunt of null values and see if that may explain what is going on.

Summary

- Statistical functions
  - count, min, max, avg, sum
- group by summarizes for each category
- The having keyword filters your data after the summarization
- The count and other statistical functions ignore null values

This lecture talked about the statistical functions: count, min, max, avg, and sum. These produce a single row summary. If you include the group by keyword, you get a row for every category. The having keyword filters your data after summarization. The count function and all the other statistical functions ignore missing values.

Homework (1/2)

- For your homework, use the titanic database.
  - This is available in Oracle using schema='simons'.
  - In SQLite, it is a standalone file named titanic_db.sqlite.
- count the number of passengers with the title "Mr" somewhere in their name.
- Run a query that counts the number of male and female children (Age <= 18)
- Run a query that identifies the ages of the youngest and oldest patients in each passenger class.

Here is your homework assignment.

Homework (2/2)

- The Survived field has values of 0 (died) and 1 (alive) and an average of this value provides a probability of survival.
  - Calculate this probability for six categories representing the combination of passenger class and sex.
  - Include only those groups where the survival probability is greater than 30%.
- Place the SQL code and the results of all your queries in a single PDF file.

One more thing for you to do.