SQL Tricks For Data Scientists — Checking Data Quality

Robert de Graaf Follow
Oct 9, 2018 · 3 min read

All data scientists know some SQL, but it can be used for a lot more than pulling data into the 'real' analysis environment.

In some ways SQL is the forgotten secret of data science — taken for granted as an necessary but slightly uncool means of getting data out of the databases it often inhabits, with none of the cache of Pandas or the tidyverse.

Fair enough, too, in some ways, as the range of functions offered within most implementations of SQL has tended to fall short of the needs of someone doing data preparation beyond the need to join tables together and apply filters to slim down the amount of data to be transferred to the environment where the analysis proper will be performed — usually over in R or Python.

Books on SQL compound the problem, as it is hard to find books that go very far beyond the core range of straightforward SELECT statements and joins, possibly leavened with some aggregate functions.

Yet many of use SQL regularly because the data we use lives in a SQL compliant database, and if we want to do something with it, we have to need to write a query, and if we're going to write a query, we might as well do it right.

Similarly, given that for many analyses, the first step is to move a bunch of data into R or Python, we might as do that right — at least, we might as well try to move the smallest, but most useful, data table we can. Hence, an analysis that establishes how useful each column is, and which highlights missing and extreme values that might lead to that row or column being excluded from the final model can be useful.

Establishing the proportions of missing values in particular columns seems like it ought to be available out of the box. It is not, but can still be achieved with a minimum of fuss, albeit with a couple of tricks.

. . .

```
SELECT CAST

(SUM (CASE WHEN column1 is NULL THEN 1 ELSE 0 END)

as float) / COUNT(*) AS ProportionMissing
```

```
FROM YourDB.YourTable
```

Effectively what we are doing is implementing the equivalent of Excel's SUMIF via a CASE statement. Obviously we need to CAST to float because SUM returns an integer, and if we forgot to CAST the query would return 0 in almost all cases.

The point of this query is that if the degree of missingness in any particular column means that the column is useless, it's a waste of time moving the column into your modeling environment.

Another basic data quality check is looking for extremes. Obviously in SQL there are the MAX() and MIN() functions, but it can also be useful to go a little further and examine multiple extreme variables. A reasonably commong way of detecting extreme values is to look for values that are an excessive number of standard deviations from the mean — 4 standard deviations is used as the benchmark below.

```
WITH STATS (Col1_AVG,Col1_SD) AS
    (SELECT STDEV(Col1),AVG(Col1)
    FROM Db1.Tbl1)

 SELECT Col1,DWT_AVG,DWT FROM STATS JOIN Tbl1
 ON 1=1
 WHERE ABS(Col1-Col1_AVG)/Col1_SD > 4
```

We use the common table expression (the section beginning on keyword 'WITH') because we are going to compare the results of aggregate functions to the values they are based on. The version above returns the extreme values themselves so they can be studied for reasonableness (when data was collected by hand, it was not uncommon for extreme values to derive from transcription errors), but reversing the greater than sign obviously simply prunes the values.

Robert de Graaf's most recent prior article for Medium was Explainability: The Last Mile.

He is also the author of the forthcoming book by Apress, Managing Your Data Science Projects

Follow him on Twitter: https://twitter.com/RobertdeGraaf2