Chapter 5

Aggregations

Contents

1 Introduction to MTA data set

- In this section we are going to introduce another data set, the NY MTA dataset, which contains information on the number of cars that pass certain plazas between January 1st, 2010 and January 7th, 2017, about 7 years of data.
- Looking at the dataset, we see that it is a long (or tall) dataset, with 6 columns. The data represents the number of cars that go through different toll plazas in the city by hour. The data is divided between cars which paid via EZ-pass and cash and split between those drivers heading away from the city ("O" direction) and those heading into the city ("I").

```
select * from cls.mta limit 10;


  plaza  mtadt         hr  direction      vehiclesez    vehiclescash
-------  ----------  ----  -----------  ------------  --------------
      2  2013-10-14    16  I                    2469             336
      2  2013-10-14    16  O                    2393             473
      2  2013-10-14    17  I                    2853             425
      2  2013-10-14    17  O                    2116             417
      2  2013-10-14    18  I                    2575             394
[...]
```

- For example, if we want to see the number of cards which are heading outbound between 2 and 3 am on the 15th of June, 2015 over the Robert F. Kennedy Bridge Manhattan Plaza (Triborough bridge into Manhattan, which is Plaza #2), can be found by writing the following query:

```
select
    *
from
    cls.mta
where
    plaza = 2
    and direction = 'I'
    and hr = 2
    and mtadt = '2015-06-15';


  plaza  mtadt         hr  direction      vehiclesez    vehiclescash
-------  ----------  ----  -----------  ------------  --------------
      2  2015-06-15     2  I                     173              58
```

- A unique row in this dataset is denoted by the items in the WHERE clause – plaza, mtadt, hr and direction.
- If we wanted look at the total number of cars, for each hour, that go through Plaza #2 we could do the following:

```
select
    plaza, mtadt, hr
    , vehiclesez + vehiclescash as totalCars
from
    cls.mta
where
    plaza = 2
    and direction = 'I'
order by mtadt, hr;


  plaza  mtadt         hr    totalcars
-------  ----------  ----  -----------
      2  2010-01-01     0          747
      2  2010-01-01     1          903
      2  2010-01-01     2          742
      2  2010-01-01     3          501
      2  2010-01-01     4          456
[...]
```

2 GROUP BY clause

- Up until this point we have been slicing data, removing rows and columns. The next syntax we will study aggregates, or collapses, data into a smaller number of rows. In other words, this operation now looks between rows in order to undertake its calculation. Importantly, this operation defines subsegments of the table that are treated as a single group.

- Consider the following query:

```
select
     MAX( vehiclescash) as maxcash
    , plaza
from
    cls.mta
group by plaza;


  maxcash    plaza
---------  -------
     1352        1
     1040        2
     1594        3
     1368        4
      674        5
[...]
```

  GROUP BY to combines similar values. This query combines data by plaza and returns the maximum number of cars that pay cash in any hour through that plaza.

- This query will return 10 rows, one for each plaza. The query calculates the the maximum value of vehiclescash for by plaza.

- The GROUP BY clause is applied and written *after* the WHERE clause. If a WHERE clause removes a row then that row will not be aggregated via the function.

```
select
     MAX( vehiclescash) as maxcash
    , plaza
from
    cls.mta
where
    plaza = 2
group by plaza;


  maxcash    plaza
---------  -------
     1040        2
```

- GROUP BY requires every column within the SELECT clause to be either inside a function or part of the GROUP BY. The following query yields an error:

```
select
     MAX( vehiclescash) as maxcash
     , hr
    , plaza
from
    cls.mta
group by plaza;

ERROR: column "cls.hr" must appear in the GROUP BY
clause or be used in an aggregate function
```

- Other aggregate functions include average ("AVG"), minimum ("MIN"), count ("COUNT") and sum ("SUM"):

```
select
    plaza
    , min( vehiclescash) as minveh
    , count( vehiclescash) as ctveh
    , sum( vehiclescash) as sumveh
    , avg( vehiclescash) as avgveh
from
    cls.mta
group by plaza
order by avg(vehiclescash) desc;


  plaza    minveh    ctveh    sumveh    avgveh
-------  --------  -------  --------  --------
     11         0    61488  38181458   620.958
      3         0   122976  67000523   544.826
      1         0   122976  54359482   442.033
      9         0   122976  53530379   435.291
      2         0   122976  38009405   309.08
[...]
```

- Implicit GROUP BY: If every column within a select statement is an aggregate function then the query will still run, even if it does not have GROUP BY put down explicitly:

```
select
    sum( vehiclescash ) as sumveh
    , avg( vehiclescash ) as avgveh
from
    cls.mta;


   sumveh    avgveh
---------  --------
330901032   283.858
```

In this case, the entire table is treated as a single group within the GROUP BY.

- There is also a special aggregation: COUNT(DISTINCT XXX), which returns the number of unique values within a given group:

```
select
    count(distinct plaza) as plazact
from
    cls.mta;


  plazact
---------
       10
```

Note that COUNT DISTINCT counts the number of unique non-null entries.

- We can include multiple columns within the GROUP BY and it will calculate the functions among unique combinations of the columns selected. For example:

```
select
    plaza
    , mtadt
    , sum(vehiclescash + vehiclesez) as totalcars
from
    cls.mta
group by plaza, mtadt
order by plaza, mtadt;


  plaza  mtadt         totalcars
-------  ----------  -----------
      1  2010-01-01        57606
      1  2010-01-02        63405
      1  2010-01-03        59496
      1  2010-01-04        72610
      1  2010-01-05        72880
[...]
```

- What if I forget to include an AS?

```
select
    count(vehiclesez)
    , max(vehiclesez)
    , max(vehiclescash)
from
    cls.mta;


  count    max    max
-------  -----  -----
1165728   8345   2116
```

Without the AS, the database returns the column with the name of the aggregate function.

- Before continuing, lets answer some simple questions about the table. What percentage of cars which pass through a toll plaza during this time period use an EZ-pass?

```
select
    sum(vehiclesez)::float / (sum(vehiclesez) + sum( vehiclescash)) as pct_EZ
from
    cls.mta;


  pct_ez
--------
0.817743
```

We can see that it is around 80%.

- COUNT AND SUM can be used to return the number of rows within the table. Looking at the queries in Table 5.1 you can see that placing a number within a count returns the number of rows. Note that the second query will return the number of rows because it counts the number of '1's that appear. It is not counting the number of rows in the first column – it is counting the number of rows that would appear if every value within that column was equal to 1. Consider the following variants on this in the following table:

Syntax	What is returned
`select count(*)`	Number of rows
`select count(1)`	Number of rows
`select 2*count(*)`	Twice the number of rows
`select 2*count(2)`	Twice the number of rows
`select 2*count(-1)`	Twice the number of rows
`select 2*count(Null)`	Zero
`select 2*sum(1)`	Twice the number of rows
`select 2*sum(2)`	Four times the number of rows

Table 5.1: Examples of special syntax for counting rows

- GROUP BY treats NULL as a special, unique value. If there are NULL values in the column being grouped, they will be treated as a single group.
- Null values within aggregate functions are not straightforward. Consider the following table ("null_test") which has a two columns ("val" and "cond"), as can be seen below:

```
select * from cls.null_test;


  val  cond
-----  ------
    1  A
    2  A
    3  A
       B
```

- SUM, MAX, MIN, COUNT and AVG all ignore Null values:

```
select
    sum(val) as st
    , max(val) as mt
    , min(val) as mnt
    , avg(val) as at
    , count(val) as ct
    , count(distinct val) as cd
from
    cls.null_test;


  st    mt    mnt    at    ct    cd
----  ----  -----  ----  ----  ----
   6     3      1     2     3     3
```

Note that this is different then when using ORDER BY, which treats Null values as larger than any other value. Note that AVG(X) is equivalent to SUM(X) / COUNT(X). With COUNT(val), the Null is ignored. However with count(*) the Null is not ignored!

```
select count(*) as ct, count(val) as ct2 from cls.null_test;


  ct    ct2
----  -----
   4      3
```

- If the entire column is Null within a group, then each of AVG, MAX, MIN, SUM will return Null and COUNT will return zero:

```
select
    cond
    , sum(val) as st
    , max(val) as mt
    , min(val) as mnt
    , avg(val) as at
    , count(val) as ct
    , count(distinct val) as cd
from
    cls.null_test
group by cond;


cond      st    mt    mnt    at    ct    cd
------  ----  ----  -----  ----  ----  ----
A          6     3      1     2     3     3
B                                   0     0
```

3 Column numbering syntax

- As with ORDER BY we can use column numbering syntax:

```
select
    plaza
    , min( vehiclescash) as minveh
    , count( vehiclescash) as ctveh
    , sum( vehiclescash) as sumveh
    , avg( vehiclescash) as avgveh
from
    cls.mta
group by 1;


  plaza    minveh    ctveh    sumveh    avgveh
-------  --------  -------  --------  --------
      1         0   122976  54359482  442.033
      2         0   122976  38009405  309.08
      3         0   122976  67000523  544.826
      4         0   120624  21397862  177.393
      5         0   122976   7798630   63.4159
[...]
```

In the query above the number 1 in the GROUP BY clause denotes the first column in the select statement. In this case, that is "plaza"

- We can add multiple columns when using column numbering syntax. For example:

```
select
    plaza
    , mtadt
    , min( vehiclescash) as minveh
    , count( vehiclescash) as ctveh
    , sum( vehiclescash) as sumveh
    , avg( vehiclescash) as avgveh
from
    cls.mta
group by 1,2;


  plaza  mtadt         minveh    ctveh    sumveh    avgveh
-------  ----------  --------  -------  --------  --------
      1  2010-01-01       249       48     28166   586.792
      1  2010-01-02       186       48     28583   595.479
      1  2010-01-03       261       48     27272   568.167
      1  2010-01-04       143       48     26210   546.042
      1  2010-01-05       103       48     25218   525.375
[...]
```

In this query, the data is grouped by two columns: plaza and mtadt. The grouping columns are specified as "1,2".

4 Aggregates and CASE Statements

- Aggregates and CASE statements can be combined in powerful ways. Let's first count the number of rows in the database where the hour is 2 and the number of vehicles paying cash is greater than 400. As demonstrated by the query below we can use a WHERE clause to only include the rows in the table which fulfill this criteria.

```
select
    sum(1) as ct
from
    cls.mta
where
    hr = 2
    and vehiclescash > 400;


  ct
----
 256
```

- Let's say that we also wish to get the number of rows in the database where the the number of vehicles paying cash is less than or equal to 5 and the hour is 2. Because we are cutting up the data into two mutually exclusive ways we need to do something other than a WHERE clause. If we remove the rows to satisfy the first condition then we remove rows that would need to be counted in the second condition.

  We can implement both criteria using a CASE statement inside an aggregate function:

```
select
    sum( case
       when hr = 2 and vehiclescash > 400 then 1
       else 0 end) as ct1
    , sum( case
       when hr = 2 and vehiclescash < 5 then 1
       else 0 end) as ct2
from
    cls.mta;


  ct1    ct2
-----  -----
  256   1465
```

- We could also use the COUNT notation, rather than a SUM, by switching the zeros to Null:

```
select
    count( case
       when hr = 2 and vehiclescash > 400 then 1
       else Null end) as ct1
    , count( case
       when hr = 2 and vehiclescash < 5 then 1
       else Null end) as ct2
from
    cls.mta;


  ct1    ct2
-----  -----
  256   1465
```

- We can group by any column expression, including a CASE statement. In the following example we use a CASE statement to categorize different rows and then use a GROUP BY statement in order count how many of each occurs.

```
SELECT
    CASE
        WHEN vehiclescash > 400 then 'More than 400'
        WHEN vehiclescash >= 5 then 'Between 5 and 400'
        ELSE 'Less than 5'
    END as breakdown_flag
    , count(1)
    , avg( vehiclescash ) as avgCash
FROM
    cls.mta
group by 1;


breakdown_flag       count    avgcash
-----------------  -------  ---------
Between 5 and 400   824717  153.548
Less than 5          10778    1.22323
More than 400       330233  618.516
```

This creates categories of data, based on vehiclescash and then returns how many rows are in each category.

- We can define a column by almost anything and then group by it. In the following example, we look at the difference between the vehicles which pay cash and which pay by EZ pass. If the difference is sufficiently large we categorize it one way and if not, another, but we remove zero's first!

```
select
    case
        when vehiclescash = 0 then 'Zero Cash'
        when abs( vehiclescash - vehiclesez)::float
            / vehiclescash < .05 then 'less than'
        else 'more'
    end
    , count(1) as ct
from
    cls.mta
group by 1;


case            ct
---------  -------
Zero Cash     6512
less than     2765
more       1156451
```

In the case above this returns 3 rows and 3 columns since we are aggregating on a column which can take one of three values. Aggregating on case statements is an incredibly powerful way to calculate statistics on

5 Named Subqueries

- Let's calculate the number of cars that go through each plaza each day in the Inbound direction using cash:

```
select
    sum( vehiclescash ) as totalcash
    , mtadt
    , plaza
from
    cls.mta
where
    direction = 'I'
group by mtadt, plaza;


  totalcash  mtadt         plaza
-----------  ----------  -------
      14783  2010-01-01        1
       8965  2010-01-01        2
      17309  2010-01-01        3
       3840  2010-01-01        4
       1454  2010-01-01        5
[...]
```

- Now, let's try to calculate how many cars go through the average plaza on the average day in the inbound direction using cash. In other words, we want to take the average of the above. In this case we can try to do the following:

```
select
    avg( sum( vehiclescash) )
from
    cls.mta
where
    direction = 'I';

ERROR: aggregate function calls cannot be nested
LINE 2: avg( sum( vehiclescash) )
```

Unfortunately we can't nest aggregation functions. To answer the question above we need to use a subquery since we need to do an aggregation *on* another aggregation.

```
select
    avg( sumcash) as avgcars
from
    (select
        sum(vehiclescash) as sumcash
        , mtadt
        , plaza
    from
        cls.mta
    where direction = 'I'
    group by mtadt, plaza
    ) as innerQ;


  avgcars
---------
  7394.15
```

To understand this query, lets start by breaking it apart and focusing on the inner query first:

```
select
    sum(vehiclescash) as sumcash
    , mtadt
    , plaza
from
    cls.mta
where direction = 'I'
group by mtadt, plaza;


  sumcash  mtadt         plaza
---------  ----------  -------
    14783  2010-01-01        1
     8965  2010-01-01        2
    17309  2010-01-01        3
     3840  2010-01-01        4
     1454  2010-01-01        5
[...]
```

The result of this inner query is a table itself wth three columns and a row for each mtadt-plaza combination. The column sumcash represents the number of cars, in total, which went through that plaza-mtadt combination using cash – which is the just the number that we want to average!

Using this table we can then take an average on it, which we do in an outer query. Importantly, when we nest queries in this fashion we have to give then a name, which we do in this case with the AS clause. As a note, just like when naming a column the AS itself is optional, though recommended.

- Lets look at another example: What percentage of the day-plaza combinations in our dataset have an inbound-to-outbound ratio of less than 90% for cash transactions? In other words, what percentage of plazas, on a given day, have more outbound traffic than inbound traffic by 10%?

  Just as before we will need to compute multiple levels of aggregation. Lets work from the inside out – first computing the number of inbound and outbound cars for each plaza-mtadt combination.

```
select
    sum( case when direction = 'I'
        then vehiclescash else 0 end ) as InboundCash
    , sum( case when direction = 'O'
        then vehiclescash else 0 end ) as OutboundCash
from
    cls.mta
group by
    plaza, mtadt


  inboundcash    outboundcash
-------------  --------------
        14783           13383
        14680           13903
        14049           13223
        13202           13008
        12688           12530
[...]
```

Note that we are grouping by columns which we are not selecting – which is allowable under most SQL variants. Since we don't need to know which row is associated with each plaza or mtadt, only the totals, we will not select it. Once we have this data we can then do the aggregation that we are interested in:

```
select
    sum( case when InboundCash <= .90 * OutboundCash
        then 1 else 0 end)::float
            / count(1) as pct
from
    (select
        sum( case when direction = 'I'
            then vehiclescash else 0 end ) as InboundCash
        , sum( case when direction = 'O'
            then vehiclescash else 0 end ) as OutboundCash
    from
        cls.mta
    group by
        plaza, mtadt) as innerQ;


      pct
---------
0.0303516
```

- In the cases above we were required to use a subquery because we wanted to do two levels of aggregation, which is a common problem. For example, let's say that we wanted to find the average number of rows per plaza, for rows which have more than 700 EZ pass cars. In this case we first need to do two levels of aggregation – first calculating the number of rows, per plaza, which fulfill the criteria and then averaging over the plaza – as can be seen below:

```
select
    avg( numrows) as avgrows
from
    (select
        count(1) as numrows
        , plaza
    from
        cls.mta
    where
        vehiclesEZ >= 700
    group by 2) as innerQ;


  avgrows
---------
    69457
```

In the case above the inner query only has 10 rows, one for each plaza while the outer query only returns the average.