Hadoop - NYC Parking Analysis

Submitted to :
Prof Peter Zadrozny

10/23/14

Submitted By
TEAM#1
Akshay Baheti
Dhruv Mevada
Manoj Gyanani

Table of Contents

1. Introduction

The project deals with analyzing New york parking violations with an aim to derive patterns that can help new yorker avoid a parking ticket.

Our team will be analyzing the NYC Parking, based on the data from the 3 different datasets. The first dataset describes NYC Parking Facilities. The second describes parking violations and the last dataset describes speed camera violations. All of these datasets have multiple attributes and we will be analyzing various patterns by following the data. The login credentials for the cluster are listed below, and the username and password is the same for all nodes.

User: dhruv
Password: mangotown166*

master: 216.121.73.39
slave1: 216.121.73.35
slave2: 216.121.73.36
slave3: 216.121.73.37
slave4: 216.121.73.38

2. Dataset Summary

The dataset consists of 3 CSV files.

1. NYC Parking Facilities: This dataset describes the parking facilities in NYC by giving details about capacity and locations of parking spots.
2. Parking Violations: This dataset gives details about parking violation tickets issued in NYC. It contains information such as area code, zip code, vehicle information, and other data. This will act as the main data set for all analysis.
3. Speed Camera Violations: This dataset gives details about speed camera violation tickets issued in NYC. This data set may be used to drive a relation in the parking and speed camera violations. The primary analysis will be on the 2nd data set.

2.1 File Format

The file format of datasets that we are using for analysis is CSV format.

2.2 Data Objects and Types

Each dataset below describes the type of fields that are included.

1. NYC Parking Facilities:

NYC Parking Facilities contains basic information about parking facilities. The fields are as follows

{

'Facility type': It defines the type of the facility, and whether it's a parking garage or a combination of parking garage and parking lot.
'**License Number':** It gives the license number of the parking facility.
'Entity Name': It gives the name of the parking facility.
'Trade Name': It gives the trade name under which the parking facility operates.
'Address': It gives the address of the parking facility.
'Address zip code': It gives the zip code of the parking facility.
'Telephone Number': It gives telephone number of the parking facility.
'Number of Spaces': It gives the number of spaces associated with the parking facility.
}

2. Parking Violations:

Parking Violations contains information about parking violations in NYC. The fields are as follows
{
'Summons Number': It gives the summons number issued for the parking violation.
'**Plate ID':** It gives plate id of the vehicle.
'Registration State': It gives the registration state of the vehicle.
'Issue Date': It gives the date on which the summon is issued.
'Violation Code': It gives the violation code.
'Vehicle Body Type': It gives the body type of the vehicle.
'Vehicle Make': It gives the vehicle maker name.
}

3. Speed Camera Violations:

Speed Camera Violations contains information about Speed Camera Violations summons issued. The fields are as follows
{
'Summons Number: It gives the summons number issued for speed camera violation.
'Plate ID': It gives plate id of vehicle.
'Registration State': It gives the registration state of the vehicle.
'Plate Type': It gives the type of plate.
'Issue Date': It gives the issue date of summon.
'Violation Code': It gives the violation code.
'Vehicle Body Type': It gives the body type of vehicle.
'Vehicle Make': It gives the name of vehicle maker.
'Issuing Agency': It gives the name of ticket issuing agency
'Street Code1': It gives the street code1 of location where speed camera violation is observed.
'Street Code2': It gives the street code2 of location where speed camera violation is observed.
'Street Code3': It gives the street code3 of location where speed camera violation is observed.
'Vehicle Expiration Date': It gives the expiration date of the vehicle.
'Violation Location': It gives the location where speed camera violation is observed.
'Violation Precinct': It gives the violation precinct.
'Issuer Precinct': It gives the issuer precinct.
'Issuer Code': It gives the issuer code.

'Issuer Command': It gives the issuer command.
'Issuer Squad': It gives the issuer squad.
'Violation Time': It gives the speed camera violation time.
'Time First Observed': It gives the time when the violation is first observed.
'Violation County': It gives the county where the violation is observed.
'Violation In Front Of Or Opposite': It gives whether violation is in front of or opposite of given house number.
'House Number': It gives the house number.
'Street Name': It gives the street name.
'Intersecting Street': It gives the intersecting street.
'Date First Observed': It gives the date when violation is first observed.
'Law Section': It gives the law section.
'Sub Division': It gives the sub division.
'Violation Legal Code': It gives the violation legal code.
'Days Parking In Effect': It gives the days for parking in effect.
'From Hours In Effect': It gives from hours in effect.
'To Hours In Effect': It gives to hours in effect.
'Vehicle Color': It gives the color of vehicle.
'Unregistered Vehicle': It gives the information on whether the vehicle is unregistered.
'Vehicle Year': It gives the year of vehicle
'Meter Number': It gives the number of the meter
'Feet From Curb': It gives the feet from curb.
'Violation Post Code': It gives the violation post code.
'Violation Description': It gives the violation description.
'No Standing or Stopping Violation': It gives whether or not it was a standing or stopping violation
'Hydrant Violation': It gives whether the violation occurred next to a hydrant
'Double Parking Violation': It gives whether the car was parked in two spaces
}

3. Acquiring the Data

All the NYC Parking data is obtained from the website **nycopendata.socrata.com**. Since the aim is to analyze the parking violations in New York we have considered 3 information tables. We simply downloaded the files from the websites.

1. Parking Violations - This table consists of all the Parking violation tickets issued in New York city. The area code, zip code vehicle details, etc.
2. Parking Facilities - The Parking facilities in New York which gives us the details of the capacity and location of the parking facilities
3. Speed Camera tickets - The table has all the information of the speed camera violations issued in NYC.

The three tables can be obtained from csv files available on nycopendata.socrata.com

4. Setting up the environment

As we are using csv files no real environment setup is required. We only need a couple of formatting changes in the csv files, such as removing line breaks from the address filed in Parking facilities file.

We also had to change the field delimiter in csv file from comma to the UNIX'|' pipe symbol. This was done because the address file already had commas in it. The commas in the address field prevented us from smoothly importing the csv file into the hive table.

4.1 Converting date

We wrote a shell script to convert the date from MM-DD-YYY to YYY-MM-DD

Input to Script - 7564315647,FDW3729,NY,12/31/2013,21,SUBN,FORD
Output of Script - 7564315647,FDW3729,NY,2013-12-3121,SUBN,FORD

covert_date.sh

```
#!/bin/sh


while read line
do
   new_date=`echo $line |  awk -F "," '{print $4}'   | awk -F "/" '{print $3 "-" $1 "-" $2}'`
   temp1=`echo $line | awk -F "," '{print $1 ","$2 "," $3 ","}'`
   temp2=`echo $line | awk -F "," '{print "," $5 ","$6 "," $7 }'`
   new_line=`echo -n "$temp1$new_date$temp2\n"`
   echo "$new_line" >> new.csv
done < Parking_Violations_Since_2010.csv
```

5. Creating Tables for Loading the Data

5.1 Creating Tables

• Type "hive" into the command line to start hive. The queries are the ones that are used to create the Hive tables for each dataset.

Parking Facilities table Query table query:

create table parking_facilities(facility_type STRING,lncs_numer STRING,entity_name STRING,trade_name STRING,address STRING,address_zip_code STRING,
            telephone_num STRING,number_of_spaces STRING) row format delimited fields terminated by '|' stored as textfile;

Parking violations table query:

```
create table parking_violations(summons_no STRING, plate_id STRING, reg_state STRING,
issue_date DATE, violation_code STRING, vehicle_body_type STRING, vehicle_make
STRING) row format delimited fields terminated by ‘,’ stored as textfile;
```

Speed camera violations query:

```
create table speed_cam_tickets(summons_number STRING, plate_id STRING, registration_state
STRING, plate_type STRING, issue_date DATE, violation_code STRING, vehicle_body_type
STRING, vehicle_make STRING, issuing_agency STRING, street_code_1 STRING,
street_code_2 STRING, street_code_3 STRING, vehicle_expiration_date DATE,
violation_location STRING, violation_precinct STRING, issuer_precinct STRING, issuer_code
STRING, issuer_command STRING, issuer_squad STRING, violation_time STRING,
time_first_observed STRING, violation_county STRING, violation_in_front_of_or_opposite
STRING, house_number STRING, street_name STRING, interesecting_street STRING,
date_first_observed DATE, law_section STRING, sub_division STRING, violation_legal_code
STRING, day_parking_in_effect STRING, from_hours_in_effect STRING, to_hours_in_effect
STRING, vehicle_color STRING, unregistered_vehicle STRING, vehicle_year STRING,
meter_number STRING, feet_from_curb STRING, violation_post_code STRING,
violation_description_string STRING, no_standing_or_stopping_violation STRING,
hydrant_violation STRING, double_parking_violation STRING ) row format delimited fields
terminated by ',' stored as textfile;
```

5.2 Loading Data into the Tables

```
LOAD DATA INPATH “/user/project_data/NYC_Parking_Facilities.csv” OVERWRITE INTO
TABLE parking_facilities;
```

```
LOAD DATA INPATH “/user/project_data/new.csv” OVERWRITE INTO TABLE
parking_violations;
```

```
LOAD DATA INPATH “/user/project_data/speed_cam_violations.csv” OVERWRITE INTO
TABLE sped_cam_violations;
```

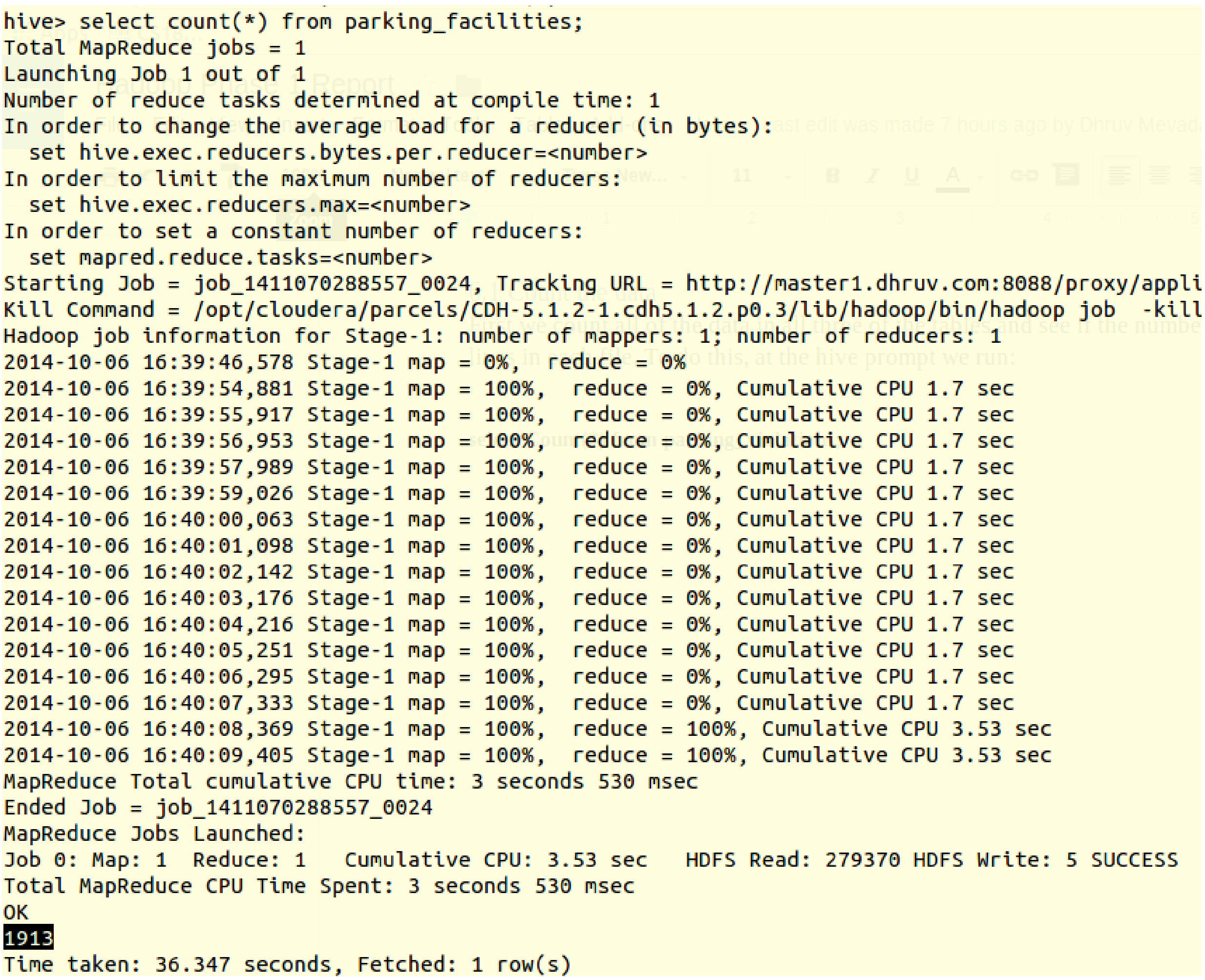
6. Loaded Data Verification

There are a couple of ways that we can verify the data by running queries on the newly entered data.

6.1 Count the data

First we count all of the data in all three of the tables and see if the numbers match the amount of lines in each file. To do this, at the hive prompt we run:

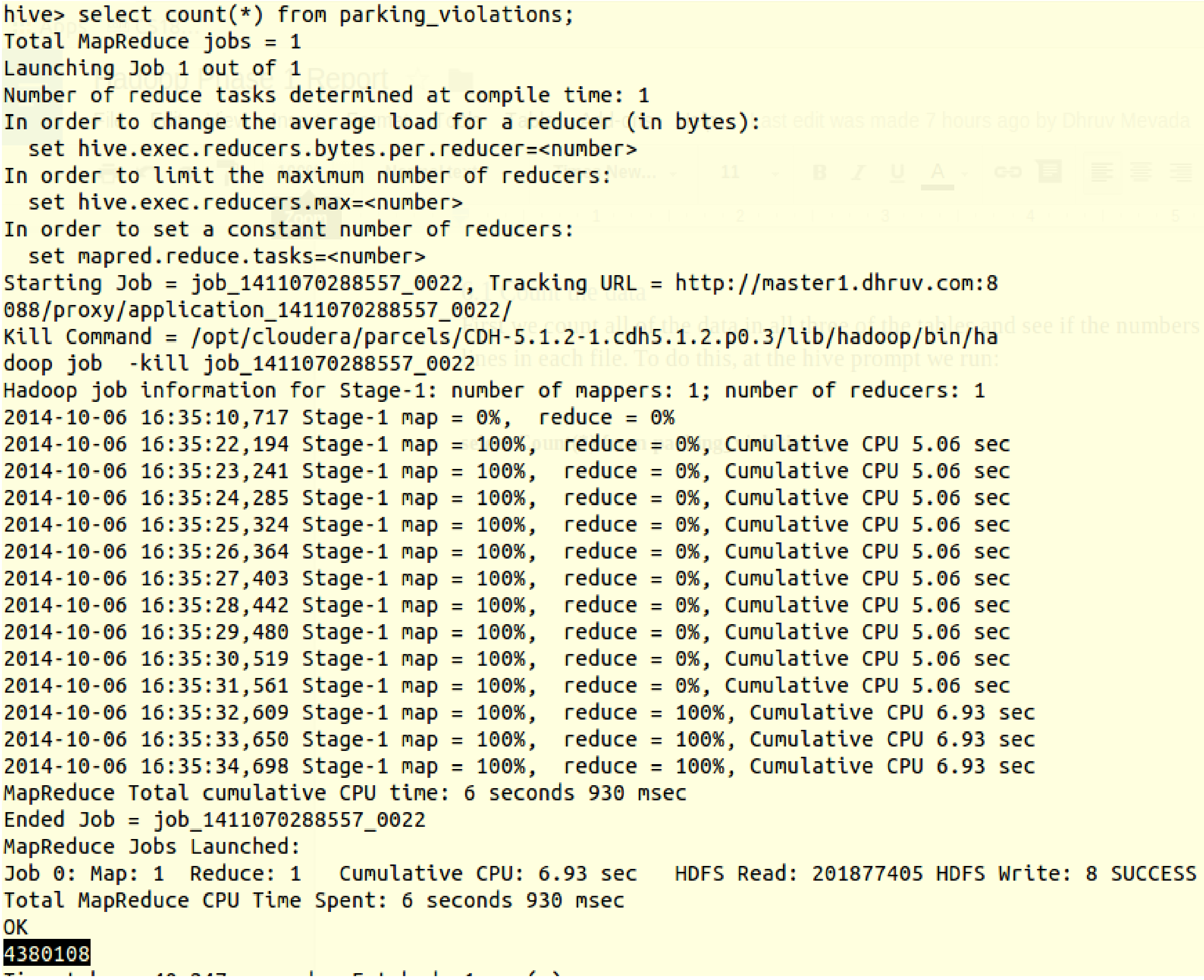
select Count(*) from parking_violations;



```
hive> select count(*) from parking_facilities;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1411070288557_0024, Tracking URL = http://master1.dhruv.com:8088/proxy/appli
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/hadoop job  -kill
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2014-10-06 16:39:46,578 Stage-1 map = 0%,  reduce = 0%
2014-10-06 16:39:54,881 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:39:55,917 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:39:56,953 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:39:57,989 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:39:59,026 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:00,063 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:01,098 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:02,142 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:03,176 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:04,216 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:05,251 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:06,295 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:07,333 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2014-10-06 16:40:08,369 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.53 sec
2014-10-06 16:40:09,405 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.53 sec
MapReduce Total cumulative CPU time: 3 seconds 530 msec
Ended Job = job_1411070288557_0024
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 3.53 sec   HDFS Read: 279370 HDFS Write: 5 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 530 msec
OK
1913
Time taken: 36.347 seconds, Fetched: 1 row(s)
```

This is the count of all the rows in excel.

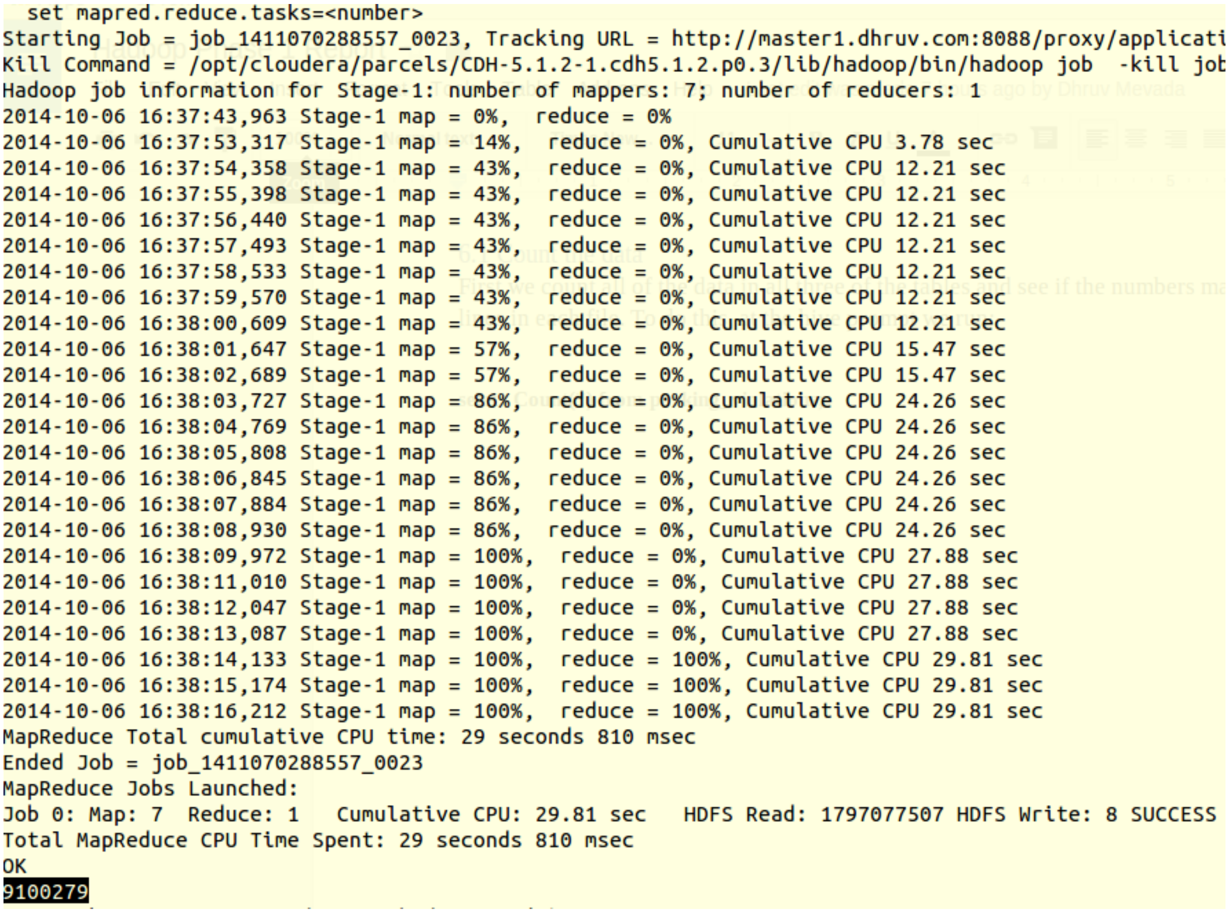
2. For parking violations table:
select Count(*) from parking_violations;



```
hive> select count(*) from parking_violations;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1411070288557_0022, Tracking URL = http://master1.dhruv.com:8
088/proxy/application_1411070288557_0022/
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/ha
doop job  -kill job_1411070288557_0022
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2014-10-06 16:35:10,717 Stage-1 map = 0%,  reduce = 0%
2014-10-06 16:35:22,194 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:23,241 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:24,285 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:25,324 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:26,364 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:27,403 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:28,442 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:29,480 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:30,519 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:31,561 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.06 sec
2014-10-06 16:35:32,609 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.93 sec
2014-10-06 16:35:33,650 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.93 sec
2014-10-06 16:35:34,698 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.93 sec
MapReduce Total cumulative CPU time: 6 seconds 930 msec
Ended Job = job_1411070288557_0022
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 6.93 sec   HDFS Read: 201877405 HDFS Write: 8 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 930 msec
OK
4380108
```

This count cannot be verified in excel as excel cannot load so many rows. So we just had to assume that all the data loaded correctly.

3. For speed camera tickets table
select Count(*) from speed_cam_tickets;



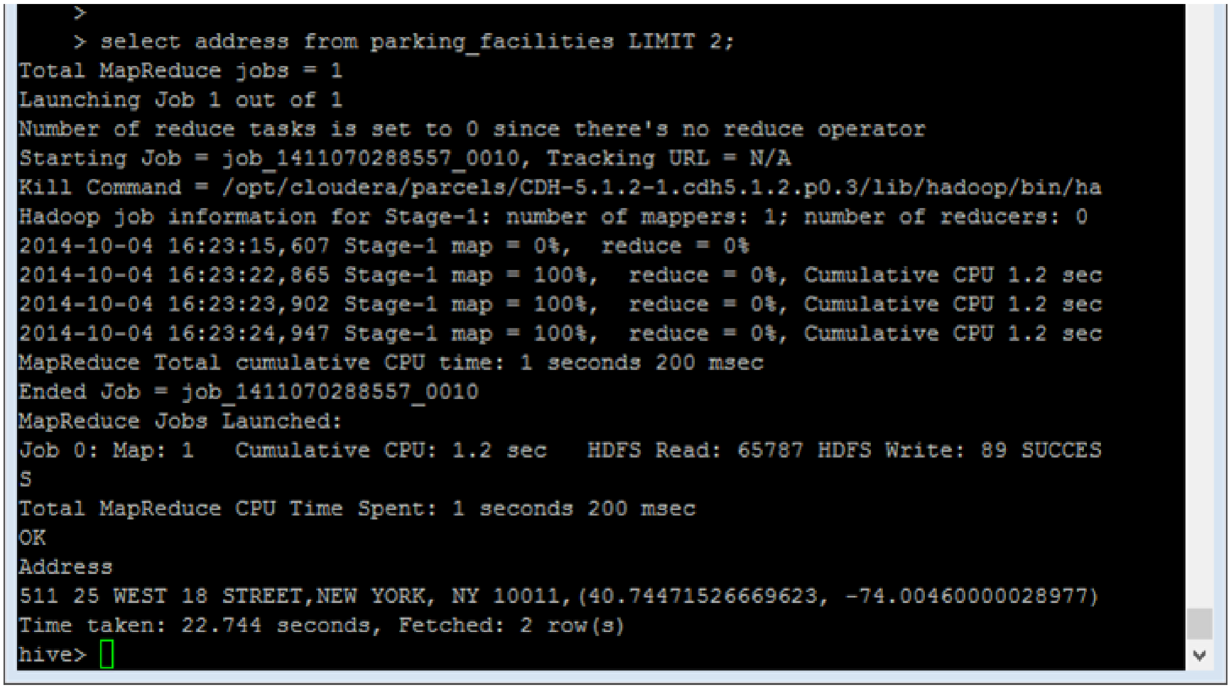
```
  set mapred.reduce.tasks=<number>
Starting Job = job_1411070288557_0023, Tracking URL = http://master1.dhruv.com:8088/proxy/applicati
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/hadoop job  -kill job
Hadoop job information for Stage-1: number of mappers: 7; number of reducers: 1
2014-10-06 16:37:43,963 Stage-1 map = 0%,  reduce = 0%
2014-10-06 16:37:53,317 Stage-1 map = 14%,  reduce = 0%, Cumulative CPU 3.78 sec
2014-10-06 16:37:54,358 Stage-1 map = 43%,  reduce = 0%, Cumulative CPU 12.21 sec
2014-10-06 16:37:55,398 Stage-1 map = 43%,  reduce = 0%, Cumulative CPU 12.21 sec
2014-10-06 16:37:56,440 Stage-1 map = 43%,  reduce = 0%, Cumulative CPU 12.21 sec
2014-10-06 16:37:57,493 Stage-1 map = 43%,  reduce = 0%, Cumulative CPU 12.21 sec
2014-10-06 16:37:58,533 Stage-1 map = 43%,  reduce = 0%, Cumulative CPU 12.21 sec
2014-10-06 16:37:59,570 Stage-1 map = 43%,  reduce = 0%, Cumulative CPU 12.21 sec
2014-10-06 16:38:00,609 Stage-1 map = 43%,  reduce = 0%, Cumulative CPU 12.21 sec
2014-10-06 16:38:01,647 Stage-1 map = 57%,  reduce = 0%, Cumulative CPU 15.47 sec
2014-10-06 16:38:02,689 Stage-1 map = 57%,  reduce = 0%, Cumulative CPU 15.47 sec
2014-10-06 16:38:03,727 Stage-1 map = 86%,  reduce = 0%, Cumulative CPU 24.26 sec
2014-10-06 16:38:04,769 Stage-1 map = 86%,  reduce = 0%, Cumulative CPU 24.26 sec
2014-10-06 16:38:05,808 Stage-1 map = 86%,  reduce = 0%, Cumulative CPU 24.26 sec
2014-10-06 16:38:06,845 Stage-1 map = 86%,  reduce = 0%, Cumulative CPU 24.26 sec
2014-10-06 16:38:07,884 Stage-1 map = 86%,  reduce = 0%, Cumulative CPU 24.26 sec
2014-10-06 16:38:08,930 Stage-1 map = 86%,  reduce = 0%, Cumulative CPU 24.26 sec
2014-10-06 16:38:09,972 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 27.88 sec
2014-10-06 16:38:11,010 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 27.88 sec
2014-10-06 16:38:12,047 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 27.88 sec
2014-10-06 16:38:13,087 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 27.88 sec
2014-10-06 16:38:14,133 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 29.81 sec
2014-10-06 16:38:15,174 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 29.81 sec
2014-10-06 16:38:16,212 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 29.81 sec
MapReduce Total cumulative CPU time: 29 seconds 810 msec
Ended Job = job_1411070288557_0023
MapReduce Jobs Launched:
Job 0: Map: 7  Reduce: 1   Cumulative CPU: 29.81 sec   HDFS Read: 1797077507 HDFS Write: 8 SUCCESS
Total MapReduce CPU Time Spent: 29 seconds 810 msec
OK
9100279
```

This count cannot be verified in excel as excel cannot load so many rows. So we just had to assume that all the data loaded correctly.

6.2 Sample data

To sample the data, you can find the first few records, last few records, and random middle records. If the data that is retrieved from the query matches the data in the csv file, then the data is correct. The first table is parking_facilities, the second is parking_violations, and the third is speed camera violations.

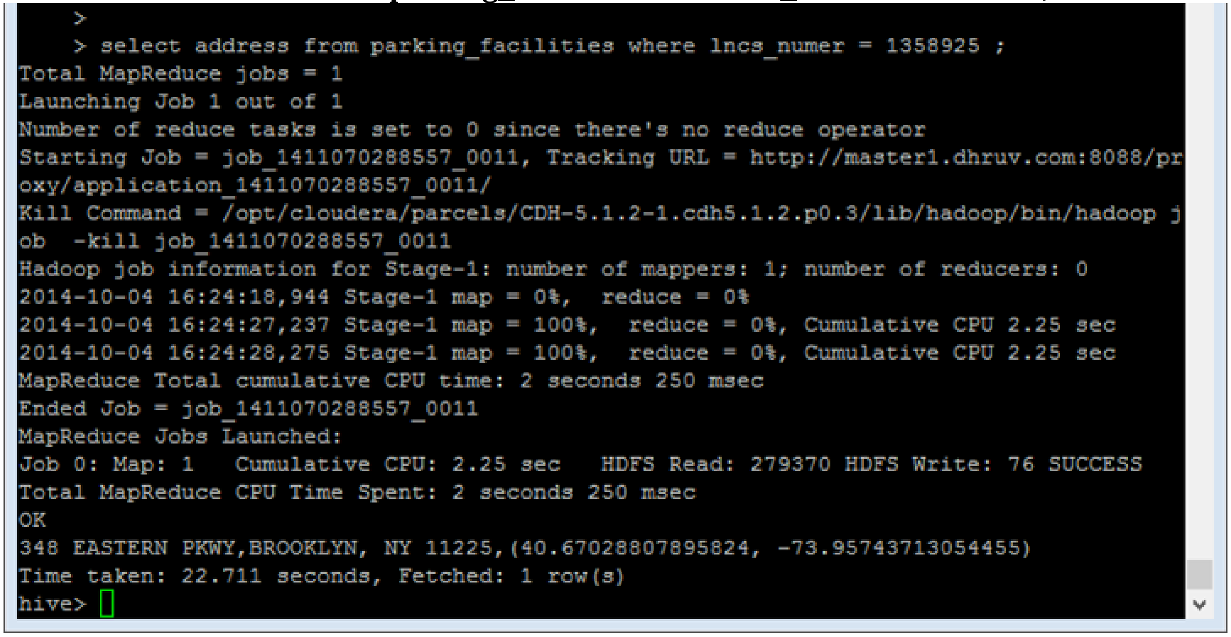
1. hive> select address from parking_facilities LIMIT 2;
The result from the query matches the first row in the csv file.



```
    >
    > select address from parking_facilities LIMIT 2;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1411070288557_0010, Tracking URL = N/A
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/ha
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-10-04 16:23:15,607 Stage-1 map = 0%,  reduce = 0%
2014-10-04 16:23:22,865 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.2 sec
2014-10-04 16:23:23,902 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.2 sec
2014-10-04 16:23:24,947 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.2 sec
MapReduce Total cumulative CPU time: 1 seconds 200 msec
Ended Job = job_1411070288557_0010
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.2 sec   HDFS Read: 65787 HDFS Write: 89 SUCCES
S
Total MapReduce CPU Time Spent: 1 seconds 200 msec
OK
Address
511 25 WEST 18 STREET,NEW YORK, NY 10011,(40.74471526669623, -74.00460000028977)
Time taken: 22.744 seconds, Fetched: 2 row(s)
hive>
```

Since this matches the data in the original csv file, we can be sure that the data is correct.

2. hive> select address from parking_facilities where lncs_numer = 1358925 ;

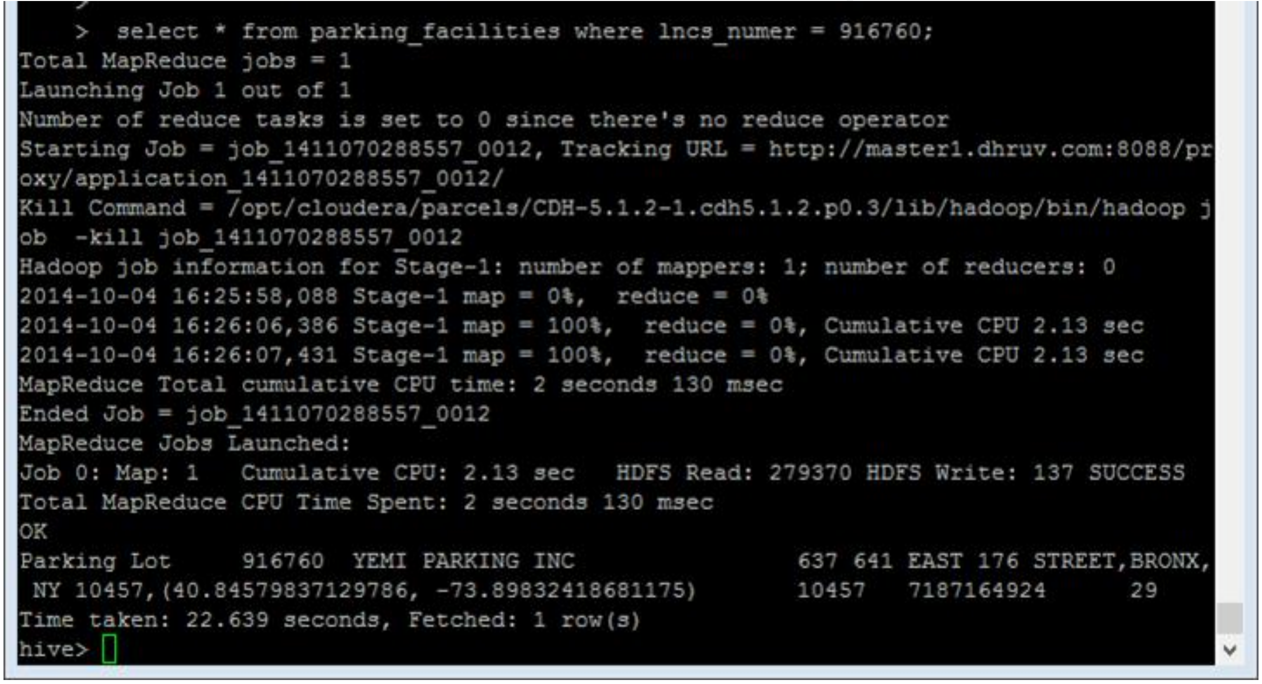


```
    >
    > select address from parking_facilities where lncs_numer = 1358925 ;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1411070288557_0011, Tracking URL = http://master1.dhruv.com:8088/pr
oxy/application_1411070288557_0011/
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/hadoop j
ob  -kill job_1411070288557_0011
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-10-04 16:24:18,944 Stage-1 map = 0%,  reduce = 0%
2014-10-04 16:24:27,237 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.25 sec
2014-10-04 16:24:28,275 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.25 sec
MapReduce Total cumulative CPU time: 2 seconds 250 msec
Ended Job = job_1411070288557_0011
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 2.25 sec   HDFS Read: 279370 HDFS Write: 76 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 250 msec
OK
348 EASTERN PKWY,BROOKLYN, NY 11225,(40.67028807895824, -73.95743713054455)
Time taken: 22.711 seconds, Fetched: 1 row(s)
hive> 
```

Since this matches the data in the original csv file, we can be sure that the data is correct.

3. hive> select * from parking_facilities where lncs_numer = 916760;

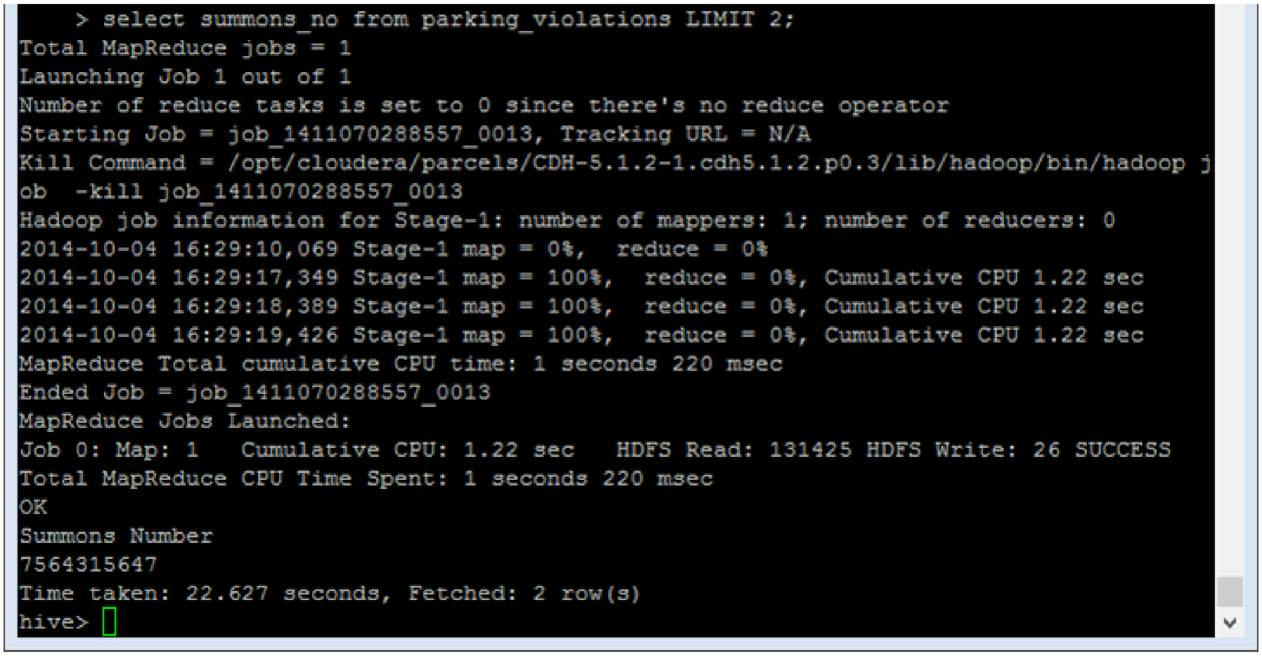
The result matches, even when a random value was chosen to be queried.



```
    >  select * from parking_facilities where lncs_numer = 916760;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1411070288557_0012, Tracking URL = http://master1.dhruv.com:8088/pr
oxy/application_1411070288557_0012/
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/hadoop j
ob  -kill job_1411070288557_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-10-04 16:25:58,088 Stage-1 map = 0%,  reduce = 0%
2014-10-04 16:26:06,386 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.13 sec
2014-10-04 16:26:07,431 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.13 sec
MapReduce Total cumulative CPU time: 2 seconds 130 msec
Ended Job = job_1411070288557_0012
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 2.13 sec   HDFS Read: 279370 HDFS Write: 137 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 130 msec
OK
Parking Lot     916760  YEMI PARKING INC                637 641 EAST 176 STREET,BRONX,
 NY 10457,(40.84579837129786, -73.89832418681175)       10457   7187164924      29
Time taken: 22.639 seconds, Fetched: 1 row(s)
hive> 
```

The second verification is the parking_violations table.

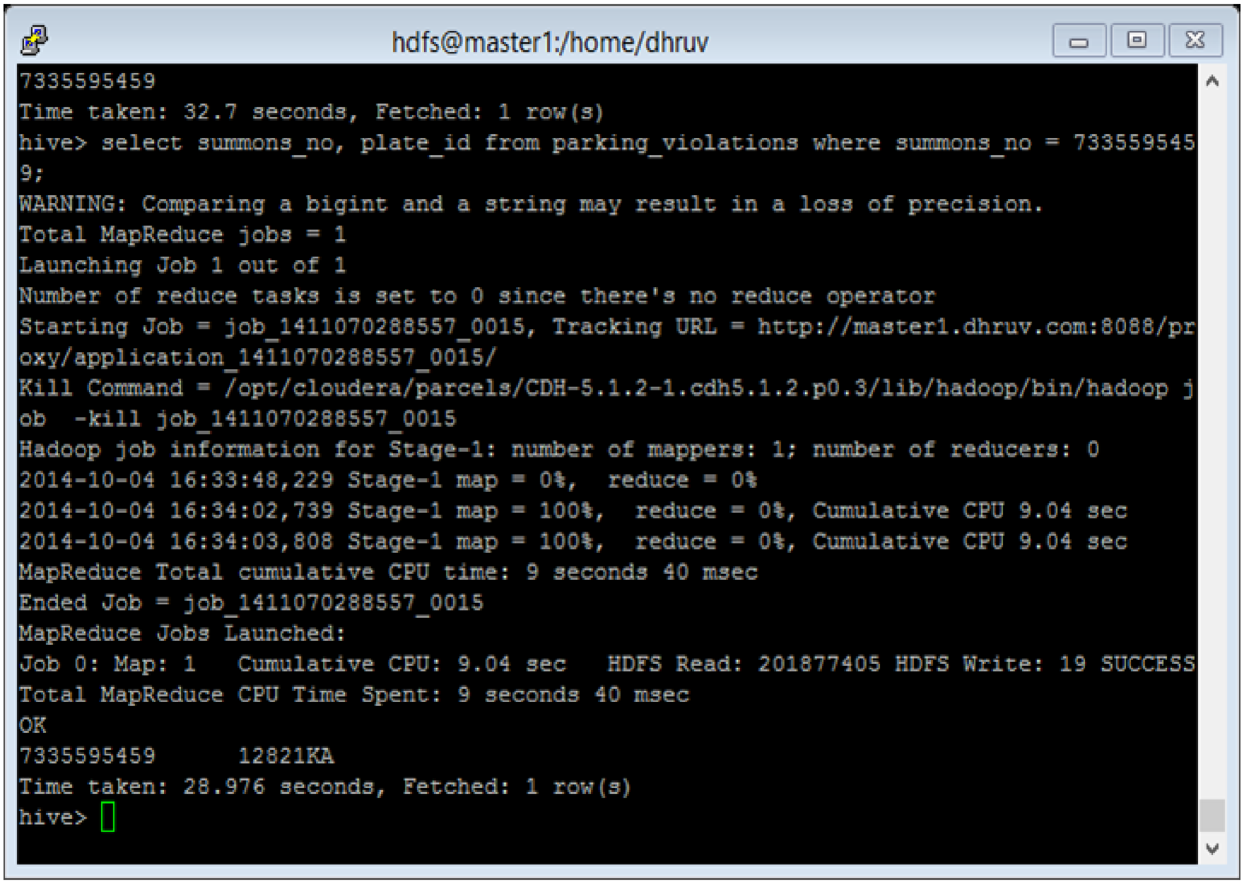
1.hive> select summons_no from parking_violations LIMIT 2;



```
    > select summons_no from parking_violations LIMIT 2;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1411070288557_0013, Tracking URL = N/A
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/hadoop j
ob  -kill job_1411070288557_0013
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-10-04 16:29:10,069 Stage-1 map = 0%,  reduce = 0%
2014-10-04 16:29:17,349 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.22 sec
2014-10-04 16:29:18,389 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.22 sec
2014-10-04 16:29:19,426 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.22 sec
MapReduce Total cumulative CPU time: 1 seconds 220 msec
Ended Job = job_1411070288557_0013
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.22 sec   HDFS Read: 131425 HDFS Write: 26 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 220 msec
OK
Summons Number
7564315647
Time taken: 22.627 seconds, Fetched: 2 row(s)
hive>
```

The first row is verified correctly, from the query.

2. hive> select summons_no, plate_id from parking_violations where summons_no = 7335595459;

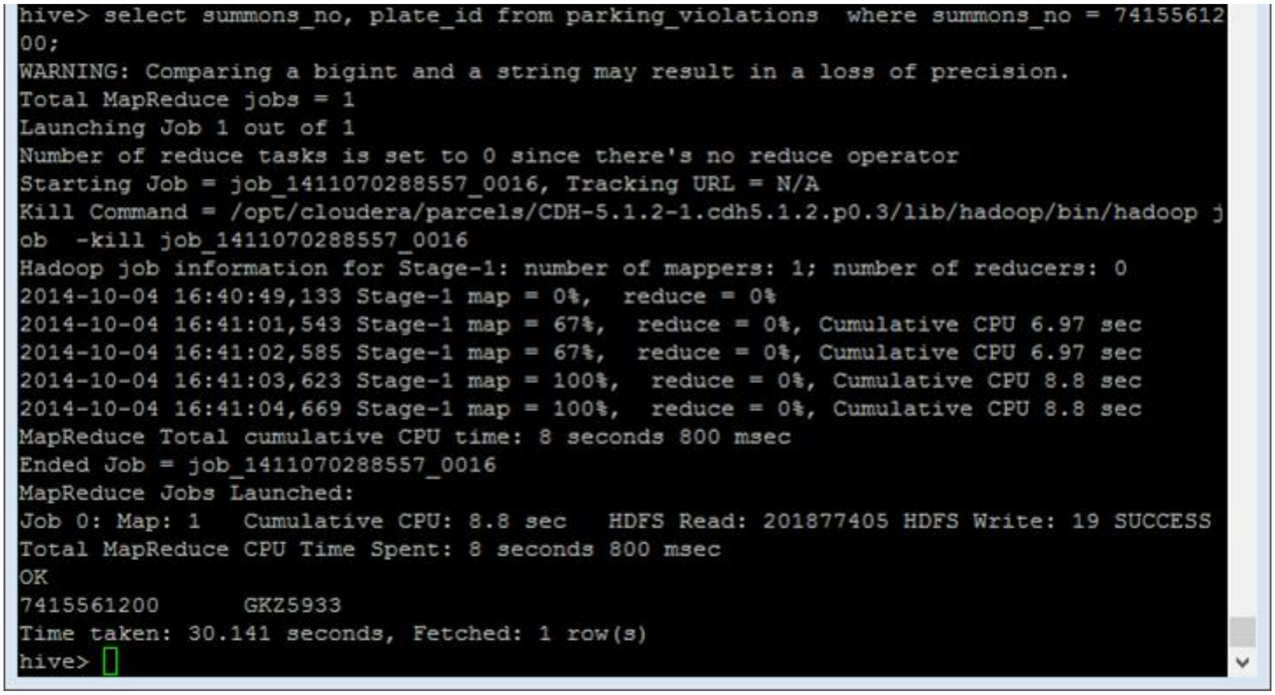


```
7335595459
Time taken: 32.7 seconds, Fetched: 1 row(s)
hive> select summons_no, plate_id from parking_violations where summons_no = 733559545
9;
WARNING: Comparing a bigint and a string may result in a loss of precision.
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1411070288557_0015, Tracking URL = http://master1.dhruv.com:8088/pr
oxy/application_1411070288557_0015/
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/hadoop j
ob  -kill job_1411070288557_0015
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-10-04 16:33:48,229 Stage-1 map = 0%,  reduce = 0%
2014-10-04 16:34:02,739 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 9.04 sec
2014-10-04 16:34:03,808 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 9.04 sec
MapReduce Total cumulative CPU time: 9 seconds 40 msec
Ended Job = job_1411070288557_0015
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 9.04 sec   HDFS Read: 201877405 HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 40 msec
OK
7335595459      12821KA
Time taken: 28.976 seconds, Fetched: 1 row(s)
hive> 
```

Upon choosing a random row, the verification is correct the second value is similar to the value of a plate_id and not some random string or other field.

3.hive> select summons_no, plate_id from parking_violations where summons_no = 7415561200

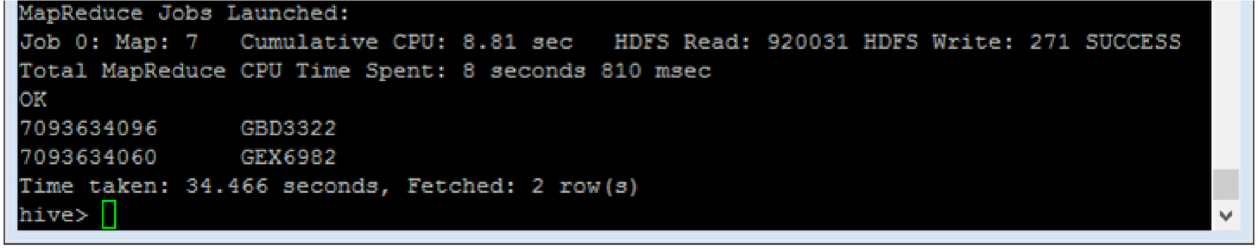
The data is verified correctly, because the last row from the query matches the last row of the csv file.



```
hive> select summons_no, plate_id from parking_violations  where summons_no = 74155612
00;
WARNING: Comparing a bigint and a string may result in a loss of precision.
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1411070288557_0016, Tracking URL = N/A
Kill Command = /opt/cloudera/parcels/CDH-5.1.2-1.cdh5.1.2.p0.3/lib/hadoop/bin/hadoop j
ob  -kill job_1411070288557_0016
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-10-04 16:40:49,133 Stage-1 map = 0%,  reduce = 0%
2014-10-04 16:41:01,543 Stage-1 map = 67%,  reduce = 0%, Cumulative CPU 6.97 sec
2014-10-04 16:41:02,585 Stage-1 map = 67%,  reduce = 0%, Cumulative CPU 6.97 sec
2014-10-04 16:41:03,623 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.8 sec
2014-10-04 16:41:04,669 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.8 sec
MapReduce Total cumulative CPU time: 8 seconds 800 msec
Ended Job = job_1411070288557_0016
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 8.8 sec   HDFS Read: 201877405 HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 800 msec
OK
7415561200      GKZ5933
Time taken: 30.141 seconds, Fetched: 1 row(s)
hive> 
```

The third verification is the speed_camera_tickets table

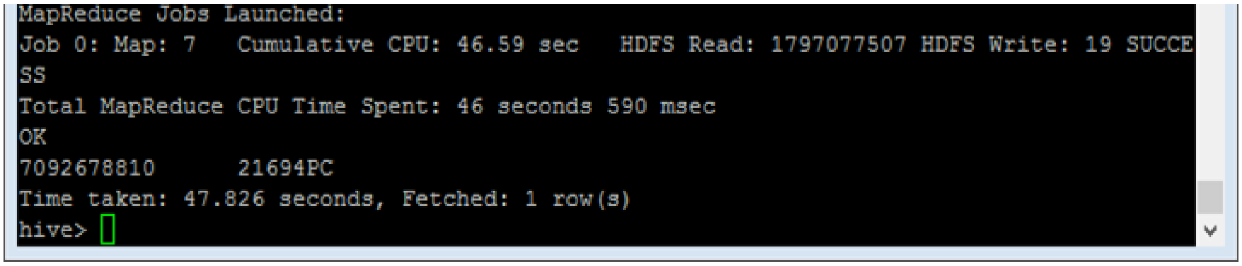
1. hive> select summons_number, plate_id from speed_cam_tickets LIMIT 2;



```
MapReduce Jobs Launched:
Job 0: Map: 7   Cumulative CPU: 8.81 sec   HDFS Read: 920031 HDFS Write: 271 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 810 msec
OK
7093634096      GBD3322
7093634060      GEX6982
Time taken: 34.466 seconds, Fetched: 2 row(s)
hive> 
```

The data from the query matches the data from the csv file, for the first two rows, and so the data is verified correctly.

2.hive> select summons_number, plate_id from speed_cam_tickets where summons_number = 7092678810

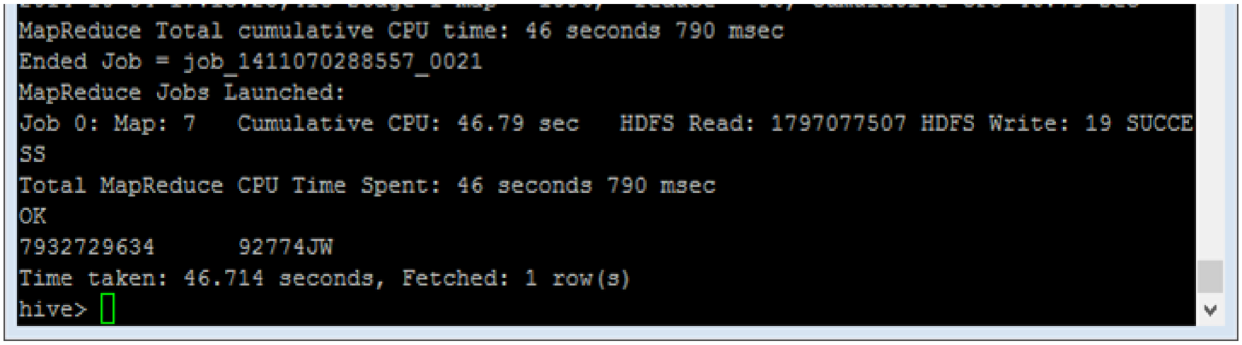


```
MapReduce Jobs Launched:
Job 0: Map: 7   Cumulative CPU: 46.59 sec   HDFS Read: 1797077507 HDFS Write: 19 SUCCE
SS
Total MapReduce CPU Time Spent: 46 seconds 590 msec
OK
7092678810      21694PC
Time taken: 47.826 seconds, Fetched: 1 row(s)
hive>
```

The above query was a random id, and it was verified correctly with the csv file.

3>hive> select summons_number, plate_id from speed_cam_tickets where summons_number = 7932729634

The query for the last row was correctly verified with the csv file.



```
MapReduce Total cumulative CPU time: 46 seconds 790 msec
Ended Job = job_1411070288557_0021
MapReduce Jobs Launched:
Job 0: Map: 7   Cumulative CPU: 46.79 sec   HDFS Read: 1797077507 HDFS Write: 19 SUCCE
SS
Total MapReduce CPU Time Spent: 46 seconds 790 msec
OK
7932729634      92774JW
Time taken: 46.714 seconds, Fetched: 1 row(s)
hive>
```

7. Phase II – Queries

For Phase II, we have come up with several queries that will help us analyze the parking services and violations in NYC. We have formulated a goal for each query, and the result of each query is also displayed via a screenshot. We have added one more table for cost analysis of violations.

The table consists of the list of violation codes and the associated fines. This will allow us to estimate the total fine collected on various violations,days,etc.

Table - violations_code_details;

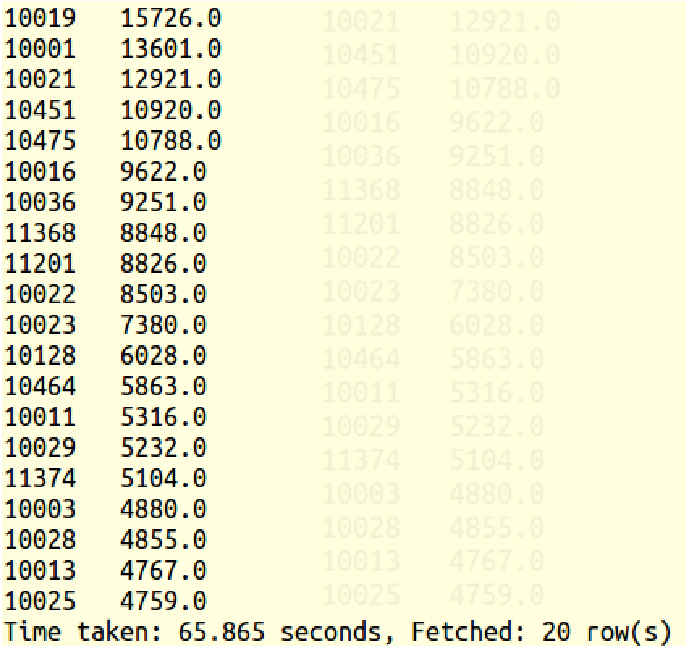
create table violations_code_details(violation_code INT, description STRING, fine INT) row format delimited fields terminated by '|' stored as textfile;

LOAD DATA INPATH "/user/project_data/NYC-Parking.csv" OVERWRITE INTO TABLE violations_code_details ;

7.1 Parking spaces by zip code

The objective is to find the zip codes with the most parking spaces available. This will help people to select location on public holidays for an outing where parking is easily available.

select address_zip_code, sum(number_of_spaces) as s from parking_facilities group by address_zip_code order by s desc limit 20;

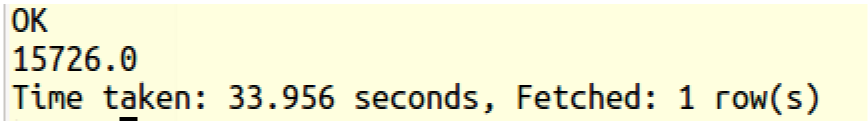


```
10019   15726.0
10001   13601.0
10021   12921.0
10451   10920.0
10475   10788.0
10016   9622.0
10036   9251.0
11368   8848.0
11201   8826.0
10022   8503.0
10023   7380.0
10128   6028.0
10464   5863.0
10011   5316.0
10029   5232.0
11374   5104.0
10003   4880.0
10028   4855.0
10013   4767.0
10025   4759.0
Time taken: 65.865 seconds, Fetched: 20 row(s)
```

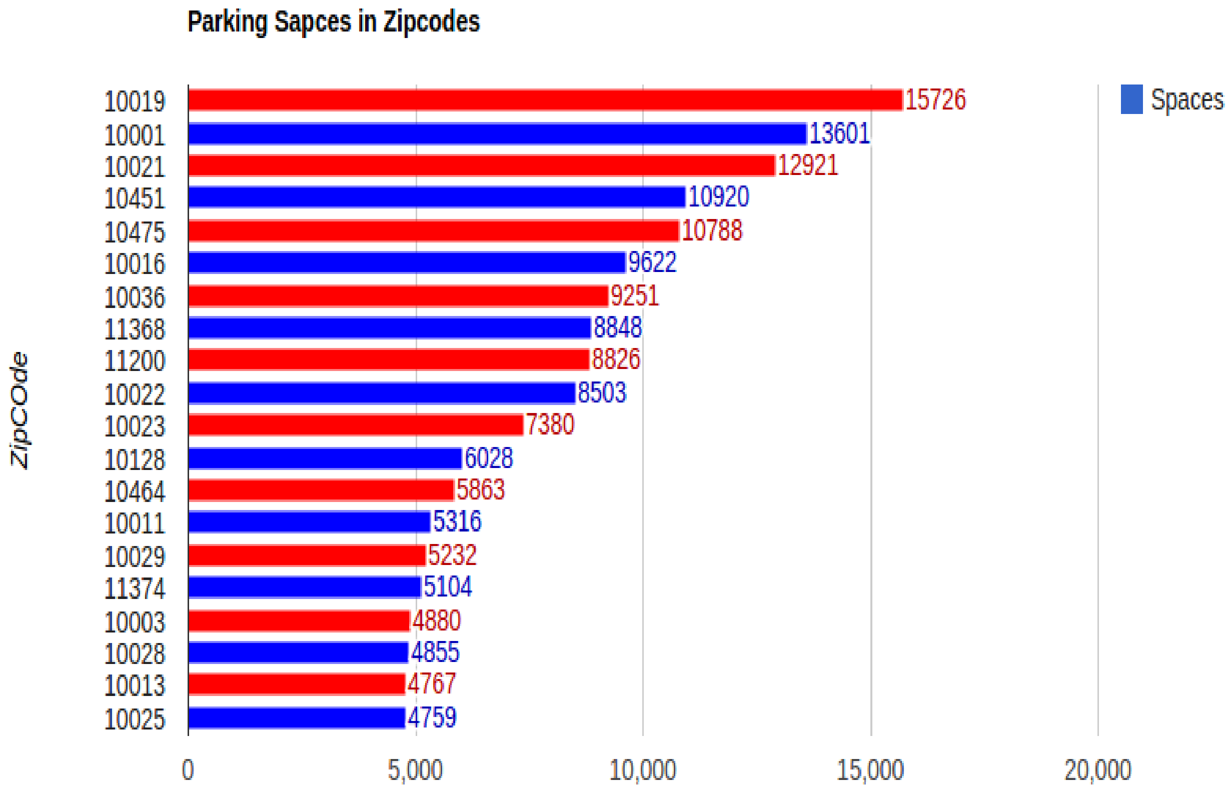
7.1.1 Verification of query

The verification can be done by randomly checking number of spaces in some zip code.

select sum(number_of_spaces) from parking_facilities where address_zip_code=10019;



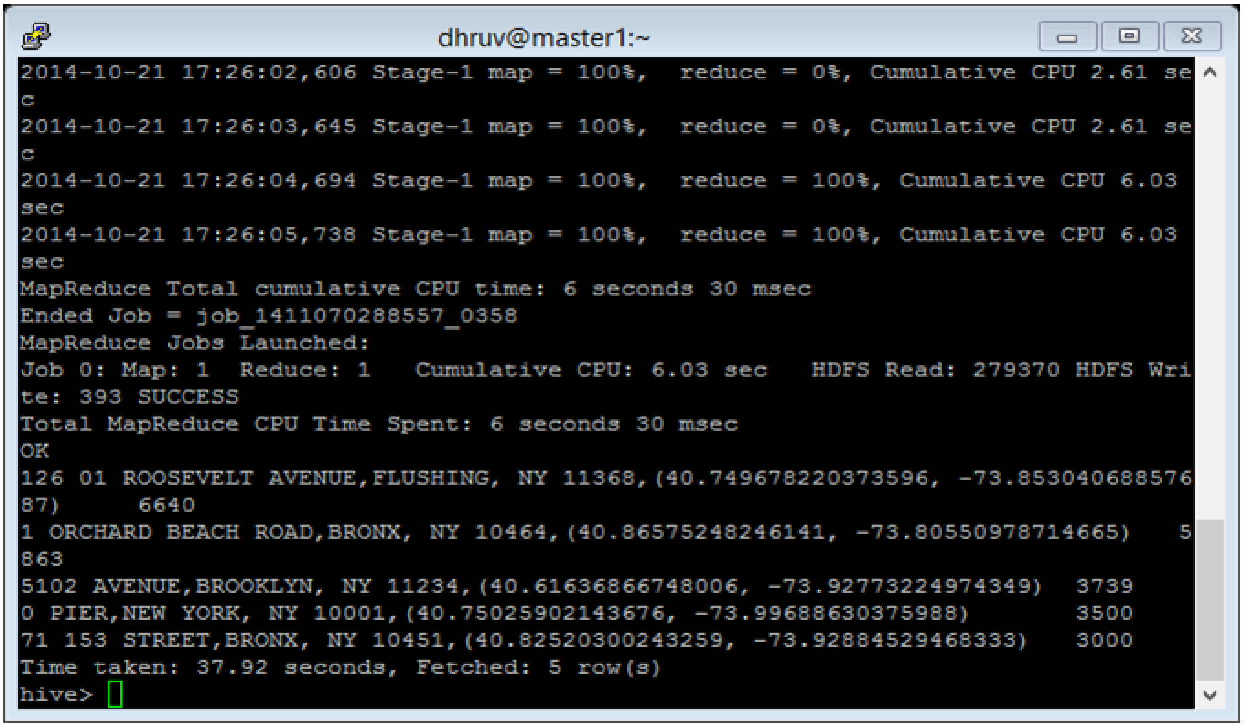
```
OK
15726.0
Time taken: 33.956 seconds, Fetched: 1 row(s)
```



7.2 Largest parking facilities in NYC

What is the best place to park in NYC in terms of the number of spaces available? We decided to search for this answer. This would help us establish the place where one has most chances of finding a parking.

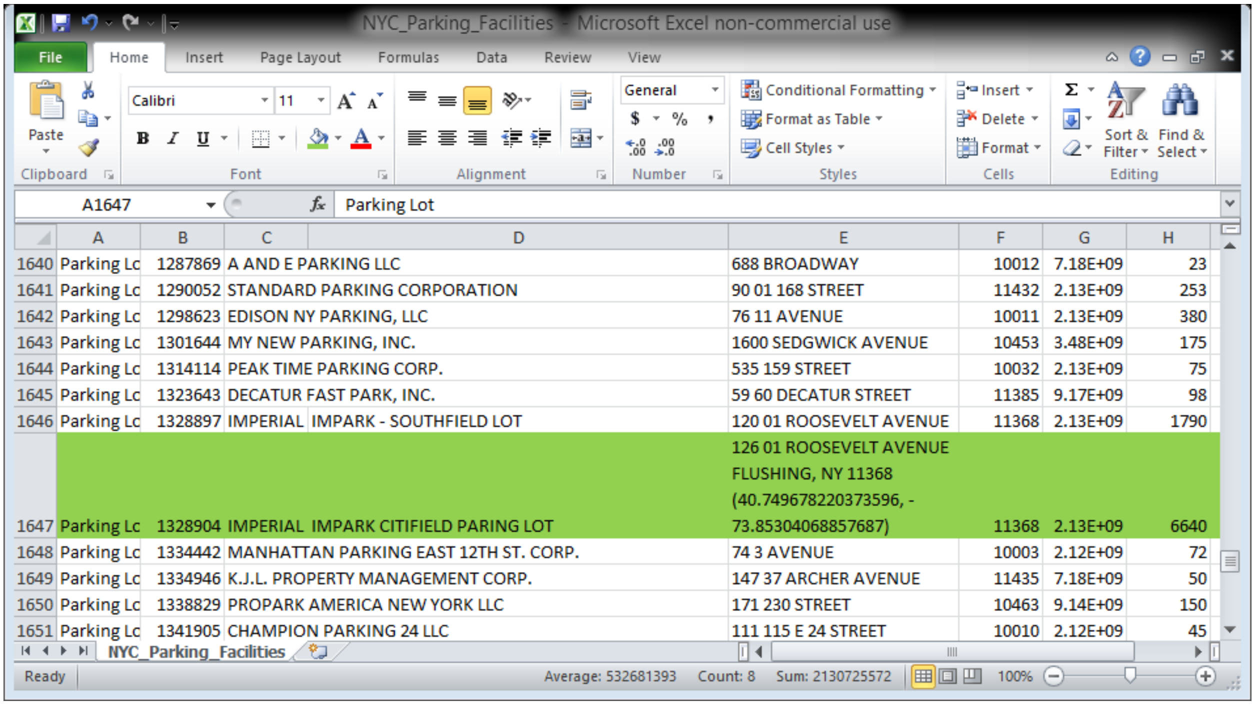
select address, number_of_spaces from (select address, number_of_spaces, rank() over (order by cast(number_of_spaces as float) desc) as r from parking_facilities) s limit 5;



```
2014-10-21 17:26:02,606 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.61 se
c
2014-10-21 17:26:03,645 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.61 se
c
2014-10-21 17:26:04,694 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.03
sec
2014-10-21 17:26:05,738 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.03
sec
MapReduce Total cumulative CPU time: 6 seconds 30 msec
Ended Job = job_1411070288557_0358
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 6.03 sec   HDFS Read: 279370 HDFS Wri
te: 393 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 30 msec
OK
126 01 ROOSEVELT AVENUE,FLUSHING, NY 11368,(40.749678220373596, -73.853040688576
87)     6640
1 ORCHARD BEACH ROAD,BRONX, NY 10464,(40.86575248246141, -73.80550978714665)    5
863
5102 AVENUE,BROOKLYN, NY 11234,(40.61636866748006, -73.92773224974349)  3739
0 PIER,NEW YORK, NY 10001,(40.75025902143676, -73.99688630375988)       3500
71 153 STREET,BRONX, NY 10451,(40.82520300243259, -73.92884529468333)   3000
Time taken: 37.92 seconds, Fetched: 5 row(s)
hive> 
```

7.2.1 Verification of query 7.1

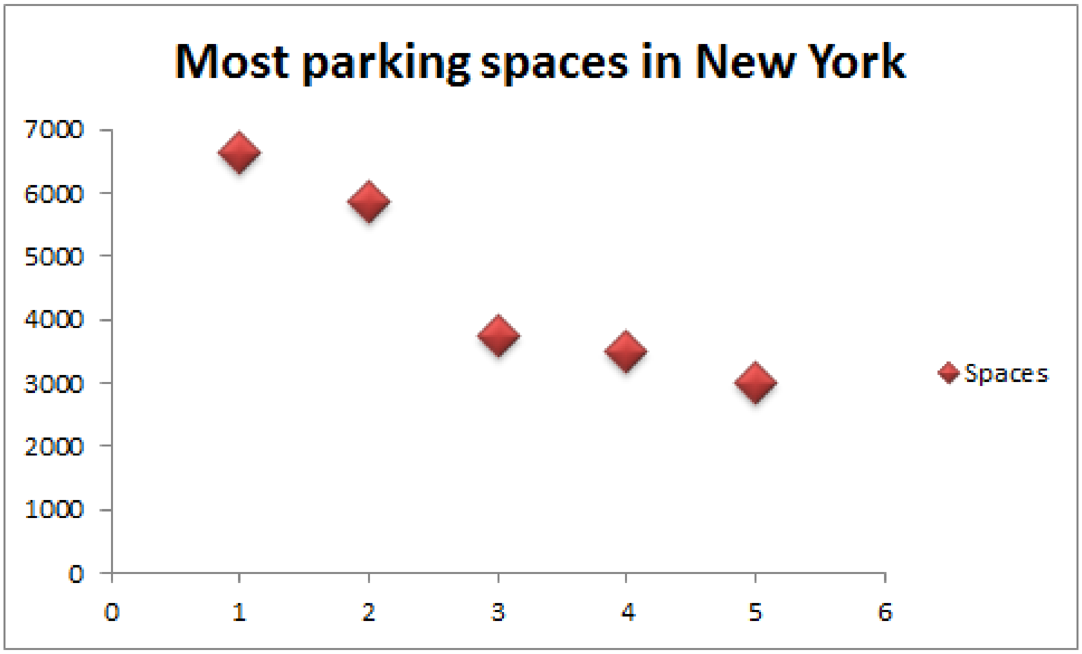
By manually searching the data set(as it is small), the parking facility located at Roosevelt Avenue does indeed have 6640 parking spaces. See the screenshot below.



A1647 | Parking Lot

	A	B	C	D	E	F	G	H
1640	Parking Lc	1287869	A AND E PARKING LLC		688 BROADWAY	10012	7.18E+09	23
1641	Parking Lc	1290052	STANDARD PARKING CORPORATION		90 01 168 STREET	11432	2.13E+09	253
1642	Parking Lc	1298623	EDISON NY PARKING, LLC		76 11 AVENUE	10011	2.13E+09	380
1643	Parking Lc	1301644	MY NEW PARKING, INC.		1600 SEDGWICK AVENUE	10453	3.48E+09	175
1644	Parking Lc	1314114	PEAK TIME PARKING CORP.		535 159 STREET	10032	2.13E+09	75
1645	Parking Lc	1323643	DECATUR FAST PARK, INC.		59 60 DECATUR STREET	11385	9.17E+09	98
1646	Parking Lc	1328897	IMPERIAL	IMPARK - SOUTHFIELD LOT	120 01 ROOSEVELT AVENUE	11368	2.13E+09	1790
1647	Parking Lc	1328904	IMPERIAL	IMPARK CITIFIELD PARING LOT	126 01 ROOSEVELT AVENUE FLUSHING, NY 11368 (40.749678220373596, - 73.85304068857687)	11368	2.13E+09	6640
1648	Parking Lc	1334442	MANHATTAN PARKING EAST 12TH ST. CORP.		74 3 AVENUE	10003	2.12E+09	72
1649	Parking Lc	1334946	K.J.L. PROPERTY MANAGEMENT CORP.		147 37 ARCHER AVENUE	11435	7.18E+09	50
1650	Parking Lc	1338829	PROPARK AMERICA NEW YORK LLC		171 230 STREET	10463	9.14E+09	150
1651	Parking Lc	1341905	CHAMPION PARKING 24 LLC		111 115 E 24 STREET	10010	2.12E+09	45

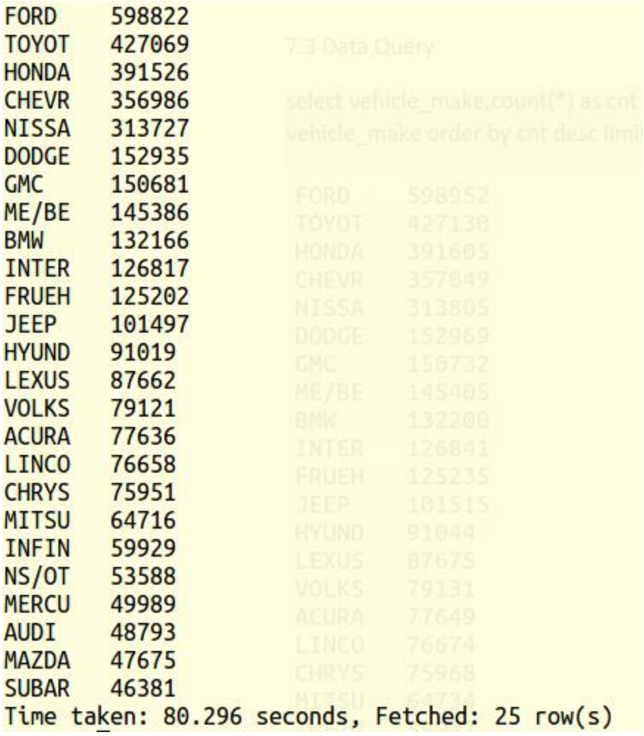
Average: 532681393 Count: 8 Sum: 2130725572



7.3 Car companies with most violations

Objective of the query to establish a relation between car companies and violations. The car companies give us a idea of the attributes of the car owner. So at a high level we can say that we are able to make a comment on the violation and car owner.

```
select vehicle_make,count(*) as cnt from parking_violations where year(issue_date) =2013
group by vehicle_make order by cnt desc limit 25;
```

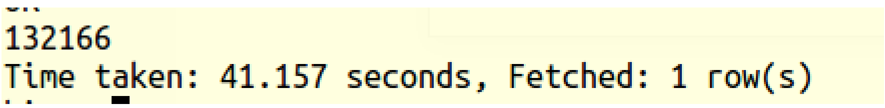


```
FORD     598822
TOYOT    427069
HONDA    391526
CHEVR    356986
NISSA    313727
DODGE    152935
GMC      150681
ME/BE    145386
BMW      132166
INTER    126817
FRUEH    125202
JEEP     101497
HYUND    91019
LEXUS    87662
VOLKS    79121
ACURA    77636
LINCO    76658
CHRYS    75951
MITSU    64716
INFIN    59929
NS/OT    53588
MERCU    49989
AUDI     48793
MAZDA    47675
SUBAR    46381
Time taken: 80.296 seconds, Fetched: 25 row(s)
```

7.3.1 Verification of query

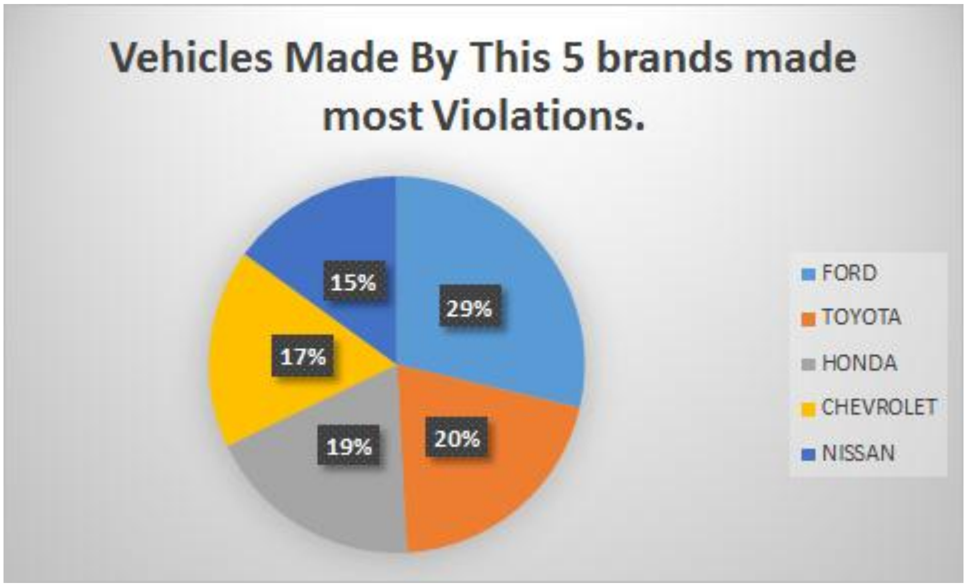
The verification of the query can be done by checking the number of violations randomly for some company.

select count(distinct summons_no) from parking_violations where vehicle_make='BMW' and year(issue_date)=2013;



```
132166
Time taken: 41.157 seconds, Fetched: 1 row(s)
```

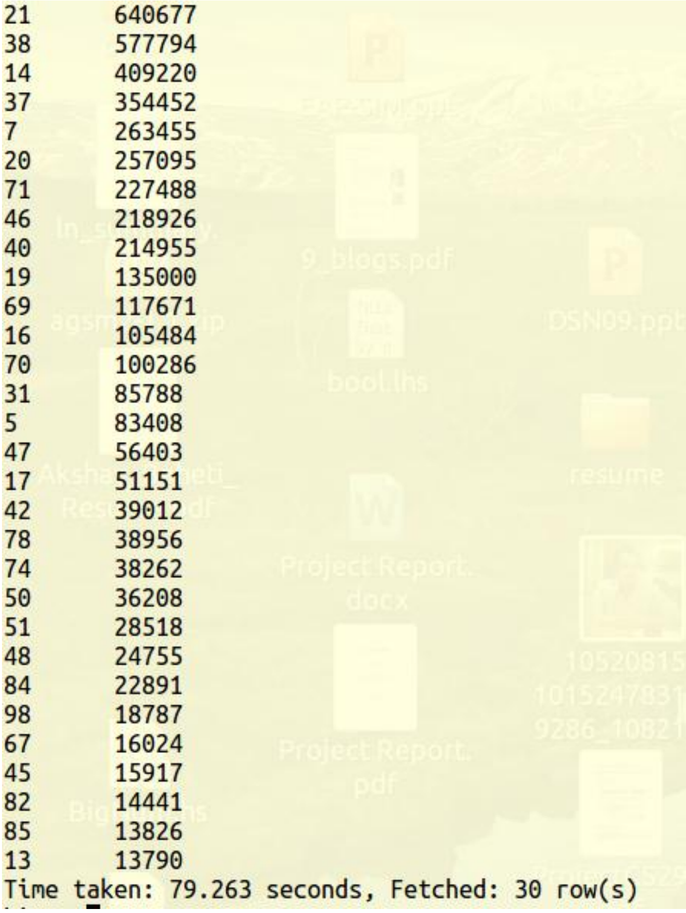
Here we see that the tickets issued to BMW’s in 2013 are equal to the result displayed in main query.



7.4 Violation codes with the most violations

The objective of the query is to find the number of violations under each violation code. This will help us inform New Yorkers about parking tips to avoid a ticket.

```
select violation_code,count(*) as cnt from parking_violations where YEAR(issue_date)=2013
group by violation_code  order by cnt desc limit 30;
```

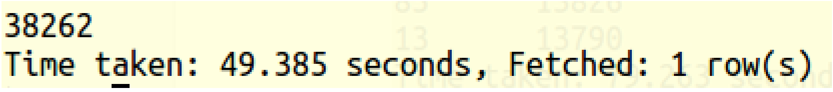


```
21      640677
38      577794
14      409220
37      354452
7       263455
20      257095
71      227488
46      218926
40      214955
19      135000
69      117671
16      105484
70      100286
31      85788
5       83408
47      56403
17      51151
42      39012
78      38956
74      38262
50      36208
51      28518
48      24755
84      22891
98      18787
67      16024
45      15917
82      14441
85      13826
13      13790
Time taken: 79.263 seconds, Fetched: 30 row(s)
```

7.4.1 Verification of query

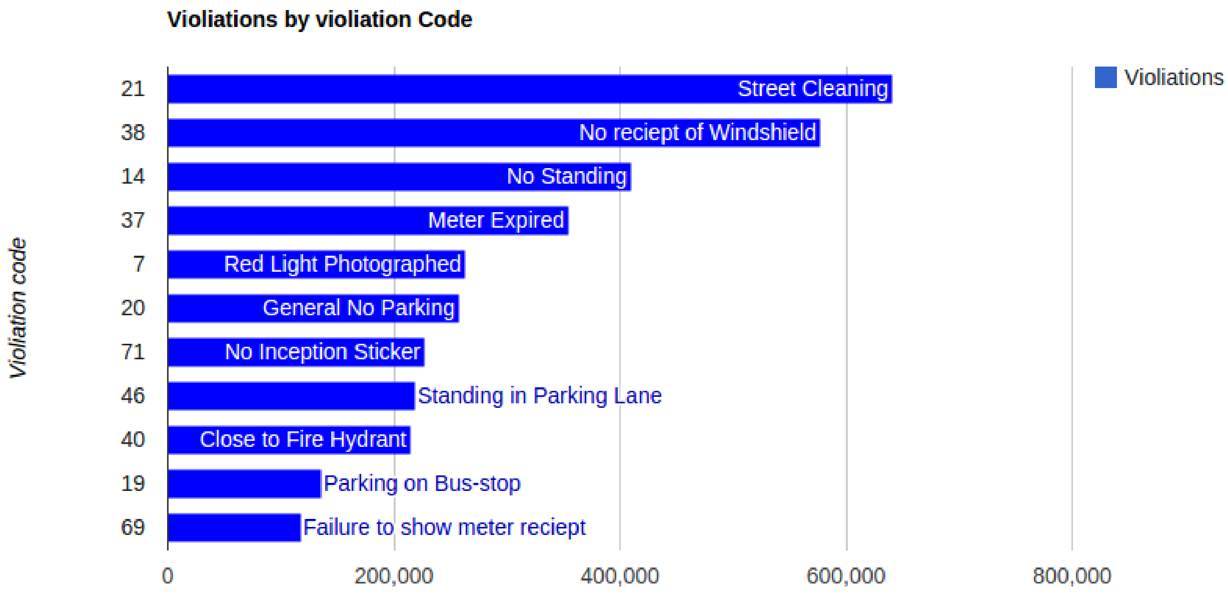
Verification can be done randomly checking the number of violation for some random violation code.

select count(distinct summons_no) as cnt from parking_violations where
YEAR(issue_date)=2013 and violation_code=74;



```
38262
Time taken: 49.385 seconds, Fetched: 1 row(s)
```

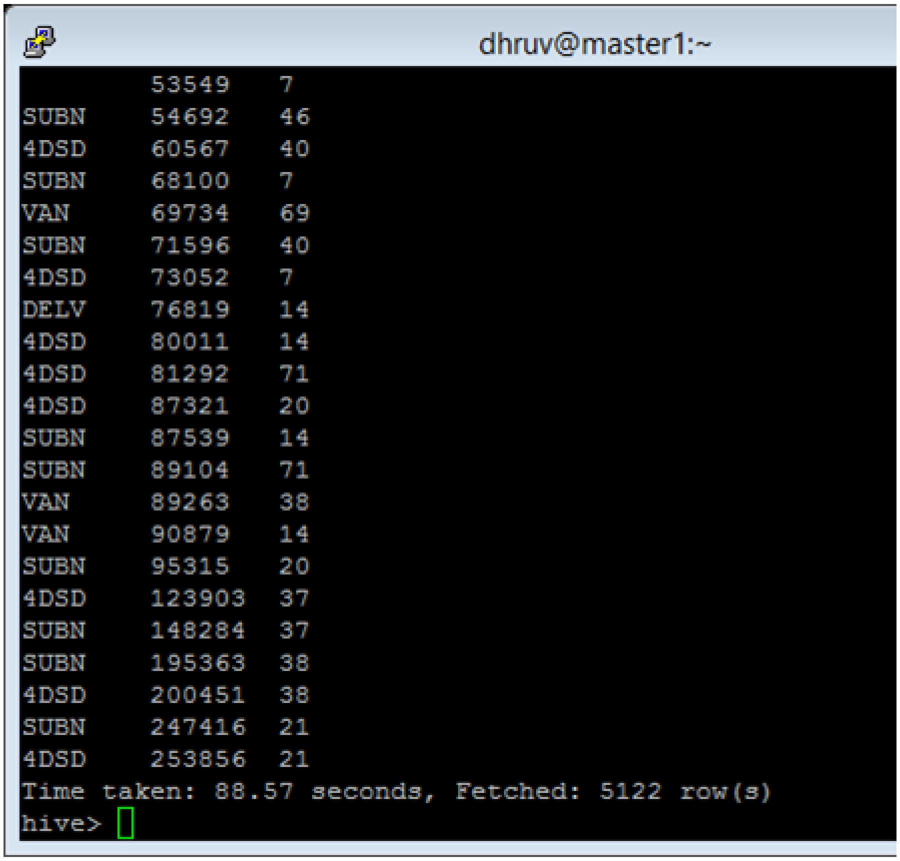
Here we see that the number of violations under violation code 74 is same as the result obtained in the main query.



7.5 Violations by body types

The objective of this query is to obtain a relation between the vehicle's body type and the violation code. This may give us some idea about how the size/purpose of the vehicle may result in a parking violation easily. Example a truck for some courier company like UPS may get tickets for standing in a no parking zone.

select parking_violations.vehicle_body_type, count(parking_violations.violation_code) as c, parking_violations.violation_code from violations_code_details join parking_violations on parking_violations.violation_code = violations_code_details.violation_code group by parking_violations.violation_code, parking_violations.vehicle_body_type order by c desc limit 5;



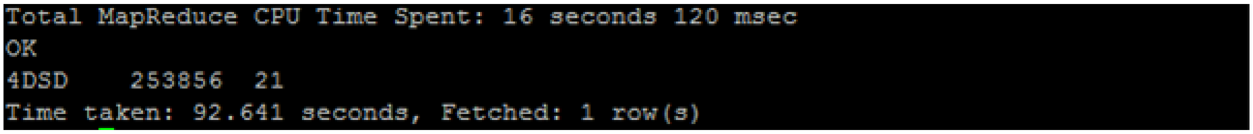
```
        53549   7
SUBN    54692   46
4DSD    60567   40
SUBN    68100   7
VAN     69734   69
SUBN    71596   40
4DSD    73052   7
DELV    76819   14
4DSD    80011   14
4DSD    81292   71
4DSD    87321   20
SUBN    87539   14
SUBN    89104   71
VAN     89263   38
VAN     90879   14
SUBN    95315   20
4DSD    123903  37
SUBN    148284  37
SUBN    195363  38
4DSD    200451  38
SUBN    247416  21
4DSD    253856  21
Time taken: 88.57 seconds, Fetched: 5122 row(s)
hive>
```

7.5.1 Verification of query

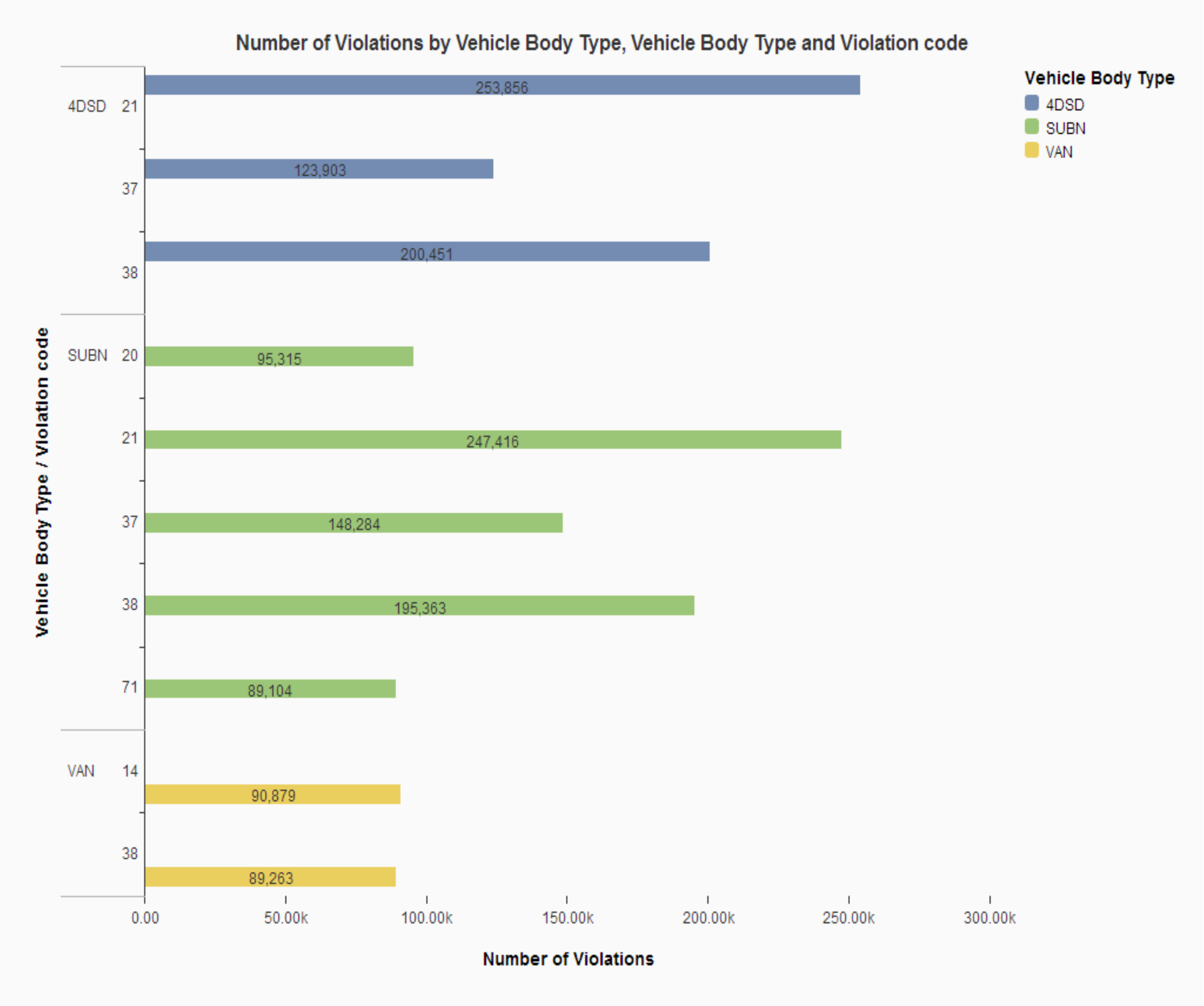
This query can be verified by taking a small subset of the violations and calculating the sum for small groups. In this case, four door sedans received the highest amount of violation tickets, at 253856 violations.

The verification query is:

select parking_violations.vehicle_body_type, count(parking_violations.violation_code) as c, parking_violations.violation_code from violations_code_details join parking_violations on parking_violations.violation_code = violations_code_details.violation_code where parking_violations.violation_code = 21 and parking_violations.vehicle_body_type = '4DSD' group by parking_violations.violation_code, parking_violations.vehicle_body_type order by c desc limit 5;



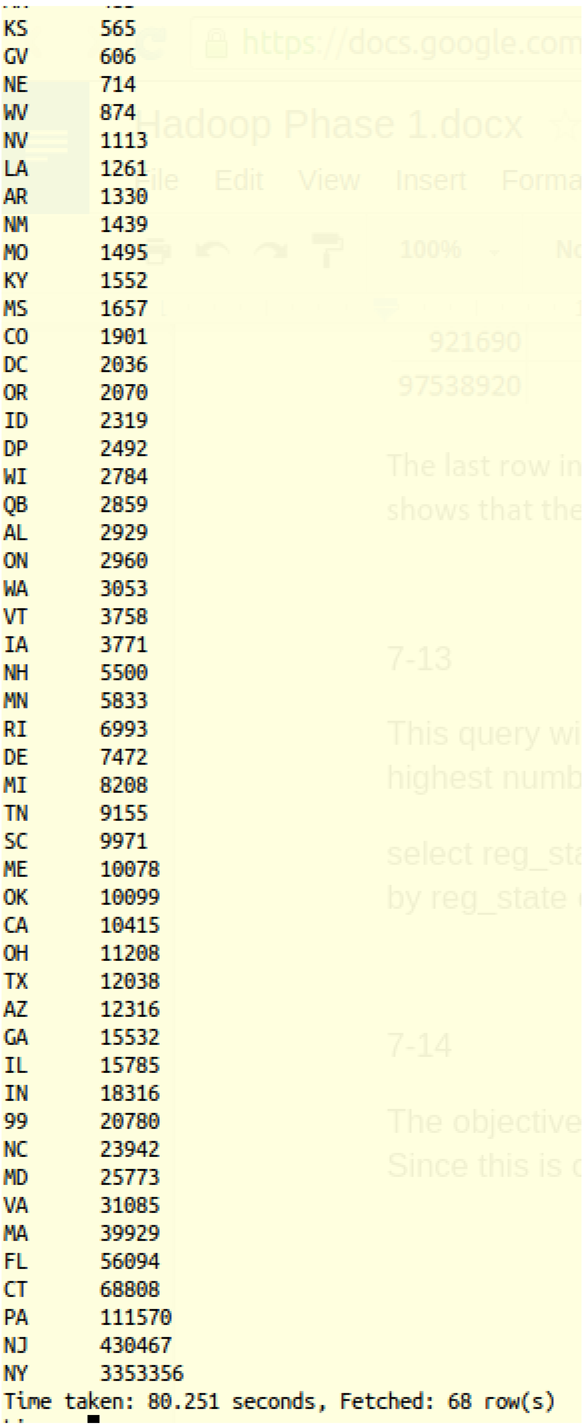
```
Total MapReduce CPU Time Spent: 16 seconds 120 msec
OK
4DSD    253856  21
Time taken: 92.641 seconds, Fetched: 1 row(s)
```



7.6 Highest number of violations by state

This query will determine the highest number of violations for the state. We expect that the highest number of violations will be from the state of NY, and other surrounding states. But on a broader level it can be the case that some parking rules are different that the visitors home state for example California. In this case we can help the visitor by warning him to be careful while parking in New York.

```
select reg_state,count(*) as cnt from parking_violations where YEAR(issue_date)=2013 group
by reg_state order by cnt desc ;
```

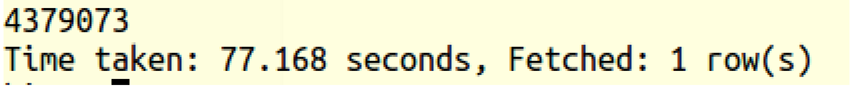


```
KS      565
GV      606
NE      714
WV      874
NV      1113
LA      1261
AR      1330
NM      1439
MO      1495
KY      1552
MS      1657
CO      1901
DC      2036
OR      2070
ID      2319
DP      2492
WI      2784
QB      2859
AL      2929
ON      2960
WA      3053
VT      3758
IA      3771
NH      5500
MN      5833
RI      6993
DE      7472
MI      8208
TN      9155
SC      9971
ME      10078
OK      10099
CA      10415
OH      11208
TX      12038
AZ      12316
GA      15532
IL      15785
IN      18316
99      20780
NC      23942
MD      25773
VA      31085
MA      39929
FL      56094
CT      68808
PA      111570
NJ      430467
NY      3353356
Time taken: 80.251 seconds, Fetched: 68 row(s)
```

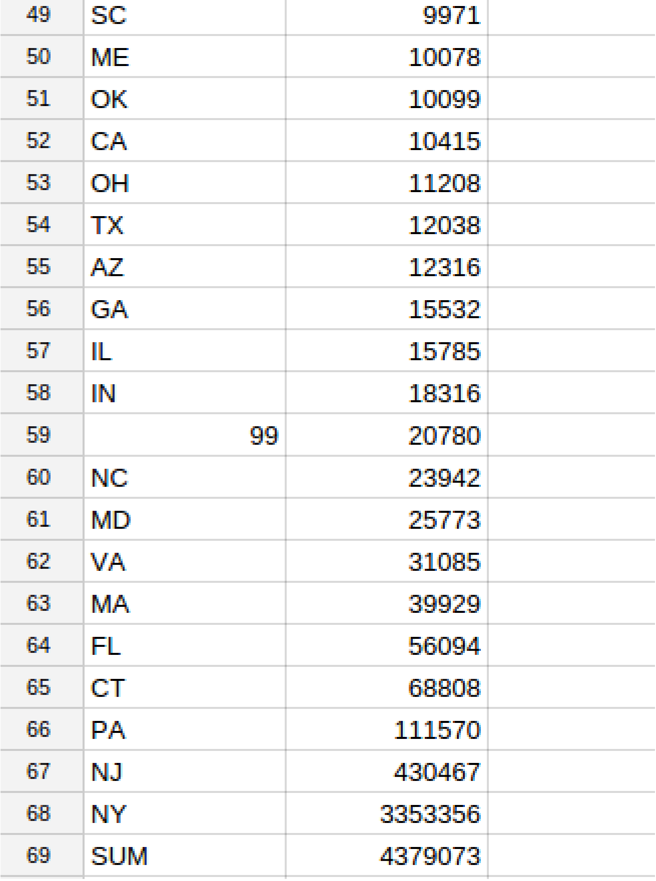
7.6.1 Verification of query

For Verification we take the count of all the tickets issued in the year 2013 and compare it with the sum all entries obtained in the above query.

select count(distinct summons_no) as cnt from parking_violations where YEAR(issue_date)=2013;

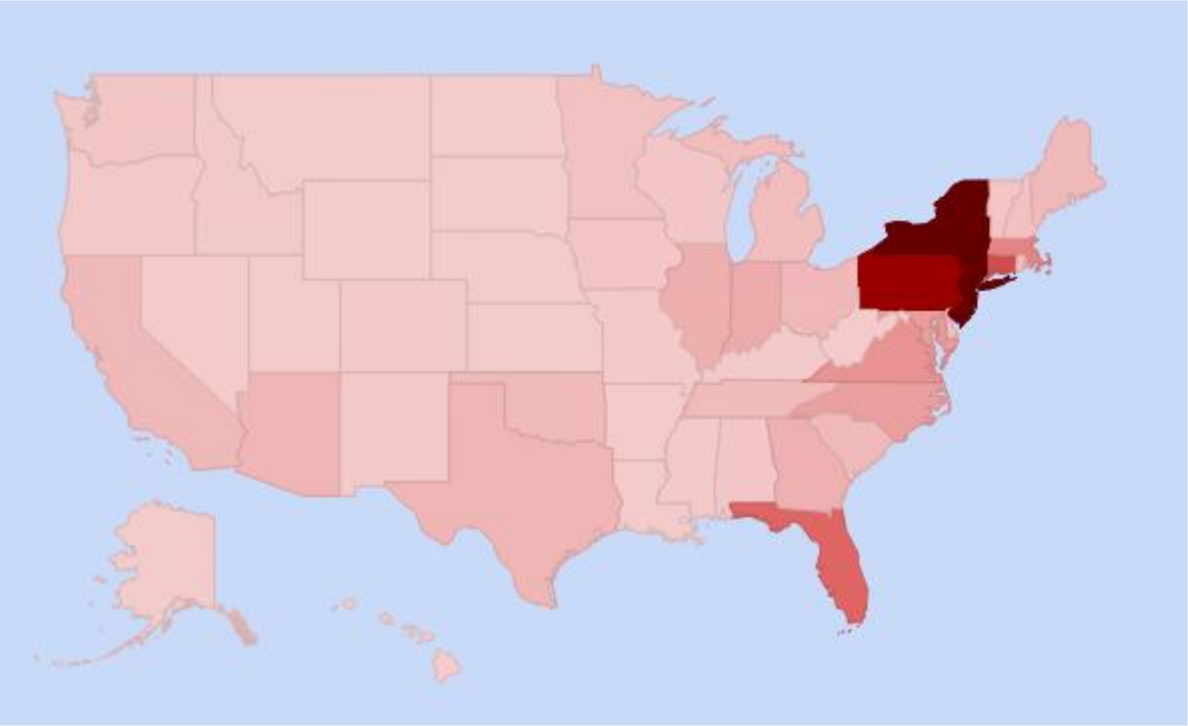


```
4379073
Time taken: 77.168 seconds, Fetched: 1 row(s)
```



49	SC	9971
50	ME	10078
51	OK	10099
52	CA	10415
53	OH	11208
54	TX	12038
55	AZ	12316
56	GA	15532
57	IL	15785
58	IN	18316
59	99	20780
60	NC	23942
61	MD	25773
62	VA	31085
63	MA	39929
64	FL	56094
65	CT	68808
66	PA	111570
67	NJ	430467
68	NY	3353356
69	SUM	4379073

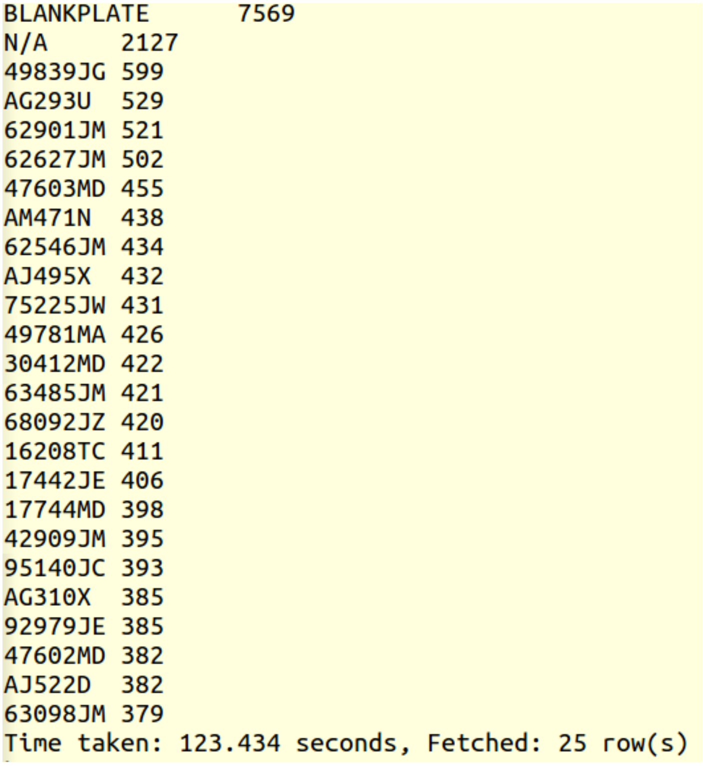
We observe that the total number of violations is the same.



7.7 Car with most violations

The objective of this query is to determine which car has had the most amount of violations. Since this is over a 2 year period, it will highlight the cars that get repeated violations. This will help us observe the car type of the with most violations.

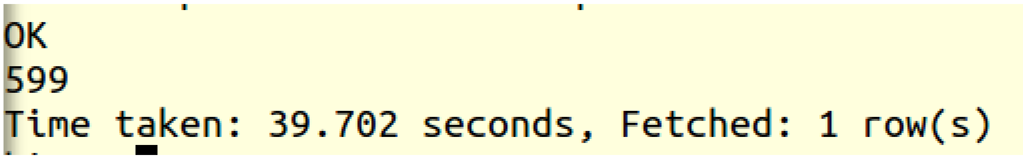
```
select plate_id,count(*) as cnt from parking_violations group by plate_id order by cnt desc limit 25;
```



```
BLANKPLATE      7569
N/A     2127
49839JG 599
AG293U  529
62901JM 521
62627JM 502
47603MD 455
AM471N  438
62546JM 434
AJ495X  432
75225JW 431
49781MA 426
30412MD 422
63485JM 421
68092JZ 420
16208TC 411
17442JE 406
17744MD 398
42909JM 395
95140JC 393
AG310X  385
92979JE 385
47602MD 382
AJ522D  382
63098JM 379
Time taken: 123.434 seconds, Fetched: 25 row(s)
```

7.7.1 Verification of query

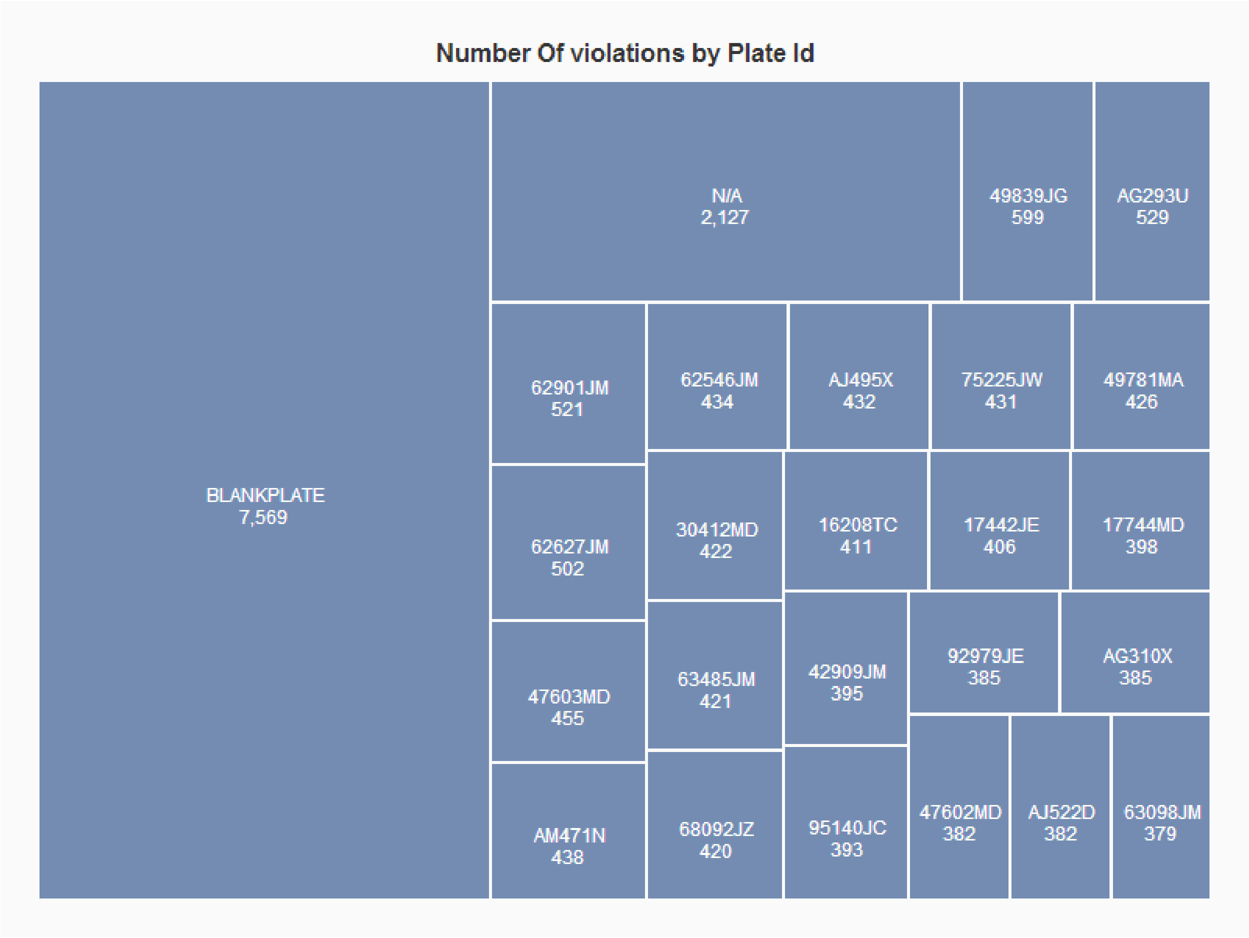
select count(distinct summons_no) as cnt from parking_violations where plate_id="49839JG";



```
OK
599
Time taken: 39.702 seconds, Fetched: 1 row(s)
```

Here we see that the violations by plate_id=49839JG are same in both the queries.

Visualization



7.8 Total Monthly violations in 2013

The objective of the query is to find the total monthly violations in New York for the year 2013. This will give us a pattern over a year for the violations.

```
select YEAR(parking_violations.issue_date),

        count(if (MONTH(parking_violations.issue_date)=1,1,null)) as jan,

        count(if (MONTH(parking_violations.issue_date)=2,1,null)) as feb,

        count(if (MONTH(parking_violations.issue_date)=3,1,null)) as mar,

        count(if (MONTH(parking_violations.issue_date)=4,1,null)) as apr,
```

count(if (MONTH(parking_violations.issue_date)=5,1,null)) as may,

count(if (MONTH(parking_violations.issue_date)=6,1,null)) as jun,

count(if (MONTH(parking_violations.issue_date)=7,1,null)) as jul,

count(if (MONTH(parking_violations.issue_date)=8,1,null)) as aug,

count(if (MONTH(parking_violations.issue_date)=9,1,null)) as sep,

count(if (MONTH(parking_violations.issue_date)=10,1,null)) as oct,

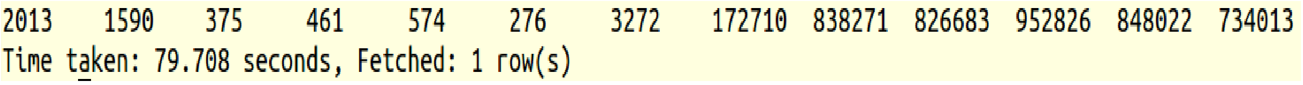
count(if (MONTH(parking_violations.issue_date)=11,1,null)) as nov,

count(if (MONTH(parking_violations.issue_date)=12,1,null)) as dec

from parking_violations

where parking_violations.issue_date is not null and YEAR(parking_violations.issue_date)=2013

group by YEAR(parking_violations.issue_date);

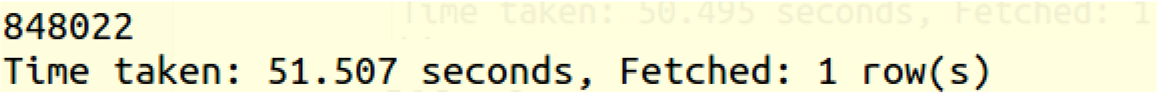


```
2013    1590    375     461     574     276     3272    172710  838271  826683  952826  848022  734013
Time taken: 79.708 seconds, Fetched: 1 row(s)
```

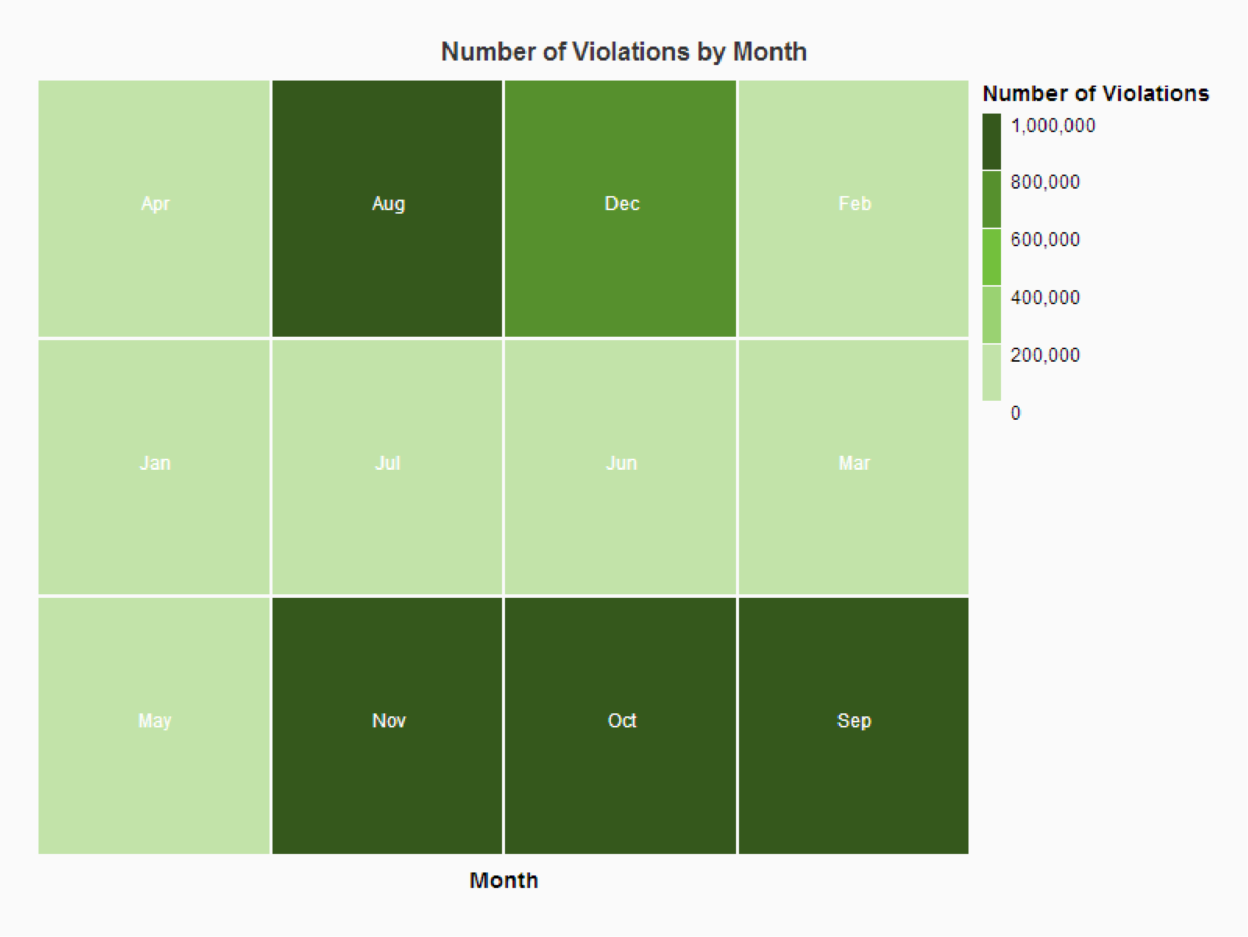
7.8.1 Verification of query

Check total violations for some random month like November 2013.

select count(distinct summons_no) from parking_violations where MONTH(issue_date) == 11 and YEAR(issue_date)=2013;



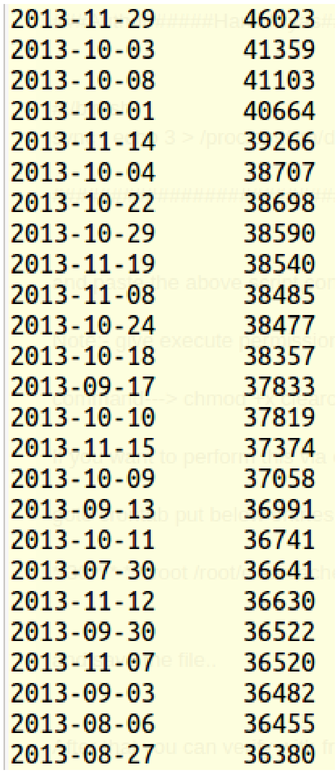
```
848022
Time taken: 51.507 seconds, Fetched: 1 row(s)
```



7.9 Day of maximum violations

Aim to is find days when maximum parking violations issued in New York on a particular day. This will help us guide citizens to be careful with parking on those days. Like in the query we observe that the most violations were issued on 29th Nov 2013. This is Black Friday.

```
select issue_date,count(*) as cnt from parking_violations group by issue_date order by cnt desc limit 25;
```

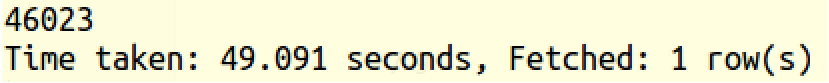


```
2013-11-29      46023
2013-10-03      41359
2013-10-08      41103
2013-10-01      40664
2013-11-14      39266
2013-10-04      38707
2013-10-22      38698
2013-10-29      38590
2013-11-19      38540
2013-11-08      38485
2013-10-24      38477
2013-10-18      38357
2013-09-17      37833
2013-10-10      37819
2013-11-15      37374
2013-10-09      37058
2013-09-13      36991
2013-10-11      36741
2013-07-30      36641
2013-11-12      36630
2013-09-30      36522
2013-11-07      36520
2013-09-03      36482
2013-08-06      36455
2013-08-27      36380
```

7.9.1 Verification of query

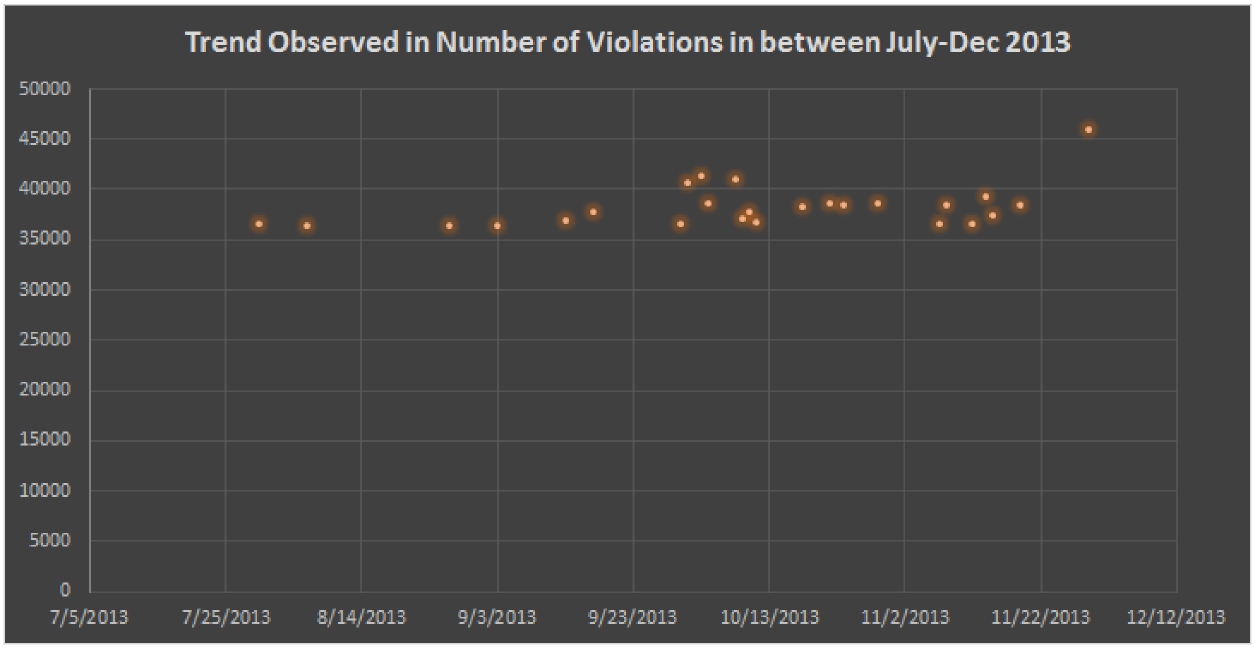
The query can be verified by checking the total violations for a particular day. We will check the number of violations for a 29-No-2013

select count(distinct summons_no) from parking_violations where issue_date='2013-11-29';



```
46023
Time taken: 49.091 seconds, Fetched: 1 row(s)
```

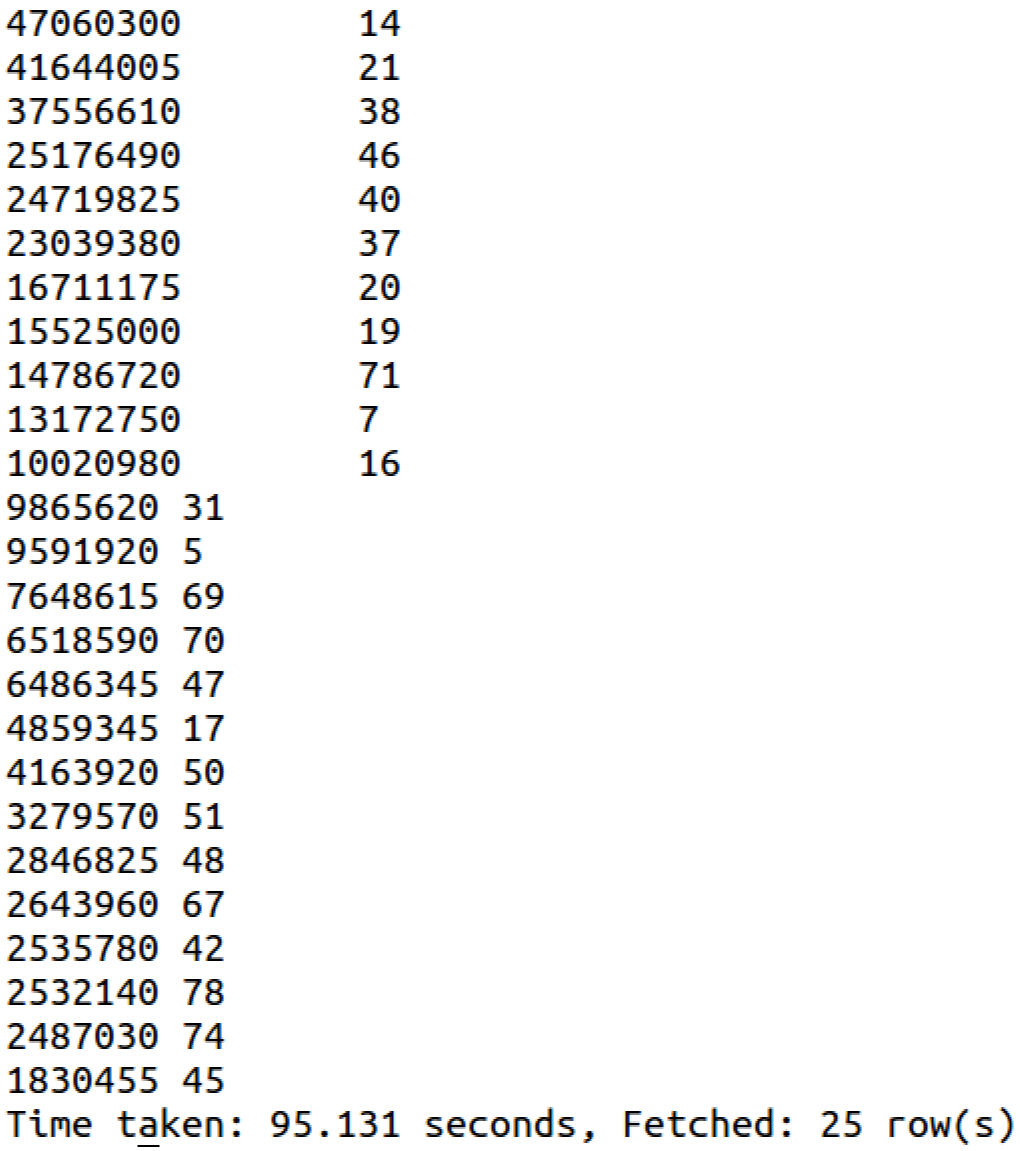
Here we see that the number of parking violations on 29th November is same as that obtained from the master query.



7.10 Violation code with the largest accumulated fines

The objective of this query is to determine which violation code had the largest fines. It multiplies each violation codes' fines by the total number of violations under that code in the parking_violations table. This will help us analyze which violation makes the most money for NYC parking Dept.

```
select sum(violations_code_details.fine) as sm,parking_violations.violation_code from
parking_violations left outer join violations_code_details on
violations_code_details.violation_code = parking_violations.violation_code where
year(parking_violations.issue_date)=2013 group by parking_violations.violation_code order by
sm desc limit 25;
```



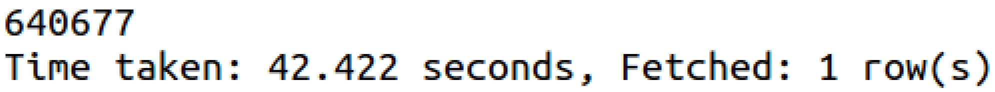
```
47060300        14
41644005        21
37556610        38
25176490        46
24719825        40
23039380        37
16711175        20
15525000        19
14786720        71
13172750        7
10020980        16
9865620 31
9591920 5
7648615 69
6518590 70
6486345 47
4859345 17
4163920 50
3279570 51
2846825 48
2643960 67
2535780 42
2532140 78
2487030 74
1830455 45
Time taken: 95.131 seconds, Fetched: 25 row(s)
```

7.10.1 Verification of query

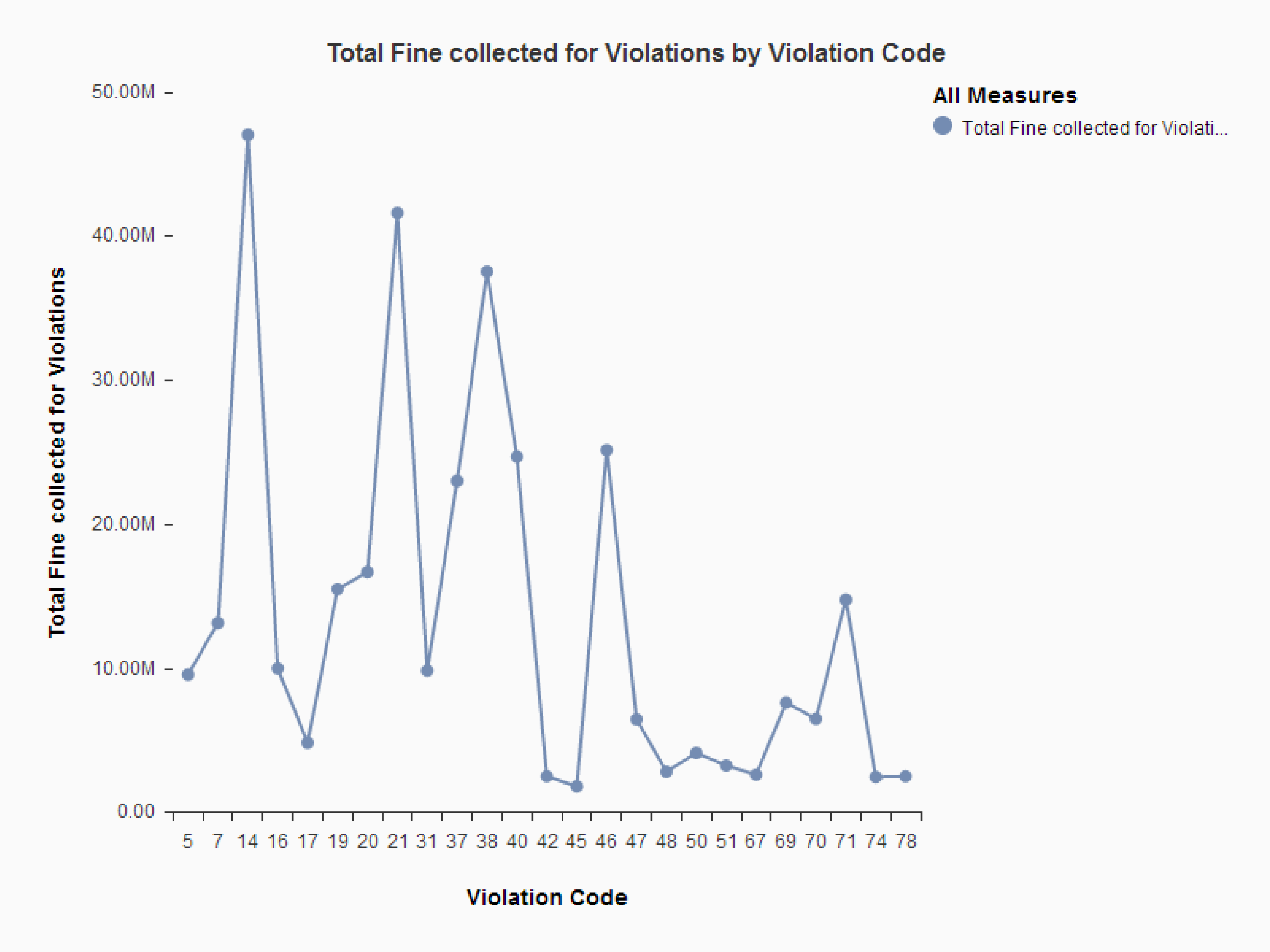
The verification of this query can be done by randomly selecting a violation code. Obtaining the count of violation of that code. And getting the total amount on for that violation.

select count(*) from parking_violations where violation_code=21 and year(issue_date)=2013;

fine for violiaton_code 21 is 65. Hence 640677 * 65 = 41644005.



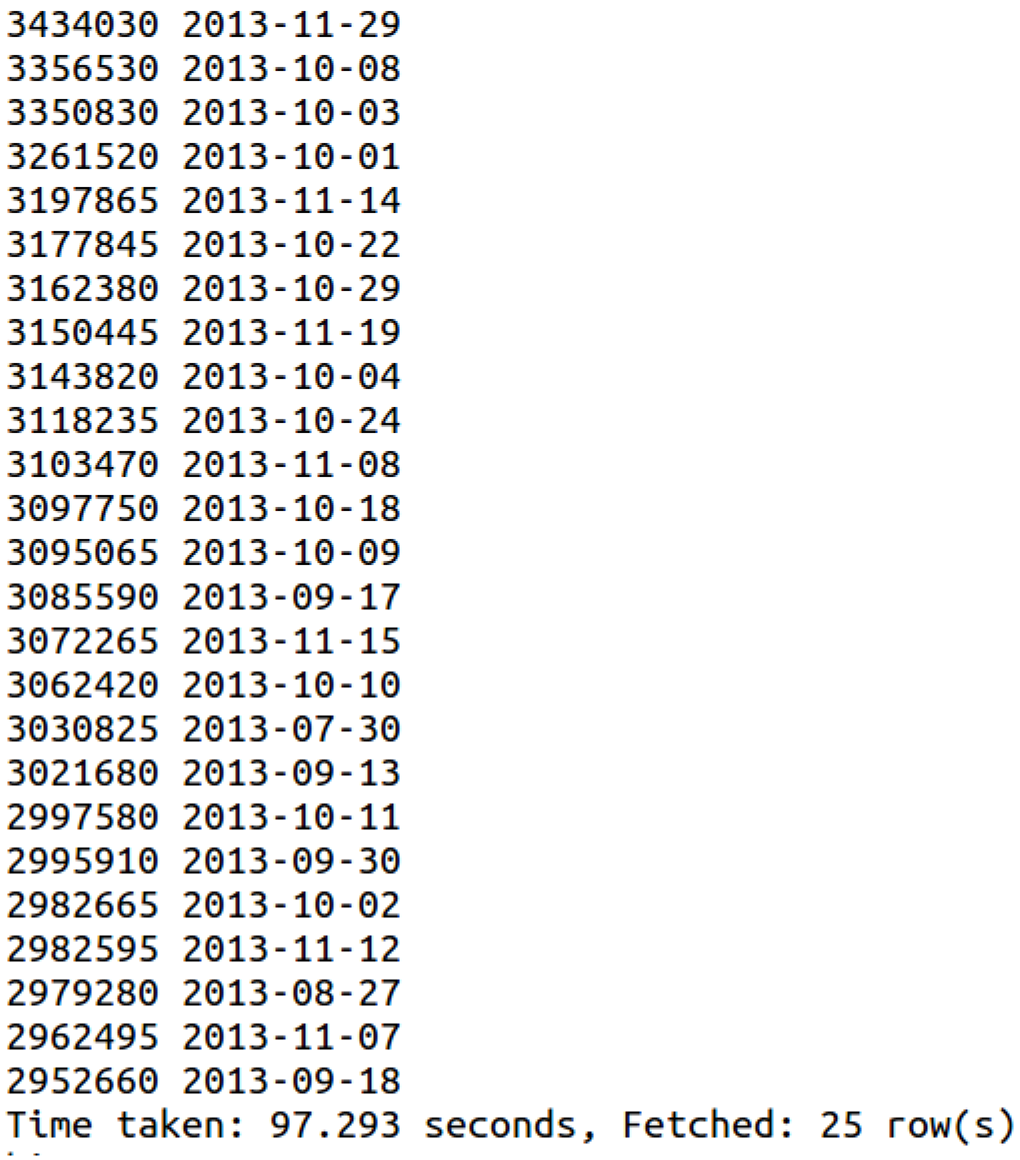
```
640677
Time taken: 42.422 seconds, Fetched: 1 row(s)
```



7.11 Total fine Collection for a day

The objective of the query is to get the result of the maximum fine collected in a single day, in this case 11/29/2013.

```
select sum(violations_code_details.fine) as sm,parking_violations.issue_date from
parking_violations left outer join violations_code_details on
violations_code_details.violation_code = parking_violations.violation_code where
year(parking_violations.issue_date)=2013 group by parking_violations.issue_date order by sm
desc limit 25;
```

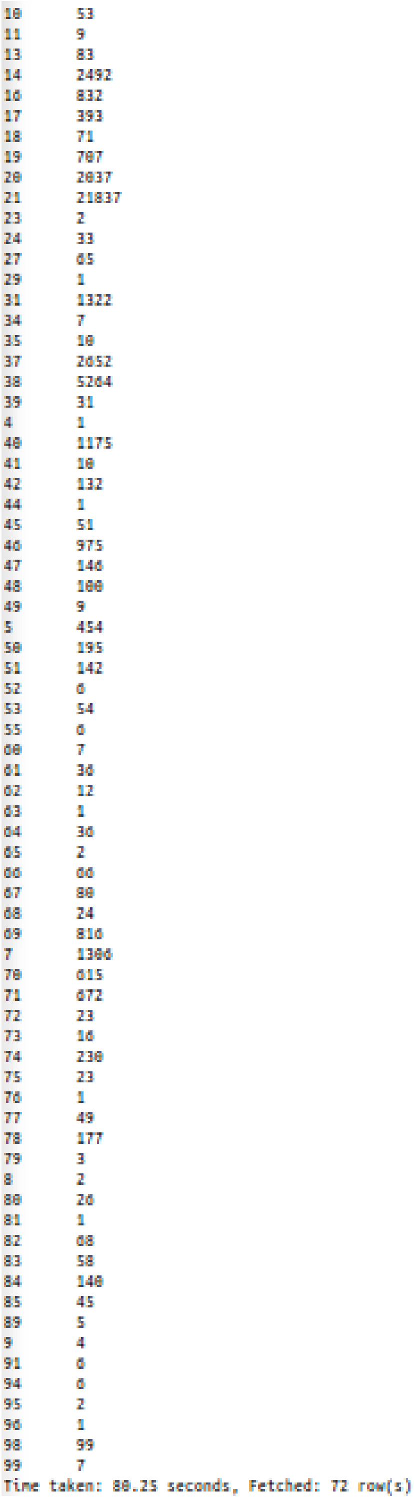


```
3434030 2013-11-29
3356530 2013-10-08
3350830 2013-10-03
3261520 2013-10-01
3197865 2013-11-14
3177845 2013-10-22
3162380 2013-10-29
3150445 2013-11-19
3143820 2013-10-04
3118235 2013-10-24
3103470 2013-11-08
3097750 2013-10-18
3095065 2013-10-09
3085590 2013-09-17
3072265 2013-11-15
3062420 2013-10-10
3030825 2013-07-30
3021680 2013-09-13
2997580 2013-10-11
2995910 2013-09-30
2982665 2013-10-02
2982595 2013-11-12
2979280 2013-08-27
2962495 2013-11-07
2952660 2013-09-18
Time taken: 97.293 seconds, Fetched: 25 row(s)
```

7.11.1 Verification of query

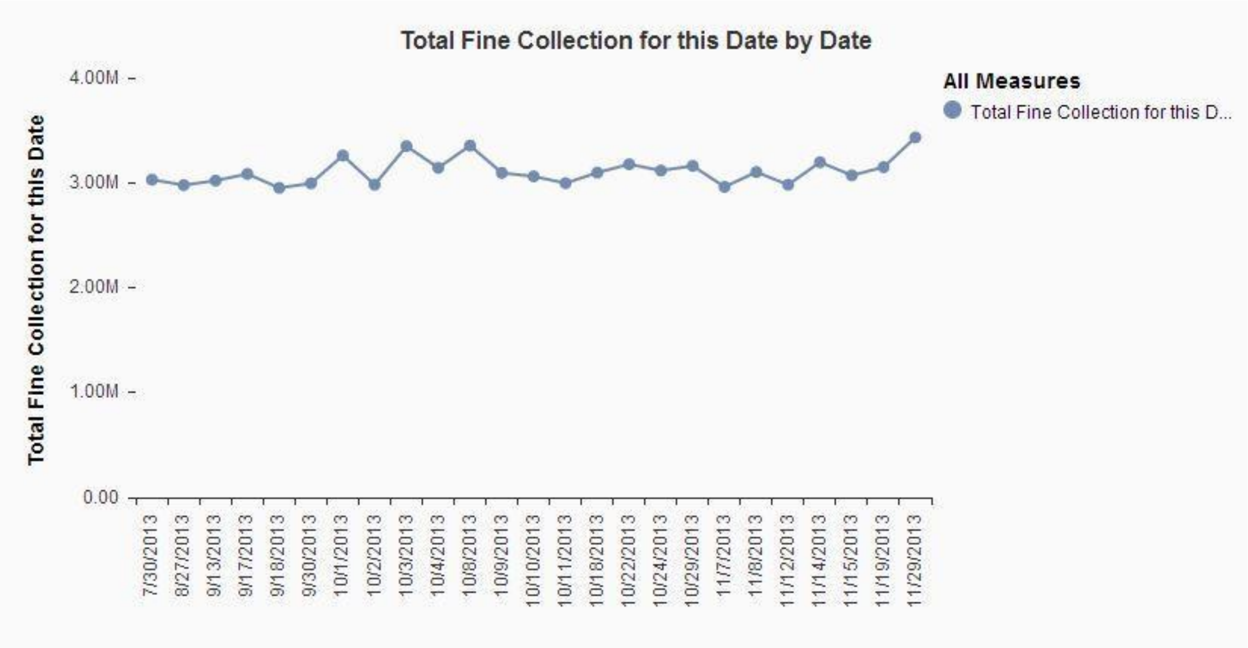
select violation_code , count(*) from parking_violations where issue_date='2013-11-29' group by violation_code order by violation_code desc ;

From the above query we get the violation code and their count for 29th Nov. Now we manually add the total fine under each violation to obtain the total fine.



```
10      53
11      9
13      83
14      2492
16      832
17      393
18      71
19      707
20      2037
21      21837
23      2
24      33
27      65
29      1
31      1322
34      7
35      10
37      2652
38      5264
39      31
4       1
40      1175
41      10
42      132
44      1
45      51
46      975
47      146
48      100
49      9
5       454
50      195
51      142
52      6
53      54
55      6
60      7
61      36
62      12
63      1
64      30
65      2
66      66
67      80
68      24
69      816
7       1306
70      615
71      672
72      23
73      16
74      230
75      23
76      1
77      49
78      177
79      3
8       2
80      26
81      1
82      68
83      58
84      140
85      45
89      5
9       4
91      6
94      6
95      2
96      1
98      99
99      7
Time taken: 80.25 seconds, Fetched: 72 row(s)
```

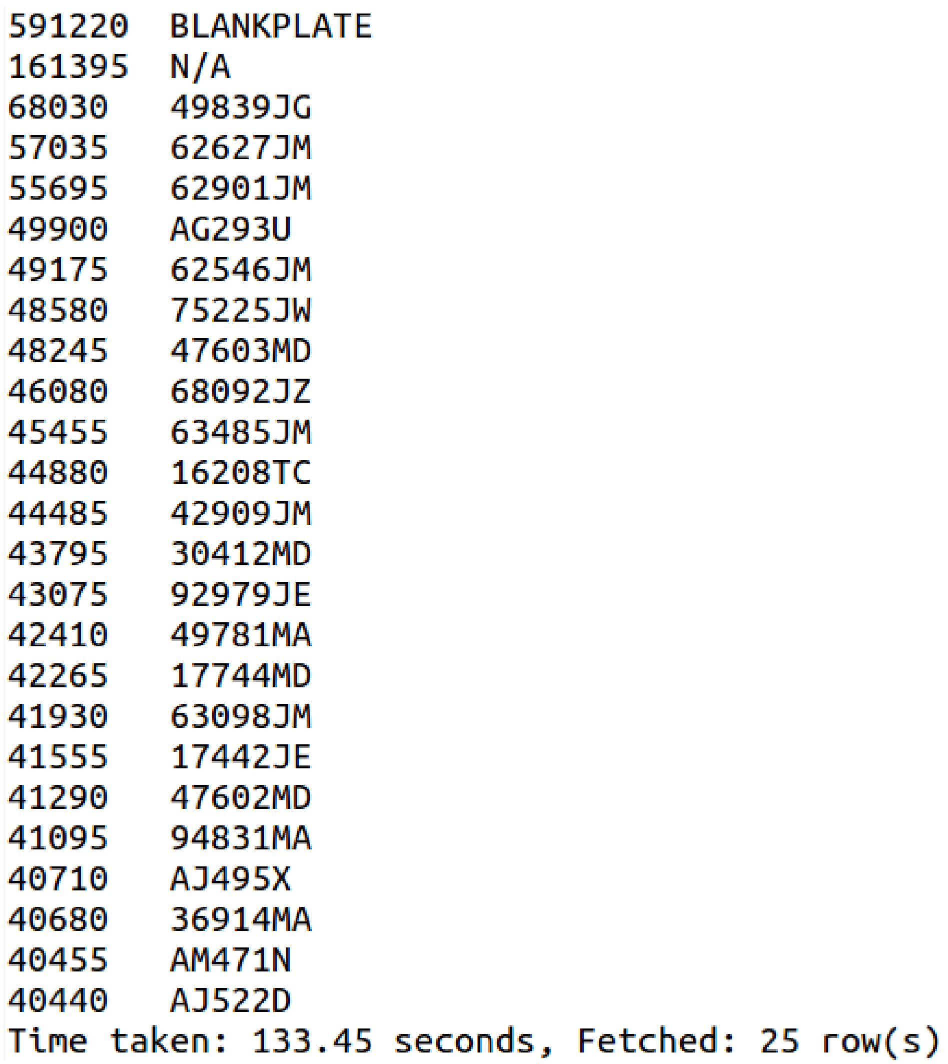
The total fine comes to 3434303 which is same as the one obtained in the main query.



7.12 Vehicle which paid the largest fine.

The objective of this query is to see which car had the largest fine.

```
select sum(violations_code_details.fine) as cnt,parking_violations.plate_id from
parking_violations left outer join violations_code_details on
violations_code_details.violation_code = parking_violations.violation_code group by
parking_violations.plate_id order by cnt desc limit 25;
```



```
591220  BLANKPLATE
161395  N/A
68030   49839JG
57035   62627JM
55695   62901JM
49900   AG293U
49175   62546JM
48580   75225JW
48245   47603MD
46080   68092JZ
45455   63485JM
44880   16208TC
44485   42909JM
43795   30412MD
43075   92979JE
42410   49781MA
42265   17744MD
41930   63098JM
41555   17442JE
41290   47602MD
41095   94831MA
40710   AJ495X
40680   36914MA
40455   AM471N
40440   AJ522D
Time taken: 133.45 seconds, Fetched: 25 row(s)
```

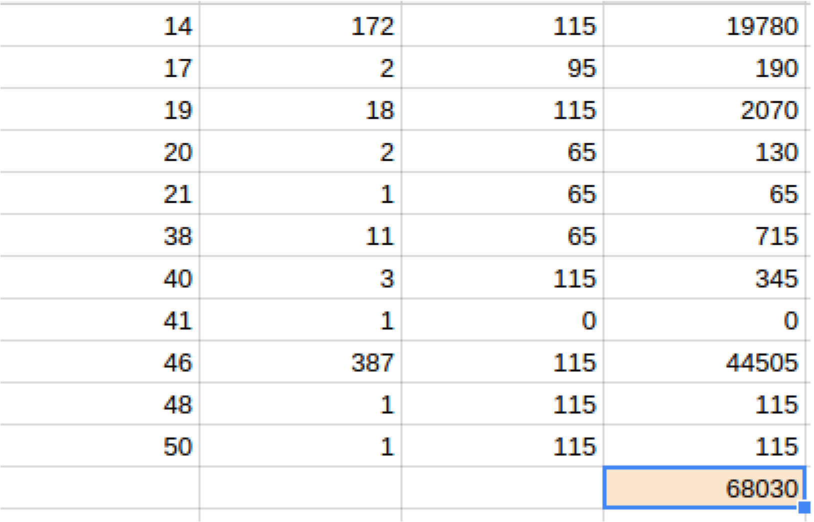
7.12.1 Verification of query

Verification of the result can be done by selecting a random vehicle and then calculating the total fine manually in excel.

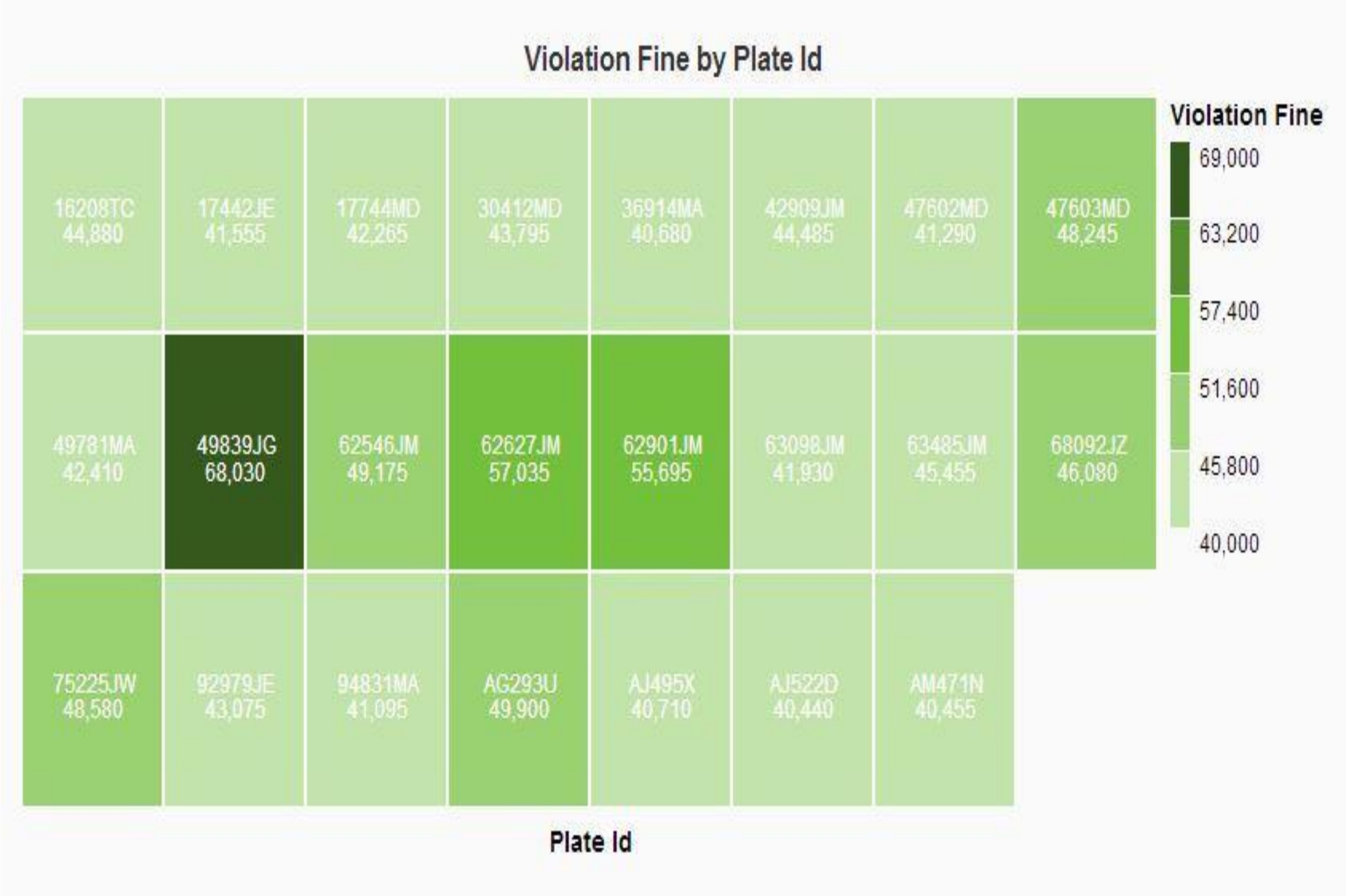
We select the random vehicle with plate id 49839JG.The following query gives us all the violations by this vehicle.

select violation_code,count(*) from parking_violations where plate_id='49839JG' group by violation_code;

Using the result of this query we will obtain the total fine in excel.



14	172	115	19780
17	2	95	190
19	18	115	2070
20	2	65	130
21	1	65	65
38	11	65	715
40	3	115	345
41	1	0	0
46	387	115	44505
48	1	115	115
50	1	115	115
			68030



7.13 Speed Camera Violation by Plate type.

The objective of this query is to find out vehicle belonging to which group is making most violations. Plate type is used as an attribute to identify to which vehicle group the vehicle belongs.

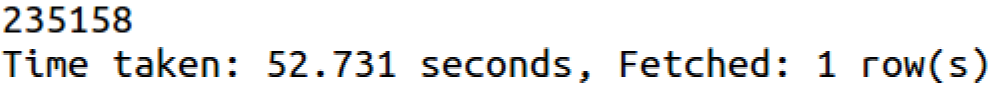
```
select plate_type ,count(summons_number) as cnt from speed_cam_tickets where
registration_state='NY' group by plate_type order by cnt desc limit 25;
```

```
PAS     4521950
COM     1868302
OMT     235158
SRF     95557
OMS     74988
IRP     53175
999     27910
MOT     23176
TRC     22335
OMR     16188
ORG     12721
MED     12343
OML     10517
SPO     7042
DLR     4699
RGL     4281
TOW     4223
SRN     4075
SCL     3596
TRA     3485
PSD     3457
VAS     3079
ITP     2549
CMH     2347
TRL     2179
Time taken: 82.499 seconds, Fetched: 25 row(s)
```

7.13.1 Verification of the query

For verifying the query we can randomly check some plate-type for the total violations. Lets check the total violations for plate_type OMT.

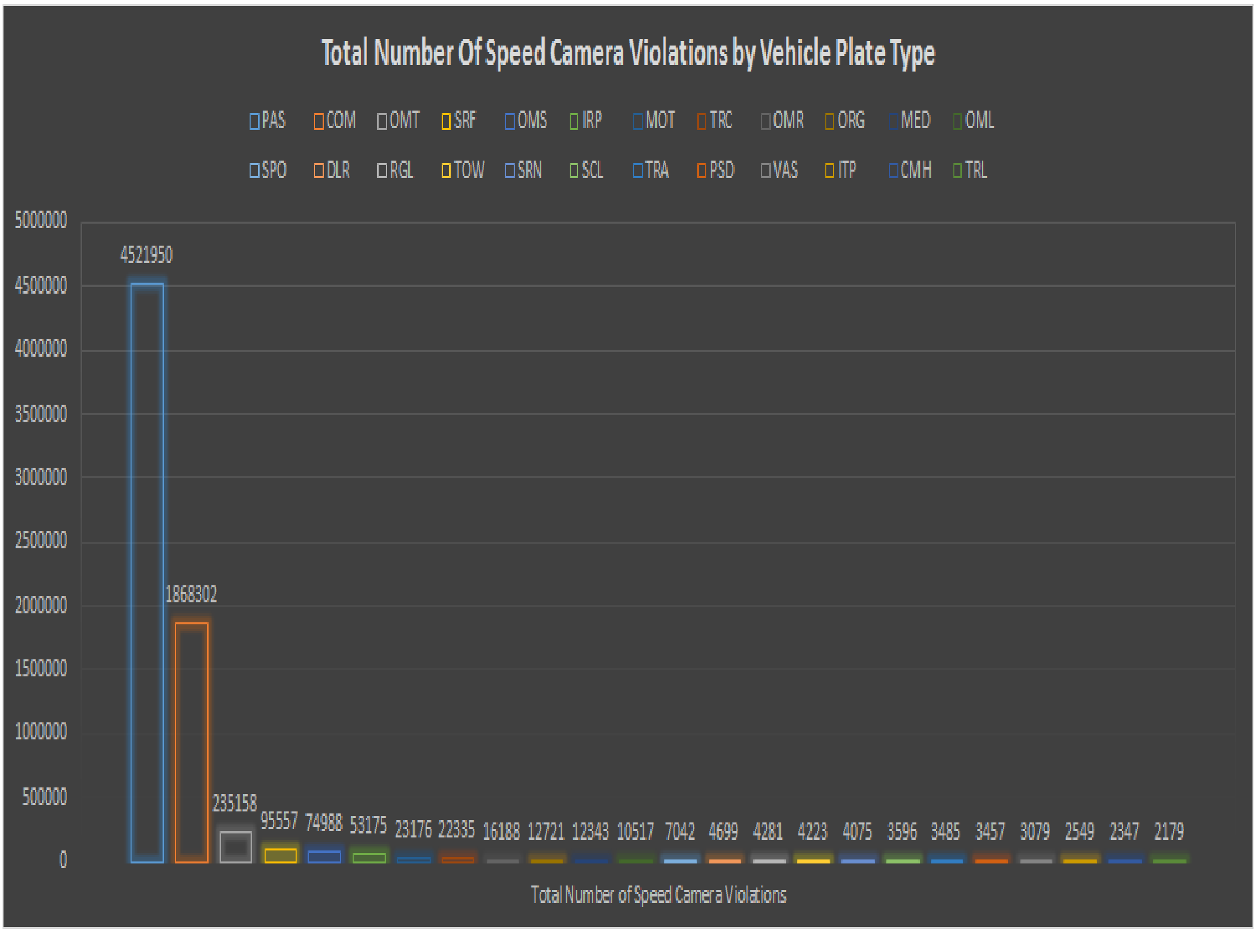
select count(distinct summons_number) as cnt from speed_cam_tickets where registration_state='NY' and plate_type='OMT';



```
235158
Time taken: 52.731 seconds, Fetched: 1 row(s)
```

Here we see that the total violations are same as the master query.

Visualization:



8. Conclusion:

Analysis of parking violations in New York have showed pattern in violation code, days and car type. This can help New Yorkers in in avoiding violations. In the course of the project we learnt using hive over Hadoop and Gogrid cluster. On Data analysis part we got a idea of asking the right question to exact meaningful information from the data. We also learnt the ways to visualize data and present a story of the analysis to make your research worthwhile.