

Analysis of Parking Violations in New York City

Justin Nichols, Bryce Lee



Motivation and Project Description

- Cities have obscured parking laws aimed to make big profits.
- In 2015, NYC collected \$565 million in traffic and parking penalties.
- Most parking tickets are given out in high quantities and cost over \$100.

Our Big Data Application:

- To reduce chances of getting fined.
- Useful to know which types of vehicles more often get fined.
- Identify correlations between which types of vehicles received the costliest violations depending on the season.
 - types: registration state, plate type, vehicle body type, vehicle make, vehicle color, vehicle year
 - seasons: summer, fall, winter, spring
- Compare and contrast development process and performance between Hadoop MapReduce and PySpark.

PARKING VIOLATION

NOTICE THIS VEHICLE IS IMPROPERLY PARKED
VIOLATIONS ARE AS FOLLOWS:

<input type="checkbox"/> VEHICLE HAS NO VALID PARKING PERMIT	<input type="checkbox"/> BLOCKING DRIVEWAY OR ACCESS
<input type="checkbox"/> PARKED IN NO PARKING AREA/SPACE	<input type="checkbox"/> BLOCKING OTHER VEHICLE
<input type="checkbox"/> PARKED IN FIRE LANE	<input type="checkbox"/> PARKED IN 2 SPACES
<input type="checkbox"/> PARKED IN HANDICAP SPACE	<input type="checkbox"/> OTHER _____
<input type="checkbox"/> PARKED IN RESERVED OR ASSIGNED SPACE	

**THIS VEHICLE'S DESCRIPTION HAS BEEN PERMANENTLY RECORDED. ANY ADDITIONAL
INFRACTIONS OF OUR REGULATIONS COULD RESULT IN TOWING AT VEHICLE OWNER'S
EXPENSE AND REVOCATION OF PARKING PRIVILEGES.**

License No. _____ State _____ Permit No. _____	Date: _____
Vehicle Make / Model _____ Color _____	Time: _____
Driver's Name (if known) _____	Location: _____
	Issued By: _____

Data Frameworks & Cloud Computing Platform

Frameworks:

- Apache Hadoop MapReduce.
- Apache Spark (PySpark interface).
- Apache Hue to view and analyze the results.



Platform:

- Cloudera virtual machine in pseudo-distributed mode.



Environments:

- Eclipse for Hadoop MapReduce.
- Basic text editor and Linux terminal for PySpark.



Dataset

- Contains parking violation data from NYC DMV.
- Approximately 9GB, across 4 CSV files.
- Around 42.3 million entries, each with 51 comma separated values.
- Columns gathered: vehicle body type, vehicle make, vehicle year, vehicle color, registration state, plate type, issue date, violation code.



Hadoop MapReduce Components

Driver:

- Utilized ToolRunner with the argument “taskType”.
 - <taskType> = registrationState, plateType, vehicleBodyType, vehicleMake, vehicleColor, vehicleYear
- Utilized LocalJobRunner.
- Linked Mapper, Reducer, and Partitioner.
- Set reduce tasks to 4.
- All output types are of type “Text”.

Partitioner:

- Parsed out the season from Mapper value.
- Distributed to one of the 4 Reducers (one for each season).

```
Map-Reduce Framework
  Map input records=40098
  Map output records=38905
  Map output bytes=598733
  Map output materialized bytes=676663
  Input split bytes=668
  Combine input records=0
  Combine output records=0
  Reduce input groups=554
  Reduce shuffle bytes=676663
  Reduce input records=38905
  Reduce output records=225
  Spilled Records=77810
  Shuffled Maps =20
  Failed Shuffles=0
  Merged Map outputs=20
  GC time elapsed (ms)=554
  CPU time spent (ms)=16210
  Physical memory (bytes) snapshot=1579941888
  Virtual memory (bytes) snapshot=8392187904
  Total committed heap usage (bytes)=935460864

Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0

File Input Format Counters
  Bytes Read=8147676
File Output Format Counters
  Bytes Written=3850
Total execution time: 84320
[training@localhost ~]$
```

Hadoop MapReduce Components (cont.)

Mapper:

- Get <taskType> argument and validate it.
- Split line using regex and filter out lines with missing or misformatted data.
- Gather the <taskType>, “issueDate”, and “violationCode” columns.
- Translate “issueDate” month into a season.
- Emit pair (<taskType>, [season, violationCode]).

Reducer:

- Translate violation code into dollar amount.
- Accumulate count and total dollars, find average.
- Filter out keys with less than 5 values.
- Emit pair (<taskType>, [count, totalDollars, average]).

<input type="checkbox"/>		_SUCCESS
<input type="checkbox"/>		part-00000
<input type="checkbox"/>		part-00001
<input type="checkbox"/>		part-00002
<input type="checkbox"/>		part-00003

PySpark Components

- Used RDDs with transformations and actions.
- Gathered <task_type> data.
- Similar operations, but different order.
- Filter, map, reduceByKey, then partitionBy.
- Emit pair (<task_type>, [count, total_dollars, average]).



```
File Edit View Search Terminal Help
training@localhost:~$
22/04/30 17:07:20 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 10.0 (TID 34) in 207 ms on localhost (1/4)
22/04/30 17:07:20 INFO storage.ShuffleBlockFetcherIterator: Getting 5 non-empty blocks out of 5 blocks
22/04/30 17:07:20 INFO storage.ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
22/04/30 17:07:20 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
22/04/30 17:07:20 INFO pyhoof.PythonRDD: Times: total = 45, boot = 175, init = 215, finish = 1
22/04/30 17:07:20 INFO output.FileOutputCommitter: Saved output of task 'attempt_202204301707_0010_m_000003_35' to hdfs://
/localhost:8020/user/training/output/_temporary/0/task_202204301707_0010_m_000003_35
22/04/30 17:07:20 INFO spark.SparkMapOutputWriter: attempt_202204301707_0010_m_000003_35: Committed
22/04/30 17:07:20 INFO executor.Executor: Finished task 3.0 in stage 10.0 (TID 35): 1862 bytes result sent to driver
22/04/30 17:07:21 INFO scheduler.DAGScheduler: Stage 10 (saveAsTextFile at NativeMethodAccessorImpl.java:2) finished in
1.142 s
22/04/30 17:07:21 INFO scheduler.DAGScheduler: Job 5 finished: saveAsTextFile at NativeMethodAccessorImpl.java:2, took 2
.481824 s
22/04/30 17:07:21 INFO scheduler.TaskSetManager: Finished task 3.0 in stage 10.0 (TID 35) in 251 ms on localhost (4/4)
22/04/30 17:07:21 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 10.0, whose tasks have all completed, from pool
1 (Total, execution time: ~ 37.69003308979187)
22/04/30 17:07:21 INFO spark.ContextCleaner: Cleaned shuffle 1
22/04/30 17:07:21 INFO spark.ContextCleaner: Cleaned shuffle 2
22/04/30 17:07:21 INFO storage.BlockManager: Removing broadcast 9
22/04/30 17:07:21 INFO storage.MemoryStore: Block broadcast 9 of size 21640 dropped from memory (free 279426591)
22/04/30 17:07:21 INFO storage.BlockManager: Removing block broadcast 9 piece0
22/04/30 17:07:21 INFO storage.MemoryStore: Block broadcast 9 piece0 of size 13854 dropped from memory (free 2794448355)
22/04/30 17:07:21 INFO storage.BlockManagerInfo: Removed broadcast 9 piece0 on localhost:40569 in memory (size: 13.5 KB,
free: 267.2 MB)
22/04/30 17:07:21 INFO storage.BlockManagerMaster: Updated info of block broadcast 9 piece0
22/04/30 17:07:21 INFO spark.ContextCleaner: Cleaned broadcast 9
22/04/30 17:07:21 INFO storage.BlockManager: Removing broadcast 10
22/04/30 17:07:21 INFO storage.MemoryStore: Block broadcast 10 of size 6888 dropped from memory (free 279447155)
22/04/30 17:07:21 INFO storage.BlockManager: Removing block broadcast 10 piece0
22/04/30 17:07:21 INFO storage.MemoryStore: Block broadcast 10 piece0 of size 4383 dropped from memory (free 279451538)
22/04/30 17:07:21 INFO storage.BlockManagerInfo: Removed broadcast 10 piece0 on localhost:40569 in memory (size: 4.3 KB,
free: 267.2 MB)
22/04/30 17:07:21 INFO storage.BlockManagerMaster: Updated info of block broadcast 10 piece0
22/04/30 17:07:21 INFO spark.ContextCleaner: Cleaned broadcast 10
22/04/30 17:07:21 INFO storage.BlockManager: Removing broadcast 11
22/04/30 17:07:21 INFO storage.MemoryStore: Block broadcast 11 of size 17376 dropped from memory (free 279588914)
22/04/30 17:07:21 INFO storage.BlockManager: Removing block broadcast 11 piece0
22/04/30 17:07:21 INFO storage.MemoryStore: Block broadcast 11 piece0 of size 47897 dropped from memory (free 279636811)
22/04/30 17:07:21 INFO storage.BlockManagerInfo: Removed broadcast 11 piece0 on localhost:40569 in memory (size: 48.8 KB,
free: 267.2 MB)
22/04/30 17:07:21 INFO storage.BlockManagerMaster: Updated info of block broadcast 11 piece0
22/04/30 17:07:21 INFO spark.ContextCleaner: Cleaned broadcast 11
training@localhost ~$
```

Registration State

- Minnesota, Indiana, Oklahoma, Idaho, New Jersey, Arizona, and Illinois had high ticket prices.
- Government plates and foreign plates (Canadian and Mexico) also had high violation costs.

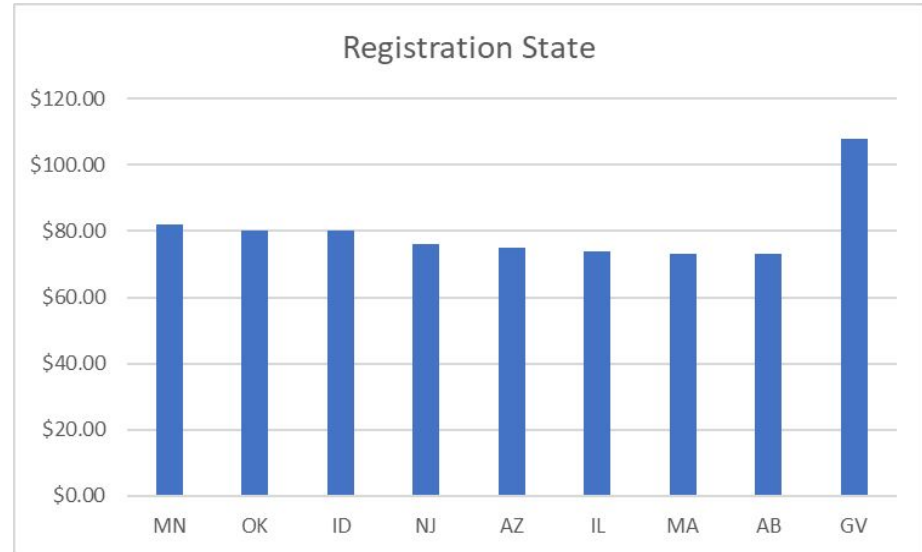
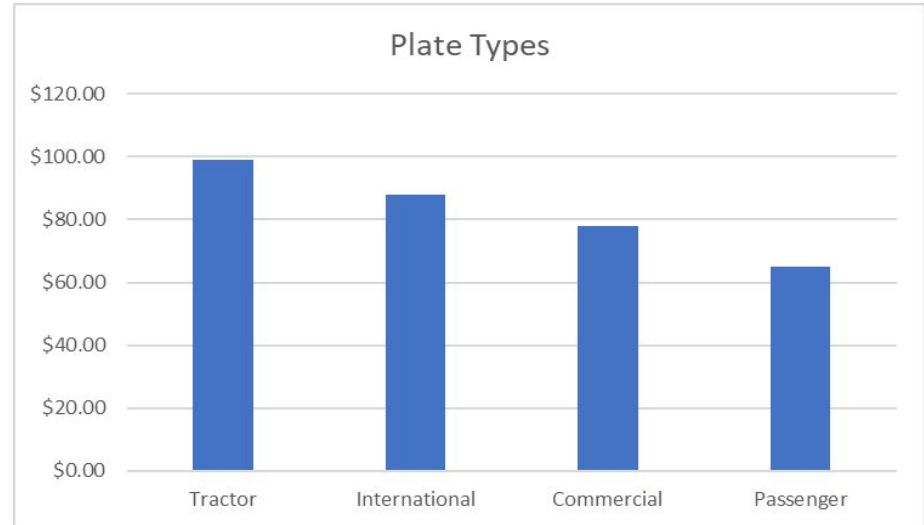


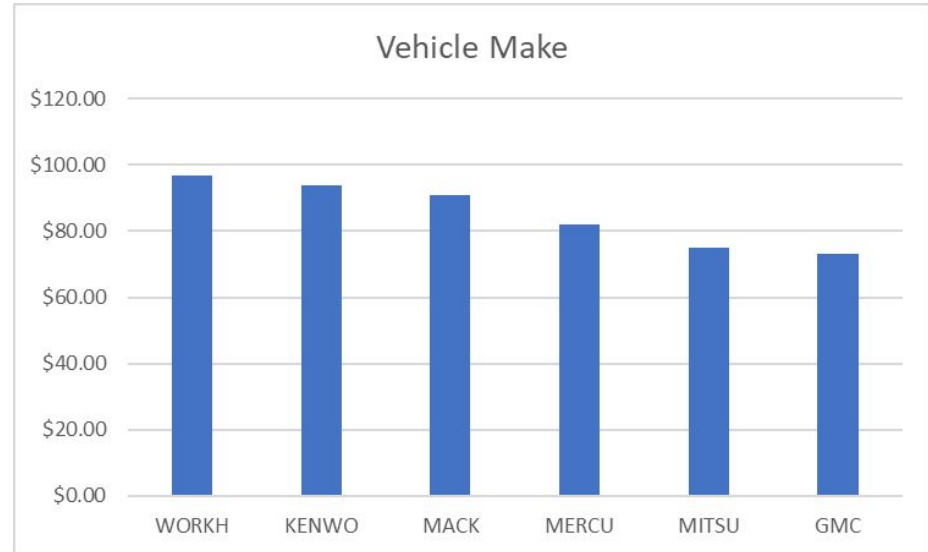
Plate type

- Passenger and commercial had the cheapest violations.
- Tractors, ATDs, and farm vehicles had the costliest violations.
- International and government plates also had high ticket prices.



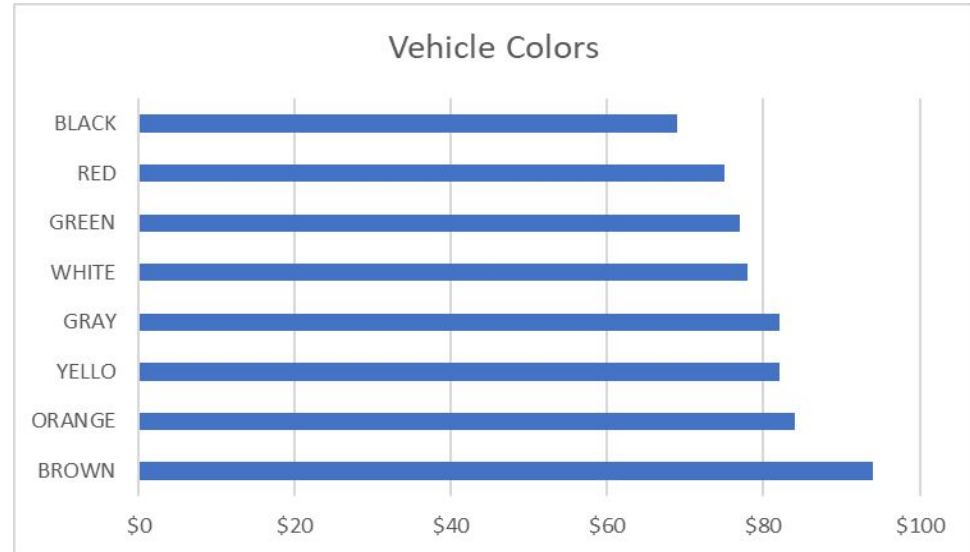
Vehicle Make

- Commercial vehicles brands were ticketed higher than passenger ones, such as Workhorse, Kentwood, and Mack.
- American based brands such as GMC, Mercury, Dodge, and Ford were ticketed higher than foreign made brands.



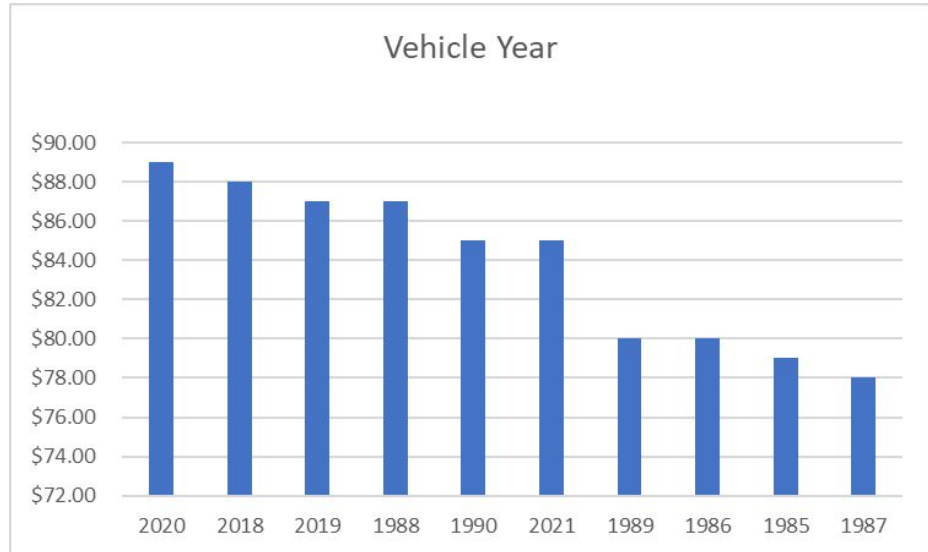
Vehicle Color

- Brown, orange, yellow, green, red, and multi-colored vehicles had higher violation costs.
- Among the three most popular colors (white, black, and gray), gray was the highest, then white, and finally black.



Vehicle Year

- New cars made from the 2010s had high ticket prices.
- Vehicles made between the years 1985-1997 correlated to a \$10-\$20 higher ticket price than those made before or after those years.



Development Results



- Both develop in an Agile environment.

Hadoop MapReduce:

- LocalJobRunner made testing easier.
- Having 4 files was more tedious.
- Useful error messages.

PySpark:

- No IDE made for cumbersome environment.
- Chained operations was beneficial.
- Having only 1 file was easy.
- Difficult error messages.

Performance Results



- PySpark was quicker than Hadoop MapReduce.

Hadoop MapReduce:

- Entire dataset: 2631.267 seconds
- Trimmed dataset: 84.32 seconds
- Slower due to many disk read and write operations.

```
Total execution time: 84320  
[training@localhost ~]$ █
```

Spark:

- Entire dataset: 512.717 seconds
- Trimmed dataset: 17.093 seconds
- Faster due to in-memory read and write operations and lazy evaluation.

```
('Total execution time: ', 17.09265398979187
```

Conclusion



- Costlier tickets given out in winter and spring.
- PySpark development was easier and performed better than Hadoop MapReduce.
- Error messages were better in Hadoop MapReduce.

Challenges:

- Erroneous data.
- Unknown data and many alterations of the same key.
 - e.g., black was abbreviated as BLCK, BL, BK, BCK.
- Limited VM memory size and disk space.

Improvements:

- Filtering process.
- New distributed environment.