Predictive Analysis to avoid parking tickets in NYC

Akhil Padgilwar (A20427219)
Ganga Gudi (A20428842)
Ridhima Bhalerao (A20422550)
Sagar Ippili (A20417999)

Introduction

Parking tickets are a bit difficult to avoid in some circumstances and hence there needs to be a way to avoid them

Getting a parking ticket in New York City is a very common situation and easily makes the driver to spend more than $100

People cannot avoid using their cars or misread the signs even after carefully following a few useful tips

We acquired motivation from ‘Ticket Wiper’ application to build a project on recommender system which would assist drivers in avoiding parking tickets.

We have also generated different analytical queries that provide insights about the data

Data Collection

Data collected from NYC OpenData.

We have chosen the files for years 2016 to 2019 containing data of parking tickets.

Data columns include Vehicle Make, Plateid, State, Plate_Type, Issue_Date, Violation_Code, Street_Name, etc.

Data Pre-processing

General data pre-processing

Down sampling of 33.5M of data has been made as the data was present in separate files as per year.

We achieved this using Python script processed on Spark.

Recommender system specific pre-processing

Data Import and merge

Selecting columns and calculating count

Indexing the ‘Street Name’ column as a new ‘Street Code’ column

Scaling/Normalizing the ‘count’ column

Visualization-specific pre-processing

Unnecessary columns that had no data in them were ignored for appropriate visualization operation.

Entries without Plate IDs have also been ignored.

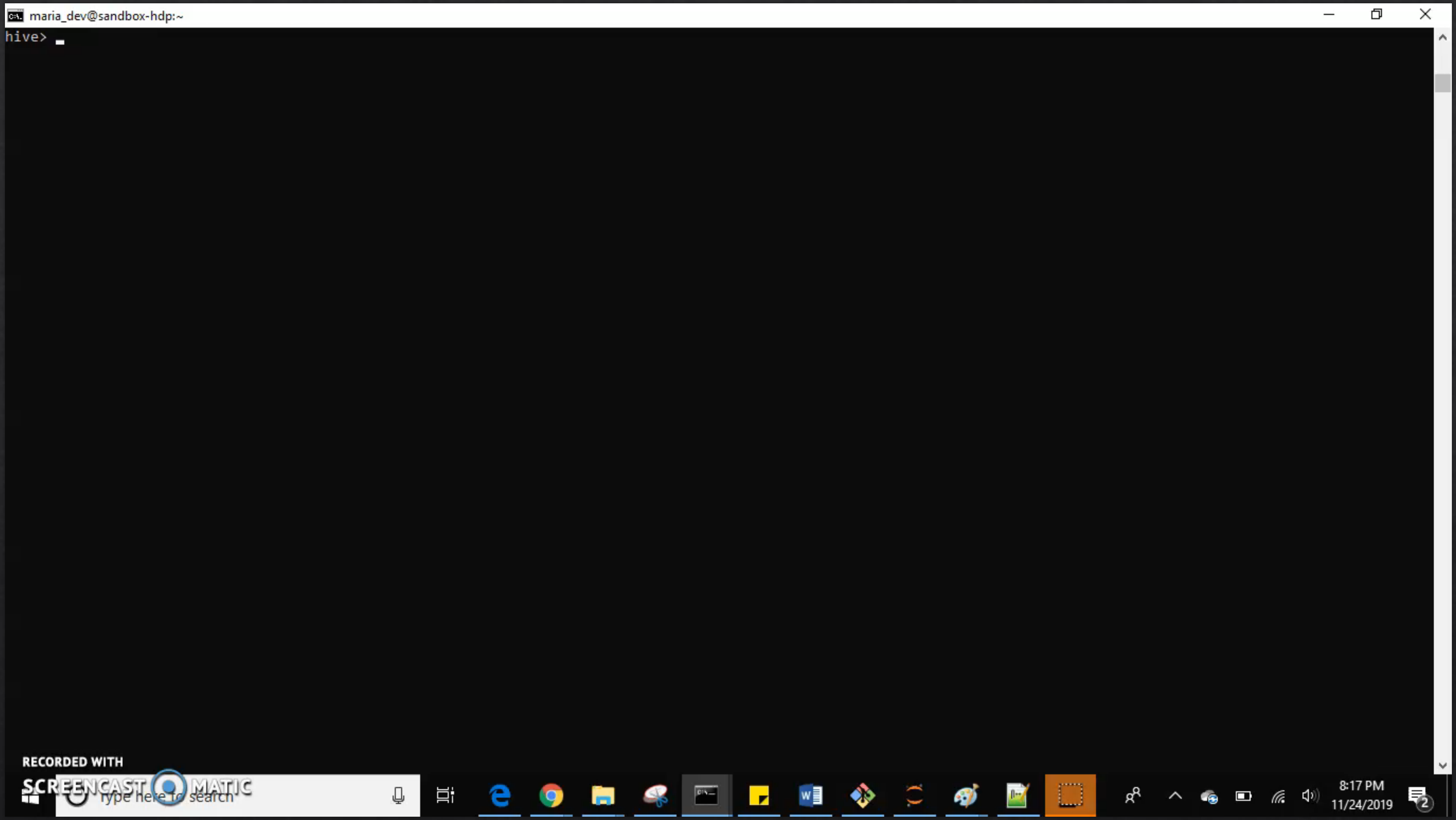
Registration state contained dirty value ‘99’ has also been eliminated.

Data Import in HDFS

We upload the data files from local storage into HDFS for performing analysis for the project and generate different analytical queries.

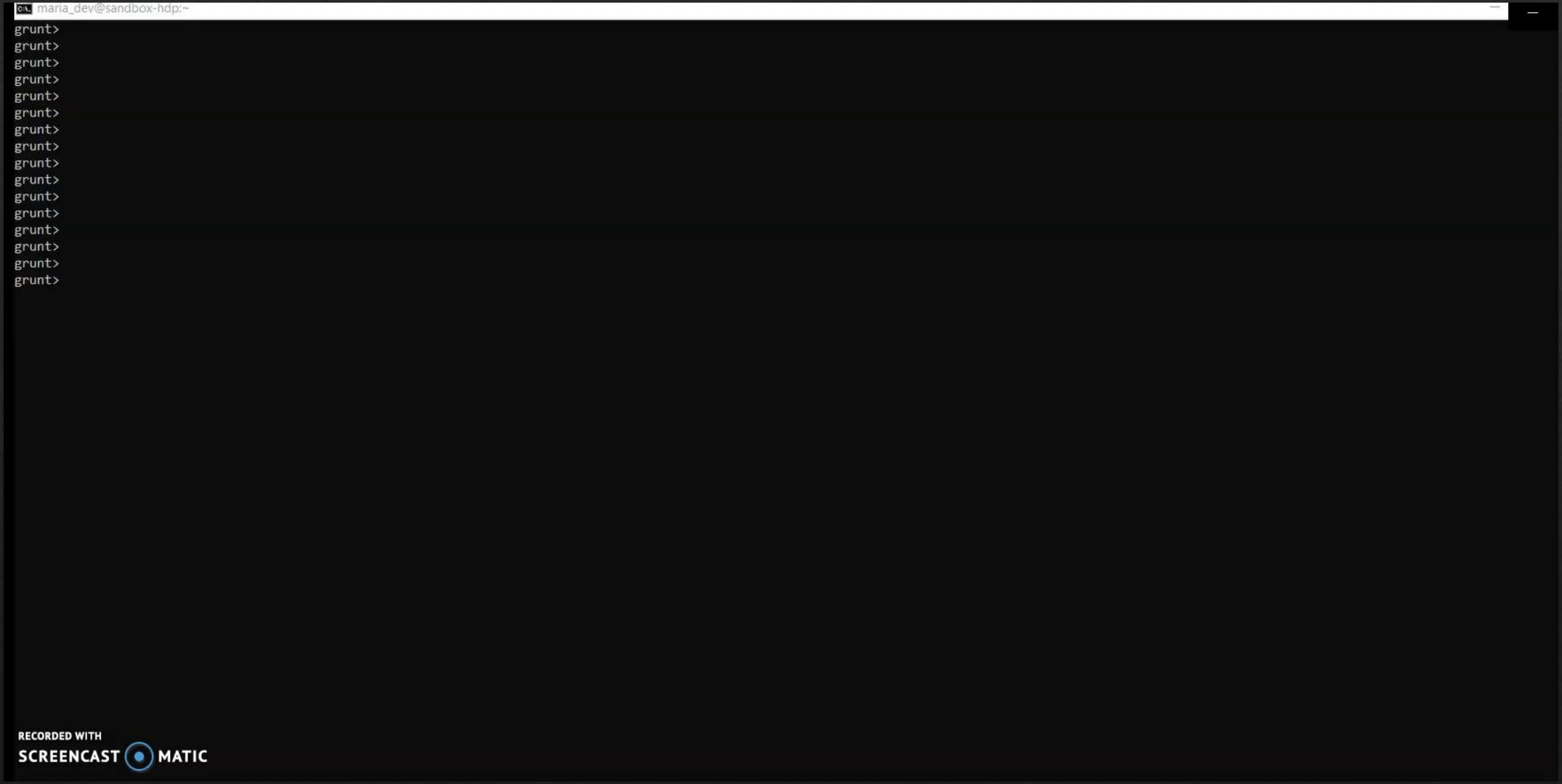
Hadoop shell commands are used for importing data.

Demo for Analytics Performed on Hive



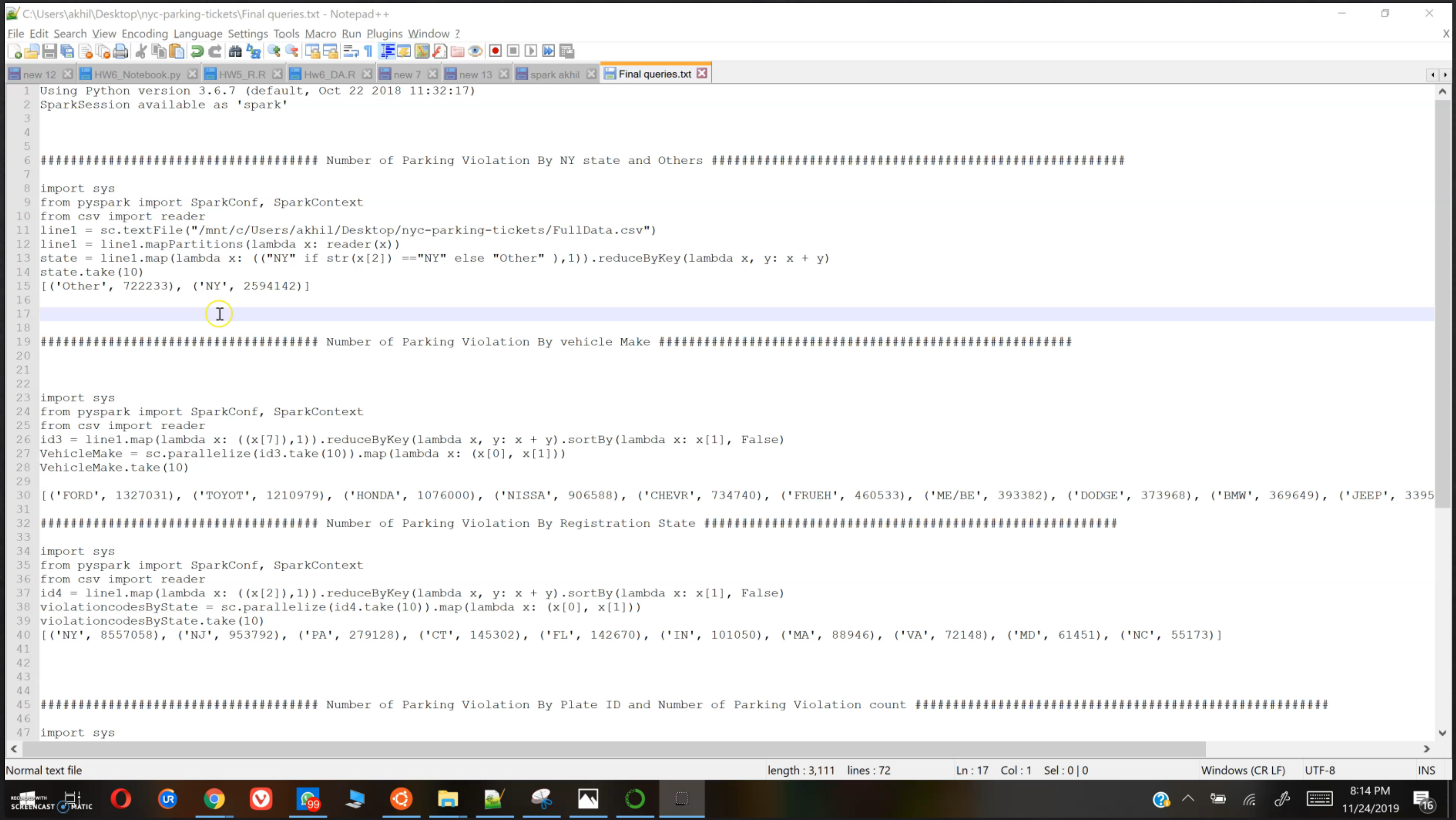
```
hive> _
```

Demo for Analytics Performed on Pig



```
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
```

Demo for Analytics Performed on PySpark


C:\Users\akhil\Desktop\nyc-parking-tickets\Final queries.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 12
HW6_Notebook.py
HW5_R.R
Hw6_DA.R
new 7
new 13
spark akhil
Final queries.txt
Normal text file
length : 3,111 lines : 72
Ln : 17 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS
8:14 PM
11/24/2019

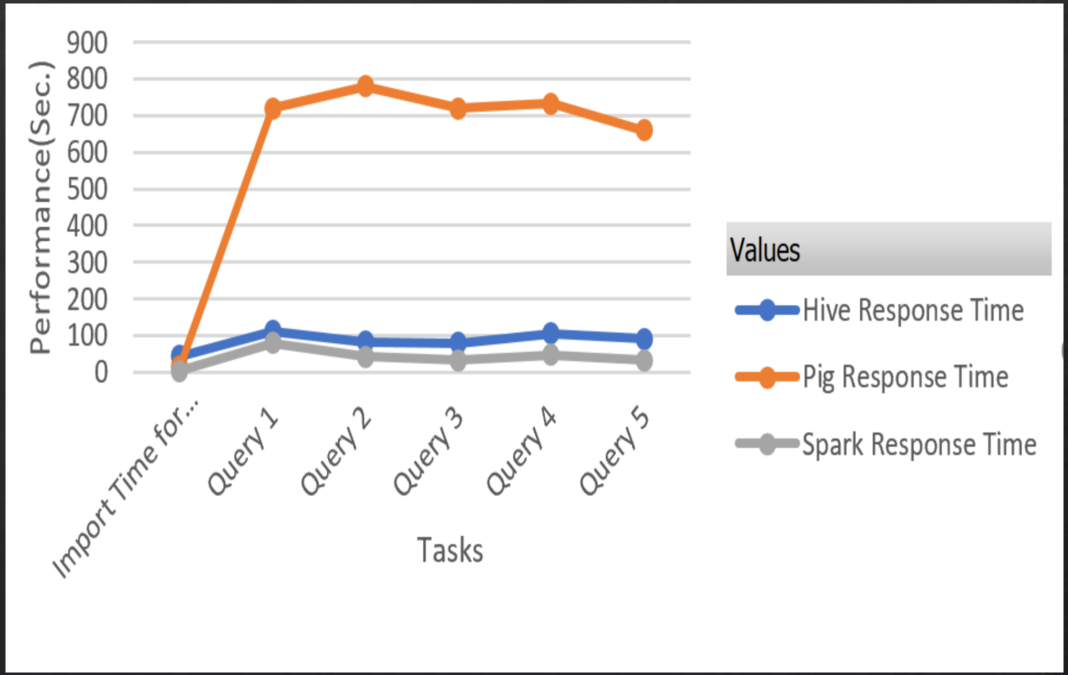
```
Using Python version 3.6.7 (default, Oct 22 2018 11:32:17)
SparkSession available as 'spark'



##################################### Number of Parking Violation By NY state and Others #######################################################

import sys
from pyspark import SparkConf, SparkContext
from csv import reader
line1 = sc.textFile("/mnt/c/Users/akhil/Desktop/nyc-parking-tickets/FullData.csv")
line1 = line1.mapPartitions(lambda x: reader(x))
state = line1.map(lambda x: (("NY" if str(x[2]) =="NY" else "Other" ),1)).reduceByKey(lambda x, y: x + y)
state.take(10)
[('Other', 722233), ('NY', 2594142)]



##################################### Number of Parking Violation By vehicle Make #######################################################



import sys
from pyspark import SparkConf, SparkContext
from csv import reader
id3 = line1.map(lambda x: ((x[7]),1)).reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], False)
VehicleMake = sc.parallelize(id3.take(10)).map(lambda x: (x[0], x[1]))
VehicleMake.take(10)

[('FORD', 1327031), ('TOYOT', 1210979), ('HONDA', 1076000), ('NISSA', 906588), ('CHEVR', 734740), ('FRUEH', 460533), ('ME/BE', 393382), ('DODGE', 373968), ('BMW', 369649), ('JEEP', 3395

##################################### Number of Parking Violation By Registration State #######################################################

import sys
from pyspark import SparkConf, SparkContext
from csv import reader
id4 = line1.map(lambda x: ((x[2]),1)).reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], False)
violationcodesByState = sc.parallelize(id4.take(10)).map(lambda x: (x[0], x[1]))
violationcodesByState.take(10)
[('NY', 8557058), ('NJ', 953792), ('PA', 279128), ('CT', 145302), ('FL', 142670), ('IN', 101050), ('MA', 88946), ('VA', 72148), ('MD', 61451), ('NC', 55173)]




##################################### Number of Parking Violation By Plate ID and Number of Parking Violation count #######################################################

import sys
```

Comparison for Data Analysis

Apache Hive

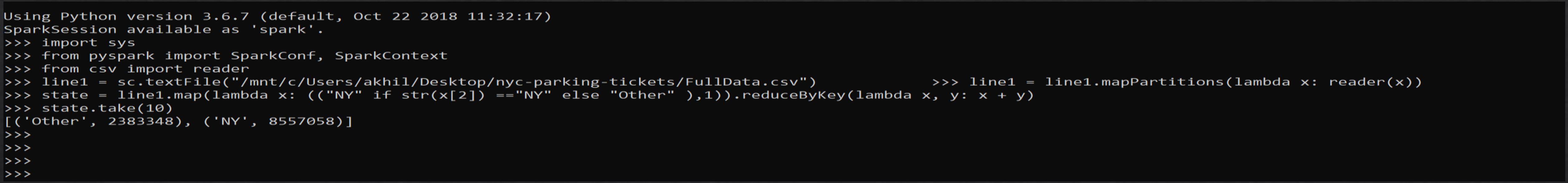
Apache Pig

Apache Spark



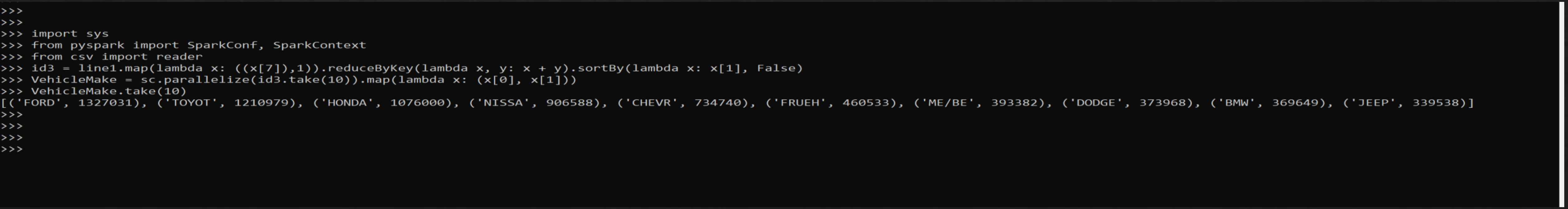
Analyzing the data

Depending on execution and response time of the queries, we choose to go ahead with the performance of Apache Spark. Examples of results are as follows:



```
Using Python version 3.6.7 (default, Oct 22 2018 11:32:17)
SparkSession available as 'spark'.
>>> import sys
>>> from pyspark import SparkConf, SparkContext
>>> from csv import reader
>>> line1 = sc.textFile("/mnt/c/Users/akhil/Desktop/nyc-parking-tickets/FullData.csv")           >>> line1 = line1.mapPartitions(lambda x: reader(x))
>>> state = line1.map(lambda x: (("NY" if str(x[2]) =="NY" else "Other" ),1)).reduceByKey(lambda x, y: x + y)
>>> state.take(10)
[('Other', 2383348), ('NY', 8557058)]
>>>
>>>
>>>
>>>
```

<u>Example 1:</u> Result of total number of violations based on whether the vehicle registration was in NY or elsewhere



```
>>>
>>>
>>> import sys
>>> from pyspark import SparkConf, SparkContext
>>> from csv import reader
>>> id3 = line1.map(lambda x: ((x[7]),1)).reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], False)
>>> VehicleMake = sc.parallelize(id3.take(10)).map(lambda x: (x[0], x[1]))
>>> VehicleMake.take(10)
[('FORD', 1327031), ('TOYOT', 1210979), ('HONDA', 1076000), ('NISSA', 906588), ('CHEVR', 734740), ('FRUEH', 460533), ('ME/BE', 393382), ('DODGE', 373968), ('BMW', 369649), ('JEEP', 339538)]
>>>
>>>
>>>
>>>
```

<u>Example 2:</u> Result of total number of violations based on vehicle make

Building the Recommender System

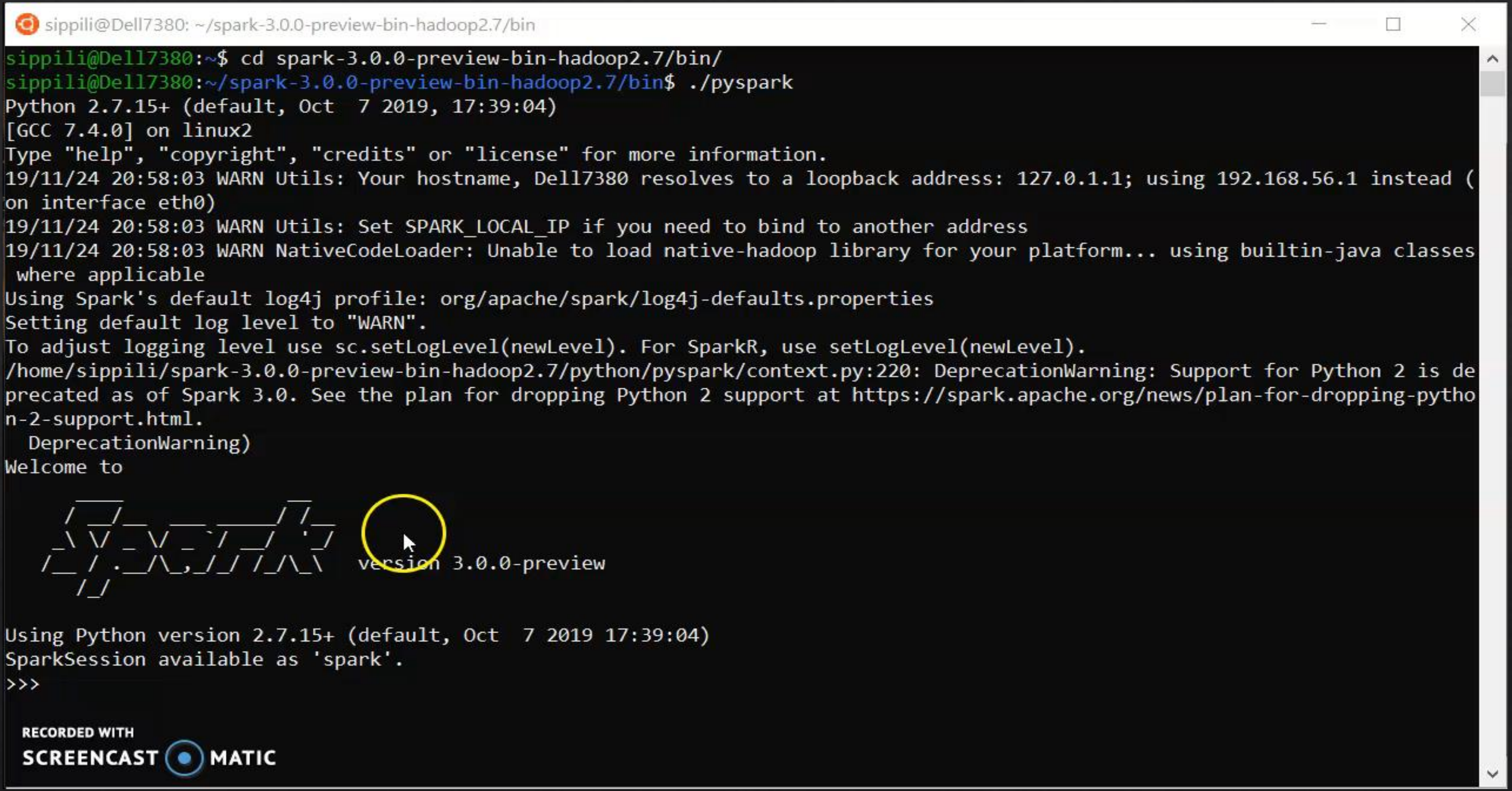
This system has been built to predict the reasons for a person to get ticketed at one's place.

Collaborative filtering is the recommender system approach has been applied in this project

Since the dataset was large, huge amount of computations with high speed and performance has been handled using PySpark

The system has been built using ALS library where prediction and evaluation has been done using the testing data

Recommender System Execution Demo



```
sippili@Dell7380:~$ cd spark-3.0.0-preview-bin-hadoop2.7/bin/
sippili@Dell7380:~/spark-3.0.0-preview-bin-hadoop2.7/bin$ ./pyspark
Python 2.7.15+ (default, Oct  7 2019, 17:39:04)
[GCC 7.4.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
19/11/24 20:58:03 WARN Utils: Your hostname, Dell7380 resolves to a loopback address: 127.0.1.1; using 192.168.56.1 instead (
on interface eth0)
19/11/24 20:58:03 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
19/11/24 20:58:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
 where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
/home/sippili/spark-3.0.0-preview-bin-hadoop2.7/python/pyspark/context.py:220: DeprecationWarning: Support for Python 2 is de
precated as of Spark 3.0. See the plan for dropping Python 2 support at https://spark.apache.org/news/plan-for-dropping-pytho
n-2-support.html.
  DeprecationWarning)
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.0-preview
      /_/

Using Python version 2.7.15+ (default, Oct  7 2019 17:39:04)
SparkSession available as 'spark'.
>>>
```

Conclusion

Three major technologies: Apache Hive, Spark and Pig have been implemented as part of this project and their performance have been compared that helped us choose the right technology for the current scenario which gave us efficient high-speed results on a very huge set of data.

We chose Apache Spark which helped us systematically extract information to solving one of the real-life problems considering huge amount of data collected from many years.

Using Spark, we successfully performed Data Integration, Machine Learning and Interactive Analysis.

Collaborative recommender system helped us predict the best reasons for anyone to get a parking ticket and how one could not commit the same mistake in the future.