# Big Data Computing

# Lab 2 Report

Rosa Sierra, Enmanuel Aquino

## Directory

The lab2.zip has the following folder structure:

- Test_lab2
    - Part1
        - Part1.py
        - Part1.sh
        - Shot_logs.csv
    - Part2
        - Black_Ticket.py
        - Test.sh
    - Part3
        - Hour_car.py
        - Test.sh

## Important Notes:

- For all file in Test_lab2, please put the data under the folder “hdfs-test-python” in https://github.com/yingmao/hdfs-test.git.
- For part1 (shot logs) save the data file under the name “shot_logs.csv”.
- For part2 and part3 save the data file under the name “nyc_parking_violations_data.csv”
- All the content of the zip can also be accessed on the github link: https://github.com/rosamsierrap/test-lab2

## Part 1

## NBA Shot Logs

*For the results of the NBA Shot Logs, run the bash file part1.sh*

**Comfortable Zones of Shooting**

**For each player, we define the comfortable zone of shooting as a matrix of, {SHOT DIST, CLOSE DEF DIST, SHOT CLOCK}. Considering the hit rate, which zone is the best for James Harden, Chris Paul, Stephen Curry and Lebron James.**

The general idea we put into practice to answer this question was to implement a K-Means algorithm with an iterative approach. In contrast to practice1, we introduce the Spark environment. After importing the necessary libraries for Spark instances, it removes all the rows containing null values.

The data is converted to an RDD and then transformed by selecting the columns containing information about the shooter, made shots, distance, defensive distance, and time. The made shots are filtered, and the remaining shots are grouped by shooter. After, K-means is performed on the remaining shots for each shooter. The K-means function used is the Euclidean distance between two points. For each shooter, a random initial set of centroids is chosen from the dataset, and the algorithm is iterated over 10 times to update the centroid position. Finally, the output of the k-means clustering is written to a text file.

<u>**Results**</u>

- lebron james: [[4.9381, 5.0135, 19.7335], [4.3557, 2.5567, 11.1062], [22.0512, 5.0729, 10.0928], [7.4103, 2.7603, 4.5103]]
- chris paul: [[6.0174, 3.8826, 16.4449], [9.4607, 3.7344, 7.223], [18.2138, 5.0337, 15.5571], [21.0702, 5.0585, 6.8415]]
- stephen curry: [[23.4638, 5.8014, 7.8217], [13.6704, 4.0535, 15.3338], [4.0831, 3.1622, 15.2093], [24.0569, 5.6875, 17.4181]]
- james harden: [[14.2963, 3.9556, 8.7778], [23.9505, 5.3021, 15.1814], [21.807, 4.1, 4.2279], [4.0505, 3.0248, 15.8842]]

## Part 2

## Parking Violations

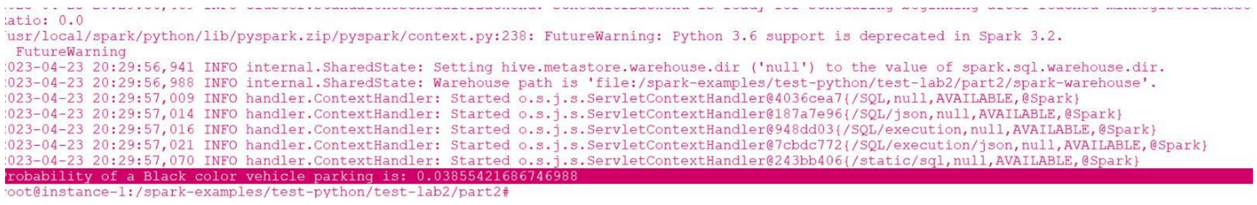
*For the results of the NY Parking violations, run the bash file test.sh*

- **Given a Black vehicle parking illegally at 34510, 10030, 34050 (street codes).What is the probability that it will get a ticket? (very rough prediction)**

The results come from getting the car color and street codes variables. After creating the spark session, we load the CSN file. Afterwards, in the RDD phase we filter the black cars from the list, and parking violations that occurred on one of the street codes specified. The action the performs is a simple count. For getting the total number of cars that receive a ticket, we just simply count total number of cars. Finally, it calculates the probability of a black colored car receiving a ticket by dividing the count of parking violations that satisfy the criteria by the total number of parking violations.

**Note:** The street codes [34510, 10030, 34050] don't appear on our testing dataset. So, the run was made by adding the street codes['25390','22040'].

**Results**



```
:atio: 0.0
'usr/local/spark/python/lib/pyspark.zip/pyspark/context.py:238: FutureWarning: Python 3.6 support is deprecated in Spark 3.2.
  FutureWarning
:023-04-23 20:29:56,941 INFO internal.SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir.
:023-04-23 20:29:56,988 INFO internal.SharedState: Warehouse path is 'file:/spark-examples/test-python/test-lab2/part2/spark-warehouse'.
:023-04-23 20:29:57,009 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@4036cea7{/SQL,null,AVAILABLE,@Spark}
:023-04-23 20:29:57,014 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@187a7e96{/SQL/json,null,AVAILABLE,@Spark}
:023-04-23 20:29:57,016 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@948dd03{/SQL/execution,null,AVAILABLE,@Spark}
:023-04-23 20:29:57,021 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@7cbdc772{/SQL/execution/json,null,AVAILABLE,@Spark}
:023-04-23 20:29:57,070 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@243bb406{/static/sql,null,AVAILABLE,@Spark}
'robability of a Black color vehicle parking is: 0.03855421686746988
'oot@instance-1:/spark-examples/test-python/test-lab2/part2#
```

## Part 3

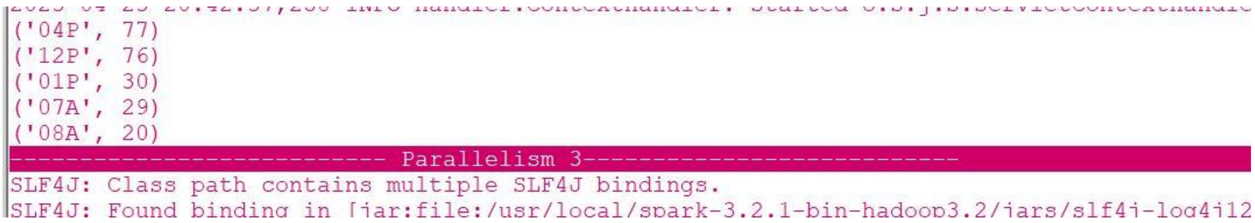
## Parking Violations

- **When are tickets most likely to be issued?**

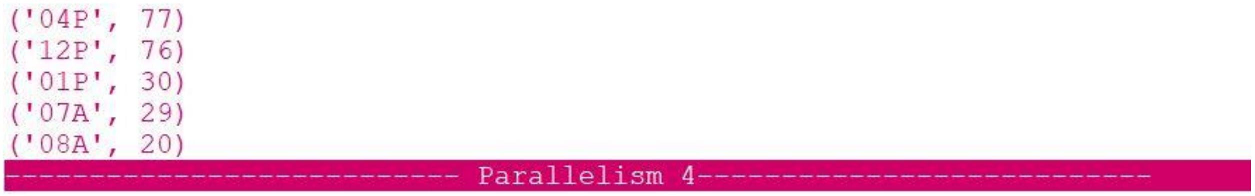
The results calculate the hours when it’s most likely to be issued. The first step is to create a Spark session. In the RDD phase, after loading the data, it selects the ‘Violation Time’ and drops all the values that are missing. The RDD is then transformed by mapping each row to a combination/tuple that contains <hour,1>, using the reduceByKey. Finally, the result is sorted by the count of violations in descending order, keeping just the top 5 hours. Of course, in terms of the running phase it’s performed via parallelism. Which gives the ability of dividing a dataset into multiple partitions, with the goal of being processed in parallel across multiple nodes in a cluster.

**<u>Results</u>**

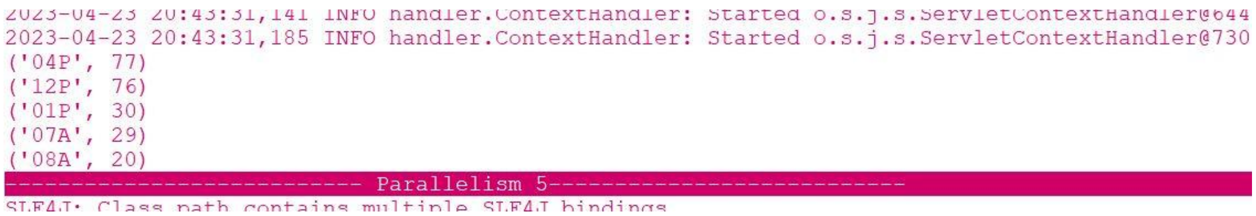
Based on the chunk of data tested, <u>the tickets are most likely to be issued at 04:00 PM.</u>



```
2023-04-23 20:42:57,[illegible] INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandle
('04P', 77)
('12P', 76)
('01P', 30)
('07A', 29)
('08A', 20)
--------------------------- Parallelism 3---------------------------
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/spark-3.2.1-bin-hadoop3.2/jars/slf4j-log4j12
```



```
('04P', 77)
('12P', 76)
('01P', 30)
('07A', 29)
('08A', 20)
--------------------------- Parallelism 4---------------------------
```



```
2023-04-23 20:43:31,141 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@644
2023-04-23 20:43:31,185 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@730
('04P', 77)
('12P', 76)
('01P', 30)
('07A', 29)
('08A', 20)
--------------------------- Parallelism 5---------------------------
SLF4J: Class path contains multiple SLF4J bindings
```