The City College of New York Department of Computer Science

THE GRADUATE CENTER CITY UNIVERSITY OF NEW YORK

DSE I2450/CSc 84030: Big Data Analytics/Scalable Computation

SPRING 2020

Final Challenge – Parking Violations in NYC

OBJECTIVES:

In this challenge, we would like to gather statistics on the number of parking violations (tickets) per street segment in NYC over the past 5 years. In particular, for each street segment in NYC, we would like to have the following:

1. The total number of parking violations for each year from 2015 to 2019.
2. The rate that the total number of violations change over the years using Ordinary Least Squares.

You are asked to write a Spark program to compute the above metrics for all the street segments in NYC.

INPUT DATA:

Parking Violations Issued – Fiscal Year 2015-2019 (around 10GB in total)

Source: https://data.cityofnewyork.us/browse?q=%22Parking%20Violations%20Issued%22
Description: Data sets contain parking violations issued during the respective *fiscal* year. More info (including data dictionary) can be found on one of the results listed in the above link. All 5 years of data are available on HDFS, in their raw exported CSV format, under the following folder:
`hdfs:///data/share/bdm/nyc_parking_violations/`

NYC Street Centerline (CSCL) (around 120k segments)

Source: https://data.cityofnewyork.us/City-Government/NYC-Street-Centerline-CSCL-/exjm-f27b
Description: The NYC Street Centerline (CSCL) is a road-bed representation of New York City streets containing address ranges and other information such as traffic directions, road types, segment types. Details on this data set are available at:
https://github.com/CityOfNewYork/nyc-geo-metadata/blob/master/Metadata/Metadata_StreetCenterline.md
The data is available in CSV format on HDFS as:
`hdfs:///data/share/bdm/nyc_cscl.csv`

METHODLOGY:

Generally, this challenge could be addressed by performing a spatial join of the violation data set with the street centerline one based on the issued location of each violation. However, there is no geospatial location (latitude and longitude) recorded for the violation data set. Instead, a street address is provided through the ***House Number***; ***Street Name***; and ***Violation County*** field. For example, if a violation was issued at "187 State St" in Brooklyn (aka. **K**ings county), values for the 3 fields would be:

House Number	Street Name	Violation County
187	State St	K

Possibly, we could use a geocoding software (e.g. DOITT's GeoSupport) or a geocoding web service (e.g. DOITT's GeoClient) to help us converting addresses into geospatial locations. However, these approaches are either restricted to a desktop environment or limited by the number of requests. In order to perform geocoding on a large amount of records such as 50 million parking violations, we will rely on the street centerline data set. The street centerline data set not only provides geometries of street segments but also records the house number range for each segment. For example, we should be able to find the following record in the data set:

PHYSICALID	FULL_STREE	BOROCODE	L_LOW_HN	L_HIGH_HN
58903	STATE ST	3	183	229

This means that the address "187 State St" in Brooklyn should be matched to the street segment with physical ID 58903. This is because the segment has the same street name ("STATE ST", case-insensitive) and borough ("3" means Brooklyn), and the target house number (187) is within its range (183 to 229).

Note that in additional to the 2 "left" fields (`L_LOW_HN` and `L_HIGH_HN`) that captures the odd house number range, there are also `R_LOW_HN` and `R_HIGH_HN` corresponding to the even house number range. You have to select the right fields to perform the lookup depending on the house number (aka. odd or even). Also, house numbers may be expressed in a compound form such as 170-044, or 56-098. These should be treated as tuples (170,44) and (56,98). A valid range from the street centerline could be from (54,0) to (56,98). Not all street segments have house number.

For the parking violations data set, the `Issue Date` field should be used to determine which year a violation belongs to. The filename, which is organized by fiscal year, should not be used. We are only interested in those violations from 2015 to 2019. Violations outside of those years should not be counted.

Ordinary Least Squares: for simplicity, only the coefficient of your regression line (aka. the slope) is of interest. This is regardless of the estimated errors, residuals and/or coefficient of determination (aka. R-squared) of your regressor.

OUTPUT DATA:

Your output must be written to HDFS in CSV format (without header). It must contain the columns below, and the records must be sorted by `PHYSICALID` of street segments:

```
SAMPLE OUTPUT (header not to be included):
PHYSICALID,COUNT_2015,COUNT_2016,COUNT_2017,COUNT_2018,COUNT_2019,OLS_COEF
...
58903,1000,900,800,700,600,-100
...
```

where `COUNT_XXXX` is the total number of violations issued at the street segment `PHYSICALID` for year `XX`, and `OLS_COEF` is the OLS coefficient.

Note that all streets segments in the street centerline data set must be present in the output even when there are no violations issued (zero counts).

SUBMISSION:

Your code will be run with exactly 30 cores (6 executors and 5 cores per executor), and no more than 10GB of memory per executor. It must not take more than 30 minutes to complete.

```
SAMPLE RUN:
spark-submit --num-executors 6 --executor-cores 5 --executor-memory 10G \
  BDM_FinalChallenge.py <OUTPUT_HDFS_FOLDER>
```

Note: the above command is only an example to demonstrate how to specify the number of executors. You may have additional files to be included with your submission, and must replace `<OUTPUT_HDFS_FOLDER>` with a real output location. There is no need to specify the input data through command line. The files are assumed to be located under `/data/share/bdm/` as specified above. This means that you can hard-code the input filenames inside your code.

Please provide the following information when submitting your final challenge:

- One or more files including your application Python file and any dependencies that it may need.
- The spark-submit command needed to run your code.
- The total time your code took to produce the results (when you last ran it successfully).
- A CSV collected from your output folder (using `hadoop fs -getmerge` …) for sanity check