Otto Heinonen – Erasmus

Advanced-Topics-in-Database-Systems project

Link to Github: https://github.com/othei99/Advanced-Topics-in-Database-Systems.git

Codes can find in github files by questions. Executing code you need to use Jupyterlab opened from Amazon SageMaker AI notebook and there group 52. If wanted to use code without open notebook via AWS, you need be sure you have access to AWS S3.

Questions/Queries & Answers

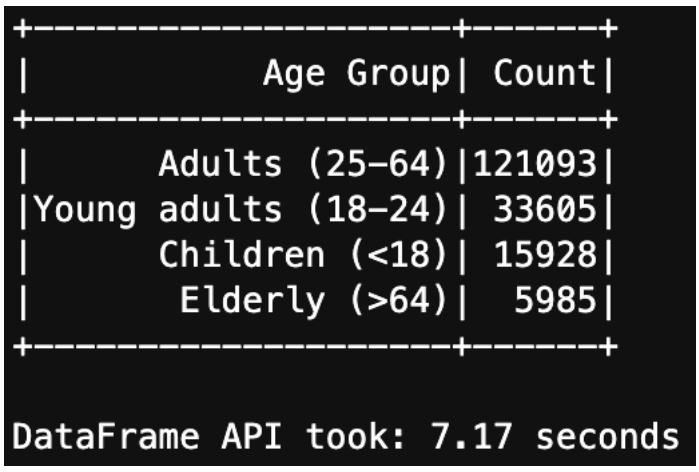
Query 1

Sort, in descending order, the age groups of victims in incidents involving any form of "aggravated assault" (i.e., include this term in the relevant description). Consider the following age groups:

Children: < 18, Young adults: 18 – 24, Adults: 25 – 64, Elderly: >64

Answer:

DataFrame API result:



RDD API result:

```
Adults (25-64): 121093
Young adults (18-24): 33605
Children (<18): 15928
Elderly (>64): 5985
RDD API took: 19.73 seconds
```

Question 1

Implement Query 1 using the DataFrame and RDD APIs. Run both implementations with 4 Spark executors. Is there a performance difference between the two APIs? Justify your answer. (20%)

Answer:

There is a significant performance difference between the two APIs. In this case, the DataFrame API performed much faster (7.17 seconds) compared to the RDD API (19.73 seconds). In most cases, the DataFrame API utilizes Spark's Catalyst optimizer and Tungsten execution engine to implement advanced query optimizations, generate efficient execution plans, and better memory management. All these work together to minimize unnecessary computations and lower execution times.

By contrast, RDD API is at an even lower level of abstraction and thus does not benefit from any of these optimizations. Directly processes data, but requires manually handling most operations, which tends to be inefficient for comparatively large datasets or complex transformations.

Therefore, DataFrame API generally performs better in terms of performance, scalability, and usability within Spark applications. The RDD API is usually very useful for low-level operations or in cases where there is a need for extremely fine granularity. However, this set of functions is lagging in terms of execution speed in general for data-processing tasks.

Query 2

Find, for each year, the 3 Police Departments with the highest percentage of closed cases. Print the year, the names (locations) of the departments, their percentages as well as their ranking. The results are given in ascending order by year and ranking (see example below).

year	precinct	closed_case_rate	#
2010	West Valley	30.57974335472044	1
2010	N Hollywood	29.23808669119627	2
2010	Mission	27.58372669119627	3

Answer:

DataFrame API result:

Top 3 for Year 2010:

Year	precinct	closed_case_rate	#
2010	Rampart	32.84713448949121	1
2010	Olympic	31.515289821999087	2
2010	Harbor	29.36028339237341	3

Top 3 for Year 2011:

Year	precinct	closed_case_rate	#
2011	Olympic	35.040060090135206	1
2011	Rampart	32.4964471814306	2
2011	Harbor	28.51336246316431	3

Top 3 for Year 2012:

Year	precinct	closed_case_rate	#
2012	Olympic	34.29708533302119	1
2012	Rampart	32.46000463714352	2
2012	Harbor	29.509585848956675	3

Top 3 for Year 2013:

Year	precinct	closed_case_rate	#
2013	Olympic	33.58217940999398	1
2013	Rampart	32.1060382916053	2
2013	Harbor	29.723638951488557	3

Top 3 for Year 2014:

Year	precinct	closed_case_rate	#
2014	Van Nuys	32.0215235281705	1
2014	West Valley	31.49754809505847	2
2014	Mission	31.224939855653567	3

Top 3 for Year 2015:

Year	precinct	closed_case_rate	#
2015	Van Nuys	32.265140677157845	1
2015	Mission	30.463762673676303	2
2015	Foothill	30.353001803658852	3

Top 3 for Year 2016:

Year	precinct	closed_case_rate	#
2016	Van Nuys	32.194518462124094	1
2016	West Valley	31.40146437042384	2
2016	Foothill	29.908647228131645	3

Top 3 for Year 2017:

Year	precinct	closed_case_rate	#
2017	Van Nuys	32.0554272517321	1
2017	Mission	31.055387158996968	2
2017	Foothill	30.469700657094183	3

Top 3 for Year 2018:

Year	precinct	closed_case_rate	#
2018	Foothill	30.731346958877126	1
2018	Mission	30.727023319615913	2
2018	Van Nuys	28.905206942590123	3

Top 3 for Year 2019:

Year	precinct	closed_case_rate	#
2019	Mission	30.727411112319235	1
2019	West Valley	30.57974335472044	2
2019	N Hollywood	29.23808669119627	3

Top 3 for Year 2020:

Year	precinct	closed_case_rate	#
2020	West Valley	30.771131982204647	1
2020	Mission	30.14974649215894	2
2020	Harbor	29.693486590038315	3

Top 3 for Year 2021:

Year	precinct	closed_case_rate	#
2021	Mission	30.318115590092276	1
2021	West Valley	28.971087440009363	2
2021	Foothill	27.993757094211126	3

Top 3 for Year 2022:

Year	precinct	closed_case_rate	#
2022	West Valley	26.536367172306498	1
2022	Harbor	26.337538060026098	2
2022	Topanga	26.234013317831096	3

Top 3 for Year 2023:

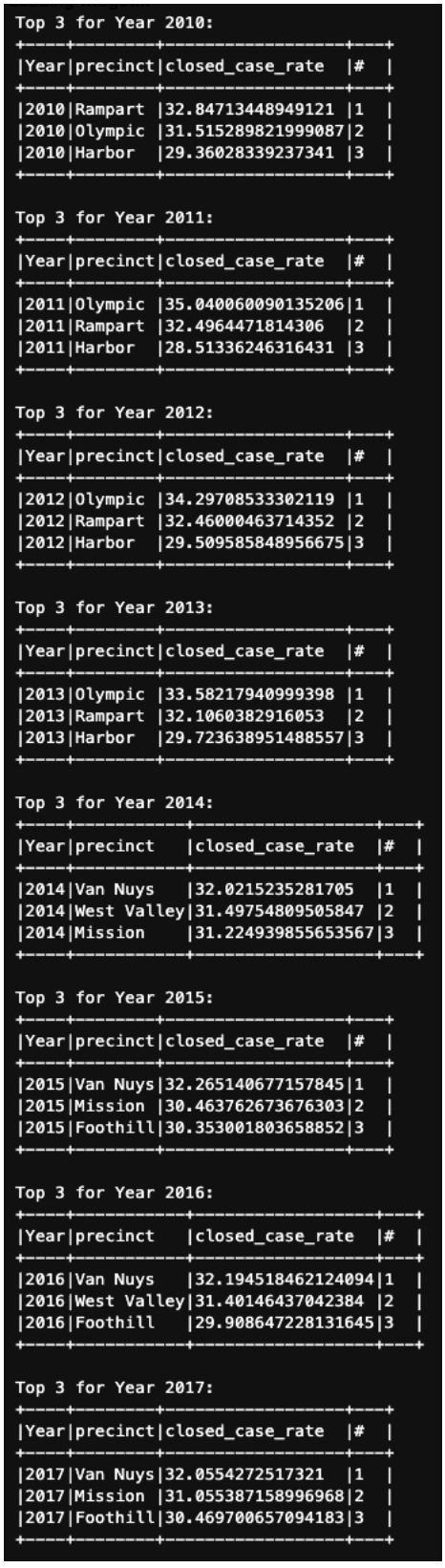
Year	precinct	closed_case_rate	#
2023	Foothill	26.76076020122974	1
2023	Topanga	26.538022616453986	2
2023	Mission	25.662731120516817	3

Top 3 for Year 2024:

Year	precinct	closed_case_rate	#
2024	N Hollywood	19.598528961078763	1
2024	Foothill	18.620882188721385	2
2024	77th Street	17.586318167150694	3

```
DataFrame API took : 29.74 seconds
```

SQL API result:



```
Top 3 for Year 2010:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2010|Rampart |32.84713448949121 |1  |
|2010|Olympic |31.515289821999087|2  |
|2010|Harbor  |29.36028339237341 |3  |
+----+--------+------------------+---+

Top 3 for Year 2011:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2011|Olympic |35.040060090135206|1  |
|2011|Rampart |32.4964471814306  |2  |
|2011|Harbor  |28.51336246316431 |3  |
+----+--------+------------------+---+

Top 3 for Year 2012:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2012|Olympic |34.29708533302119 |1  |
|2012|Rampart |32.46000463714352 |2  |
|2012|Harbor  |29.509585848956675|3  |
+----+--------+------------------+---+

Top 3 for Year 2013:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2013|Olympic |33.58217940999398 |1  |
|2013|Rampart |32.1060382916053  |2  |
|2013|Harbor  |29.723638951488557|3  |
+----+--------+------------------+---+

Top 3 for Year 2014:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2014|Van Nuys   |32.0215235281705  |1  |
|2014|West Valley|31.49754809505847 |2  |
|2014|Mission    |31.224939855653567|3  |
+----+-----------+------------------+---+

Top 3 for Year 2015:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2015|Van Nuys|32.265140677157845|1  |
|2015|Mission |30.463762673676303|2  |
|2015|Foothill|30.353001803658852|3  |
+----+--------+------------------+---+

Top 3 for Year 2016:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2016|Van Nuys   |32.194518462124094|1  |
|2016|West Valley|31.40146437042384 |2  |
|2016|Foothill   |29.908647228131645|3  |
+----+-----------+------------------+---+

Top 3 for Year 2017:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2017|Van Nuys|32.0554272517321  |1  |
|2017|Mission |31.055387158996968|2  |
|2017|Foothill|30.469700657094183|3  |
+----+--------+------------------+---+
```

```
Top 3 for Year 2018:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2018|Foothill|30.731346958877126|1  |
|2018|Mission |30.727023319615913|2  |
|2018|Van Nuys|28.905206942590123|3  |
+----+--------+------------------+---+

Top 3 for Year 2019:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2019|Mission    |30.727411112319235|1  |
|2019|West Valley|30.57974335472044 |2  |
|2019|N Hollywood|29.23808669119627 |3  |
+----+-----------+------------------+---+

Top 3 for Year 2020:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2020|West Valley|30.771131982204647|1  |
|2020|Mission    |30.14974649215894 |2  |
|2020|Harbor     |29.693486590038315|3  |
+----+-----------+------------------+---+

Top 3 for Year 2021:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2021|Mission    |30.318115590092276|1  |
|2021|West Valley|28.971087440009363|2  |
|2021|Foothill   |27.993757094211126|3  |
+----+-----------+------------------+---+

Top 3 for Year 2022:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2022|West Valley|26.536367172306498|1  |
|2022|Harbor     |26.337538060026098|2  |
|2022|Topanga    |26.234013317831096|3  |
+----+-----------+------------------+---+

Top 3 for Year 2023:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2023|Foothill|26.76076020122974 |1  |
|2023|Topanga |26.538022616453986|2  |
|2023|Mission |25.662731120516817|3  |
+----+--------+------------------+---+

Top 3 for Year 2024:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2024|N Hollywood|19.598528961078763|1  |
|2024|Foothill   |18.620882188721385|2  |
|2024|77th Street|17.586318167150694|3  |
+----+-----------+------------------+---+
```

```
SQL API took: 26.65 seconds
```

Question 2

a) Implement Query 2 using the DataFrame and SQL APIs. Report and compare the

execution times between the two implementations.

Answer:

DataFrame API applies PySpark's transformation methods including groupBy, filter, withColumn, which chains explicitly the operations and took 29.74 seconds to complete, whereas SQL API implements operations using Spark's SQL engine with SQL-style queries, which was finished in 26.65 seconds.

The SQL API benefits from Spark's catalyst optimizer in a much better way: the SQL statements are optimized for far better execution plans. The DataFrame API needs to invoke explicit method calls in Python, which may result in minor overhead, whereas the SQL API's declarative way will be better able to optimize the query pipeline.

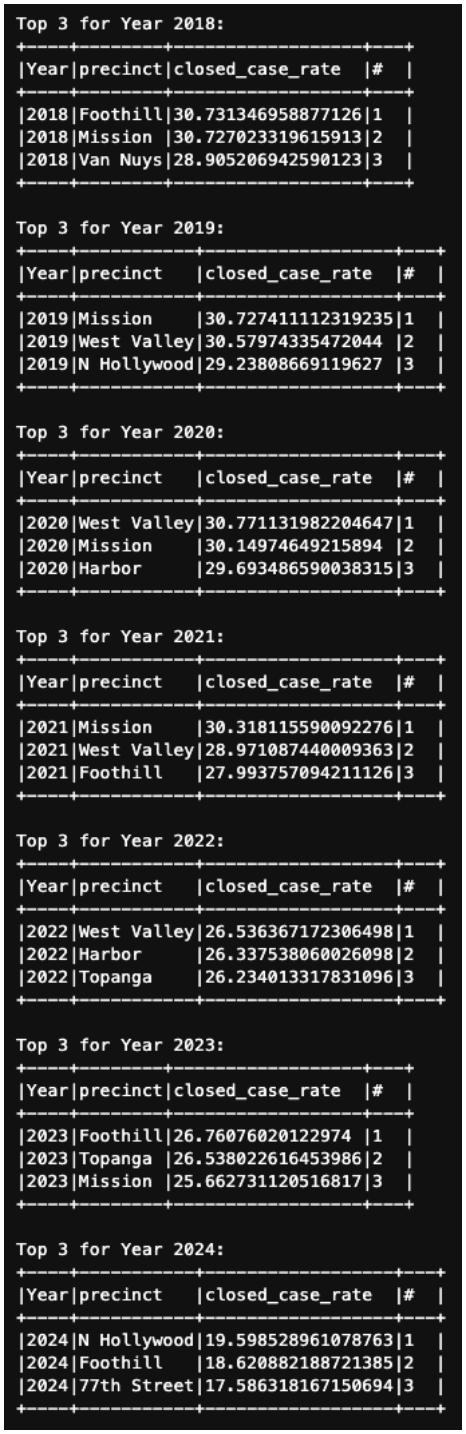
In brief, for performance-critical scenarios where data processing can be expressed with SQL, usage of the SQL API is definitely the best choice. If it require programmatic flexibility or interaction with custom Python logic then DataFrame API will be best option.

b) Write Spark code that converts the main data set to parquet file format and stores a single .parquet file in your team's S3 bucket. Choose one of the two implementations of subquery a) (DataFrame or SQL) and compare the execution times of your application when the data is imported as .csv and as .parquet.

Answer:

```
DataFrame writed successfully to S3: s3://groups-bucket-dblab-905418150721/group52/main_data_set.parquet
```

```
Top 3 for Year 2010:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2010|Rampart |32.84713448949121 |1  |
|2010|Olympic |31.515289821999087|2  |
|2010|Harbor  |29.36028339237341 |3  |
+----+--------+------------------+---+

Top 3 for Year 2011:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2011|Olympic |35.040060090135206|1  |
|2011|Rampart |32.4964471814306  |2  |
|2011|Harbor  |28.51336246316431 |3  |
+----+--------+------------------+---+

Top 3 for Year 2012:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2012|Olympic |34.29708533302119 |1  |
|2012|Rampart |32.46000463714352 |2  |
|2012|Harbor  |29.509585848956675|3  |
+----+--------+------------------+---+

Top 3 for Year 2013:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2013|Olympic |33.58217940999398 |1  |
|2013|Rampart |32.1060382916053  |2  |
|2013|Harbor  |29.723638951488557|3  |
+----+--------+------------------+---+

Top 3 for Year 2014:
+----+----------+------------------+---+
|Year|precinct  |closed_case_rate  |#  |
+----+----------+------------------+---+
|2014|Van Nuys  |32.0215235281705  |1  |
|2014|West Valley|31.49754809505847 |2  |
|2014|Mission   |31.224939855653567|3  |
+----+----------+------------------+---+

Top 3 for Year 2015:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2015|Van Nuys|32.265140677157845|1  |
|2015|Mission |30.463762673676303|2  |
|2015|Foothill|30.353001803658852|3  |
+----+--------+------------------+---+

Top 3 for Year 2016:
+----+----------+------------------+---+
|Year|precinct  |closed_case_rate  |#  |
+----+----------+------------------+---+
|2016|Van Nuys  |32.194518462124094|1  |
|2016|West Valley|31.40146437042384 |2  |
|2016|Foothill  |29.908647228131645|3  |
+----+----------+------------------+---+

Top 3 for Year 2017:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2017|Van Nuys|32.0554272517321  |1  |
|2017|Mission |31.055387158996968|2  |
|2017|Foothill|30.469700657094183|3  |
+----+--------+------------------+---+
```



```
Top 3 for Year 2018:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2018|Foothill|30.731346958877126|1  |
|2018|Mission |30.727023319615913|2  |
|2018|Van Nuys|28.905206942590123|3  |
+----+--------+------------------+---+

Top 3 for Year 2019:
+----+----------+------------------+---+
|Year|precinct  |closed_case_rate  |#  |
+----+----------+------------------+---+
|2019|Mission   |30.727411112319235|1  |
|2019|West Valley|30.57974335472044 |2  |
|2019|N Hollywood|29.23808669119627 |3  |
+----+----------+------------------+---+

Top 3 for Year 2020:
+----+----------+------------------+---+
|Year|precinct  |closed_case_rate  |#  |
+----+----------+------------------+---+
|2020|West Valley|30.771131982204647|1  |
|2020|Mission   |30.14974649215894 |2  |
|2020|Harbor    |29.693486590038315|3  |
+----+----------+------------------+---+

Top 3 for Year 2021:
+----+----------+------------------+---+
|Year|precinct  |closed_case_rate  |#  |
+----+----------+------------------+---+
|2021|Mission   |30.318115590092276|1  |
|2021|West Valley|28.971087440009363|2  |
|2021|Foothill  |27.993757094211126|3  |
+----+----------+------------------+---+

Top 3 for Year 2022:
+----+----------+------------------+---+
|Year|precinct  |closed_case_rate  |#  |
+----+----------+------------------+---+
|2022|West Valley|26.536367172306498|1  |
|2022|Harbor    |26.337538060026098|2  |
|2022|Topanga   |26.234013317831096|3  |
+----+----------+------------------+---+

Top 3 for Year 2023:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2023|Foothill|26.76076020122974 |1  |
|2023|Topanga |26.538022616453986|2  |
|2023|Mission |25.662731120516817|3  |
+----+--------+------------------+---+

Top 3 for Year 2024:
+----+----------+------------------+---+
|Year|precinct  |closed_case_rate  |#  |
+----+----------+------------------+---+
|2024|N Hollywood|19.598528961078763|1  |
|2024|Foothill  |18.620882188721385|2  |
|2024|77th Street|17.586318167150694|3  |
+----+----------+------------------+---+
```

```
Execution time: 11.19 seconds
```

I used the DataFrame API in the context of this dataset. For executing the .csv version, it took about 29.74 seconds: reading two .csv files separately, merging them, and finally performing

aggregations and calculations. Reading and parsing time for the .csv format significantly contributed to the total time. Execution time for using the .parquet format stood at 11.19 seconds. Parquet is a columnar format highly optimized and performs well with Spark in reading more data and efficiently processing it. It doesn't have the heavy parsing overhead of the .csv and enjoys columnar compression.

The improvement comes mainly because of the way Parquet handles columnar data access, which is effective for the analytical workload of this application. The use of the .parquet file format in Spark processing tasks is recommended for better performance and scalability while using big datasets.

Query 3

Using the 2010 Census population data and the 2015 Census household income data, calculate the following for each area of Los Angeles: The average annual income per person and the ratio of total crimes per person. The results should be summarized in a table.

Answer:

zip (LA)	crimes (2010)	population (2010)	crimes_per_person (2010)	estimated_income (2015)	average_annual_per_person
90094	136	5464	0.024890190336749635	$104367.0	$19.10084187408492
90230	452	31766	0.014229049927595543	$74823.0	$2.3554429263992946
90293	435	12132	0.03585558852621167	$82055.0	$6.763517969007583
90292	514	21576	0.023822766036336672	$100507.0	$4.658277715980719
90291	2457	28341	0.08669418863131152	$80111.0	$2.826682191877492
90405	47	27186	0.001728831015964099	$77948.0	$2.8672110645185023
90045	3299	39480	0.0835612968591692	$75684.0	$1.9170212765957446
90066	1755	55277	0.031749190440870524	$68132.0	$1.232556035964325
90401	19	6722	0.002826539720321333	$62703.0	$9.328027372805712
90245	7	16654	4.2031944277651017E-4	$85727.0	$5.147532124414555
90266	5	35135	1.4230823964707557E-4	$143527.0	$4.085014942365163
90008	3015	32327	0.09326569121786742	$36564.0	$1.1310669100133015
90043	2802	44789	0.0625600035723057	$38180.0	$0.8524414476768849
90056	27	7827	0.0034495975469528554	$84099.0	$10.74472978152549
90301	102	36568	0.002789323999124918	$37424.0	$1.0234084445416758
90250	2	93193	2.146083933342633E-5	$46172.0	$0.4954449368514803
90278	1	40071	2.4955703626063736E-5	$107010.0	$2.6705098450250806
90304	11	28210	3.899326479971641E-4	$36412.0	$1.2907479617157036
90302	31	29415	0.0010538840727519973	$41426.0	$1.4083290838007818
90254	2	19506	1.0253255408592229E-4	$111187.0	$5.70014354557572

only showing top 20 rows

Time: 15.450810194015503 seconds

Question 3

Implement Query 3 using DataFrame or SQL API. Use hint & explain methods to find out which join strategies the catalyst optimizer uses. Experiment by forcing Spark to use different

strategies (between BROADCAST, MERGE, SHUFFLE_HASH, SHUFFLE_REPLICATE_NL) and comment on the results you observe. Which of the available Spark join strategies is (are) the most appropriate and why?

Answer:

BROADCAST Strategy

```
# Use BROADCAST join to combine crime data and income data
combined_df = crime_ratio_df.join(
    broadcast(tulo_df.select("zip", "estimated_median_income (2015)")),
    "zip",
    "inner"
)
```

```
+--------+------------+----------------+----------------------+----------------------+------------------------+
|zip (LA)|crimes (2010)|population (2010)|crimes_per_person (2010)|estimated_income (2015)|average_annual_per_person|
+--------+------------+----------------+----------------------+----------------------+------------------------+
|90094   |136         |5464            |0.024890190336749635  |$104367.0             |$19.10084187408492      |
|90266   |5           |35135           |1.4230823964707557E-4 |$143527.0             |$4.085014942365163      |
|90230   |452         |31766           |0.014229049927595543  |$74823.0              |$2.3554429263992946     |
|90293   |435         |12132           |0.0358555852621167    |$82055.0              |$6.763517969007583      |
|90292   |514         |21576           |0.023822766036336672  |$100507.0             |$4.658277715980719      |
|90291   |2457        |28341           |0.08669418863131152   |$80111.0              |$2.826682191877492      |
|90405   |47          |27186           |0.001728831015964099  |$77948.0              |$2.8672110645185023     |
|90034   |2234        |57964           |0.038541163480781175  |$58004.0              |$1.0006900835001036     |
|90045   |3299        |39480           |0.0835612968591692    |$75684.0              |$1.9170212765957446     |
|90066   |1755        |55277           |0.031749190440870524  |$68132.0              |$1.232556035964325      |
|90401   |19          |6722            |0.002826539720321333  |$62703.0              |$9.328027372805712      |
|90245   |7           |16654           |4.2031944277651017E-4 |$85727.0              |$5.147532124414555      |
|90008   |3015        |32327           |0.09326569121786742   |$36564.0              |$1.1310669100133015     |
|90043   |2802        |44789           |0.0625600035723057    |$38180.0              |$0.8524414476768849     |
|90056   |27          |7827            |0.0034495975469528554 |$84099.0              |$10.74472978152549      |
|90047   |3114        |48606           |0.06406616467102827   |$39269.0              |$0.8079043739456034     |
|90301   |102         |36568           |0.002789323999124918  |$37424.0              |$1.0234084445416758     |
|90250   |2           |93193           |2.146083933342633E-5  |$46172.0              |$0.4954449368514803     |
|90304   |11          |28210           |3.899326479971641E-4  |$36412.0              |$1.2907479617157036     |
|90303   |2           |26176           |7.640586797066015E-5  |$39671.0              |$1.5155485941320292     |
+--------+------------+----------------+----------------------+----------------------+------------------------+
only showing top 20 rows

BROADCAST Join Time: 14.191836595535278 seconds
```

MERGE Strategy

```
# Use a MERGE join to combine crime data and census population data
crime_ratio_df = crime_counts.join(
    aggregated_population.hint("merge"),
    "zip",
    "inner"
)
```

```
+--------+-------------+----------------+----------------------+---------------------+-------------------------+
|zip (LA)|crimes (2010)|population (2010)|crimes_per_person (2010)|estimated_income (2015)|average_annual_per_person|
+--------+-------------+----------------+----------------------+---------------------+-------------------------+
|90001   |794          |57110           |0.013902994221677465  |$33887.0             |$0.593363684118368       |
|90002   |2492         |51223           |0.04865002049860414   |$30413.0             |$0.5937371883724109      |
|90003   |6132         |66266           |0.0925361422147104    |$30805.0             |$0.46486886185977727     |
|90004   |2991         |62180           |0.04810228369250563   |$40612.0             |$0.6531360566098424      |
|90005   |1700         |37681           |0.04511557548897322   |$31142.0             |$0.8264642658103554      |
|90006   |2891         |59185           |0.0488468361916026    |$31521.0             |$0.5325842696629214      |
|90007   |2798         |40920           |0.06837732160312805   |$22304.0             |$0.5450635386119257      |
|90008   |3015         |32327           |0.09326569121786742   |$36564.0             |$1.1310669100133015      |
|90010   |733          |3800            |0.19289473684210526   |$45786.0             |$12.048947368421052      |
|90011   |5288         |103892          |0.05089901051091518   |$30251.0             |$0.291177376506372       |
|90012   |1680         |31103           |0.054014082242870465  |$31576.0             |$1.0152075362505224      |
|90013   |2059         |11772           |0.17490655793408086   |$19887.0             |$1.6893476044852191      |
|90014   |858          |7005            |0.12248394004282655   |$23642.0             |$3.3750178443968593      |
|90015   |2607         |18986           |0.1373117033603708    |$29684.0             |$1.5634678183924997      |
|90016   |2912         |47596           |0.06118161190015968   |$38330.0             |$0.8053197747709891      |
|90017   |1984         |23768           |0.0834735779198923    |$22754.0             |$0.9573375967687647      |
|90018   |2649         |49310           |0.05372135469478807   |$33864.0             |$0.6867572500506997      |
|90019   |3100         |64458           |0.048093332092215085  |$46571.0             |$0.722501473827919       |
|90020   |1216         |38967           |0.031205892165165398  |$38849.0             |$0.9969717966484462      |
|90021   |1306         |3951            |0.33054922804353326   |$12813.0             |$3.242976461655277       |
+--------+-------------+----------------+----------------------+---------------------+-------------------------+
only showing top 20 rows

MERGE Join Time: 14.373335599899292 seconds
```

SHUFFLE_HASH Strategy

```python
# Use SHUFFLE_HASH join to combine crime and population data
crime_ratio_df = crime_counts.join(
    aggregated_population.hint("shuffle_hash"),
    "zip",
    "inner"
)
```

```
+--------+-------------+----------------+----------------------+---------------------+-------------------------+
|zip (LA)|crimes (2010)|population (2010)|crimes_per_person (2010)|estimated_income (2015)|average_annual_per_person|
+--------+-------------+----------------+----------------------+---------------------+-------------------------+
|90008   |3015         |32327           |0.09326569121786742   |$36564.0             |$1.1310669100133015      |
|90064   |1426         |25403           |0.056135102153288985  |$87283.0             |$3.4359327638467896      |
|90062   |2395         |32821           |0.07297157307821212   |$34588.0             |$1.053837482099875       |
|90011   |5288         |103892          |0.05089901051091518   |$30251.0             |$0.291177376506372       |
|90021   |1306         |3951            |0.33054922804353326   |$12813.0             |$3.242976461655277       |
|90405   |47           |27186           |0.001728831015964099  |$77948.0             |$2.8672110645185023      |
|90007   |2798         |40920           |0.06837732160312805   |$22304.0             |$0.5450635386119257      |
|90034   |2234         |57964           |0.038541163480781175  |$58004.0             |$1.0006900835001036      |
|90037   |4458         |62276           |0.07158455905966986   |$27179.0             |$0.43642815852013617     |
|90066   |1755         |55277           |0.031749190440870524  |$68132.0             |$1.232556035964325       |
|90401   |19           |6722            |0.002826539720321333  |$62703.0             |$9.328027372805712       |
|90403   |1            |24525           |4.077471967380224E-5  |$78151.0             |$3.186585117227319       |
|90016   |2912         |47596           |0.06118161190015968   |$38330.0             |$0.8053197747709891      |
|90232   |78           |15149           |0.00514885470988184   |$77004.0             |$5.083107795894119       |
|90404   |15           |21360           |7.022471910112359E-4  |$66623.0             |$3.1190543071161048      |
|90018   |2649         |49310           |0.05372135469478807   |$33864.0             |$0.6867572500506997      |
|90248   |334          |9947            |0.033577963204986426  |$53306.0             |$5.359002714386247       |
|90044   |4563         |89779           |0.050824803127680195  |$29206.0             |$0.3253099277113802      |
|90745   |25           |57251           |4.3667359522104416E-4 |$71443.0             |$1.2478908665350823      |
|90061   |1775         |26872           |0.06605388508484668   |$33731.0             |$1.255247097350402       |
+--------+-------------+----------------+----------------------+---------------------+-------------------------+
only showing top 20 rows

SHUFFLE_HASH Join Time: 13.013461351394653 seconds
```

SHUFFLE_REPLICATE_NL Strategy

```
# Use SHUFFLE_REPLICATE_NL join to combine crime and population data
crime_ratio_df = crime_counts.join(
    aggregated_population.hint("shuffle_replicate_nl"),
    "zip",
    "inner"
)
```

```
+--------+-------------+----------------+----------------------+---------------------+------------------------+
|zip (LA)|crimes (2010)|population (2010)|crimes_per_person (2010)|estimated_income (2015)|average_annual_per_person|
+--------+-------------+----------------+----------------------+---------------------+------------------------+
|90094   |136          |5464            |0.024890190336749635  |$104367.0            |$19.10084187408492      |
|90293   |435          |12132           |0.03585558852621167   |$82055.0             |$6.763517969007583      |
|90292   |514          |21576           |0.023822766036336672  |$100507.0            |$4.658277715980719      |
|90291   |2457         |28341           |0.08669418863131152   |$80111.0             |$2.826682191877492      |
|90405   |47           |27186           |0.001728831015964099  |$77948.0             |$2.867211064518 5023    |
|90045   |3299         |39480           |0.0835612968591692    |$75684.0             |$1.9170212765957446     |
|90066   |1755         |55277           |0.031749190440870524  |$68132.0             |$1.232556035964325      |
|90401   |19           |6722            |0.002826539720321333  |$62703.0             |$9.328027372805712      |
|90266   |5            |35135           |1.4230823964707557E-4 |$143527.0            |$4.085014942365163      |
|90008   |3015         |32327           |0.09326569121786742   |$36564.0             |$1.1310669100133015     |
|90043   |2802         |44789           |0.0625600035723057    |$38180.0             |$0.8524414476768849     |
|90056   |27           |7827            |0.0034495975469528554 |$84099.0             |$10.74472978152549      |
|90230   |452          |31766           |0.014229049927595543  |$74823.0             |$2.3554429263992946     |
|90301   |102          |36568           |0.002789323999124918  |$37424.0             |$1.0234084445416758     |
|90250   |2            |93193           |2.146083933342633E-5  |$46172.0             |$0.4954449368514803     |
|90278   |1            |40071           |2.4955703626063736E-5 |$107010.0            |$2.6705098450250806     |
|90304   |11           |28210           |3.899326479971641E-4  |$36412.0             |$1.2907479617157036     |
|90302   |31           |29415           |0.0010538840727519973 |$41426.0             |$1.4083290838007818     |
|90254   |2            |19506           |1.0253255408592229E-4 |$111187.0            |$5.70014354557572       |
|90245   |7            |16654           |4.2031944277651017E-4 |$85727.0             |$5.147532124414555      |
+--------+-------------+----------------+----------------------+---------------------+------------------------+
only showing top 20 rows

SHUFFLE_REPLICATE_NL Join Time: 12.394477605819702 seconds
```

Question 3

Implement Query 3 using DataFrame or SQL API. Use hint & explain methods to find out which join strategies the catalyst optimizer uses. Experiment by forcing Spark to use different strategies (between BROADCAST, MERGE, SHUFFLE_HASH, SHUFFLE_REPLICATE_NL) and comment on the results you observe. Which of the available Spark join strategies is (are) the most appropriate and why?

Answer:

Based on the time of execution for joining these datasets, SHUFFLE_REPLICATE_NL and SHUFFLE_HASH are probably the two best joining methods for that dataset and workload. The time of 12.39 seconds for SHUFFLE_REPLICATE_NL demonstrates that replication of smaller datasets across partitions becomes highly effective in this case. It is particularly effective when the one dataset is relatively small compared to the other, enabling replication without imposing heavy overheads on memory space or through the network. For example, SHUFFLE_HASH also performed exceptionally well in 13.01 seconds, taking advantage of hash-

based partitioning to evenly distribute the data across the cluster. The approach is most needed when the join keys are evenly distributed, thus minimizing shuffle cost and skew.

Broadcast Join's time is 14.19 seconds, which also reflects a very good performance with the small-to-fit memory broadcast dataset, thus making it feasible for a dramatically reduced dataset against another during in-memory joins. MERGE Join completed in 14.37 seconds and was still worth considering as long as the datasets were sorted on the join keys since it avoided most of the shuffle and memory overhead. Finally, the Last but not least: Standard Join took 15.45 seconds, the slowest one among all joins, stressing the absence of optimizations such as broadcasting, shuffling, or merging, thus making it unfit for bigger datasets.

In sum, the time parameter of 12.39 seconds makes SHUFFLE_REPLICATE_NL the most appropriate strategy with superior performance to efficiently handling the workload. SHUFFLE_HASH, at 13.01 seconds, is a very strong secondary choice, especially with those dataset characteristics that match well with hash-based partitioning methods. Broadcast Join at 14.1 can be considered where there are smaller datasets or memory constraint situations.

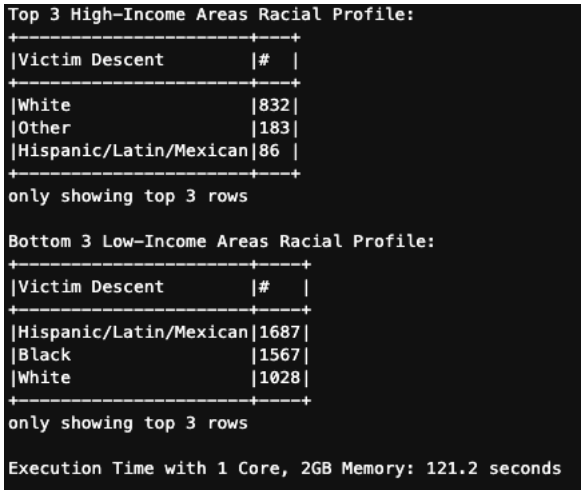
Query 4

Find the racial profile of registered crime victims (`Vict Descent`) in Los Angeles for the year 2015 in the 3 areas with the highest per capita income. Do the same for the 3 areas with the lowest income. Use the mapping of the descent codes to the full description from the Race and Ethnicity codes dataset. The results should be printed in two separate tables from highest to lowest number of victims per racial group (see example result below).

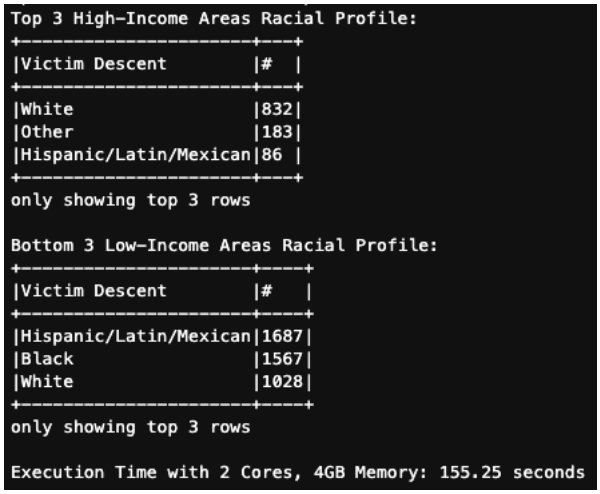
Victim Descent	#
White	413
Black	274
Unknown	132
Hispanic/Latin/Mexican	12

Answer:

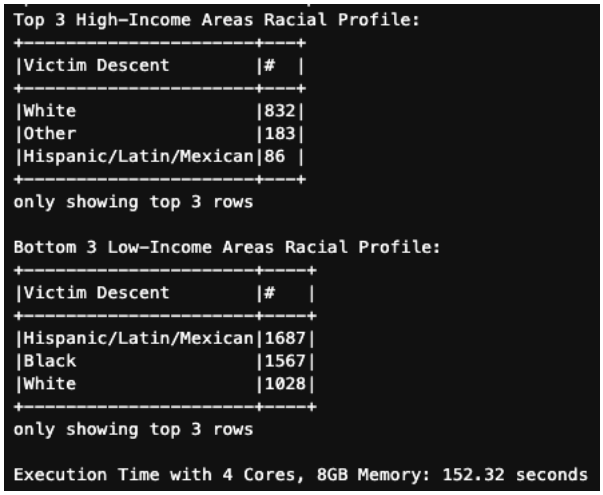
1 core/2 GB memory



2 cores/4GB memory



4 cores/8GB memory



Question 4

Implement Query 4 using the DataFrame or SQL API. Execute your implementation by scaling the total computational resources you will use: Specifically, you are asked to execute your implementation with 2 executors with the following configurations:

• 1 core/2 GB memory

• 2 cores/4GB memory

• 4 cores/8GB memory

Comment on the results. (20%)

Answer:

Efficiency was proven by the fact that the configuration of a single-core machine with 2GB of memory had the fastest execution time of 121.2 seconds, after which the increased execution time, when measured with additional resources of 2 cores and 4GB, ran to 155.25 seconds. Apparently, the overhead created by any additional processes involves dealing with coordination of tasks or task utilization. Using even 4 cores with 8GB further reduces execution time to only 152.32 seconds but never approaches the efficiency of the first configuration.

This indicates that the workload does not scale as well with more resources. Some, or all of these reasons, might include small input data, insufficient parallelizable tasks, or serious bottlenecks in I/O such as reading from the S3. Thus, the configuration of 1 core and 2GB of memory has been found most useful for this workload.

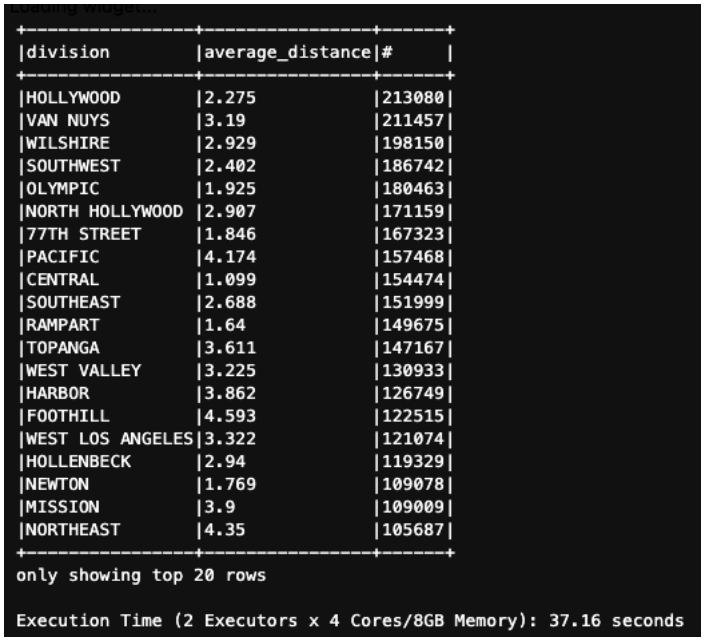
Query 5

Calculate, for each police station, the number of crimes that took place closest to it, as well as its average distance from the locations where the specific incidents occurred. The results should be displayed sorted by number of incidents, in descending order (see example below).

division	average_distance	#
77TH STREET	2.208	7045
RAMPART	2.009	4595
FOOTHILL	3.597	3047
PACIFIC	2.739	2132

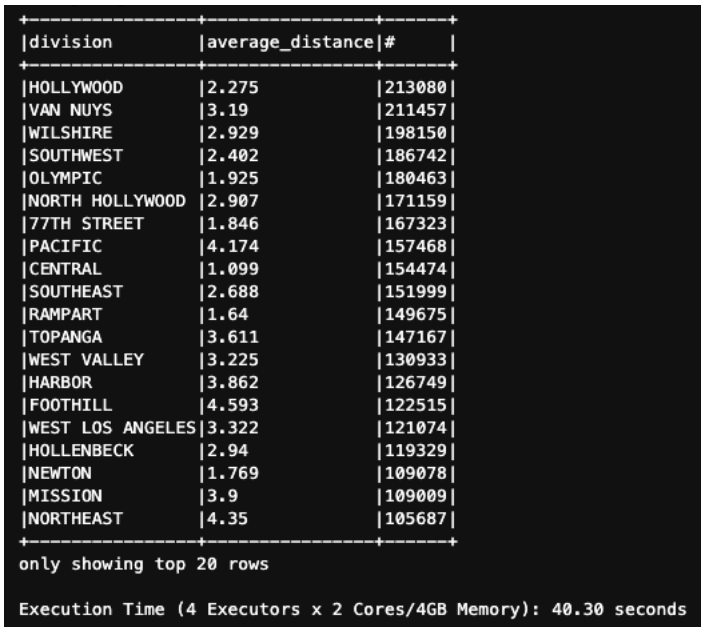
Answer:

2 executors × 4 cores/8GB memory



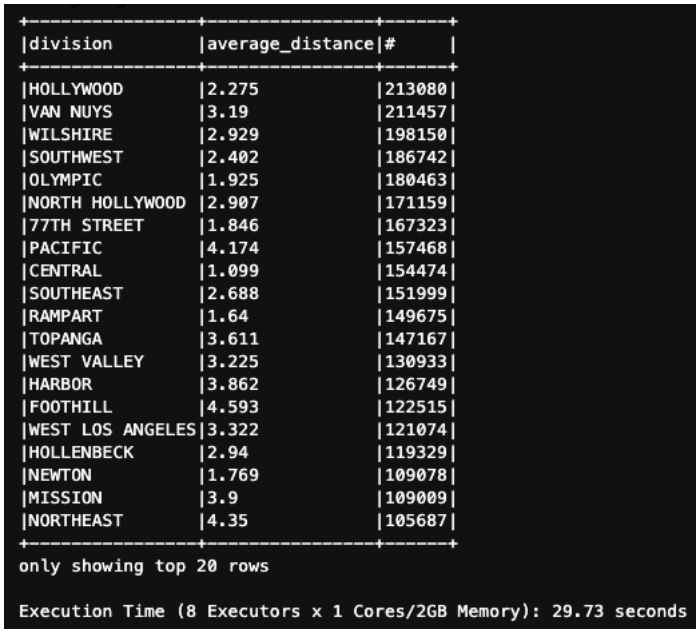
```
+---------------+----------------+------+
|division       |average_distance|#     |
+---------------+----------------+------+
|HOLLYWOOD      |2.275           |213080|
|VAN NUYS       |3.19            |211457|
|WILSHIRE       |2.929           |198150|
|SOUTHWEST      |2.402           |186742|
|OLYMPIC        |1.925           |180463|
|NORTH HOLLYWOOD|2.907           |171159|
|77TH STREET    |1.846           |167323|
|PACIFIC        |4.174           |157468|
|CENTRAL        |1.099           |154474|
|SOUTHEAST      |2.688           |151999|
|RAMPART        |1.64            |149675|
|TOPANGA        |3.611           |147167|
|WEST VALLEY    |3.225           |130933|
|HARBOR         |3.862           |126749|
|FOOTHILL       |4.593           |122515|
|WEST LOS ANGELES|3.322          |121074|
|HOLLENBECK     |2.94            |119329|
|NEWTON         |1.769           |109078|
|MISSION        |3.9             |109009|
|NORTHEAST      |4.35            |105687|
+---------------+----------------+------+
only showing top 20 rows

Execution Time (2 Executors x 4 Cores/8GB Memory): 37.16 seconds
```

4 executors × 2 cores/4GB memory



```
+---------------+----------------+------+
|division       |average_distance|#     |
+---------------+----------------+------+
|HOLLYWOOD      |2.275           |213080|
|VAN NUYS       |3.19            |211457|
|WILSHIRE       |2.929           |198150|
|SOUTHWEST      |2.402           |186742|
|OLYMPIC        |1.925           |180463|
|NORTH HOLLYWOOD|2.907           |171159|
|77TH STREET    |1.846           |167323|
|PACIFIC        |4.174           |157468|
|CENTRAL        |1.099           |154474|
|SOUTHEAST      |2.688           |151999|
|RAMPART        |1.64            |149675|
|TOPANGA        |3.611           |147167|
|WEST VALLEY    |3.225           |130933|
|HARBOR         |3.862           |126749|
|FOOTHILL       |4.593           |122515|
|WEST LOS ANGELES|3.322          |121074|
|HOLLENBECK     |2.94            |119329|
|NEWTON         |1.769           |109078|
|MISSION        |3.9             |109009|
|NORTHEAST      |4.35            |105687|
+---------------+----------------+------+
only showing top 20 rows

Execution Time (4 Executors x 2 Cores/4GB Memory): 40.30 seconds
```

8 executors × 1 core/2 GB memory



```
+---------------+----------------+------+
|division       |average_distance|#     |
+---------------+----------------+------+
|HOLLYWOOD      |2.275           |213080|
|VAN NUYS       |3.19            |211457|
|WILSHIRE       |2.929           |198150|
|SOUTHWEST      |2.402           |186742|
|OLYMPIC        |1.925           |180463|
|NORTH HOLLYWOOD|2.907           |171159|
|77TH STREET    |1.846           |167323|
|PACIFIC        |4.174           |157468|
|CENTRAL        |1.099           |154474|
|SOUTHEAST      |2.688           |151999|
|RAMPART        |1.64            |149675|
|TOPANGA        |3.611           |147167|
|WEST VALLEY    |3.225           |130933|
|HARBOR         |3.862           |126749|
|FOOTHILL       |4.593           |122515|
|WEST LOS ANGELES|3.322          |121074|
|HOLLENBECK     |2.94            |119329|
|NEWTON         |1.769           |109078|
|MISSION        |3.9             |109009|
|NORTHEAST      |4.35            |105687|
+---------------+----------------+------+
only showing top 20 rows

Execution Time (8 Executors x 1 Cores/2GB Memory): 29.73 seconds
```

Question 5

Implement Query 5 using the DataFrame or SQL API. Execute your implementation using a total of 8 cores and 16GB of memory with the following configurations:

• 2 executors × 4 cores/8GB memory

• 4 executors × 2 cores/4GB memory

• 8 executors × 1 core/2 GB memory

Comment on the results. (20%)

Answer:

The best run time was 29.73 seconds when set to 8 executors of 1 core and 2GB's memory. In this setup, increased parallelism was efficient in spreading the work across a higher number of executors despite limited resources per executor.

The configuration of two executors with four cores and 8GB of memory delivered an average execution time of 37.16 seconds. Although much resource was given to each executor, it could not beat the result achieved by the 8-executor combination, possibly owing to lesser-efficient parallelism.

It was the 4-executor system that actually exhibited the slowest performance at 40.30 seconds. It had 2 cores, and each of the 4 executors had 4GB of memory. Although there was a balance between the parallelism and resource allocation , this was not as efficient in terms of resource allocation compared to the other setups.

So it appears that, under this workload and dataset, increasing the number of smaller executors maximises performance.