ΠροχωρημέναΘέματα Βάσεων

ΕξαμηνιαίαΕργασία

Δημοσθένης Αθανασίου(03120041)
Γεώργιος Κίτσιος(03119801)

Ιανουάριος 2024

Περιεχόμενα

Ερωτήματα

Query 1

Να ταξινομηθούν, σε φθίνουσα σειρά, οι ηλικιακές ομάδες των θυμάτων σε περιστατικά που περιλαμβάνουν οποιαδήποτε μορφή "βαριάς σωματικής βλάβης". Θεωρείστε τις εξής ηλικιακές ομάδες:

- **Παιδιά:** < 18
- **Νεαροί ενήλικοι:** 18 – 24
- **Ενήλικοι:** 25 – 64
- **Ηλικιωμένοι:** > 64

Θα αξιοποιήσουμε τα σχετικά dataset :

- **Los Angeles Crime Data (2010-2019)**
- **Los Angeles Crime Data (2020-)**

Αφού πρώτα ορίσουμε τους 4 spark executors για τις υλοποιήσεις μας :

```
%%configure -f
{
    "conf": {
        "spark.executor.instances": "4",
        "spark.executor.memory": "1g",
        "spark.executor.cores": "1",
        "spark.driver.memory": "2g"
    }
}
```

Παρακάτω παρουσιάζεται ο κώδικας που χρησιμοποιήσαμε στο περιβάλλον του Jupyter Notebook σε python για την επίλυση των ζητουμένων με την χρήση RDDs και Dataframes αντίστοιχα. Αρχικά φορτώνουμε τα datasets , αφαιρούμε τα headers , τα ενώνουμε και φιλτράρουμε την στήλη Crm Cd Desc για εγκλήματα με aggravated assault και δημιουργώτις ηλικιακές ομάδες σύμφωνα με τις οποίες τα ομαδοποιώ .Τέλος εκτυπώνω τα σύνολα των θυμάτων κάθε ηλικιακής ομάδας και τους χρόνους εκτέλεσης για κάθε υλοποίηση.

```python
import time
import csv
from io import StringIO

# Start time
start_time_rdd = time.time()

# Load data from both files as RDDs
crime_rdd_2010_2019 = sc.textFile("s3://initial-notebook-data-bucket-dblab-905418150721/CrimeData/Crime_Data_from_2010_to_2019_20241101.csv")
crime_rdd_2020_present = sc.textFile("s3://initial-notebook-data-bucket-dblab-905418150721/CrimeData/Crime_Data_from_2020_to_Present_20241101.cs

# Extract headers
header_2010_2019 = crime_rdd_2010_2019.first()
header_2020_present = crime_rdd_2020_present.first()

# Remove headers from both RDDs
crime_data_2010_2019 = crime_rdd_2010_2019.filter(lambda line: line != header_2010_2019)
crime_data_2020_present = crime_rdd_2020_present.filter(lambda line: line != header_2020_present)

# Combine both RDDs into one
combined_crime_data = crime_data_2010_2019.union(crime_data_2020_present)

# Parse CSV correctly
crime_parsed = combined_crime_data.map(lambda line: list(csv.reader(StringIO(line)))[0]).filter(lambda x: len(x) > 27)

# Filter for "aggravated assault" in crime description
aggravated_assault = crime_parsed.filter(lambda x: "aggravated assault" in x[9].strip().lower())

# Categorize ages
def categorize_age(age):
    try:
        age = int(age)
        if age < 18:
            return "Children"
        elif 18 <= age <= 24:
            return "Young Adults"
        elif 25 <= age <= 64:
            return "Adults"
        else:
            return "Seniors"
    except ValueError:
        return None

age_group_rdd = aggravated_assault.map(lambda x: (categorize_age(x[11]), 1)).filter(lambda x: x[0] is not None)

# Count victims by age group
age_group_counts = age_group_rdd.reduceByKey(lambda a, b: a + b)

# Sort results by count in descending order
sorted_age_groups = age_group_counts.sortBy(lambda x: x[1], ascending=False)

# Collect and display results
print(sorted_age_groups.collect())

# End time
end_time_rdd = time.time()
print(f"RDD Execution Time: {end_time_rdd - start_time_rdd} seconds")
```

```python
from pyspark.sql.functions import col, when, count
import time

# Start time
start_time_df = time.time()

# Load the first dataset as DataFrame
crime_df_2010_2019 = spark.read.csv(
    "s3://initial-notebook-data-bucket-dblab-905418150721/CrimeData/Crime_Data_from_2010_to_2019_20241101.csv",
    header=True,
    inferSchema=True
)

# Rename columns to trim spaces (if needed)
crime_df_2010_2019 = crime_df_2010_2019.toDF(*[col_name.strip() for col_name in crime_df_2010_2019.columns])

# Load the second dataset as DataFrame
crime_df_2020_present = spark.read.csv(
    "s3://initial-notebook-data-bucket-dblab-905418150721/CrimeData/Crime_Data_from_2020_to_Present_20241101.csv",
    header=True,
    inferSchema=True
)

# Rename columns to trim spaces (if needed)
crime_df_2020_present = crime_df_2020_present.toDF(*[col_name.strip() for col_name in crime_df_2020_present.columns])

# Ensure schemas match
crime_df_2020_present = crime_df_2020_present.select(crime_df_2010_2019.columns)

# Combine both DataFrames
combined_crime_df = crime_df_2010_2019.union(crime_df_2020_present)

# Filter for "AGGRAVATED ASSAULT"
filtered_df = combined_crime_df.filter(col("Crm Cd Desc").contains("AGGRAVATED ASSAULT"))

# Categorize ages into groups
categorized_df = filtered_df.withColumn(
    "AgeGroup",
    when(col("Vict Age") < 18, "Children")
    .when((col("Vict Age") >= 18) & (col("Vict Age") <= 24), "Young Adults")
    .when((col("Vict Age") >= 25) & (col("Vict Age") <= 64), "Adults")
    .otherwise("Seniors")
)

# Count victims by age group
result_df = categorized_df.groupBy("AgeGroup").agg(count("*").alias("Count")).orderBy(col("Count").desc())

# Display results
result_df.show()

# End time
end_time_df = time.time()
print(f"DataFrame Execution Time: {end_time_df - start_time_df} seconds")
```

Καθώς και τα αποτελέσματα τους :

```
[('Adults', 121093), ('Young Adults', 33605), ('Children', 15928), ('Seniors', 5985)]
RDD Execution Time: 41.49943780899048 seconds
```

```
+------------+------+
|    AgeGroup| Count|
+------------+------+
|      Adults|121093|
|Young Adults| 33605|
|    Children| 15928|
|     Seniors|  5985|
+------------+------+

DataFrame Execution Time: 19.49064564704895 seconds
```

Όπως ήταν αναμενόμενο παρατηρούμε ότι και οι δύο υλοποιήσεις μας δίνουν το ίδιο αριθμητικό αποτέλεσμα , ωστόσο η υλοποίηση με dataframes είναι αρκετά πιο γρήγορη και αποδοτική .Αυτό συμβαίνει γιατί όπως περιμέναμε τα DataFrames δουλεύουν σε επίπεδο στήλης (columnar storage), που σημαίνει

ότι μπορούν να διαβάσουν μόνο τις απαραίτητες στήλες αντί να φορτώνουν ολόκληρες τις εγγραφές, αξιοποιούν τον catalyst optimizer και χρησιμοποιούν off-heap storage και είναι καλύτερα στη διαχείριση της μνήμης.

Query 2

Να βρεθούν, για κάθε έτος, τα 3 Αστυνομικά Τμήματα με το υψηλότερο ποσοστό κλεισμένων (περατωμένων) υποθέσεων. Να τυπωθούν το έτος, τα ονόματα (τοποθεσίες) των τμημάτων, τα ποσοστά τους καθώς και οι αριθμοί του ranking τους στην ετήσια κατάταξη. Τα αποτελέσματα να δοθούν σε σειρά αύξουσα ως προς το έτος και το ranking (δείτε παράδειγμα στον Πίνακα 2).

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

Πίνακας 2: Υπόδειγμα αποτελέσματος Query 2

α)

Ο κώδικας ξεκινά με τη φόρτωση δύο συνόλων δεδομένων εγκλημάτων (2010–2019 και 2020–σήμερα) από ένα S3 bucket και την ένωση τους σε ένα ενιαίο DataFrame. Στη συνέχεια, φιλτράρει και μετασχηματίζει τα δεδομένα, επιλέγοντας τις απαραίτητες στήλες (ημερομηνία, όνομα περιοχής και κατάσταση υπόθεσης) και δημιουργώντας νέες στήλες για τον αριθμό κλεισμένων υποθέσεων και τον συνολικό αριθμό υποθέσεων, ενώ εξάγει και το έτος από την ημερομηνία. Ακολουθεί ομαδοποίηση των δεδομένων ανά έτος και όνομα περιοχής, όπου υπολογίζεται το ποσοστό κλεισμένων υποθέσεων για κάθε περιοχή. Με βάση αυτά, ορίζεται ένα παράθυρο κατά έτος και κατατάσσονται οι περιοχές με φθίνουσα σειρά ποσοστού κλεισμένων υποθέσεων. Φιλτράρονται οι τρεις κορυφαίες περιοχές για κάθε έτος και τα τελικά αποτελέσματα ταξινομούνται με αύξουσα σειρά ως προς το έτος και τη σειρά κατάταξης. Τέλος, εμφανίζονται τα αποτελέσματα και υπολογίζεται ο χρόνος εκτέλεσης για την εκάστοτε υλοποίηση. Παρακάτω φαίνεται ο κώδικας αλλά και το αποτέλεσμα της υλοποίησης της λύσης με Dataframe:

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, row_number, when, sum as _sum
from pyspark.sql.window import Window
import time

# Create SparkSession
spark = SparkSession.builder \
    .appName("Query2 DataFrame and SQL with Multiple Datasets") \
    .getOrCreate()

# Load datasets from S3
data_2010_2019 = spark.read.csv(
    "s3a://initial-notebook-data-bucket-dblab-905418150721/CrimeData/Crime_Data_from_2010_to_2019_20241101.csv",
    header=True,
    inferSchema=True
)
data_2020_present = spark.read.csv(
    "s3a://initial-notebook-data-bucket-dblab-905418150721/CrimeData/Crime_Data_from_2020_to_Present_20241101.csv",
    header=True,
    inferSchema=True
)

# Combine the two datasets
data_combined = data_2010_2019.union(data_2020_present)

# Start timer for DataFrame API implementation
start_df = time.time()

# Step 1: Filter necessary columns
data_filtered = data_combined.select(
    col("DATE OCC").alias("date_occ"),
    col("AREA NAME").alias("area_name"),
    col("Status Desc").alias("status_desc")
)

# Step 2: Add closed_cases and total_cases columns
data_filtered = data_filtered.withColumn(
    "closed_cases",
    when(~col("status_desc").isin("UNK", "Invest Cont"), 1).otherwise(0)
).withColumn(
    "total_cases",
    when(col("status_desc").isNotNull(), 1).otherwise(0)
)

# Step 3: Extract year from "DATE OCC"
data_filtered = data_filtered.withColumn("year", col("date_occ").substr(7, 4))

# Step 4: Group by year and area_name, calculate closed_case_rate
grouped_data = data_filtered.groupBy("year", "area_name").agg(
    (_sum("closed_cases") / _sum("total_cases") * 100).alias("closed_case_rate")
)

# Step 5: Define window for ranking
window_spec = Window.partitionBy("year").orderBy(col("closed_case_rate").desc())

# Step 6: Add ranking column
ranked_data = grouped_data.withColumn("rank", row_number().over(window_spec))

# Step 7: Filter for top 3 precincts per year
top_3_precincts = ranked_data.filter(col("rank") <= 3)

# Step 8: Order results by year and rank
final_result_df = top_3_precincts.orderBy("year", "rank")

# Stop timer for DataFrame API implementation
end_df = time.time()

# Show final DataFrame results
final_result_df.show(n=final_result_df.count(), truncate=False)

# Calculate execution time for DataFrame API
execution_time_df = end_df - start_df
print(f"DataFrame API Execution Time: {execution_time_df} seconds")
```

year	area_name	closed_case_rate	rank
2010	Rampart	32.84713448949121	1
2010	Olympic	31.515289821999087	2
2010	Harbor	29.36028339237341	3
2011	Olympic	35.040060090135206	1
2011	Rampart	32.4964471814306	2
2011	Harbor	28.51336246316431	3
2012	Olympic	34.29708533302119	1
2012	Rampart	32.46000463714352	2
2012	Harbor	29.509585848956675	3
2013	Olympic	33.58217940999398	1
2013	Rampart	32.1060382916053	2
2013	Harbor	29.723638951488557	3
2014	Van Nuys	32.0215235281705	1
2014	West Valley	31.49754809505847	2
2014	Mission	31.224939855653567	3
2015	Van Nuys	32.265140677157845	1
2015	Mission	30.463762673676303	2
2015	Foothill	30.353001803658852	3
2016	Van Nuys	32.194518462124094	1
2016	West Valley	31.40146437042384	2
2016	Foothill	29.908647228131645	3
2017	Van Nuys	32.0554272517321	1
2017	Mission	31.055387158996968	2
2017	Foothill	30.469700657094183	3
2018	Foothill	30.731346958877126	1
2018	Mission	30.727023319615913	2
2018	Van Nuys	28.905206942590123	3
2019	Mission	30.727411112319235	1
2019	West Valley	30.57974335472044	2
2019	N Hollywood	29.23808669119627	3
2020	West Valley	30.771131982204647	1
2020	Mission	30.14974649215894	2
2020	Harbor	29.693486590038315	3
2021	Mission	30.318115590092276	1
2021	West Valley	28.971087440009363	2
2021	Foothill	27.993757094211126	3
2022	West Valley	26.536367172306498	1
2022	Harbor	26.337538060026098	2
2022	Topanga	26.234013317831096	3
2023	Foothill	26.76076020122974	1
2023	Topanga	26.538022616453986	2
2023	Mission	25.662731120516817	3
2024	N Hollywood	19.598528961078763	1
2024	Foothill	18.620882188721385	2
2024	77th Street	17.586318167150694	3

```
DataFrame API Execution Time: 0.06362700462341309 seconds
```

Αντίστοιχα παρουσιάζουμε τον κώδικα και την λύση για την υλοποίηση με SQL:

```
# Register combined dataset as a temporary SQL view
data_combined.createOrReplaceTempView("crime_data")

# Start timer for SQL API implementation
start_sql = time.time()

# SQL Query to calculate closed_case_rate and rank top 3 precincts
query = """
SELECT year, area_name, closed_case_rate, rank
FROM (
    SELECT
        year,
        area_name,
        (SUM(CASE WHEN status_desc NOT IN ('UNK', 'Invest Cont') THEN 1 ELSE 0 END) * 100.0) /
        SUM(CASE WHEN status_desc IS NOT NULL THEN 1 ELSE 0 END) AS closed_case_rate,
        ROW_NUMBER() OVER (PARTITION BY year ORDER BY
        (SUM(CASE WHEN status_desc NOT IN ('UNK', 'Invest Cont') THEN 1 ELSE 0 END) * 100.0) /
        SUM(CASE WHEN status_desc IS NOT NULL THEN 1 ELSE 0 END) DESC) AS rank
    FROM (
        SELECT
            SUBSTRING(`DATE OCC`, 7, 4) AS year,
            `AREA NAME` AS area_name,
            `Status Desc` AS status_desc
        FROM crime_data
    ) AS year_data
    GROUP BY year, area_name
) AS ranked_data
WHERE rank <= 3
ORDER BY year, rank
"""

# Execute SQL query
final_result_sql = spark.sql(query)

# Stop timer for SQL API implementation
end_sql = time.time()

# Show final SQL results
final_result_sql.show(n=final_result_sql.count(), truncate=False)

# Calculate execution time for SQL API
execution_time_sql = end_sql - start_sql
print(f"SQL API Execution Time: {execution_time_sql} seconds")
```

year	area_name	closed_case_rate	rank
2010	Rampart	32.84713448949121	1
2010	Olympic	31.51528982199909	2
2010	Harbor	29.36028339237341	3
2011	Olympic	35.04006009013520	1
2011	Rampart	32.49644718143060	2
2011	Harbor	28.51336246316431	3
2012	Olympic	34.29708533302119	1
2012	Rampart	32.46000463714352	2
2012	Harbor	29.50958584895668	3
2013	Olympic	33.58217940999398	1
2013	Rampart	32.10603829160530	2
2013	Harbor	29.72363895148855	3
2014	Van Nuys	32.02152352817050	1
2014	West Valley	31.49754809505847	2
2014	Mission	31.22493985565357	3
2015	Van Nuys	32.26514067715784	1
2015	Mission	30.46376267367630	2
2015	Foothill	30.35300180365885	3
2016	Van Nuys	32.19451846212410	1
2016	West Valley	31.40146437042384	2
2016	Foothill	29.90864722813165	3
2017	Van Nuys	32.05542725173210	1
2017	Mission	31.05538715899697	2
2017	Foothill	30.46970065709418	3
2018	Foothill	30.73134695887712	1
2018	Mission	30.72702331961591	2
2018	Van Nuys	28.90520694259012	3
2019	Mission	30.72741111231923	1
2019	West Valley	30.57974335472044	2
2019	N Hollywood	29.23808669119627	3
2020	West Valley	30.77113198220465	1
2020	Mission	30.14974649215894	2
2020	Harbor	29.69348659003831	3
2021	Mission	30.31811559009228	1
2021	West Valley	28.97108744000936	2
2021	Foothill	27.99375709421112	3
2022	West Valley	26.53636717230650	1
2022	Harbor	26.33753806002610	2
2022	Topanga	26.23401331783110	3
2023	Foothill	26.76076020122974	1
2023	Topanga	26.53802261645399	2
2023	Mission	25.66273112051682	3
2024	N Hollywood	19.59852896107876	1
2024	Foothill	18.62088218872138	2
2024	77th Street	17.58631816715069	3

SQL API Execution Time: 0.02743077278137207 seconds

Παρατηρούμε ότι η υλοποίηση με SQL api είναι πιο αποδοτική και γρήγορη.

β)

Για το ερώτημα αυτό παραθέτουμε τον κώδικα Spark που μετατρέπει το κυρίως data set σε parquet2 file format και αποθηκεύει ένα μοναδικό .parquet αρχείο στο S3 bucket της ομάδας μας :

```
# Save the combined dataset as a Parquet file
data_combined.write.parquet("s3://groups-bucket-dblab-905418150721/group6/query3/", mode="overwrite")
```

Έπειτα επιλέγουμε την πιο αποδοτική υλοποίηση με SQL του προηγούμενου ερωτήματος και συγκρίνουμε τον χρόνο εκτέλεσής του όταν εισάγουμε τα δεδομένα με την παραπάνω parquet2 file format :

```
# Ανάγνωση του Parquet αρχείου
parquet_data = spark.read.parquet("s3://groups-bucket-dblab-905418150721/group6/query3/")

# Καταχώρηση του Parquet DataFrame ως SQL view
parquet_data.createOrReplaceTempView("crime_data_parquet")
# Start timer for SQL API implementation
start_sql = time.time()
# Εκτέλεση του SQL Query στο Parquet αρχείο
query = """
SELECT year, area_name, closed_case_rate, rank
FROM (
    SELECT
        year,
        area_name,
        (SUM(CASE WHEN status_desc NOT IN ('UNK', 'Invest Cont') THEN 1 ELSE 0 END) * 100.0) /
        SUM(CASE WHEN status_desc IS NOT NULL THEN 1 ELSE 0 END) AS closed_case_rate,
        ROW_NUMBER() OVER (PARTITION BY year ORDER BY
        (SUM(CASE WHEN status_desc NOT IN ('UNK', 'Invest Cont') THEN 1 ELSE 0 END) * 100.0) /
        SUM(CASE WHEN status_desc IS NOT NULL THEN 1 ELSE 0 END) DESC) AS rank
    FROM (
        SELECT
            SUBSTRING(`DATE OCC`, 7, 4) AS year,
            `AREA NAME` AS area_name,
            `Status Desc` AS status_desc
        FROM crime_data_parquet
    ) AS year_data
    GROUP BY year, area_name
) AS ranked_data
WHERE rank <= 3
ORDER BY year, rank
"""

# Εκτέλεση του ερωτήματος
final_result_sql = spark.sql(query)
# Stop timer for SQL API implementation
end_sql = time.time()

# Εμφάνιση όλων των αποτελεσμάτων
final_result_sql.show(n=final_result_sql.count(), truncate=False)
# Calculate execution time for SQL API
execution_time_sql = end_sql - start_sql
print(f"SQL API Execution Time: {execution_time_sql} seconds")
```

year	area_name	closed_case_rate	rank
2010	Rampart	32.84713448949121	1
2010	Olympic	31.51528982199909	2
2010	Harbor	29.36028339237341	3
2011	Olympic	35.04006009013520	1
2011	Rampart	32.49644718143060	2
2011	Harbor	28.51336246316431	3
2012	Olympic	34.29708533302119	1
2012	Rampart	32.46000463714352	2
2012	Harbor	29.50958584895668	3
2013	Olympic	33.58217940999398	1
2013	Rampart	32.10603829160530	2
2013	Harbor	29.72363895148855	3
2014	Van Nuys	32.02152352817050	1
2014	West Valley	31.49754809505847	2
2014	Mission	31.22493985565357	3
2015	Van Nuys	32.26514067715784	1
2015	Mission	30.46376267367630	2
2015	Foothill	30.35300180365885	3
2016	Van Nuys	32.19451846212410	1
2016	West Valley	31.40146437042384	2
2016	Foothill	29.90864722813165	3
2017	Van Nuys	32.05542725173210	1
2017	Mission	31.05538715899697	2
2017	Foothill	30.46970065709418	3
2018	Foothill	30.73134695887712	1
2018	Mission	30.72702331961591	2
2018	Van Nuys	28.90520694259012	3
2019	Mission	30.72741111231923	1
2019	West Valley	30.57974335472044	2
2019	N Hollywood	29.23808669119627	3
2020	West Valley	30.77113198220465	1
2020	Mission	30.14974649215894	2
2020	Harbor	29.69348659003831	3
2021	Mission	30.31811559009228	1
2021	West Valley	28.97108744000936	2
2021	Foothill	27.99375709421112	3
2022	West Valley	26.53636717230650	1
2022	Harbor	26.33753806002610	2
2022	Topanga	26.23401331783110	3
2023	Foothill	26.76076020122974	1
2023	Topanga	26.53802261645399	2
2023	Mission	25.66273112051682	3
2024	N Hollywood	19.59852896107876	1
2024	Foothill	18.62088218872138	2
2024	77th Street	17.58631816715069	3

SQL API Execution Time: 0.021721124649047 85 seconds

Παρατηρούμε ότι η εισαγωγή των δεδομένων σε μορφή parquet βελτιώνει την απόδοση και την ταχύτητα όπως ήταν αναμενόμενο.

Query 3

Χρησιμοποιώντας ως αναφορά τα δεδομένα της απογραφής 2010 για τον πληθυσμό και εκείνα της απογραφής του 2015 για το εισόδημα ανα νοικοκυριό, να υπολογίσετε για κάθε περιοχή του Los Angeles τα παρακάτω: Το μέσο ετήσιο εισόδημα ανά άτομο και την αναλογία συνολικού αριθμού εγκλημάτων ανά άτομο. Τα αποτελέσματα να συγκεντρωθούν σε ένα πίνακα. Αρχικά φορτώσαμε τα 2010_Census_Blocks.geojson, LA_income_2015.csv, Crime_Data_from_2010_to_2019_20241101.csv και Crime_Data_from_2020_to_Present_20241101.csv. Μετά δημιουργήσαμε τον πίνακα LA_areas κάνοντας group by COMM και βάζοντας στη στήλη geometry τη γεωγραφική σύμπτυξη όλων των τετραγώνων σε κάθε περιοχή. Στη συνέχεια, μετά από κατάλληλη επεξεργασία, κάναμε join το income με το census_block με βάση τα zip codes. Μετά κάναμε group με βάση τη στήλη "COMM" βάζοντας στον πίνακα που προκύπτει το άθροισμα των πληθυσμών, το άθροισμα των νοικοκυριών καθώς και το μέσο εισόδημα ανά νοικοκυριό για κάθε κοινότητα. Στη συνέχεια, για να βρούμε σε ποιες περιοχές έγιναν τα εγκλήματα κάναμε join χρησιμοποιώντας την συνάρτηση ST_Within και έπειτα κάναμε group by COMM αθρόιζοντας τα εγκλήματα ανά περιοχή στη στήλη crime_count. Τέλος, κάναμε join τους δύο επιμέρους πίνακες με βάση τη στήλη COMM και βάλαμε τη ζητούμενη πληροφορία στον τελικό πίνακα.

COMM	Avg_Median_Income	crimes_per_person
Van Nuys	43827.914666666664	0.9170415838361292
North Hills	56409.98540145985	0.655059866962306
Northridge	60909.710144927536	0.8133372019923594
Encino	88326.6334231806	0.703747432052705
North Hollywood	46568.82010582011	0.8013964204433706
Canoga Park	52561.75073313783	0.8960521181480413
Reseda	51637.0938697318	0.6108411423161002
Panorama City	37423.00787401575	0.6498237536867851
Lake Balboa	49953.90217391304	0.6017674035141959
Winnetka	62067.63815789474	0.6688561626642122
Reseda Ranch	51743.6	0.7724644128113879
Westhills	100075.0	0.01278772378516624
West Hills	80861.76146788991	0.4833172462369238
Burbank	67848.98430899215	0.009763886200890266
Tarzana	68257.6	0.7385797062390513
Sun Valley	52475.485537190085	0.6616657237060304
Valley Glen	48792.15286624204	0.5546786523216308
Woodland Hills	86340.41317365269	0.7102428855689965
Mid-city	46571.0	0.8071692586651789
Glendale	63853.143798024146	7.093715281218867E-4

only showing top 20 rows

Χρησιμοποιώντας την εντολή joined_data_income.explain(mode="formatted") παρατηρήσαμε ότι ο catalyst optimizer χρησιμοποιεί τη στρατηγική "Broadcast Hash Join" για να κάνει join. Η επιλογή αυτή ήταν αποδοτική διότι ο πίνακας

income data ήταν αρκετά μικρός έτσι ώστε να γίνει broadcast σε όλους τους executors. Οι υπόλοιπες στρατηγικές δεν είναι τοσο αποδοτικές για την περίπτωση μας. Για παράδειγμα η Sort Merge Join είναι αποδοτική για μεγάλα δεδομένα αλλά απαιτείται ταξινόμηση. Επίσης η Shuffled Hash Join χρησιμοποιείται όταν δεν είναι αποδοτική η broadcast για μεσαίου όγκου datasets. Η SHUFFLE REPLICATE χρησιμοποιείται σε ειδικές περιπτώσεις και απαιτεί πολλούς πόρους.

Query 4

Να βρεθεί το φυλετικό προφίλ των καταγεγραμμένων θυμάτων εγκλημάτων (Vict Descent) στο Los Angeles για το έτος 2015 στις 3 περιοχές με το υψηλότερο κατά κεφαλήν εισόδημα. Να γίνει το ίδιο για τις 3 περιοχές με το χαμηλότερο εισόδημα. Να χρησιμοποιήσετε την αντιστοίχιση των κωδικών καταγωγής με την πλήρη περιγραφή από το σύνολο δεδομένων Race and Ethnicity codes. Τα αποτελέσματα να τυπωθούν σε δύο ξεχωριστούς πίνακες από το υψηλότερο στο χαμηλότερο αριθμό θυμάτων ανά φυλετικό γκρουπ (δείτε παράδειγμα αποτελέσματος στον Πίνακα 3).

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

Πίνακας 3: Υπόδειγμα αποτελέσματος Query 4

Στο ερώτημα αυτό θα εκτελέσουμε τον κώδικα τρεις φορές κάθε φορά με ένα από τα παρακάτων configuration :

```
%%configure -f
{
    "conf": {
        "spark.executor.instances": "2",
        "spark.executor.memory": "2g",
        "spark.executor.cores": "1",
        "spark.driver.memory": "2g"
    }
}
```

```
%%configure -f
{
    "conf": {
        "spark.executor.instances": "2",
        "spark.executor.memory": "4g",
        "spark.executor.cores": "2",
        "spark.driver.memory": "4g"
    }
}
```

```
%%configure -f
{
    "conf": {
        "spark.executor.instances": "2",
        "spark.executor.memory": "8g",
        "spark.executor.cores": "4",
        "spark.driver.memory": "8g"
    }
}
```

Αρχικά, φορτώνεται ένα αρχείο GeoJSON από το S3 που περιέχει δεδομένα απογραφής (Census Blocks) και επεξεργάζεται ώστε να δημιουργηθεί ένα DataFrame με τις γεωμετρικές πληροφορίες και τις ιδιότητες του αρχείου. Στη συνέχεια, φορτώνονται δεδομένα εγκλημάτων από το S3 και μετασχηματίζονται ώστε να περιέχουν μια γεωμετρική στήλη (geom) βασισμένη στις συντεταγμένες (LAT, LON). Επιπλέον, φορτώνονται δεδομένα εισοδήματος από το S3, όπου η στήλη εισοδήματος καθαρίζεται από χαρακτήρες όπως και , και μετατρέπεται σε ακέραιο τύπο. Στη συνέχεια, πραγματοποιείται η πρώτη ένωση (join) μεταξύ των δεδομένων απογραφής και εισοδήματος χρησιμοποιώντας το ταχυδρομικό κώδικα (ZIP Code και ZCTA10). Ακολουθεί η φόρτωση δεδομένων φυλετικής και εθνοτικής κατηγοριοποίησης και πραγματοποιείται δεύτερη ένωση μεταξύ των δεδομένων απογραφής-εισοδήματος και των δεδομένων εγκλημάτων με χρήση γεωχωρικής συνθήκης , για να εντοπιστούν τα εγκλήματα που βρίσκονται εντός των γεωμετρικών ορίων κάθε περιοχής. Μετατρέπεται η ημερομηνία του εγκλήματος σε τύπο Timestamp και φιλτράρονται τα δεδομένα για τα εγκλήματα που συνέβησαν το 2015. Στη συνέχεια, πραγματοποιείται τρίτη ένωση με τα δεδομένα φυλετικής και εθνοτικής κατηγοριοποίησης, ώστε να προστεθούν περισσότερες πληροφορίες για την καταγωγή των θυμάτων. Υπολογίζονται οι τρεις περιοχές με το υψηλότερο και χαμηλότερο μέσο εισόδημα, και τα δεδομένα εγκλημάτων φιλτράρονται για αυτές τις περιοχές.
Για τις τρεις περιοχές με το υψηλότερο και χαμηλότερο εισόδημα, υπολογίζονται οι φυλετικές κατηγορίες των θυμάτων, μετρώντας την εμφάνιση κάθε κατηγορίας και ταξινομώντας τα αποτελέσματα κατά φθίνουσα σειρά. Τέλος, εμφανίζονται τα αποτελέσματα για τα φυλετικά προφίλ των περιοχών με το υψηλότερο και χαμηλότερο εισόδημα και υπολογίζεται ο συνολικός χρόνος εκτέλεσης του κώδικα.Τα αποτελέσματα για κάθε εκτέλεση φαίνονται παρακάτω:

```
Racial Profile for the Top 3 Highest Income Areas:
+--------------------+---+
|      Victim Descent|  #|
+--------------------+---+
|               White|645|
|               Other|123|
|Hispanic/Latin/Me...| 71|
|             Unknown| 48|
|               Black| 38|
|         Other Asian| 24|
|             Chinese|  1|
|American Indian/A...|  1|
+--------------------+---+

Racial Profile for the Top 3 Lowest Income Areas:
+--------------------+---+
|      Victim Descent|  #|
+--------------------+---+
|Hispanic/Latin/Me...|799|
|               Black|329|
|               White|284|
|               Other|187|
|         Other Asian| 37|
|             Unknown|  9|
|              Korean|  4|
|American Indian/A...|  1|
|    Pacific Islander|  1|
+--------------------+---+

Execution Time: 151.16 seconds
```

```
Racial Profile for the Top 3 Highest Income Areas:
+--------------------+---+
|      Victim Descent|  #|
+--------------------+---+
|               White|645|
|               Other|123|
|Hispanic/Latin/Me...| 71|
|             Unknown| 48|
|               Black| 38|
|         Other Asian| 24|
|             Chinese|  1|
|American Indian/A...|  1|
+--------------------+---+

Racial Profile for the Top 3 Lowest Income Areas:
+--------------------+---+
|      Victim Descent|  #|
+--------------------+---+
|Hispanic/Latin/Me...|799|
|               Black|329|
|               White|284|
|               Other|187|
|         Other Asian| 37|
|             Unknown|  9|
|              Korean|  4|
|American Indian/A...|  1|
|    Pacific Islander|  1|
+--------------------+---+

Execution Time: 142.59 seconds
```

```
Racial Profile for the Top 3 Highest Income Areas:
+--------------------+---+
|      Victim Descent|  #|
+--------------------+---+
|               White|645|
|               Other|123|
|Hispanic/Latin/Me...| 71|
|             Unknown| 48|
|               Black| 38|
|         Other Asian| 24|
|             Chinese|  1|
|American Indian/A...|  1|
+--------------------+---+

Racial Profile for the Top 3 Lowest Income Areas:
+--------------------+---+
|      Victim Descent|  #|
+--------------------+---+
|Hispanic/Latin/Me...|799|
|               Black|329|
|               White|284|
|               Other|187|
|         Other Asian| 37|
|             Unknown|  9|
|              Korean|  4|
|American Indian/A...|  1|
|    Pacific Islander|  1|
+--------------------+---+

Execution Time: 136.01 seconds
```

Παρατηρώντας τα αποτελέσματα των τριών εκτελέσεων με διαφορετικές ρυθμίσεις, παρατηρείται ότι ο χρόνος εκτέλεσης μειώνεται προοδευτικά από την πρώτη στη δεύτερη και τρίτη εκτέλεση, ενώ τα παραγόμενα αποτελέσματα παραμένουν απολύτως αμετάβλητα. Αυτό υποδεικνύει ότι οι βελτιώσεις στις ρυθμίσεις ή στη χρήση των διαθέσιμων πόρων έχουν άμεσο αντίκτυπο στην απόδοση, χωρίς να επηρεάζουν την ακρίβεια ή την αξιοπιστία των αποτελεσμάτων.

Query 5

Να υπολογιστεί, ανά αστυνομικό τμήμα, ο αριθμός εγκλημάτων που έλαβαν χώρα πλησιέστερα σε αυτό, καθώς και η μέση απόστασή του από τις τοποθεσίες όπου σημειώθηκαν τα συγκεκριμένα περιστατικά. Τα αποτελέσματα να εμφανιστούν ταξινομημένα κατά αριθμό περιστατικών, με φθίνουσα σειρά (δείτε παράδειγμα στον Πίνακα 4).

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

Πίνακας 4: Υπόδειγμα αποτελέσματος Query 5.

Τρέξαμε τον κώδικα με τα ακόλουθα 3 configurations:

```
%%configure -f
{
    "conf": {
        "spark.executor.instances": "2",
        "spark.executor.memory": "8g",
        "spark.executor.cores": "4",
        "spark.driver.memory": "2g"
    }
}
```

```
%%configure -f
{
    "conf": {
        "spark.executor.instances": "4",
        "spark.executor.memory": "4g",
        "spark.executor.cores": "2",
        "spark.driver.memory": "2g"
    }
}
```

```
%%configure -f
{
    "conf": {
        "spark.executor.instances": "8",
        "spark.executor.memory": "2g",
        "spark.executor.cores": "1",
        "spark.driver.memory": "2g"
    }
}
```

Παρατηρήσαμε ότι τα ερωτήματα έτρεξαν σε παρόμοιο χρόνο και για τα 3 configurations. Αρχικά, δημιουργήθηκαν τα DataFrames stations_df και crimes_df και κατασκευάστηκαν οι στήλες station_point και crime_point από τις αντίστοιχες τιμές γεωγραφικού πλάτους (latitude) και γεωγραφικού μήκους (longitude).

Στη συνέχεια, πραγματοποιήθηκε *cross join* ανάμεσα στα δύο DataFrames, υπολογίζοντας στη στήλη Distance του νέου DataFrame, crimes_and_stations, τις αποστάσεις μεταξύ όλων των τοποθεσιών των εγκλημάτων και όλων των τοποθεσιών των αστυνομικών τμημάτων.

Τέλος, αφού μετατρέψαμε το παραπάνω dataframe σε temporary view εφαρμόσαμε το κατάλληλο sql query και καταλήξαμε στο τελικό αποτέλεσμα.

DIVISION	Average Distance	Number of Crimes
HOLLYWOOD	2.076263960178721	224340
VAN NUYS	2.953369742819789	210134
SOUTHWEST	2.191398805780884	188901
WILSHIRE	2.5926655329787796	185996
77TH STREET	1.7165449719701007	171827
OLYMPIC	1.7236036971780915	170897
NORTH HOLLYWOOD	2.6430060941415636	167854
PACIFIC	3.850070655307896	161359
CENTRAL	0.992476437456893	153871
RAMPART	1.5345341879190046	152736
SOUTHEAST	2.421866215888184	152176
WEST VALLEY	3.035671216314083	138643
TOPANGA	3.296954841755553	138217
FOOTHILL	4.2509217084249915	134896
HARBOR	3.702561599356505	126747
HOLLENBECK	366.92130446148906	119294
WEST LOS ANGELES	2.792457289034107	115781
NEWTON	1.6346357397097449	111110
NORTHEAST	3.6236655246040765	108109
MISSION	3.690942614278604	103355
DEVONSHIRE	2.8247654128008253	77094

Github repository

Παρακάτω παραθέτουμε και τον σύνδεσμο για το αποθετήριο github όπου βρίσκεται όλος ο κώδικας σχετικά με queries της παραπάνω εργασίας: GitHub Reposit https://github.com/giorgoskitsios/advanced_databases_project