



Προχωρημένα Θέματα Βάσεων Δεδομένων

Εξαμηνιαία Εργασία

Ακαδημαϊκό έτος 2025-26, 9ο Εξάμηνο

Διδάσκων: Δημήτριος Τσουμάκος

Υπεύθυνος Εργαστηρίου: Νικόλαος Χαλβαντζής

Περιεχόμενα:

- *Περιγραφή Εργασίας*
- *Query 1:* Κατάταξη ηλικιακών ομάδων θυμάτων σε περιστατικά βαριάς σωματικής βλάβης.
- *Query 2:* Διαχρονική ανάλυση των κυριότερων φυλετικών ομάδων θυμάτων (Top-3) ανά έτος.
- *Query 3:* Ανάλυση συχνότητας μεθόδων διάπραξης εγκλημάτων (Modus Operandi).
- *Query 4:* Γεωχωρική συσχέτιση αστυνομικών τμημάτων με πλήθος και απόσταση καταγεγραμμένων εγκλημάτων.
- *Query 5:* Συσχέτιση μέσου εισοδήματος και δείκτη εγκληματικότητας στις περιοχές με το υψηλότερο και χαμηλότερο εισόδημα (2020-2021).

Στοιχεία Ομάδας:

- *Αριθμός Ομάδας (groupID):* 11
- *Όνοματεπώνυμα και AM Μελών:*
 - Νικόλαος Πανταζόπουλος (03121844)
 - Μαγδαληνή Μαρία Πλιάτσικα (03121220)
- *GitHub Link:* <https://github.com/Nickp03/Big-Data-Analysis-with-Hadoop-and-Spark>

Περιγραφή Εργασίας

Η παρούσα εξαμηνιαία εργασία, που εκπονήθηκε στο πλαίσιο του μαθήματος "Προχωρημένα Θέματα Βάσεων Δεδομένων" της Σχολής HMMY του ΕΜΠ, στοχεύει στην εξοικείωση με την επεξεργασία και ανάλυση Μεγάλων Δεδομένων (Big Data) μέσω της πλατφόρμας Apache Spark σε περιβάλλον AWS Cloud. Αξιοποιώντας ένα σύνολο δεδομένων καταγραφής εγκλημάτων στο Los Angeles, η εργασία εξετάζει την υλοποίηση πέντε (5) σύνθετων ερωτημάτων (Queries) που καλύπτουν ανάλυση τάσεων, γεωγραφική επεξεργασία με τη βιβλιοθήκη Apache Sedona και συσχέτιση κοινωνικοοικονομικών δεικτών. Μέσω της συγκριτικής αξιολόγησης διαφορετικών Spark APIs (RDD, DataFrame, SQL) και στρατηγικών ένωσης (Join Strategies), καθώς και πειραμάτων κλιμάκωσης (scalability) με μεταβλητούς υπολογιστικούς πόρους, η μελέτη αναδεικνύει τα πλεονεκτήματα του Catalyst Optimizer και τη σημασία της βελτιστοποίησης της απόδοσης σε κατανευμημένα συστήματα.

Για το benchmarking του χρόνου περιμένουμε πριν τρέξουμε διαφορετικές υλοποιήσεις έτσι ώστε το spark να μην “θυμάται” τα αποτελέσματα της προηγούμενης και να πάρουμε αποτελέσματα πιο κοντά στα θεωρητικά.

Query 1

Το Query εκτελέστηκε με τρεις υλοποιήσεις (DataFrame API χωρίς UDF, DataFrame API με UDF, RDD API) όπως ζητείται.

Υλοποίηση με DataFrame API χωρίς UDF

Η υλοποίηση αξιοποιεί αποκλειστικά το DataFrame API του Spark και ενσωματωμένες συναρτήσεις (filter, withColumn, when, groupBy, orderBy) επιτρέποντας στον Catalyst optimizer να βελτιστοποιήσει πλήρως το execution plan.

Ο καθορισμός των ηλικιακών ομάδων γίνεται δηλωτικά μέσω when/otherwise, αποφεύγοντας Python UDFs και άρα το κόστος σειριοποίησης και τη μετάβαση εκτός JVM. Η υλοποίηση με DataFrame χωρίς UDF ήταν η ταχύτερη καθώς όλες οι πράξεις υλοποιούνται με ενσωματωμένες συναρτήσεις του Spark στη JVM. Η υλοποίηση χρειάστηκε περίπου 10 (± 1) δευτερόλεπτα.

Execution Time: 11.0353 seconds	
age_group	count
Adults	121660
Young Adults	33758
Children	16014
Elderly	6011

Υλοποίηση με DataFrame API και UDF

Η υλοποίηση ακολουθεί την ίδια λογική επεξεργασίας με την πρώτη προσέγγιση (φίλτραρισμα περιστατικών “aggravated assault”, μετατροπή τύπου ηλικίας, ομαδοποίηση και καταμέτρηση), όμως ο καθορισμός των ηλικιακών ομάδων γίνεται μέσω Python UDF.

Η υλοποίηση με DataFrame API με UDF ήταν πιο αργή σε σχέση με την αντίστοιχη υλοποίηση μόνο με DataFrame API, λόγω του κόστους serialization μεταξύ JVM και Python

και του γεγονότος ότι ο Catalyst optimizer αντιμετωπίζει το UDF σαν black box και άρα δεν μπορεί να βελτιστοποιήσει τη λογική του. Η υλοποίηση χρειάστηκε περίπου 14 (± 2) δευτερόλεπτα.

Execution Time: 15.1313 seconds	
age_group	count
Adults	121660
Young Adults	33758
Children	16014
Elderly	6011

Υλοποίηση με RDD API

Η υλοποίηση με το RDD API εκφράζει όλη τη λογική του Query 1 σε χαμηλού επιπέδου μετασχηματισμούς (map, filter, reduceByKey, sortBy). Ο έλεγχος του φιλτραρίσματος (“aggravated assault”), η επικύρωση της ηλικίας και η ανάθεση ηλικιακής ομάδας υλοποιούνται χειροκίνητα σε Python συναρτήσεις χωρίς τη χρήση του DataFrame API.

Η υλοποίηση με RDD API παρουσίασε τη χειρότερη επίδοση, καθώς δεν επωφελείται από τον Catalyst optimizer ούτε από τις βελτιστοποιήσεις του DataFrame API. Η διαχείριση των δεδομένων γίνεται σε επίπεδο Python objects, κάτι που αυξάνει το κόστος serialization και shuffling, ενώ κάνει λιγότερο αποδοτική διαχείριση μνήμης. Η υλοποίηση χρειάστηκε περίπου 20 (± 2) δευτερόλεπτα, χρόνο διπλάσιο από την υλοποίηση με DataFrame API.

Execution Time: 22.8356 seconds	
Adults	121660
Young Adults	33758
Children	16014
Elderly	6011

Query 2

Και οι δύο υλοποιήσεις (DataFrame και SQL) παρήγαγαν το ίδιο αποτέλεσμα και είχαν παραπλησίους χρόνους εκτέλεσης (περίπου 35 δευτερόλεπτα), με το SQL API (δεύτερη εικόνα) να είναι ελαφρώς πιο αργό σε ορισμένες μετρήσεις, αλλά ουσιαστικά ισοδύναμο. Η ισοδυναμία στην απόδοση οφείλεται στον **Catalyst Optimizer** του Spark, ο οποίος λειτουργεί ως ο “εγκέφαλος” πίσω και από τα δύο API. Όπως αναφέρεται στις διαφάνειες, είτε γράφουμε κώδικα σε SQL είτε σε DataFrame, το Spark μετατρέπει το ερώτημα σε ένα **Unresolved Logical Plan**. Στη συνέχεια, ο Catalyst αναλύει το πλάνο, το βελτιστοποιεί (Logical Optimization) και τελικά παράγει το ίδιο **Physical Plan** (φυσικό πλάνο εκτέλεσης) και για τις δύο περιπτώσεις. Επομένως, οι όποιες μικρές διαφορές στο χρόνο οφείλονται κυρίως στο ελάχιστο overhead της ανάλυσης του SQL string, αλλά η βαριά επεξεργασία (join, grouping) εκτελείται με τον ίδιο ακριβώς τρόπο στο backend (μέσω RDDs και Tungsten engine).

Execution Time: 34.7147 seconds			
year	Victim Descent	#	%
2025	Unknown	37	38.1
2025	Hispanic/Latin/Mexican	34	35.1
2025	White	13	13.4
2024	Unknown	49188	38.6
2024	Hispanic/Latin/Mexican	28576	22.4
2024	White	22958	18.0
2023	Hispanic/Latin/Mexican	69401	29.9
2023	Unknown	59529	25.6
2023	White	44615	19.2
2022	Hispanic/Latin/Mexican	73111	31.1
2022	Unknown	52130	22.2
2022	White	46695	19.8
2021	Hispanic/Latin/Mexican	63676	30.3
2021	Unknown	46499	22.2
2021	White	44523	21.2
2020	Hispanic/Latin/Mexican	61606	30.8
2020	Unknown	43958	22.0
2020	White	42638	21.3
2019	Hispanic/Latin/Mexican	72458	33.1
2019	White	48863	22.3

Execution Time: 35.2878 seconds			
year	Victim Descent	#	%
2025	Unknown	37	38.1
2025	Hispanic/Latin/Mexican	34	35.1
2025	White	13	13.4
2024	Unknown	49188	38.6
2024	Hispanic/Latin/Mexican	28576	22.4
2024	White	22958	18.0
2023	Hispanic/Latin/Mexican	69401	29.9
2023	Unknown	59529	25.6
2023	White	44615	19.2
2022	Hispanic/Latin/Mexican	73111	31.1
2022	Unknown	52130	22.2
2022	White	46695	19.8
2021	Hispanic/Latin/Mexican	63676	30.3
2021	Unknown	46499	22.2
2021	White	44523	21.2
2020	Hispanic/Latin/Mexican	61606	30.8
2020	Unknown	43958	22.0
2020	White	42638	21.3
2019	Hispanic/Latin/Mexican	72458	33.1
2019	White	48863	22.3

Query 3

Στο ερώτημα αυτό εξετάστηκαν διαφορετικές στρατηγικές ένωσης (Join Strategies) μεταξύ του μεγάλου πίνακα εγκλημάτων και του μικρού πίνακα MO_codes:

- BROADCAST Join (~23 sec - Η ταχύτερη): Η στρατηγική αυτή ήταν η πιο γρήγορη διότι ο πίνακας MO_codes είναι πολύ μικρός. Το Spark "εκπέμπει" (broadcasts) ολόκληρο τον μικρό πίνακα σε όλους τους Executors. Έτσι, αποφεύγεται το Shuffle του μεγάλου πίνακα εγκλημάτων. Κάθε executor κάνει το join τοπικά με τα δεδομένα που ήδη έχει, χωρίς να χρειαστεί να μετακινήσει terabytes δεδομένων μέσω του δικτύου.

```

--- Testing Strategy: BROADCAST ---
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- BroadcastHashJoin [MO_code#963], [MO_code#153], Inner, BuildRight, false
  :- HashAggregate(keys=[MO_code#963], functions=[count(1)], schema specialized)
  :  +- Exchange hashpartitioning(MO_code#963, 1000), ENSURE_REQUIREMENTS, [plan_id=3262]
  :    +- HashAggregate(keys=[MO_code#963], functions=[partial_count(1)], schema specialized)
  :      +- Filter (length(MO_code#963) > 0)
  :        +- Generate explode(split(Mocodes#10, , -1)), false, [MO_code#963]
  :          +- Union
  :            :- Filter isnotnull(Mocodes#10)
  :              +- FileScan csv [Mocodes#10] Batched: false, DataFilters: [isnotnull(Mocodes#10)], Format: CSV, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/LA_C..., PartitionFilters: [], PushedFilters: [IsNotNull(Mocodes)], ReadSchema: struct<Mocodes:string>
  :                +- Filter isnotnull(Mocodes#77)
  :                  +- FileScan csv [Mocodes#77] Batched: false, DataFilters: [isnotnull(Mocodes#77)], Format: CSV, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/M0_c..., PartitionFilters: [], PushedFilters: [IsNotNull(Mocodes)], ReadSchema: struct<Mocodes:string>
  +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, true]), false, [plan_id=3265]
    +- Project [split(value#151, , 2)[0] AS MO_code#153, split(value#151, , 2)[1] AS MO_desc#154]
      +- Filter ((length(split(value#151, , 2)[0]) > 0) AND isnotnull(split(value#151, , 2)[0]))
        +- FileScan text [value#151] Batched: false, DataFilters: [(length(split(value#151, , 2)[0]) > 0), isnotnull(split(value#151, , 2)[0])], Format: Text, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/M0_c..., PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value:string>

Time taken for BROADCAST: 23.3821 seconds
+-----+-----+
| MO_code | MO_desc | count |
+-----+-----+
| 0344 | Removes vict property | 1002900 |
| 1822 | Stranger | 548422 |
| 0416 | Hit-Hit w/ weapon | 404773 |
| 0329 | Vandalized | 377536 |
| 0913 | Victim knew Suspect | 278618 |
| 2000 | Domestic violence | 256188 |
| 1300 | Vehicle involved | 219082 |
| 0400 | Force used | 213165 |
| 1402 | Evidence Booked (any crime) | 177478 |
| 1609 | Smashed | 131229 |
| 1309 | Susp uses vehicle | 122108 |
| 1202 | Victim was aged (60 & over) or blind/physically disabled/unable to care for self | 120238 |
| 0325 | Took merchandise | 120159 |
| 1814 | Susp is/was current/former boyfriend/girlfriend | 118073 |
| 0444 | Pushed | 116763 |
| 1501 | Other MO (see rpt) | 115589 |
| 1307 | Breaks window | 113609 |
| 0334 | Brandishes weapon | 105665 |
| 2004 | Suspect is homeless/transient | 93426 |
| 0432 | Intimidation | 83562 |
+-----+-----+
only showing top 20 rows

```

- SHUFFLE HASH Join (~29 sec): Εδώ το Spark αναγκάζεται να κάνει **Shuffle** και τους δύο πίνακες με βάση το κλειδί (MO_code) ώστε οι ίδιες τιμές να βρεθούν στον ίδιο κόμβο. Στη συνέχεια χτίζει ένα Hash Table για τη μικρότερη πλευρά. Ο χρόνος που χάθηκε οφείλεται στο κόστος μεταφοράς δεδομένων (Network I/O) του Shuffle, το οποίο δεν υπήρχε στο Broadcast.

```

--- Testing Strategy: SHUFFLE_HASH ---
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- ShuffledHashJoin [MO_code#1461], [MO_code#153], Inner, BuildRight
  :- HashAggregate(keys=[MO_code#1461], functions=[count(1)], schema specialized)
  :  +- Exchange hashpartitioning(MO_code#1461, 1000), ENSURE_REQUIREMENTS, [plan_id=4915]
  :    +- HashAggregate(keys=[MO_code#1461], functions=[partial_count(1)], schema specialized)
  :      +- Filter (length(MO_code#1461) > 0)
  :        +- Generate explode(split(Mocodes#10, , -1)), false, [MO_code#1461]
  :          +- Union
  :            :- Filter isnotnull(Mocodes#10)
  :              +- FileScan csv [Mocodes#10] Batched: false, DataFilters: [isnotnull(Mocodes#10)], Format: CSV, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/LA_C..., PartitionFilters: [], PushedFilters: [IsNotNull(Mocodes)], ReadSchema: struct<Mocodes:string>
  :                +- Filter isnotnull(Mocodes#77)
  :                  +- FileScan csv [Mocodes#77] Batched: false, DataFilters: [isnotnull(Mocodes#77)], Format: CSV, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/M0_c..., PartitionFilters: [], PushedFilters: [IsNotNull(Mocodes)], ReadSchema: struct<Mocodes:string>
  +- Exchange hashpartitioning(MO_code#153, 1000), ENSURE_REQUIREMENTS, [plan_id=4919]
    +- Project [split(value#151, , 2)[0] AS MO_code#153, split(value#151, , 2)[1] AS MO_desc#154]
      +- Filter ((length(split(value#151, , 2)[0]) > 0) AND isnotnull(split(value#151, , 2)[0]))
        +- FileScan text [value#151] Batched: false, DataFilters: [(length(split(value#151, , 2)[0]) > 0), isnotnull(split(value#151, , 2)[0])], Format: Text, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/M0_c..., PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value:string>

Time taken for SHUFFLE_HASH: 29.0601 seconds
+-----+-----+
| MO_code | MO_desc | count |
+-----+-----+
| 0344 | Removes vict property | 1002900 |
| 1822 | Stranger | 548422 |
| 0416 | Hit-Hit w/ weapon | 404773 |
| 0329 | Vandalized | 377536 |
| 0913 | Victim knew Suspect | 278618 |
| 2000 | Domestic violence | 256188 |
| 1300 | Vehicle involved | 219082 |
| 0400 | Force used | 213165 |
| 1402 | Evidence Booked (any crime) | 177478 |
| 1609 | Smashed | 131229 |
| 1309 | Susp uses vehicle | 122108 |
| 1202 | Victim was aged (60 & over) or blind/physically disabled/unable to care for self | 120238 |
| 0325 | Took merchandise | 120159 |
| 1814 | Susp is/was current/former boyfriend/girlfriend | 118073 |
| 0444 | Pushed | 116763 |
| 1501 | Other MO (see rpt) | 115589 |
| 1307 | Breaks window | 113609 |
| 0334 | Brandishes weapon | 105665 |
| 2004 | Suspect is homeless/transient | 93426 |
| 0432 | Intimidation | 83562 |
+-----+-----+
only showing top 20 rows

```

- SORT MERGE Join (~33 sec - Η πιο αργή): Αυτή είναι η default στρατηγική για μεγάλα datasets, αλλά εδώ ήταν η πιο αργή. Όπως περιγράφεται στη θεωρία, απαιτεί τρία βαριά στάδια: **Shuffle** (μετακίνηση δεδομένων), **Sort** (ταξινόμηση και των δύο

πινάκων στο δίσκο/μνήμη) και **Merge**. Η διαδικασία της ταξινόμησης (Sorting) προσθέτει σημαντικό υπολογιστικό κόστος σε σχέση με το απλό Hash join.

```
--- Testing Strategy: MERGE ---
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Sort [MO_code#1129 ASC NULLS FIRST], false, 0
  +- SortMergeJoin [MO_code#1129], [MO_code#153], Inner
    :- Sort [MO_code#1129 ASC NULLS FIRST], false, 0
      +- HashAggregate(keys=[MO_code#1129], functions=[count(1)], schema specialized)
        :  +- Exchange hashpartitioning(MO_code#1129, 1000), ENSURE_REQUIREMENTS, [plan_id=3787]
        :    +- HashAggregate(keys=[MO_code#1129], functions=[partial_count(1)], schema specialized)
          :      +- Filter ((length(MO_code#1129) > 0))
          :        +- Generate explode(split(Mocodes#10, , -1)), false, [MO_code#1129]
          :          +- Union
          :            :- Filter isnotnull(Mocodes#10)
          :              +- FileScan csv [Mocodes#10] Batched: false, DataFilters: [isnotnull(Mocodes#10)], Format: CSV, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/LA_C..., PartitionFilters: [], PushedFilters: [isNotNull(Mocodes)], ReadSchema: struct<Mocodes:string>
          :                +- Filter isnotnull(Mocodes#77)
          :                  +- FileScan csv [Mocodes#77] Batched: false, DataFilters: [isnotnull(Mocodes#77)], Format: CSV, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/LA_C..., PartitionFilters: [], PushedFilters: [isNotNull(Mocodes)], ReadSchema: struct<Mocodes:string>
    +- Sort [MO_code#153 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(MO_code#153, 1000), ENSURE_REQUIREMENTS, [plan_id=3791]
        +- Project [split(value#151, , 2)[0] AS MO_code#153, split(value#151, , 2)[1] AS MO_desc#154]
          +- Filter ((length(split(value#151, , 2)[0]) > 0) AND isnotnull(split(value#151, , 2)[0]))
            +- FileScan text [value#151] Batched: false, DataFilters: [(length(split(value#151, , 2)[0]) > 0), isnotnull(split(value#151, , 2)[0])], Format: Text, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/MO_c..., PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value:string>

Time taken for MERGE: 33.1475 seconds
+-----+-----+
| MO_code | MO_desc | count |
+-----+-----+
| 0344   | Removes vict property | 1002900|
| 1822   | Stranger | 548422 |
| 0416   | Hit-Hit w/ weapon | 404773 |
| 0329   | Vandalized | 377536 |
| 0913   | Victim knew Suspect | 278618 |
| 2000   | Domestic violence | 256188 |
| 1300   | Vehicle involved | 219082 |
| 0400   | Force used | 213165 |
| 1402   | Evidence Booked (any crime) | 177478 |
| 1609   | Smashed | 131229 |
| 1309   | Susp uses vehicle | 122108 |
| 1202   | Victim was aged (60 & over) or blind/physically disabled/unable to care for self | 120238 |
| 0325   | Took merchandise | 120159 |
| 1814   | Susp is/was current/former boyfriend/girlfriend | 118073 |
| 0444   | Pushed | 116763 |
| 1501   | Other MO (see rpt) | 115589 |
| 1307   | Breaks window | 113609 |
| 0334   | Brandishes weapon | 105665 |
| 1204   | Suspect is homeless/transient | 93426 |
| 0432   | Intimidation | 83562 |
+-----+-----+
only showing top 20 rows
```

- **Cartesian / Shuffle Replicate NL (~29.5 sec)**: Παρόλο που θεωρητικά είναι η πιο αργή (πολυπλοκότητα $O(M^N)$), εδώ λειτουργησε σχετικά γρήγορα μόνο επειδή ο πίνακας MO_codes ήταν εξαιρετικά μικρός. Αν και οι δύο πίνακες ήταν μεγάλοι, αυτή η στρατηγική θα είχε αποτύχει ή θα έπαιρνε ώρες.

```
--- Testing Strategy: SHUFFLE_REPLICATE_NL ---
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- CartesianProduct (MO_code#1627 = MO_code#153)
  :- HashAggregate(keys=[MO_code#1627], functions=[count(1)], schema specialized)
  :  +- Exchange hashpartitioning(MO_code#1627, 1000), ENSURE_REQUIREMENTS, [plan_id=5454]
  :    +- HashAggregate(keys=[MO_code#1627], functions=[partial_count(1)], schema specialized)
  :      +- Filter (length(MO_code#1627) > 0)
  :        +- Generate explode(split(Mocodes#10, , -1)), false, [MO_code#1627]
  :          +- Union
  :            :- Filter isnotnull(Mocodes#10)
  :              +- FileScan csv [Mocodes#10] Batched: false, DataFilters: [isnotnull(Mocodes#10)], Format: CSV, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/LA_C..., PartitionFilters: [], PushedFilters: [isNotNull(Mocodes)], ReadSchema: struct<Mocodes:string>
  :                +- Filter isnotnull(Mocodes#77)
  :                  +- FileScan csv [Mocodes#77] Batched: false, DataFilters: [isnotnull(Mocodes#77)], Format: CSV, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/LA_C..., PartitionFilters: [], PushedFilters: [isNotNull(Mocodes)], ReadSchema: struct<Mocodes:string>
  +- Project [split(value#151, , 2)[0] AS MO_code#153, split(value#151, , 2)[1] AS MO_desc#154]
    +- Filter ((length(split(value#151, , 2)[0]) > 0) AND isnotnull(split(value#151, , 2)[0]))
      +- FileScan text [value#151] Batched: false, DataFilters: [(length(split(value#151, , 2)[0]) > 0), isnotnull(split(value#151, , 2)[0])], Format: Text, Location: InMemoryFileIndex(1 paths)[s3://initial-notebook-data-bucket-dblab-905418150721/project_data/MO_c..., PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value:string>

Time taken for SHUFFLE_REPLICATE_NL: 29.5357 seconds
+-----+-----+
| MO_code | MO_desc | count |
+-----+-----+
| 0344   | Removes vict property | 1002900|
| 1822   | Stranger | 548422 |
| 0416   | Hit-Hit w/ weapon | 404773 |
| 0329   | Vandalized | 377536 |
| 0913   | Victim knew Suspect | 278618 |
| 2000   | Domestic violence | 256188 |
| 1300   | Vehicle involved | 219082 |
| 0400   | Force used | 213165 |
| 1402   | Evidence Booked (any crime) | 177478 |
| 1609   | Smashed | 131229 |
| 1309   | Susp uses vehicle | 122108 |
| 1202   | Victim was aged (60 & over) or blind/physically disabled/unable to care for self | 120238 |
| 0325   | Took merchandise | 120159 |
| 1814   | Susp is/was current/former boyfriend/girlfriend | 118073 |
| 0444   | Pushed | 116763 |
| 1501   | Other MO (see rpt) | 115589 |
| 1307   | Breaks window | 113609 |
| 0334   | Brandishes weapon | 105665 |
| 1204   | Suspect is homeless/transient | 93426 |
| 0432   | Intimidation | 83562 |
+-----+-----+
only showing top 20 rows
```

Ακόμα επιλέχθηκε η χρήση του **RDD API** (Resilient Distributed Datasets) για να επιδειχθεί η ευελιξία του Spark σε μη δομημένους μετασχηματισμούς. Η χρήση RDD για string parsing (στο MO_codes) είναι συχνά πιο διαισθητική (Pythonic way), αλλά χάνει τα optimizations

του Catalyst Optimizer που προσφέρουν τα DataFrames. Ωστόσο, για το συγκεκριμένο task του split/explode, η απόδοση κρίθηκε ικανοποιητική.

```
RDD Execution Time: 31.2831 seconds

--- RDD Results ---
Code: 0344, Count: 1002900, Desc: Removes victim property
Code: 1822, Count: 548422, Desc: Stranger
Code: 0416, Count: 404773, Desc: Hit-Hit w/ weapon
Code: 0329, Count: 377536, Desc: Vandalized
Code: 0913, Count: 278618, Desc: Victim knew Suspect
Code: 2000, Count: 256188, Desc: Domestic violence
Code: 1300, Count: 219082, Desc: Vehicle involved
Code: 0400, Count: 213165, Desc: Force used
Code: 1402, Count: 177470, Desc: Evidence Booked (any crime)
Code: 1609, Count: 131229, Desc: Smashed
Code: 1309, Count: 122108, Desc: Suspect uses vehicle
Code: 1202, Count: 120238, Desc: Victim was aged (60 & over) or blind/physically disabled/unable to care for self
Code: 0325, Count: 120159, Desc: Took merchandise
Code: 1814, Count: 118073, Desc: Suspect is/was current/former boyfriend/girlfriend
Code: 0444, Count: 116763, Desc: Pushed
Code: 1501, Count: 115589, Desc: Other MO (see rpt)
Code: 1307, Count: 113609, Desc: Breaks window
Code: 0334, Count: 105665, Desc: Brandishes weapon
Code: 2004, Count: 93426, Desc: Suspect is homeless/transient
Code: 0432, Count: 83562, Desc: Intimidation
```

Query 4

Για την αλλαγή του config της υλοποίησης κάνουμε επανεκκίνηση τον kernel και αλλάζουμε το config, έτσι ώστε να μην χρειάζεται να έχουμε 3 διαφορετικά κελιά config.

Για την υλοποίηση του Query 4 καθαρίζονται αρχικά τα δεδομένα από λανθασμένες γεωγραφικές εγγραφές και χρησιμοποιείται η βιβλιοθήκη Sedona για τη δημιουργία γεωμετρικών σημείων και τον υπολογισμό αποστάσεων. Στη συνέχεια, εκτελείται cross join των αστυνομικών τμημάτων, ώστε να υπολογιστεί η απόσταση κάθε εγκλήματος από όλα τα τμήματα. Με χρήση window function επιλέγεται το πλησιέστερο τμήμα ανά έγκλημα και τέλος γίνεται ομαδοποίηση ανά division για τον υπολογισμό πλήθους περιστατικών και μέσης απόστασης.

Από την έξοδο του explain για το cross join παρατηρείται ότι ο optimizer εκτελεί **BroadcastExchange** και επιλέγει **BroadcastNestedLoopJoin**, που είναι η πιο αποδοτική επιλογή για CROSS JOIN όταν η μία πλευρά (εν προκειμένω τα divisions) είναι μικρή. Σύμφωνα με αυτή ο μικρός πίνακας γίνεται broadcast σε όλους τους executors και εκτελείται τοπικά join με τα εγκλήματα χωρίς shuffle, ώστε να υπολογιστούν αποδοτικά οι αποστάσεις κάθε εγκλήματος από κάθε τμήμα και να εντοπιστεί το πλησιέστερο. Με αυτόν τον τρόπο αποφεύγει μεγάλο shuffle/costly cartesian χωρίς broadcast.

Όσον αφορά τις εκτελέσεις με διαφορετικά configurations προέκυψε ο ακόλουθος πίνακας:

Configurations	Χρόνος εκτέλεσης (seconds)
1 core, 2 GB memory	50.46
2 cores, 4GB memory	39.55

4 cores, 8GB memory

29.57

Παρατηρείται ότι οι χρόνοι εκτέλεσης μειώνονται όσο αυξάνονται οι διαθέσιμοι υπολογιστικοί πόροι, γεγονός που δείχνει ότι το Query 4 επωφελείται από τον παραλληλισμό. Με 1 core και 2GB μνήμης η εκτέλεση είναι η πιο αργή, καθώς το cross join, οι υπολογισμοί αποστάσεων και οι window συναρτήσεις εκτελούνται με περιορισμένους πόρους. Με 2 cores και 4GB παρατηρείται αισθητή βελτίωση, ενώ με 4 cores και 8GB ο χρόνος μειώνεται περαιτέρω, καθώς μειώνονται τα bottlenecks σε CPU και μνήμη. Παρ' όλα αυτά, η βελτίωση δεν είναι γραμμική, λόγω του κόστους των shuffle και sort στα window και aggregation στάδια.

division	average_distance	#
HOLLYWOOD	0.02	212878
VAN NUYS	0.029	209305
WILSHIRE	0.026	198499
SOUTHWEST	0.022	186976
OLYMPIC	0.017	172251
NORTH HOLLYWOOD	0.026	171399
77TH STREET	0.017	166133
PACIFIC	0.038	158898
CENTRAL	0.01	155274
RAMPART	0.015	150304
SOUTHEAST	0.024	143597
TOPANGA	0.032	139462
WEST VALLEY	0.029	129457
HARBOR	0.035	127073
WEST LOS ANGELES	0.03	121301
FOOTHILL	0.041	120663
HOLLENBECK	0.026	119726
NEWTON	0.016	109339
NORTHEAST	0.039	105837
MISSION	0.035	103198
DEVONSHIRE	0.028	76403

```

Execution time: 50.46 seconds
== Physical Plan ==
AdaptiveSparkPlan (26)
+- Sort (25)
  +- Exchange (24)
    +- HashAggregate (23)
      +- Exchange (22)
        +- HashAggregate (21)
          +- Project (20)
            +- Filter (19)
              +- Window (18)
                +- WindowGroupLimit (17)
                  +- Sort (16)
                    +- Exchange (15)
                      +- WindowGroupLimit (14)
                        +- Sort (13)
                          +- Project (12)
                            +- BroadcastNestedLoopJoin Cross BuildRight (11)
                              :- Union (7)
                                : :- Project (3)
                                :   : +- Filter (2)
                                :   :   +- Scan csv (1)
                                :   : :- Project (6)
                                :   :     +- Filter (5)
                                :   :     +- Scan csv (4)
                              +- BroadcastExchange (10)
                                +- Project (9)
                                  +- Scan csv (8)

```

division	average_distance	#
HOLLYWOOD	0.02	212878
VAN NUYS	0.029	209305
WILSHIRE	0.026	198499
SOUTHWEST	0.022	186976
OLYMPIC	0.017	172251
NORTH HOLLYWOOD	0.026	171399
77TH STREET	0.017	166133
PACIFIC	0.038	158098
CENTRAL	0.01	155274
RAMPART	0.015	150304
SOUTHEAST	0.024	143597
TOPANGA	0.032	139462
WEST VALLEY	0.029	129457
HARBOR	0.035	127073
WEST LOS ANGELES	0.03	121301
FOOTHILL	0.041	120663
HOLLENBECK	0.026	119726
NEWTON	0.016	109339
NORTHEAST	0.039	105837
MISSION	0.035	103198
DEVONSHIRE	0.028	76403

```

Execution time: 39.55 seconds
== Physical Plan ==
AdaptiveSparkPlan (26)
+- Sort (25)
  +- Exchange (24)
    +- HashAggregate (23)
      +- Exchange (22)
        +- HashAggregate (21)
          +- Project (20)
            +- Filter (19)
              +- Window (18)
                +- WindowGroupLimit (17)
                  +- Sort (16)
                    +- Exchange (15)
                      +- WindowGroupLimit (14)
                        +- Sort (13)
                          +- Project (12)
                            +- BroadcastNestedLoopJoin Cross BuildRight (11)
                              :- Union (7)
                              :  :- Project (3)
                              :  :  +- Filter (2)
                              :  :  +- Scan csv (1)
                              :  :- Project (6)
                              :  :  +- Filter (5)
                              :  :  +- Scan csv (4)
                            +- BroadcastExchange (10)
                              +- Project (9)
                                +- Scan csv (8)

```

division	average_distance	#
HOLLYWOOD	0.02	212878
VAN NUYS	0.029	209305
WILSHIRE	0.026	198499
SOUTHWEST	0.022	186976
OLYMPIC	0.017	172251
NORTH HOLLYWOOD	0.026	171399
77TH STREET	0.017	166133
PACIFIC	0.038	158098
CENTRAL	0.01	155274
RAMPART	0.015	150304
SOUTHEAST	0.024	143597
TOPANGA	0.032	139462
WEST VALLEY	0.029	129457
HARBOR	0.035	127073
WEST LOS ANGELES	0.03	121301
FOOTHILL	0.041	120663
HOLLENBECK	0.026	119726
NEWTON	0.016	109339
NORTHEAST	0.039	105837
MISSION	0.035	103198
DEVONSHIRE	0.028	76403

```

Execution time: 29.57 seconds
== Physical Plan ==
AdaptiveSparkPlan (26)
+- Sort (25)
  +- Exchange (24)
    +- HashAggregate (23)
      +- Exchange (22)
        +- HashAggregate (21)
          +- Project (20)
            +- Filter (19)
              +- Window (18)
                +- WindowGroupLimit (17)
                  +- Sort (16)
                    +- Exchange (15)
                      +- WindowGroupLimit (14)
                        +- Sort (13)
                          +- Project (12)
                            +- BroadcastNestedLoopJoin Cross BuildRight (11)
                              :- Union (7)
                                : :- Project (3)
                                : : :- Filter (2)
                                : : : :- Scan csv (1)
                                : : :- Project (6)
                                : : : :- Filter (5)
                                : : : : :- Scan csv (4)
                              +- BroadcastExchange (10)
                                +- Project (9)
                                  +- Scan csv (8)

```

Query 5

Παρατηρήθηκε ότι η διαμόρφωση με λίγους ισχυρούς executors (2 executors x 8GB RAM - "Fat Executors") ήταν αποδοτικότερη από τη διαμόρφωση με πολλούς αδύναμους executors (8 executors x 2GB RAM - "Thin Executors"), όπου ο χρόνος αυξήθηκε αντί να μειωθεί. Κατά την ανάλυση του πλάνου εκτέλεσης (.explain()), παρατηρήθηκε ότι το Spark επέλεξε αυτόματα τη στρατηγική **Broadcast Hash Join**. Αυτό οφείλεται στο γεγονός ότι το DataFrame των εισοδημάτων (Income_Data), μετά την ομαδοποίηση ανά κοινότητα, έχει μικρό μέγεθος.

- **Memory Overhead για Hash Tables:** Στο Broadcast Hash Join, ο μικρός πίνακας αποστέλλεται σε κάθε Executor και χτίζεται στη μνήμη (Execution Memory) ως **Hash Table**, ώστε να γίνονται τα lookups σε χρόνο O(1). Οι "Thin" executors (π.χ. με 1-2GB RAM) θα δυσκολεύονταν να διατηρήσουν αυτό το Hash Table στη μνήμη ταυτόχρονα με τα ενδιάμεσα αποτελέσματα της επεξεργασίας, οδηγώντας σε συχνό

Garbage Collection ή ακόμα και σε Spill στο δίσκο. Με τα 8GB RAM, το Hash Table χωράει άνετα στο Heap.

- **Μείωση Network Traffic (Shuffle):** Σε αντίθεση με το Sort-Merge Join που απαιτεί ακριβό Shuffle (μετακίνηση δεδομένων μεταξύ κόμβων) και για τους δύο πίνακες, το Broadcast Join μετακινεί μόνο τον μικρό πίνακα. Επομένως, δεν χρειαζόμαστε υψηλό παραλληλισμό δικτύου (πολλούς executors), αλλά **ισχυρούς κόμβους με μνήμη** που να μπορούν να εκτελέσουν γρήγορα τα joins στα τοπικά δεδομένα.
- **Αποδοτικότητα Πόρων:** Εφόσον ο πίνακας γίνεται broadcast, το να έχουμε πολλούς μικρούς executors θα σήμαινε ότι αντιγράφουμε τα ίδια δεδομένα (το Hash Table) πολλές φορές (μία για κάθε JVM), σπαταλώντας συνολική μνήμη στο cluster. Με λιγότερους και μεγαλύτερους executors, το overhead της αντιγραφής ελαχιστοποιείται.

Για μνημοβόρες εργασίες όπως η γεωχωρική ανάλυση, το **Scale Up** (Fat Executors) είναι προτιμότερο από το Scale Out (Thin Executors) διότι μεγιστοποιεί τη διαθέσιμη μνήμη ανά task και μειώνει το overhead της επικοινωνίας.

Τα αποτελέσματα ερμηνεύονται ως εξής: Αν ο αριθμός είναι κοντά στο -1: Υπάρχει ισχυρή αρνητική συσχέτιση (όσο ανεβαίνει το εισόδημα, πέφτει το έγκλημα). Αν ο αριθμός είναι κοντά στο 0: Δεν υπάρχει γραμμική συσχέτιση. Συχνά στα άκρα όπως βλέπουμε και στα αποτελέσματα (πολύ πλούσιοι vs πολύ φτωχοί) η συσχέτιση είναι πιο έντονη από ό,τι στο γενικό σύνολο.

Execution Time: 24.8516 seconds
Correlation (All Communities): -0.165119587145925
Correlation (Top 10 & Bottom 10 Income Areas): -0.22162106566579637
--- Top 10 Wealthiest Areas Stats ---
+-----+-----+-----+-----+-----+
COMM Total_Population Total_Crimes_2y Median_Income Crimes_Per_Person_Yearly
+-----+-----+-----+-----+-----+
PACIFIC PALISADES 20952 1362 212115.0 0.03250
PLAYA VISTA 16230 1294 166667.0 0.03986
BRENTWOOD 30334 2145 143000.0 0.03536
MARINA DEL REY 11373 67 137813.0 0.00295
MARINA PENINSULA 4903 472 137813.0 0.04813
PORTER RANCH 35717 1395 130322.0 0.01953
ENCINO 45045 3893 118878.0 0.04321
WESTCHESTER 50760 7592 115943.0 0.07478
PLAYA DEL REY 2958 378 110884.0 0.06389
CENTURY CITY 13600 1376 109704.0 0.05059
+-----+-----+-----+-----+-----+
--- Top 10 Poorest Areas Stats ---
+-----+-----+-----+-----+-----+
COMM Total_Population Total_Crimes_2y Median_Income Crimes_Per_Person_Yearly
+-----+-----+-----+-----+-----+
HARVARD HEIGHTS 15806 1754 41068.0 0.05549
KOREATOWN 47297 5330 42990.5 0.05635
WESTLAKE 56428 7579 43796.0 0.06716
WATTS 42246 4782 46920.5 0.05660
BALDWIN HILLS 30096 3847 49379.0 0.06391
LENNOX 20323 20 50052.0 0.00049
EAST HOLLYWOOD 24474 2908 50822.0 0.05941
PANORAMA CITY 69747 5112 51485.0 0.03665
LEIMERT PARK 15827 2052 52327.0 0.06483
EAST LOS ANGELES 118771 54 52878.5 0.00023
+-----+-----+-----+-----+-----+

```

Execution Time: 28.1783 seconds
Correlation (All Communities): -0.16511958714592517
Correlation (Top 10 & Bottom 10 Income Areas): -0.22162106566579637

```

--- Top 10 Wealthiest Areas Stats ---

	COMM	Total_Population	Total_Crimes_2y	Median_Income	Crimes_Per_Person_Yearly
PACIFIC PALISADES	20952	1362	212115.0	0.03250	
PLAYA VISTA	16230	1294	166667.0	0.03986	
BRENTWOOD	30334	2145	143000.0	0.03536	
MARINA DEL REY	11373	67	137813.0	0.00295	
MARINA PENINSULA	4903	472	137813.0	0.04813	
PORTER RANCH	35717	1395	130322.0	0.01953	
ENCINO	45045	3893	118878.0	0.04321	
WESTCHESTER	50760	7592	115943.0	0.07478	
PLAYA DEL REY	2958	378	110884.0	0.06389	
CENTURY CITY	13600	1376	109704.0	0.05059	

--- Top 10 Poorest Areas Stats ---

	COMM	Total_Population	Total_Crimes_2y	Median_Income	Crimes_Per_Person_Yearly
HARVARD HEIGHTS	15806	1754	41068.0	0.05549	
KOREATOWN	47297	5330	42990.5	0.05635	
WESTLAKE	56428	7579	43796.0	0.06716	
WATTS	42246	4782	46920.5	0.05660	
BALDWIN HILLS	30096	3847	49379.0	0.06391	
LENNOX	20323	20	50052.0	0.00049	
EAST HOLLYWOOD	24474	2908	50822.0	0.05941	
PANORAMA CITY	69747	5112	51485.0	0.03665	
LEIMERT PARK	15827	2052	52327.0	0.06483	
EAST LOS ANGELES	118771	54	52878.5	0.00023	

```

Execution Time: 41.3967 seconds
Correlation (All Communities): -0.16511958714592512
Correlation (Top 10 & Bottom 10 Income Areas): -0.22162106566579637

```

--- Top 10 Wealthiest Areas Stats ---

	COMM	Total_Population	Total_Crimes_2y	Median_Income	Crimes_Per_Person_Yearly
PACIFIC PALISADES	20952	1362	212115.0	0.03250	
PLAYA VISTA	16230	1294	166667.0	0.03986	
BRENTWOOD	30334	2145	143000.0	0.03536	
MARINA DEL REY	11373	67	137813.0	0.00295	
MARINA PENINSULA	4903	472	137813.0	0.04813	
PORTER RANCH	35717	1395	130322.0	0.01953	
ENCINO	45045	3893	118878.0	0.04321	
WESTCHESTER	50760	7592	115943.0	0.07478	
PLAYA DEL REY	2958	378	110884.0	0.06389	
CENTURY CITY	13600	1376	109704.0	0.05059	

--- Top 10 Poorest Areas Stats ---

	COMM	Total_Population	Total_Crimes_2y	Median_Income	Crimes_Per_Person_Yearly
HARVARD HEIGHTS	15806	1754	41068.0	0.05549	
KOREATOWN	47297	5330	42990.5	0.05635	
WESTLAKE	56428	7579	43796.0	0.06716	
WATTS	42246	4782	46920.5	0.05660	
BALDWIN HILLS	30096	3847	49379.0	0.06391	
LENNOX	20323	20	50052.0	0.00049	
EAST HOLLYWOOD	24474	2908	50822.0	0.05941	
PANORAMA CITY	69747	5112	51485.0	0.03665	
LEIMERT PARK	15827	2052	52327.0	0.06483	
EAST LOS ANGELES	118771	54	52878.5	0.00023	