Αναφορά Εξαμηνιαίας Εργασίας

Προχωρημένα Θέματα Βάσεων Δεδομένων

Γεώργιος Μπαρής - 03119866
Ιωάννης Κωνσταντίνος Χατζής - 03119923

Εθνικό Μετσόβιο Πολυτεχνίο
-
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

30 Ιανουαρίου 2024

Περιεχόμενα

1 Εισαγωγή

Στην εξαμηνιαία εργασία του μαθήματος *Προχωρημένα Θέματα Βάσεων Δεδομένων* κληθήκαμε να εργασθούμε στο Spark/Hadoop Framework και να φέρουμε εις πέρας ζητούμενα που αφορούσαν την τροποποίηση και διαχείριση μεγάλου όγκου δεδομένων από το dataset: Los Angeles Crime Data[1][2], με την βοήθεια δευτερευόντων datasets:

- LA Police Stations[3]
- Median Household Income by Zip Code (Los Angeles County) 2015 & Reverse Geocoding[4]

2 Εγκατάσταση & Προ-επεξεργασία Δεδομένων

2.1 Ζητούμενο 1

Η εργασία υλοποιήθηκε με την χρήση δύο εικονικών μηχανών από το `public cloud okeanos-knossos` με τα εξής χαρακτηριστικά:

1. Ubuntu Server LTS 16.04 OS
2. 4 CPUs
3. 8 GB RAM
4. 30 GB disk capacity

Έγινε αναβάθμιση στο λειτουργικό σύστημα των **Virtual Machine** ώστε να τρέχει στην πιο πρόσφατη έκδοση `Ubuntu 22.04.3 LTS`. Η εγκατάσταση του `Apache Spark` έγινε με βάσει του οδηγού που δόθηκε στο εργαστήριο. Το περιβάλλον εργασίας είναι πλήρως κατανεμημένο μεταξύ των δύο κόμβων `"okeanos-master"` και `"okeanos-worker"`. Το σύστημά μας έχει `public ip : 83.212.73.135` και οι `web` εφαρμογές λειτουργούν στις ακόλουθες διευθύνσεις :

- Dfs: http://83.212.73.135:9870
- Yarn: http://83.212.73.135:8088/cluster
- Spark History Server: http://83.212.73.135:18080

Όλοι οι κώδικες μπορούν να βρεθούν στο **Github**[5] αποθετήριο.

Η εκτέλεση κάθε **query** γίνεται με την εντολή:

```
spark-submit <filename.py>
```

[1] https://catalog.data.gov/dataset/crime-data-from-2010-to-2019
[2] https://catalog.data.gov/dataset/crime-data-from-2020-to-present
[3] https://geohub.lacity.org/datasets/lahub::lapd-police-stations/explore
[4] http://www.dblab.ece.ntua.gr/files/classes/data.tar.gz
[5] https://github.com/georgebaris/advanced_db_project.git

2.2 Ζητούμενο 2

Το βασικό **DataFrame** παράγεται στο αρχείο `create_dataframe.py`.

Αρχικά διαβάζουμε το **DataFrame**, τροποποιούμε τον τύπο δεδομένων των πεδίων `Date Rptd, DATE OCC, Vict Age, LAT, LON` στους τύπους δεδομένων που αναφέρονται:

- Date Rptd: date
- DATE OCC: date
- Vict Age: integer
- LAT: double
- LON: double

Επίσης παρατηρήσαμε ότι στο αρχικό dataset υπάρχουν διπλότυπα (duplicates) γραμμών, τα οποία και αφαιρέσαμε εξ αρχής καθώς δεν είναι χρήσιμα και περίσσιος όγκος δεδομένων. Κατά την εκτέλεση του αρχείου βλέπουμε να τυπώνεται ο τύπος δεδομένων κάθε στήλης καθώς και ο συνολικός αριθμός γραμμών.

`Total number of rows: 2181564`

Παρακάτω φαίνονται εκτυπωμένες οι πρώτες γραμμές του `DataFrame`:

DR_NO	Date Rptd	DATE OCC	TIME OCC	AREA	AREA NAME	Rpt Dist No	Part 1-2	Crm Cd	Crm Cd Desc	Mocodes	Vict Age	Vict Sex	Vict Descent	Premis Cd
080611351	2011-06-13	2011-06-12	2130	06	Hollywood	0636	1	310	BURGLARY	0344 1607	22	M	H	502
0817	2020-09-20	2020-09-19	1700	17	Devonshire	1777	1	510	VEHICLE - STOLEN	NULL	0	NULL	NULL	101
100100506	2010-01-05	2010-01-04	1650	01	Central	0162	1	442	SHOPLIFTING - PET...	0344 1402	23	M	B	404
100100508	2010-01-08	2010-01-07	2005	01	Central	0182	1	330	BURGLARY FROM VEH...	0344	46	M	H	101
100100521	2010-01-14	2010-01-14	1445	01	Central	0118	2	624	BATTERY - SIMPLE ...	0400 0429 2000	38	F	B	101
100100564	2010-01-30	2010-01-29	1630	01	Central	0139	1	330	BURGLARY FROM VEH...	0344 1609	25	F	W	101
100100570	2010-01-31	2010-01-31	0130	01	Central	0139	1	440	THEFT PLAIN - PET...	0344	31	M	H	210
100100574	2010-02-01	2010-01-31	1930	01	Central	0145	2	624	BATTERY - SIMPLE ...	0416	40	M	W	207
100100608	2010-02-11	2010-02-10	1900	01	Central	0152	1	220	ATTEMPTED ROBBERY	0416	27	M	H	210
100100616	2010-02-11	2010-02-11	1515	01	Central	0192	1	341	THEFT-GRAND ($950...	1212	0	M	H	102
100100622	2010-02-13	2010-02-13	0055	01	Central	0171	2	624	BATTERY - SIMPLE ...	0416	24	M	O	101
100100654	2010-02-27	2010-02-27	1955	01	Central	0174	2	946	OTHER MISCELLANEO...	NULL	0	M	W	101
100100676	2010-03-07	2010-03-06	1930	01	Central	0105	2	745	VANDALISM - MISDE...	0329	44	M	H	502
100100714	2010-03-15	2010-03-14	2050	01	Central	0162	2	624	BATTERY - SIMPLE ...	0416	16	F	H	406
100100730	2010-03-23	2010-03-20	1215	01	Central	0111	2	647	THROWING OBJECT A...	NULL	26	M	B	122
100100777	2010-04-06	2010-04-05	1940	01	Central	0157	1	230	ASSAULT WITH DEAD...	0416	24	M	H	710
100100801	2010-04-14	2010-04-13	1730	01	Central	0139	2	930	CRIMINAL THREATS ...	0421	27	M	W	729
100100826	2010-04-17	2010-04-17	1300	01	Central	012A	2	745	VANDALISM - MISDE...	0329 1402	0	M	O	101
100100835	2010-04-17	2010-04-17	2100	01	Central	0158	1	230	ASSAULT WITH DEAD...	0400 0416 0417	48	M	H	102
100100848	2010-04-25	2010-04-24	2015	01	Central	0166	1	230	ASSAULT WITH DEAD...	0411	28	F	B	710

Figure 1: Πρώτες γραμμές του Dataframe (1)

Premis Desc	Weapon Used Cd	Weapon Desc	Status	Status Desc	Crm Cd 1	Crm Cd 2	Crm Cd 3	Crm Cd 4	LOCATION	Cross Street	LAT	LON
MULTI-UNIT DWELLI...	NULL	NULL	IC	Invest Cont	310	NULL	NULL	NULL	1800 N CHEROKEE ...	NULL	34.1032	-118.3348
STREET	NULL	NULL	IC	Invest Cont	510	NULL	NULL	NULL	9100 RUBIO ...	NULL	34.2367	-118.4955
DEPARTMENT STORE	NULL	NULL	AA	Adult Arrest	442	NULL	NULL	NULL	700 W 7TH ...	NULL	34.048	-118.2577
STREET	NULL	NULL	IC	Invest Cont	330	NULL	NULL	NULL	PICO ...	GRAND ...	34.0389	-118.2643
STREET	400	STRONG-ARM (HANDS...	IC	Invest Cont	624	NULL	NULL	NULL	900 N BROADWAY	NULL	34.064	-118.2375
STREET	NULL	NULL	IC	Invest Cont	330	NULL	NULL	NULL	800 TRACTION ...	NULL	34.0654	-118.236
RESTAURANT/FAST FOOD	NULL	NULL	IC	Invest Cont	440	NULL	NULL	NULL	600 E 2ND ...	NULL	34.0472	-118.2371
BAR/COCKTAIL/NIGH...	400	STRONG-ARM (HANDS...	AO	Adult Other	624	NULL	NULL	NULL	5TH	LOS ANGELES	34.0461	-118.2476
RESTAURANT/FAST FOOD	400	STRONG-ARM (HANDS...	AA	Adult Arrest	220	NULL	NULL	NULL	6TH ...	FLOWER	34.0502	-118.2574
SIDEWALK	NULL	NULL	AA	Adult Arrest	341	NULL	NULL	NULL	PICO	GRAND	34.0389	-118.2643
STREET	400	STRONG-ARM (HANDS...	IC	Invest Cont	624	NULL	NULL	NULL	800 W OLYMPIC ...	NULL	34.045	-118.264
STREET	NULL	NULL	AA	Adult Arrest	946	NULL	NULL	NULL	W 7TH ...	S SPRING ...	34.0445	-118.2523
MULTI-UNIT DWELLI...	NULL	NULL	IC	Invest Cont	745	NULL	NULL	NULL	1300 N BROADWAY	NULL	34.0695	-118.2324
OTHER STORE	400	STRONG-ARM (HANDS...	IC	Invest Cont	624	NULL	NULL	NULL	600 W 9TH ...	NULL	34.0451	-118.2684
VEHICLE, PASSENGE...	NULL	NULL	IC	Invest Cont	647	NULL	NULL	NULL	CESAR E CHAVEZ	FIGUEROA ...	34.0627	-118.2463
OTHER PREMISE	308	STICK	AA	Adult Arrest	230	NULL	NULL	NULL	500 S SAN PEDRO	NULL	34.0442	-118.2439
SPECIALTY SCHOOL/...	511	VERBAL THREAT	IC	Invest Cont	930	NULL	NULL	NULL	900 E 3RD ...	NULL	34.0459	-118.2351
STREET	NULL	NULL	IC	Invest Cont	745	998	NULL	NULL	1ST	MAIN ...	34.0522	-118.2434
SIDEWALK	400	STRONG-ARM (HANDS...	IC	Invest Cont	230	NULL	NULL	NULL	6TH	CERES ...	34.0394	-118.2405
OTHER PREMISE	207	OTHER KNIFE	IC	Invest Cont	230	NULL	NULL	NULL	600 S SAN PEDRO ...	NULL	34.0423	-118.2452

Figure 2: Πρώτες γραμμές του Dataframe (2)

Field Name	**Type**	**Nullable**
DR_NO	string	true
Date Rptd	date	true
DATE OCC	date	true
TIME OCC	string	true
AREA	string	true
AREA NAME	string	true
Rpt Dist No	string	true
Part 1-2	long	true
Crm Cd	long	true
Crm Cd Desc	string	true
Mocodes	string	true
Vict Age	integer	true
Vict Sex	string	true
Vict Descent	string	true
Premis Cd	long	true
Premis Desc	string	true
Weapon Used Cd	long	true
Weapon Desc	string	true
Status	string	true
Status Desc	string	true
Crm Cd 1	long	true
Crm Cd 2	long	true
Crm Cd 3	long	true
Crm Cd 4	long	true
LOCATION	string	true
Cross Street	string	true
LAT	double	true
LON	double	true

Table 1: Schema of the DataFrame

3 Κύρια Ερωτήματα

Για την υλοποίηση των Queries κληθήκαμε να τα υλοποιήσουμε σε DataFrame, SQL APIs και για το Query 2 και σε RDD API και να συγκρίνουμε τους χρόνους εκτέλεσης για κάθε υλοποίηση.
Για κάθε query για να μετρήσουμε την διάρκεια της εκτέλεσης του χρησιμοποιήσαμε την βιβλιοθήκη `time` της `Python` και μέσω αυτής προκύπτουν τα αποτελέσματα που παρουσιάζονται στους πίνακες για κάθε ερώτημα.

3.1 Ζητούμενο 3

Κληθήκαμε να υλοποιήσουμε το `Query 1` το οποίο ζητά να βρεθούν ανά έτος οι τρεις (3) μήνες με υψηλότερη καταγραφή εγκλημάτων και να παρουσιαστούν τα

αποτελέσματα ανά έτος, με φθίνουσα κατάταξη ως προς τον αριθμό των καταγραφών. Για την υλοποίηση δημιουργούμε δύο derived columns, Year & Month που προκύπτουν από το DATE OCC, μετράμε για κάθε μήνα και χρόνο τις καταγραφές και κρατάμε τους 3 υψηλότερους μήνες. Η εκτέλεση του Query έγινε όπως υποδεικνύεται με 4 Spark executors.

Ο Πίνακας 2 παρουσιάζει τα αποτελέσματα του ζητουμένου με την σειρά που αναφέρεται παραπάνω.

2010-2016				2017-Today			
Year	Month	Crime_Total	Rank	Year	Month	Crime_Total	Rank
2010	1	11538	1	2017	10	20433	1
2010	3	11513	2	2017	7	20193	2
2010	4	10976	3	2017	1	19834	3
2011	1	18137	1	2018	1	6259	1
2011	7	17283	2	2018	8	5815	2
2011	10	17034	3	2018	7	5632	3
2012	1	17944	1	2019	7	19122	1
2012	8	17661	2	2019	8	18979	2
2012	5	17502	3	2019	3	18858	3
2013	1	8691	1	2020	1	5259	1
2013	8	8008	2	2020	2	5132	2
2013	12	8001	3	2020	5	4891	3
2014	5	5296	1	2021	10	19310	1
2014	6	5248	2	2021	7	18663	2
2014	7	4830	3	2021	8	18376	3
2015	3	10200	1	2022	5	20425	1
2015	5	10018	2	2022	10	20280	2
2015	7	9785	3	2022	6	20213	3
2016	12	15347	1	2023	8	19782	1
2016	10	14995	2	2023	7	19718	2
2016	1	14864	3	2023	1	19642	3

Table 2: Crime Data by Year and Month Descending

Ο Πίνακας 3 παρουσιάζει τους χρόνους μεταξύ των εκτελέσεων στα διαφορετικά APIs.

API	**Execution Time (seconds)**
DataDrame	11.3895
SQL	11.4259

Table 3: Χρόνοι εκτέλεσης ανά API - Query 1

Από τους χρόνους δεν φαίνεται ουσιαστική διαφορά στην εκτέλεση. Αυτό οφείλεται ότι για την υλοποίηση και στις δύο περιπτώσεις των διαφορετικών API η διαχείριση των δεδομένων στα πλαίσια των queries είναι βελτιστοποιημένη από το ίδιο το framework.

3.2 Ζητούμενο 4

Η υλοποίηση του Query 2 έγινε με την χρήση 3 διαφορετικών μεθόδων. Κληθήκαμε να ταξινομήσουμε τα τμήματα της ημέρας, ανάλογα με τις καταγραφές εγκλημάτων που έλαβαν χώρα στο δρόμο και να τα ταξινομήσουμε σε φθίνουσα σειρά των καταγραφών του κάθε τμήματος. Για την υλοποίηση χρησιμοποιήσαμε δική μας συνάρτηση (UDF - User Defined Function) που κατηγοριοποιεί το segment από τα πρώτα 2 ψηφία του `"TIME OCC"` . Αρχικά φιλτράρουμε μόνο τα logs που στην στήλη `"Premis Desc"` έχουν την κατηγορία "STREET" και έπειτα προσθέτουμε νέα στήλη με όνομα "Day_Segment" της οποίας οι τιμές προκύπτουν από την προαναφερθείσα συνάρτηση. Τέλος, απλά παρουσιάζουμε τα αποτελέσματα ομαδοποιώντας ανά μέρος της ημέρας και κατά φθίνουσα σειρά.

Οι χρόνοι για την εκτέλεση του Query στα διαφορετικά API φαίνεται στον Πίνακα 4. Όπως και πριν, έτσι και στο ζητούμενο αυτό εκτελέστηκε με 4 Spark executors.

API	**Execution Time (seconds)**
DataDrame	10.2699
SQL	8.6825
RDD	19.3205

Table 4: Χρόνοι εκτέλεσης ανά API - Query 2

Από τους χρόνους παρατηρούμε μια ελαφρά καλύτερη επίδοση της SQL σε σχέση με το DataDrame API. Αυτό που έχει μεγάλη διαφορά $x2$ χρόνο είναι το RDD API το οποίο θεωρούμε ότι οφείλεται στο ότι τα άλλα APIs είναι optimizable και το RDD δεν είναι "schema-aware" που το κάνει να είναι λιγότερο αποδοτικό. Τελευταία αιτία που επηρεάζει ως ένα βαθμό τον χρόνο εκτέλεσης του ερωτήματος είναι η μετατροπή του output σε DataFrame για την επίτευξη πιο ευπαρουσίαστης εικόνας αποτελέσματος.

Παρακάτω στον Πίνακα 5 φαίνονται τα αποτελέσματα του Query 2.

Segment	Count
Night	175928
Evening	136977
Afternoon	109270
Morning	91830

Table 5: Crimes by Day Segment

3.3 Ζητούμενο 5

Στο Ζητούμενο αυτό **(Query 3)** σκοπός ήταν για το έτος 2015, να βρούμε τις 3 περιοχές του Los Angeles με υψηλότερο και 3 με χαμηλότερο εισόδημα και έπειτα να βρούμε την καταγωγή των θυμάτων εγκλημάτων και να τα παρουσιάσουμε αθροιστικά, με βάση τον αριθμό καταγραφών για κάθε καταγωγή, σε φθίνουσα σειρά. Η υλοποίηση έγινε ως εξής:

- Από το κύριο DataFrame κρατάμε μόνο καταγραφές με "DATE OCC"==2015 & "Vict Descent"!=Null.
- Έπειτα κάνουμε join το τροποποιημένο κύριο DataFrame με το δευτερεύον Geocoding DataFrame στις κοινές συντεταγμένες για να αντιστοιχιστούν τα "ZIP codes" των περιοχών.
- Από το επόμενο βοηθητικό DataFrame Income 2015 κάνουμε join με το κύριο στην στήλη "ZIP codes" για να αντιστοιχιστεί το εισόδημα.
- Κρατάμε τα 3 μεγαλύτερα εισοδήματα και "ZIP codes" και αντίστοιχα τα 3 μικρότερα σε μία λίστα.
- Μέσω βοηθητικής συνάρτησης αποκωδικοποιούμε την καταγωγή σε ολόκληρο όνομα και παρουσιάζουμε την λίστα με τις καταγωγές και τον αριθμό καταγραφών θυμάτων σε φθίνουσα σειρά.

Στον πίνακα 6 παρουσιάζεται το αποτέλεσμα του ζητουμένου.

Vict Descent	**Crime Count**
Black	914
Hispanic	859
White	592
Other	269
Asian	84
Unknown	59
Korean	8
Japanese	3
Italian	2
Caucasian	2
Filipino	2

Table 6: Crime Count by Victim Descent in 2015

Για την εκτέλεση κληθήκαμε να συγκρίνουμε τους χρόνους εκτέλεσης για 2, 3 και 4 Spark executors. Τα αποτελέσματα φαίνονται στον πίνακα 7. Οι διαφορές στους χρόνους και η παρατήρηση ότι για λιγότερους executors ο χρόνος είναι καλύτερος έχει ορισμένες πιθανές εξηγήσεις. Αρχικά, το αυξημένο overhead από διαδικασίες που ανατίθενται. Έπειτα τα πολλαπλά joins αποδίδουν μη αποδοτικούς χρόνους όταν κατανέμεται το φορτίο, μιας και είναι αποδοτικότερο κάποια από αυτά να γίνουν σειριακά.

API / # of executors	4	3	2
DataFrame	20.9337 sec	21.7335 sec	16.3127 sec
SQL	18.4258 sec	16.4078 sec	12.4836 sec

Table 7: Χρόνοι εκτέλεσης για διαφορετικό αριθμό executors per API - Query 3

3.4 Ζητούμενο 6

Όσον αφορά το ζητούμενο αυτό μας ανατέθηκε το τελευταίο και πιο εκτενές σε περιπτώσεις **Query (4)**. Για τον υπολογισμό των αποστάσεων χρησιμοποιήσαμε διαφορετική υλοποίηση συνάρτησης από την προτεινόμενη. Επίσης, για την αποφυγή λάθος αποτελεσμάτων και μεγαλύτερων χρόνων εκτέλεσης, αφαιρέσαμε τα logs με συντεταγμένες που ανήκουν στο Null Island[6].

Year	Average Distance in km	Count
2010	2.651	5486
2011	2.793	7232
2012	2.836	6532
2013	2.731	2271
2014	2.683	2320
2015	2.654	3500
2016	2.734	6076
2017	2.724	7786
2018	2.566	2242
2019	2.74	7129
2020	2.49	2248
2021	2.64	9745
2022	2.609	10026
2023	2.556	9017

Table 8: Average Distance and Count by Year - A

[6]Null Island faulty LAT,LON:Ένα σημαντικό μέρος των συντεταγμένων των καταγραφών φέρεται να έχει περασμένες τις ¨μηδενικές συντεταγμένες¨, και προκύπτουν αποτελέσματα χωρίς λογική συνέχεια.

Το πρώτο μέρος του Query και το α΄ ζητούμενο αφορά τον υπολογισμό εγκλημάτων στα οποία καταγράφηκε χρήση (οποιασδήποτε μορφής) πυροβόλου όπλου και η μέση απόσταση από το αστυνομικό τμήμα που ανέλαβε το περιστατικό.Στον πίνακα 8 φαίνονται τα αποτελέσματα του ζητουμένου. Υλοποιήθηκε με την βοήθεια του δευτερεύοντος DataFrame: LAPD police stations στα εξής βήματα:

- Φιλτράρω των κωδικό των πυροβόλων όπλων '$1xx$' από την στήλη του κύριου DataFrame, "Weapon Used Cd".
- Κάνω join τα δύο DataFrames (main - LAPD) στο κοινό πεδίο τους `"main.AREA==lapd.PREC"`.
- Υπολογίζω την απόσταση σε νέα στήλη και τέλος ανα χρόνο κάνω sort για κάθε χρονιά τις καταγραφές.

Division	Average Distance in km	Count
SOUTHWEST	2.606	71344
77TH STREET	2.644	67523
CENTRAL	1.006	63052
RAMPART	1.531	55229
SOUTHEAST	2.087	45910
HOLLYWOOD	1.428	44560
NEWTON	2.047	39366
HOLLENBECK	2.592	38609
HARBOR	3.939	38016
OLYMPIC	1.749	30840
NORTHEAST	3.980	27356
WILSHIRE	2.412	26850
MISSION	4.710	26797
PACIFIC	3.895	25889
VAN NUYS	2.138	25045
WEST VALLEY	3.375	24598
NORTH HOLLYWOOD	2.536	23426
TOPANGA	3.512	19910
FOOTHILL	4.240	19710
WEST LOS ANGELES	3.648	19154
DEVONSHIRE	3.987	18250

Table 9: Average Distance and Count by Division - A

Στο β΄ ζητούμενο του μέρους καλούμαστε να υπολογίσουμε τις καταγραφές εγκλημάτων με χρήση όπλου (οποιοσδήποτε κωδικός διάφορος του Null) και να τα παρουσιάσουμε ανά Αστυνομικό Τμήμα, στα οποία ανατέθηκαν τα συμβάντα. Το αποτέλεσμα φαίνεται στον πίνακα 9.

Τα βήματα της υλοποίησης:

- Φιλτράρισμα των non-Null τιμών του πεδίου "Weapon Used Cd".
- Join Main & LAPD DataFrame ώστε `"main.AREA==lapd.PREC"`.
- Υπολογισμός απόστασης και παρουσίαση καταγραφών ανά Αστυνομικό τμήμα.

Year	Average Distance in km	Count
2010	2.28	5486
2011	2.462	7232
2012	2.506	6532
2013	2.416	2271
2014	2.16	2320
2015	2.462	3500
2016	2.415	6076
2017	2.392	7786
2018	2.35	2242
2019	2.43	7129
2020	2.269	2248
2021	2.353	9745
2022	2.313	10026
2023	2.275	9017

Table 10: Average Distance and Count by Year - B

Στο δεύτερο μέρος κληθήκαμε να υλοποιήσουμε τα ίδια ερωτήματα αλλά με την **διαφορά** ότι τα αστυνομικά τμήματα που θα λαμβάναμε στο τέλος για τον υπολογισμό της απόστασης θα ήταν αυτά που έχουν την **μικρότερη απόσταση από τον τόπο της καταγραφής** και όχι το τμήμα που ανέλαβε την υπόθεση.

Division	Average Distance in km	Count
SOUTHWEST	2.084	67692
HOLLYWOOD	1.863	59232
CENTRAL	0.852	57332
RAMPART	1.331	54937
77TH STREET	1.672	54643
WILSHIRE	2.516	46857
OLYMPIC	1.691	44018
SOUTHEAST	2.281	42670
HOLLENBECK	2.576	39151
VAN NUYS	2.796	37856
HARBOR	3.683	36912
NEWTON	1.613	29835
PACIFIC	3.869	24523
WEST VALLEY	2.811	24501
NORTH HOLLYWOOD	2.602	24077
FOOTHILL	3.977	23845
TOPANGA	3.053	20287
NORTHEAST	3.757	20214
MISSION	3.787	17693
WEST LOS ANGELES	2.697	15700
DEVONSHIRE	2.844	9459

Table 11: Average Distance and Count by Division - B

Στο α΄ ζητούμενο η υλοποίηση είναι παρόμοια με την υλοποίηση α΄ του πρώτου μέρους. Για να βρούμε όμως το πλησιέστερο Αστυνομικό Τμήμα χρησιμοποιούμε `cross join` το οποίο είναι πολύ πιο κοστοβόρο και χρονοβόρο, αλλά είναι ο πιο απλός τρόπος για να βρεθεί η ελάχιστη απόσταση μεταξύ κάθε καταγραφής εγκλήματος και αστυνομικού τμήματος. Από τις αποστάσεις κρατάμε το log με την ελάχιστη τιμή και συνεχίζουμε την παρουσίαση του ζητήματος όπως και στο πρώτο μέρος. Αντίστοιχα πράττουμε και στο β΄ ερώτημα του δευτέρου μέρους με χρήση cross join και η υπόλοιπη διαδικασία παραμένει ίδια.Τα αποτελέσματα του δεύτερου μέρους για το α΄ και β΄ ερώτημα, παρουσιάζονται στους πίνακες 10 και 11 αντίστοιχα.

Query	Execution Time (seconds)
DF_1a	12.7367
SQL_1a	11.2110
DF_1b	13.2750
SQL_1b	14.4775
DF_2a	25.0340
SQL_2a	17.1889
DF_2b	61.8993
SQL_2b	63.0198

Table 12: Χρόνοι εκτέλεσης για τις περιπτώσεις του Query 4

Οι συγκρίσεις μεταξύ των χρόνων εκτέλεσης των Queries και των παραλλαγών τους στα διαφορετικά APIs (4 executors) φαίνονται στον πίνακα 12. Για τα ερωτήματα του πρώτου μέρους παρατηρούμε παρόμοιους χρόνους για τα δύο APIs , ενώ για το δεύτερο μέρος το α΄ ερώτημα μπορούμε να δούμε ότι υπάρχει μια υπεροχή της SQL όσον αφορά το efficiency που χειρίζεται το cross join και αποδίδει μικρότερο χρόνο. Για το τελευταίο ερώτημα είναι σαφής η μεγάλη διαφορά στον χρόνο εκτέλεσης σε σχέση με τα υπόλοιπα (και για τα δύο APIs) αφού χειρίζεται cross join και υπολογισμό τιμή για πολύ μεγαλύτερο όγκο δεδομένων.

3.5 Ζητούμενο 7

Στο τελικό ζητούμενο ζητήθηκε για τα "joins" των `Query 3 & 4`, να χρησιμοποιηθούν οι μέθοδοι `hint() & explain()` των DataFrame/SQL APIs ώστε να εκτελεστούν με διαφορετικό τρόπο και να πάρουμε πλάνο μέσω του Spark History UI για τις αποδοτικότερες στρατηγικές join.

Παρακάτω παρουσιάζονται στον Πίνακα 13 ενδεικτικά δεδομένα από το Spark History UI . Εκεί τρέχουμε τις εντολές `hint() & explain()` για να δούμε την απόδοση του κάθε query όσον αφορά τον χρόνο εκτέλεσης, της κατανομής δεδομένων και του shuffle μεταξύ των workers.
Οι εντολές χρησιμοποιούνται ως εξής:

- Στο `join:` `df_joined = df1.join(df2.hint("JOIN TYPE"), "condition")`
- Join types: `broadcast, merge, shuffle_hash, shuffle_hash_nl`
- Για την παρουσίαση του πλάνου (join strategy): `df_joined.explain()`

Από τα δεδομένα και το text plan που πήραμε για τα queries μέσω της εντολής `.explain()` (χρησιμοποιείται ως) στο μέρος του join που χρησιμοποιούμε, εξάγουμε τα παρακάτω συμπεράσματα:

- Για τα απλά joins παρατηρούμε ότι η πιο αποδοτική στρατηγική είναι το "broadcast". Για μεγάλα DataFrames η επίδοση μειώνεται καθώς δεν μπορεί να κατανεμηθεί όμοια το φορτίο στους executors και πολλές εκτελέσεις

Table 13: Spark History UI - Join Types Comparison

Join Type	**Ex. Time**	**Data Shuffle**	**CPU Usage/load**
Broadcast	35s	Μικρό data shuffle	Όμοια κατ.
Merge	38s	Μεγάλο data shuffle	Σχ. όμοια καταν
Shuffle_hash	35s	Μεγάλο data shuffle	Σχ. ανόμοια κατ.
shuffle_replicate_nl	>1m	Πολύ Μεγάλο shuffle	Πολύ ανόμοια κατ.

τείνουν να ¨κολλάνε' σε έναν worker με τους υπόλοιπους να περιμένουν απάντηση από αυτόν.

- Όσον αφορά την χρήση "cross-join" μπορούμε να συμπεράνουμε ότι είναι μία αρκετά κοστοβόρα διαδικασία και δεν συνίσταται για συνδυασμούς μεγάλων όγκων δεδομένων. Στην περίπτωση που χρησιμοποιήθηκε τα δεδομένα έχουν ήδη φιλτραριστεί και έχουν περιορισθεί σε $100 - 300$ χιλιάδες γραμμές το πολύ. Παρόλα αυτά η χρήση του broadcast join φαίνεται να είναι η πιο αποδοτική σε κάθε περίπτωση, που θέλουμε να αντλήσουμε πληροφορίες από μικρότερο DataFrame.

4 Επίλογος

Σε αυτή την εργασία, εξερευνήσαμε ενδελελεχώς μεγάλο μέρος της διαχείρισης μεγάλλων όγκων δεδομένων, της μετατροπής και της παρουσίασής τους με την βοήθεια του Apache Spark, παρουσιάζοντας μια λεπτομερή ανάλυση μέσω διαφόρων μεθοδολογιών. Η συνολική μελέτη και ενασχόληση παρά ορισμένες τεχνικές δυσκολίες που αντιμετωπίσθηκαν κατά την πορεία, δείχνει την χρησιμότητα, τις δυνατότητες και την προοπτική του Framework, κάνοντας πολύ πιο κατανοητό και ενδιαφέρον το συνολικό concept του project.

5 Παράρτημα

5.1 Παράρτημα Α - Κώδικας Βοηθητικών συναρτήσεων

Στο παράρτημα αυτό παρατίθεται κώδικας και επεξήγηση του σε σημεία που θεωρούμε οτι χρήζουν περαιτέρω ανάλυσης.

```python
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, FloatType
from math import radians, sin, cos, sqrt, atan2

descent_mapping = {
    'A': 'Asian', 'B': 'Black', 'C': 'Caucasian', 'D': 'Indian',
    'F': 'Filipino', 'G': 'German', 'H': 'Hispanic', 'I': 'Italian',
    'J': 'Japanese', 'K': 'Korean', 'L': 'Laotian', 'O': 'Other',
    'P': 'Pacific Islander', 'S': 'Samoan', 'U': 'Hawaiian',
    'V': 'Vietnamese', 'W': 'White', 'X': 'Unknown',
    'Z': 'Asian Indian', '-': 'Not Specified', None: 'Unknown'
}

def map_descent(code):
    return descent_mapping.get(code, 'Unknown')

def get_distance(lat1, lon1, lat2, lon2):
    lat1, lon1, lat2, lon2 = map(radians, [float(lat1), float(lon1), float(lat2), float(lon2)])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    radius = 6371.0
    return radius * c

def get_day_segment(time_str):
    try:
        hour = int(time_str['TIME OCC'][:2])

        # Classify into day segments
        if 5 <= hour < 12:
            return 'Morning'
        elif 12 <= hour < 17:
            return 'Afternoon'
        elif 17 <= hour < 21:
            return 'Evening'
        else:
            return 'Night'
    except ValueError:
        return 'Unknown'

# Registering UDFs
map_descent_udf = udf(map_descent, StringType())
get_distance_udf = udf(get_distance, FloatType())
get_day_segment_udf = udf(get_day_segment, StringType())
```

Listing 1: UDF registration script

Όσον αφορά το `udfs.py` script που φαίνεται στο παραπάνω απόκομμα 1, παρουσιάζονται συγκεντρωτικά οι `User Defined Function` που χρησιμοποιήσαμε και κάναμε register στο σύστημα για τον υπολογισμό των αποστάσεων (Ζητούμενο 6 - 3.4), ο διαχωρισμός του μέρους της μέρας (Ζητούμενο 4 - 3.2) και η αντιστοίχηση του χαρακτήρα προέλευσης με το κανονικό όνομα (Ζητούμενο 5 - 3.3).

Συγκεκριμένα για την συνάρτηση υπολογισμού της απόστασης `get_distance` χρησιμοποιήσαμε διαφορετική υλοποίηση από την προτεινόμενη (geopy library). Μετατρέπουμε τις συντεταγμένες σε radians ,υπολογίζουμε τις διαφορές των συντεταγμένων και χρησιμοποιούμε τον τύπο haversine για τον υπολογισμό της γωνιακής απόστασης **c** που πολλαπλασιάζουμε με την ακτίνα της Γης σε χιλιόμετρα.

5.2 Παράρτημα Β - Κώδικας Ερωτημάτων

```
spark.sql("""
    -- Filter for 2015 crimes and select required columns
    WITH Crimes2015 AS (
        SELECT DR_NO, `DATE OCC`, `AREA NAME`, `Vict Descent`,
    LOCATION, LAT, LON
        FROM df_main
        WHERE YEAR(`DATE OCC`) = 2015 AND `Vict Descent` IS NOT
    NULL
    ),
    -- Join with geodf to get ZIP codes
    CrimeWithZip AS (
        SELECT c.*, g.ZIPcode
        FROM Crimes2015 c
        INNER JOIN geodf g ON c.LAT = g.LAT AND c.LON = g.LON
    ),
    -- Join with income data
    CrimeWithIncome AS (
    SELECT cwz.*, i.`Zip Code`,
           CAST(REGEXP_REPLACE(i.`Estimated Median Income`, '[\\$
    ,]', '') AS DOUBLE) AS IncomeNumeric
    FROM CrimeWithZip cwz
    INNER JOIN incdf15 i ON cwz.ZIPcode = i.`Zip Code`
    ),
    -- Get distinct ZIP codes based on income
    DistinctZipIncome AS (
        SELECT DISTINCT `Zip Code`, IncomeNumeric
        FROM CrimeWithIncome
    ),
    -- Get top 3 and bottom 3 ZIP codes
    TopBottomZip AS (
        (SELECT `Zip Code` FROM DistinctZipIncome ORDER BY
    IncomeNumeric DESC LIMIT 3)
        UNION ALL
        (SELECT `Zip Code` FROM DistinctZipIncome ORDER BY
    IncomeNumeric ASC LIMIT 3)
    ),
    -- Filter crimes for these ZIP codes
    FilteredCrimes AS (
```

```
        SELECT *
        FROM CrimeWithIncome
        WHERE `Zip Code` IN (SELECT `Zip Code` FROM TopBottomZip)
    )
    SELECT
        COALESCE(
    CASE `Vict Descent`
        WHEN 'A' THEN 'Asian'
        WHEN 'B' THEN 'Black'
        WHEN 'C' THEN 'Caucasian'
        WHEN 'D' THEN 'Indian'
        WHEN 'F' THEN 'Filipino'
        WHEN 'G' THEN 'German'
        WHEN 'H' THEN 'Hispanic'
        WHEN 'I' THEN 'Italian'
        WHEN 'J' THEN 'Japanese'
        WHEN 'K' THEN 'Korean'
        WHEN 'L' THEN 'Laotian'
        WHEN 'O' THEN 'Other'
        WHEN 'P' THEN 'Pacific Islander'
        WHEN 'S' THEN 'Samoan'
        WHEN 'U' THEN 'Hawaiian'
        WHEN 'V' THEN 'Vietnamese'
        WHEN 'W' THEN 'White'
        WHEN 'X' THEN 'Unknown'
        WHEN 'Z' THEN 'Asian Indian'
        WHEN '-' THEN 'Not Specified'
        ELSE 'Unknown'
    END,
    'Unknown'
) AS `Vict Descent Name`,
        COUNT(*) AS `Crime Count`
    FROM FilteredCrimes
    GROUP BY `Vict Descent Name`
    ORDER BY `Crime Count` DESC
""").show()
```

Listing 2: Query 3 - SQL

Για το απόκομμα 2 που περιλαμβάνει την υλοποίηση του Query 3 (SQL) 3.3 παρατίθεται η επεξήγηση σε βήματα:

- Αρχικά φιλτράρουμε τις καταγραφές μόνο από το 2015 με έγκυρες συντεταγμένες
- Από αυτό κάνουμε join με τις συντεταγμένες του geocoding για να κρατήσουμε το Zip Code.
- Κάνουμε join με το Income 2015 στο ZIP Code και μετατρέπουμε την στήλη 'Estimated Median Income' σε αριθμό χωρίς το μέρος του νομίσματος.
- Κρατάμε μόνο distinct Zip codes για να πάρουμε τα σωστά αποτελέσματα ανάλογα με το εισόδημα.
- Επιλέγουμε τα 3 μικρότερα και 3 μεγαλύτερα εισοδήματα.

- Στο Temporary View 'FilteredCrime' κρατάμε καταγραφές με Zip Code που ταιριάζει με τα επιλεγμένα 6.
- Στο τέλος κάνουμε μετάφραση των ονομάτων και μετράμε τις καταγραφές για κάθε περίπτωση, κάνουμε Group By βάσει της καταγωγής και τα παρουσιάζουμε σε φθίνουσα σειρά.

```
spark.sql("""
    WITH FilteredData AS (
        SELECT
            DR_NO, LAT, LON, `DATE OCC`, AREA
        FROM df_main
        WHERE CAST(`Weapon Used Cd` / 100 AS INT) = 1
    ),
    CrossJoined AS (
        SELECT
            f.DR_NO, f.LAT, f.LON, f.`DATE OCC`, f.AREA,
            l.Y, l.X, l.DIVISION
        FROM FilteredData f
        CROSS JOIN lapd l
    ),
    Distances AS (
        SELECT
            *,
            get_distance(LAT, LON, Y, X) AS DISTANCE,
            ROW_NUMBER() OVER (PARTITION BY DR_NO ORDER BY
    get_distance(LAT, LON, Y, X)) AS rank
        FROM CrossJoined
    ),
    ClosestPrecincts AS (
        SELECT
            DR_NO, `DATE OCC`, AREA, DIVISION, DISTANCE
        FROM Distances
        WHERE rank = 1
    )
    SELECT
        YEAR(`DATE OCC`) AS Year,
        ROUND(MEAN(DISTANCE), 3) AS `average distance in km`,
        COUNT(*) AS Count
    FROM ClosestPrecincts
    GROUP BY YEAR(`DATE OCC`)
    ORDER BY YEAR(`DATE OCC`)
""").show()
```

Listing 3: Query 4 2a - SQL

Το απόκομμα 3 αποτελεί την υλοποίηση του ερωτήματος α΄ του δεύτερου μέρους όσον αφορά το Query 4 3.4. Θεωρούμε την υλοποίηση άξια αναφοράς καθώς χρησιμοποιούμε το "cross join". Παρακάτω τα βήματα:

- Αρχικά φιλτράρουμε από το DataFrame μόνο τις χρήσιμες στήλες, καθώς οι μετατροπές που θα κάνουμε παρακάτω θα είναι κοστοβόρες και χρονοβόρες, και κρατάμε μόνο τις καταγραφές με κωδικό όπλου που αρχίζει με 1.
- Στη συνέχεια εκτελούμε cross join των καταγραφών που έχουμε με τις γεωγραφικές συντεταγμένες του DataFrame LAPD.

- Στο επόμενο βήμα χρησιμοποιούμε την δική μας συνάρτηση `get_distance` για τον υπολογισμό κάθε απόστασης.
- Με το `ROW_NUMBER()` αποδίδουμε ένα "rank" για κάθε τμήμα και κάθε για κάθε έγκλημα ¨σορταρισμένο' βάσει της απόστασης. Έτσι βρίσκουμε το πλησιέστερο A.T. στο έγκλημα.
- Από αυτά κρατάμε μόνο όσες καταγραφές έχουν "Rank==1", δηλαδή είναι οι καταγραφές με την απόσταση του πλησιέστερου τμήματος.
- Τέλος, παρουσιάζω τα εγκλήματα ανά έτος και τον αριθμό των καταγραφών, έχοντας βρει την μέση απόσταση για κάθε έγκλημα της συγκεκριμένης χρονιάς.

```
    df_main = df_main.filter((col("LAT") != 0.0) & (col("LON") !=
    0.0))
# Select relevant columns and perform a cross join
df_main_firearms = df_main.filter((col("Weapon Used Cd") / 100).
    cast("int") == 1).select("DR_NO", "LAT", "LON", "DATE OCC", "
    AREA")
cross_joined = df_main_firearms.crossJoin(lapd)

with_distances = cross_joined.withColumn("DISTANCE", get_distance(
    col("LAT"), col("LON"), col("Y"), col("X")))

windowSpec = Window.partitionBy(cross_joined["DR_NO"]).orderBy("
    DISTANCE")

ranked = with_distances.withColumn("rank", row_number().over(
    windowSpec))

# Filter to keep only the closest precinct (rank = 1) for each row
    in the main DataFrame
closest_precincts = ranked.filter(col("rank") == 1).drop("rank")

# Now, closest_precincts DataFrame has each row from df_main along
    with the details of its closest precinct
q4_2a = closest_precincts.groupBy(year(closest_precincts["DATE OCC"
    ]).alias("Year")) \
    .agg(round(avg("DISTANCE"), 3).alias("average distance in km"),
     count("*").alias("Count")) \
    .orderBy("Year", ascending=True)

q4_2a.show(q4_2a.count())
```

Listing 4: Query 4 2a - DF

Για το τελευταίο απόκομμα (4), που είναι υλοποίηση σε Python - Spark DataFrame API, σχολιάζουμε τις ουσιαστικές αλλαγές σε σχέση με την προηγούμενη υλοποίηση σε SQL API. Στο κομμάτι αυτό, συνεπώς, θα σχολιάσουμε το φιλτράρισμα των κοντινότερων αποστάσεων.

Μετά το cross join χρησιμποιούμε το `Window.partitionBy()` που χωρίζει το DataFrame view στο οποίο εργαζόμαστε, σε ομάδες ανάλογα με το unique

log identifier "DR_NO" που αντιστοιχεί σε κάθε καταγεγραμμένο έγκλημα. Έπειτα, προσθέτουμε την στήλη "rank" και όπως πριν κρατάμε τις καταγραφές με 'βαθμό' ίσο με ένα, δηλαδή την μικρότερη απόσταση. Τέλος, ομαδοποιούμε και παρουσιάζουμε όπως και στην προηγούμενη υλοποίηση.

Για τις υπόλοιπες υλοποιήσεις δεν παρατίθεται κώδικας ή αποκόμματα εντός της αναφοράς, καθώς την καθιστά υπερβολικά εκτενή. Περιγράψαμε αυτές που θεωρήσαμε ότι ήταν σημαντικότερες ή άξιες αναφοράς. Για αναλυτικότερη περιήγηση στις υλοποιήσεις, ο κώδικας υπάρχει στο Github repository[7].

[7]https://github.com/georgebaris/advanced_db_project