Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Προχωρημένα Θέματα Βάσεων Δεδομένων

Κυριακή Καρατζούνη el20634

Βικέντιος Βιτάλης el18803

Ομάδα 17

Αναφορά

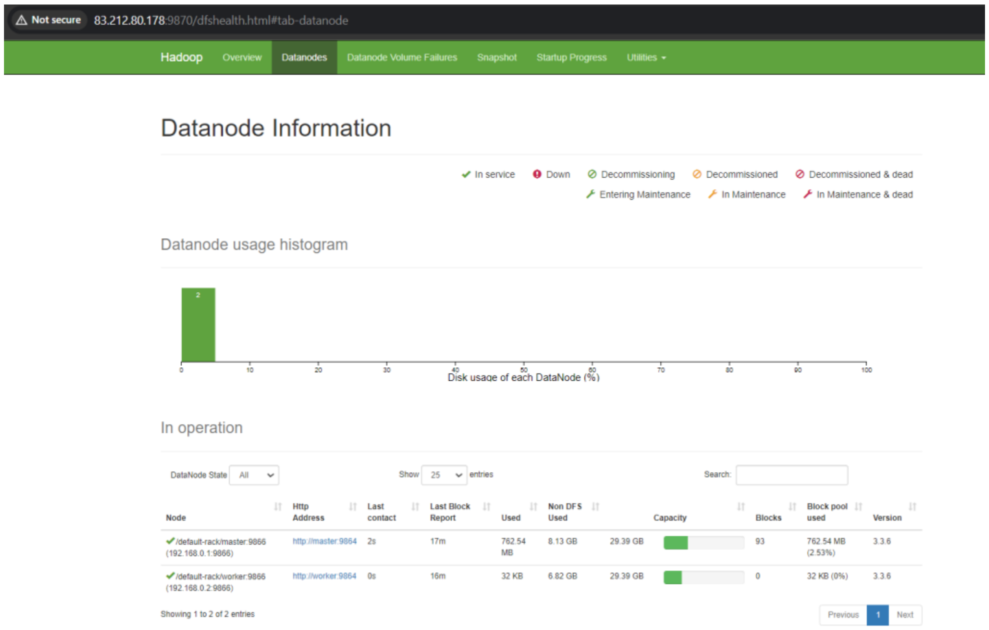
Github repository:
https://github.com/VikentiosVitalis/advanced_topics_in_database_systems

Ζητούμενο 1. Αρχικά δημιουργήσαμε στην υπηρεσία Okeanos Knossos ένα δίκτυο (cluster) 2 κόμβων σύμφωνα με τον εργαστηριακό οδηγό ‘’Advanced Topics in Database Systems: Lab guide.ipynb’’ κι εγκαταστήσαμε το λογισμικό και στους δύο κόμβους. Μέσω του WinSCP συνδεθήκαμε στον master node κι από την επιφάνεια εργασίας των Windows μεταφορτώσαμε τα σύνολα δεδομένων. Στο αρχείο files/documents/setup.pdf του Github repository, υπάρχει ο αναλυτικός οδηγός εγκατάστασης. Παρατίθονται τα UIs από τις υπηρεσίες HDFS, YARN και Spark History Server αντίστοιχα:

http://83.212.80.178:9870/dfshealth.html#tab-datanode

http://83.212.80.178:8088/cluster/nodes

http://83.212.80.178:18080/



Not secure 83.212.80.178:9870/dfshealth.html#tab-datanode

Datanode Information

✓ In service ● Down ⊘ Decommissioning ⊘ Decommissioned ⊘ Decommissioned & dead
Entering Maintenance In Maintenance In Maintenance & dead

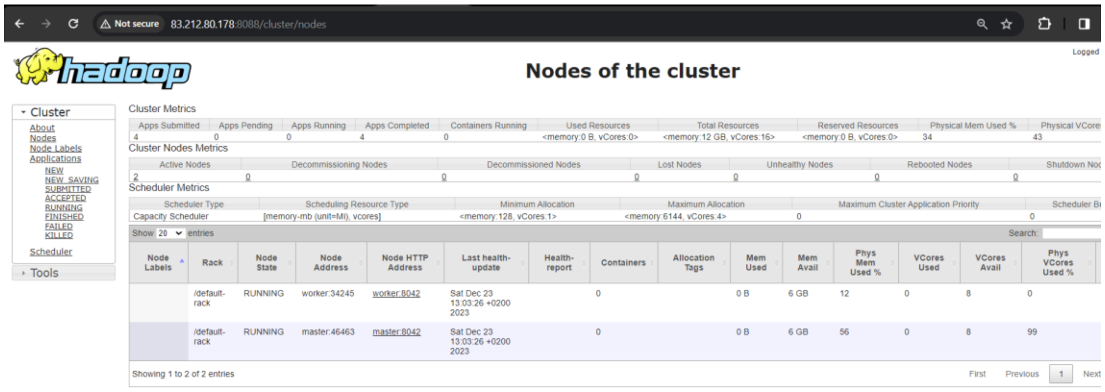
Datanode usage histogram

In operation

DataNode State: All   Show 25 entries   Search:

Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Version
✓ /default-rack/master:9866 (192.168.0.1:9866)	http://master:9864	2s	17m	762.54 MB	8.13 GB	29.39 GB 🟩	93	762.54 MB (2.53%)	3.3.6
✓ /default-rack/worker:9866 (192.168.0.2:9866)	http://worker:9864	0s	16m	32 KB	6.82 GB	29.39 GB 🟩	0	32 KB (0%)	3.3.6

Showing 1 to 2 of 2 entries   Previous 1 Next



Not secure 83.212.80.178:8088/cluster/nodes

Logged

Nodes of the cluster

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources	Reserved Resources	Physical Mem Used %	Physical VCore
4	0	0	4	0	<memory:0 B, vCores:0>	<memory:12 GB, vCores:16>	<memory:0 B, vCores:0>	34	43

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Noc
2	0	0	0	0	0	0

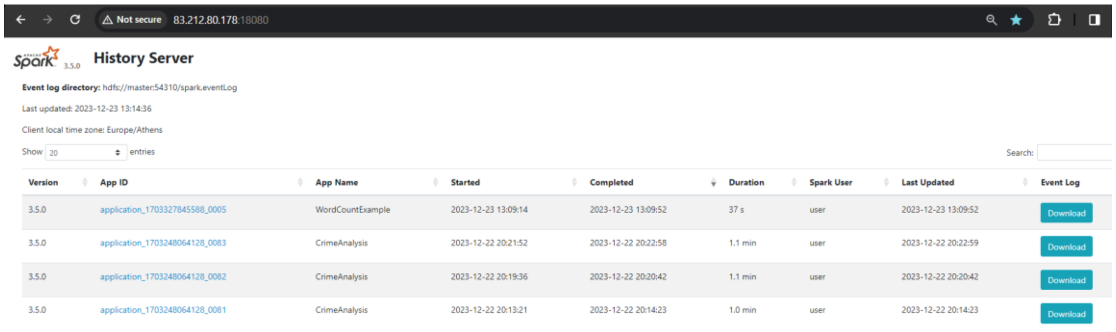
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority	Scheduler B
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:128, vCores:1>	<memory:6144, vCores:4>	0	0

Show 20 entries   Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Allocation Tags	Mem Used	Mem Avail	Phys Mem Used %	VCores Used	VCores Avail	Phys VCores Used %
	/default-rack	RUNNING	worker:34245	worker:8042	Sat Dec 23 13:03:26 +0200 2023		0		0 B	6 GB	12	0	8	0
	/default-rack	RUNNING	master:46463	master:8042	Sat Dec 23 13:03:26 +0200 2023		0		0 B	6 GB	56	0	8	99

Showing 1 to 2 of 2 entries   First Previous 1 Next



Not secure 83.212.80.178:18080

History Server

Spark 3.5.0

Event log directory: hdfs://master:54310/spark.eventLog

Last updated: 2023-12-23 13:14:36

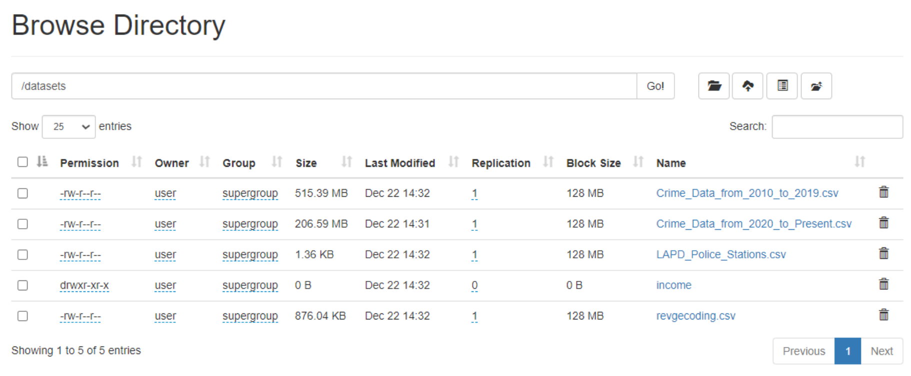
Client local time zone: Europe/Athens

Show 20 entries   Search:

Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.5.0	application_1703327845588_0005	WordCountExample	2023-12-23 13:09:14	2023-12-23 13:09:52	37 s	user	2023-12-23 13:09:52	Download
3.5.0	application_1703248064128_0083	CrimeAnalysis	2023-12-22 20:21:52	2023-12-22 20:22:58	1.1 min	user	2023-12-22 20:22:59	Download
3.5.0	application_1703248064128_0082	CrimeAnalysis	2023-12-22 20:19:36	2023-12-22 20:20:42	1.1 min	user	2023-12-22 20:20:42	Download
3.5.0	application_1703248064128_0081	CrimeAnalysis	2023-12-22 20:13:21	2023-12-22 20:14:23	1.0 min	user	2023-12-22 20:14:23	Download

Με τις παρακάτω εντολές μεταφορτώνουμε τα αρχεία μας στην Hadoop Distributed File System (HDFS) υπηρεσία. Μεταβένοντας στα Utilities > Browse the file system είναι ορατά τα μεταφορτωμένα αρχεία. Ακολουθούν οι εντολές μεταφόρτωσης:

- hadoop fs -mkdir hdfs://master:54310/datasets
- hadoop fs -mkdir hdfs://master:54310/datasets/income
- hadoop fs -put datasets/Crime_Data_from_2010_to_2019.csv hdfs://master:54310/datasets/.
- hadoop fs -put datasets/Crime_Data_from_2020_to_Present.csv hdfs://master:54310/datasets/.
- hadoop fs -put datasets/revgecoding.csv hdfs://master:54310/datasets/.
- hadoop fs -put datasets/LAPD_Police_Stations.csv hdfs://master:54310/datasets/.
- hadoop fs -put datasets/income/LA_income_2015.csv hdfs://master:54310/datasets/income/.
- hadoop fs -put datasets/income/LA_income_2017.csv hdfs://master:54310/datasets/income/.
- hadoop fs -put datasets/income/LA_income_2019.csv hdfs://master:54310/datasets/income/.
- hadoop fs -put datasets/income/LA_income_2021.csv hdfs://master:54310/datasets/income/.



Browse Directory

/datasets

Show 25 entries

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	user	supergroup	515.39 MB	Dec 22 14:32	1	128 MB	Crime_Data_from_2010_to_2019.csv
-rw-r--r--	user	supergroup	206.59 MB	Dec 22 14:31	1	128 MB	Crime_Data_from_2020_to_Present.csv
-rw-r--r--	user	supergroup	1.36 KB	Dec 22 14:32	1	128 MB	LAPD_Police_Stations.csv
drwxr-xr-x	user	supergroup	0 B	Dec 22 14:32	0	0 B	income
-rw-r--r--	user	supergroup	876.04 KB	Dec 22 14:32	1	128 MB	revgecoding.csv

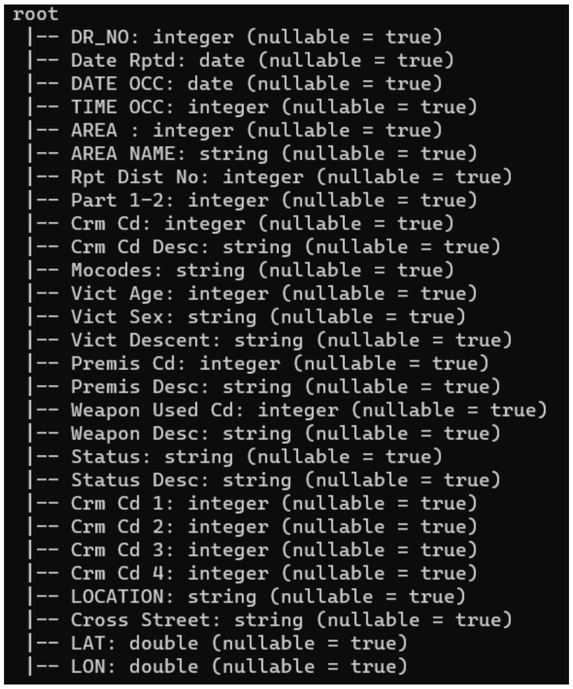
Showing 1 to 5 of 5 entries

Ζητούμενο 2.

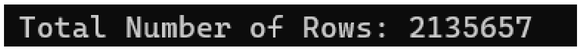
Query: dataframe.py

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import IntegerType, DoubleType, DateType
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("CrimeDataAnalysis").getOrCreate()

file_path = 'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'

df = spark.read.csv(file_path, header=True, inferSchema=True)

df = df.withColumn("Date Rptd", col("Date Rptd").cast(DateType()))
df = df.withColumn("DATE OCC", col("DATE OCC").cast(DateType()))
df = df.withColumn("Vict Age", col("Vict Age").cast(IntegerType()))
df = df.withColumn("LAT", col("LAT").cast(DoubleType()))
df = df.withColumn("LON", col("LON").cast(DoubleType()))

df.printSchema()

print("Total Number of Rows:", df.count())

spark.stop()
```

Αρχικοποιούμε το Spark session, διαβάζουμε το αρχείο Crime_Data_from_2010_to_2019.csv, μετατρέπουμε τις στήλες στους αντίστοιχους τύπους δεδομένων, διαβάζουμε το αρχείο csv σε περιβάλλον Spark, τυπώνουμε τους τύπους δεδομένων κάθε στήλης και τις συνολικές γραμμές σύμφωνα με τα ζητούμενα της εκφώνησης (Date Rptd: date, DATE OCC: date ,Vict Age: integer, LAT: double, LON: double). Η διαδικασία μετατροπής γίνεται μέσω της cast η οποία μετατρέπει τύπους δεδομένων και διασφαλίζει την εξαγωγή ορθών συμπερασμάτων από τα δεδομένα. Παρακάτω φαίνεται η έξοδος στο Apache Spark περιβάλλον μετά την εκτέλεση του script μέσω spark-submit dataframe.py



```
root
 |-- DR_NO: integer (nullable = true)
 |-- Date Rptd: date (nullable = true)
 |-- DATE OCC: date (nullable = true)
 |-- TIME OCC: integer (nullable = true)
 |-- AREA : integer (nullable = true)
 |-- AREA NAME: string (nullable = true)
 |-- Rpt Dist No: integer (nullable = true)
 |-- Part 1-2: integer (nullable = true)
 |-- Crm Cd: integer (nullable = true)
 |-- Crm Cd Desc: string (nullable = true)
 |-- Mocodes: string (nullable = true)
 |-- Vict Age: integer (nullable = true)
 |-- Vict Sex: string (nullable = true)
 |-- Vict Descent: string (nullable = true)
 |-- Premis Cd: integer (nullable = true)
 |-- Premis Desc: string (nullable = true)
 |-- Weapon Used Cd: integer (nullable = true)
 |-- Weapon Desc: string (nullable = true)
 |-- Status: string (nullable = true)
 |-- Status Desc: string (nullable = true)
 |-- Crm Cd 1: integer (nullable = true)
 |-- Crm Cd 2: integer (nullable = true)
 |-- Crm Cd 3: integer (nullable = true)
 |-- Crm Cd 4: integer (nullable = true)
 |-- LOCATION: string (nullable = true)
 |-- Cross Street: string (nullable = true)
 |-- LAT: double (nullable = true)
 |-- LON: double (nullable = true)
```



```
Total Number of Rows: 2135657
```

Ζητούμενο 3.

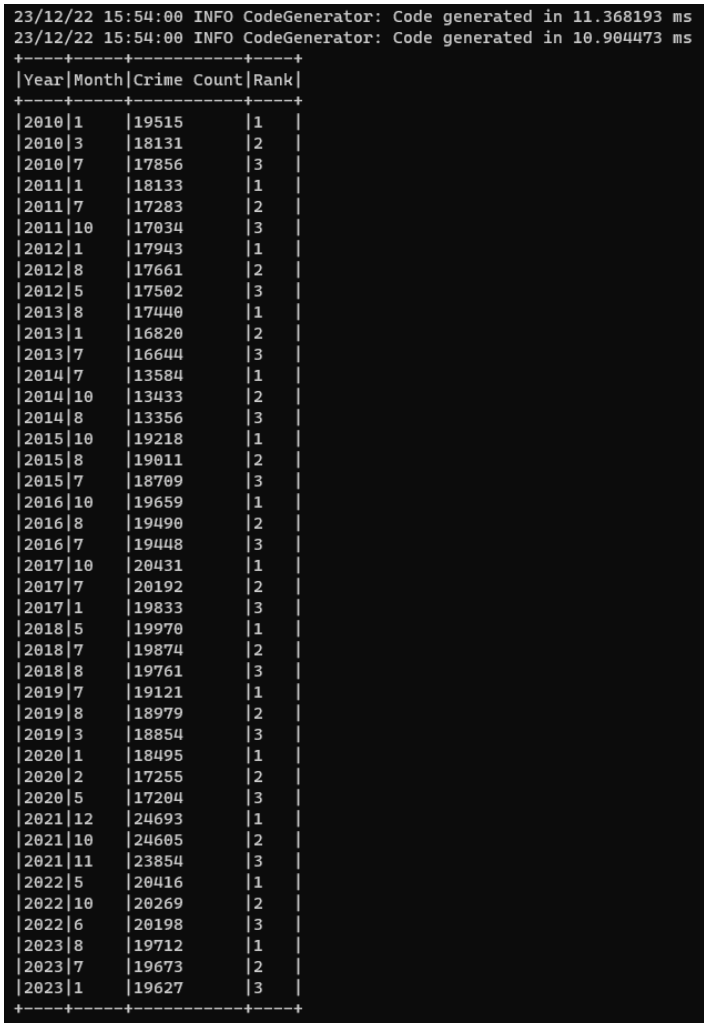
Query: q1df.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, year, month, desc, to_timestamp
from pyspark.sql.window import Window
import pyspark.sql.functions as F

spark = SparkSession.builder \
    .appName("CrimeDataAnalysis") \
    .config("spark.executor.instances", "4") \
    .getOrCreate()

file_path_2010_to_2019 =
'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
file_path_2020_to_present =
'hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv'

crime_data_2010_to_2019 = spark.read.csv(file_path_2010_to_2019, header=True,
inferSchema=True)
```

```
crime_data_2020_to_present = spark.read.csv(file_path_2020_to_present,
header=True, inferSchema=True)

crime_data = crime_data_2010_to_2019.union(crime_data_2020_to_present)


crime_data = crime_data.withColumn('DATE OCC', to_timestamp(col('DATE OCC'),
'MM/dd/yyyy hh:mm:ss a'))
crime_data = crime_data.filter(crime_data['DATE OCC'].isNotNull())
crime_data = crime_data.withColumn('Year', year('DATE OCC'))
crime_data = crime_data.withColumn('Month', month('DATE OCC'))

grouped_data = crime_data.groupBy('Year',
'Month').count().withColumnRenamed('count', 'Crime Count')

windowSpec = Window.partitionBy('Year').orderBy(desc('Crime Count'))

top_months = grouped_data.withColumn('Rank', F.rank().over(windowSpec)) \
                         .filter(col('Rank') <= 3) \
                         .orderBy('Year', 'Rank')

top_months.show(top_months.count(), truncate=False)

spark.stop()
```

Στο Query q1df.py έχουμε, *import SparkSession*: Εισάγουμε το SparkSession, το οποίο είναι το σημείο εισόδου για τον προγραμματισμό του Spark με το API DataFrame. Μέσω της *from pyspark.sql.functions import col, year, month, desc, to_timestamp*, εισάγουμε συγκεκριμένες συναρτήσεις από την PySpark SQL. Συνάρτηση στηλών (col), εξαγωγής έτους και μηνός από ημερομηνίες (year, month), συνάρτηση φθίνουσας σειράς (desc) και μια συνάρτηση για τη μετατροπή συμβολοσειρών σε χρονοσφραγίδες (to_timestamp). *from pyspark.sql.window import Window*: Συνάρτηση Window για υπολογισμούς με βάση το παράθυρο/σύνολο γραμμών. Δημιουργούμε μια SparkSession, η οποία είναι το σημείο εισόδου για την ανάγνωση δεδομένων και την εκτέλεση λειτουργιών SQL. Ορίζουμε το όνομα της εφαρμογής σε "CrimeDataAnalysis". Διαμορφώνουμε τον αριθμό των πυρήνων σε 4. Η getOrCreate() είτε ανακτά μια υπάρχουσα συνεδρία Spark είτε δημιουργεί μια νέα, εάν δεν υπάρχει. Ορίζουμε τα μονοπάτια αρχείων (paths) για δύο σύνολα δεδομένων που είναι αποθηκευμένα στο

HDFS (Hadoop Distributed File System). Διαβάζουμε τα αρχεία CSV σε DataFrames. Η επιλογή *header=True* δηλώνει ότι η πρώτη σειρά των αρχείων περιέχει ονόματα στηλών. Η επιλογή *inferSchema=True* επιτρέπει στο Spark να συμπεραίνει αυτόματα τους τύπους των στηλών. Ενώνουμε τα δύο DataFrames, συγχωνεύοντας τα δεδομένα από το 2010 έως το 2019 με τα δεδομένα από το 2020 έως σήμερα σε ένα ενιαίο DataFrame. Μετατρέπουμε τη στήλη "*DATE OCC*" σε μορφή timestamp. Φιλτράρουμε τις γραμμές όπου η στήλη "*DATE OCC*" είναι null. Προσθέτουμε νέες στήλες *"Year"* και *"Month*" που εξάγονται από τη χρονοσφραγίδα "*DATE OCC".* Ομαδοποιούμε τα δεδομένα ανά "Έτος" και "Μήνας", μετράμε τον αριθμό των εγκλημάτων για κάθε ομάδα και μετονομάζουμε τη στήλη μέτρησης από *“count”* σε *"Crime Count".* Καθορίζουμε μια προδιαγραφή παραθύρου για την κατάτμηση των δεδομένων ανά "Έτος" και ταξινομούμε τα δεδομένα με βάση τον "Αριθμό εγκλημάτων" σε φθίνουσα σειρά. Φιλτράρουμε τα δεδομένα για να διατηρήσουμε μόνο τους 3 κορυφαίους μήνες όσον αφορά τον αριθμό εγκλημάτων για κάθε έτος. Ταξινομούμε το τελικό αποτέλεσμα κατά "Έτος" και "Κατάταξη". Εμφανίζουμε τα τελικά επεξεργασμένα δεδομένα (top_months) που δείχνουν τους 3 πρώτους μήνες με τον υψηλότερο αριθμό εγκλημάτων για κάθε έτος. Ο αριθμός των εμφανιζόμενων γραμμών είναι ίσος με τον συνολικό αριθμό του πλαισίου δεδομένων top_months. Σταματάμε τη συνεδρία Spark, απελευθερώνοντας τους πόρους. Εκτελούμε το Query μέσω της εντολής spark-submit q1df.py. Παρακάτω παρατίθονται τα αποτελέσματα μέσα από το περιβάλλον του Spark.



```
23/12/22 15:54:00 INFO CodeGenerator: Code generated in 11.368193 ms
23/12/22 15:54:00 INFO CodeGenerator: Code generated in 10.904473 ms
+----+-----+-----------+----+
|Year|Month|Crime Count|Rank|
+----+-----+-----------+----+
|2010|1    |19515      |1   |
|2010|3    |18131      |2   |
|2010|7    |17856      |3   |
|2011|1    |18133      |1   |
|2011|7    |17283      |2   |
|2011|10   |17034      |3   |
|2012|1    |17943      |1   |
|2012|8    |17661      |2   |
|2012|5    |17502      |3   |
|2013|8    |17440      |1   |
|2013|1    |16820      |2   |
|2013|7    |16644      |3   |
|2014|7    |13584      |1   |
|2014|10   |13433      |2   |
|2014|8    |13356      |3   |
|2015|10   |19218      |1   |
|2015|8    |19011      |2   |
|2015|7    |18709      |3   |
|2016|10   |19659      |1   |
|2016|8    |19490      |2   |
|2016|7    |19448      |3   |
|2017|10   |20431      |1   |
|2017|7    |20192      |2   |
|2017|1    |19833      |3   |
|2018|5    |19970      |1   |
|2018|7    |19874      |2   |
|2018|8    |19761      |3   |
|2019|7    |19121      |1   |
|2019|8    |18979      |2   |
|2019|3    |18854      |3   |
|2020|1    |18495      |1   |
|2020|2    |17255      |2   |
|2020|5    |17204      |3   |
|2021|12   |24693      |1   |
|2021|10   |24605      |2   |
|2021|11   |23854      |3   |
|2022|5    |20416      |1   |
|2022|10   |20269      |2   |
|2022|6    |20198      |3   |
|2023|8    |19712      |1   |
|2023|7    |19673      |2   |
|2023|1    |19627      |3   |
+----+-----+-----------+----+
```

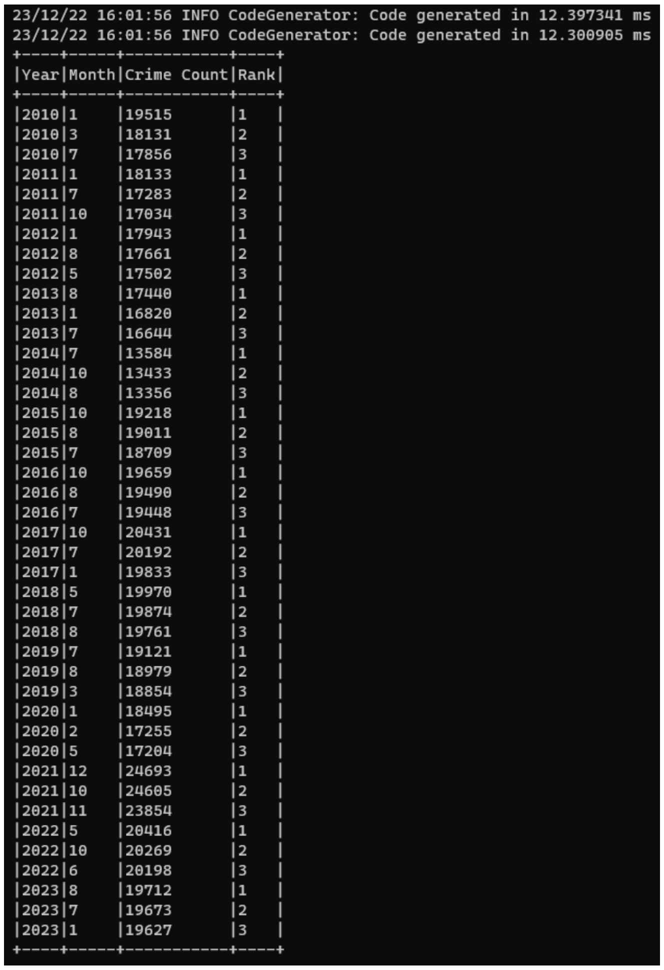
Η εξήγηση του κώδικα για το συγκεκριμένο query είναι πλήρως αναλυτική, ακολουθούν συντομότερες τεχνικές περιγραφές για τις υπόλοιπες υλοποιήσεις.

Query: q1sql.py

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_timestamp

spark = SparkSession.builder \
    .appName("CrimeDataAnalysis") \
    .config("spark.executor.instances", "4") \
    .getOrCreate()

file_path_2010_to_2019 =
'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
file_path_2020_to_present =
'hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv'
```

```
crime_data_2010_to_2019 = spark.read.csv(file_path_2010_to_2019, header=True,
inferSchema=True)
crime_data_2020_to_present = spark.read.csv(file_path_2020_to_present,
header=True, inferSchema=True)

crime_data = crime_data_2010_to_2019.union(crime_data_2020_to_present)

crime_data = crime_data.withColumn('DATE OCC', to_timestamp('DATE OCC',
'MM/dd/yyyy hh:mm:ss a'))

crime_data.createOrReplaceTempView("crime_data")

query = """
SELECT Year, Month, `Crime Count`, Rank
FROM (
    SELECT
        YEAR(`DATE OCC`) AS Year,
        MONTH(`DATE OCC`) AS Month,
        COUNT(*) AS `Crime Count`,
        RANK() OVER (PARTITION BY YEAR(`DATE OCC`) ORDER BY COUNT(*) DESC) AS
Rank
    FROM crime_data
    WHERE `DATE OCC` IS NOT NULL
    GROUP BY Year, Month
) AS RankedData
WHERE Rank <= 3
ORDER BY Year, Rank
"""

top_months = spark.sql(query)

top_months.show(top_months.count(), truncate=False)

spark.stop()
```

Εν συνεχεία, έχουμε το q1sql.py με χρήση SQL API. Δημιουργούμε περίοδο Spark με 4 executors, τοποθετούμε τα csv αρχεία σε Spark Data Frames, τα ενωποιούμε, μετατρέπουμε την στήλη ‘DATE OCC’ σε τύπο δεδομένου datetime, καταχωρούμε το Data Frame σε ένα προσωρινό SQL view (virtual table) και γράφουμε το SQL query που υλοποιεί το ζητούμενο. Τυπώνουμε τους 3 μήνες με τα περισσότερα καταγεγραμμένα εγκλήματα. Η λογική υλοποίησης είναι ίδια με το DataFrame API από άποψη ζητουμένου.



```
23/12/22 16:01:56 INFO CodeGenerator: Code generated in 12.397341 ms
23/12/22 16:01:56 INFO CodeGenerator: Code generated in 12.300905 ms
+----+-----+-----------+----+
|Year|Month|Crime Count|Rank|
+----+-----+-----------+----+
|2010|1    |19515      |1   |
|2010|3    |18131      |2   |
|2010|7    |17856      |3   |
|2011|1    |18133      |1   |
|2011|7    |17283      |2   |
|2011|10   |17034      |3   |
|2012|1    |17943      |1   |
|2012|8    |17661      |2   |
|2012|5    |17502      |3   |
|2013|8    |17440      |1   |
|2013|1    |16820      |2   |
|2013|7    |16644      |3   |
|2014|7    |13584      |1   |
|2014|10   |13433      |2   |
|2014|8    |13356      |3   |
|2015|10   |19218      |1   |
|2015|8    |19011      |2   |
|2015|7    |18709      |3   |
|2016|10   |19659      |1   |
|2016|8    |19490      |2   |
|2016|7    |19448      |3   |
|2017|10   |20431      |1   |
|2017|7    |20192      |2   |
|2017|1    |19833      |3   |
|2018|5    |19970      |1   |
|2018|7    |19874      |2   |
|2018|8    |19761      |3   |
|2019|7    |19121      |1   |
|2019|8    |18979      |2   |
|2019|3    |18854      |3   |
|2020|1    |18495      |1   |
|2020|2    |17255      |2   |
|2020|5    |17204      |3   |
|2021|12   |24693      |1   |
|2021|10   |24605      |2   |
|2021|11   |23854      |3   |
|2022|5    |20416      |1   |
|2022|10   |20269      |2   |
|2022|6    |20198      |3   |
|2023|8    |19712      |1   |
|2023|7    |19673      |2   |
|2023|1    |19627      |3   |
+----+-----+-----------+----+
```

Παρατηρώντας τους χρόνους εκτέλεσης, συμπεραίνουμε ότι οι υλοποίησεις DataFrame API και SQL API είναι πολύ κοντινές από άποψη αποδοτικότητας, με την DataFrame API να πετυχαίνει ελαφρώς καλύτερο χρόνο. Αυτό συμβαίνει διότι στην περίπτωση μας το σύνολο των δεδομένων επεξεργάζεται σχεδόν εφάμιλλα κι από τα δυο APIs. Πειραματιστήκαμε με τη εντολή spark.time(df.show())και καταλήξαμε πως ο χρόνος εκτέλεσης στο Spark περιβάλλον είναι ο καλύτερη επιλογή κριτηρίου σύγκρισης.

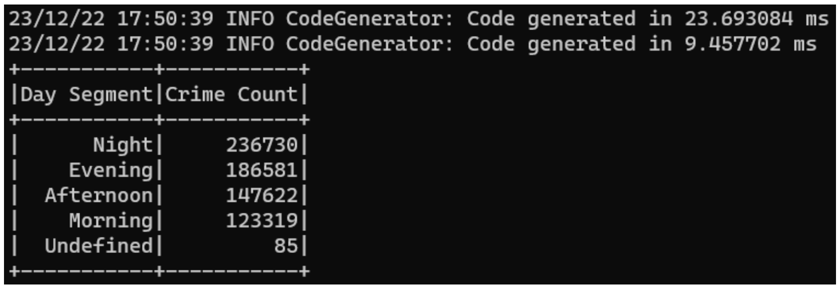
Ζητούμενο 4.

Query: q2df.py

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

spark = SparkSession.builder \
    .appName("CrimeDataAnalysis") \
    .config("spark.executor.instances", "4") \
    .getOrCreate()

def classify_time_segment(time):
    if 500 <= time < 1159:
        return 'Morning'
    elif 1200 <= time < 1659:
        return 'Afternoon'
    elif 1700 <= time < 2059:
        return 'Evening'
    elif (2100 <= time <= 2359) or (0 <= time < 459):
        return 'Night'
    else:
        return 'Undefined'

classify_time_segment_udf = udf(classify_time_segment, StringType())

file_path_2010_to_2019 =
'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
file_path_2020_to_present =
'hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv'

crime_data_2010_to_2019 = spark.read.csv(file_path_2010_to_2019, header=True,
inferSchema=True)
crime_data_2020_to_present = spark.read.csv(file_path_2020_to_present,
header=True, inferSchema=True)

crime_data = crime_data_2010_to_2019.union(crime_data_2020_to_present)

crime_data = crime_data.withColumn('Day Segment',
classify_time_segment_udf(crime_data['TIME OCC']))

street_crimes = crime_data.filter(crime_data['Premis Desc'].like('%STREET%'))
```

```
segment_crime_counts = street_crimes.groupBy('Day
Segment').count().withColumnRenamed('count', 'Crime Count')

sorted_segment_crime_counts = segment_crime_counts.orderBy('Crime Count',
ascending=False)

sorted_segment_crime_counts.show()

spark.stop()
```

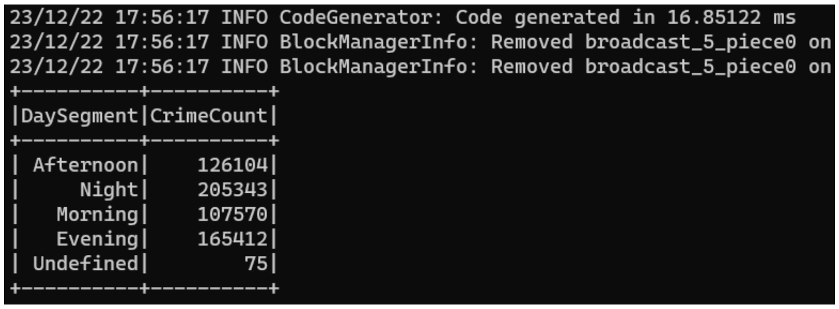
Για την υλοποίηση του q2df.py χρησιμοποιώντας DataFrame δημιουργούμε μια περίοδο Spark, φτιάχνουμε μια συνάρτηση κατηγοριοποίησης της ημέρας σε Πρωϊ, Μεσημέρι, Απόγευμα και Βράδυ, καταχωρούμε την συνάρτηση που φτιάξαμε ως ορισμένη από τον χρήστη, εφαρμόζουμε την συνάρτηση classify_time_segment, διαβάζουμε τα αρχεία και τα ενωποιούμε τοποθετώντας τα σε Data Frames. Παρακάτω φαίνεται το αποτέλεσμα της εκτέλεσης του q2df.py χρησιμοποιώντας DataFrame.


```
23/12/22 17:50:39 INFO CodeGenerator: Code generated in 23.693084 ms
23/12/22 17:50:39 INFO CodeGenerator: Code generated in 9.457702 ms
+-----------+-----------+
|Day Segment|Crime Count|
+-----------+-----------+
|      Night|     236730|
|    Evening|     186581|
|  Afternoon|     147622|
|    Morning|     123319|
|  Undefined|         85|
+-----------+-----------+
```

Query: q2rdd.py

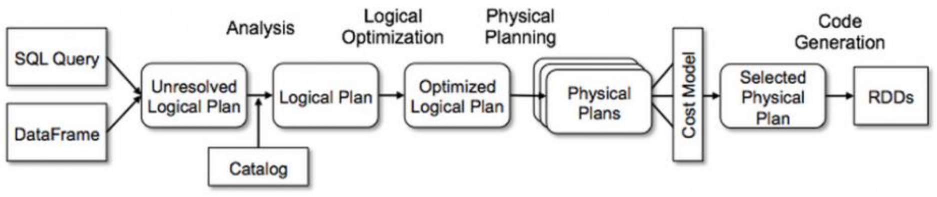
```
from pyspark.sql import SparkSession
from pyspark import SparkContext

spark = SparkSession.builder \
    .appName("CrimeDataAnalysis") \
    .config("spark.executor.instances", "4") \
    .getOrCreate()

sc = spark.sparkContext

def classify_time_segment(time):
    if 500 <= time < 1159:
        return 'Morning'
    elif 1200 <= time < 1659:
        return 'Afternoon'
    elif 1700 <= time < 2059:
        return 'Evening'
    elif (2100 <= time <= 2359) or (0 <= time < 459):
        return 'Night'
    else:
        return 'Undefined'

file_path_2010_to_2019 =
'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
file_path_2020_to_present =
'hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv'

crime_data_2010_to_2019 = sc.textFile(file_path_2010_to_2019)
crime_data_2020_to_present = sc.textFile(file_path_2020_to_present)

crime_data = crime_data_2010_to_2019.union(crime_data_2020_to_present)

header = crime_data.first()
crime_data = crime_data.filter(lambda line: line != header)

crime_data = crime_data.map(lambda line: line.split(','))\
                       .filter(lambda cols: 'STREET' in cols[15])\
                       .map(lambda cols: (classify_time_segment(int(cols[3])),
1))

segment_crime_counts = crime_data.reduceByKey(lambda a, b: a + b)

sorted_segment_crime_counts = segment_crime_counts.sortBy(lambda x: x[1],
ascending=False)

for segment, count in sorted_segment_crime_counts.collect():
    print(f"{segment}: {count}")

spark.stop()
```

Εν συνεχεία, έχουμε το q2rdd.py με χρήση Resilient Distributed Dataset Application Programming Interface (RDD API). Δημιουργούμε περίοδο Spark, φτιάχνουμε την συνάρτηση κατηγοριοποίησης της ημέρας σε Πρωϊ, Μεσημέρι, Απόγευμα και Βράδυ, διαβάζουμε τα αρχεία σε RDDs και τα ενωποιούμε, χωρίζουμε κάθε γραμμή σε στήλες, εφαρμόζουμε τη συνάρτηση και φιλτράρουμε τα εγκλήματα δρόμου. Παρακάτω φαίνεται το αποτέλεσμα της εκτέλεσης με χρήση RDD API.



```
23/12/22 17:56:17 INFO CodeGenerator: Code generated in 16.85122 ms
23/12/22 17:56:17 INFO BlockManagerInfo: Removed broadcast_5_piece0 on
23/12/22 17:56:17 INFO BlockManagerInfo: Removed broadcast_5_piece0 on
+----------+----------+
|DaySegment|CrimeCount|
+----------+----------+
| Afternoon|    126104|
|     Night|    205343|
|   Morning|    107570|
|   Evening|    165412|
| Undefined|        75|
+----------+----------+
```

Παρατηρώντας τους χρόνους εκτέλεσης, όταν χρησιμοποιούμε τα DataFrame, ο κώδικας ειναι πιο αποδοτικός. Αυτό συμβαίνει επειδή τα DataFrames στο Spark είναι χτισμένα πάνω στη μηχανή Spark SQL, η οποία χρησιμοποιεί τον βελτιστοποιητή Catalyst. Επιπλέον, τα DataFrames βελτιστοποιούν καλύτερα τη χρήση μνήμης για δομημένα δεδομένα σε σύγκριση με τα RDDs. Αυτό έχει ως αποτέλεσμα καλύτερες επιδόσεις, ειδικά για μεγάλα σύνολα δεδομένων.



Catalyst Optimizer[1]

Ζητούμενο 5.

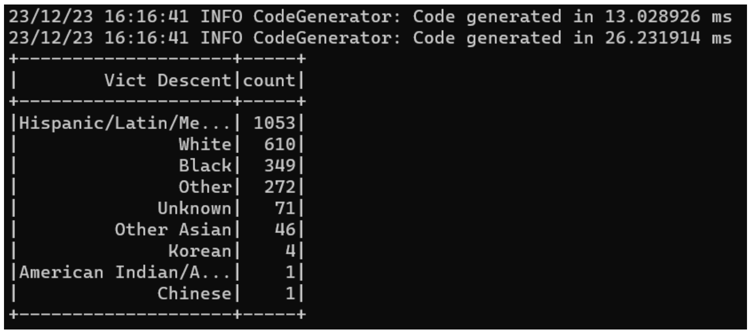
Query: q3df.py

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_date, col, split, udf, regexp_replace, year
from pyspark.sql.types import StringType

spark = SparkSession.builder \
    .appName("CrimeVictimAnalysis") \
    .config("spark.executor.instances", "2") \
    .getOrCreate()
```

```python
crime_data_path = 'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
income_data_path = 'hdfs://master:54310/datasets/income/LA_income_2015.csv'
revgecoding_path = 'hdfs://master:54310/datasets/revgecoding.csv'

crime_data = spark.read.csv(crime_data_path, header=True, inferSchema=True)
revgecoding = spark.read.csv(revgecoding_path, header=True, inferSchema=True)
income_data = spark.read.csv(income_data_path, header=True, inferSchema=True)

income_data = income_data.withColumn('Estimated Median Income',
regexp_replace('Estimated Median Income', '[\$,]', '').cast('float'))

crime_data = crime_data.withColumn('DATE OCC', to_date('DATE OCC', 'MM/dd/yyyy
hh:mm:ss a'))

crime_2015 = crime_data.filter((year(col('DATE OCC')) == 2015) & (col('Vict
Descent').isNotNull()))

crime_2015 = crime_2015.join(revgecoding, ['LAT', 'LON'], 'left_outer')
crime_2015 = crime_2015.withColumn('ZIPcode', split(col('ZIPcode'),
',').getItem(0))

top_3_zip = income_data.orderBy('Estimated Median Income',
ascending=False).limit(3)
bottom_3_zip = income_data.orderBy('Estimated Median Income',
ascending=True).limit(3)
selected_zip_codes = top_3_zip.union(bottom_3_zip).select('Zip Code')

selected_crimes = crime_2015.join(selected_zip_codes, crime_2015.ZIPcode ==
selected_zip_codes['Zip Code'])

def descent_mapping(code):
    mapping = {
        'A': 'Other Asian', 'B': 'Black', 'C': 'Chinese', 'D': 'Cambodian',
        'F': 'Filipino', 'G': 'Guamanian', 'H': 'Hispanic/Latin/Mexican',
        'I': 'American Indian/Alaskan Native', 'J': 'Japanese', 'K': 'Korean',
        'L': 'Laotian', 'O': 'Other', 'P': 'Pacific Islander', 'S': 'Samoan',
        'U': 'Hawaiian', 'V': 'Vietnamese', 'W': 'White', 'X': 'Unknown',
        'Z': 'Asian Indian'
    }
    return mapping.get(code, 'Unknown')

descent_udf = udf(descent_mapping, StringType())

selected_crimes = selected_crimes.withColumn('Vict Descent', descent_udf('Vict
Descent'))
```

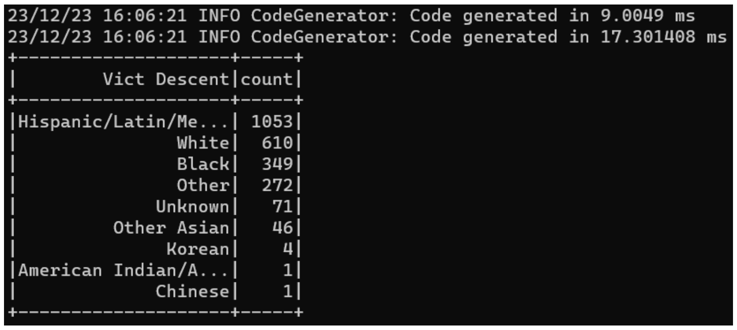
```
victim_count_by_descent = selected_crimes.groupBy('Vict
Descent').count().orderBy('count', ascending=False)

victim_count_by_descent.show()

spark.stop()
```

Για την υλοποίηση του Query q3df.py δημιουργούμε περίοδο Spark με 2 executors, φορτώνουμε και διαβάζουμε τα αρχεία, μετατρέπουμε τα δεδομένα εισοδήματος σε αριθμούς αφού αφαιρέσουμε το σύμβολο του δολαρίου και τα κόμματα, φιλτράρουμε μόνο τα δεδομένα για το 2015 και αποκλείουμε τις περιπτώσεις χωρίς καταγωγή θύματος. Έπειτα, κάνουμε “MAP” τα “LAT” και “LON” σε “ZIP” Codes, εντοπίζουμε τα 3 ZIP Codes με το υψηλότερο και χαμηλότερο εισόδημα και δημιουργούμε την συνάρτηση *descent_mapping*, ώστε να κάνουμε την αντιστοίχιση των γραμμάτων με τις καταγωγές. Εμφανίζουμε τα αποτελέσματα και επαναλαμβάνουμε την ίδια διαδικασία χρησιμοποιώντας 3 και 4 executors. Παρακάτω φαίνεται η έξοδος κι ο χρόνος του q3df.py χρησιμοποιώντας 2 Spark executors.



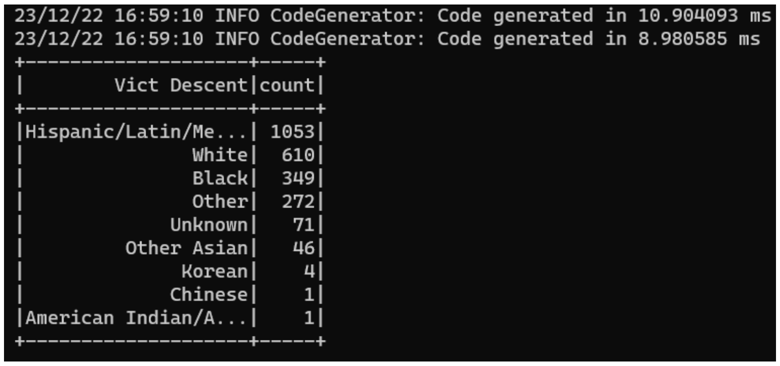
```
23/12/23 16:16:41 INFO CodeGenerator: Code generated in 13.028926 ms
23/12/23 16:16:41 INFO CodeGenerator: Code generated in 26.231914 ms
+--------------------+-----+
|        Vict Descent|count|
+--------------------+-----+
|Hispanic/Latin/Me...| 1053|
|               White|  610|
|               Black|  349|
|               Other|  272|
|             Unknown|   71|
|         Other Asian|   46|
|              Korean|    4|
|American Indian/A...|    1|
|             Chinese|    1|
+--------------------+-----+
```

Έξοδος και χρόνος του q3df.py χρησιμοποιώντας 3 Spark executors.



```
23/12/23 16:06:21 INFO CodeGenerator: Code generated in 9.0049 ms
23/12/23 16:06:21 INFO CodeGenerator: Code generated in 17.301408 ms
+--------------------+-----+
|        Vict Descent|count|
+--------------------+-----+
|Hispanic/Latin/Me...| 1053|
|               White|  610|
|               Black|  349|
|               Other|  272|
|             Unknown|   71|
|         Other Asian|   46|
|              Korean|    4|
|American Indian/A...|    1|
|             Chinese|    1|
+--------------------+-----+
```

Έξοδος και χρόνος του q3df.py χρησιμοποιώντας 4 Spark executors.



```
23/12/22 16:59:10 INFO CodeGenerator: Code generated in 10.904093 ms
23/12/22 16:59:10 INFO CodeGenerator: Code generated in 8.980585 ms
+--------------------+-----+
|        Vict Descent|count|
+--------------------+-----+
|Hispanic/Latin/Me...| 1053|
|               White|  610|
|               Black|  349|
|               Other|  272|
|             Unknown|   71|
|         Other Asian|   46|
|              Korean|    4|
|             Chinese|    1|
|American Indian/A...|    1|
+--------------------+-----+
```

Παρατηρούμε ότι η υλοποίηση με 4 executors είναι πιο αποδοτική και γρήγορη, αφού όσο περισσότερους εκτελεστές έχουμε, τόσο περισσότερες εργασίες μπορούν να εκτελούνται παράλληλα.

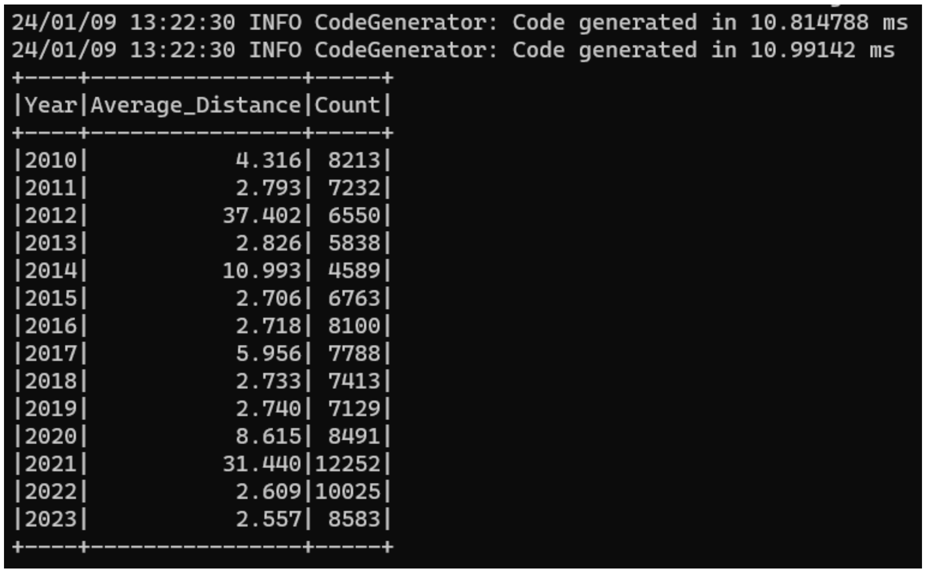
Ζητούμενο 6.

Query: q41adf.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_date, year, udf, col, format_number
from pyspark.sql.types import FloatType
import pandas as pd
import math

spark = SparkSession.builder \
    .appName("CrimeAnalysis") \
    .getOrCreate()

crime_data_path_2010_2019 =
'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
crime_data_path_2020_present =
'hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv'
police_stations_path = 'hdfs://master:54310/datasets/LAPD_Police_Stations.csv'

crime_data_2010_2019 = spark.read.csv(crime_data_path_2010_2019, header=True,
inferSchema=True)
crime_data_2020_present = spark.read.csv(crime_data_path_2020_present,
header=True, inferSchema=True)
police_stations = spark.read.csv(police_stations_path, header=True,
inferSchema=True)
```

```
crime_data = crime_data_2010_2019.union(crime_data_2020_present)

crime_data = crime_data.withColumn('DATE OCC', to_date('DATE OCC', 'MM/dd/yyyy
hh:mm:ss a'))
crime_data = crime_data.withColumn('Year', year('DATE OCC'))

firearm_crimes = crime_data.filter(crime_data['Weapon Used Cd'].between(100,
199))

police_stations_pd = police_stations.toPandas()
area_to_coords = {row['PREC']: (row['Y'], row['X']) for index, row in
police_stations_pd.iterrows()}

def haversine(lat1, lon1, lat2, lon2):
    R = 6371
    dLat = math.radians(lat2 - lat1)
    dLon = math.radians(lon2 - lon1)
    a = math.sin(dLat/2) * math.sin(dLat/2) + math.cos(math.radians(lat1)) *
math.cos(math.radians(lat2)) * math.sin(dLon/2) * math.sin(dLon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    distance = R * c
    return distance

def get_distance(lat1, lon1, area):
    lat2, lon2 = area_to_coords.get(area, (0, 0))
    return haversine(lat1, lon1, lat2, lon2)

get_distance_udf = udf(get_distance, FloatType())

firearm_crimes = firearm_crimes.withColumn('Distance', get_distance_udf('LAT',
'LON', 'AREA '))

annual_stats = firearm_crimes.groupBy('Year').agg(
    {'Distance': 'mean', 'DR_NO': 'count'}
).select(
    "Year", format_number("avg(Distance)", 3).alias('Average_Distance'),
"count(DR_NO)"
).withColumnRenamed('count(DR_NO)', 'Count')

annual_stats = annual_stats.orderBy('Year')

annual_stats.show()

spark.stop()
```

Για την υλοποίηση του q41adf.py δημιουργούμε περίοδο Spark, φορτώνουμε, διαβάζουμε και ενώνουμε τα αρχεία, φιλτράρουμε τα δεδομένα, ώστε να περιλαμβάνονται μόνο περιστατικά που αφορούν πυροβόλα όπλα. Έπειτα, φορτώνουμε τα δεδομένα των αστυνομικών τμημάτων και δημιουργούμε ένα λεξικό που να απεικονίζει τις περιοχές των αστυνομικών τμημάτων στις συντεταγμένες τους. Μετά, ορίζουμε την συνάρτηση *haversine*, η οποία υπολογίζει την απόσταση μεγάλου κύκλου μεταξύ δύο σημείων στην επιφάνεια της γης και χρησιμοποιούμε τον μαθηματικό τύπο Haversine για τον υπολογισμό της απόστασης μεταξύ των εκάστοτε δύο συντεταγμένων. Τελικά, υπολογίζουμε την απόσταση από την τοποθεσία κάθε εγκλήματος που σχετίζεται με πυροβόλο όπλο έως το πλησιέστερο αστυνομικό τμήμα κι εμφανίζουμε τα επιθυμητά αποτελέσματα. Παρακάτω φαίνεται το αποτέλεσμα της υλοποίησης του q41adf.py χρησιμοποιώντας DataFrame.

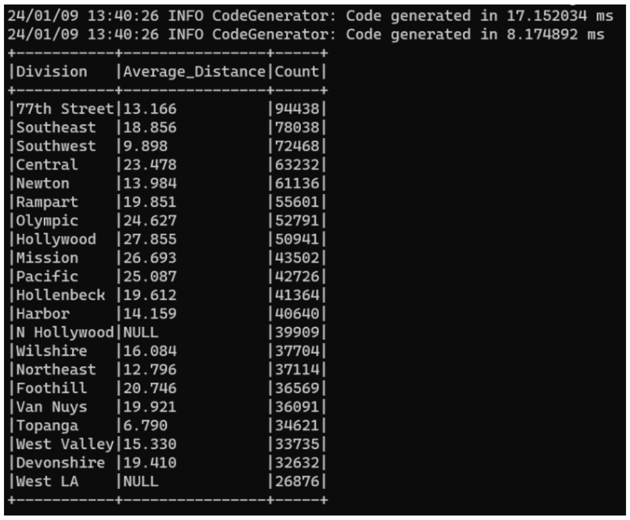


```
24/01/09 13:22:30 INFO CodeGenerator: Code generated in 10.814788 ms
24/01/09 13:22:30 INFO CodeGenerator: Code generated in 10.99142 ms
+----+----------------+-----+
|Year|Average_Distance|Count|
+----+----------------+-----+
|2010|           4.316| 8213|
|2011|           2.793| 7232|
|2012|          37.402| 6550|
|2013|           2.826| 5838|
|2014|          10.993| 4589|
|2015|           2.706| 6763|
|2016|           2.718| 8100|
|2017|           5.956| 7788|
|2018|           2.733| 7413|
|2019|           2.740| 7129|
|2020|           8.615| 8491|
|2021|          31.440|12252|
|2022|           2.609|10025|
|2023|           2.557| 8583|
+----+----------------+-----+
```

Query: q41bdf.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_date, year, udf, col, format_number, upper
from pyspark.sql.types import FloatType
import math

spark = SparkSession.builder \
    .appName("CrimeAnalysis") \
    .getOrCreate()

crime_data_path_2010_2019 =
'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
crime_data_path_2020_present =
'hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv'
police_stations_path = 'hdfs://master:54310/datasets/LAPD_Police_Stations.csv'

crime_data_2010_2019 = spark.read.csv(crime_data_path_2010_2019, header=True,
inferSchema=True)
crime_data_2020_present = spark.read.csv(crime_data_path_2020_present,
header=True, inferSchema=True)
police_stations = spark.read.csv(police_stations_path, header=True,
inferSchema=True)

crime_data = crime_data_2010_2019.union(crime_data_2020_present)

def haversine(lat1, lon1, lat2, lon2):
    if None in [lat1, lon1, lat2, lon2]:
        return None
    R = 6371
    dLat = math.radians(lat2 - lat1)
    dLon = math.radians(lon2 - lon1)
    a = math.sin(dLat/2) * math.sin(dLat/2) + math.cos(math.radians(lat1)) *
math.cos(math.radians(lat2)) * math.sin(dLon/2) * math.sin(dLon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    distance = R * c
    return distance if not math.isnan(distance) else None

get_distance_udf = udf(haversine, FloatType())

crime_data = crime_data.withColumn('DATE OCC', to_date('DATE OCC', 'MM/dd/yyyy
hh:mm:ss a'))
crime_data = crime_data.withColumn('Year', year('DATE OCC'))

weapon_crimes = crime_data.filter(crime_data['Weapon Used Cd'].isNotNull())
weapon_crimes = weapon_crimes.join(police_stations, upper(weapon_crimes['AREA
NAME']) == police_stations['DIVISION'], 'left_outer')

weapon_crimes = weapon_crimes.withColumn('Distance',
get_distance_udf(weapon_crimes['LAT'], weapon_crimes['LON'],
police_stations['Y'], police_stations['X']))

station_stats = weapon_crimes.groupBy('AREA NAME').agg(
    {'Distance': 'mean', 'DR_NO': 'count'}
).withColumnRenamed('avg(Distance)', 'Average_Distance')\
  .withColumnRenamed('count(DR_NO)', 'Count')\
  .withColumnRenamed('AREA NAME', 'Division')\
  .select('Division', format_number('Average_Distance',
3).alias('Average_Distance'), 'Count')\
  .orderBy('Count', ascending=False)

station_stats.show(station_stats.count(), truncate=False)

spark.stop()
```

Για την υλοποίηση του q42bdf.py δημιουργούμε μια περίοδο Spark, φορτώνουμε κι ενωποιούμε τα δεδομένα. Περιλαμβάνει όλα τα εγκλήματα που σχετίζονται με οποιαδήποτε μορφής όπλων και χρησιμοποιεί ακριβές γεωγραφικό μήκος και πλάτος για τον υπολογισμό της απόστασης. Η συνάρτηση Haversine τροποποιείται για να διαχειρίζεται τιμές None και αποτελέσματα NaN (Not a Number) στον υπολογισμό της απόστασης. Τυπώνονται τα στατιστικά στοιχεία ανά αστυνομικό τμήμα ταξινομημένα ανά αριθμό περιστατικών με φθίνουσα σειρά.



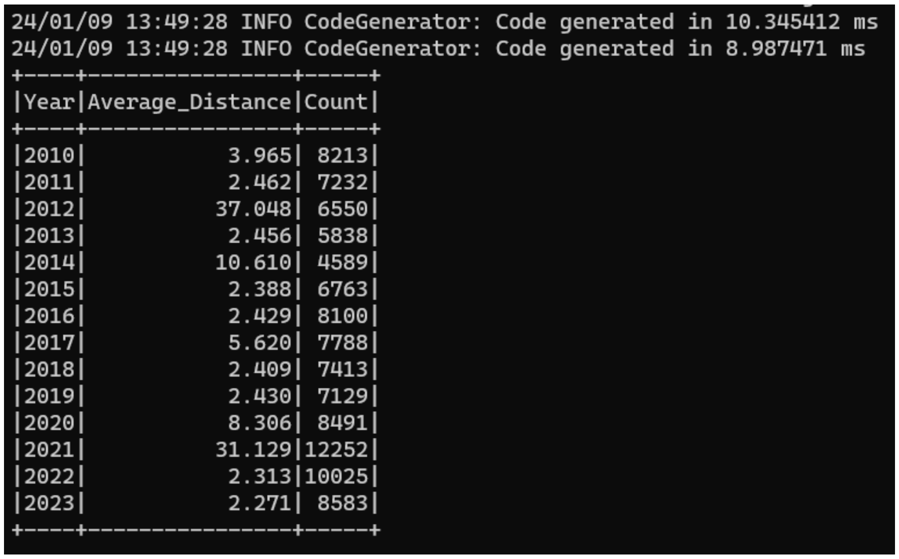
```
24/01/09 13:40:26 INFO CodeGenerator: Code generated in 17.152034 ms
24/01/09 13:40:26 INFO CodeGenerator: Code generated in 8.174892 ms
+-----------+----------------+-----+
|Division   |Average_Distance|Count|
+-----------+----------------+-----+
|77th Street|13.166          |94438|
|Southeast  |18.856          |78038|
|Southwest  |9.898           |72468|
|Central    |23.478          |63232|
|Newton     |13.984          |61136|
|Rampart    |19.851          |55601|
|Olympic    |24.627          |52791|
|Hollywood  |27.855          |50941|
|Mission    |26.693          |43502|
|Pacific    |25.087          |42726|
|Hollenbeck |19.612          |41364|
|Harbor     |14.159          |40640|
|N Hollywood|NULL            |39909|
|Wilshire   |16.084          |37704|
|Northeast  |12.796          |37114|
|Foothill   |20.746          |36569|
|Van Nuys   |19.921          |36091|
|Topanga    |6.790           |34621|
|West Valley|15.330          |33735|
|Devonshire |19.410          |32632|
|West LA    |NULL            |26876|
+-----------+----------------+-----+
```

Query: q42adf.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_date, year, udf, col, min, format_number
from pyspark.sql.types import FloatType
import pandas as pd
import math

spark = SparkSession.builder \
    .appName("CrimeAnalysis") \
    .getOrCreate()

crime_data_path_2010_2019 =
'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
crime_data_path_2020_present =
'hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv'
police_stations_path = 'hdfs://master:54310/datasets/LAPD_Police_Stations.csv'

crime_data_2010_2019 = spark.read.csv(crime_data_path_2010_2019, header=True,
inferSchema=True)
crime_data_2020_present = spark.read.csv(crime_data_path_2020_present,
header=True, inferSchema=True)
police_stations = spark.read.csv(police_stations_path, header=True,
inferSchema=True)

crime_data = crime_data_2010_2019.union(crime_data_2020_present)

crime_data = crime_data.withColumn('DATE OCC', to_date('DATE OCC', 'MM/dd/yyyy
hh:mm:ss a'))
crime_data = crime_data.withColumn('Year', year('DATE OCC'))

firearm_crimes = crime_data.filter(crime_data['Weapon Used Cd'].between(100,
199))

police_stations_pd = police_stations.toPandas()
stations_coords = [(row['Y'], row['X']) for index, row in
police_stations_pd.iterrows()]

def haversine(lat1, lon1, lat2, lon2):
    R = 6371
    dLat = math.radians(lat2 - lat1)
    dLon = math.radians(lon2 - lon1)
    a = math.sin(dLat/2) * math.sin(dLat/2) + math.cos(math.radians(lat1)) *
math.cos(math.radians(lat2)) * math.sin(dLon/2) * math.sin(dLon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    distance = R * c
    return distance

def nearest_station_distance(lat, lon):
    nearest_distance = float('inf')
    for lat2, lon2 in stations_coords:
        distance = haversine(lat, lon, lat2, lon2)
        if distance < nearest_distance:
            nearest_distance = distance
    return nearest_distance

nearest_station_distance_udf = udf(nearest_station_distance, FloatType())

firearm_crimes = firearm_crimes.withColumn('Nearest_Station_Distance',
nearest_station_distance_udf('LAT', 'LON'))

annual_stats = firearm_crimes.groupBy('Year').agg(
    {'Nearest_Station_Distance': 'mean', 'DR_NO': 'count'}
).select(
    "Year", format_number("avg(Nearest_Station_Distance)",
3).alias('Average_Distance'), "count(DR_NO)"
).withColumnRenamed('count(DR_NO)', 'Count')

annual_stats = annual_stats.orderBy('Year')

annual_stats.show()

spark.stop()
```

Για την υλοποίηση του q42adf.py δημιουργούμε περίοδο Spark, φορτώνουμε, διαβάζουμε και ενώνουμε τα αρχεία, φιλτράρουμε τα δεδομένα, ώστε να περιλαμβάνονται περιστατικά οποιασδήποτε μορφής πυροβόλων όπλων.

Μετατρέπουμε τα δεδομένα των αστυνομικών τμημάτων από Spark Dataframe σε Pandas Dataframe κι εξάξουμε τις συντεταγμένες τους. Αυτή η μετατροπή γίνεται προκειμένου να απλοποιήσουμε τον υπολογισμό της εύρεσης του κοντινότερου αστυνομικού τμήματος. Απαιτείται η επεξεργασία κάθε γραμμής των δεδομένων εγκλημάτων με κάθε γραμμή των δεδομένων των αστυνομικών τμημάτων. Έτσι, μέσω των Pandas Dataframe έχουμε την επεξεργασία τοπικά κι αποδοτικότερα. Έπειτα, φορτώνουμε τα δεδομένα των και δημιουργούμε ένα λεξικό που να απεικονίζει τις περιοχές των αστυνομικών τμημάτων στις συντεταγμένες τους. Μετά, ορίζουμε την συνάρτηση *haversine*, η οποία υπολογίζει την απόσταση μεγάλου κύκλου μεταξύ δύο σημείων στην επιφάνεια της γης και χρησιμοποιούμε τον μαθηματικό τύπο Haversine για τον υπολογισμό της απόστασης κάθε εγκλήματος από το κοντινότερο αστυνομικό τμήμα. Προσθέτουμε μια στήλή στο DataFrame των εγκλημάτων που δείχνει την απόσταση από το κοντινότερο τμήμα και ομαδοποιούμε τα δεδομένα με βάση το έτος και τα ταξινομούμε σε αύξουσα σειρά με βάση το πλήθος. Παρακάτω παρατίθεται το αποτέλεσμα της εκτέλεσης του q42adf.py χρησιμοποιώντας DataFrame.

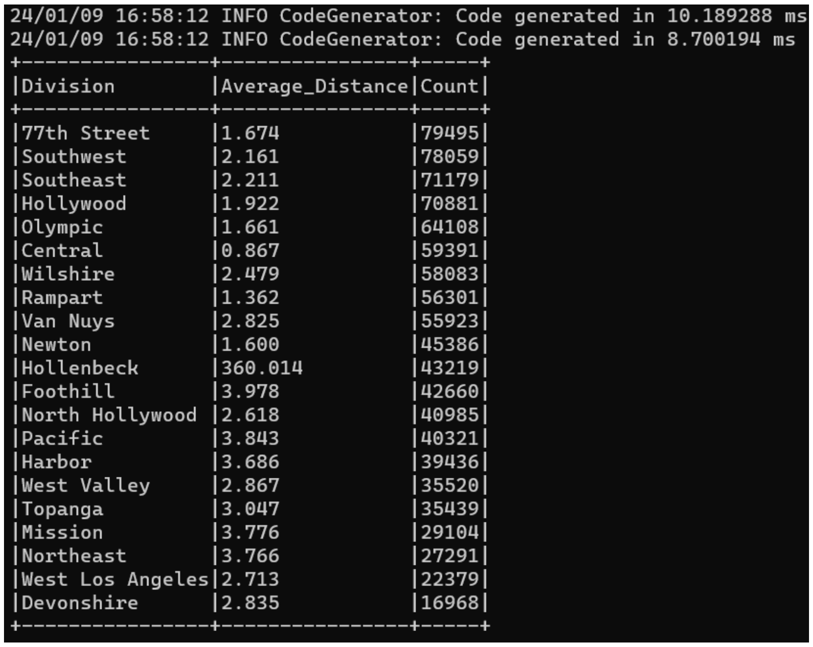

```
24/01/09 13:49:28 INFO CodeGenerator: Code generated in 10.345412 ms
24/01/09 13:49:28 INFO CodeGenerator: Code generated in 8.987471 ms
+----+----------------+-----+
|Year|Average_Distance|Count|
+----+----------------+-----+
|2010|           3.965| 8213|
|2011|           2.462| 7232|
|2012|          37.048| 6550|
|2013|           2.456| 5838|
|2014|          10.610| 4589|
|2015|           2.388| 6763|
|2016|           2.429| 8100|
|2017|           5.620| 7788|
|2018|           2.409| 7413|
|2019|           2.430| 7129|
|2020|           8.306| 8491|
|2021|          31.129|12252|
|2022|           2.313|10025|
|2023|           2.271| 8583|
+----+----------------+-----+
```

Query: q42bdf.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_date, udf, col, format_number, initcap
from pyspark.sql.types import FloatType
import math

spark = SparkSession.builder \
    .appName("CrimeAnalysis") \
    .getOrCreate()

crime_data_path_2010_2019 =
'hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv'
crime_data_path_2020_present =
'hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv'
police_stations_path = 'hdfs://master:54310/datasets/LAPD_Police_Stations.csv'

crime_data_2010_2019 = spark.read.csv(crime_data_path_2010_2019, header=True,
inferSchema=True)
crime_data_2020_present = spark.read.csv(crime_data_path_2020_present,
header=True, inferSchema=True)
```

```
police_stations = spark.read.csv(police_stations_path, header=True,
inferSchema=True)

crime_data = crime_data_2010_2019.union(crime_data_2020_present)

def haversine(lat1, lon1, lat2, lon2):
    if None in [lat1, lon1, lat2, lon2]:
        return None
    R = 6371
    dLat = math.radians(lat2 - lat1)
    dLon = math.radians(lon2 - lon1)
    a = math.sin(dLat/2) * math.sin(dLat/2) + math.cos(math.radians(lat1)) *
math.cos(math.radians(lat2)) * math.sin(dLon/2) * math.sin(dLon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    distance = R * c
    return distance if not math.isnan(distance) else None

get_distance_udf = udf(haversine, FloatType())

crime_data = crime_data.withColumn('DATE OCC', to_date('DATE OCC', 'MM/dd/yyyy
hh:mm:ss a'))
weapon_crimes = crime_data.filter(crime_data['Weapon Used Cd'].isNotNull())

local_police_stations = police_stations.collect()

def find_nearest_station(crime):
    min_distance = float('inf')
    nearest_station = None
    for station in local_police_stations:
        distance = haversine(crime['LAT'], crime['LON'], station['Y'],
station['X'])
        if distance is not None and distance < min_distance:
            min_distance = distance
            nearest_station = station['DIVISION']
    return (crime['DR_NO'], nearest_station, min_distance)

nearest_stations_rdd = weapon_crimes.rdd.map(lambda crime:
find_nearest_station(crime))

columns = ['DR_NO', 'Division', 'Distance']
nearest_stations_df = nearest_stations_rdd.toDF(columns)

station_stats = nearest_stations_df.groupBy('Division').agg(
    {'Distance': 'mean', 'DR_NO': 'count'}
).withColumnRenamed('avg(Distance)', 'Average_Distance')\
  .withColumnRenamed('count(DR_NO)', 'Count')

station_stats = station_stats.withColumn('Division', initcap('Division'))
station_stats = station_stats.select(
    'Division', format_number('Average_Distance', 3).alias('Average_Distance'),
'Count'
).orderBy('Count', ascending=False)

station_stats.show(station_stats.count(), truncate=False)

spark.stop()
```

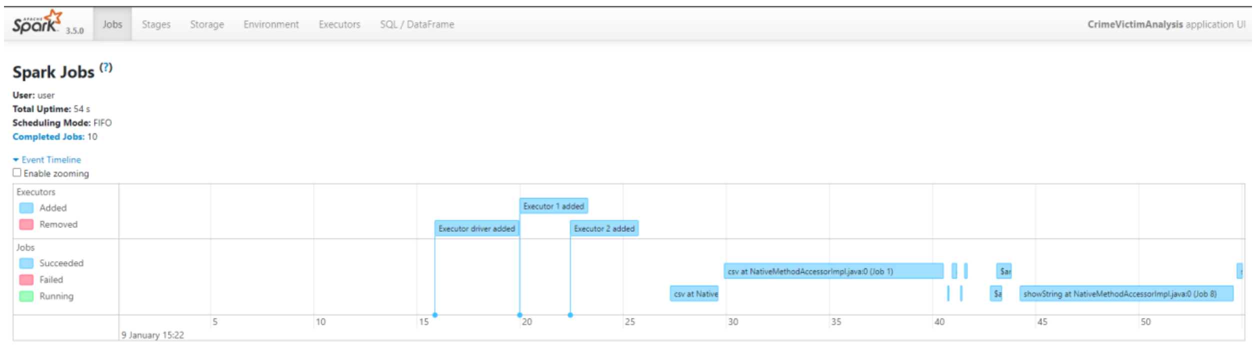
Αντίστοιχα, για την υλοποίηση του q42bdf.py δημιουργούμε μια περίοδο Spark, φορτώνουμε, διαβάζουμε και ενώνουμε τα αρχεία, φιλτράρουμε τα δεδομένα, ώστε να περιλαμβάνονται περιστατικά που αφορούν οποιασδήποτε μορφής όπλων. Η συνάρτηση Haversine τροποποιείται για να διαχειρίζεται τιμές None και αποτελέσματα NaN (Not a Number) στον υπολογισμό της απόστασης. Χρησιμοποιούμε RDD προκειμένου να αντιστοιχίσουμε κάθε έγκλημα στο πλησιέστερο αστυνομικό τμήμα και συγκεντρώνουμε τα δεδομένα ανά αστυνομικό τμήμα υπολογίζοντας την μέση απόσταση και τον αριθμό των εγκλημάτων. Χρησιμοποιούμε την *initcap* για να γράψουμε με κεφαλαία τις ονομασίες των αστυνομικών τμημάτων κι εμφανίζουμε τα αποτελέσματα σε φθίνουσα σειρά με βάση το πλήθος.


```
24/01/09 16:58:12 INFO CodeGenerator: Code generated in 10.189288 ms
24/01/09 16:58:12 INFO CodeGenerator: Code generated in 8.700194 ms
+-----------------+----------------+-----+
|Division         |Average_Distance|Count|
+-----------------+----------------+-----+
|77th Street      |1.674           |79495|
|Southwest        |2.161           |78059|
|Southeast        |2.211           |71179|
|Hollywood        |1.922           |70881|
|Olympic          |1.661           |64108|
|Central          |0.867           |59391|
|Wilshire         |2.479           |58083|
|Rampart          |1.362           |56301|
|Van Nuys         |2.825           |55923|
|Newton           |1.600           |45386|
|Hollenbeck       |360.014         |43219|
|Foothill         |3.978           |42660|
|North Hollywood  |2.618           |40985|
|Pacific          |3.843           |40321|
|Harbor           |3.686           |39436|
|West Valley      |2.867           |35520|
|Topanga          |3.047           |35439|
|Mission          |3.776           |29104|
|Northeast        |3.766           |27291|
|West Los Angeles|2.713           |22379|
|Devonshire       |2.835           |16968|
+-----------------+----------------+-----+
```

Ζητούμενο 7.

Στο q3df.py και στο q41bdf.py έχουμε joins, στα οποία προσθέτουμε την εντολή hint() για το εκάστοτε join, δηλαδή Broadcast, Merge, Shuffle Hash και Shuffle Replicate Nl και την explain() προκειμένου να τυπωθεί ο τρόπος που οργανώνεται η εκτέλεση εσωτερικά του job. Λαμβάνουμε από το Spark UI και το Spark περιβάλλον το γραφικό και περιγραφικό πλάνο της οργάνωσης το οποίο για κάθε περίπτωση παραθέτουμε με εικόνες.

Q3 DataFrame Broadcast 2 Executors



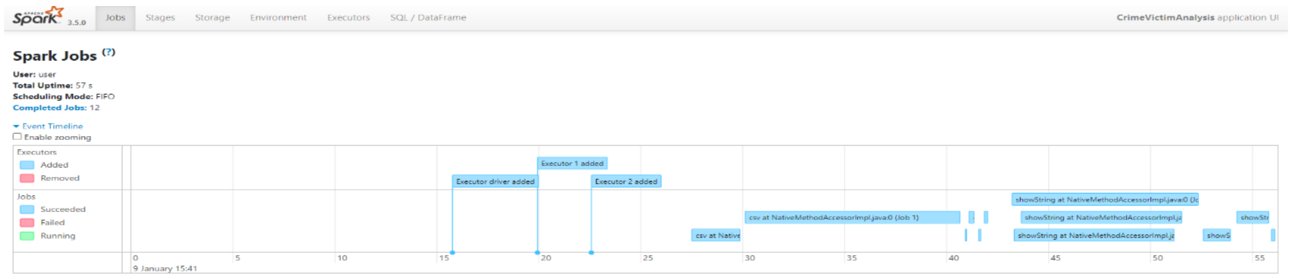
Spark Jobs (?)

User: user
Total Uptime: 54 s
Scheduling Mode: FIFO
Completed Jobs: 10

Physical Plan from Spark Environment q3dfBroadcast_2:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [count#348L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=163]

      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])

         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=160]

            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])

               +- Project [pythonUDF0#351 AS Vict Descent#286]

                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]

                     +- Project [Vict Descent#30]

                        +- BroadcastHashJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner, BuildRight, false

                           :- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]

                           :  +- BroadcastHashJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))], [knownfloatingpointnormalized(normalizenanandzero(LAT#90)), 
knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner, BuildRight, false

                           :     :- Project [Vict Descent#30, LAT#43, LON#44]

                           :     :  +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

                           :     :     +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: 
InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>

                           :     +- BroadcastExchange HashedRelationBroadcastMode(List(knownfloatingpointnormalized(normalizenanandzero(input[0, double, false])), knownfloatingpointnormalized(normalizenanandzero(input[1, double, false]))),false), [plan_id=149]

                           :        +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

                           :           +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 
paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

                           +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [plan_id=153]

                              +- Union

                                 :- Filter isnotnull(Zip Code#113)

                                 :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

                                 :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

                                 :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], 
PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

                                 +- Filter isnotnull(Zip Code#214)

                                    +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

                                       +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                                          +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], 
PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Merge 2 Executors



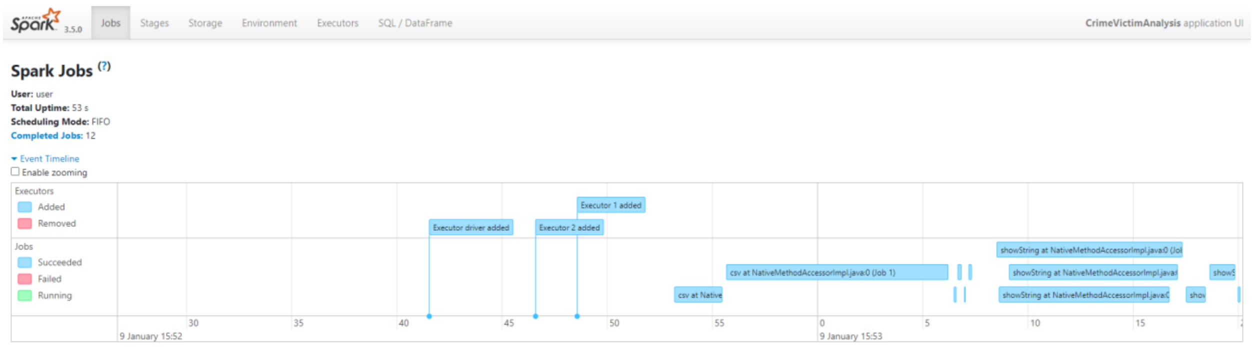
Spark Jobs (?)

User: user
Total Uptime: 57 s
Scheduling Mode: FIFO
Completed Jobs: 12

Physical Plan from Spark Environment q3dfMerge_2:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [count#348L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=171]

      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])

         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=168]

            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])

               +- Project [pythonUDF0#351 AS Vict Descent#286]

                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]

                     +- Project [Vict Descent#30]

                        +- SortMergeJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner

                           :- Sort [cast(ZIPcode#184 as int) ASC NULLS FIRST], false, 0

                           :  +- Exchange hashpartitioning(cast(ZIPcode#184 as int), 200), ENSURE_REQUIREMENTS, [plan_id=158]

                           :     +- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]

                           :        +- SortMergeJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))], [knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner

                           :           :- Sort [knownfloatingpointnormalized(normalizenanandzero(LAT#43)) ASC NULLS FIRST, knownfloatingpointnormalized(normalizenanandzero(LON#44)) ASC NULLS FIRST], false, 0

                           :           :  +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44)), 200), ENSURE_REQUIREMENTS, [plan_id=150]

                           :           :     +- Project [Vict Descent#30, LAT#43, LON#44]

                           :           :        +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

                           :           :           +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>

                           :           +- Sort [knownfloatingpointnormalized(normalizenanandzero(LAT#90)) ASC NULLS FIRST, knownfloatingpointnormalized(normalizenanandzero(LON#91)) ASC NULLS FIRST], false, 0

                           :              +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91)), 200), ENSURE_REQUIREMENTS, [plan_id=151]

                           :                 +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

                           :                    +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

                           +- Sort [Zip Code#113 ASC NULLS FIRST], false, 0

                              +- Exchange hashpartitioning(Zip Code#113, 200), ENSURE_REQUIREMENTS, [plan_id=159]

                                 +- Union

                                    :- Filter isnotnull(Zip Code#113)

                                    :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

                                    :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

                                    :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

                                    +- Filter isnotnull(Zip Code#214)

                                       +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

                                          +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                                             +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Shuffle Hash 2 Executors



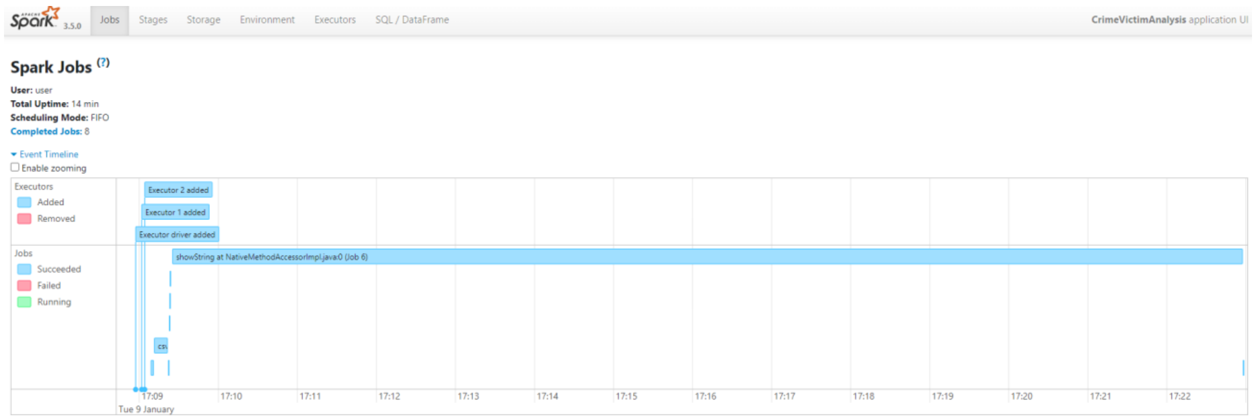
Spark Jobs (?)

User: user
Total Uptime: 53 s
Scheduling Mode: FIFO
Completed Jobs: 12

Physical Plan from Spark Environment q3dfShuffleHash_2:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- ShuffledHashJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner, BuildLeft

   :- Exchange hashpartitioning(cast(ZIPcode#184 as int), 200), ENSURE_REQUIREMENTS, [plan_id=127]

   :  +- Project [LAT#43, LON#44, DR_NO#17, Date Rptd#18, DATE OCC#124, TIME OCC#20, AREA #21, AREA NAME#22, Rpt Dist No#23, Part 1-2#24, Crm Cd#25, Crm Cd Desc#26, Mocodes#27, Vict Age#28, Vict Sex#29, Vict Descent#30, Premis Cd#31, Premis Desc#32, Weapon Used Cd#33, Weapon Desc#34, Status#35, Status Desc#36, Crm Cd 1#37, Crm Cd 2#38, ... 5 more fields]

   :     +- ShuffledHashJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))], [knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner, BuildLeft

   :        :- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44)), 200), ENSURE_REQUIREMENTS, [plan_id=121]

   :        :  +- Project [DR_NO#17, Date Rptd#18, cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date) AS DATE OCC#124, TIME OCC#20, AREA #21, AREA NAME#22, Rpt Dist No#23, Part 1-2#24, Crm Cd#25, Crm Cd Desc#26, Mocodes#27, Vict Age#28, Vict Sex#29, Vict Descent#30, Premis Cd#31, Premis Desc#32, Weapon Used Cd#33, Weapon Desc#34, Status#35, Status Desc#36, Crm Cd 1#37, Crm Cd 2#38, Crm Cd 3#39, Crm Cd 4#40, ... 4 more fields]

   :        :     +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

   :        :        +- FileScan csv [DR_NO#17,Date Rptd#18,DATE OCC#19,TIME OCC#20,AREA #21,AREA NAME#22,Rpt Dist No#23,Part 1-2#24,Crm Cd#25,Crm Cd Desc#26,Mocodes#27,Vict Age#28,Vict Sex#29,Vict Descent#30,Premis Cd#31,Premis Desc#32,Weapon Used Cd#33,Weapon Desc#34,Status#35,Status Desc#36,Crm Cd 1#37,Crm Cd 2#38,Crm Cd 3#39,Crm Cd 4#40,... 4 more fields] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DR_NO:int,Date Rptd:string,DATE OCC:string,TIME OCC:int,AREA :int,AREA NAME:string,Rpt Dis...

   :        +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91)), 200), ENSURE_REQUIREMENTS, [plan_id=122]

   :           +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

   :              +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

   +- Exchange hashpartitioning(Zip Code#113, 200), ENSURE_REQUIREMENTS, [plan_id=128]

      +- Union

         :- Filter isnotnull(Zip Code#113)

         :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

         :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

         :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

         +- Filter isnotnull(Zip Code#214)

            +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

               +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                  +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Shuffle Replicate Nl 2 Executors



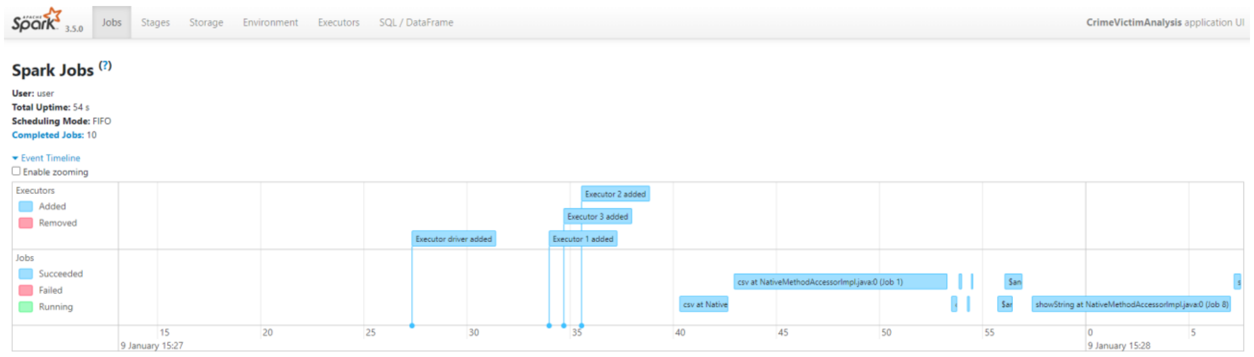
Spark Jobs (?)

User: user
Total Uptime: 14 min
Scheduling Mode: FIFO
Completed Jobs: 8

Physical Plan from Spark Environment q3dfShuffleRep_2:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [count#348L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=241]

      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])

         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=238]

            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])

               +- Project [pythonUDF0#351 AS Vict Descent#286]

                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]

                     +- Project [Vict Descent#30]

                        +- CartesianProduct (cast(ZIPcode#184 as int) = Zip Code#113)

                           :- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]

                           :  +- CartesianProduct ((knownfloatingpointnormalized(normalizenanandzero(LAT#43)) = knownfloatingpointnormalized(normalizenanandzero(LAT#90))) AND (knownfloatingpointnormalized(normalizenanandzero(LON#44)) = knownfloatingpointnormalized(normalizenanandzero(LON#91))))

                           :     :- Project [Vict Descent#30, LAT#43, LON#44]

                           :     :  +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

                           :     :     +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>

                           :     +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

                           :        +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

                           +- Union

                              :- Filter isnotnull(Zip Code#113)

                              :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

                              :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

                              :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

                              +- Filter isnotnull(Zip Code#214)

                                 +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

                                    +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                                       +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Broadcast 3 Executors



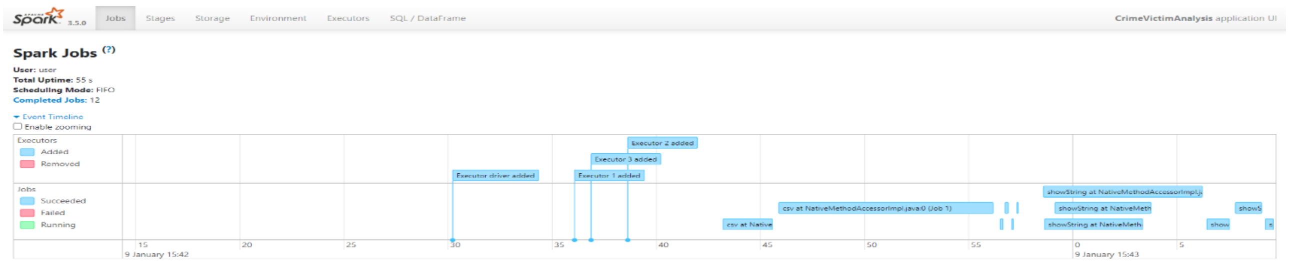
Spark Jobs (?)

User: user
Total Uptime: 54 s
Scheduling Mode: FIFO
Completed Jobs: 10

Physical Plan from Spark Environment q3dfBroadcast_3:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [count#348L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=163]

      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])

         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=160]

            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])

               +- Project [pythonUDF0#351 AS Vict Descent#286]

                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]

                     +- Project [Vict Descent#30]

                        +- BroadcastHashJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner, BuildRight, false

                           :- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]

                           :  +- BroadcastHashJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))], [knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner, BuildRight, false

                           :     :- Project [Vict Descent#30, LAT#43, LON#44]

                           :     :  +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

                           :     :     +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>

                           :     +- BroadcastExchange HashedRelationBroadcastMode(List(knownfloatingpointnormalized(normalizenanandzero(input[0, double, false])), knownfloatingpointnormalized(normalizenanandzero(input[1, double, false]))),false), [plan_id=149]

                           :        +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

                           :           +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

                           +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [plan_id=153]

                              +- Union

                                 :- Filter isnotnull(Zip Code#113)

                                 :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

                                 :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

                                 :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

                                 +- Filter isnotnull(Zip Code#214)

                                    +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

                                       +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                                          +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Merge 3 Executors



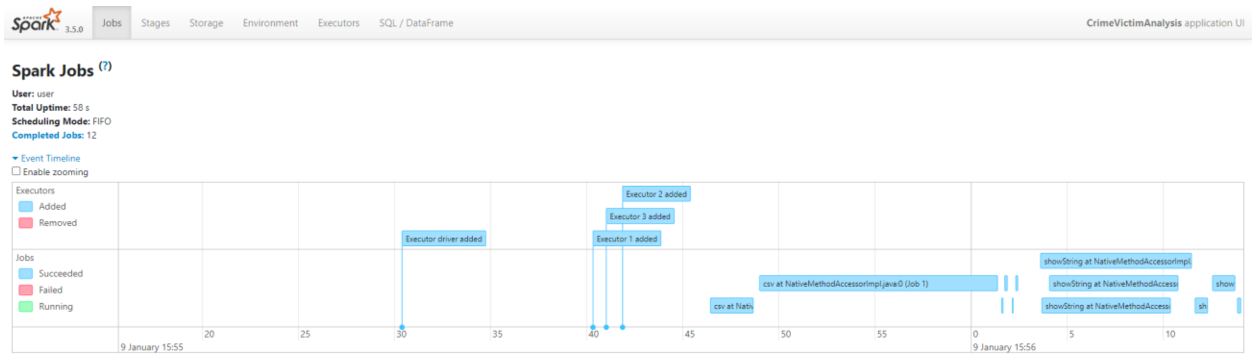
Spark Jobs (?)

User: user
Total Uptime: 55 s
Scheduling Mode: FIFO
Completed Jobs: 12

Physical Plan from Spark Environment q3dfMerge_3:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [count#348L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=171]

      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])

         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=168]

            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])

               +- Project [pythonUDF0#351 AS Vict Descent#286]

                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]

                     +- Project [Vict Descent#30]

                        +- SortMergeJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner

                           :- Sort [cast(ZIPcode#184 as int) ASC NULLS FIRST], false, 0

                           :  +- Exchange hashpartitioning(cast(ZIPcode#184 as int), 200), ENSURE_REQUIREMENTS, [plan_id=158]

                           :     +- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]

                           :        +- SortMergeJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))], [knownfloatingpointnormalized(normalizenanandzero(LAT#90)),
knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner

                           :           :- Sort [knownfloatingpointnormalized(normalizenanandzero(LAT#43)) ASC NULLS FIRST, knownfloatingpointnormalized(normalizenanandzero(LON#44)) ASC NULLS FIRST], false, 0

                           :           :  +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44)), 200), ENSURE_REQUIREMENTS, [plan_id=150]

                           :           :     +- Project [Vict Descent#30, LAT#43, LON#44]

                           :           :        +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

                           :           :           +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location:
InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>

                           :           +- Sort [knownfloatingpointnormalized(normalizenanandzero(LAT#90)) ASC NULLS FIRST, knownfloatingpointnormalized(normalizenanandzero(LON#91)) ASC NULLS FIRST], false, 0

                           :              +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91)), 200), ENSURE_REQUIREMENTS, [plan_id=151]

                           :                 +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

                           :                    +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1
paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

                           +- Sort [Zip Code#113 ASC NULLS FIRST], false, 0

                              +- Exchange hashpartitioning(Zip Code#113, 200), ENSURE_REQUIREMENTS, [plan_id=159]

                                 +- Union

                                    :- Filter isnotnull(Zip Code#113)

                                    :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

                                    :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

                                    :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [],
PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

                                    +- Filter isnotnull(Zip Code#214)

                                       +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

                                          +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                                             +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [],
PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Shuffle Hash 3 Executors



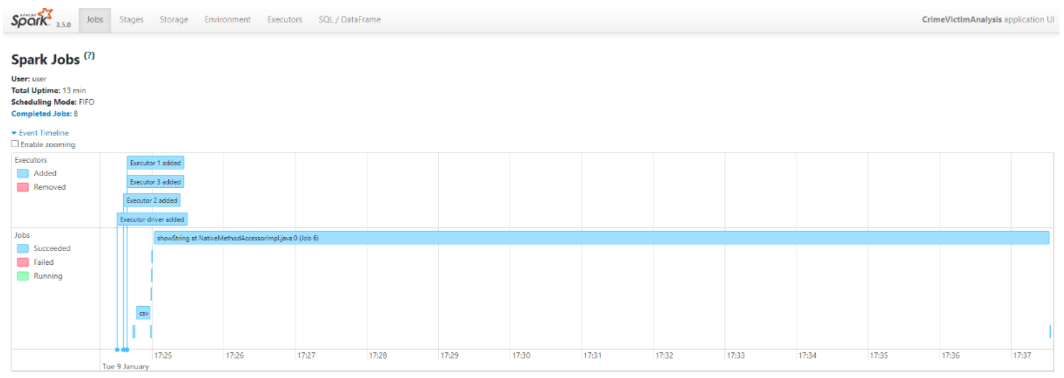
Spark Jobs (?)

User: user
Total Uptime: 58 s
Scheduling Mode: FIFO
Completed Jobs: 12

Physical Plan from Spark Environment q3dfShuffleHash_3:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- ShuffledHashJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner, BuildLeft

   :- Exchange hashpartitioning(cast(ZIPcode#184 as int), 200), ENSURE_REQUIREMENTS, [plan_id=127]

   :  +- Project [LAT#43, LON#44, DR_NO#17, Date Rptd#18, DATE OCC#124, TIME OCC#20, AREA #21, AREA NAME#22, Rpt Dist No#23, Part 1-2#24, Crm Cd#25, Crm Cd Desc#26, Mocodes#27, Vict Age#28, Vict Sex#29, Vict Descent#30, Premis Cd#31, Premis Desc#32, Weapon Used Cd#33, Weapon Desc#34, Status#35, Status Desc#36, Crm Cd 1#37, Crm Cd 2#38, ... 5 more fields]

   :     +- ShuffledHashJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))], [knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner, BuildLeft

   :        :- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44)), 200), ENSURE_REQUIREMENTS, [plan_id=121]

   :        :  +- Project [DR_NO#17, Date Rptd#18, cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date) AS DATE OCC#124, TIME OCC#20, AREA #21, AREA NAME#22, Rpt Dist No#23, Part 1-2#24, Crm Cd#25, Crm Cd Desc#26, Mocodes#27, Vict Age#28, Vict Sex#29, Vict Descent#30, Premis Cd#31, Premis Desc#32, Weapon Used Cd#33, Weapon Desc#34, Status#35, Status Desc#36, Crm Cd 1#37, Crm Cd 2#38, Crm Cd 3#39, Crm Cd 4#40, ... 4 more fields]

   :        :     +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

   :        :        +- FileScan csv [DR_NO#17,Date Rptd#18,DATE OCC#19,TIME OCC#20,AREA #21,AREA NAME#22,Rpt Dist No#23,Part 1-2#24,Crm Cd#25,Crm Cd Desc#26,Mocodes#27,Vict Age#28,Vict Sex#29,Vict Descent#30,Premis Cd#31,Premis Desc#32,Weapon Used Cd#33,Weapon Desc#34,Status#35,Status Desc#36,Crm Cd 1#37,Crm Cd 2#38,Crm Cd 3#39,Crm Cd 4#40,... 4 more fields] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DR_NO:int,Date Rptd:string,DATE OCC:string,TIME OCC:int,AREA :int,AREA NAME:string,Rpt Dis...

   :        +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91)), 200), ENSURE_REQUIREMENTS, [plan_id=122]

   :           +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

   :              +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

   +- Exchange hashpartitioning(Zip Code#113, 200), ENSURE_REQUIREMENTS, [plan_id=128]

      +- Union

         :- Filter isnotnull(Zip Code#113)

         :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

         :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

         :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

         +- Filter isnotnull(Zip Code#214)

            +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

               +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                  +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Shuffle Replicate Nl 3 Executors



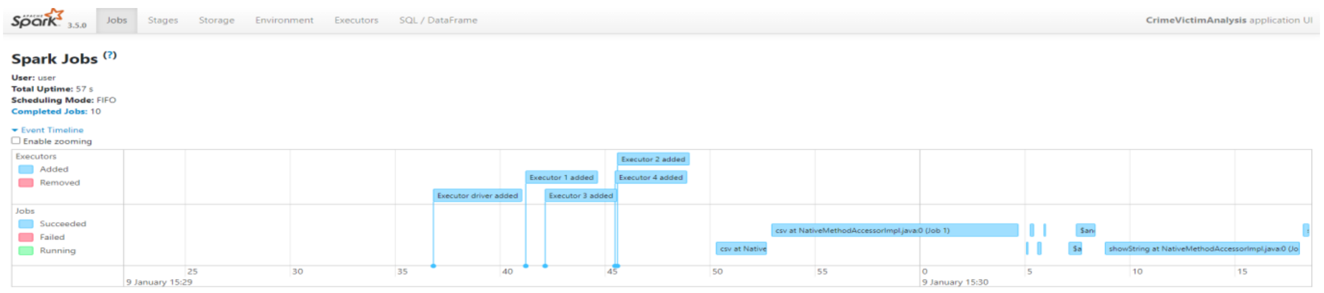
Spark Jobs (?)

User: user
Total Uptime: 13 min
Scheduling Mode: FIFO
Completed Jobs: 8

Physical Plan from Spark Environment q3dfShuffleRep_3:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [count#348L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=241]

      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])

         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=238]

            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])

               +- Project [pythonUDF0#351 AS Vict Descent#286]

                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]

                     +- Project [Vict Descent#30]

                        +- CartesianProduct (cast(ZIPcode#184 as int) = Zip Code#113)

                           :- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]

                           :  +- CartesianProduct ((knownfloatingpointnormalized(normalizenanandzero(LAT#43)) = knownfloatingpointnormalized(normalizenanandzero(LAT#90))) AND (knownfloatingpointnormalized(normalizenanandzero(LON#44)) = knownfloatingpointnormalized(normalizenanandzero(LON#91))))

                           :     :- Project [Vict Descent#30, LAT#43, LON#44]

                           :     :  +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

                           :     :     +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>

                           :     +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

                           :        +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

                           +- Union

                              :- Filter isnotnull(Zip Code#113)

                              :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

                              :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

                              :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

                              +- Filter isnotnull(Zip Code#214)

                                 +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

                                    +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                                       +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Broadcast 4 Executors



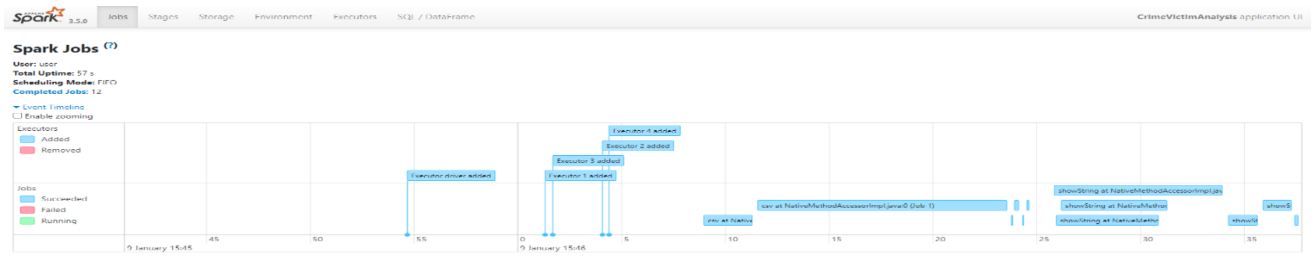
Spark Jobs (?)

User: user
Total Uptime: 57 s
Scheduling Mode: FIFO
Completed Jobs: 10

Physical Plan from Spark Environment q3dfBroadcast_4:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [count#348L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=163]

      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])

         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=160]

            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])

               +- Project [pythonUDF0#351 AS Vict Descent#286]

                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]

                     +- Project [Vict Descent#30]

                        +- BroadcastHashJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner, BuildRight, false

                           :- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]

                           :  +- BroadcastHashJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))],
[knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner, BuildRight, false

                           :     :- Project [Vict Descent#30, LAT#43, LON#44]

                           :     :  +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

                           :     :     +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType,
Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)],
ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>

                           :     +- BroadcastExchange HashedRelationBroadcastMode(List(knownfloatingpointnormalized(normalizenanandzero(input[0, double, false])),
knownfloatingpointnormalized(normalizenanandzero(input[1, double, false]))),false), [plan_id=149]

                           :        +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

                           :           +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location:
InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

                           +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)),false), [plan_id=153]

                              +- Union

                                 :- Filter isnotnull(Zip Code#113)

                                 :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

                                 :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

                                 :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1
paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

                                 +- Filter isnotnull(Zip Code#214)

                                    +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

                                       +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                                          +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1
paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Merge 4 Executors



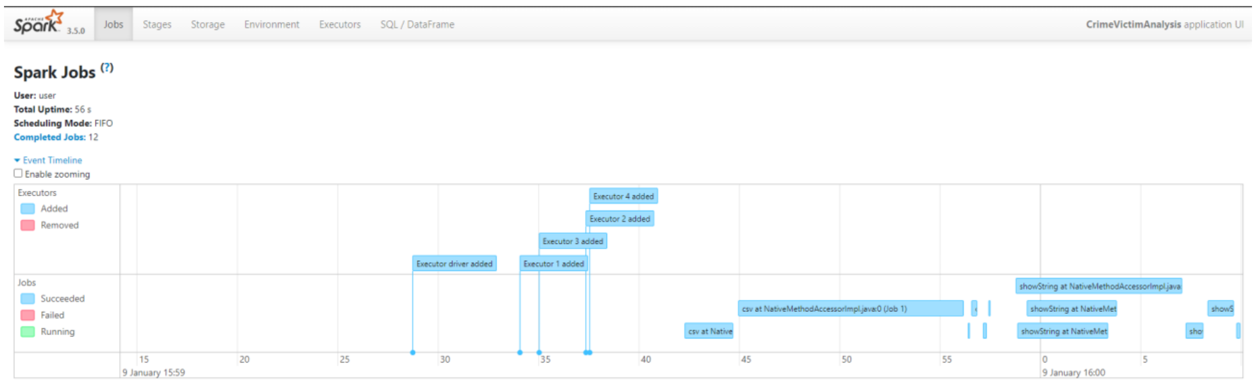
Spark Jobs (?)

User: user
Total Uptime: 57 s
Scheduling Mode: FIFO
Completed Jobs: 12

Physical Plan from Spark Environment q3dfMerge_4:

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Sort [count#348L DESC NULLS LAST], true, 0
   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=171]
      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])
         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=168]
            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])
               +- Project [pythonUDF0#351 AS Vict Descent#286]
                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]
                     +- Project [Vict Descent#30]
                        +- SortMergeJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner
                           :- Sort [cast(ZIPcode#184 as int) ASC NULLS FIRST], false, 0
                           :  +- Exchange hashpartitioning(cast(ZIPcode#184 as int), 200), ENSURE_REQUIREMENTS, [plan_id=158]
                           :     +- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]
                           :        +- SortMergeJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))], [knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner
                           :           :- Sort [knownfloatingpointnormalized(normalizenanandzero(LAT#43)) ASC NULLS FIRST, knownfloatingpointnormalized(normalizenanandzero(LON#44)) ASC NULLS FIRST], false, 0
                           :           :  +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44)), 200), ENSURE_REQUIREMENTS, [plan_id=150]
                           :           :     +- Project [Vict Descent#30, LAT#43, LON#44]
                           :           :        +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))
                           :           :           +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>
                           :           +- Sort [knownfloatingpointnormalized(normalizenanandzero(LAT#90)) ASC NULLS FIRST, knownfloatingpointnormalized(normalizenanandzero(LON#91)) ASC NULLS FIRST], false, 0
                           :              +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91)), 200), ENSURE_REQUIREMENTS, [plan_id=151]
                           :                 +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))
                           :                    +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>
                           +- Sort [Zip Code#113 ASC NULLS FIRST], false, 0
                              +- Exchange hashpartitioning(Zip Code#113, 200), ENSURE_REQUIREMENTS, [plan_id=159]
                                 +- Union
                                    :- Filter isnotnull(Zip Code#113)
                                    :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])
                                    :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]
                                    :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
                                    +- Filter isnotnull(Zip Code#214)
                                       +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])
                                          +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]
                                             +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Shuffle Hash 4 Executors



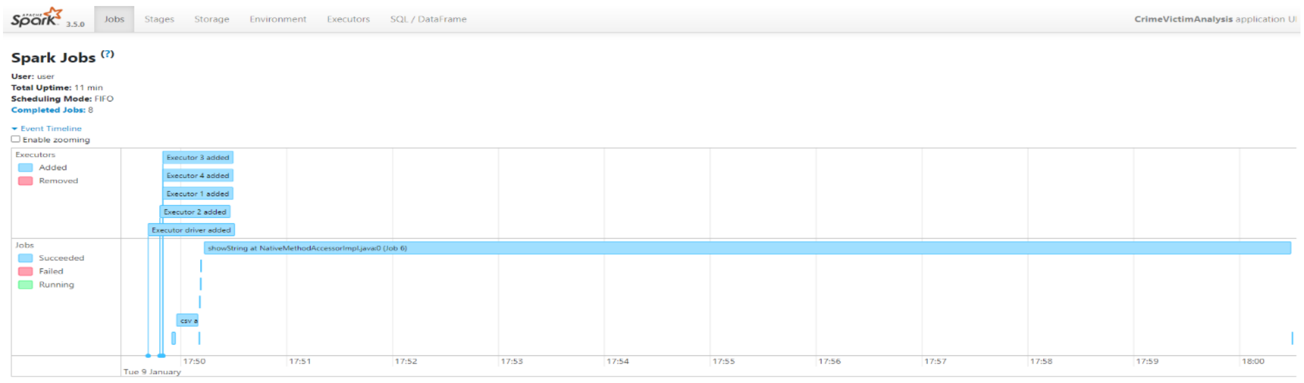
Spark Jobs (?)

User: user
Total Uptime: 56 s
Scheduling Mode: FIFO
Completed Jobs: 12

Physical Plan from Spark Environment q3dfShuffleHash_4:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- ShuffledHashJoin [cast(ZIPcode#184 as int)], [Zip Code#113], Inner, BuildLeft

   :- Exchange hashpartitioning(cast(ZIPcode#184 as int), 200), ENSURE_REQUIREMENTS, [plan_id=127]

   :  +- Project [LAT#43, LON#44, DR_NO#17, Date Rptd#18, DATE OCC#124, TIME OCC#20, AREA #21, AREA NAME#22, Rpt Dist No#23, Part 1-2#24, Crm Cd#25, Crm Cd Desc#26, Mocodes#27, Vict Age#28, Vict Sex#29, Vict Descent#30, Premis Cd#31, Premis Desc#32, Weapon Used Cd#33, Weapon Desc#34, Status#35, Status Desc#36, Crm Cd 1#37, Crm Cd 2#38, ... 5 more fields]

   :     +- ShuffledHashJoin [knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44))], [knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91))], Inner, BuildLeft

   :        :- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#43)), knownfloatingpointnormalized(normalizenanandzero(LON#44)), 200), ENSURE_REQUIREMENTS, [plan_id=121]

   :        :  +- Project [DR_NO#17, Date Rptd#18, cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date) AS DATE OCC#124, TIME OCC#20, AREA #21, AREA NAME#22, Rpt Dist No#23, Part 1-2#24, Crm Cd#25, Crm Cd Desc#26, Mocodes#27, Vict Age#28, Vict Sex#29, Vict Descent#30, Premis Cd#31, Premis Desc#32, Weapon Used Cd#33, Weapon Desc#34, Status#35, Status Desc#36, Crm Cd 1#37, Crm Cd 2#38, Crm Cd 3#39, Crm Cd 4#40, ... 4 more fields]

   :        :     +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

   :        :        +- FileScan csv [DR_NO#17,Date Rptd#18,DATE OCC#19,TIME OCC#20,AREA #21,AREA NAME#22,Rpt Dist No#23,Part 1-2#24,Crm Cd#25,Crm Cd Desc#26,Mocodes#27,Vict Age#28,Vict Sex#29,Vict Descent#30,Premis Cd#31,Premis Desc#32,Weapon Used Cd#33,Weapon Desc#34,Status#35,Status Desc#36,Crm Cd 1#37,Crm Cd 2#38,Crm Cd 3#39,Crm Cd 4#40,... 4 more fields] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)], ReadSchema: struct<DR_NO:int,Date Rptd:string,DATE OCC:string,TIME OCC:int,AREA :int,AREA NAME:string,Rpt Dis...

   :        +- Exchange hashpartitioning(knownfloatingpointnormalized(normalizenanandzero(LAT#90)), knownfloatingpointnormalized(normalizenanandzero(LON#91)), 200), ENSURE_REQUIREMENTS, [plan_id=122]

   :           +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

   :              +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

   +- Exchange hashpartitioning(Zip Code#113, 200), ENSURE_REQUIREMENTS, [plan_id=128]

      +- Union

         :- Filter isnotnull(Zip Code#113)

         :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

         :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

         :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

         +- Filter isnotnull(Zip Code#214)

            +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

               +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                  +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q3 DataFrame Shuffle Replicate Nl 4 Executors



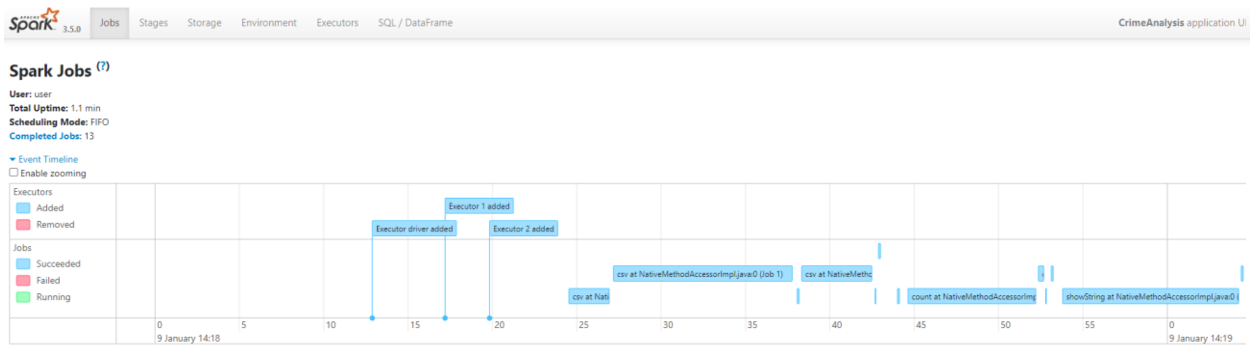
Spark Jobs (?)

User: user
Total Uptime: 11 min
Scheduling Mode: FIFO
Completed Jobs: 8

Physical Plan from Spark Environment q3dfShuffleRep_4:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [count#348L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(count#348L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=241]

      +- HashAggregate(keys=[Vict Descent#286], functions=[count(1)])

         +- Exchange hashpartitioning(Vict Descent#286, 200), ENSURE_REQUIREMENTS, [plan_id=238]

            +- HashAggregate(keys=[Vict Descent#286], functions=[partial_count(1)])

               +- Project [pythonUDF0#351 AS Vict Descent#286]

                  +- BatchEvalPython [descent_mapping(Vict Descent#30)#285], [pythonUDF0#351]

                     +- Project [Vict Descent#30]

                        +- CartesianProduct (cast(ZIPcode#184 as int) = Zip Code#113)

                           :- Project [Vict Descent#30, split(ZIPcode#92, ,, -1)[0] AS ZIPcode#184]

                           :  +- CartesianProduct ((knownfloatingpointnormalized(normalizenanandzero(LAT#43)) = knownfloatingpointnormalized(normalizenanandzero(LAT#90))) AND
(knownfloatingpointnormalized(normalizenanandzero(LON#44)) = knownfloatingpointnormalized(normalizenanandzero(LON#91))))

                           :     :- Project [Vict Descent#30, LAT#43, LON#44]

                           :     :  +- Filter ((year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType, Some(Europe/Athens), false) as date)) = 2015) AND isnotnull(Vict Descent#30))

                           :     :     +- FileScan csv [DATE OCC#19,Vict Descent#30,LAT#43,LON#44] Batched: false, DataFilters: [(year(cast(gettimestamp(DATE OCC#19, MM/dd/yyyy hh:mm:ss a, TimestampType,
Some(Europe/Athens), ..., Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Vict Descent)],
ReadSchema: struct<DATE OCC:string,Vict Descent:string,LAT:double,LON:double>

                           :     +- Filter ((isnotnull(LAT#90) AND isnotnull(LON#91)) AND isnotnull(split(ZIPcode#92, ,, -1)[0]))

                           :        +- FileScan csv [LAT#90,LON#91,ZIPcode#92] Batched: false, DataFilters: [isnotnull(LAT#90), isnotnull(LON#91), isnotnull(split(ZIPcode#92, ,, -1)[0])], Format: CSV, Location:
InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/revgecoding.csv], PartitionFilters: [], PushedFilters: [IsNotNull(LAT), IsNotNull(LON)], ReadSchema: struct<LAT:double,LON:double,ZIPcode:string>

                           +- Union

                              :- Filter isnotnull(Zip Code#113)

                              :  +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 DESC NULLS LAST], output=[Zip Code#113])

                              :     +- Project [Zip Code#113, cast(regexp_replace(Estimated Median Income#115, [\$,], , 1) as float) AS Estimated Median Income#119]

                              :        +- FileScan csv [Zip Code#113,Estimated Median Income#115] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1
paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>

                              +- Filter isnotnull(Zip Code#214)

                                 +- TakeOrderedAndProject(limit=3, orderBy=[Estimated Median Income#119 ASC NULLS FIRST], output=[Zip Code#214])

                                    +- Project [Zip Code#214, cast(regexp_replace(Estimated Median Income#216, [\$,], , 1) as float) AS Estimated Median Income#119]

                                       +- FileScan csv [Zip Code#214,Estimated Median Income#216] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1
paths)[hdfs://master:54310/datasets/income/LA_income_2015.csv], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<Zip Code:int,Estimated Median Income:string>
```

Q41b DataFrame Broadcast



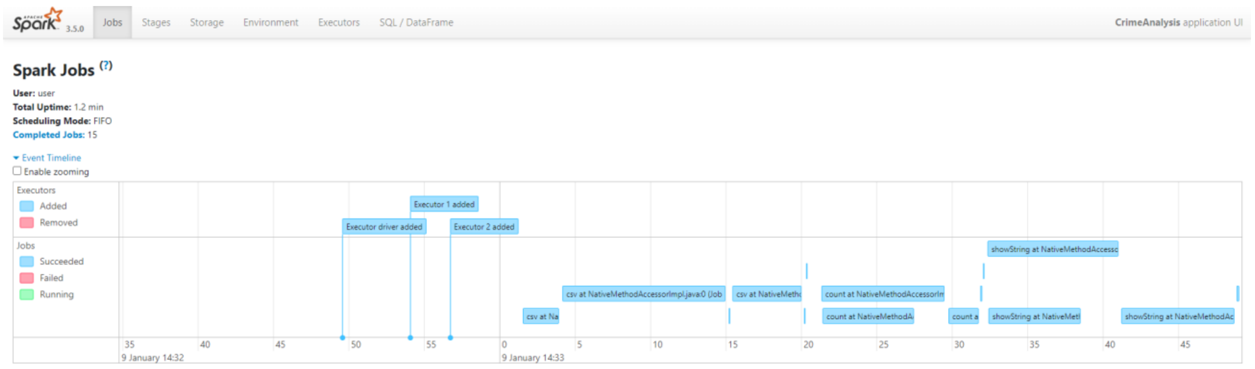
Spark Jobs (?)

User: user
Total Uptime: 1.1 min
Scheduling Mode: FIFO
Completed Jobs: 13

Physical Plan from Spark Environment q41bdfBroadcast:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [Count#420L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(Count#420L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=172]

      +- HashAggregate(keys=[AREA NAME#22], functions=[avg(Distance#336), count(DR_NO#17)])

         +- Exchange hashpartitioning(AREA NAME#22, 200), ENSURE_REQUIREMENTS, [plan_id=169]

            +- HashAggregate(keys=[AREA NAME#22], functions=[partial_avg(Distance#336), partial_count(DR_NO#17)])

               +- Project [DR_NO#17, AREA NAME#22, pythonUDF0#433 AS Distance#336]

                  +- BatchEvalPython [haversine(LAT#43, LON#44, Y#164, X#163)#335], [pythonUDF0#433]

                     +- Project [DR_NO#17, AREA NAME#22, LAT#43, LON#44, X#163, Y#164]

                        +- BroadcastHashJoin [upper(AREA NAME#22)], [DIVISION#166], LeftOuter, BuildRight, false

                           :- Union

                           :  :- Project [DR_NO#17, AREA NAME#22, LAT#43, LON#44]

                           :  :  +- Filter isnotnull(Weapon Used Cd#33)

                           :  :     +- FileScan csv [DR_NO#17,AREA NAME#22,Weapon Used Cd#33,LAT#43,LON#44] Batched: false, DataFilters: [isnotnull(Weapon Used Cd#33)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Weapon Used Cd)], ReadSchema: struct<DR_NO:int,AREA NAME:string,Weapon Used Cd:int,LAT:double,LON:double>

                           :  +- Project [DR_NO#90, AREA NAME#95, LAT#116, LON#117]

                           :     +- Filter isnotnull(Weapon Used Cd#106)

                           :        +- FileScan csv [DR_NO#90,AREA NAME#95,Weapon Used Cd#106,LAT#116,LON#117] Batched: false, DataFilters: [isnotnull(Weapon Used Cd#106)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Weapon Used Cd)], ReadSchema: struct<DR_NO:int,AREA NAME:string,Weapon Used Cd:int,LAT:double,LON:double>

                           +- BroadcastExchange HashedRelationBroadcastMode(List(input[2, string, false]),false), [plan_id=162]

                              +- Filter isnotnull(DIVISION#166)

                                 +- FileScan csv [X#163,Y#164,DIVISION#166] Batched: false, DataFilters: [isnotnull(DIVISION#166)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/LAPD_Police_Stations.csv], PartitionFilters: [], PushedFilters: [IsNotNull(DIVISION)], ReadSchema: struct<X:double,Y:double,DIVISION:string>
```

Q41b DataFrame Merge



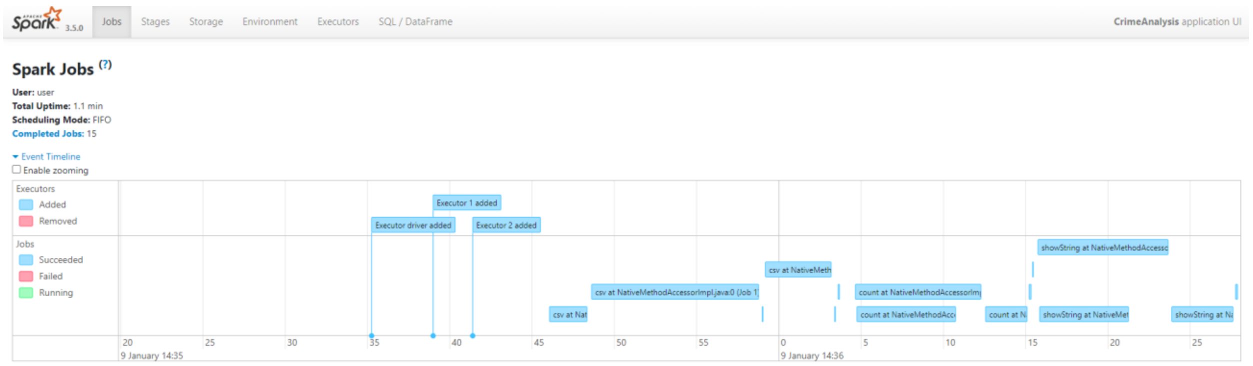
Spark Jobs (?)

User: user
Total Uptime: 1.2 min
Scheduling Mode: FIFO
Completed Jobs: 15

Physical Plan from Spark Environment q41bdfMerge:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [Count#418L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(Count#418L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=133]

      +- HashAggregate(keys=[AREA NAME#22], functions=[avg(Distance#334), count(DR_NO#17)])

         +- Exchange hashpartitioning(AREA NAME#22, 200), ENSURE_REQUIREMENTS, [plan_id=130]

            +- HashAggregate(keys=[AREA NAME#22], functions=[partial_avg(Distance#334), partial_count(DR_NO#17)])

               +- Project [DR_NO#17, AREA NAME#22, pythonUDF0#431 AS Distance#334]

                  +- BatchEvalPython [haversine(LAT#43, LON#44, Y#164, X#163)#333], [pythonUDF0#431]

                     +- Project [DR_NO#17, AREA NAME#22, LAT#43, LON#44, X#163, Y#164]

                        +- SortMergeJoin [upper(AREA NAME#22)], [DIVISION#166], LeftOuter

                           :- Sort [upper(AREA NAME#22) ASC NULLS FIRST], false, 0

                           :  +- Exchange hashpartitioning(upper(AREA NAME#22), 200), ENSURE_REQUIREMENTS, [plan_id=120]

                           :     +- Union

                           :        :- Project [DR_NO#17, AREA NAME#22, LAT#43, LON#44]

                           :        :  +- Filter isnotnull(Weapon Used Cd#33)

                           :        :     +- FileScan csv [DR_NO#17,AREA NAME#22,Weapon Used Cd#33,LAT#43,LON#44] Batched: false, DataFilters: [isnotnull(Weapon Used Cd#33)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Weapon Used Cd)], ReadSchema: struct<DR_NO:int,AREA NAME:string,Weapon Used Cd:int,LAT:double,LON:double>

                           :        +- Project [DR_NO#90, AREA NAME#95, LAT#116, LON#117]

                           :           +- Filter isnotnull(Weapon Used Cd#106)

                           :              +- FileScan csv [DR_NO#90,AREA NAME#95,Weapon Used Cd#106,LAT#116,LON#117] Batched: false, DataFilters: [isnotnull(Weapon Used Cd#106)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Weapon Used Cd)], ReadSchema: struct<DR_NO:int,AREA NAME:string,Weapon Used Cd:int,LAT:double,LON:double>

                           +- Sort [DIVISION#166 ASC NULLS FIRST], false, 0

                              +- Exchange hashpartitioning(DIVISION#166, 200), ENSURE_REQUIREMENTS, [plan_id=121]

                                 +- Filter isnotnull(DIVISION#166)

                                    +- FileScan csv [X#163,Y#164,DIVISION#166] Batched: false, DataFilters: [isnotnull(DIVISION#166)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/LAPD_Police_Stations.csv], PartitionFilters: [], PushedFilters: [IsNotNull(DIVISION)], ReadSchema: struct<X:double,Y:double,DIVISION:string>
```

Q41b DataFrame Shuffle Hash



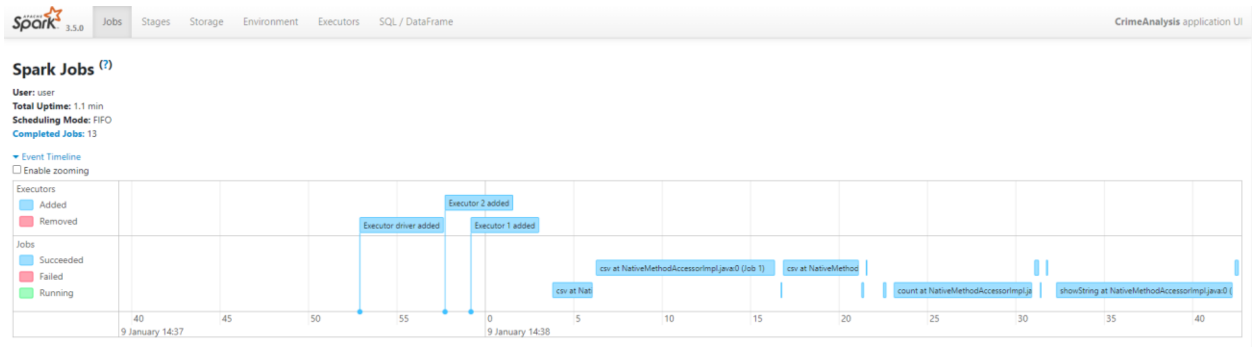
Spark Jobs (?)

User: user
Total Uptime: 1.1 min
Scheduling Mode: FIFO
Completed Jobs: 15

Physical Plan from Spark Environment q41bdfShuffleHash

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [Count#418L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(Count#418L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=131]

      +- HashAggregate(keys=[AREA NAME#22], functions=[avg(Distance#334), count(DR_NO#17)])

         +- Exchange hashpartitioning(AREA NAME#22, 200), ENSURE_REQUIREMENTS, [plan_id=128]

            +- HashAggregate(keys=[AREA NAME#22], functions=[partial_avg(Distance#334), partial_count(DR_NO#17)])

               +- Project [DR_NO#17, AREA NAME#22, pythonUDF0#431 AS Distance#334]

                  +- BatchEvalPython [haversine(LAT#43, LON#44, Y#164, X#163)#333], [pythonUDF0#431]

                     +- Project [DR_NO#17, AREA NAME#22, LAT#43, LON#44, X#163, Y#164]

                        +- ShuffledHashJoin [upper(AREA NAME#22)], [DIVISION#166], LeftOuter, BuildLeft

                           :- Exchange hashpartitioning(upper(AREA NAME#22), 200), ENSURE_REQUIREMENTS, [plan_id=120]

                           :  +- Union

                           :     :- Project [DR_NO#17, AREA NAME#22, LAT#43, LON#44]

                           :     :  +- Filter isnotnull(Weapon Used Cd#33)

                           :     :     +- FileScan csv [DR_NO#17,AREA NAME#22,Weapon Used Cd#33,LAT#43,LON#44] Batched: false, DataFilters: [isnotnull(Weapon Used Cd#33)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Weapon Used Cd)], ReadSchema: struct<DR_NO:int,AREA NAME:string,Weapon Used Cd:int,LAT:double,LON:double>

                           :     +- Project [DR_NO#90, AREA NAME#95, LAT#116, LON#117]

                           :        +- Filter isnotnull(Weapon Used Cd#106)

                           :           +- FileScan csv [DR_NO#90,AREA NAME#95,Weapon Used Cd#106,LAT#116,LON#117] Batched: false, DataFilters: [isnotnull(Weapon Used Cd#106)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Weapon Used Cd)], ReadSchema: struct<DR_NO:int,AREA NAME:string,Weapon Used Cd:int,LAT:double,LON:double>

                           +- Exchange hashpartitioning(DIVISION#166, 200), ENSURE_REQUIREMENTS, [plan_id=121]

                              +- Filter isnotnull(DIVISION#166)

                                 +- FileScan csv [X#163,Y#164,DIVISION#166] Batched: false, DataFilters: [isnotnull(DIVISION#166)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/LAPD_Police_Stations.csv], PartitionFilters: [], PushedFilters: [IsNotNull(DIVISION)], ReadSchema: struct<X:double,Y:double,DIVISION:string>
```

Q41b DataFrame Shuffle Replicate Nl



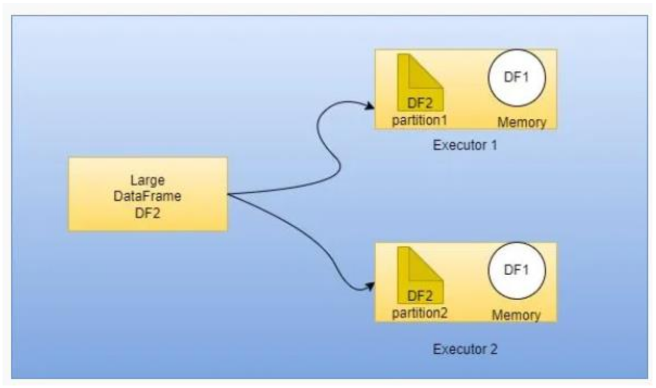
Spark Jobs (?)

User: user
Total Uptime: 1.1 min
Scheduling Mode: FIFO
Completed Jobs: 13

Physical Plan from Spark Environment q41bdfShuffleRep:

```
== Physical Plan ==

AdaptiveSparkPlan isFinalPlan=false

+- Sort [Count#418L DESC NULLS LAST], true, 0

   +- Exchange rangepartitioning(Count#418L DESC NULLS LAST, 200), ENSURE_REQUIREMENTS, [plan_id=129]

      +- HashAggregate(keys=[AREA NAME#22], functions=[avg(Distance#334), count(DR_NO#17)])

         +- Exchange hashpartitioning(AREA NAME#22, 200), ENSURE_REQUIREMENTS, [plan_id=126]

            +- HashAggregate(keys=[AREA NAME#22], functions=[partial_avg(Distance#334), partial_count(DR_NO#17)])

               +- Project [DR_NO#17, AREA NAME#22, pythonUDF0#431 AS Distance#334]

                  +- BatchEvalPython [haversine(LAT#43, LON#44, Y#164, X#163)#333], [pythonUDF0#431]

                     +- Project [DR_NO#17, AREA NAME#22, LAT#43, LON#44, X#163, Y#164]

                        +- BroadcastHashJoin [upper(AREA NAME#22)], [DIVISION#166], LeftOuter, BuildRight, false

                           :- Union

                           :  :- Project [DR_NO#17, AREA NAME#22, LAT#43, LON#44]

                           :  :  +- Filter isnotnull(Weapon Used Cd#33)

                           :  :     +- FileScan csv [DR_NO#17,AREA NAME#22,Weapon Used Cd#33,LAT#43,LON#44] Batched: false, DataFilters: [isnotnull(Weapon Used Cd#33)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2010_to_2019.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Weapon Used Cd)], ReadSchema: struct<DR_NO:int,AREA NAME:string,Weapon Used Cd:int,LAT:double,LON:double>

                           :  +- Project [DR_NO#90, AREA NAME#95, LAT#116, LON#117]

                           :     +- Filter isnotnull(Weapon Used Cd#106)

                           :        +- FileScan csv [DR_NO#90,AREA NAME#95,Weapon Used Cd#106,LAT#116,LON#117] Batched: false, DataFilters: [isnotnull(Weapon Used Cd#106)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/Crime_Data_from_2020_to_Present.csv], PartitionFilters: [], PushedFilters: [IsNotNull(Weapon Used Cd)], ReadSchema: struct<DR_NO:int,AREA NAME:string,Weapon Used Cd:int,LAT:double,LON:double>

                           +- BroadcastExchange HashedRelationBroadcastMode(List(input[2, string, false]),false), [plan_id=119]

                              +- Filter isnotnull(DIVISION#166)

                                 +- FileScan csv [X#163,Y#164,DIVISION#166] Batched: false, DataFilters: [isnotnull(DIVISION#166)], Format: CSV, Location: InMemoryFileIndex(1 paths)[hdfs://master:54310/datasets/LAPD_Police_Stations.csv], PartitionFilters: [], PushedFilters: [IsNotNull(DIVISION)], ReadSchema: struct<X:double,Y:double,DIVISION:string>
```

Broadcast Join: Αυτή η μέθοδος είναι ιδανική όταν ένα από τα σύνολα δεδομένων είναι πολύ μικρότερο από το άλλο. Το μικρότερο σύνολο δεδομένων μπορεί να χωρέσει στη μνήμη κάθε κόμβου. Ελαχιστοποιεί την ανακατανομή δεδομένων στο δίκτυο, επειδή το μικρότερο σύνολο δεδομένων μεταδίδεται σε όλους τους κόμβους. Αυτό οδηγεί σε σημαντική βελτίωση των επιδόσεων, ειδικά για μεγάλα σύνολα δεδομένων.



Broadcast Join[2]

Merge Join: Η μέθοδος Join είναι καλή για σύνολα δεδομένων που είναι πολύ μεγάλα για να μεταδοθούν. Ταξινομεί τα σύνολα δεδομένων με βάση τα κλειδιά σύνδεσης και στη συνέχεια εκτελεί τη συγχώνευση. Περιλαμβάνει την ανακατανομή των δεδομένων στο δίκτυο, η οποία μπορεί να είναι δαπανηρή.

Shuffle Hash Join: Χρήσιμη μέθοδος όταν και τα δύο σύνολα δεδομένων είναι μεγάλα αλλά εξακολουθούν να είναι αρκετά μικρά ώστε να χωράνε στη μνήμη των worker nodes όταν κατατμηθούν. Κατακερματίζει (hash tables) τα σύνολα δεδομένων και τα ανακατεύει στους κόμβους.

Shuffle and Replicate Nested Loop Join (Shuffle Replicate NL): Αυτή η μέθοδος ανακατεύει το ένα σύνολο δεδομένων και αναπαράγει το άλλο για κάθε διαχωριστικό. Είναι γενικά η λιγότερο αποδοτική στρατηγική σύνδεσης και χρησιμοποιείται μόνο για συγκεκριμένες περιπτώσεις όπου άλλες συνδέσεις δεν είναι εφαρμόσιμες.

Physical Plan: Δέντρο που περιέχει πιο συγκεκριμένη περιγραφή του τρόπου με τον οποίο πρέπει να εκτελεστούν τα εσωτερικά jobs για κάθε query.

Συμπεράσματα:

- Query q3df.py

Παρατηρούμε από τα αποτελέσματα που λάβαμε ότι η Broadcast Join, Merge Join και Shuffle Hash είναι πιο αποδοτικές με πολύ κοντινούς χρόνους εκτέλεσης για 2,3,4 Spark executors αντίστοιχα (53-57 δευτερόλεπτα). Η Shuffle Replicate NL απέχει πολύ από θέμα αποδοτικότητας λόγω του δεκαπλάσιου και πάνω χρόνου εκτέλεσης (11-14 λεπτά).

Περιμέναμε την καλύτερη απόδοση από την Broadcast και Merge join (επαληθεύεται) ενδιάμεση την Shuffle Hash (εφάμιλλη με τις προηγούμενες) και την Shuffle Replicate NL αποτελεί την λιγότερο κατάλληλη μέθοδο join καθώς απαιτεί την χρήση πολλών πόρων (επαληθεύεται). Αυτό συμβαίνει λόγω του μεγέθους των συγκεκριμένων αρχείων που έχουμε να επεξεργαστούμε στο q3df.py.

- Query q41bdf.py

Οι χρόνοι από τις Broadcast Hash Join, Shuffle Hash και Shuffle Replicate NL (1.1 λεπτά) είναι σχεδόν εφάμιλλοι. Η Merge Join είναι ελαφρώς χειρότερη απο τις υπόλοιπες join μεθόδους (1.2 λεπτά). Πρακτικά είναι σχεδόν εφάμιλλες λόγω των συγκεκριμένων σημείων που πραγματοποιούνται τα joins και των συνόλων δεδομένων που επεξεργαζόμαστε στο q41bdf.py.

Βιβλιογραφία

[1] The Databricks Data Intelligence Platform, Calalyst Optimizer, https://www.databricks.com/glossary/catalyst-optimizer

[2] Broadcast Join in Spark, https://sparkbyexamples.com/spark/broadcast-join-in-spark/