


## Analysis and Evaluation on NY parking Tickets

The City of New York Notice of Parking Violation					
<p>YOU MUST ANSWER WITHIN 30 DAYS OF THE DATE OF THIS TICKET. IF YOU DO NOT RESPOND, PENALTIES AND INTEREST WILL BE ADDED AND YOUR VEHICLE MAY BE BOOTED OR TOWED.</p> <p style="text-align: right; font-size: small;">N/S/Not Shown N/A/Not Applicable</p>					
Permit Displayed	Permit Number		Type		
N/S	N/A		N/A		
Name of the Operator, if present. If not present: OWNER OF THE VEHICLE BEARING LICENSE					
Plate	CD	Exp. Date	State	Plate Type	
	NS	N/S	OR	PAS	
Make	Color	Year		Body Type	
KIA	BLK	N/S		SUBN	
VIN #					
THE OPERATOR AND OWNER OF THE ABOVE VEHICLE ARE CHARGED AS FOLLOWS:					
In Violation of NYC Traffic Rules, Section: 4-08(h)(10)(ii)					
Failure to Dsply Muni Rec					
DAYS/HRS: EXCEPT Su/ 8 A- 7 P					
Place of Occurrence					
Front of 7613 3rd Ave					
VC	Meter #	Operational	Limit	County	Pct.
38				K	070
Date/Time of Offense			Date/Time 1st Observed		
08/21/17 01:59 PM			N/A		
Complainant's Comments:					
<b>FINE AMOUNT: \$ 35.00</b>					
Agency	Command		Tax Reg #		
TRAFFIC	T-302		363497		
Complainant's Name					


  
**08480**

Gruppo 'The Organizer Team': Vanessa Menta 499608, Corrado Silvestri 466191

Sommario	
Analysis and Evaluation on NY parking Tickets .....	1
1.Introduzione .....	3
2.Architettura .....	4
3.NYC ticket, Dataset Analysis .....	6
4.Apache Cassandra .....	8
5.Batch Data Processing .....	10
5.1Spark Sql .....	10
5.2 Apache Mllib .....	10
5.2.1 Classificazione .....	10
6.Risultati Ottenuti .....	13
6.1 Risultati .....	13
6.1.1 Provenienza dei veicoli .....	13
6.1.2 tipologia di vettura .....	13
6.1.3 Infrazioni più comuni .....	14
6.1.4 Circoscrizioni più multate .....	14
6.1.5 Le targhe più multate .....	15
6.1.6 Case automobilistiche .....	15
6.1.7 Multe per ogni anno .....	16
6.1.8 Mesi con più multe .....	16
6.2 Risultati Classificazione .....	17
7. Conclusioni e Sviluppi Futuri .....	18

## 1.Introduzione

Questo progetto prende in esame il dataset disponibile su Kaggle: <https://www.kaggle.com/new-york-city/nyc-parking-tickets>, relativo alle multe collezionate dal dipartimento di finanza di NYC.

Su Kaggle sono disponibili 4 diversi dataset riguardanti il periodo temporale da Agosto 2013 a Giugno 2017.

L'obiettivo della nostra analisi è effettuare delle operazioni di tipo puramente "Batch", in modo da fornire un supporto alle decisioni inerenti ai parcheggi. A tale scopo, verranno individuate le zone della città maggiormente a rischio multa, i mesi in cui si registrano più multe, ma anche gli automobilisti che solitamente tendono ad essere meno attenti a dove lasciano la propria autovettura. Infatti, verranno classificati in base alla casa automobilistica, allo stato di provenienza della targa e la tipologia di vettura.

I risultati ottenuti tramite le queries implementate con Apache Spark verranno salvati sul database No-SQL Cassandra in modo da massimizzare la disponibilità dei dati.

Infine, tramite Apache MLlib verranno effettuate delle analisi di Machine Learning sui dati per fare delle previsioni future, in particolare, si vorrà prevedere la provenienza degli automobilisti che saranno multati.

La relazione è ordinata secondo il seguente schema logico. Nel capitolo 2 viene fatta una panoramica sull'architettura utilizzata per questo progetto, specificando i motivi delle scelte effettuate e su quale calcolatore il progetto è stato realizzato.

Nel capitolo 3 viene presentato il dataset NYC Tickets soffermandoci sulla fase di analisi e sulla pulizia dei dati effettuata. Il capitolo 4 contiene una panoramica sul Database scelto, Apache Cassandra, descrivendone l'implementazione e le qualità raggiunte.

La tecnologia di Batch Data Processing utilizzate nel progetto sono descritte nel capitolo 5, dove è presente una panoramica su Spark e Spark MLlib e come sono state utilizzate nel progetto.

Infine presentiamo nel capitolo 6 i risultati ottenuti e le relative considerazioni, mentre nel capitolo 7 le conclusioni e le osservazioni finali.

## 2.Architettura

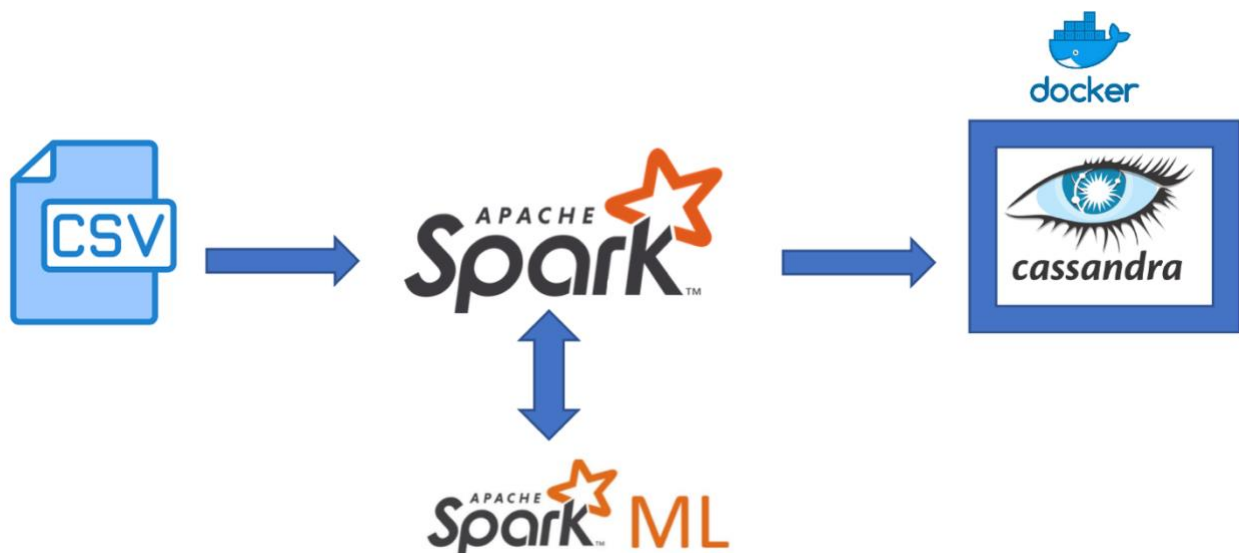
Per il nostro progetto di Batch Analysis sul dataset delle multe della città di New York, la scelta è ricaduta sull'utilizzo del framework Apache Spark. Le sue funzioni saranno quelle di estrarre i dati presenti nel dataset e creare un dataframe Spark, su cui sarà possibile effettuare interrogazioni per comprendere i trend delle multe nel periodo Agosto '13- Giugno '2017 in fase di Batch Data Processing.

Successivamente, le relazioni estratte saranno memorizzate e il dataset sarà ripulito per poter effettuare delle previsioni con algoritmi di Machine Learning.

Per la memorizzazione si è deciso di optare per il database NoSql 'Apache Cassandra' che viene configurato e attivato sul nodo di computazione prima di iniziare la fase di analisi.

Per la parte di Machine Learning effettueremo una classificazione grazie al modello incluso nella libreria di Apache Spark MLLib Logistic Regression.

Nell'immagine sottostante una panoramica dell'architettura utilizzata:



Si è voluto includere il Database NoSql Apache Cassandra in un container docker. Questa scelta è utile per aumentare la Partition Tolerance, ovvero che il sistema continua ad operare anche in caso di partizionamento della rete. Inoltre la possibilità di creare facilmente un nodo del cluster Cassandra all'interno del contenitore Docker permette di aumentare la disponibilità dei dati offerta dal sistema.

I dati saranno salvati all'interno del database dopo essere stati manipolati da Spark per estrarre informazioni relative ai trend. La struttura Column Family di Cassandra è stata inoltre un altro motivo che ha influenzato la scelta del db data la tipologia dei dati.

Infine, utilizzando la libreria scalabile di algoritmi di Machine Learning vengono effettuate delle previsioni future. In particolare, viene utilizzata la tecnica di classificazione della Logistic Regression che ci consentirà di effettuare delle previsioni grazie al processo di apprendimento induttivo: attraverso lo studio degli innumerevoli esempi forniti nella parte di dataset utilizzata come training set, il modello di Logistic Regression

verrà addestrato e successivamente, sarà in grado di effettuare previsioni accurate per una certa variabile target quando gli saranno presentati degli esempi mai visti prima, ovvero l'insieme di test set.

In conclusione, l'intero sistema è stato realizzato su un calcolatore avente le seguenti specifiche:

- 1) Hardware
  - a) Processore Intel i7-9700k da 8 core ,3.6 GHz, 12 Mb Cache L3
  - b) 32 Gb di RAM DDR4 3000Mhz
  - c) 500 Gb hard disk M2
- 2) Software
  - a) Ubuntu LTS 20.04
  - b) Pyspark 3.0
  - c) Jupyter
  - d) Docker 19.03.8
  - e) Cassandra 3.11

Il sistema è stato configurato partendo prima dalla creazione del contenitore docker con dentro 4 nodi di cassandra per la creazione del cluster. Successivamente, si è provveduto a lavorare su un Jupiter file tramite PySpark, effettuando la memorizzazione in Cassandra per ogni risultato delle queries e utilizzando Spark MLlib per la Logistic Regression.

### 3.NYC ticket, Dataset Analysis

Il dataset è relativo alle multe di New York per il periodo temporale Agosto '13 -Giugno '17 reperito sul sito di Kaggle al seguente link: ' <https://www.kaggle.com/new-york-city/nyc-parking-tickets> ' .

Il dataset si presentava nella forma di 4 file csv separati per anno contenenti circa 10 milioni di righe per file. Le colonne si presentano in questa forma:

Summons Number: Number	Law Section: Number
Plate ID: Plain Text	Sub Division: Plain Text
Registration State: Plain Text	Violation Legal Code: Plain Text
Plate Type: Plain Text	Days Parking In Effect: Plain Text
Issue Date: Date & Time	From Hours In Effect: Plain Text
Violation Code: Number	To Hours In Effect: Plain Text
Vehicle Body Type: Plain Text	Vehicle Color: Plain Text
Vehicle Make: Plain Text	Unregistered Vehicle?: Plain Text
Issuing Agency: Plain Text	Vehicle Year: Number
Street Code1: Number	Meter Number: Plain Text
Street Code2: Number	Feet From Curb: Number
Street Code3: Number	Violation Post Code: Plain Text
Vehicle Expiration Date: Number	Violation Description: Plain Text
Violation Location: Plain Text	No Standing or Stopping Violation: Plain Text
Violation Precinct: Number	Hydrant Violation: Plain Text
Issuer Precinct: Number	Double Parking Violation: Plain Text
Issuer Code: Number	Latitude: Number *
Issuer Command: Plain Text	Longitude: Number *
Issuer Squad: Plain Text	Community Board: Number *
Violation Time: Plain Text	Community Council: Number *
Time First Observed: Plain Text	Census Tract: Number *
Violation County: Plain Text	BIN: Number *
Violation In Front Of Or Opposite: Plain Text	BBL: Number *
House Number: Plain Text	NTA: Plain Text *
Street Name: Plain Text	
Intersecting Street: Plain Text	
Date First Observed: Number	
	* Colonne non presenti nel dataset 2017

Durante una prima fase di analisi del dataset dedicata a studiarne i campi e decidere quali tipo di trend e/o previsioni realizzare sono saltate all'occhio alcune problematiche:

- Era presente una differenza nel numero di colonne, 43, nel dataset relativo al 2017, rispetto alle 51 colonne delle altre annate.
- Erano presenti date fuori il range di analisi oppure date future che rendevano impossibile l'esistenza di quel record.
- Alcune colonne presentavano un'enorme numero di valori nulli rispetto alla grandezza del dataset, rendendo impossibile la loro analisi.

Si è deciso di effettuare uno step preventivo di pulizia tramite il file python 'first\_dataset\_filter.py' presente nel repository di questo progetto, in cui viene effettuata l'unione dei dataset e vengono eliminate le 8 colonne non presenti nel dataset del 2017.

Inoltre, prima di generare un dataset unico per il periodo 13-17, viene gestito il campo 'Issue Date' presente originariamente nella forma MM/DD/YYYY che viene trasformato nel formato YYYY/MM/DD che si presta meglio per l'esecuzione delle interrogazioni. Vengono inoltre filtrati gli anni non appartenenti all'intervallo temporale 13-17.

Una volta terminata l'esecuzione del file 'first\_dataset\_filter.py' viene generato un dataset unico contenente 42332687 di righe e 43 colonne.

Si è deciso invece di gestire la problematica delle colonne con valori nulli all'interno di Spark, per il forte orientamento alle colonne che presenta il framework.

Una volta che il dataset è stato caricato su Spark vengono effettuate ulteriori operazioni di pulizia ed adattamento delle colonne del dataframe per l'estrazione dei trend desiderati. In particolare:

1. Vengono mostrate per ogni colonna il conteggio dei valori nulli presenti. Analizzando i risultati prodotti vengono selezionate le colonne che hanno oltre l'80% di valori nulli rispetto al conteggio totale delle righe del dataset. Le seguenti colonne sono state dunque rimosse

"Time First Observed", "Intersecting Street", "Violation Legal Code", "Unregistered Vehicle?", "Meter Number", "No Standing or Stopping Violation", "Hydrant Violation", "Double Parking Violation", "Census Tract"
---

2. Viene riorganizzato il campo 'Issue Date', separando Year, Month e Day ed aggiungendola come colonna al dataframe
3. Viene ripulito il campo 'Plate ID' dai valori nulli che presenta affinché uno dei trend analizzati non presenti falsi risultati.

Successivamente dal dataset 'nyc\_tickets' sarà creata la vista sul dataframe 'nycTable' per effettuare le queries ed ottenere i risultati ed i trend desiderati e memorizzarli nel database Cassandra.

Infinite verrà manipolato ancora una volta il dataset 'nyc\_tickets' per generare un input adatto all'implementazione del modello Logistic Regression.

In una prima fase selezioniamo i campi d'interesse ritenuti rilevanti per effettuare la predizione. Vengono selezionati i campi 'Registration State', 'Day', 'Year', 'Month', 'Plate Type', 'Issuer Precinct', 'Violation Precinct', 'Issuing Agency' più utili ai fini della predizione. Abbiamo creato un nuovo dataframe 'ml\_nyc' su cui effettueremo i soliti lavori di ricerca dei nulli e conversione dei campi delle colonne.

Vengono convertite le colonne 'Issuer Precinct' e 'Violation Precinct' ad interi per facilitare l'utilizzo delle tecniche di machine learning e tramite Spark vengono eliminati i valori nulli presenti nel nuovo dataframe. Al termine di queste operazioni il dataframe 'ml\_nyc' in input al modello Logistic Regression avrà il seguente schema :

<pre> -- Registration State: string (nullable = true)  -- Day: integer (nullable = true)  -- Year: integer (nullable = true)  -- Month: integer (nullable = true)  -- Plate Type: string (nullable = true)  -- Issuer Precinct: integer (nullable = true)  -- Violation Precinct: integer (nullable = true)  -- Issuing Agency: string (nullable = true) ml_nyc.count()= 42323852</pre>
---

## 4. Apache Cassandra

Per la memorizzazione dei risultati delle queries sul dataset abbiamo scelto di utilizzare il Database No-SQL Cassandra. In tale database Aggregate Oriented di tipo Column-Family, l'unità di memorizzazione fondamentale che definisce come interagire sui dati è la colonna. Inoltre, la struttura di Cassandra permette una doppia organizzazione dei dati sia per righe per colonne: infatti, i valori vengono raggruppati in famiglie di colonne e ogni colonna sarà una collezione (una famiglia) di coppie <nome, valore>. Data tale organizzazione del db lo abbiamo ritenuto la miglior scelta per svolgere la nostra analisi.

Cassandra è un database di tipo peer-to-peer, ovvero, in termini di replicazione, non vi è alcun nodo che svolge la funzione di "master" bensì tutte le repliche hanno lo stesso peso e tutti i nodi possono effettuare operazioni sia di lettura che di scrittura. Siccome lo stesso dato viene replicato su nodi multipli, la perdita di una delle repliche non preclude il funzionamento e l'accesso ai dati, infatti in questo modo tale database riesce ad avere una buona fault-tolerance.

Soprattutto, abbiamo scelto Cassandra per la nostra analisi per la necessità poter scalare il dataset e avere un'elevata disponibilità senza compromettere le performance. Infatti, Cassandra può essere classificato come un database di tipo AP secondo le proprietà del CAP teorema:

- Availability, ogni richiesta effettuata riceve una risposta.
- Partition tolerance, il sistema continua a funzionare nonostante partizionamenti della rete all'interno del cluster distribuito.

Come detto, volevamo poter sfruttare al meglio le qualità di availability e partition tolerance e per realizzare ciò abbiamo deciso di utilizzare Cassandra all'interno di un contenitore Docker così da poter gestire al meglio il numero e lo stato dei nodi creati per il cluster Cassandra e sfruttare le qualità offerte dall'esecuzione in contenitori che favorisce la scalabilità orizzontale del sistema.

Una volta scaricata l'immagine Docker di Apache Cassandra, abbiamo creato uno script 'docker-compose.yml' in cui vengono creati 4 container, ognuno contenente un nodo Cassandra attivo a cui viene assegnata una coppia <indirizzo\_ip, port> della rete p2p, successivamente clusterizzati. Inoltre viene effettuato il collegamento tra i database per la replicazione dei dati e viene creato il keyspace='mykeyspace' per il cluster. I nodi sono memorizzati in altrettante sottocartelle del sistema dove i dati saranno conservati.

Una volta effettuate le operazioni di analisi ed estrazione dei trend queste saranno memorizziamo i loro risultati su un nodo qualunque del cluster Cassandra. Infatti ogni qualvolta uno dei 4 nodi viene scritto, l'informazione sarà replicata ai restanti contenitori aventi all'interno i nodi di Cassandra, rendendola accessibile da qualsiasi delle 4 coppie <indirizzo ip, port> e garantendo disponibilità al nostro sistema. Inoltre in caso di crash o fallimento di uno dei contenitori, il sistema resterà comunque attivo senza presentare alcun disservizio o soffrire a causa di un partizionamento della rete.

Infine, la scelta di utilizzare docker per implementare Cassandra apporta innumerevoli qualità al nostro progetto. E' possibile infatti aumentare il numero di nodi presenti nel cluster di Cassandra facilmente, incrementando la scalabilità orizzontale del nostro sistema. Inoltre ogni applicazione con le relative dipendenze è isolata rispetto al sistema host.

Spark si connette al cluster Cassandra tramite la libreria cassandra.cluster di python specificando indirizzo IP del cluster, port e keyspace.

Una volta ottenuti i risultati delle query, i risultati verranno scritti in Cassandra secondo il seguente esempio:



```
session.execute("CREATE TABLE IF NOT EXISTS best_month (ind int primary key, month int, total_month int)")
stmt = session.prepare("INSERT INTO best_month (ind, month, total_month) VALUES (?, ?, ?)")

for ind, item in best_month.toPandas().iterrows():
    results = session.execute(stmt, [ind, item['month'], item['total_month']])
```

Viene creata la TABLE del risultato sul database, contenente i nomi e tipi delle colonne . Successivamente verrà iterato il risultato e creato uno statement in cui saranno inseriti i valori che dovranno essere memorizzati.

## 5.Batch Data Processing

### 5.1Spark Sql

Abbiamo effettuato la nostra Batch Analysis attraverso Spark SQL, ovvero il modulo di Apache Spark che integra la possibilità di esprimere calcoli in linguaggio relazionale con le API funzionali di Spark.

In esso, i dati vengono memorizzati secondo una tecnica chiamata Parquet, secondo la quale la memorizzazione avviene per colonne perché è proprio in base alle colonne che verranno effettuate le aggregazioni e non per righe. Infatti vengono utilizzati i Dataframe per collezionare i dati.

Grazie a questa struttura Parquet, Spark SQL risulta essere più efficiente e più veloce rispetto ad Hive.

Una volta letto il dataset 'nyc\_tickets\_13\_17' ed aver effettuato le operazioni di pulizia come elencato nel capitolo 3, tramite Spark Sql vengono eseguite query sul Dataframe contenuto del 'new\_york\_tickets' dopo aver effettuato la creazione della vista 'nycTable'.

Le query sono strutturate secondo questo esempio :

```
best_year=spark.sql("SELECT `Year` as year, count('Summons Number') as `total_year` FROM nycTable GROUP BY `Year` ORDER BY `total_year` DESC LIMIT 50")
best_year.show(5)
```

Per ogni query viene creato un nuovo dataframe che successivamente sarà memorizzato in Cassandra. Tutte le query stamperanno soltanto i primi 5 risultati in ordine decrescente tuttavia nel database vengono memorizzati i primi 50 risultati, per un eventuale analisi più approfondita. Il dataframe sarà scandito da Spark sql per effettuare la query d'inserimento nel database. Ad ogni query verrà aggiunto un campo 'id' intero crescente che sarà memorizzato nel database, così da poter facilmente indicizzare i risultati ottenuti. Infatti alle prime posizioni della classifica corrisponderanno id bassi.

### 5.2 Apache Mllib

Spark fornisce la sua libreria scalabile di algoritmi di Machine Learning Mllib la quale consente di effettuare importanti operazioni di Regression, Classification e Clustering.

Nella nostra analisi abbiamo utilizzato Apache Mllib col fine di effettuare una classificazione, ovvero di mappare lo spazio delle istanze con delle classi, cioè dei valori puramente discreti.

#### 5.2.1 Classificazione

Per effettuare la classificazione abbiamo implementato una Learning Pipeline in cui:

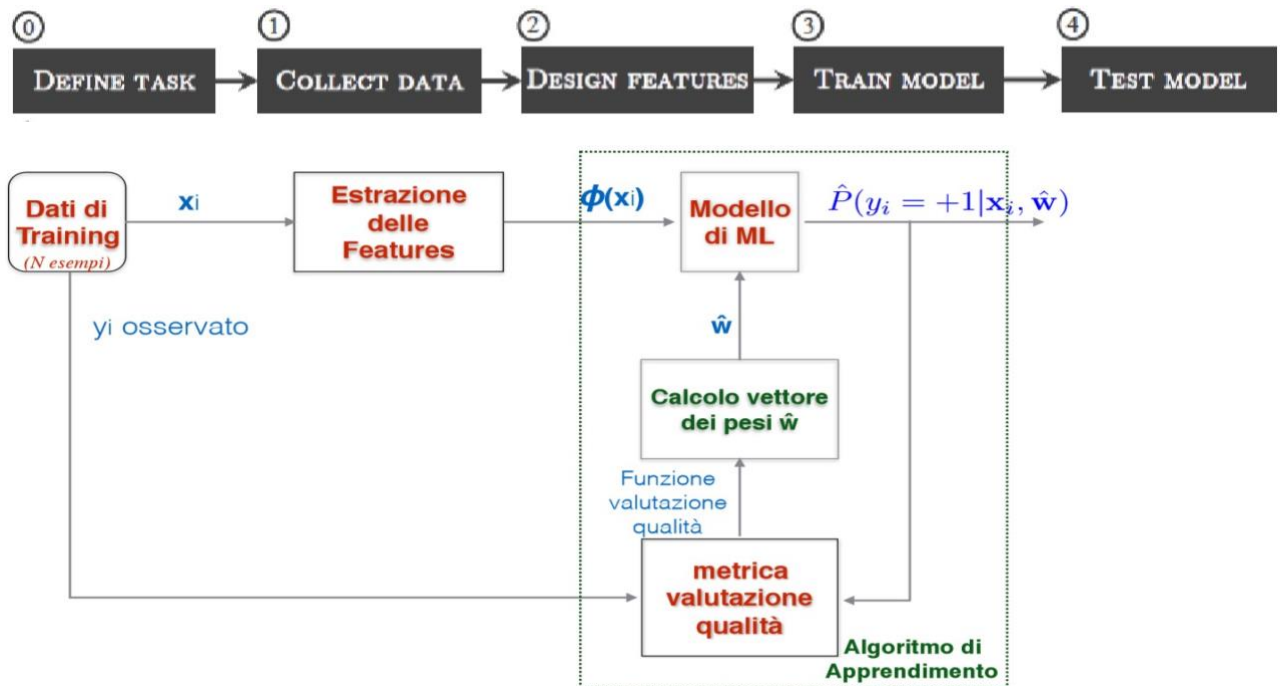
1. A partire dal dataset vengono individuate le features migliori per descrivere il modello, in particolare abbiamo scelto: la provenienza della targa, il distretto dove è stata effettuata la violazione, il distretto dell'automobilista multato, il giorno, il mese, l'anno, l'agenzia dell'assicurazione dell'automobilista, lo stato di provenienza dell'autovettura.
2. Il dataset viene suddiviso tra training set e test set con un rapporto di: train 70% e test 30%.

```
(train, test) = dataset.randomSplit([0.7, 0.3], seed=100)
```

3. Abbiamo scelto come modello per effettuare la classificazione la Logistic Regression, la quale dovrà "allenare" ovvero calibrare i parametri sul training set mediante dei metodi di ottimizzazione, ogni esempio preso in considerazione dal training set andrà a contribuire alla funzione che approssima il comportamento delle funzioni sconosciute in base a quelle già viste: dato un certo vettore dei pesi per ciascuno degli esempi di training disponibili, possiamo calcolare la probabilità di avere in uscita un certo valore. Una volta che il modello è stato addestrato sui dati di training, al variare delle features è in grado di comprendere il comportamento per qualunque target gli venga richiesta. In

particolare, nel nostro modello abbiamo deciso di prendere come variabile target il “Registration State”, ovvero la provenienza delle vetture degli automobilisti multati.

4. Sull’insieme di test vengono valutate le prestazioni del modello precedentemente addestrato. Si assegna infine un punteggio di accuratezza della previsione.



Prima di poter utilizzare le features selezionate, bisogna assicurarsi che siano tutte di tipo int. Le features che sono di tipo String infatti vengono infatti trasformate applicando:

- Lo StringIndexer, trasforma le colonne di String in colonne di indici;
- Il OneHotEncoder, crea delle nuove colonne che indicano la presenza (o l’assenza) per ogni valore possibile nella colonna di input, è come se creasse un array.

Infine viene utilizzato il VectorAssembler per combinare tutte le colonne delle features in un singolo vettore.

Le colonne che contengono stringhe devono essere trasformate in interi. Prima viene applicato lo String indexer e successivamente il OneHotEncoder

```
cat_col = ['Plate Type', 'Issuing Agency']
stages = []
for categoricalCol in cat_col:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + "Index")
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]
```

```
label_stringIdx = StringIndexer(inputCol='Registration State', outputCol="label")
stages += [label_stringIdx]
```

Creiamo il vettore con le features da considerare

```
assemblerInputs = [c + "classVec" for c in cat_col]
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

Le operazioni sopra descritte sono implementate nelle stages di una Pipeline, per rendere il calcolo più fluido.

Effettuiamo le operazioni elencate in pipeline:

```
half_Pipeline = Pipeline().setStages(stages)
```

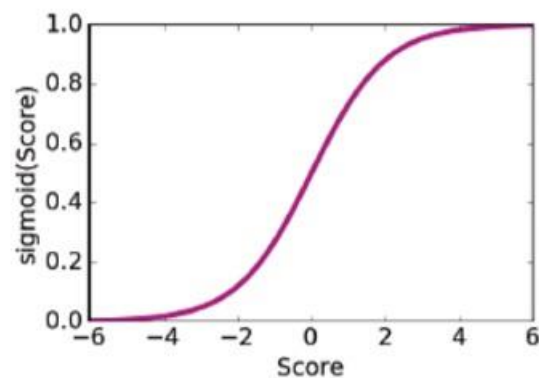
```
pipelineModel = half_Pipeline.fit(ml_nyc)
```

```
final_ml_nyc = pipelineModel.transform(ml_nyc)
```

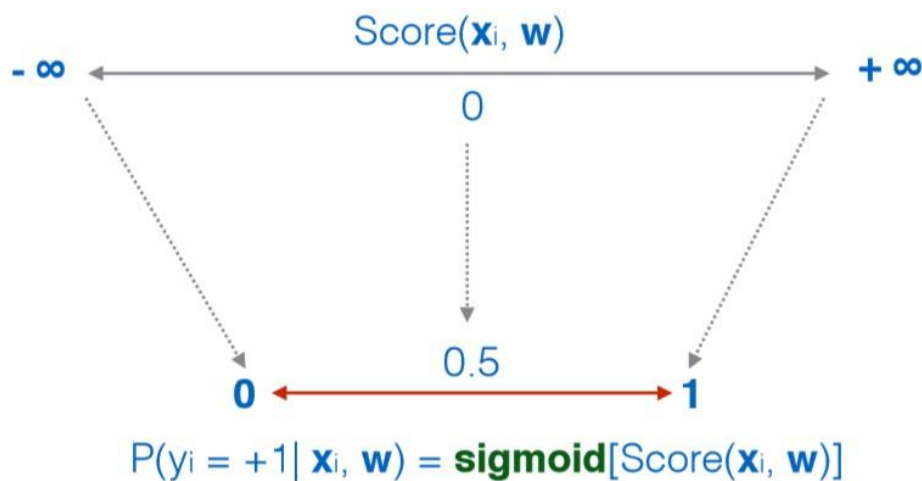
```
selection = ["label", "features"]  
dataset = final_ml_nyc.select(selection)
```

A questo punto le features selezionate per i dati di training possono essere allenate attraverso la Logistic Regression, la quale costituisce un mapping tra lo "score", ovvero il punteggio di rilevanza di ogni feature e una probabilità. Per trovare tale corrispondenza, la Logistic Regression si serve della funzione Sigmoid, la quale prendendo un qualunque input  $x$  definito sull'intervallo  $(-\infty, +\infty)$  restituisce un valore di probabilità dell'intervallo  $[0,1]$ . Tale funzione è definita come:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$



Il modello risultante di Logistic Regression sarà così definito:



```
logreg = LogisticRegression(labelCol="label", featuresCol="features", maxIter=10)  
logregModel = logreg.fit(train)
```

```
predictions = logregModel.transform(test)
```

Infine, si calcolano le prestazioni del modello utilizzato sia con una Binary Classification sia con una MultiClass Classification.

## 6. Risultati Ottenuti

In questo capitolo verranno mostrati quali sono state le queries effettuate sul dataframe e quali sono stati i trend estratti nel periodo Agosto'13-Giugno'17. Inoltre andremo ad analizzare i risultati di efficienza ed accuratezza calcolati dal modello Logistic Regression applicato. Come già riferito in precedenza, ogni query mostrerà soltanto 5 risultati ma nel database saranno memorizzate le prime 50 righe di ogni dataframe prodotto.

### 6.1 Risultati

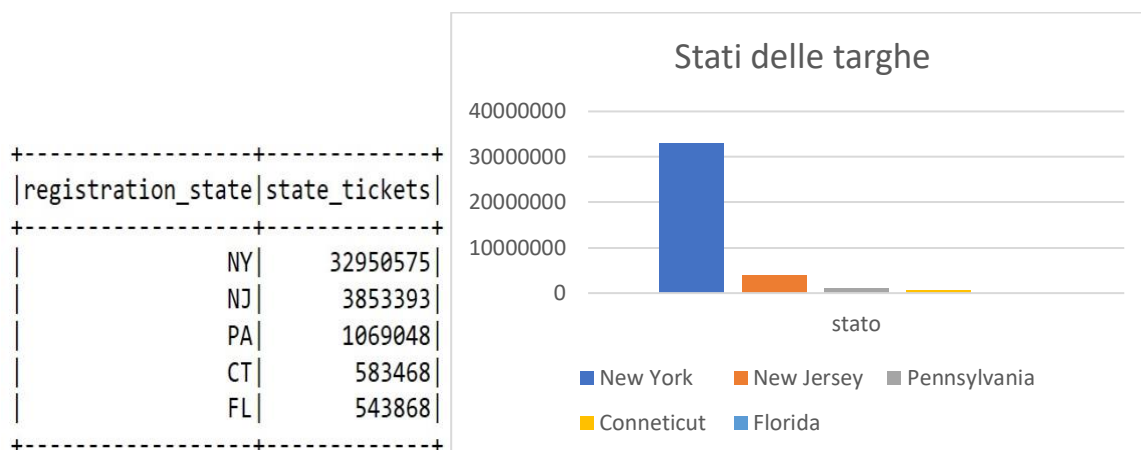
#### 6.1.1 Provenienza dei veicoli

La prima query eseguita si occupa di individuare gli stati di provenienza dei veicoli per cui si registrano più multe:

```
states_tickets=spark.sql("SELECT `Registration State` as registration_state,count('Registration State') as `state_tickets` FROM nycTable GROUP BY `registration_state` ORDER BY `state_tickets` DESC LIMIT 50")
states_tickets.show(5)
```

L'obiettivo è quello di ottenere una graduatoria dei veicoli multati in base al loro stato di appartenenza.

Naturalmente essendo il dataset relativo alla sola area metropolitana di New York, come mostrato nei risultati, oltre il 75% delle multe avviene su automobili provenienti dallo stato omonimo.



#### 6.1.2 tipologia di vettura

Successivamente, abbiamo ricercato quali sono i tipi di veicoli maggiormente multati. La descrizione del tipo di veicolo è inclusa nel campo 'Veichle Body Type' in forma di sigla e per ognuna di esse abbiamo cercato il risultato.

```
number_tickets_bt=spark.sql("SELECT `Vehicle Body Type` as vehicle_body_type, count(*) as `number_of_tickets_for_the_vehicle_body_type` FROM nycTable GROUP BY `vehicle_body_type` ORDER BY `number_of_tickets_for_the_vehicle_body_type` DESC LIMIT 50")
number_tickets_bt.show(5)
```

Vengono riportati i significati delle sigle del risultato:

SUBN-> Veicoli Suburban

VAN->Furgoni

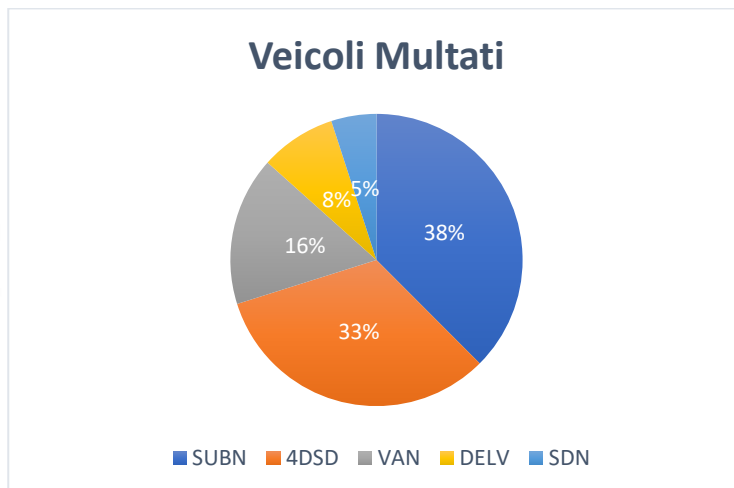
SDN->Berlina Classica

4DSD-> Berlina 4 Porte

DELV->Delivery Truck

Si evidenzia come oltre il 60% dei veicoli multati siano appartenenti soltanto alle categorie Suburban e Berlina 4 porte.

vehicle_body_type	number_of_tickets_for_the_vehicle_body_type
SUBN	13708699
4DSD	11922806
VAN	6022624
DELV	3065094
SDN	1826268



#### 6.1.3 Infrazioni più comuni

Abbiamo analizzato inoltre il tipo di infrazioni commesse quando i veicoli vengono multati, mirata ad analizzare quali siano le violazioni più gettonate dagli automobilisti. Anche per questo risultato è presente un codice identificativo della violazione. Per dare un significato ai dati estratti abbiamo ricercato i seguenti codici assegnando il tipo di infrazione commessa ed il relativo prezzo:

- 21 = "street cleaning", se si parcheggia quando la strada deve essere pulita. (65\$)
- 38 = se non viene esposto il biglietto del parcheggio. (65\$)
- 14 = divieto di sosta. (115\$)
- 36 = eccesso di velocità in zona scolastica. (50\$)
- 37 = parchimetro scaduto. (65\$)

violation_code	number_of_tickets_violation
21	5929420
38	4867892
14	3608660
36	3583082
37	2817819

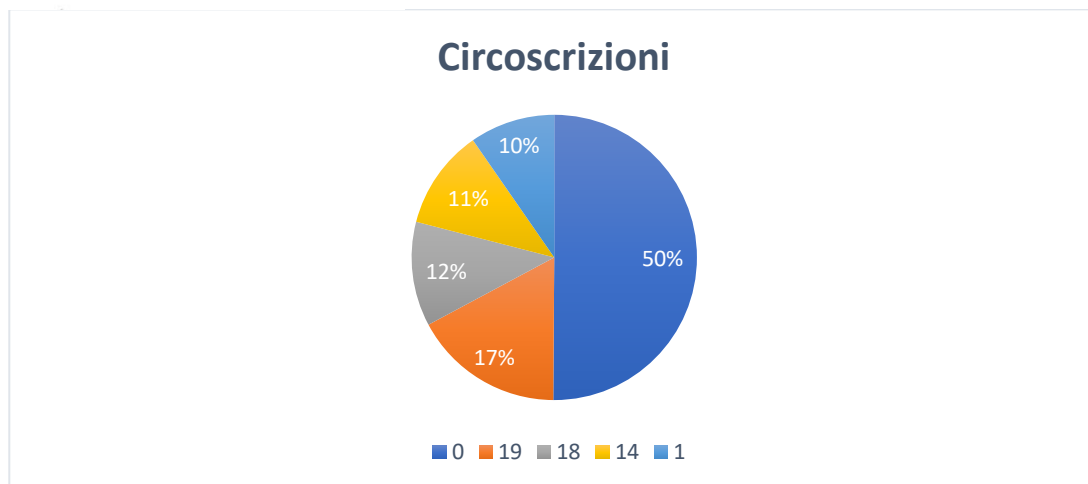


#### 6.1.4 Circoscrizioni più multate

Come sappiamo, i vari comandi di polizia dell'area metropolitana di New York sono suddivisi in circoscrizioni detti 'Precinct'. Per i risultati ottenuti abbiamo ricercato l'indirizzo della circoscrizione per identificare a quali dei 5 distretti di New York si riferissero i risultati:

```
violation_precinct=spark.sql("SELECT `Violation Precinct` as violation_precinct, count(`Violation Precinct`) as `ticket_frequency` FROM nycTable GROUP BY `violation_precinct` ORDER BY `Ticket_Frequency` DESC LIMIT 50")
violation_precinct.show(5)
```

violation_precinct	ticket_frequency	
0.0	6411256	•0 = 233 West 10 Street;
19.0	2185007	•19 = 153 East 67th Street;
18.0	1514670	•18 = 306 West 54Th Street;
14.0	1449738	•14 = 357 West 35th Street;
1.0	1234312	•1 = 16 Ericsson Palace;



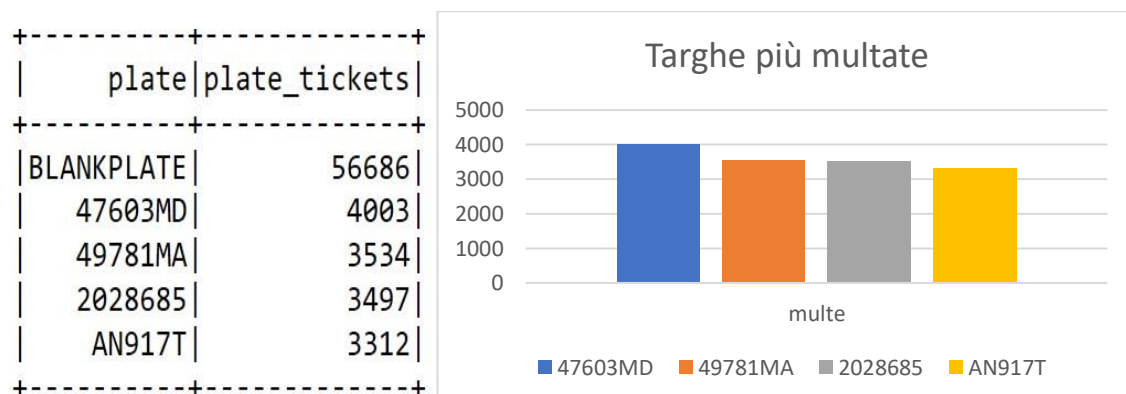
Le zone dove vengono rilevate il maggior numero di infrazioni sono nella sola area di Manhattan, sintomo di una maggiore attenzione da parte della polizia o di maggiori limitazioni dal punto di vista territoriale.

#### 6.1.5 Le targhe più multate

Abbiamo identificato le 5 targhe di vetture che hanno collezionato più multe in tutto il periodo preso in considerazione:

```
most_tickets=spark.sql("SELECT `Plate ID` as plate, count(`Plate ID`) as `plate_tickets` FROM nycTable GROUP BY `plate` ORDER BY `plate_tickets` DESC LIMIT 50")
most_tickets.show(5)
```

Ne segue, che la maggior parte delle auto multate sono risultate sprovviste di targa o quest'ultima non è stata proprio registrata.



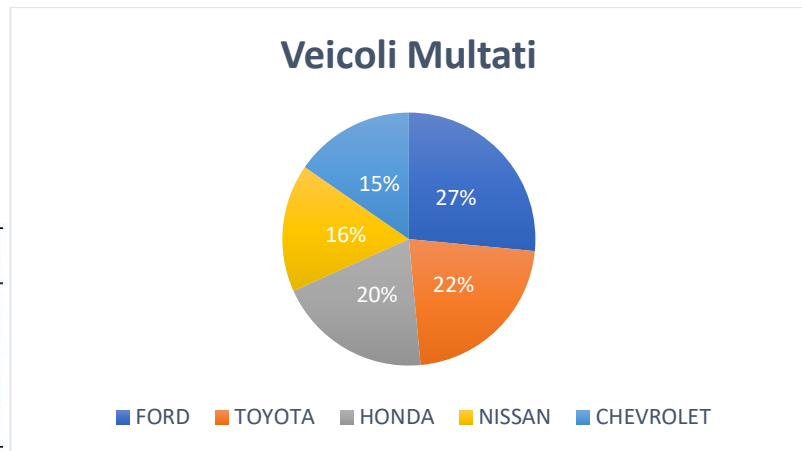
#### 6.1.6 Case automobilistiche

Inoltre abbiamo ricercato quali siano le case automobilistiche le cui macchine prodotte hanno ricevuto più multe:

```
common_makers = spark.sql("SELECT `Vehicle Make` as vehicle_maker, count('Vehicle Make') as maker_ticket_freq FROM nycTable GROUP BY `vehicle_maker` ORDER BY `maker_ticket_freq` DESC LIMIT 50")
common_makers.show(5)
```

Dal risultato emerge come i veicoli Ford siano i più multati dagli agenti, seguita da Toyota ed Honda entrambe case giapponesi.

vehicle_maker	maker_ticket_freq
FORD	5374953
TOYOT	4460841
HONDA	4004792
NISSA	3315479
CHEVR	3114838



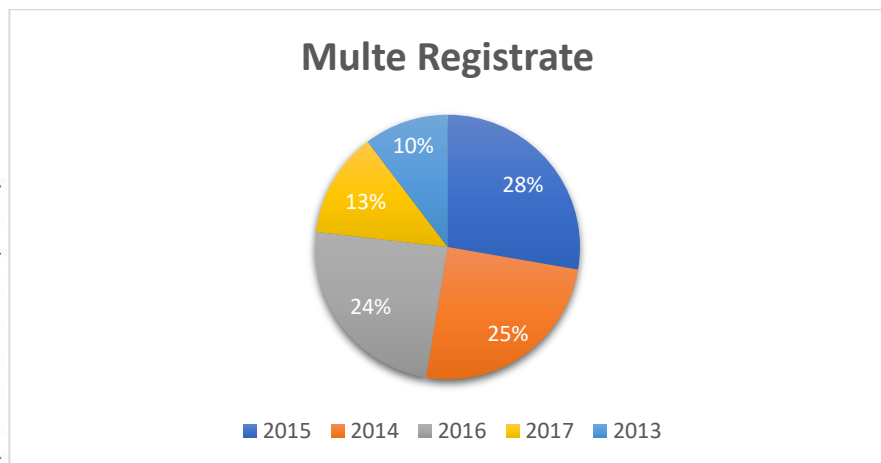
Abbiamo confrontato il numero di multe effettuate per ogni anno del dataset:

```
best_year=spark.sql("SELECT `Year` as year, count('Summons Number') as `total_year` FROM nycTable GROUP BY `Year` ORDER BY `total_year` DESC LIMIT 50")
best_year.show(5)
```

#### 6.1.7 Multe per ogni anno

Analizzando i risultati prodotti, si evince come l'assenza dei record di sette mesi per l'anno 2013 e sei per l'anno 2017 restituisca dei risultati dimezzati rispetto al trend ricavato. Possiamo ipotizzare quindi che per entrambi gli anni, a record completi, otteniamo circa gli stessi valori che presentano gli anni 2014, 2015 e 2016. Anche ipotizzando un raddoppio dei record per gli anni 13 e 17, comunque il 2015 si rivela essere l'anno con più multe.

year	total_year
2015	11736901
2014	10536290
2016	10240094
2017	5433075
2013	4377492

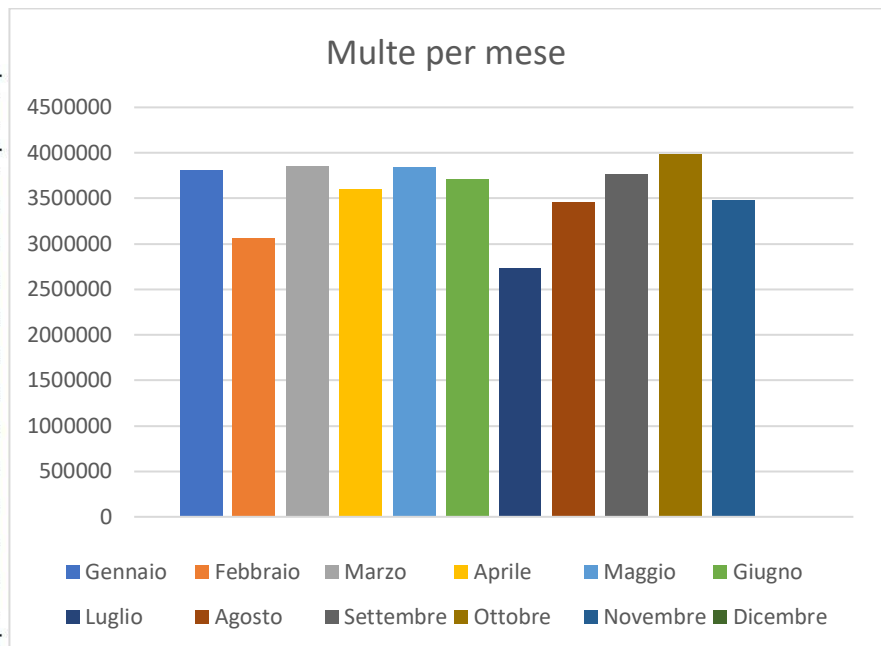


#### 6.1.8 Mesi con più multe

Col fine di individuare una certa stagionalità nei dati, abbiamo confrontato le multe registrate per ogni mese dell'anno sui 5 anni disponibili per l'analisi nel dataset.



month	total_month
10	3985513
3	3851127
5	3838998
1	3808758
9	3756397
6	3709627
4	3600577
11	3481936
8	3454190
12	3054589
2	3053541
7	2728599



I mesi che hanno fatto registrare nel periodo 2013-2017 più multe sono stati quelli di Ottobre, Marzo e Maggio.

## 6.2 Risultati Classificazione

Con la Binary Classification si effettua un controllo sull'efficienza del modello, infatti è una classificazione basata sul fatto che esistano soltanto due possibilità mutualmente escludenti.

Con la MultiClass Classification invece si stima l'accuratezza della predizione, ovvero la percentuale delle istanze classificate correttamente sul totale delle istanze presentate.

Efficienza del modello:

```
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
evaluator.evaluate(predictions)
```

0.6623773421608148

Accuratezza del modello:

```
# Accuracy: Evaluate model
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", metricName = 'accuracy')
evaluator.evaluate(predictions)
```

0.7798278859368312

Dato che la MultiClass Classification prende più classi come possibilità ma che risultano essere mutualmente escludenti, essa riesce a fornire una stima migliore per indagare l'accuratezza del modello.

Si mostra come al 77,9% il Logistic Regression Model sopra definito sarà in grado di effettuare una previsione corretta sul "Registration State", cioè la provenienza delle vetture degli automobilisti che saranno multati.

## 7. Conclusioni e Sviluppi Futuri

Lo scopo di questo progetto è stato quello di analizzare il dataset delle multe effettuate nella città di New York tramite un processo di Batch Analysis utilizzando tecnologie appartenenti al mondo dei Big Data che ci hanno permesso di estrarre informazioni dettagliate e trend in maniera efficiente da un dataset di notevoli dimensioni. Siamo riusciti a gestire le informazioni estratte e abbiamo effettuato la memorizzazione nel database NoSql Cassandra che potrebbe essere consultato per supportare un eventuale processo decisionale.

Essendo il dataset molto versatile, in futuro si potrebbe ampliare l'analisi aggiungendo altre queries per poter ottenere più andamenti possibili. Infatti nel nostro studio, abbiamo effettuato queries molto generiche col fine di estrarre un numero maggiore di andamenti per i dati a disposizione, senza entrare nel dettaglio per vari campi analizzati. Sarebbe possibile invece effettuare nuove queries con una grana più fine per ottenere ulteriori trend più specifici.

Un possibile sviluppo futuro per il sistema potrebbe essere quello di includere una fase di data ingestion dei dati tramite Apache Kafka e una successiva elaborazione in Stream Processing con Apache Storm in modo da poter gestire le infrazioni in maniera giornaliera.

Infatti i dati sulle multe potrebbero essere ricevuti in tempo reale dalla polizia di New York City ed analizzati per fornire una situazione dettagliata in tempo reale della città.

Infine un sistema completo potrebbe essere d'aiuto agli automobilisti di New York come guida agli spostamenti: valutando il mese, la zona della città e il veicolo che guidano potrebbero così decidere se è il caso di muoversi in macchina e successivamente parcheggiare senza rischiare una multa o prendere i mezzi.