SRH University Heidelberg

Data Engineering

Project Report

on

New York Cabs

Authors:
Rohit Bewoor (11011831)
Sanika Avinash Medankar (11011861)
Tushar Dwiwedi (11011834)

Submitted on 11.01.2019

Authorship Details

1 - Introduction -- Tushaar

2 - Information on Data Source -- Rohit

2.1 Description for Yellow Taxi - June 2018 .csv file

3 - Pipeline Structure and Overview

3.1 Data Ingestion, Storage and Visualization overview -- Sanika

3.2 Kafka Producer -- Sanika

3.3 Kafka Consumer -- Rohit

3.4 Storage for Permanent Repository --- Rohit

4 - Analysis and Visualization

4.1 Analysis of the data from MongoDb -- All three Rohit, Sanika, Tushaar

4.2 Visualization -- see below
Plot 1, 2, 3, 4, 5, 6 -- Tushaar
Plot 7, 8, 13 -- Sanika
Plot 9, 10, 11, 12 -- Rohit

5 - Challenges and Learnings -- 50% Sanika and 50% Tushaar

1 - Introduction

NYC Taxi tripwise data is being analysed for the month of June 2018 for the Yellow taxi fleet.

This project will give an insight on how to fetch data from CSV file, pass it through a Kafka pipeline and store it into a NoSQL Database (in our case MongoDB). MongoDb is then queried for analysis and visualization. Due to huge size of each CSV file and limited processing power on laptops, we selected and downloaded the Yellow Taxi (June) file (see section on Data Source for details).
The entire implementation is in a non-distributed environment on a single laptop.

Some basic questions are answered in the section on Analysis and Visualisation using parameters like fare, distance, passengers and payment method used for the trips. A derived field of trip duration is also used for analysis.

2 - Information on Data Source

New York City Taxi and Limousine Commission is a US Government agency responsible for licensing and regulating over 50,000 vehicles and approximately 100,000 drivers in New York City. Vendors appointed by them provide trip wise information via a Trip Sheet Data which is captured into a central pool and is made available publicly.
Monthwise (from Jan 2009 to June 2018) NYTaxi trip data is available as a CSV file for each of the three types of taxis: Green, Yellow and FHV. File sizes range from around 700Mb to 2.7Gb.
Due to file size and processing power constraints we are only working with one of the smaller files: Yellow Taxi - June 2018 data of approximately 750Mb size.

2.1 Data Description for Yellow Taxi

The data dictionary describes the fields (see References section). Each row has information such as trip distance, fare breakdown, number of passengers, pick up and drop off timestamps and locations, mode of payment, etc. The timestamps are in DD-MM-YYYY hh:mm:ss format and the location information is a unique number that indicates a general area. The range of the pickup locations is not explicitly given in the data dictionary but appears to range from 1 thru 265.

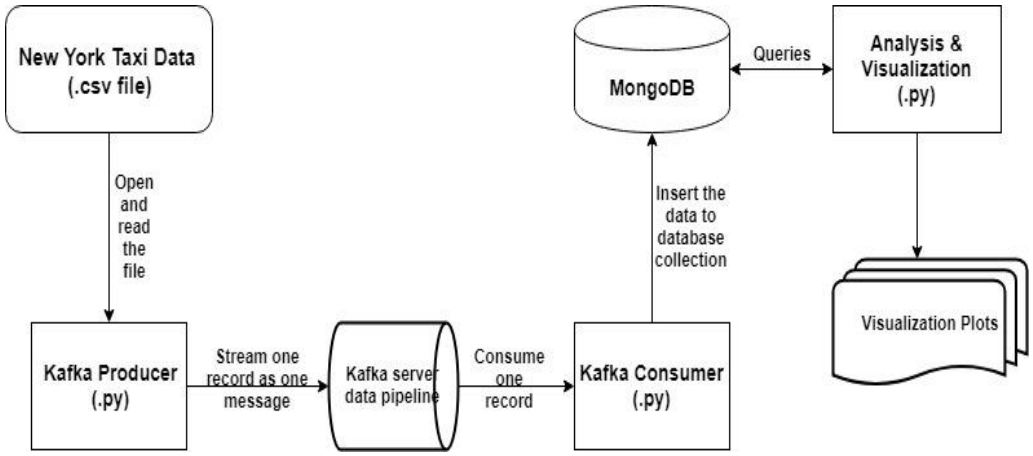
NOTE: For brevity the field Pick Up Datetime is referred to as PUdt from now. Similarly the Drop Off Datetime is called DOdt.

Data is well structured data with 17 columns and each trip is recorded as one row in the CSV file. The first row is a header with columns names and the second row contains only commas. Actual data starts from the third row and has the following fields for each trip: VendorID, PUdt, DOdt, number of Passengers, Distance covered, Rate Code, whether trip info was Stored and Forwarded, PU location, DO location, Payment Method, Fare breakdown (across multiple fields with grand total amount).
Total rows in file (including the first two rows) = 8,713,832

Analysis of the file showed that a few rows (around 500) have PUdt of May-2018 or even of the year 2009 and 2010.

3 - Pipeline Structure and Overview

3.1 Data Ingestion, Storage and Visualization overview



Data Ingestion:
The data is taken from a static data source i.e. CSV file and sent into the Kafka pipeline. Since the data in the CSV is for the whole month of June, not all the data is used for analysis. Only the first 5 days of June data is used for data streaming. After opening this file each row is read and send by the producer to the kafka pipeline. To stream data only from the PUdt of June 2018, a check on the PUdt is applied by specifying necessary start and end date parameters and comparing with those bounds before sending message to Kafka.
One could have implemented the project by treating the data as a static source CSV file that is downloaded from the NYT website and directly processed into NoSQL DB. But we are implementing a Kafka pipeline from a learning outcome perspective.

In our implementation, the Kafka Producer is a python script that sends the messages. A second python script (Kafka Consumer in the diagram above) reads all the messages from Kafka.

Data Storage:
Kafka Consumer python script consumes the data from the Kafka topic and has been tested running it as a Single instance and also running five instances in parallel (as the topic was created with 5 partitions). After inspecting the PUdt from the message to ensure it falls within the a June 2018 date, the entire field values are extracted from the message and populated into a JSON format to write to the Mongo database collection.

Data Visualization:
Queries to the Mongo database are issued to extract information that can then be visualised via suitable plots. A trip duration field is derived by finding the difference between the PUdt and DOdt fields available in the trip data.

3.2 Kafka Producer

Kafka Producer is used to publish stream CSV's row wise contents to a Kafka topic which was created with 5 partitions. To avoid laptop crash with memory issues, the processing is done by reading one row at a time and sending it to as a message. Two values for the upper and lower boundaries of the PUdt are hardcoded in the producer program and are to be specified before running. In our project the upper bound is "2018-07-01 00:00:00" and the lower bound is "2018-06-01 00:00:00" to exclude any rows with non-June data. Thus only those rows with a PUdt within June will be sent to Kafka. This prevented around 500 rows from being sent in our case. Note we want to actually process only the first 28 days data but are purposely sending data spanning the next 2 days also.

```
boundLowerPUDatetime = '2018-06-01 00:00:00'   # the pickup datetime from csv to be greater or equal to this
boundUpperPUDatetime = '2018-07-01 00:00:00'   # the pickup datetime from csv to be lower than this

with open(fileLocation, "r") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=",")
    for row in csv_reader:
        rowsReadcount = rowsReadcount + 1
        if rowsReadcount % 500000 == 0:
            print(f'Processing csv row number {rowsReadcount}')
        if rowsReadcount == (maxRows2Read + 1):
            print('Reached rows to read limit set by user')
#            print('Reached limit for first 24 hours and exiting last entry should have PickUp time of 01-06-2018  23:43:00')
            break
        try:
            row_tpep_pickup_datetime = row[1]
        except:
            print(f'Skipped row as Error during PU date assignment for row={rowsReadcount}')
            continue
#        print(f'row_tpep_pickup_datetime={row_tpep_pickup_datetime}')
        if (row_tpep_pickup_datetime < boundUpperPUDatetime and row_tpep_pickup_datetime >= boundLowerPUDatetime):
            joinedRow = ','.join(row)
            producer.send(topicName, joinedRow.encode('utf-8'))
            countSent2Producer = countSent2Producer + 1
            if countSent2Producer % 500000 == 0:
                print(f'Sent {rowsReadcount} row as the {countSent2Producer} th message to Kafka')
        elif not (row[0] == 'VendorID'):
#            print(f'Row {rowsReadcount} not sent to producer, row PUDatetime={row_tpep_pickup_datetime} and outside bounds')
            boundNotMatchRowsCount = boundNotMatchRowsCount + 1
```

Example of the output displayed after the producer runs. It took about 80 minutes to run and sent almost the entire 8.71 million rows to Kafka while leaving out the 500 odd rows with non-June PUdt.

```
StartTime is: Wed Jan  9 23:08:02 2019
Skipped row as Error during PU date assignment for row=2
Processing csv row number 100000 at  Wed Jan  9 23:08:25 2019
Processing csv row number 200000 at  Wed Jan  9 23:08:40 2019
Processing csv row number 300000 at  Wed Jan  9 23:09:01 2019
```

```
Sent 8500439 row as the 8500000 th message to Kafka
Processing csv row number 8600000 at  Thu Jan 10 00:25:43 2019
Processing csv row number 8700000 at  Thu Jan 10 00:26:45 2019


Processed 8713832 rows from csv
Sent 8713307 messages to Kafka
NOT SENT 524 rows as OB

EndTime is: Thu Jan 10 00:26:53 2019
```

3.3 Kafka Consumer

Salient features of the consumer implementation:

a) Use of Group name: allows running of multiple instances of the consumer program in parallel and speeds up processing and writing to MongoDb. Topic had been created with 5 partitions to support up to 5 such above instances. Below are snapshots from command prompt queries to Kafka server about number of messages in topic and the offsets for the individual consumers.

```
19 consumer = KafkaConsumer(
20      bootstrap_servers=['localhost:9092'],
21      auto_offset_reset='earliest',
22 #     auto_offset_reset='latest',    # default is latest
23      enable_auto_commit=True,
24      auto_commit_interval_ms=2000,
25      group_id='SingleRunning1')
```

<u>Multi Instance Consumer Run:</u>
Topic was created, producer was started and after about 10 seconds time we started FIVE instances of the consumer program in parallel with group name “MultipleRunning30-1”.

```
C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 5 --topic TestNYTJune30days
Created topic "TestNYTJune30days".

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-run-class.bat kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic TestNYTJune30days --time -1
TestNYTJune30days:0:0
TestNYTJune30days:1:0
TestNYTJune30days:2:0
TestNYTJune30days:3:0
TestNYTJune30days:4:0

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-run-class.bat kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic TestNYTJune30days --time -1
TestNYTJune30days:0:56440
TestNYTJune30days:1:57115
TestNYTJune30days:2:56264
TestNYTJune30days:3:56457
TestNYTJune30days:4:56625

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group MultipleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 1         2773            81819           79046           kafka-python-1.4.4-1c2398d1-82de-4b69-9d4f-921c888e3662 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         4730            81132           76402           kafka-python-1.4.4-d3519efc-c2f1-4462-8074-647d6515d0da /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 3         9774            81143           71369           kafka-python-1.4.4-d9b6dbd1-bee1-44ae-a215-b01918cdaee7 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         3509            81354           77845           kafka-python-1.4.4-08e3a573-8e61-4e7c-8ba4-a19a2bfad366 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         3668            81667           77999           kafka-python-1.4.4-da815fbb-9da4-4144-b007-1a1268948001 /172.19.182.0   kafka-python-1.4.4
```

At certain times, Kafka Broker did not use all the 5 instances and rebalanced to assign only 3 instances to the 5 partitions. See the transition below.

```
C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group MultipleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 1         366940          383229          16289           kafka-python-1.4.4-1a58e744-2e96-4a68-9d37-e5744e472694 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 3         376553          381756          5203            kafka-python-1.4.4-85a7433b-a831-4433-9328-c0b395ef15a4 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         364993          382982          17989           kafka-python-1.4.4-3c3701a8-299e-4fb2-958f-de536f0d7b6e /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         376754          382319          5565            kafka-python-1.4.4-d3519efc-c2f1-4462-8074-647d6515d0da /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         362262          381865          19603           kafka-python-1.4.4-08e3a573-8e61-4e7c-8ba4-a19a2bfad366 /172.19.182.0   kafka-python-1.4.4

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group MultipleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 0         413721          425183          11462           kafka-python-1.4.4-08e3a573-8e61-4e7c-8ba4-a19a2bfad366 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         407500          426371          18871           kafka-python-1.4.4-08e3a573-8e61-4e7c-8ba4-a19a2bfad366 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         413248          426660          13412           kafka-python-1.4.4-08e3a573-8e61-4e7c-8ba4-a19a2bfad366 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 3         414601          424709          10108           kafka-python-1.4.4-1a58e744-2e96-4a68-9d37-e5744e472694 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         419157          425612          6455            kafka-python-1.4.4-1a58e744-2e96-4a68-9d37-e5744e472694 /172.19.182.0   kafka-python-1.4.4

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group MultipleRunning30-1 --describe
Warning: Consumer group 'MultipleRunning30-1' is rebalancing.

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST            CLIENT-ID
TestNYTJune30days 3         460568          462166          1598            -               -               -
TestNYTJune30days 2         458388          463839          5451            -               -               -
TestNYTJune30days 1         456945          464016          7071            -               -               -
TestNYTJune30days 0         454404          462916          8512            -               -               -
TestNYTJune30days 4         462129          463171          1042            -               -               -

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group MultipleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 0         459254          470037          10783           kafka-python-1.4.4-064ec3ba-39c8-4151-ab9b-4703efbe68b9 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         457425          471342          13917           kafka-python-1.4.4-064ec3ba-39c8-4151-ab9b-4703efbe68b9 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         458388          471057          12669           kafka-python-1.4.4-1836a805-adc8-4a04-b151-ae557a699047 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 3         460568          469488          8920            kafka-python-1.4.4-1836a805-adc8-4a04-b151-ae557a699047 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         467238          470411          3173            kafka-python-1.4.4-1a58e744-2e96-4a68-9d37-e5744e472694 /172.19.182.0   kafka-python-1.4.4
```

Consumer run has almost ended now with only two consumer instances still consuming.

```
C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group MultipleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 2         1744224         1744224         0               kafka-python-1.4.4-64673952-c9ce-4894-a1d3-93373c61f946 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 3         1739851         1739851         0               kafka-python-1.4.4-74442612-37d0-46de-97d7-f83005ca0708 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         1691953         1745187         53234           kafka-python-1.4.4-357e2413-a5f6-4a59-b105-3ecdbf0b1daf /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         1743286         1743286         0               kafka-python-1.4.4-a8fe2346-00d9-4d59-883e-a53b40b02b56 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         1694649         1740759         46110           kafka-python-1.4.4-032cb984-0cd2-47bf-be4a-7c2476354155 /172.19.182.0   kafka-python-1.4.4

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group MultipleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 2         1744224         1744224         0               kafka-python-1.4.4-64673952-c9ce-4894-a1d3-93373c61f946 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 3         1739851         1739851         0               kafka-python-1.4.4-74442612-37d0-46de-97d7-f83005ca0708 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         1745187         1745187         0               kafka-python-1.4.4-357e2413-a5f6-4a59-b105-3ecdbf0b1daf /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         1743286         1743286         0               kafka-python-1.4.4-a8fe2346-00d9-4d59-883e-a53b40b02b56 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         1740759         1740759         0               kafka-python-1.4.4-032cb984-0cd2-47bf-be4a-7c2476354155 /172.19.182.0   kafka-python-1.4.4
```

<u>Single Instance Consumer Run:</u>

Now producer program has already run, we now again ran only ONE consumer with group name “SingleRunning30-1”.

Here we can clearly see the single consumer slowly reading messages from each partition. In the first snapshot below, it has not even started consuming from some of the partitions as yet.

```
C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group SingleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 3         9669            1739851         1730182         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         0               1744224         1744224         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         0               1745187         1745187         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         0               1740759         1740759         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         0               1743286         1743286         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group SingleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 3         10020           1739851         1729831         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         1350            1744224         1742874         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         0               1745187         1745187         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         0               1740759         1740759         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         9989            1743286         1733297         kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
```

After around 115 mins of starting it consumes all the messages

```
C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group SingleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 3         1726717         1739851         13134           kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         1744224         1744224         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         1745187         1745187         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         1740759         1740759         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         1739587         1743286         3699            kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group SingleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 3         1731497         1739851         8354            kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         1744224         1744224         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         1745187         1745187         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         1740759         1740759         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         1743286         1743286         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4

C:\Everything\Kafka2.1.0-Scala2.12\bin\windows>kafka-consumer-groups.bat --bootstrap-server localhost:9092 --group SingleRunning30-1 --describe

TOPIC            PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
TestNYTJune30days 3         1739851         1739851         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 2         1744224         1744224         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 1         1745187         1745187         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 0         1740759         1740759         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
TestNYTJune30days 4         1743286         1743286         0               kafka-python-1.4.4-1b0b57f2-996f-4a1e-9a36-7d474afbccc3 /172.19.182.0   kafka-python-1.4.4
```

Comparison of execution time for the single consumer v/s multi consumer (5) runs:

# of consumers in parallel	Runtime	Remarks
5	82 mins	Same computer (hardware Win10 with 8Gb ram). Producer and Multiple Consumers running with group name MultipleRunning30-1 at same time. At times, some consumers have already processed messages in the partition and are idle. Single instance consumer run after Producer ended already. So it ran without any downtime unlike some consumers in the multi-instance scenario. Group name SingleRunning30-1 Approx. 8.7 million messages consumed.
1	115 mins	

b) Even though the time period decided for analysis is only 28 days starting midnight of 01.06.2018 to midnight of 29.06.2018, we are reading messages outside this timeframe from the Kafka pipeline. Messages with PUdt within first 28 days are stored in one collection#1 and all (for 2 days in this case) are routed to another collection#2 in the same database.

```
boundLowerPUDatetime1 = '2018-06-01 00:00:00'   # the pickup datetime from csv to be greater or equal to this
boundUpperPUDatetime1 = '2018-06-29 00:00:00'   # the pickup datetime from csv to be lower than this
boundLowerPUDatetime2 = '2018-06-29 00:00:00'   # the pickup datetime from csv to be greater or equal to this
boundUpperPUDatetime2 = '2018-07-01 00:00:00'   # the pickup datetime from csv to be lower than this
    '"total_amount" : ' + str(msgAsList[16]) + '}'
    #print(f'Msg {msgsReadCount} - JSON built as=\n{NYTNewRowJsonBuiltUpAsString}')
    if ( str(msgAsList[1]) < boundUpperPUDatetime1 and str(msgAsList[1]) >= boundLowerPUDatetime1 ):
        try:
            collection1.insert_one(loads(NYTNewRowJsonBuiltUpAsString))
            #print(f"Collection 1 -- NO ERROR for {msgsReadCount} th message insert")
            countDocsWritten1 = countDocsWritten1 + 1
            if countDocsWritten1 % 200000 == 0:
                print(f"Collection 1 - write SUCCESS - {msgsReadCount} th message as {countDocsWritten1} th insert at ", datet
            if countDocsWritten1 % 500000 == 0:
                print(f'Msg {msgsReadCount} - JSON built as=\n{NYTNewRowJsonBuiltUpAsString}')
        except Exception as e:
            print(f'Collection 1 ERROR Insert :: {msgsReadCount} th message\nError:: ', type(e), e)
    elif ( str(msgAsList[1]) < boundUpperPUDatetime2 and str(msgAsList[1]) >= boundLowerPUDatetime2 ):
        try:
            collection2.insert_one(loads(NYTNewRowJsonBuiltUpAsString))
            #print(f"Collection 2 -- NO ERROR for {msgsReadCount} th message insert")
            countDocsWritten2 = countDocsWritten2 + 1
            if countDocsWritten2 % 200000 == 0:
                print(f"Collection 2 - write SUCCESS - {msgsReadCount} th message as {countDocsWritten2} th insert at ", datet
            if countDocsWritten2 % 500000 == 0:
                print(f'Msg {msgsReadCount} - JSON built as=\n{NYTNewRowJsonBuiltUpAsString}')
        except Exception as e:
            print(f'Collection 2 ERROR Insert :: {msgsReadCount} th message\nError:: ', type(e), e)
else:
    print(f"{msgsReadCount} th message NOT having 16 commas so IGNORED MESSAGE")
```

c) Json is explicitly built in the consumer program itself. We cannot assume that the first message any instance of the consumer will access the header row from the CSV file. This could be done in the following way there are limited number of field.

```
        NYTNewRowJsonBuiltUpAsString='{' +                                          \
        '"VendorID" : ' + str(msgAsList[0]) + ', ' +                                \
        '"tpep_pickup_datetime" : ' + '"' + str(msgAsList[1]) +  '"' + ', ' +       \
        '"tpep_dropoff_datetime" : ' + '"' + str(msgAsList[2]) +  '"' + ', ' +      \
        '"passenger_count" : ' + str(msgAsList[3]) + ', ' +                         \
        '"trip_distance" : ' + trip_distanceAsString + ', ' +                       \
        '"RatecodeID" : ' + str(msgAsList[5]) + ', ' +                              \
        '"store_and_fwd_flag" : ' + '"' + str(msgAsList[6]) + '"' + ', ' +          \
        '"PULocationID" : ' + str(msgAsList[7]) + ', ' +                            \
        '"DOLocationID" : ' + str(msgAsList[8]) + ', ' +                            \
        '"payment_type" : ' + str(msgAsList[9]) + ', ' +                            \
        '"fare_amount" : ' + str(msgAsList[10]) + ', ' +                            \
        '"extra" : ' + str(msgAsList[11]) + ', ' +                                  \
        '"mta_tax" : ' + str(msgAsList[12]) + ', ' +                                \
        '"tip_amount" : ' + str(msgAsList[13]) + ', ' +                             \
        '"tolls_amount" : ' + str(msgAsList[14]) + ', ' +                           \
        '"improvement_surcharge" : ' + str(msgAsList[15]) + ', ' +                  \
        '"total_amount" : ' + str(msgAsList[16]) + '}'
        #print(f'Msg {msgsReadCount} - JSON built as=\n{NYTNewRowJsonBuiltUpAsString}')
```

3.4 Storage for Permanent Repository

MongoDb is being used for permanent storage. Allows for storing any changes in the data format as and when it occurs without affecting earlier entries. Also, no suitable unique key could be decided with certainty using the fields that are received from the CSV file. Therefore using a SQL DB is a challenge and we decided to implement using NoSQL MongoDb.

```
>
> db.TestNYTJune30daysCol2Jun01to28.count()
8155479
> db.TestNYTJune30daysCol2Jun29to30.count()
557828
>
```

Example of an entry in the database:

```
>
> db.TestNYTJune30daysCol2Jun01to28.find().skip(2140345).limit(1).pretty()
{
        "_id" : ObjectId("5c36b29c7fd09d108c12f0d9"),
        "VendorID" : 1,
        "tpep_pickup_datetime" : "2018-06-08 10:31:35",
        "tpep_dropoff_datetime" : "2018-06-08 10:57:38",
        "passenger_count" : 1,
        "trip_distance" : 5.9,
        "RatecodeID" : 1,
        "store_and_fwd_flag" : "N",
        "PULocationID" : 87,
        "DOLocationID" : 170,
        "payment_type" : 1,
        "fare_amount" : 23,
        "extra" : 0,
        "mta_tax" : 0.5,
        "tip_amount" : 4.75,
        "tolls_amount" : 0,
        "improvement_surcharge" : 0.3,
        "total_amount" : 28.55
}
```

4 - Analysis and Visualization

4.1 Analysis of the data from MongoDB

We decided to analyse the data by slicing it on a daily basis to cover 4 weeks = 28 days. Later we took just one 24 window of “from 2018.06.01 00:00:00 to less than 2018.06.02 00:00:00”. store the information into two data structures to be used later for the visualisation if required.

First is a matrix with 28 rows and 16 columns. Each row holds the aggregated or computed values for that day. The columns are for the various aggregate values such as total trips, total passengers, number of trips paid for by different payment methods, total fares, etc. Calculated fields are average values per trip for fare, distance and passengers as well as for the average duration of the trip per day.

Snapshots of part of the console output from the data structures:

From the matrix:
Note: Due to extremely long query time run time we are only running those queries that we decided to actually proceed with for analysis and some of the matrix parameters remain unpopulated and initialised to 0.

```
Analysis data:


Total Trips per day:
Day1 = 320529.0000 -- Day2 = 308393.0000 -- Day3 = 265244.0000 -- Day4 = 279353.0000 -- Day5 = 298140.0000 -- Day6 = 315354.00
00 -- Day7 = 319315.0000 -- Day8 = 315660.0000 -- Day9 = 293934.0000 -- Day10 = 249740.0000 -- Day11 = 278488.0000 -- Day12 =
295377.0000 -- Day13 = 312651.0000 -- Day14 = 316168.0000 -- Day15 = 281883.0000 -- Day16 = 275862.0000 -- Day17 = 247840.0000
-- Day18 = 281485.0000 -- Day19 = 298528.0000 -- Day20 = 307880.0000 -- Day21 = 309069.0000 -- Day22 = 311362.0000 -- Day23 =
273667.0000 -- Day24 = 243786.0000 -- Day25 = 267868.0000 -- Day26 = 287191.0000 -- Day27 = 297916.0000 -- Day28 = 302796.0000
--

Total Passenger per day:
Day1 = 507899.0000 -- Day2 = 509665.0000 -- Day3 = 431153.0000 -- Day4 = 436819.0000 -- Day5 = 466506.0000 -- Day6 = 494194.00
00 -- Day7 = 499014.0000 -- Day8 = 501788.0000 -- Day9 = 485392.0000 -- Day10 = 408601.0000 -- Day11 = 436463.0000 -- Day12 =
463919.0000 -- Day13 = 491022.0000 -- Day14 = 497015.0000 -- Day15 = 438738.0000 -- Day16 = 453013.0000 -- Day17 = 411547.0000
-- Day18 = 444826.0000 -- Day19 = 470030.0000 -- Day20 = 487199.0000 -- Day21 = 487827.0000 -- Day22 = 499181.0000 -- Day23 =
455177.0000 -- Day24 = 402607.0000 -- Day25 = 425725.0000 -- Day26 = 455372.0000 -- Day27 = 469436.0000 -- Day28 = 479499.0000
--

Average Passerger/Trip per day:
Day1 = 1.5846 -- Day2 = 1.6526 -- Day3 = 1.6255 -- Day4 = 1.5637 -- Day5 = 1.5647 -- Day6 = 1.5671 -- Day7 = 1.5628 -- Day8 =
1.5896 -- Day9 = 1.6514 -- Day10 = 1.6361 -- Day11 = 1.5673 -- Day12 = 1.5706 -- Day13 = 1.5705 -- Day14 = 1.5720 -- Day15 =
1.5565 -- Day16 = 1.6422 -- Day17 = 1.6605 -- Day18 = 1.5803 -- Day19 = 1.5745 -- Day20 = 1.5824 -- Day21 = 1.5784 -- Day22 =
1.6032 -- Day23 = 1.6633 -- Day24 = 1.6515 -- Day25 = 1.5893 -- Day26 = 1.5856 -- Day27 = 1.5757 -- Day28 = 1.5836 --

Count of StoreFwd=YES per day:
Day1 = 0.0000 -- Day2 = 0.0000 -- Day3 = 0.0000 -- Day4 = 0.0000 -- Day5 = 0.0000 -- Day6 = 0.0000 -- Day7 = 0.0000 -- Day8 =
0.0000 -- Day9 = 0.0000 -- Day10 = 0.0000 -- Day11 = 0.0000 -- Day12 = 0.0000 -- Day13 = 0.0000 -- Day14 = 0.0000 -- Day15 =
0.0000 -- Day16 = 0.0000 -- Day17 = 0.0000 -- Day18 = 0.0000 -- Day19 = 0.0000 -- Day20 = 0.0000 -- Day21 = 0.0000 -- Day22 =
0.0000 -- Day23 = 0.0000 -- Day24 = 0.0000 -- Day25 = 0.0000 -- Day26 = 0.0000 -- Day27 = 0.0000 -- Day28 = 0.0000 --
```

```
Total Fare Amount per day:
Day1 = 5323133.7500 -- Day2 = 4713918.0300 -- Day3 = 4514625.0000 -- Day4 = 4711105.2000 -- Day5 = 4984462.5000 -- Day6 = 5270
349.5900 -- Day7 = 5493762.6200 -- Day8 = 5309576.8800 -- Day9 = 4563087.7700 -- Day10 = 4265889.4200 -- Day11 = 4698539.5000
-- Day12 = 4942538.4200 -- Day13 = 5258794.0100 -- Day14 = 5452836.0600 -- Day15 = 4714051.8100 -- Day16 = 4203697.6900 -- Day
17 = 4079151.6500 -- Day18 = 4688624.2500 -- Day19 = 4986225.0700 -- Day20 = 5261322.2100 -- Day21 = 5456459.9200 -- Day22 = 5
199650.6800 -- Day23 = 4237056.4500 -- Day24 = 4141198.4000 -- Day25 = 4600853.6800 -- Day26 = 4741510.3400 -- Day27 = 501022
1.3100 -- Day28 = 5152645.4900 --

Average Fare/Trip per day:
Day1 = 16.6073 -- Day2 = 15.2854 -- Day3 = 17.0206 -- Day4 = 16.8643 -- Day5 = 16.7185 -- Day6 = 16.7125 -- Day7 = 17.2048 --
Day8 = 16.8206 -- Day9 = 15.5242 -- Day10 = 17.0813 -- Day11 = 16.8716 -- Day12 = 16.7330 -- Day13 = 16.8200 -- Day14 = 17.246
6 -- Day15 = 16.7234 -- Day16 = 15.2384 -- Day17 = 16.4588 -- Day18 = 16.6567 -- Day19 = 16.7027 -- Day20 = 17.0889 -- Day21 =
17.6545 -- Day22 = 16.6997 -- Day23 = 15.4825 -- Day24 = 16.9870 -- Day25 = 17.1758 -- Day26 = 16.5100 -- Day27 = 16.8176 -- D
ay28 = 17.0169 --

Total Distance per day:

Day1 = 929332.0200 -- Day2 = 864685.7100 -- Day3 = 871336.2100 -- Day4 = 864082.7100 -- Day5 = 862439.5600 -- Day6 = 911810.15
00 -- Day7 = 950960.9500 -- Day8 = 927880.4100 -- Day9 = 852363.0500 -- Day10 = 843888.1400 -- Day11 = 866978.6600 -- Day12 =
867238.9300 -- Day13 = 894533.0400 -- Day14 = 949691.7000 -- Day15 = 879013.2300 -- Day16 = 817087.6700 -- Day17 = 832938.6000
-- Day18 = 854696.8700 -- Day19 = 867371.8100 -- Day20 = 914718.7600 -- Day21 = 926386.4200 -- Day22 = 912975.6200 -- Day23 =
803520.7700 -- Day24 = 819535.4700 -- Day25 = 866733.4100 -- Day26 = 845639.8400 -- Day27 = 871754.2100 -- Day28 = 873963.1900
--

Average Distance/Trip per day:
Day1 = 2.8994 -- Day2 = 2.8038 -- Day3 = 3.2850 -- Day4 = 3.0932 -- Day5 = 2.8927 -- Day6 = 2.8914 -- Day7 = 2.9781 -- Day8 =
2.9395 -- Day9 = 2.8998 -- Day10 = 3.3791 -- Day11 = 3.1132 -- Day12 = 2.9360 -- Day13 = 2.8611 -- Day14 = 3.0038 -- Day15 =
3.1184 -- Day16 = 2.9619 -- Day17 = 3.3608 -- Day18 = 3.0364 -- Day19 = 2.9055 -- Day20 = 2.9710 -- Day21 = 2.9973 -- Day22 =
2.9322 -- Day23 = 2.9361 -- Day24 = 3.3617 -- Day25 = 3.2357 -- Day26 = 2.9445 -- Day27 = 2.9262 -- Day28 = 2.8863 --

Average Duration/Trip per day:
Day1 = 18.0245 -- Day2 = 16.5878 -- Day3 = 17.4535 -- Day4 = 16.7467 -- Day5 = 17.7927 -- Day6 = 17.6929 -- Day7 = 18.2760 --
Day8 = 18.0297 -- Day9 = 16.7153 -- Day10 = 17.2717 -- Day11 = 16.6433 -- Day12 = 17.2841 -- Day13 = 18.1475 -- Day14 = 18.632
5 -- Day15 = 17.1974 -- Day16 = 15.9101 -- Day17 = 16.1499 -- Day18 = 16.8686 -- Day19 = 17.6193 -- Day20 = 18.4110 -- Day21 =
20.0156 -- Day22 = 18.3481 -- Day23 = 16.7157 -- Day24 = 17.7253 -- Day25 = 17.1957 -- Day26 = 17.4983 -- Day27 = 17.8239 -- D
ay28 = 19.0022 --
```

The second data structure is a list. It can accommodate computed values for the top 5 maximum number of trips per hour starting from a certain PU (Pick-up) location, ending at a certain DO (Drop-Off) location. However for the combination of unique PU to DO location are captured into it holds the top 10 routes and their counts.
Snapshot of the list contents is below. As we decided to focus only on the combination route and ignored the other two, one sees certain elements in the list are left unpopulated and initialised to their default values that were hardcoded the program.

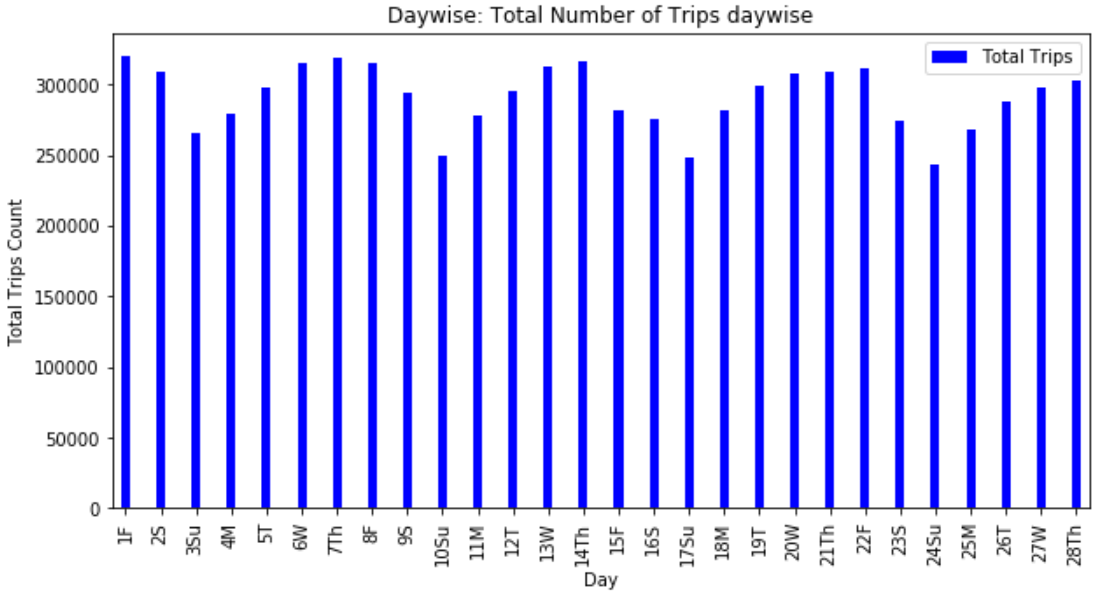
The console displays the full information for each of the 28 days.

```
The location list AFTER populating from db::

Day 1
D1  --  2018-06-01 00:00:00  --  2018-06-02 00:00:00
[['TopPULoc1', 'PULoc1-UNFILLED', 0], ['TopPULoc2', 'PULoc2-UNFILLED', 0], ['TopPULoc3', 'PULoc3-UNFILLED', 0], ['TopPULoc4',
'PULoc4-UNFILLED', 0], ['TopPULoc5', 'PULoc5-UNFILLED', 0]]
[['TopDOLoc1', 'DOLoc1-UNFILLED', 0], ['TopDOLoc2', 'DOLoc1-UNFILLED', 0], ['TopDOLoc3', 'DOLoc3-UNFILLED', 0], ['TopDOLoc4',
'DOLoc4-UNFILLED', 0], ['TopDOLoc5', 'DOLoc5-UNFILLED', 0]]
[['TopPUDOCombo1', '264 TO 264', 3276], ['TopPUDOCombo2', '237 TO 236', 2090], ['TopPUDOCombo3', '236 TO 236', 1878], ['TopPUD
OCombo4', '236 TO 237', 1764], ['TopPUDOCombo5', '237 TO 237', 1471], ['TopPUDOCombo6', '239 TO 238', 914], ['TopPUDOCombo7',
'239 TO 142', 908], ['TopPUDOCombo8', '237 TO 162', 859], ['TopPUDOCombo9', '238 TO 239', 814], ['TopPUDOCombo10', '141 TO 23
6', 814]]

Day 2
D2  --  2018-06-02 00:00:00  --  2018-06-03 00:00:00
[['TopPULoc1', 'PULoc1-UNFILLED', 0], ['TopPULoc2', 'PULoc2-UNFILLED', 0], ['TopPULoc3', 'PULoc3-UNFILLED', 0], ['TopPULoc4',
'PULoc4-UNFILLED', 0], ['TopPULoc5', 'PULoc5-UNFILLED', 0]]
[['TopDOLoc1', 'DOLoc1-UNFILLED', 0], ['TopDOLoc2', 'DOLoc1-UNFILLED', 0], ['TopDOLoc3', 'DOLoc3-UNFILLED', 0], ['TopDOLoc4',
'DOLoc4-UNFILLED', 0], ['TopDOLoc5', 'DOLoc5-UNFILLED', 0]]
[['TopPUDOCombo1', '264 TO 264', 3495], ['TopPUDOCombo2', '237 TO 236', 1415], ['TopPUDOCombo3', '236 TO 237', 1232], ['TopPUD
OCombo4', '236 TO 236', 1146], ['TopPUDOCombo5', '237 TO 237', 1081], ['TopPUDOCombo6', '239 TO 142', 1038], ['TopPUDOCombo7',
'79 TO 79', 906], ['TopPUDOCombo8', '239 TO 238', 828], ['TopPUDOCombo9', '142 TO 239', 787], ['TopPUDOCombo10', '263 TO 141',
713]]

Day 3
D3  --  2018-06-03 00:00:00  --  2018-06-04 00:00:00
[['TopPULoc1', 'PULoc1-UNFILLED', 0], ['TopPULoc2', 'PULoc2-UNFILLED', 0], ['TopPULoc3', 'PULoc3-UNFILLED', 0], ['TopPULoc4',
'PULoc4-UNFILLED', 0], ['TopPULoc5', 'PULoc5-UNFILLED', 0]]
[['TopDOLoc1', 'DOLoc1-UNFILLED', 0], ['TopDOLoc2', 'DOLoc1-UNFILLED', 0], ['TopDOLoc3', 'DOLoc3-UNFILLED', 0], ['TopDOLoc4',
'DOLoc4-UNFILLED', 0], ['TopDOLoc5', 'DOLoc5-UNFILLED', 0]]
[['TopPUDOCombo1', '264 TO 264', 3033], ['TopPUDOCombo2', '237 TO 236', 1071], ['TopPUDOCombo3', '236 TO 237', 1058], ['TopPUD
OCombo4', '236 TO 236', 1001], ['TopPUDOCombo5', '239 TO 238', 893], ['TopPUDOCombo6', '79 TO 79', 837], ['TopPUDOCombo7', '23
9 TO 142', 825], ['TopPUDOCombo8', '237 TO 237', 820], ['TopPUDOCombo9', '238 TO 239', 740], ['TopPUDOCombo10', '142 TO 239',
733]]

Day 4
```

4.2 Visualization

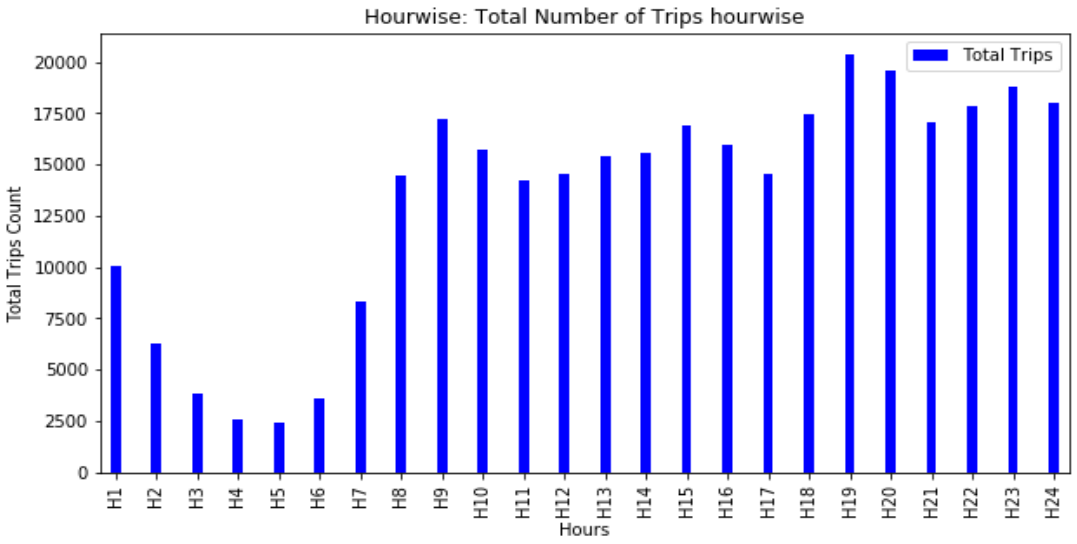
Plot 1: Total Number of trips per day from 01.06 to 28.06

- Total trips vary from around 265k to 315k
- Maximum demand on Thursdays and Fridays, lowest on Sunday



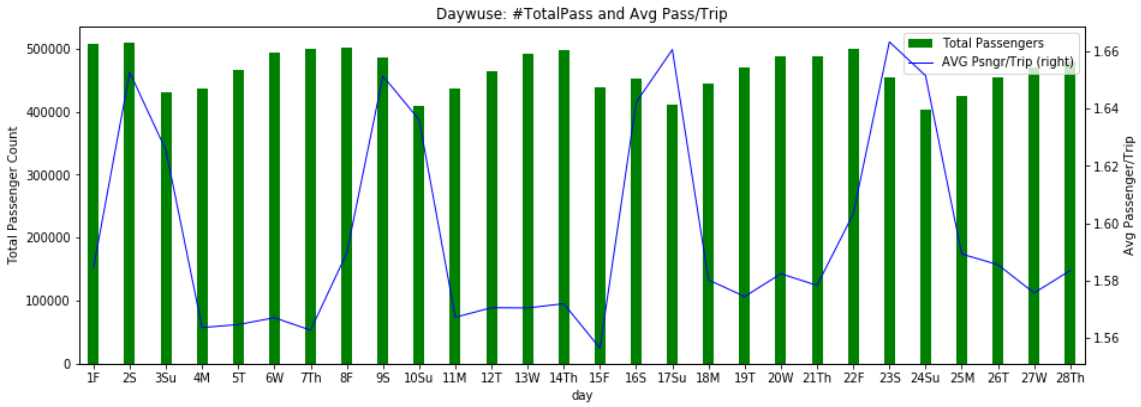
Plot 2: Total Number of trips per hour on 01.06.2018 (Friday)

- H2 to H6: Low number of trips during dead of night till H5 and then very slow turnaround towards rising numbers seen for H7 between 0500 to 0600
- H8 to H18: Number of trips almost steady around 16000 per hour
- H19 to H24: Number of trips rises to 20k+ and remains higher than before H18 - probably due to Friday night and late turn in by NY population.



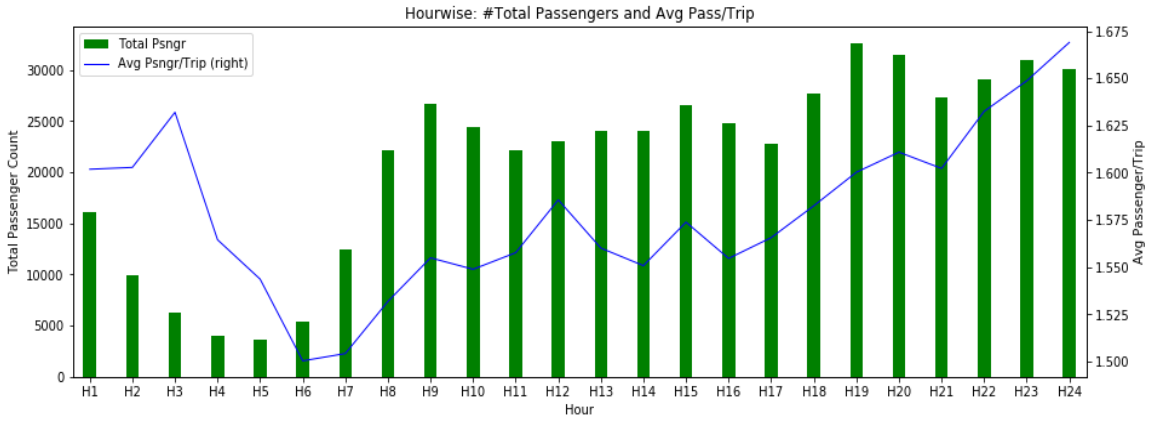
Plot 3: Daily variation of Total passengers v/s average number of passengers/trip (28 days)

- Average Passengers per Trip low from Monday to Thursday and higher on weekends
- Possibly single ride tendency increases on weekends
- Total number of passengers taking taxis follows the total trips trend cycle in general



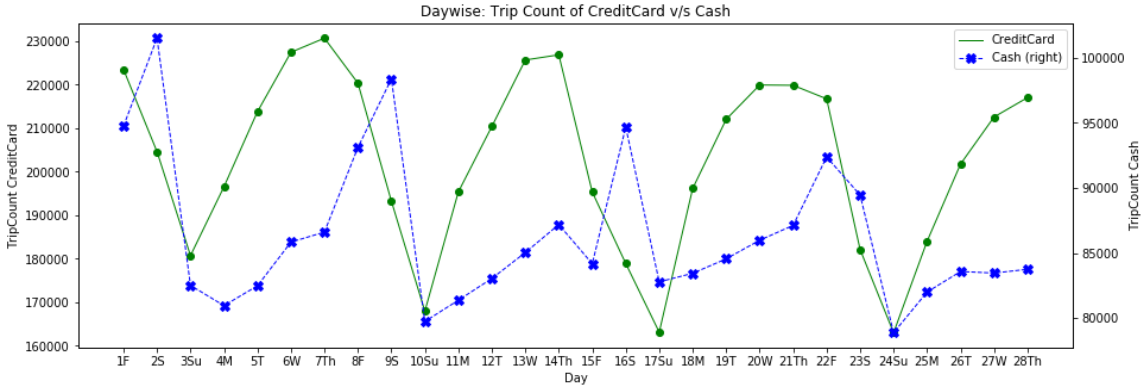
Plot 4: Total passengers per hour and the average number of passengers per trip

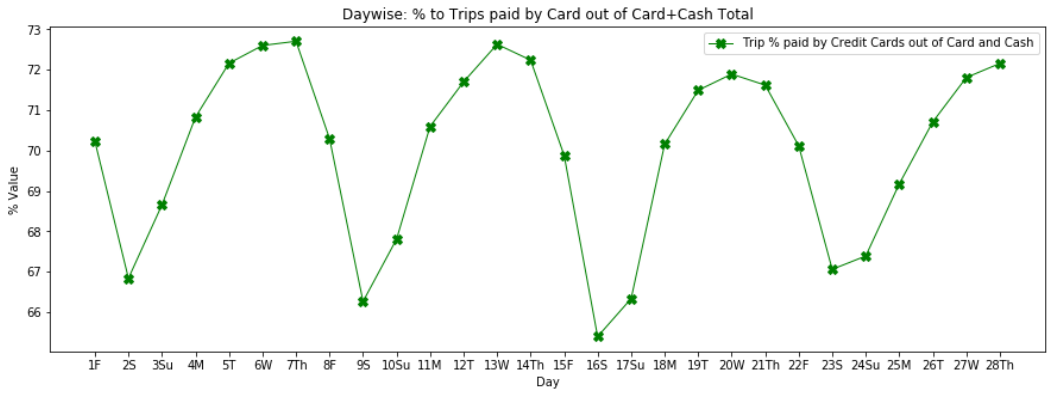
- H3 to H5: Total passengers dropped by 40% (6293 to 3677), (total number of trips fell by 40%), Avg Psng/Trip fell 5% (1.63 to 1.54) => during this time period, the tendency for multi passenger trips is higher
- H12 to H14: Passenger total remained steady or increased very slightly, but the avg Psng/Trip fell by about 3% => single ride tendency increased.



Plot 5: Daily variation (28 days): Number of trips paid for by Credit Card v/s Cash
Plot 6: Daily variation (28 days): % to Trips paid by Card out of Card+Cash Total

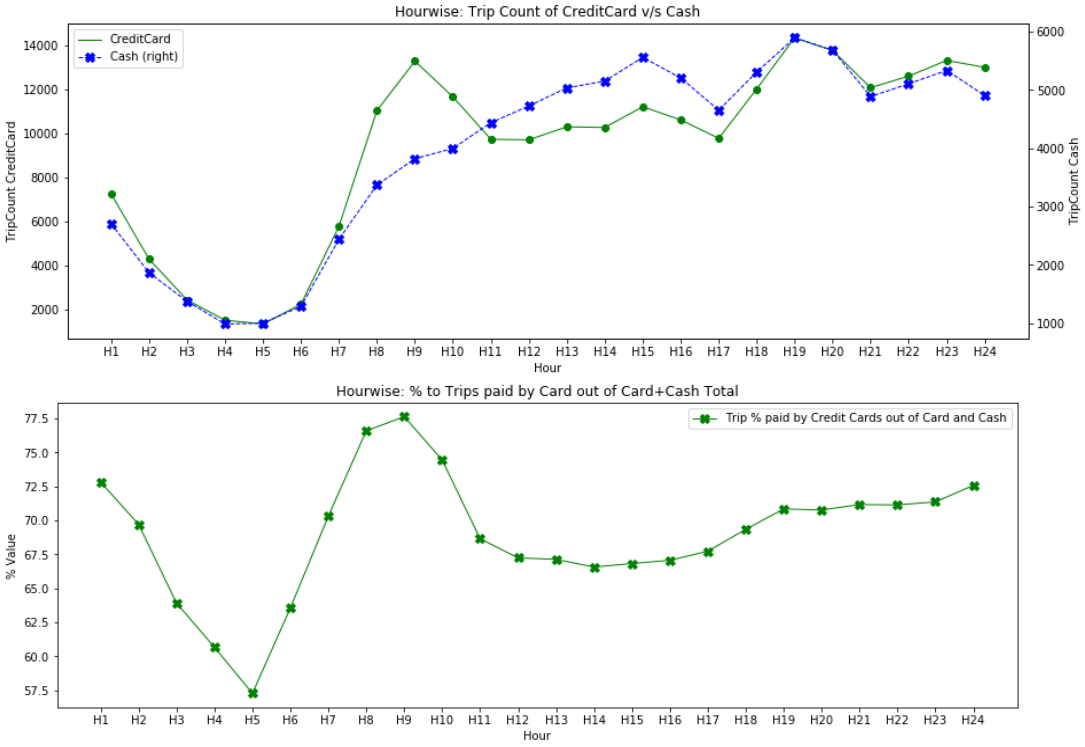
- Share of credit card payments as against cash is maximum from Tuesdays to Thursdays and Lowest on weekends
- 3 out 4 Saturdays rank highest in the week for number of trips paid for with cash during a week





Plot 7: Hour wise variation on 01.06: Number of trips paid for by Credit Card v/s Cash
Plot 8: Hour wise variation on 01.06: % to Trips paid by Card out of Card+Cash Total

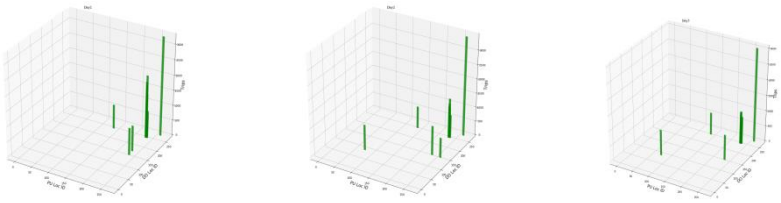
- H5 to H9: Ratio of Card payment against trips paid for by only Cash i.e Card payment continued to increase steadily from approx 60% to 80%
- H9 to H17: The overall use dropped to about 65% by H12 and remained steady till H17
- H17 to H24: Increased slightly till H19 and remained steady at approx. 72%



Plot 9: Daily Top 10 demand routes 28 days - PU to DO Location

- PU Loc 264 to DO Loc 264 appears within top 10 each day and it has the highest demand daily with over 3000 trips per day
- Trips to and fro for combinations of LocID's 236 and 237 feature consistently in the next top 4 routes by demand
- Except on Saturday's and Sunday's, all the top 10 locations ID's of PU to DO combo are concentrated around the 200 locations

There was a GIF which showed the daily variation by cycling through the plots for 28 DAYS data. As the PDF report will not support the GIF attached below are three sample images. Please open the link for the document on google drive to see the running GIF file. Link to google drive document: https://docs.google.com/document/d/10vX5QoYbKUnLSj1bU1SzATlTtqGUT6LednKgAYoKKeo/edit?usp=sharing



Plot 10: Hourly Top 5 in demand routes on 01.06 - PU to DO Location

- H1 to H22: The top 5 PU to DO locations are almost the same throughout
- Location 264: PU Loc 264 to DO Loc 264 appears in 23 out of the 24 hours within top 5 list. Also, for 17 of these hours it has the highest number of trips.
- Variability in trip count for top most PU-DO combination is from low 20's (H4 to H5) and peaks around 225 range (H22 to H24)

0000-0100 hrs

```
PULoc 48 to DOLoc 68 = 26 trips
PULoc 79 to DOLoc 79 = 31 trips
PULoc 249 to DOLoc 79 = 27 trips
PULoc 79 to DOLoc 107 = 25 trips
PULoc 264 to DOLoc 264 = 110 trips
```

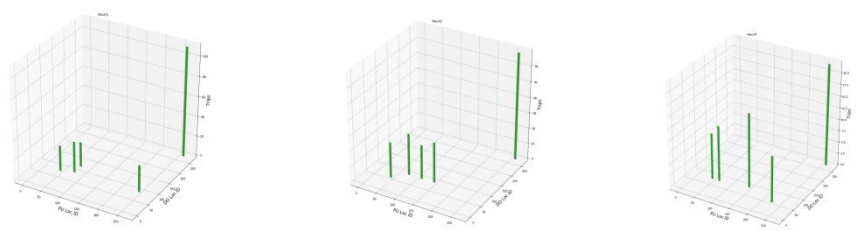
0100-0200 hrs

```
PULoc 48 to DOLoc 48 = 23 trips
PULoc 79 to DOLoc 79 = 27 trips
PULoc 114 to DOLoc 79 = 22 trips
PULoc 148 to DOLoc 79 = 26 trips
PULoc 264 to DOLoc 264 = 69 trips
```

2300-0000

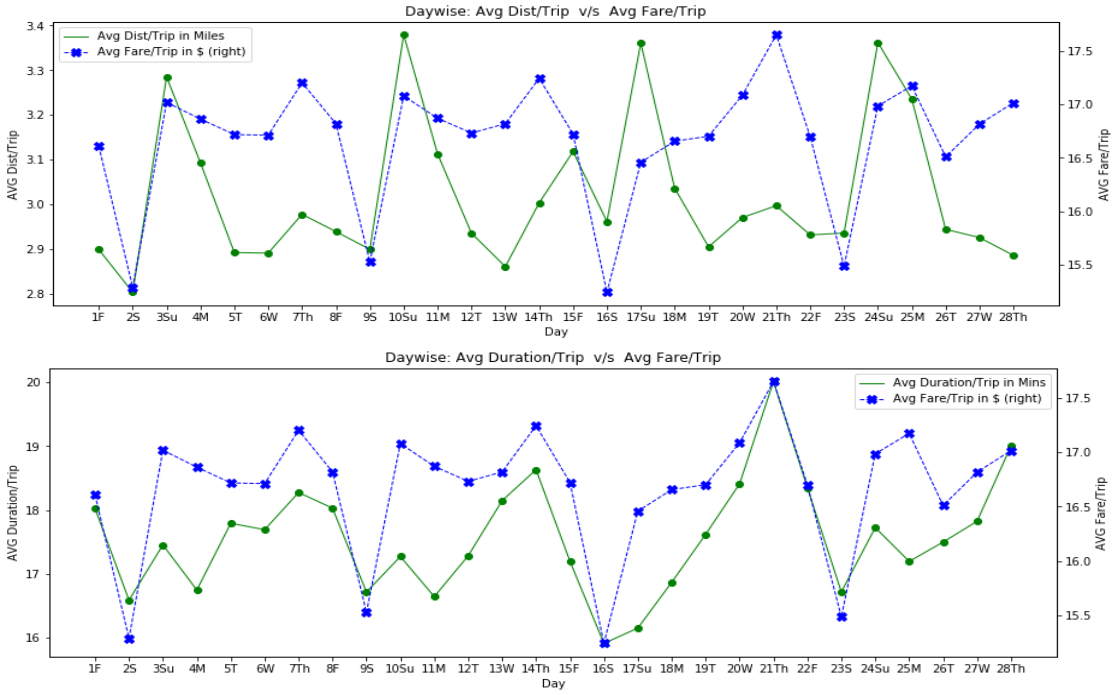
```
PULoc 48 to DOLoc 48 = 54 trips
PULoc 48 to DOLoc 68 = 60 trips
PULoc 79 to DOLoc 79 = 50 trips
PULoc 79 to DOLoc 148 = 52 trips
PULoc 264 to DOLoc 264 = 228 trips
```

There was a GIF which showed the daily variation by cycling through the plots for 24 HOURS data. As the PDF report will not support the GIF attached below are three sample images. Please open the link for the document on google drive to see the running GIF file.
Link to google drive document:
https://docs.google.com/document/d/10vX5QoYbKUnLSj1bU1SzATlTtqGUT6LednKgAYoKKeo/edit?usp=sharing



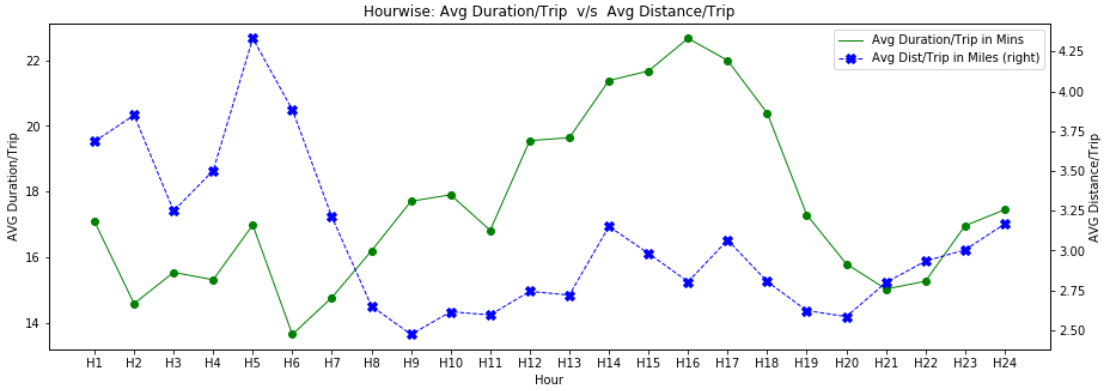
Plots 11, 12: Daily variation of Fare/Trip - the effect of Duration/Trip and Distance/Trip for 28 days of June

- Avg Fare/trip evinces clear pattern in first two weeks of June with troughs on Saturday, peaks on Sunday and Thursday.
- Avg Distance is consistently high on Sundays and that impacts the Avg fare making it high even though the Avg Duration is on lower side.
- On Thursday the high Avg Duration has same effect and low Avg Distance has limited impact
- On 17th as duration remained low the regular peak on Sunday was subdued even with high Avg Dist, the 21st Thur the fare is really high as duration goes higher than normal



Plots 13: Hourly variation of Average Duration/Trip and Distance/Trip on 01.06

- During early hours (H2 to H7) duration per trip remains low while the distance per trip remains high - supports low congestion during dead of night.
- H6 to H17, congestion picks up as distance per trip falls to a low steady value but duration per trip continues to increase.
- H17 onwards duration keeps falling as distance falls again.



5 - Challenges and Learnings

a) The CSV file had over 8 million rows and opening it in Excel truncated and showed only around first one million rows. Visual inspection indicated that the rows towards the start of the file were for 01.06.2018 and spanned row #3 (with PUdt as "01-06-2018 00:15:40") till probably around row #325000. But the data was not sorted on the PUdt and it would not be fool proof to assume a 01.06.2018 date data would not appear in the truncated portion of the file.
Sorting on the PUdt showed there were records with dates of May 2018 and a few stray records from 2008 and 2009 too.

A1 | tpep_pickup_datetime

	A	B	C
1	tpep_pickup_datetir	tpep_dropoff_datetime	
2	31-12-2008 13:51	31-12-2008 14:21	
3	01-01-2009 17:22	01-01-2009 18:05	
4	31-05-2018 06:08	31-05-2018 06:18	
5	31-05-2018 06:33	31-05-2018 06:35	
6	31-05-2018 06:40	31-05-2018 06:45	

Therefore, we had to implement a program based check in the producer python script to read the whole CSV file but to only process rows who's PUdt falls within our predefined boundary conditions (hardcoded in the program). Only these rows will be sent to the Kafka pipeline.

b) We took time to understand the parameters of Kafka consumer such as effective use of group_id, auto_offset_reset, consumer_timeout_ms and enable_auto_commit. Not been able to use either the synchronous or an asynchronous type commit issued explicitly from the consumer. Therefore, we are relying purely on auto-commit being enabled and reducing the default auto_commit_interval_ms from 5 seconds to 2 seconds.

c) Mongo query building is arcane, takes effort and care especially those involving aggregate queries with sections for $match, $project, $group, $limit and $sort. E.g. the one to "find top 5 COMBINATION OF Pick Up TO Drop Off locations by count".

pipeline=[{"$match": {"tpep_pickup_datetime": {"$gte": "2018-06-01 00:00:00", "$lt": "2018-06-02 00:00:00"}}} ,{"$project": { "PUDOCombo": { "$concat": [{"$substr":["$PULocationID", 0, -1]}, " TO " , {"$substr":["$DOLocationID", 0, -1]}]}}} ,{"$group":{"_id": "$PUDOCombo", "countPUtoDOLocTrips": {"$sum": 1}}} ,{"$sort": {"countPUtoDOLocTrips": -1}},{"$limit": 5}]
pipeline[0]['$match']['tpep_pickup_datetime']['$gte'] = timeBoundLower
pipeline[0]['$match']['tpep_pickup_datetime']['$lt'] = timeBoundUpper
cursorTop5ComboDOtoPULocsByCount = collection.aggregate(pipeline)

There is the other query below to find the duration of the trip that involves first converting the the PUdt and DOdt fields from string to ISO Date format, then finding the different, dividing to convert to minutes and then grouping using average.

pipeline=pipeline = [{"$match": {"tpep_pickup_datetime": {"$gte": "2018-06-01 00:00:01", "$lt": "2018-06-01 00:00:02"}}}, {"$project": {"_id":0, "TripDurationInMins": { "$divide": [{"$subtract": [{ "$dateFromString": {"dateString": "$tpep_dropoff_datetime"} } , { "$dateFromString": {"dateString": "$tpep_pickup_datetime"} }]} , 60000] } } }, {"$group": { "_id": 1, "AvgTripDurationInMins": {"$avg": "$TripDurationInMins"} } }]
pipeline[0]['$match']['tpep_pickup_datetime']['$gte'] = timeBoundLower
pipeline[0]['$match']['tpep_pickup_datetime']['$lt'] = timeBoundUpper
cursorAvgDurationPerTrip = collection.aggregate(pipeline)

d) During testing it was found that if the trip distance was 0 then it was passed in by our consumer as .00 and created an error while building the JSON for MongoDb write. So special code was inserted to handle this situation.

```
52        msgAsList = msgAsString.split(",")
53        trip_distanceAsString = str(msgAsList[4])
54        if trip_distanceAsString[0:1] == '.':       # a 0 value comes as  .00 | so checking and making as 0.00
55            trip_distanceAsString = '0' + trip_distanceAsString
```

e) An attempt was made to analyse the data streamed in kafka using spark. There are some hurdles for integrating the kafka server and spark. Even after spark installations were completed there were issues with the spark initializations.
We wanted to show a real time analysis to display :

- The trip with highest fare streamed thus far
- The trip with highest distance.
- The running count of the trips originating with a certain pick up location (PU) id eg. Location id 264 has 80 trips so far.

Since, we were unable to start Spark, we could not perform the above mentioned analysis.

6 - Bibliography

1) NYC Taxi related site
http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
http://www.nyc.gov/html/tlc/html/home/home.shtml
2) Data dictionary for Yellow Taxi file
http://www.nyc.gov/html/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf
3) Kafka and Zookeeper Related Sites
https://towardsdatascience.com/kafka-python-explained-in-10-lines-of-code-800e3e07dad1

https://www.programcreek.com/python/example/98440/kafka.KafkaConsumer
https://blog.workwell.io/how-to-manage-your-kafka-consumers-from-the-producer-9933b88085dd
https://medium.com/@Ankitthakur/apache-kafka-installation-on-mac-using-homebrew-a367cdefd273
4) Mongo Related Sites
https://stackoverflow.com/questions/17053323/how-to-order-mongodb-aggregation-with-match-sort-and-limit
http://www.forwardadvance.com/course/mongo/mongo-aggregation/aggregation-count
https://docs.mongodb.com/manual/reference/operator/aggregation/dateFromString/
https://stackoverflow.com/questions/41564510/convert-string-date-to-timestamp-in-mongodb
5) Python related
https://realpython.com/python-csv/
https://realpython.com/working-with-large-excel-files-in-pandas/
http://zetcode.com/python/pymongo/
https://www.geeksforgeeks.org/python-string-split/
https://api.mongodb.com/python/current/api/pymongo/mongo_client.html#pymongo.mongo_client.MongoClient
https://www.youtube.com/watch?v=WX0MDddgpA4
https://www.youtube.com/watch?v=w9tAoscq3C4
6) Spark Implementation
https://medium.com/@GalarnykMichael/install-spark-on-windows-pyspark-4498a5d8d66c
https://medium.com/@mukeshkumar_46704/getting-streaming-data-from-kafka-with-spark-streaming-using-python-9cd0922fa904
https://spark.apache.org/downloads.html
https://www.youtube.com/watch?v=l8jMvfOU3vQ
http://spark.apache.org/docs/latest/building-spark.html#setting-up-mavens-memory-usage
7) Miscellaneous
https://gifmaker.me/