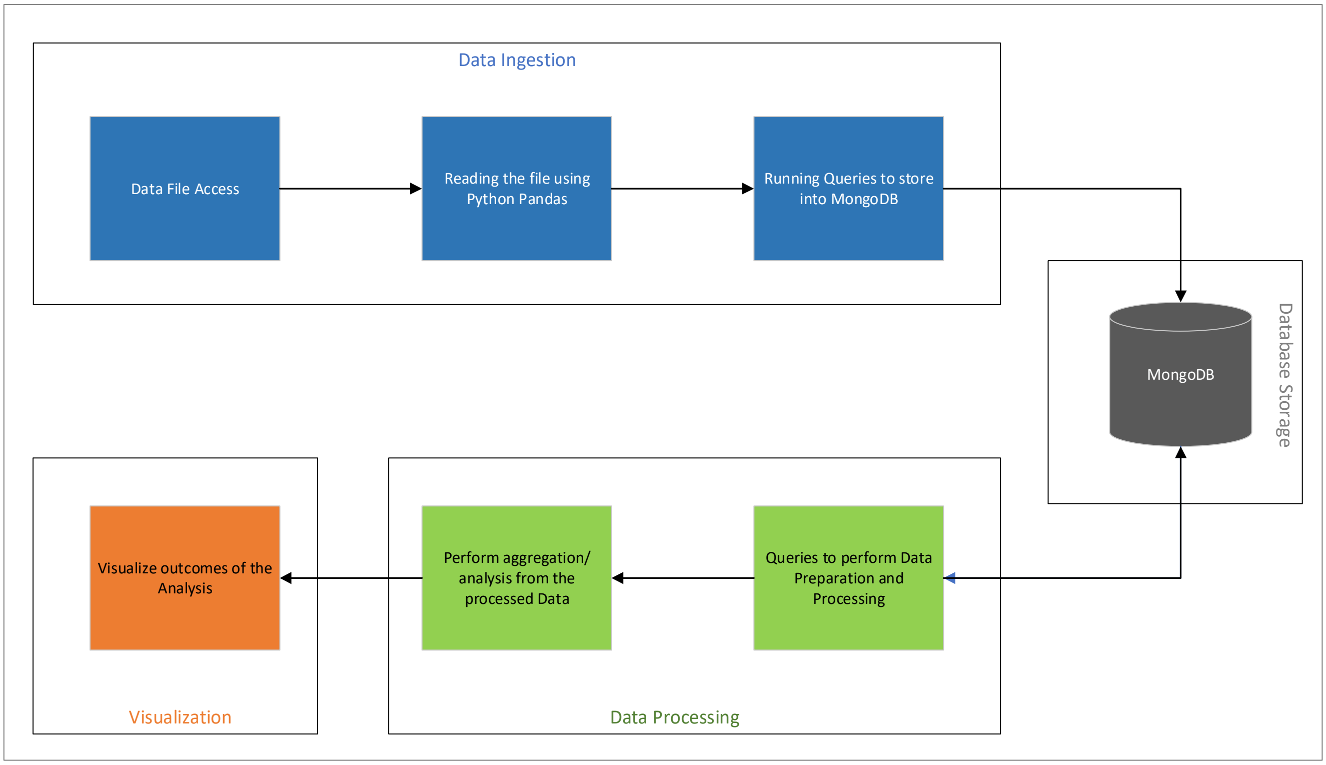
DATA ENGINEERING – DATA PIPELINE AND PROCESSING/VISUALIZATION PROJECT

Topic/Dataset: NYC TAXI TRIPS DATASET

Project Team:
Chinmoy Sarangi (Matriculation No.: 11012375)
Gaurang Sharma (Matriculation No.: 11012446)

Basic Architecture Model

Contributor: *Chinmoy Sarangi*


Data Ingestion
Data File Access
Reading the file using Python Pandas
Running Queries to store into MongoDB
MongoDB
Database Storage
Visualize outcomes of the Analysis
Perform aggregation/ analysis from the processed Data
Queries to perform Data Preparation and Processing
Visualization
Data Processing

Tools used to implement the Data Pipeline and Visualize the Analysis:

Database: *MongoDB*

Programming Language: *Python programming with Jupyter Notebook*

Libraries implemented: *Pandas, PyMongo, Bokeh, Matplotlib, Seaborn, DataShader, etc*

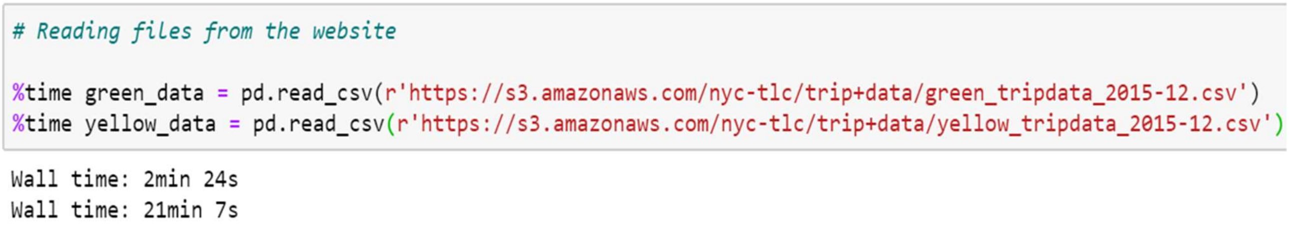
Data Ingestion:
Contributor: *Chinmoy Sarangi*

Steps Taken:

The trips files were first attempted to be read directly from the NYC Taxi trips dataset page. This was done to ensure that we were always reading correct data. This is because, in the past, the NYC taxi commission has made corrections to the datasets to fix certain attributes or the data entries and introduce some new features due to addition of new surcharges.

The file access was achieved via Python Pandas which provided us ample amount of information on how the data looked like, without having to access or store the file locally, hence saving disk space. However, we quickly realized that the file sizes were huge, and it took excessively long time to read the data.

Below are the wall times taken when attempting to read directly from the website. The code and time taken to read the files online are as follows:

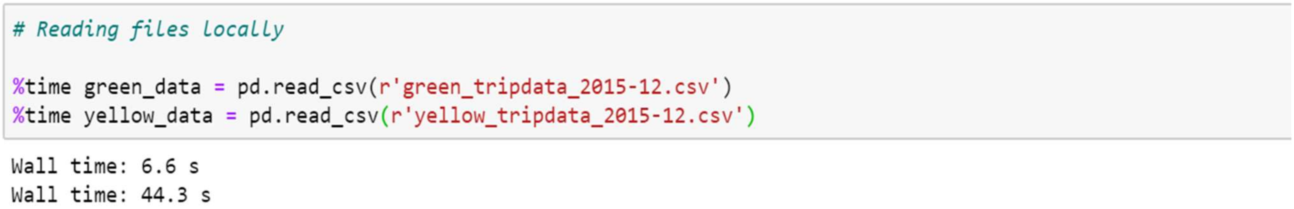


```
# Reading files from the website

%time green_data = pd.read_csv(r'https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_2015-12.csv')
%time yellow_data = pd.read_csv(r'https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2015-12.csv')
```

```
Wall time: 2min 24s
Wall time: 21min 7s
```

.

So, we stored the file locally and the results were dramatically different. The file reading was faster and was done in no time.



```
# Reading files locally

%time green_data = pd.read_csv(r'green_tripdata_2015-12.csv')
%time yellow_data = pd.read_csv(r'yellow_tripdata_2015-12.csv')
```

```
Wall time: 6.6 s
Wall time: 44.3 s
```

The different wall timings (i.e. the overall time taken for execution of that statement) are due to the size of the datasets.

So, it was clear that the pipeline would have to be implemented with locally stored data in order to reduce the difficulties faced in terms of delayed data import and subsequent rollover on to other processes that follow.

Since the dataset for yellow taxi trips was humongous, hence we chose to go with green taxi trips data, which was significantly smaller in size and easier to manage throughout the rest of the process, right until the visualization stage.

Data Storage in MongoDB:
Contributor: *Chinmoy Sarangi & Gaurang Sharma*

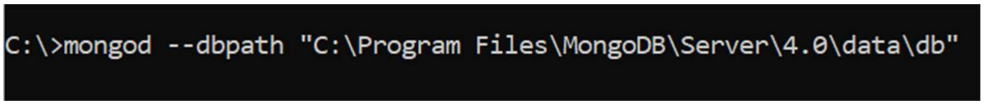
Steps Taken:

The data derived from the Python Pandas operation was then exported into the MongoDB using the PyMongo programming.

1. Initializing MongoDB server and MongoDB Community Compass

Contributor: *Chinmoy Sarangi*

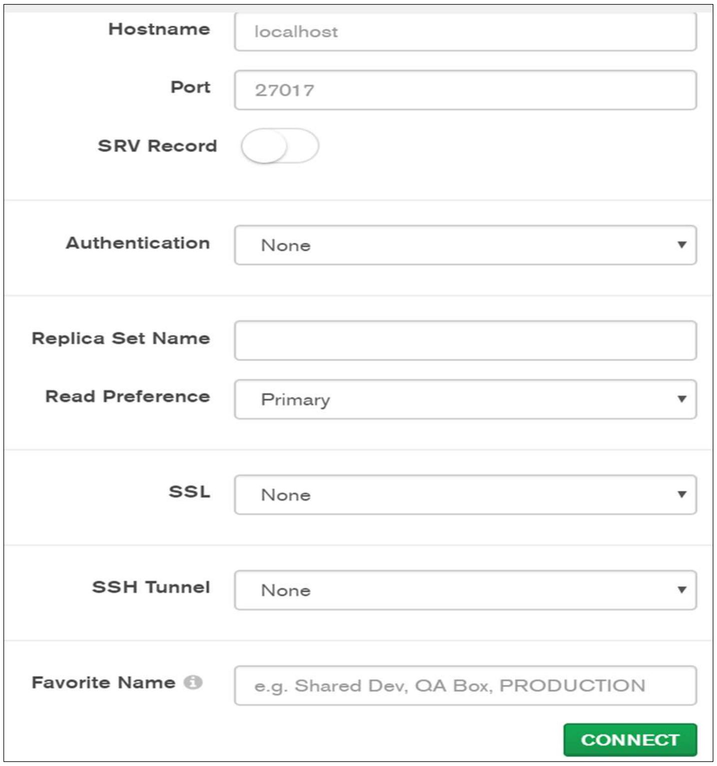
The Mongo instance is executed from the command prompt, which is run with administrator privileges.



```
C:\>mongod --dbpath "C:\Program Files\MongoDB\Server\4.0\data\db"
```

The above command line activates mongo with **hostname**: localhost and **port number**: 27017.

After this, launch Mongo Compass Community Edition installed on the system. Below screen with details in grey will open up:



Hostname: localhost

Port: 27017

SRV Record: ○

Authentication: None ▾

Replica Set Name:

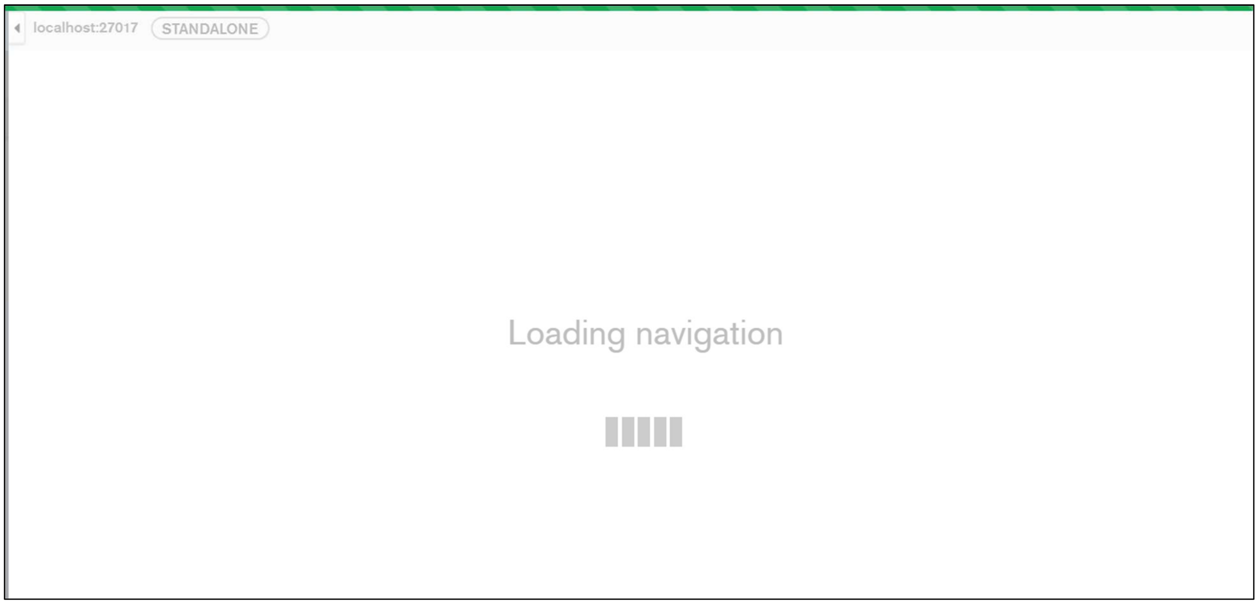
Read Preference: Primary ▾

SSL: None ▾

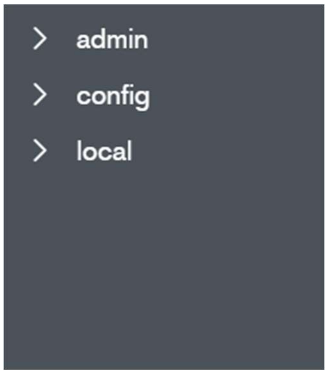
SSH Tunnel: None ▾

Favorite Name ⓘ: e.g. Shared Dev, QA Box, PRODUCTION

Click on **CONNECT.** This will initiate connection created with above command set.



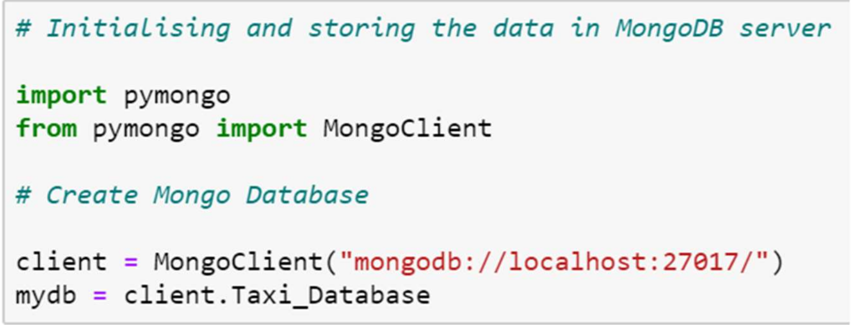
The screen will load with below 3 entries listed as default within the Mongo Compass console:



2. Creating Database and Database Collection:

Contributor: *Gaurang Sharma*

By using the PyMongo connection, 1000 rows of sample data were exported to MongoDB from the Pandas DataFrame. The Database name created is: **Taxi_Database**.

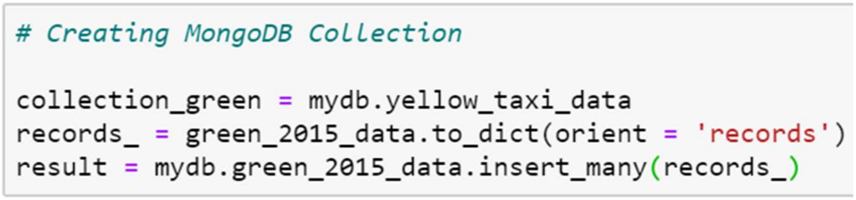


```
# Initialising and storing the data in MongoDB server

import pymongo
from pymongo import MongoClient

# Create Mongo Database

client = MongoClient("mongodb://localhost:27017/")
mydb = client.Taxi_Database
```

3. Inserting the Data into Collection:

Contributor: *Gaurang Sharma*

But, although a placeholder database may have been created, it does not get recognised until it has collections created within it. Collection name created is - **green_2015_data**.

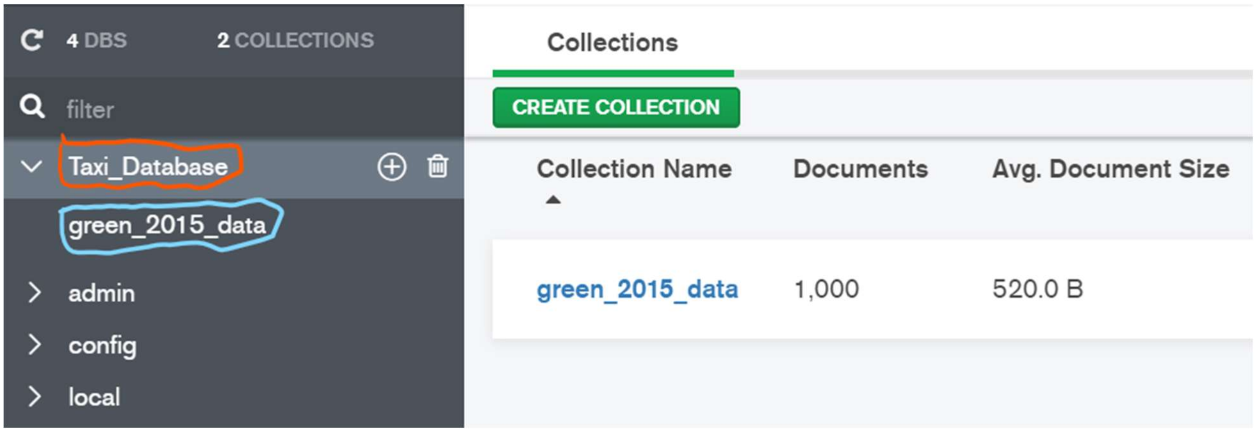


```
# Creating MongoDB Collection

collection_green = mydb.yellow_taxi_data
records_ = green_2015_data.to_dict(orient = 'records')
result = mydb.green_2015_data.insert_many(records_)
```

4. Verifying and Checking Database

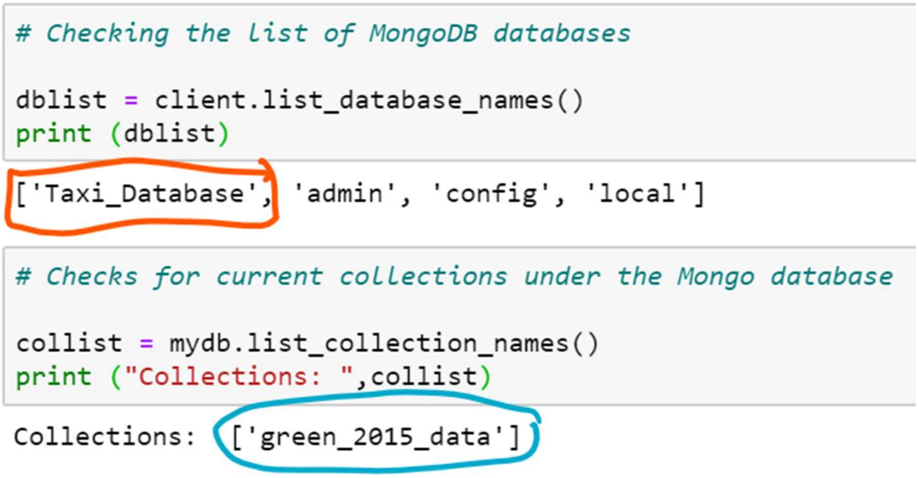
Contributor: *Chinmoy Sarangi*

To ensure that the Databases were indeed created, the MongoDB Compass Console was refreshed to verify the same. As can be seen, the above Database and the associated collection for the green taxi trips have been created with 1000 records.



Collection Name	Documents	Avg. Document Size
green_2015_data	1,000	520.0 B

To make sure it is created from Jupyter console, following commands can be run in order to verify the Database and the Collection that were created in the Mongo console earlier:



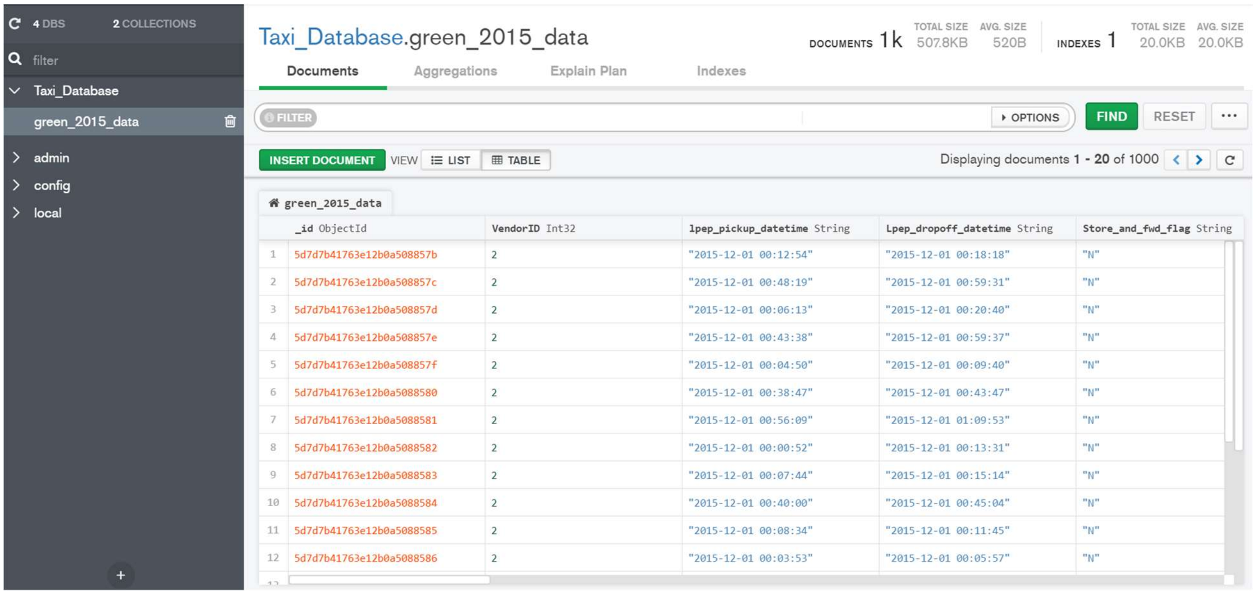
```
# Checking the list of MongoDB databases

dblist = client.list_database_names()
print (dblist)
```

```
['Taxi_Database', 'admin', 'config', 'local']
```

```
# Checks for current collections under the Mongo database

collist = mydb.list_collection_names()
print ("Collections: ",collist)
```

```
Collections: ['green_2015_data']
```

When we click on the collection within MongoDB Compass console, it gives us the data stored.



Taxi_Database.green_2015_data

DOCUMENTS 1k TOTAL SIZE 507.8KB AVG. SIZE 520B

INDEXES 1 TOTAL SIZE 20.0KB AVG. SIZE 20.0KB

Displaying documents 1 - 20 of 1000

green_2015_data

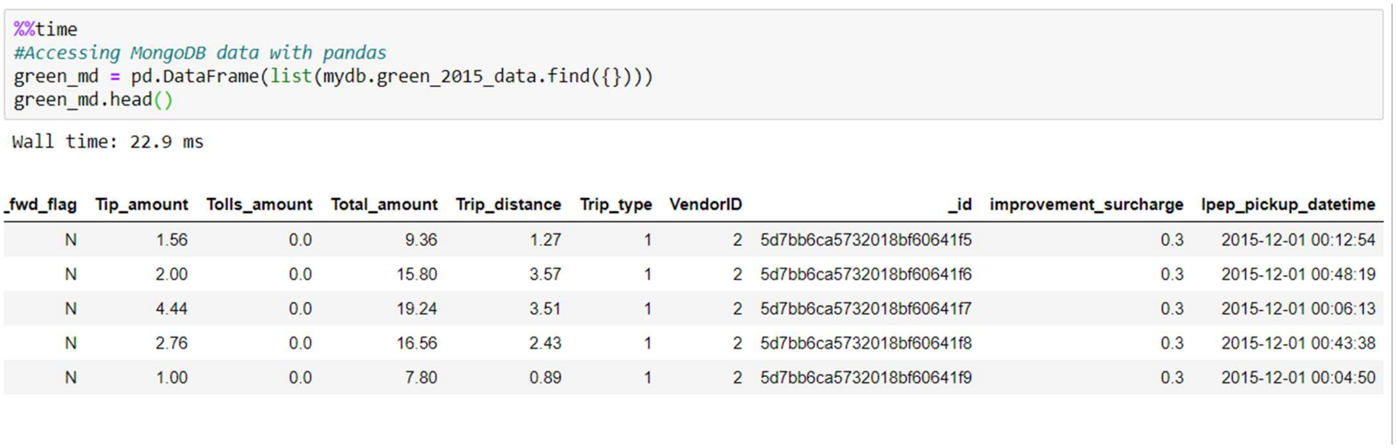
	_id ObjectId	VendorID Int32	lpep_pickup_datetime String	Lpep_dropoff_datetime String	Store_and_fwd_flag String
1	5d7d7b41763e12b0a508857b	2	"2015-12-01 00:12:54"	"2015-12-01 00:18:18"	"N"
2	5d7d7b41763e12b0a508857c	2	"2015-12-01 00:48:19"	"2015-12-01 00:59:31"	"N"
3	5d7d7b41763e12b0a508857d	2	"2015-12-01 00:06:13"	"2015-12-01 00:20:40"	"N"
4	5d7d7b41763e12b0a508857e	2	"2015-12-01 00:43:38"	"2015-12-01 00:59:37"	"N"
5	5d7d7b41763e12b0a508857f	2	"2015-12-01 00:04:50"	"2015-12-01 00:09:40"	"N"
6	5d7d7b41763e12b0a5088580	2	"2015-12-01 00:38:47"	"2015-12-01 00:43:47"	"N"
7	5d7d7b41763e12b0a5088581	2	"2015-12-01 00:56:09"	"2015-12-01 01:09:53"	"N"
8	5d7d7b41763e12b0a5088582	2	"2015-12-01 00:00:52"	"2015-12-01 00:13:31"	"N"
9	5d7d7b41763e12b0a5088583	2	"2015-12-01 00:07:44"	"2015-12-01 00:15:14"	"N"
10	5d7d7b41763e12b0a5088584	2	"2015-12-01 00:40:00"	"2015-12-01 00:45:04"	"N"
11	5d7d7b41763e12b0a5088585	2	"2015-12-01 00:08:34"	"2015-12-01 00:11:45"	"N"
12	5d7d7b41763e12b0a5088586	2	"2015-12-01 00:03:53"	"2015-12-01 00:05:57"	"N"

Data Processing:
Contributor: *Gaurang Sharma & Chinmoy Sarangi*

Steps Taken:

As the data is stored in MongoDB now, we use the data further for Processing and Analyzing.

1. Accessing the store data with Python.
Contributor: *Gaurang Sharma*



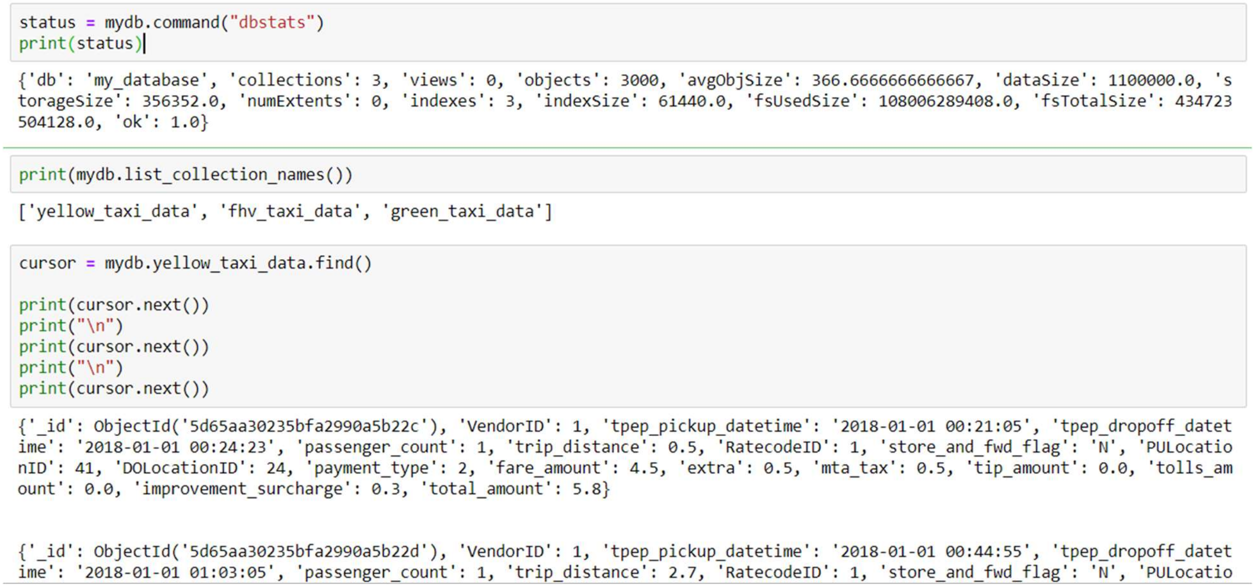
```
%%time
#Accessing MongoDB data with pandas
green_md = pd.DataFrame(list(mydb.green_2015_data.find({})))
green_md.head()
```

```
Wall time: 22.9 ms
```

_fwd_flag	Tip_amount	Tolls_amount	Total_amount	Trip_distance	Trip_type	VendorID	_id	improvement_surcharge	lpep_pickup_datetime
N	1.56	0.0	9.36	1.27	1	2	5d7bb6ca5732018bf60641f5	0.3	2015-12-01 00:12:54
N	2.00	0.0	15.80	3.57	1	2	5d7bb6ca5732018bf60641f6	0.3	2015-12-01 00:48:19
N	4.44	0.0	19.24	3.51	1	2	5d7bb6ca5732018bf60641f7	0.3	2015-12-01 00:06:13
N	2.76	0.0	16.56	2.43	1	2	5d7bb6ca5732018bf60641f8	0.3	2015-12-01 00:43:38
N	1.00	0.0	7.80	0.89	1	2	5d7bb6ca5732018bf60641f9	0.3	2015-12-01 00:04:50

Loaded the data from the database to Pandas DataFrame as it is easier to process and analyze with Pandas.

2. Ran some MongoDB queries with PyMongo.
Contributor: *Gaurang Sharma & Chinmoy Sarangi*



```
status = mydb.command("dbstats")
print(status)
```

```
{'db': 'my_database', 'collections': 3, 'views': 0, 'objects': 3000, 'avgObjSize': 366.6666666666667, 'dataSize': 1100000.0, 's
torageSize': 356352.0, 'numExtents': 0, 'indexes': 3, 'indexSize': 61440.0, 'fsUsedSize': 108006289408.0, 'fsTotalSize': 434723
504128.0, 'ok': 1.0}
```

```
print(mydb.list_collection_names())
```

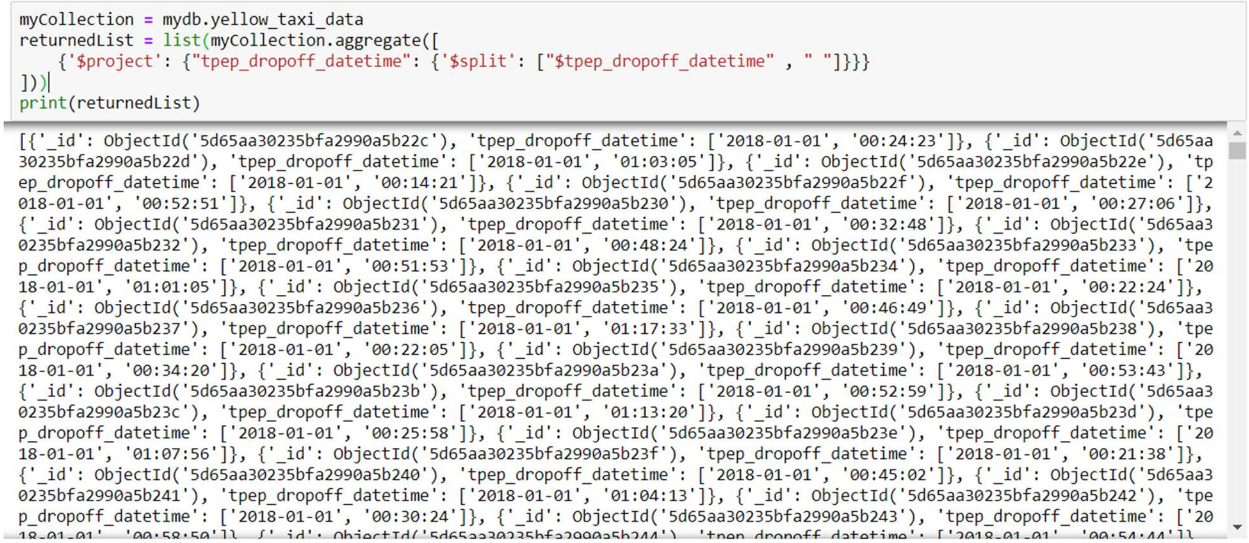
```
['yellow_taxi_data', 'fhv_taxi_data', 'green_taxi_data']
```

```
cursor = mydb.yellow_taxi_data.find()

print(cursor.next())
print("\n")
print(cursor.next())
print("\n")
print(cursor.next())
```

```
{'_id': ObjectId('5d65aa30235bfa2990a5b22c'), 'VendorID': 1, 'tpep_pickup_datetime': '2018-01-01 00:21:05', 'tpep_dropoff_datet
ime': '2018-01-01 00:24:23', 'passenger_count': 1, 'trip_distance': 0.5, 'RatecodeID': 1, 'store_and_fwd_flag': 'N', 'PULocatio
nID': 41, 'DOLocationID': 24, 'payment_type': 2, 'fare_amount': 4.5, 'extra': 0.5, 'mta_tax': 0.5, 'tip_amount': 0.0, 'tolls_am
ount': 0.0, 'improvement_surcharge': 0.3, 'total_amount': 5.8}


{'_id': ObjectId('5d65aa30235bfa2990a5b22d'), 'VendorID': 1, 'tpep_pickup_datetime': '2018-01-01 00:44:55', 'tpep_dropoff_datet
ime': '2018-01-01 01:03:05', 'passenger_count': 1, 'trip_distance': 2.7, 'RatecodeID': 1, 'store_and_fwd_flag': 'N', 'PULocatio
```

3. Tried 'split' query with PyMongo.
Contributor: *Gaurang Sharma*

We wanted to split date and time from the column, so to easily calculate trip time.

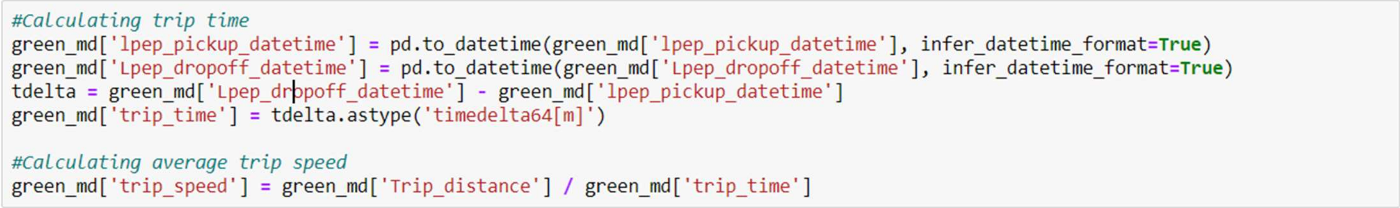


```
myCollection = mydb.yellow_taxi_data
returnedList = list(myCollection.aggregate([
    {'$project': {"tpep_dropoff_datetime": {'$split': ["$tpep_dropoff_datetime" , " "]}}}
]))
print(returnedList)
```

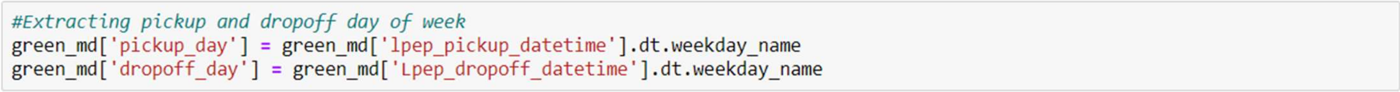
```
[{'_id': ObjectId('5d65aa30235bfa2990a5b22c'), 'tpep_dropoff_datetime': ['2018-01-01', '00:24:23']}, {'_id': ObjectId('5d65aa
30235bfa2990a5b22d'), 'tpep_dropoff_datetime': ['2018-01-01', '01:03:05']}, {'_id': ObjectId('5d65aa30235bfa2990a5b22e'), 'tp
ep_dropoff_datetime': ['2018-01-01', '00:14:21']}, {'_id': ObjectId('5d65aa30235bfa2990a5b22f'), 'tpep_dropoff_datetime': ['2
018-01-01', '00:52:51']}, {'_id': ObjectId('5d65aa30235bfa2990a5b230'), 'tpep_dropoff_datetime': ['2018-01-01', '00:27:06']},
{'_id': ObjectId('5d65aa30235bfa2990a5b231'), 'tpep_dropoff_datetime': ['2018-01-01', '00:32:48']}, {'_id': ObjectId('5d65aa3
0235bfa2990a5b232'), 'tpep_dropoff_datetime': ['2018-01-01', '00:48:24']}, {'_id': ObjectId('5d65aa30235bfa2990a5b233'), 'tpe
p_dropoff_datetime': ['2018-01-01', '00:51:53']}, {'_id': ObjectId('5d65aa30235bfa2990a5b234'), 'tpep_dropoff_datetime': ['20
18-01-01', '01:01:05']}, {'_id': ObjectId('5d65aa30235bfa2990a5b235'), 'tpep_dropoff_datetime': ['2018-01-01', '00:22:24']},
{'_id': ObjectId('5d65aa30235bfa2990a5b236'), 'tpep_dropoff_datetime': ['2018-01-01', '00:46:49']}, {'_id': ObjectId('5d65aa3
0235bfa2990a5b237'), 'tpep_dropoff_datetime': ['2018-01-01', '01:17:33']}, {'_id': ObjectId('5d65aa30235bfa2990a5b238'), 'tpe
p_dropoff_datetime': ['2018-01-01', '00:22:05']}, {'_id': ObjectId('5d65aa30235bfa2990a5b239'), 'tpep_dropoff_datetime': ['20
18-01-01', '00:34:20']}, {'_id': ObjectId('5d65aa30235bfa2990a5b23a'), 'tpep_dropoff_datetime': ['2018-01-01', '00:53:43']},
{'_id': ObjectId('5d65aa30235bfa2990a5b23b'), 'tpep_dropoff_datetime': ['2018-01-01', '00:52:59']}, {'_id': ObjectId('5d65aa3
0235bfa2990a5b23c'), 'tpep_dropoff_datetime': ['2018-01-01', '01:13:20']}, {'_id': ObjectId('5d65aa30235bfa2990a5b23d'), 'tpe
p_dropoff_datetime': ['2018-01-01', '00:25:58']}, {'_id': ObjectId('5d65aa30235bfa2990a5b23e'), 'tpep_dropoff_datetime': ['20
18-01-01', '01:07:56']}, {'_id': ObjectId('5d65aa30235bfa2990a5b23f'), 'tpep_dropoff_datetime': ['2018-01-01', '00:21:38']},
{'_id': ObjectId('5d65aa30235bfa2990a5b240'), 'tpep_dropoff_datetime': ['2018-01-01', '00:45:02']}, {'_id': ObjectId('5d65aa3
0235bfa2990a5b241'), 'tpep_dropoff_datetime': ['2018-01-01', '01:04:13']}, {'_id': ObjectId('5d65aa30235bfa2990a5b242'), 'tpe
p_dropoff_datetime': ['2018-01-01', '00:30:24']}, {'_id': ObjectId('5d65aa30235bfa2990a5b243'), 'tpep_dropoff_datetime': ['20
18-01-01', '00:58:50']}, {'_id': ObjectId('5d65aa30235bfa2990a5b244'), 'tpep_dropoff_datetime': ['2018-01-01', '00:54:44']},
```

After which we realized that processing is easier and better done with Pandas (datatype = DataFrame) rather than MongoDB (data type = json) queries.

4. Calculating Trip Time, Average Trip Speed & Extracting Day of the Week with Python.
Contributor: *Gaurang Sharma*

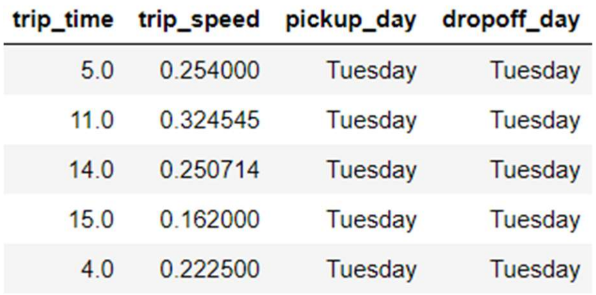


```
#Calculating trip time
green_md['lpep_pickup_datetime'] = pd.to_datetime(green_md['lpep_pickup_datetime'], infer_datetime_format=True)
green_md['Lpep_dropoff_datetime'] = pd.to_datetime(green_md['Lpep_dropoff_datetime'], infer_datetime_format=True)
tdelta = green_md['Lpep_dropoff_datetime'] - green_md['lpep_pickup_datetime']
green_md['trip_time'] = tdelta.astype('timedelta64[m]')

#Calculating average trip speed
green_md['trip_speed'] = green_md['Trip_distance'] / green_md['trip_time']
```



```
#Extracting pickup and dropoff day of week
green_md['pickup_day'] = green_md['lpep_pickup_datetime'].dt.weekday_name
green_md['dropoff_day'] = green_md['Lpep_dropoff_datetime'].dt.weekday_name
```

Result:



trip_time	trip_speed	pickup_day	dropoff_day
5.0	0.254000	Tuesday	Tuesday
11.0	0.324545	Tuesday	Tuesday
14.0	0.250714	Tuesday	Tuesday
15.0	0.162000	Tuesday	Tuesday
4.0	0.222500	Tuesday	Tuesday

Time is calculated in minutes and speed in m/

5. Description of each column.

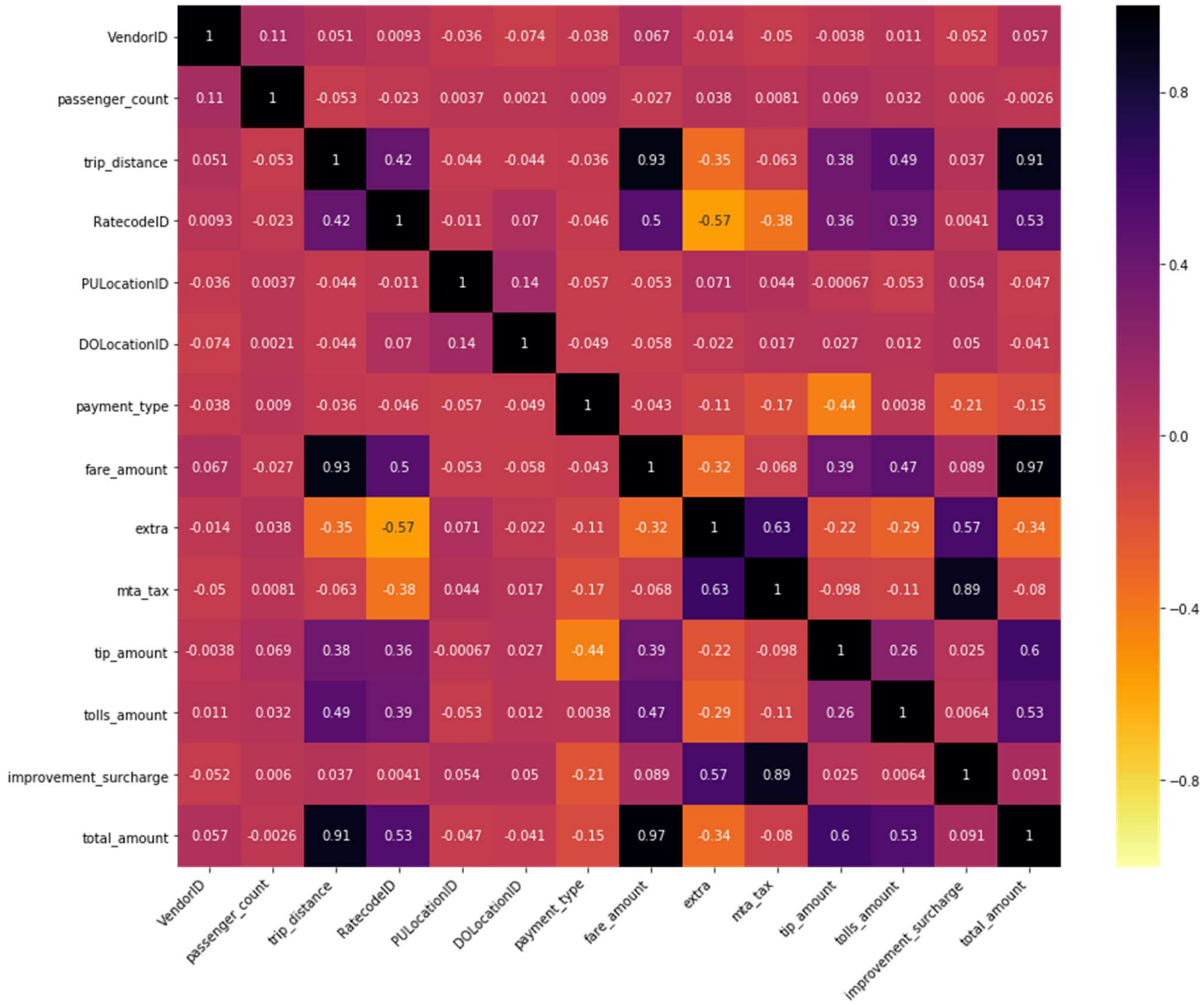
Contributor: *Chinmoy Sarangi & Gaurang Sharma*

DataFrame.describe() gives a description of every column, such as count, mean, std, min, max, etc.

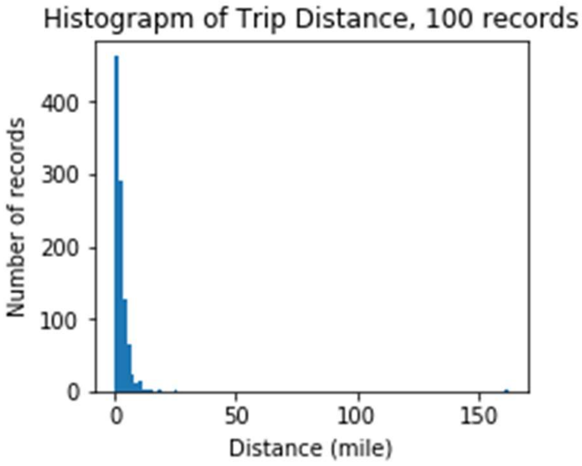
	Fare_amount	Passenger_count	Tip_amount	Tolls_amount	Total_amount	Trip_distance	trip_time	trip_speed
mean	10.982500	1.371000	0.941770	0.055400	13.236070	2.665120	17.502000	inf
min	-9.000000	0.00000	1.854485	0.551499	26.826616	5.570901	109.124141	NaN
max	826.000000	6.000000	36.000000	5.540000	827.300000	162.290000	1438.000000	inf

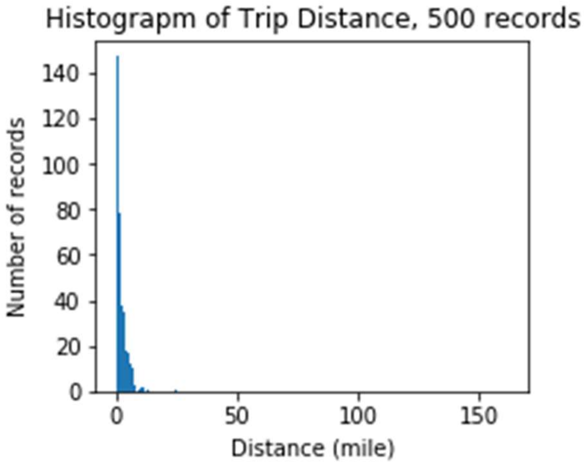
Data Visualization:
Contributor: *Gaurang Sharma*

1. Correlation between the columns.
Contributor: *Gaurang Sharma*



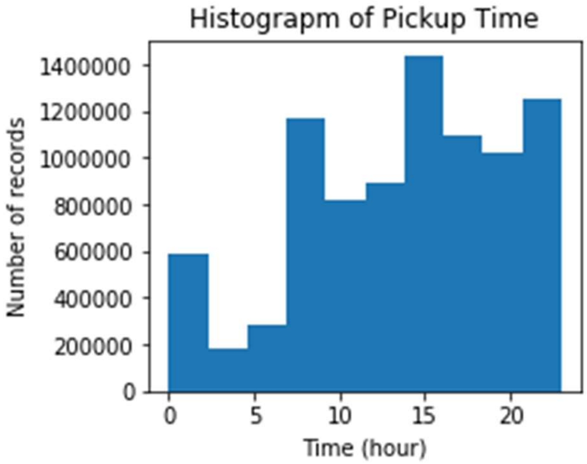
2. Trip Distance
Contributor: *Gaurang Sharma*

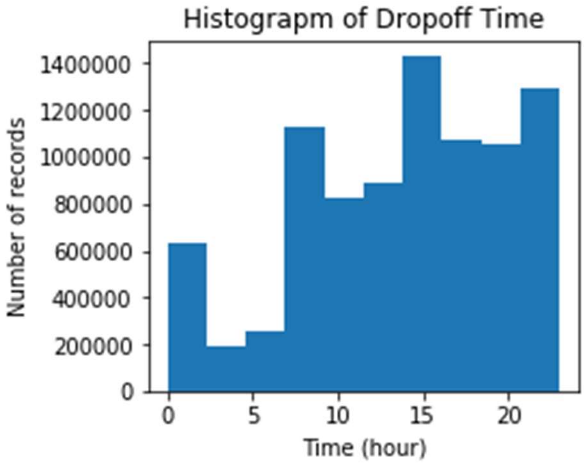




3. Time of Pickup & Drop-off Trips
Contributor: *Gaurang Sharma*

Extracted hour & minute of pickup and plotted it on a Histogram.

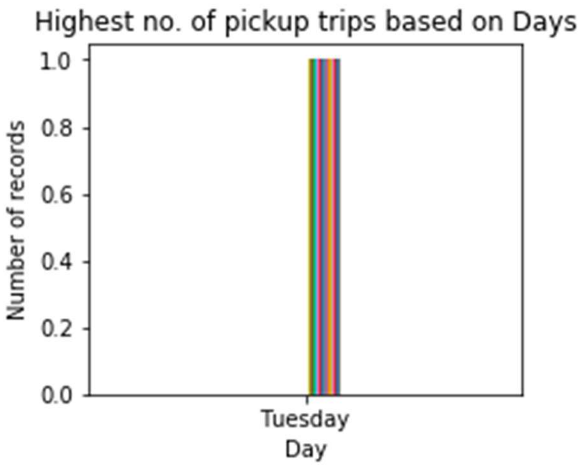


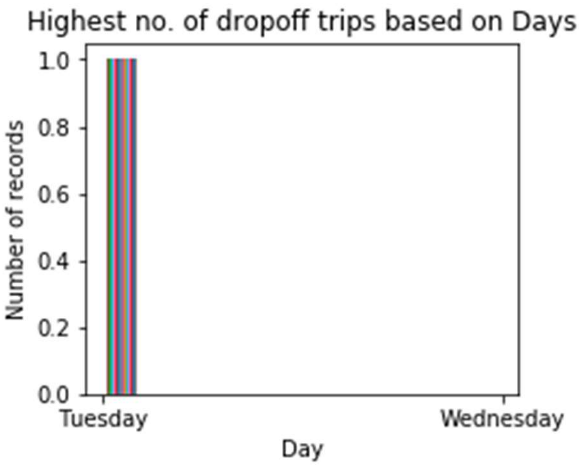


Hour 15 (3 pm) seems to be the rush hour.

4. Day of Pickup & Drop-off Trips

Contributor: *Gaurang Sharma*

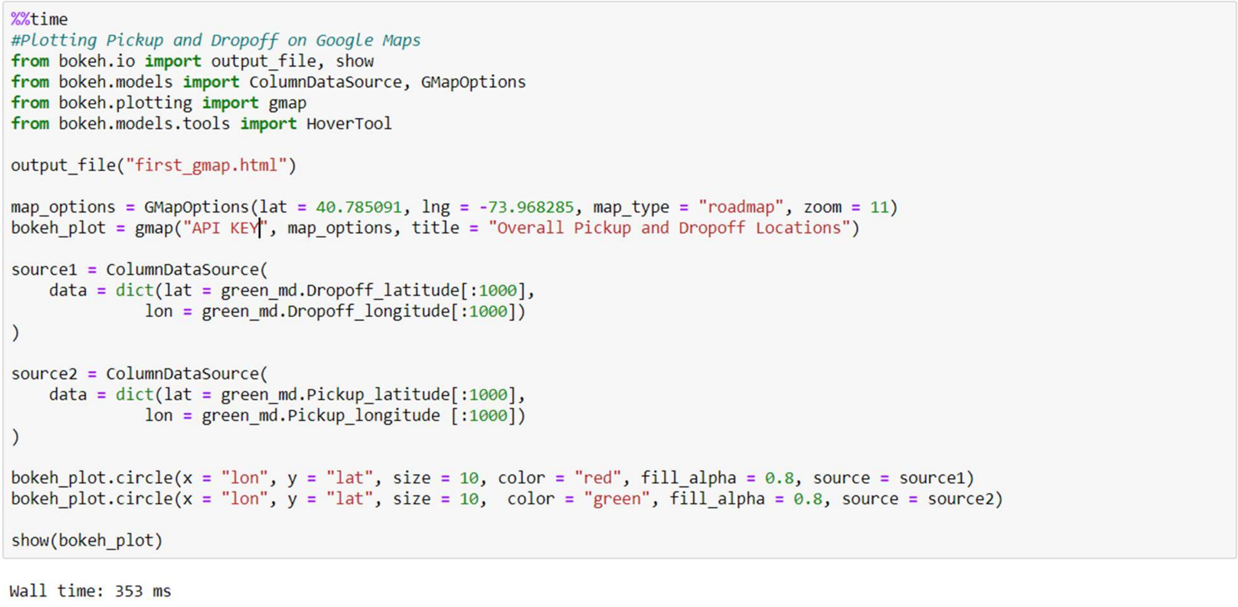




5. GeoMapping

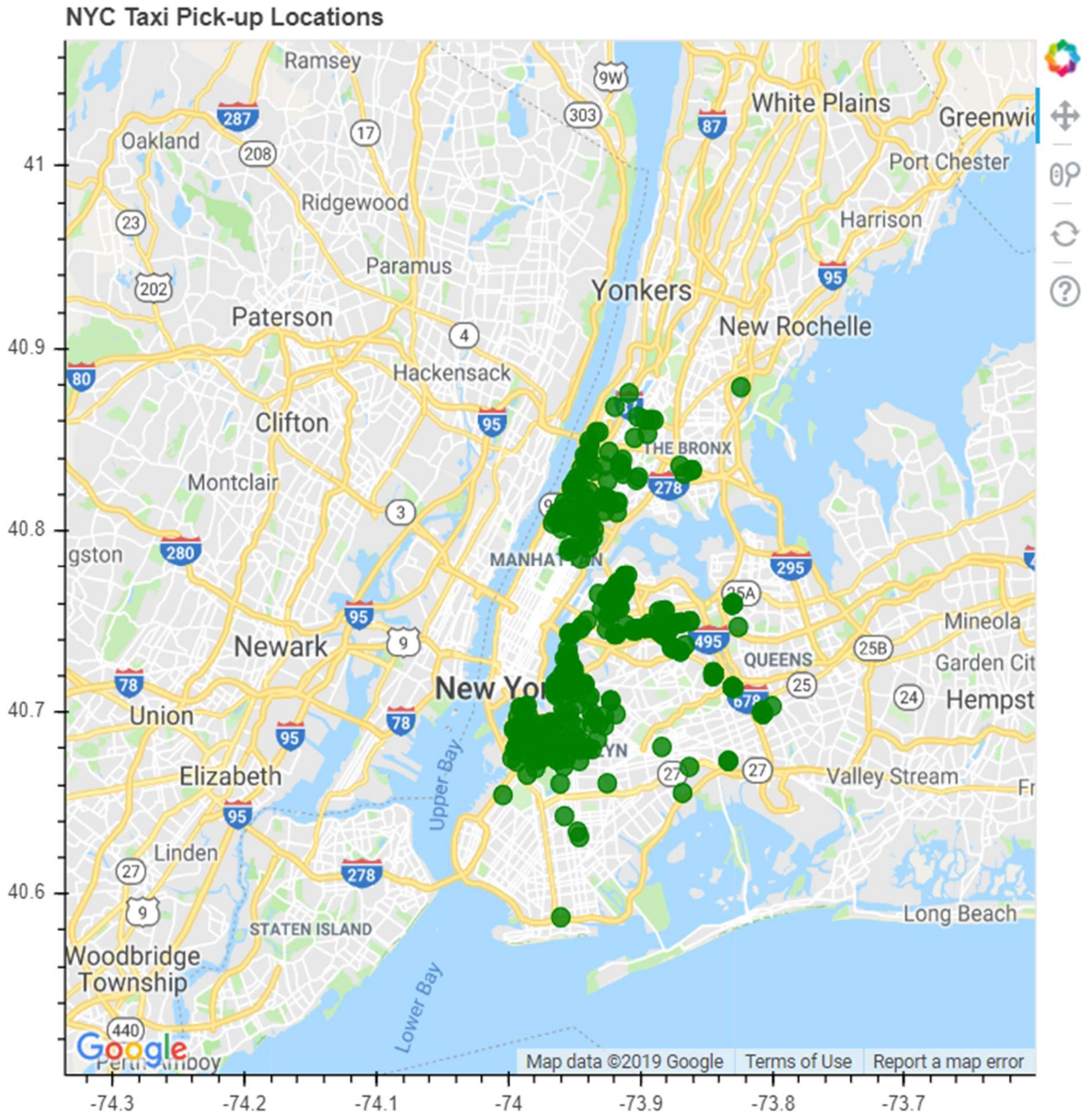
Contributor: *Gaurang Sharma*

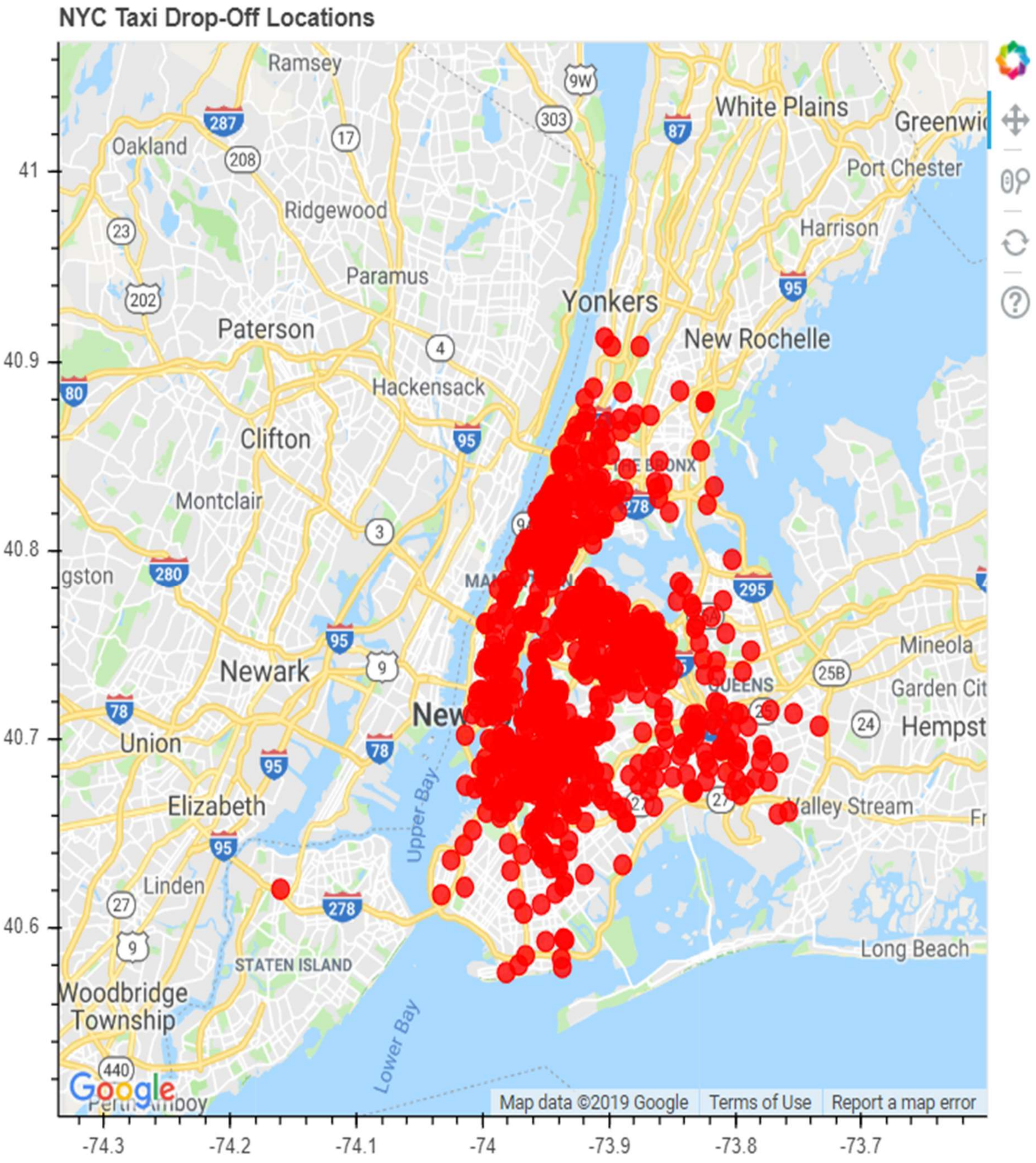
Library Bokeh along with Google API Key was used to map the pickup and drop-off taxi locations on Google Map.
1st Output shows all Pickup points, 2nd has all Drop-off points and 3rd has both included.



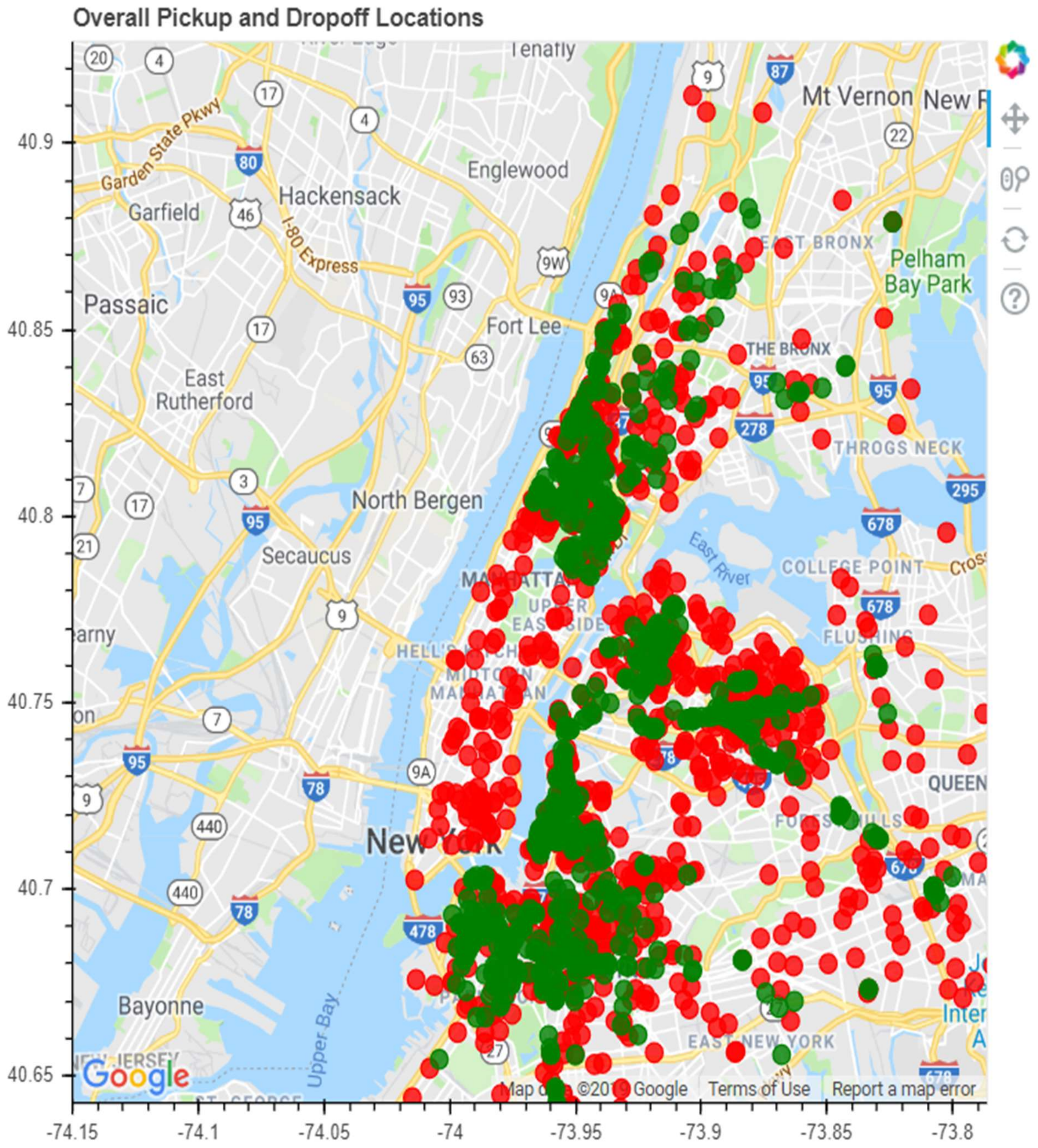
```
%%time
#Plotting Pickup and Dropoff on Google Maps
from bokeh.io import output_file, show
from bokeh.models import ColumnDataSource, GMapOptions
from bokeh.plotting import gmap
from bokeh.models.tools import HoverTool

output_file("first_gmap.html")

map_options = GMapOptions(lat = 40.785091, lng = -73.968285, map_type = "roadmap", zoom = 11)
bokeh_plot = gmap("API KEY", map_options, title = "Overall Pickup and Dropoff Locations")

source1 = ColumnDataSource(
    data = dict(lat = green_md.Dropoff_latitude[:1000],
              lon = green_md.Dropoff_longitude[:1000])
)

source2 = ColumnDataSource(
    data = dict(lat = green_md.Pickup_latitude[:1000],
              lon = green_md.Pickup_longitude [:1000])
)

bokeh_plot.circle(x = "lon", y = "lat", size = 10, color = "red", fill_alpha = 0.8, source = source1)
bokeh_plot.circle(x = "lon", y = "lat", size = 10,  color = "green", fill_alpha = 0.8, source = source2)

show(bokeh_plot)
```

```
Wall time: 353 ms
```


NYC Taxi Pick-up Locations
41
40.9
40.8
40.7
40.6
-74.3
-74.2
-74.1
-74
-73.9
-73.8
-73.7
Ramsey
Oakland
Ridgewood
Paramus
Paterson
Hackensack
Clifton
Montclair
Newark
Union
Elizabeth
Linden
Woodbridge Township
Perth Amboy
STATEN ISLAND
New York
MANHATTAN
THE BRONX
QUEENS
Yonkers
White Plains
New Rochelle
Port Chester
Harrison
Greenwich
Mineola
Garden City
Hempstead
Valley Stream
Long Beach
Upper Bay
Lower Bay
Map data ©2019 Google
Terms of Use
Report a map error



NYC Taxi Drop-Off Locations



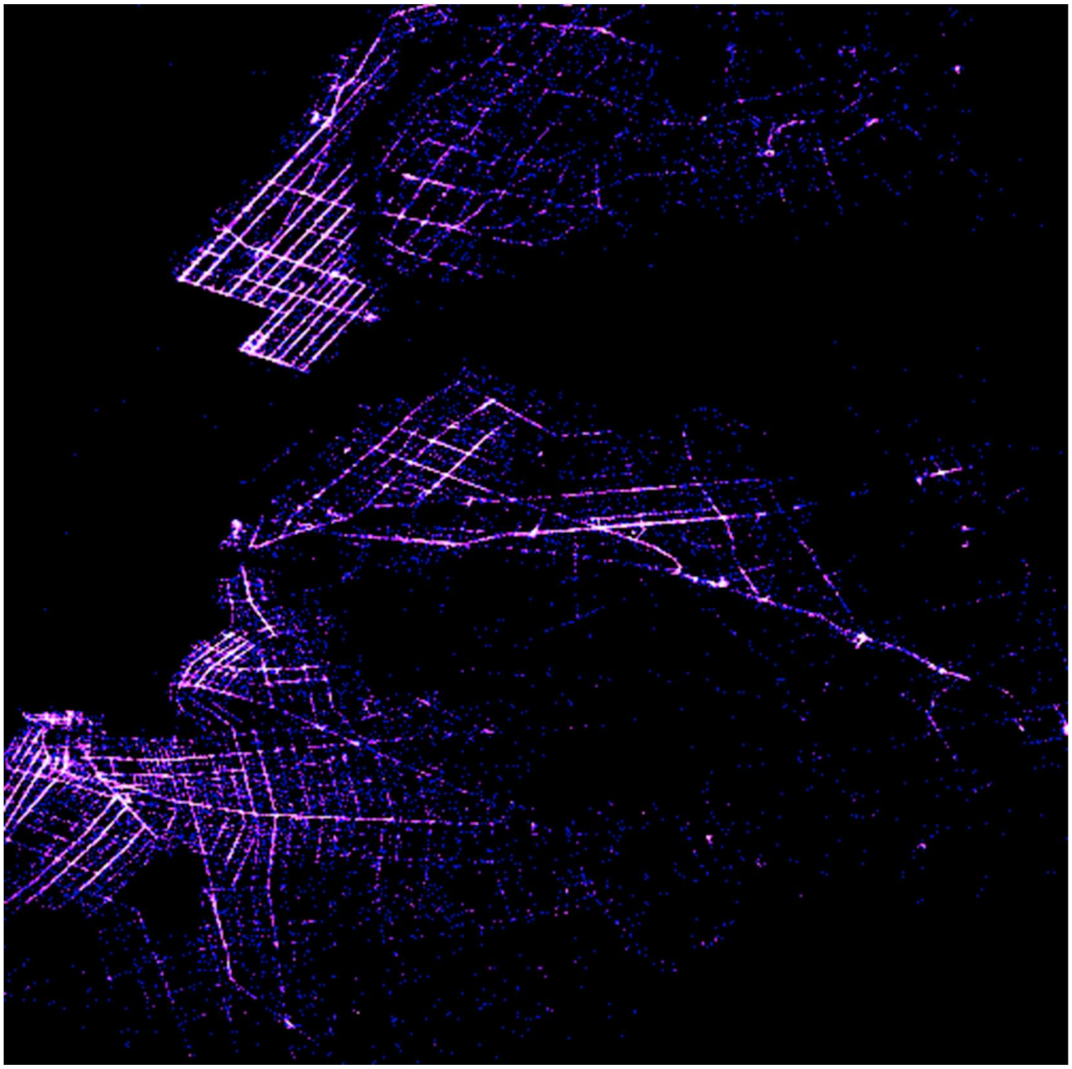
Overall Pickup and Dropoff Locations

6. Data Shader

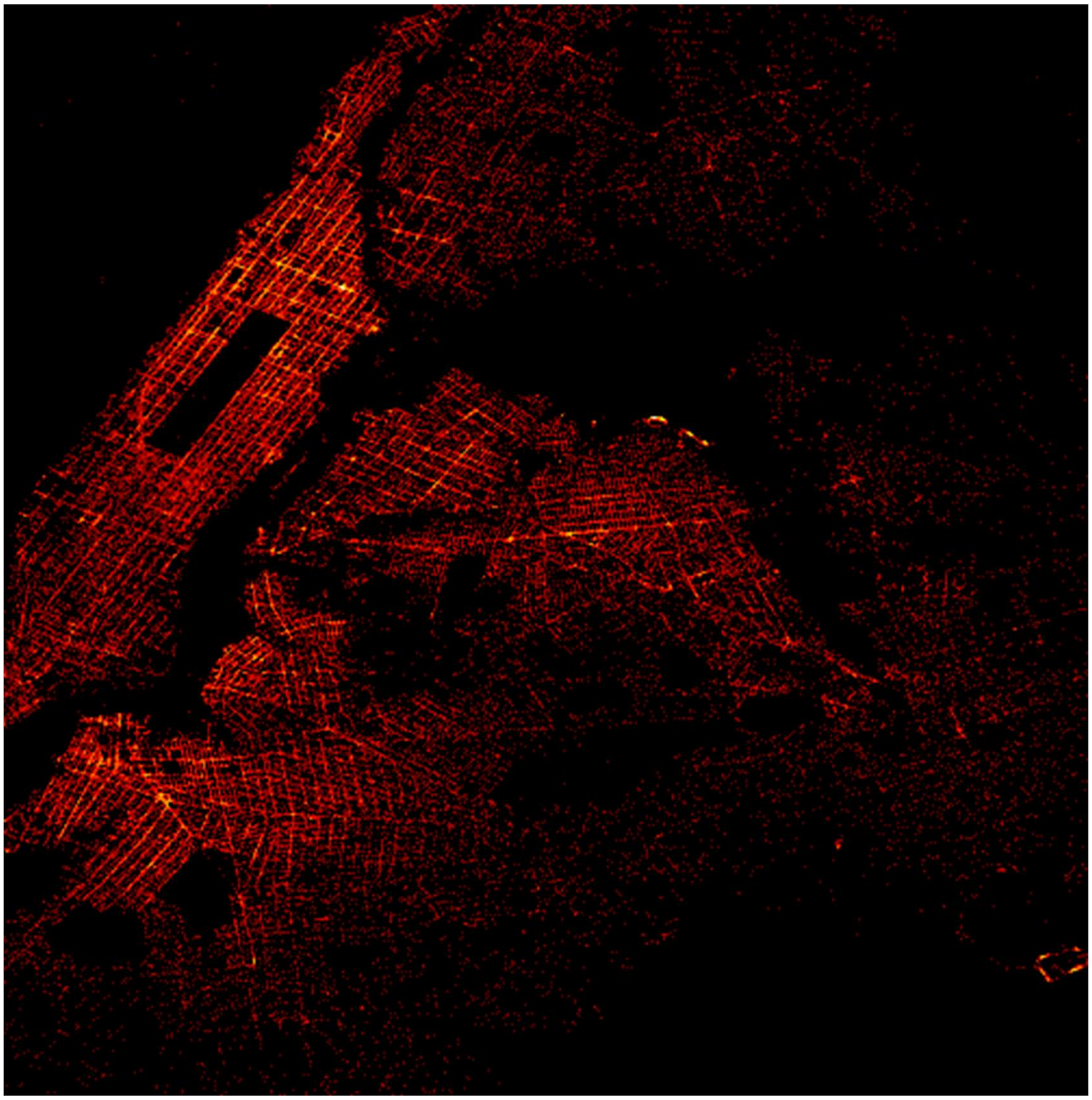
Contributor: *Gaurang Sharma*

We wanted to show some cool visualizations of the taxi points on Google Map.
DataShader is a library which helps with that.
After numerous attempts, we achieved this result.

All Pickup Points on the NYC Map



All Drop-Off Points on the NYC Map



Challenges Encountered
Contributor: *Chinmoy Sarangi and Gaurang Sharma*

- Not only checking for entering unique values, but also accommodate for new entries which are unique in nature instead of random samples which may enter the duplicate entries. This would have been solved with robust resources with more sophisticated systems in place which have capabilities to perform this task of checking for duplicates during the ingestion of the data.

- Issues while reading huge amount of data.

- Docker container could not be implemented due to a series of System crashes post installation.

- Due to lack of more sophisticated systems, unique and fresh data ingestion was challenging.

- Finding absolute queries to link Python and MongoDB reliably. This is because most of the earlier PyMongo codes are in various versions of Python 2.0 iterations, which are incompatible with Python 3 version.

- The newer datasets posted by the NYC Taxi commission do not have latitude and longitude data, starting from January of 2016. Hence, we had to use December 2015 dataset for green taxis in order to achieve faster results and be able to save time for further analysis.

- Also, the newer datasets only had location IDs, which had to be joined with the file containing all the taxi zones in order get the exact pick-up and drop-off locations. The latitude and longitude had to be then derived using the assisting shapefiles, which were uploaded separately.

- Using shapefiles meant, using newer Python libraries called Geopandas and Shapefiles. The output of the shape files in a DataFrame was only a single row output with comma separated values in a polygon format, which was challenging to breakdown further.

- The "Highest no. Of Trips on a Day" analysis is from the sample data of 1000 lines. When trying the same analysis with the original data (more than a million rows of data), it gave an memory error and the system crashed down.

Business Questions answered by the above Processing/Visualization: Contributor: *Chinmoy Sarangi & Gaurang Sharma*

- Where is the highest pickup and drop and what could be the reasons?
- Which attribute (columns) has the highest or lowest dependency on other attributes (columns)?
- What days have highest tip amounts and is it depending on duration/distance?
- What could be the rush hours?
- Which day/s had the highest number of trips?
- Which location has the highest no. of passenger pickups or drop off?
- Calculating the trip time, trip speed & extracting day of the week.
- Are there any anomalies in relation between trip distance/duration and the fare amounts?

References:
Contributor: *Chinmoy Sarangi and Gaurang Sharma*

https://www.youtube.com/watch?v=djfnjtYB2co
https://www.kaggle.com/kerneler/starter-tlc-trip-record-data-029ff9b7-4
https://towardsdatascience.com/data-visualization-with-bokeh-in-python-part-one-getting-started-a11655a467d4
https://pydatascience.org/category/bokeh/
http://zetcode.com/python/pymongo/
https://www.w3schools.com/python/python_mongodb_query.asp
https://www.quora.com/How-can-I-import-a-CSV-file-data-to-MongoDB-using-Python-Flask
https://api.mongodb.com/python/current/tutorial.html
https://docs.mongodb.com/manual/reference/operator/aggregation/abs/
https://www.youtube.com/watch?v=XDAnFZqJDvI
https://www.youtube.com/watch?v=BjlQdOK3n5w -- uses NYC data and bokeh
http://datashader.org/topics/nyc_taxi.html
https://www.kaggle.com/hanriver0618/nyc-taxi-data-exploration-visualization
https://michaeljsanders.com/2017/04/17/python-vlookup.html
https://gis.stackexchange.com/questions/262505/python-cant-read-shapefile
Data Analysis and Map-Reduce with MongoDB and PyMongo -- https://av.tib.eu/media/20068
https://medium.com/@jiaminhan/looking-through-the-taxi-meter-analysis-of-the-nyc-green-taxi-data-c1dbe5619afe
https://github.com/JiaminHan/nyc_greentaxi/blob/master/analysis.ipynb
https://stackoverflow.com/questions/25888396/how-to-get-latitude-longitude-with-python
https://developers.google.com/maps/documentation/javascript/get-api-key
https://developers.google.com/maps/documentation/geocoding/intro#RegionCodes
https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.datetime.html
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html
https://towardsdatascience.com/large-scale-visualizations-and-mapping-with-datashader-d465f5c47fb5