Data Engineering – Data Pipeline and Processing/Visualization Project

**Steps taken to implement the Data Pipeline:**

**Tools Chosen:** Python programming with Jupyter Notebook and MongoDB

**Basic Architecture Model**

**Contributor:** Chinmoy Sarangi



**Data Ingestion:**

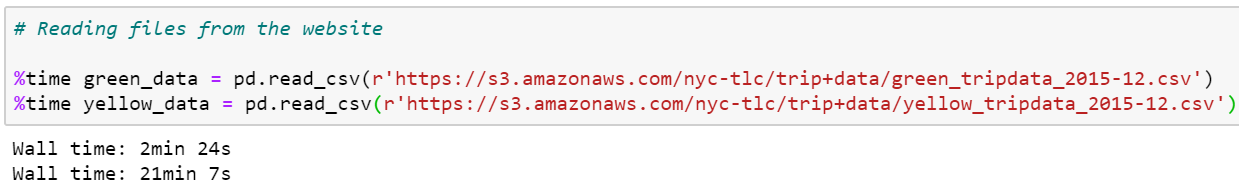
**Contributor:** Chinmoy Sarangi

**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:

.

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



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.

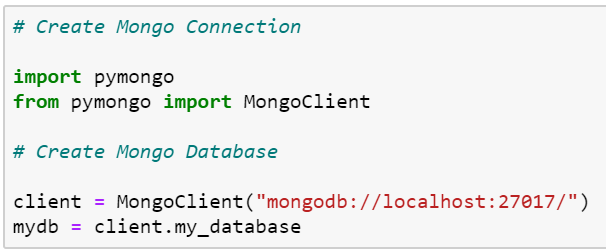
Since the dataset for yellow taxi trips was humongous, hence we chose to go with green taxi trips data, which was significantly smaller in size and easier to manage throughout the rest of the process, right until the visualization stage.

**Data Storage in MongoDB:**

**Contributor:** Chinmoy Sarangi

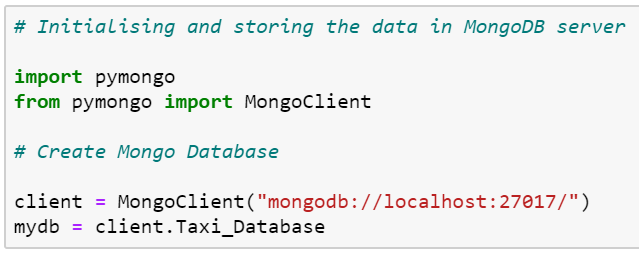
**Steps Taken:**

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

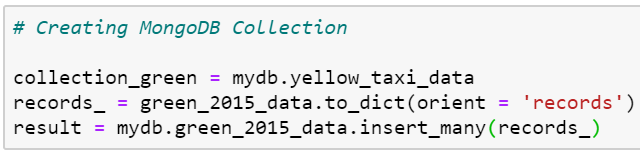


Creating Database Collection:

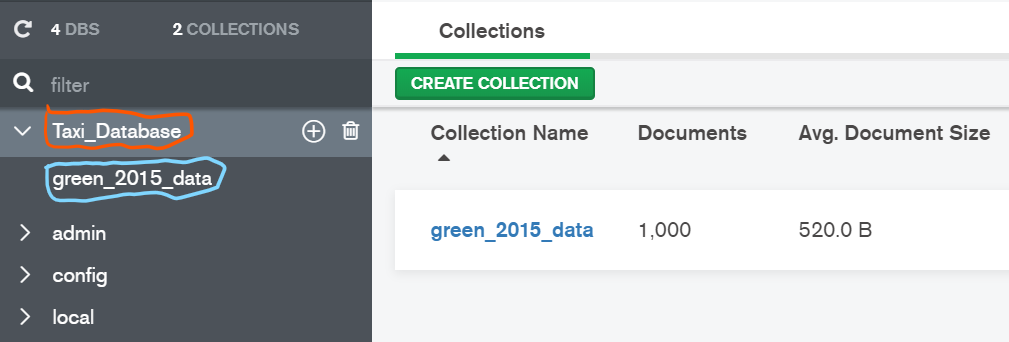
By using the Pymongo connection, 1000 rows of sample data were exported to MongoDB from the Pandas DataFrame. The Database name created is: **Taxi\_Database**.



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**.



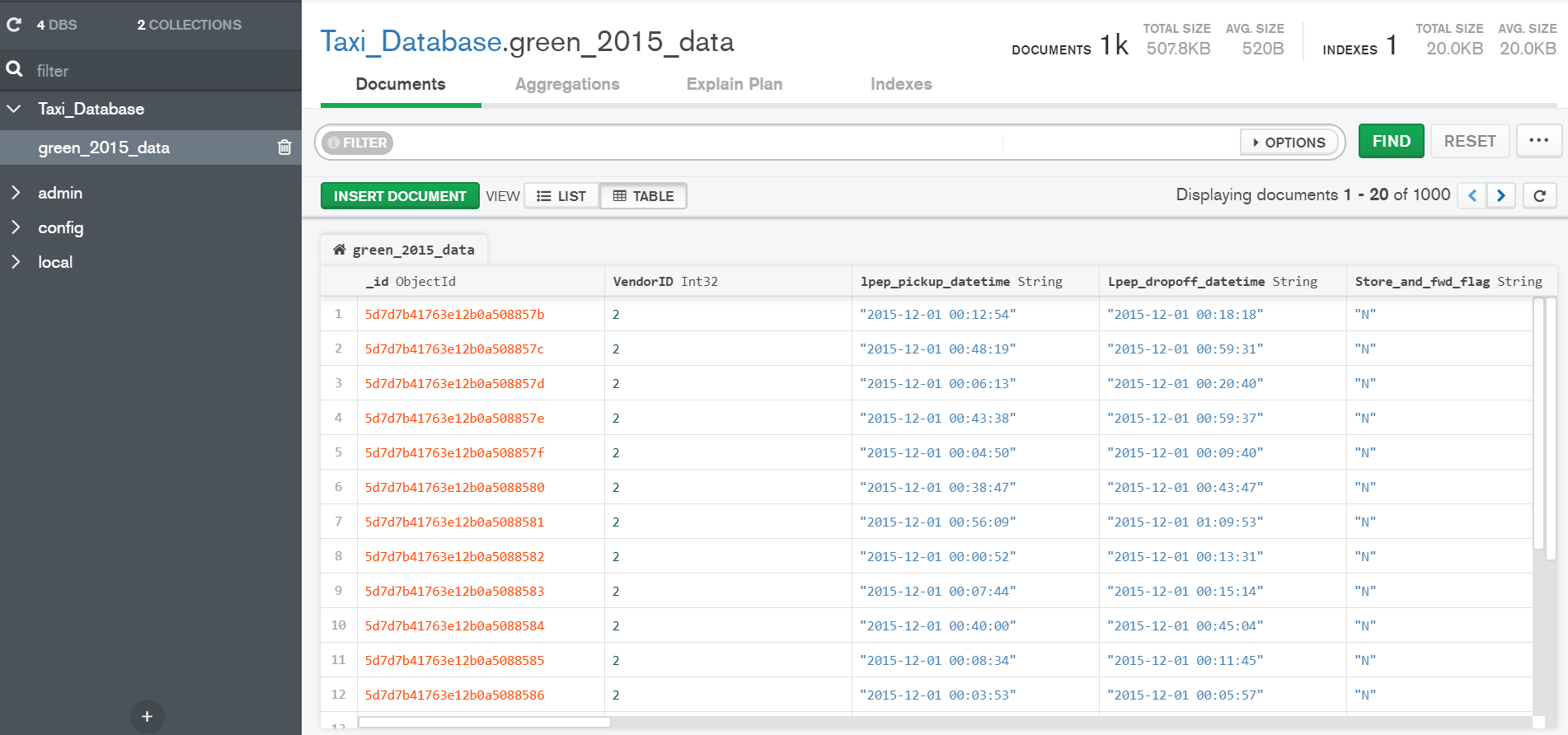
To ensure that the Databases were indeed created, the MongoDB Compass Console was refreshed to verify the same. As can be seen, the above Database and the associated collection for the green taxi trips have been created with 1000 records.



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



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



**Ideas:**

What high -level business questions can we answer with our analysis and visualizations? And what could be the newer future ideas?

**Steps done after accessing data from Mongo:**

Perform column splitting before pushing to Mongo.

Perform the join with taxi zone lookup with both yellow and green trip taxis before pushing it to Mongo DB.

**Challenges Encountered:**

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.

**Business Questions:**

Where is the highest pickup and drop and what could be the reasons?

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.

Are there any anomalies in relation between trip distance/duration and the fare amounts?