Algorithm and Data Models

Class Assignment-2

NEW YORK CITY TAXI DATASET

GROUP E: Ming Dai Nishra Shah Shaima Patel Vanish Patel

Submitted to- Sean Chester

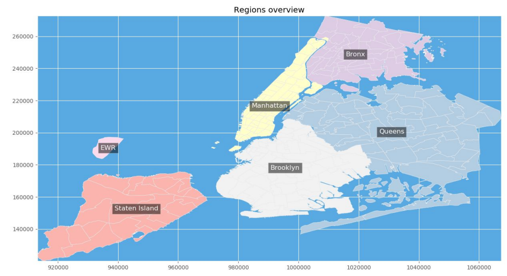
1. INTRODUCTION

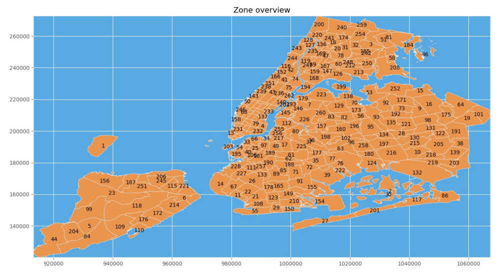
- The **Dataset**(tsample.csv)contains over 112 million rows and 17 attributes with strong spatio-temporal detailing
- Provided the subset of this dataset which contains **2^{20} rows and using its Spatial, Temporal and Spatio-Temporal aspects** we have tried to generate interesting and informative insights
- According to the research, there are roughly 200 million taxi rides in New York City each year. People use taxis in a frequency much higher than any other cities of US. Instead of booking a taxi by phone one day ahead of time, New York taxi drivers pick up passengers on the street. Exploiting an understanding of taxi supply and demand could increase the efficiency of the city's taxi system.
- This time we have tried many tools and libraries (Intervaltree, Tableau, geopandas, Polygon, itertools, pandas, shapefile, matplotlib, mapshaper etc)

2. DATA VISUALIZATION

(TRANSFORMING RAW DATA TO INSIGHTS)

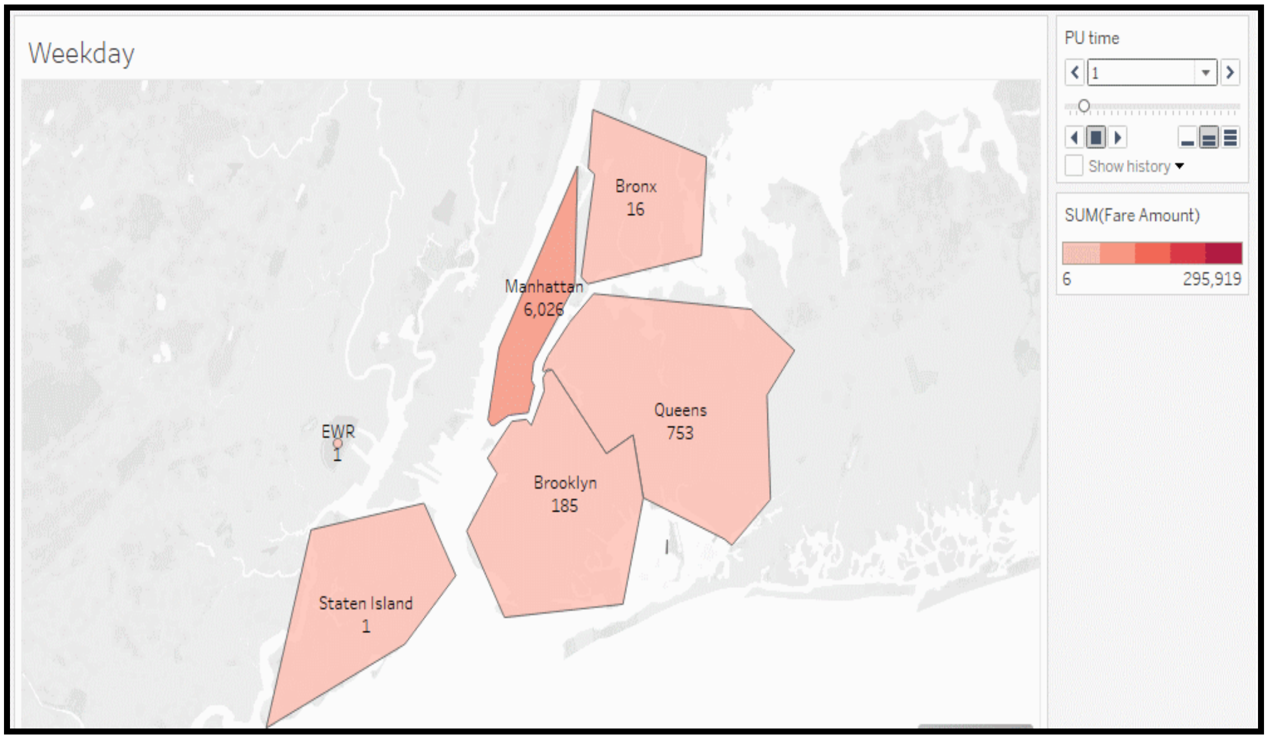
Preview the spatial data:





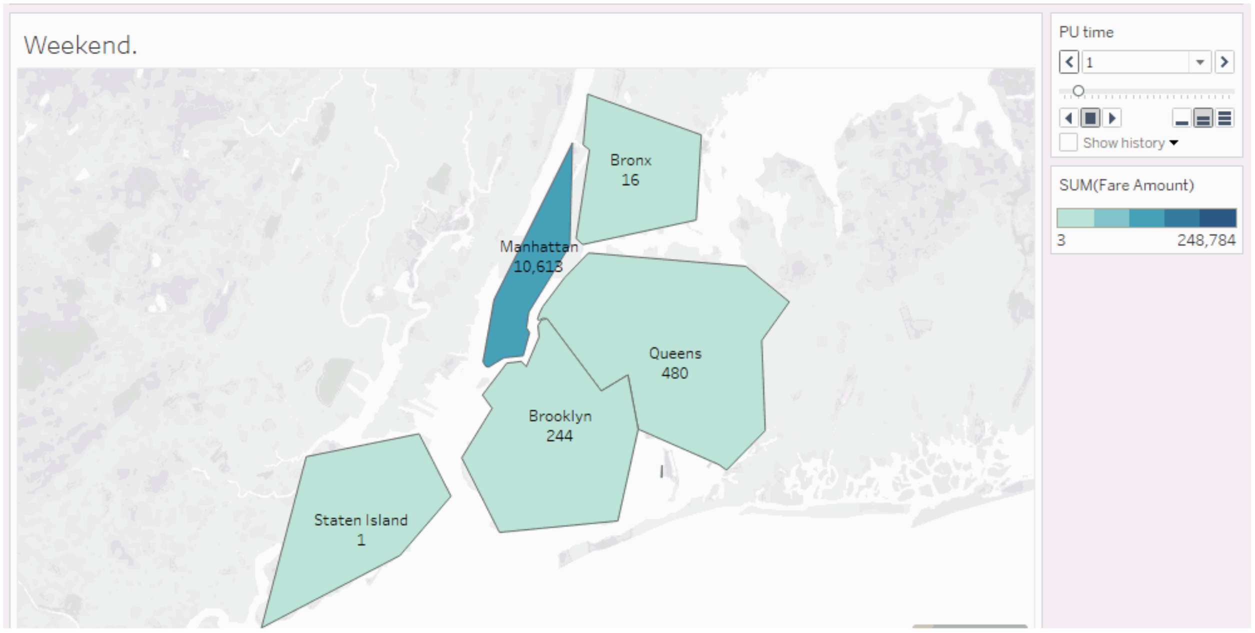
PLOT-1: COUNT OF TOLL AMOUNT AND SUM OF FARE AMOUNT PER BOROUGH (YEAR 2018-1049348 rows data)

- First, we found that the most data actually happened continuously in 4 days in 2018, that is Thursday , Friday, Saturday and Sunday. Thus, we decided to divide 2018 data into two parts by WEEKDAY and WEEKEND (as for each group is two days) to visualize the hours about the fare amount and toll paid.
- Columns used : PULocationID; tpep_pickup_datetime; fare_amount; tolls_amount
- The color is changed by the fare amount, and the number or label shown in the picture is the count of toll paid.



- So, this is the GIF created and has every hour data added to it.
- On the top right is the PU time i.e. Pickup Time in 24 hour format.
- We see the color changing from light pink to red on the basis of total amount of Fare collected per hour by each Borough.
- **INSIGHTS:**Therefore, we see a huge number of people travel in and Manhattan and use toll booth every hour all day, especially between 8am to 12pm (because of work) and then during evening hours too. (Considering the expensive lifestyle in Manhattan, we can assume they're returning back home after work.)

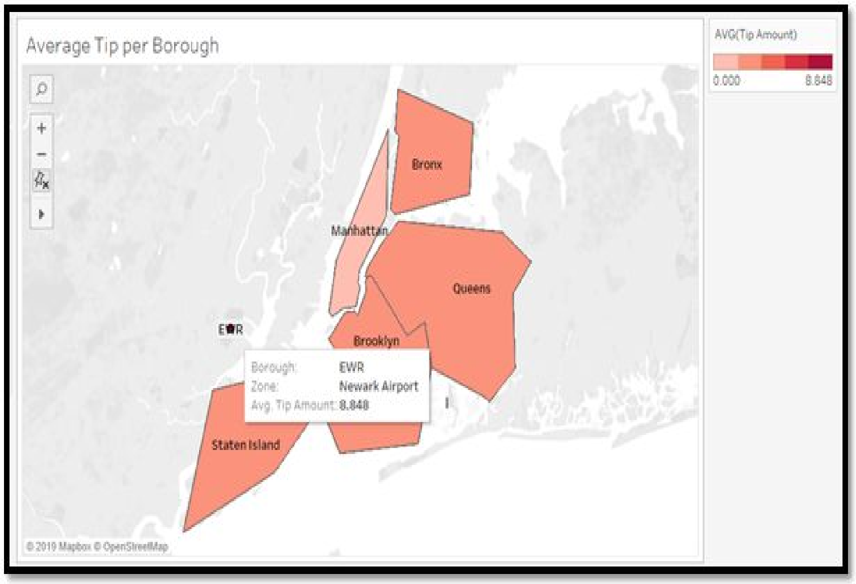
- Similarly, we took the WEEKEND data and created GIF.



- **INSIGHTS:** We see even on weekends people take taxis to hang out in Manhattan; with friends or go shopping, dinners etc. Looks like Manhattan is the hotspot of the New York and people really enjoy being there for various reasons. We see the trend is similar with the weekday in New York, at Manhattan but the number of tolls paid are slightly less than on weekdays.

PLOT-2: <u>AVERAGE TIP AMOUNT DURING DROPOFF PER BOROUGH</u>

FILES USED: taxi-sample.csv and taxi-zone.shp (Created full outer join with this temporal and spatial features), Columns used : PULocationID; tip_amount



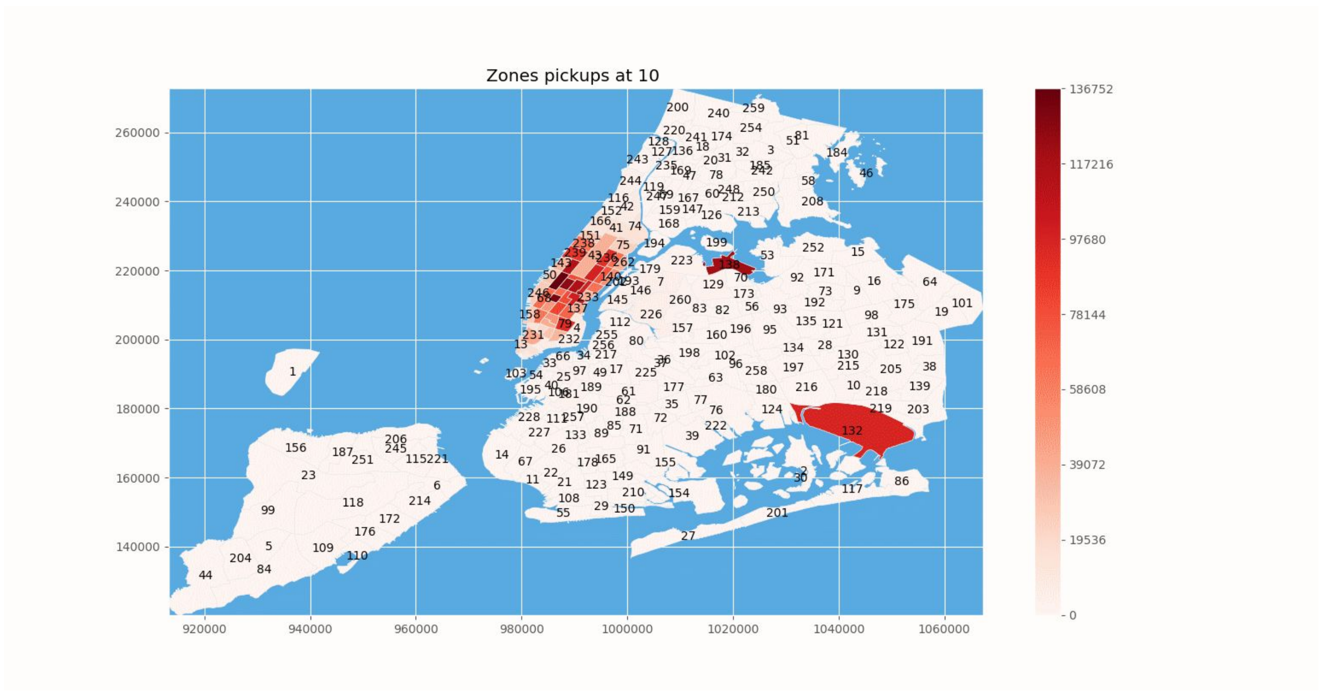
INSIGHTS: EWR, is the Newark Airport, people who are going to the airport are usually in a hurry and this can be a long distance trip so we can assume that people give higher tips to the taxi drivers just as an appreciation for their ride or making it on time. Therefore, the average of tips are higher during drop off at EWR.(Red color indicates that)

PLOT-3: NUMBER OF PICK UP PER ZONE

FILES USED: taxi-sample.csv and taxi-zone.shp (Created full outer join with this temporal and spatial features)

Columns used : tpep_pickup_datetime; DOLocationID

The following GIF picture shows all 2^20 data, pick up time changed by hour.



INSIGHTS: We can clearly see that the highest drop offs have been in Manhattan, which has tons of opportunities for every age group. The second and third zones are JFK Airport and LaGuardia Airport located in Queens region. Hence, taxi is probably the preferred mode of transportation while going to the airport and to be on time.

3. Data Modelling

- Data pre-process

We found that while dealing with the temporal data, there exists some misleading data. Here, we performed **data cleaning** to clean the future and duplicated data.

```
delete = data_all['tpep_pickup_datetime']>'2019-10-12 12:25:53'
data_all = data_all.drop_duplicates()
data_all = data_all.dropna(axis=0, how='any')
```

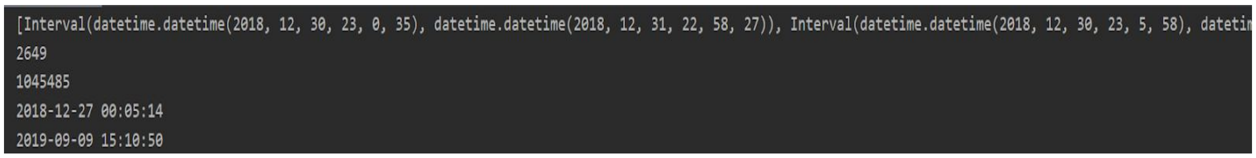
- Temporal data modeling

For the efficiency, we built an **interval tree** to model it. We extracted the drop off and pick up time from the table; transferred it into datetime format and inserted one to another then we got a temporal data file:

```
import datetime
import pandas as pd
from intervaltree import Interval, IntervalTree
from pylab import *
def data_adress():
  f = open(r'time_start.txt', 'r')
  time_start = list(f)
  f = open(r'time_end.txt', 'r')
  time_end = list(f)
  f.close()
  for i in range(0, len(time_start)):
            time_start[i]  =  datetime.datetime.strptime(re.sub("[']",  "",  time_start[i][0:19]),
('%Y-%m-%d %H:%M:%S'))
  for i in range(0, len(time_end)):
             time_end[i]  =  datetime.datetime.strptime(re.sub("[']",  "",  time_end[i][0:19]),
('%Y-%m-%d %H:%M:%S'))
  interval_data = []
  for i in range(0, len(time_start)):
     if (time_start[i] != time_end[i]):
        interval_data.append(((time_start[i], time_end[i])))
  tree = IntervalTree.from_tuples(interval_data)
  dt1 = datetime.datetime.strptime('2018-12-31 22:51:36', ('%Y-%m-%d %H:%M:%S'))
  result_interval = sorted(tree[dt1])
  length_result = len(result_interval)
```

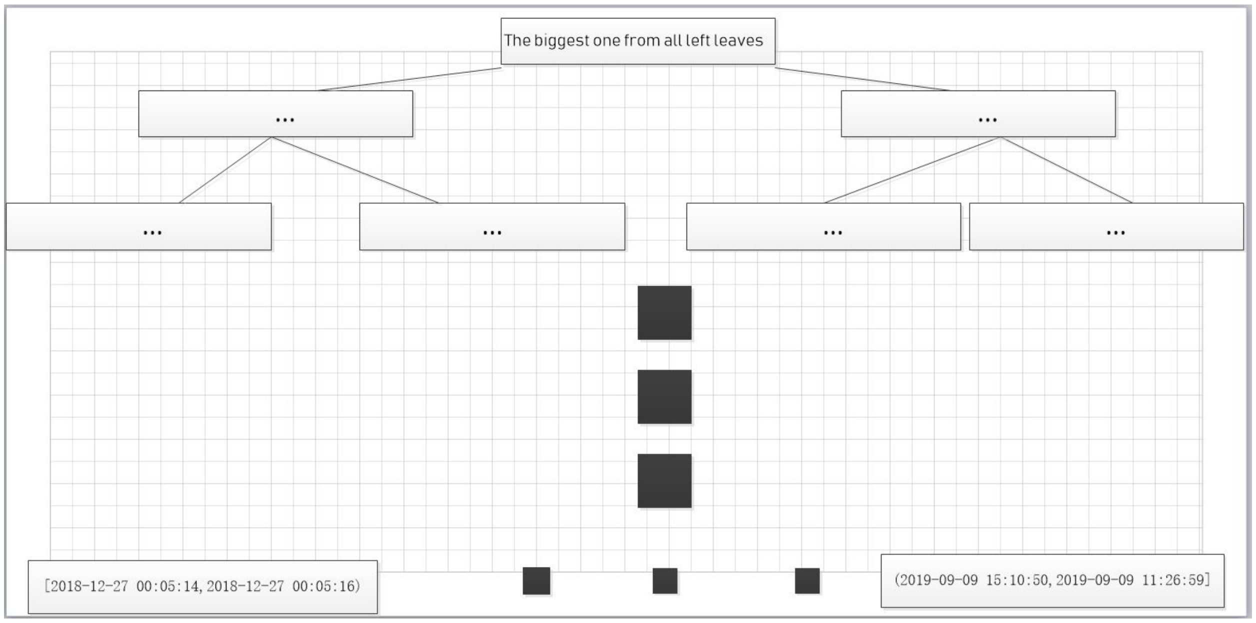
```
        print(result_interval)    print(length_result)    print(len(tree))    print(tree.begin())
print(tree.end())
def main():
  data_adress()
if __name__ == '__main__':
  main()
```

We stored all the time interval into this tree, and **it improved about 50%** in average of efficiency on the query time. The output of this query is :



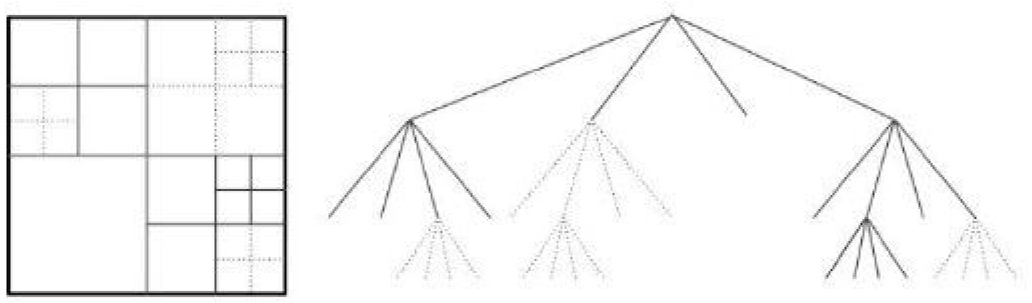
```
[Interval(datetime.datetime(2018, 12, 30, 23, 0, 35), datetime.datetime(2018, 12, 31, 22, 58, 27)), Interval(datetime.datetime(2018, 12, 30, 23, 5, 58), datetim
2649
1045485
2018-12-27 00:05:14
2019-09-09 15:10:50
```

And it costs **36 seconds**, so it is a very efficient data model for big data related to temporal data.



- Spatial data modeling

We used **Quad tree by geopandas library** in python to deal with spatial data. It is known that a spatial index such as R-tree can drastically speed up GIS operations like intersections and joins. Spatial indices are key features of spatial databases like PostGIS, but they're also available for DIY coding in Python.



```
taxi_zones = pd.read_csv('taxi_zones.csv', index_col=0, header=0, parse_dates=True)
gdf = gpd.GeoDataFrame(taxi_zones, geometry=geometryList)
spatial_index = gdf.sindex
possible_matches_index = list(spatial_index.intersection(polygon.bounds))
possible_matches = gdf.iloc[possible_matches_index]
precise_matches = possible_matches[possible_matches.intersects(polygon)]
```

A R-tree represents individual objects and their bounding boxes (the "R" is for "Rectangle") as the lowest level of the spatial index. It then aggregates nearby objects and represents them with their aggregate bounding box in the next higher level of the index. At yet higher levels, the R-tree aggregates bounding boxes and represents them by *their* bounding box, iteratively, until everything is nested into one top-level bounding box.

To search, the R-tree takes a query box and, starting at the top level, sees which (if any) bounding boxes intersect it. It then expands each intersecting bounding box and sees which of the child bounding boxes inside it intersect the query box. This proceeds recursively until all intersecting boxes are searched down to the lowest level, and returns the matching objects from the lowest level.

4. Algorithmic Considerations

Processor - Intel i7 7th gen CPU 2.5GHz 64-bit OS x64 based processor

RAM - 16gb

Tool/software - Anaconda version 2019.07/ Tableau 2019.3

Language – Python 3.7

Average Time taken for the execution of the above visualisations:

- Preview the spatial data-

  for 10000 spatial dataset - 35.31 seconds

  for 1,048,519 spatial dataset - 55.9 seconds

- COUNT OF TOLL AMOUNT AND SUM OF FARE AMOUNT PER BOROUGH(Tableau)- 10-12 seconds

- NUMBER OF PICK UP PER ZONE

  i. Loading file (CSV with 1,048,519 records) to Database and converting the file into table in the form of chunks:

  when executed with Anaconda in pycharm -3 minutes (approx.)

  ii. Executing SQL statements to fetch from table

  with pandas - 1.5 minutes (approx.) (for instance, it took 1min 20sec)

  with geopandas - 52 seconds (approx.)

- We successfully tested our code for different amount of data; like 10k, 100k,and 1,048,519 rows.

- We were successful in fetching the records quicker **using the interval tree, quadtree in data model and geopandas, tableau** as tools to improve our efficiency.

5.Relationship to Spatio-temporal Challenges

We went through some of the challenges during the working of our spatio-temporal data but here we are presenting some of the most important related to the scalability part and the solutions.

• Loading 1,048,519 records:

First we got out of memory exception when we were trying to load 1,048,519 rows using pandas. In order to process these larger files, we tried database in the backend and used data chunking in order to load csv file into the database. SQLAlchemy library was used to perform this operation. Because the data was loaded in chunks, smaller RAM did not restrict from processing large files.

• To minimize the query time

Although we were preprocessing the data and adding it to the database, but we were fetching data from the database with the help of pandas.

After trying to deal with spatial data in quad tree and temporal data in an interval tree, the efficiency has been improved a lot , and tools and libraries such as tableau , Intervaltree and geopandas can be implemented in the future work when dealing with the spatio-temporal data.