Traffic Pattern Exploration Using New York Yellow Taxi Dataset

Final Report for CMSC 12300 - Fast & Furious

Hyun Ki Kim[1], Andi Liao[2], Wiston Zunda Xu[3], Weiwei Zheng[4]

Github Repository: https://github.com/liaoandi/fast_and_furious

June 4, 2018

I. Basic Information about Dataset

1. New York City yellow taxi data

The main dataset used in this project is the NYC taxi public dataset. The New York City Taxi & Limousine Commission has released a staggeringly detailed historical dataset covering over 1.1 billion individual taxi trips in the city from January 2009 through Dec 2017. This so-called TLC trip data includes three categories: the yellow taxi, the green taxi and the For-Hire Vehicle ("FHV"). To be specific, the yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts, while FHV trip records include fields capturing the dispatching base license number and the pick-up date, time, and taxi zone location ID.

Taken as a whole, the detailed trip-level data is more than just a vast list of taxi pickup and drop off coordinates: it is a story of New York. How bad is the rush hour traffic from Manhattan to JFK? Where does the citizens in New York city hang out on Saturday nights? What time do investment bankers get to work? The dataset addresses all of these questions and many more. Therefore, our group tried to investigate the traffic pattern of New York city to figure out whether we could solve the above question or even find something more interesting behind the TLC trip data.

The project only focuses on yellow taxi records. Unlike green taxi and For-Hire Vehicles (FHVs), yellow taxi provide transportation exclusively through street-hails, so it will reveal New Yorkers' behavior well. On average, the data size is around 1.2 GB per month, including over 10 million rows (e.g. over 10 million individual trip record).

The second dataset we referred to is the Weather Underground dataset. Weatherunderground.com provides detailed historical data of different cities in the United States and the records are loaded in a fixed interval (one record per hour or so). By using the Wunderweather API, we scraped weather records about weather conditions from 2015 July to

[1] University of Chicago, MA in Computational Social Sciences, <mailto:hyunkikim@uchicago.edu>
[2] University of Chicago, MA in Computational Social Sciences, <mailto:liaoad17@uchicago.edu>
[3] University of Chicago, MA in Computational Social Sciences, <mailto:zunda@uchicago.edu>
[4] University of Chicago, MA in Computational Social Sciences, <mailto:weiweiz@uchicago.edu>

2016 June in New York city and matched the weather condition to the taxi dataset. The code can be checked in /code/index/weather and weather data can be checked in /data/weather_201507_201606.csv.

II. Hypotheses

Though the project doesn't have specific hypotheses, our work can be divided into two parts. First, we try to look at causality and make prediction in the taxi dataset by applying machine learning mindset to simple algorithms written in mapreduce. In the big data context, it is hard for us to use complicated machining learning algorithm with well-known written packages such as sklearn. Therefore, we wrote mapreduce code which referred to algorithm of simple and multiple linear regression and tried to conduct simple data analysis on large datasets. In addition, we used different algorithms to match pair of trips to make prediction and compare the performance with predicting with single trip. We assumed making prediction by matching pairs would have comparable accuracy as using single trip when complicated machine learning approaches are not available.

Second, considering NYC taxi dataset doesn't provide detailed information about guests and drivers but has a large enough scale to generalize people's taxi hailing behavior, we are tempted to deanonymize hidden traffic pattern by leveraging big data technique to find out interesting facts which might be illuminating to human knowledge in different fields.

III. Generate index

In this project, we are interested in the impact of weather, location and time on traffic time (time of the trip per mile) and tip rate (credit card tip amount over total fare amount). But given that most variables in the three dimensions are categorical, it is impractical to create so many dummy variables and do analysis on machines with limited data processing power. Thus, we came up with a trick to measure difference between different categories by generating index for the categorical variables we are interested in.

Using a sample subset (1.2 MB, 12 thousand rows) generated from 12-month dataset from 2015/07 to 2016/06 (1.6 GB, 10 million rows each month), we made categorical variables in weather, location and time to represent different combinations in each dimension. Then we calculated the mean of dependent variables of those different categories and got a value within 0 to 1 by comparing the means of different categories in each dimension. We got a new variable representing variables in weather, location and time ranging from 0 to one and call this normalized value "index", regarding specific dependent variable. Based the result from the sample, we then map indices to different trips on the whole dataset to study causality and prediction. We generated six groups of indices for weather, location and time. Sample data can be checked in data/sample_trip.csv and detailed index data can be seen in /data/index folder.

1. Pick-up time index

Although there are both pick-up time and drop-off time in the dataset, only pickup-time is utilized to generate index. We created three indices for the time dimension – month index, weekday index and hour index, which represent the month, the day of the week and the hour a trip took place respectively. The code of this part can be found in /code/index/time.

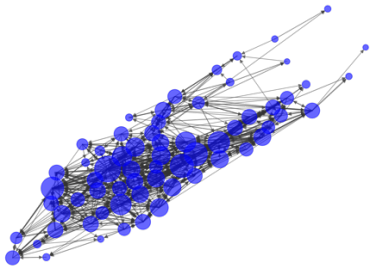
2. Location index

Since different places have different traffic situation, location is another important aspect we need to consider when analyzing trip duration time and tip rates. In our raw dataset, there are two pairs of variables, pick-up coordinates and drop-off coordinates, which reflect the location information of an individual trip. Although the duration time of a single trip not only depends on the pick-up location and drop-off location, but also the route that taxi driver, considering the difficulty of obtaining the route information of a single trip, we decided to just focus on the starting point and the ending point.

The first step of getting the location index is to cluster those geographic points, i.e. to classify those pick-up locations and drop-off locations into different clusters, and the relatively nearest points will be classified into the same cluster. We assume that different district have different traffic situation and locations within the same district share the similar traffic patterns. The algorithm we used is k-means. Although K-means method clusters the points based on calculating the Euclidean distance, which is different from the geographic distance, considering all those points are in the same city, there will not be a big difference between geographic distance and Euclidean distance.

After we drew a random sample from the whole year dataset, we trained two k-means models on pick-up locations and drop-off locations separately based on the sample dataset, here we set the number of clusters as n. Then we used these two models to predict the cluster of the remaining data in the whole dataset to get the pick-up cluster label and drop-off cluster label for each individual trip record. The figure shown below is the 100 clusters of locations predicted by fitted k-means models, the blue circles in the figure represents the clusters' center, and the line between each circle means there exist drip between these two locations. The bolder the line is, the larger the amount of taxi trips between two clusters is. We can see from the figure the shape of Manhattan, the most prosperous area in New York City (after we dropped some trivial nodes when plotting). Though when we mapped the index to the larger dataset, we only used 50 clusters to get larger generality. Code for this part can be seen in /code/index/location.

Figure 1: Fifty clusters of locations trained by k-means models



3. Weather index

The weather conditions are not recorded in a fixed interval, so we mapped weather data of the closest moment of each trip's pick-up time. We divided the sample file into different chunks and map the weather data scraped beforehand (/data/weather_201507_201606.csv) to different the chunks separately with MPI. After we mapped the weather information to the sample, we used mrjob code to calculate different means of dependent variables (traffic time and tip rate)

of different weather pair (condition and visibility) and normalized the value to 0 to 1 dataset (sample data in /data/index/weather_index/sample_weather_time.csv).

We also wrote a simple mapreduce file to calculate the total occurrence of each weather pair and get rid of outliers. There are in total 196 categories of weather variable in the dataset and you can check the counts in /data/index/weather_index/clean_weather_count.csv.

After we got the index of each weather pair category (different for traffic time and tip rate), we assumed the weather condition of each hour is the same and set the weather of earliest moment in an hour in the scraped data as the weather condition for the whole hour. It makes it more efficient to map time to index rather than map time to weather then to index.

After we got all the index for weather, location and time we mapped the indices to the whole dataset in one year using mapreduce (code can be seen in /code/index/mapindex.py). Though we mapped all the indices for traffic time and tip rate, due to the limit processing power, we only run following analysis (except for the deanonymizing part) on traffic time.

IV. Single Trip

Recall that one of the question we mentioned earlier is "How bad is the rush hour traffic from Manhattan to JFK?" People living or traveling in such a metropolitan city may really concerned about how much time they will take from one place to another if they choose to take taxi. Thus, after generating the weather index, location index and pick-up time index as mentioned before, we wondered whether we could utilize those traffic related index as the traffic pattern of New York city to predict the duration time of a trip.

Algorithm

Considering that we generating six indexes (e.g. pick-up weather index, drop-off weather index, location index, pick-up hour index, weekday index and month index) for each individual trip, the first simple model we implemented is multivariate linear regression model, the formula is:

$$y = \beta_0 + \beta_1 X1 + \beta_2 X2 + \beta_3 X3 + \beta_4 X4 + \beta_5 X5 + \beta_6 X6$$

Here y is the time (minutes) the taxi takes to drive 1 kilometer, and X1, X2, …, X6 represents the six indexes we generated. Since we are not sure whether we can use python package successfully on google cloud, we wrote our own algorithm and codes by MapReduce to solve the solutions of the coefficients in this multivariate linear regression model. The algorithm we used is Cholesky decomposition, and we will calculate the sample covariance of explanatory variables and covariance between explanatory variable and dependent variable through MapReduce, then get the coefficients of this multivariate linear regression model through Cholesky decomposition.

Method

The first step is to get the sample covariance of explanatory variables and covariance between explanatory variable and dependent variable. MapReduce method with three steps is utilized, the first mapper is to extract the index and y variable (travelling time) from the file and then

calculate the sample covariance of index X = [X1, X2, …, X5, X6] by numpy.outer function, also calculate the covariance between X and y by X*y. The second mapper is to transforms numpy arrays of two covariance calculated in the first step into json-encodable list format and sends to reducer, the final reducer step is to aggregates results produced by each mapper and obtains the two covariance for all data, then using Cholesky decomposition obtains the coefficients of linear regression.

Results

We draw two sample data from the whole dataset, and each dataset includes 10 million rows, we will use the first sample data as the training dataset to train our multivariate linear regression model by our algorithm and then use the second sample dataset to test our model. (sample data see /data/raw_sample_time.csv)

The coefficients we obtained through our MapRreduce method is:

Figure 2: Results of running MapReduce to solve parameters of linear regression

```
xuzundadeMacBook-Pro:Desktop winston$ python3 MRLinearRegression.py raw_time_tot
al_1.csv
No configs found; falling back on auto-configuration
Creating temp directory /var/folders/cc/d4d82zl95s97wvxv0hsknd680000gn/T/MRLinea
rRegression.winston.20180604.041251.406600
Running step 1 of 1...
betas [-0.00290415  0.00348077 -0.00210198  0.0005363   0.00019483 -0.00014763
  0.00126603]
<class 'numpy.float64'>
```

From the figure 2 shown above, we find the betas obtained from our method are as follows: the coefficient for pick-up weather index is -0.0029, the coefficient for drop-off weather index is 0.00348, the coefficient for location index is -0.0021, the coefficients for three time-index is 0.00054, 0.000195 and -0.000147 respectively. To verify the accuracy of our algorithm and method, we tried python package to run the linear regression on the same dataset, the results are as follows:

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.0029	0.003	-0.874	0.382	-0.009	0.004
x2	0.0035	0.003	1.049	0.294	-0.003	0.010
x3	-0.0021	0.000	-6.353	0.000	-0.003	-0.001
x4	0.0005	0.000	3.370	0.001	0.000	0.001
x5	0.0002	0.000	0.865	0.387	-0.000	0.001
x6	-0.0001	0.000	-1.008	0.314	-0.000	0.000
const	0.0013	0.001	2.233	0.026	0.000	0.002

Table 1: Results of running python package to solve parameters of linear regression

From the Table 1 shown above, we find the betas obtained from our method is the same as the results obtained from using python package. Also from the table shown above, we noticed that the coefficients for X3 and X4 is statistical significant, which represents location index and hour index respectively. This result also corresponds with the real situation, which means the pick-up location, drop-off location and the pick-up time will influence the duration time of a trip.

To measure the prediction effects, we also wrote another MapReduce method to calculate the root of mean squared error of the fitted multivariate linear regression model. We predict the y variable of the second sample data based on the model we fitted using the first sample dataset, and the calculated RMSE is 0.0462. The RMSE shows although the multivariate linear regression can help us to figure out which index can be used as explanatory variable in predicting duration time of a trip, but the prediction accuracy is not that good if we only consider the single trip in our analysis.

V. Matching Pair

1. Causality

Algorithm

After we got the coefficient of each index on the dependent variable. In this section, we want to see how the trend of each index accords with the trend of change of y. First, we match pairs of trips by controlling any five of the indices, which means each trip of the given dataset have six pairs in terms of different indices. After controlling the indices of the other five, we can see how the difference in the last index influence the difference of the dependent variable. Based on the most similar trips we found, we establish a simple linear regression model for *X: differences in index* and *y: differences in travelling time* for all the trips using the formula as following:

$$y = \beta_1 X + \beta_0$$

$$\beta_1 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(y_i - \bar{y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{X}$$

The regression coefficient reflects the impact of index on travelling time, and as we have normalized index in the previous sections, the result of regression coefficient can be compared directly between different indices.

Method

MapReduce method with two steps is utilized. The first step is to extract index and travelling time of the compared trip and its counterpart, calculate differences of index and travelling time, means, sums of squared difference and sums of produce differences. The second step yields regression coefficients based on the parameters from the first step.

Result

Due to the limited capacity Google Cloud cluster, we use a subsample of 5000 trips to be matched, and another subsample of 50000 trips as matching pool (format as /data/raw_sample_time.csv). Using 11 cores to run the jobs, it took about 40 min to finish the fixed_effect.py task. Regression coefficients are calculated inside Virtual Machine.

The simple regression result for each index are close: they have nearly identical coefficients. The absolute values of regression coefficient are small, indicating that when other five indices are controlled, the difference of a single index can hardly influence the difference of unit travelling time. But we can still see trivial difference between different indices. The result can be seen in the following table. Without using The code of this part can be found via ../code/matching pair/causality.

Table 2. Results from simple linear regression in causality part

	pick-up weather	drop-off weather	location	weekday	hour	month
B 0	0.000217027953 57650943	0.00021702795 35765095	0.000217027953 57650816	0.000217027953 57650948	0.000217027953 57650948	0.000217027953 57650962
B 1	- 0.000754124761 3518914	- 0.00075412476 13518913	- 0.000754124761 3518912	- 0.000754124761 3518863	- 0.000754124761 3518881	- 0.000754124761 3518882

2. Prediction

We also used matching pair to get the prediction of sample trip by looping over another dataset to look for the most similar trip. We predict the value of y the given trip as the same of its most similar trip. Two algorithms are used to match pair. Then we calculate the MSE (mean squared error) of the two algorithms and compared them to the MSE of dummy regressor by means (predict all the value of the given dataset as the mean). Because the matching process took so long to finished, we only match the sample data (1.6 MB, 12 thousand rows) to a subset of the mapped index whole dataset (160 MB, 500 thousand rows. It took about two hours to finish by using the first algorithm and more than four hours the second to finish predicting. The sample input data to be predicted can be checked in /data/sample_trip_2.csv & raw_sample_time.csv The code in this part can be looked at in /code/matching_pair/prediction/msemr.py.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

I. Sum of squared difference

The first algorithm we used is by comparing the summation of the squared differences of the six indices of all the trips in another dataset. We wrote mapreduce code to match the sample trips, which can be checked in code/matching_pair/prediction/matchpredictmr.py. The result for this algorithm is as follow. We can see the performance of our model is not so good, but one possible reason might be we just used a very small subset as the matching pool.

Table 3. Results for the first matching pair algorithm

	Algorithm 1	Dummy regressor
MSE	0.0000211	0.000000777

II. Difference of each index

The second algorithm we used is by comparing absolute difference of each index of the two trips, and if all indices of the two trips are the closest to each other they are seen as a matched pair. We wrote mpi code to match the sample trips, which can be checked in /code/matching_pair/prediction/matchpredictmpi.py. The result mse for this algorithm is as follow:

Table 4. Results for the first matching pair algorithm

	Algorithm 1	Dummy regressor
MSE	0.000037327	0.0000324809

By comparing the two matching algorithms, we can see the second one performs slightly better than the first one. However, since the second task took so long to finish, we use an even smaller subset of data to complete the task. So which algorithm is better is hard to tell.

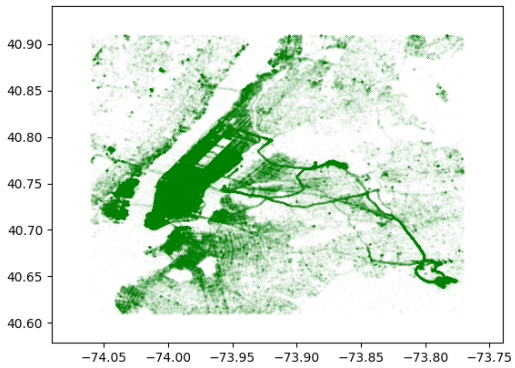
VI. Passenger Privacy

New York City Taxi and Limousine Commission provided taxi data including exact pick up and drop off longitude and latitude from January 2009 to June 2016, but they changed their policy to only provide taxi zone after July 2016. It was because passenger privacy issue was raised, mainly from the 2nd Annual NYC Taxi and Limousine Commission Hackathon on October 2016. Therefore, we were curious how much de-anonymization can be done on anonymized public big dataset. We thought it could be possible to de-anonymize certain passenger if they have unusual tipping behavior or take taxi at unpopular places.

Firstly, we gathered all the taxi trips that have tip greater than $15. Since the algorithm was basic going over the trips and filtering tip amount, which has run time of O(n), it was faster to use run MapReduce in multiple instances rather than using Google Cloud Dataproc. Variable name, data type, and order changed couple times (2010/01, 2010/08, 2014/01, 2015/07), but it was not a big problem with MapReduce.

After combining all the trips with $15 plus tips, we added another filter of tip rate greater than 50% to find unusual trips. Tip rate was calculated by dividing tip amount with fare amount, and tip amount are provided only when passengers had paid by credit card. Then we ran a second MapReduce to find top 150 pick up and drop off locations. After manually searching coordinates in the Google Maps, we found top places were John F. Kennedy International Airport, LaGuardia Airport, Pennsylvania Station, and the Grand Central Terminal. However, there were also some interesting places such as near famous nightclubs, company headquarters, and luxurious apartments. Figure 1 shows pick up and drop off location of trips that have tip amount greater than $ 15.

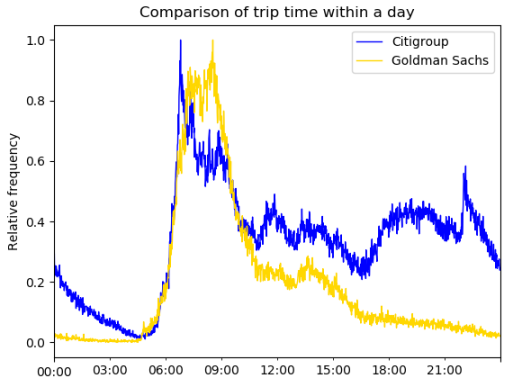
Figure 3. Pick up and drop off location of trips with tip amount $15 or more



Then we found lists of nightclubs, company headquarters, and luxurious apartments in New York City and filtered location that are not in the center of Manhattan as it is hard to tell where passenger is coming from. Then we ran third MapReduce to label trips that have pick up or drop off location neighboring known places. However, we only analyzed trips from January 2015 to June 2016 because businesses could not have been in the same places as now if we go back too far. Also, this MapReduce also had run time linear to the number of trips, so it would be simply spending more time to run with more month.

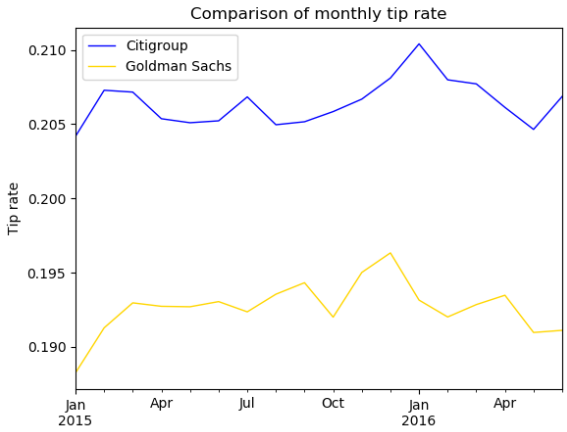
After labeling each trip, we were able to compare businesses. Figure 2 shows number of taxi trips near two investment banks in New York City (Citigroup and Goldman Sachs).

Figure 5. Comparison of trip time within a day



Pick of trips surge earlier for Citigroup than Goldman Sachs in the morning, suggesting workers or visitors of Citigroup taking Yellow taxi to these places come earlier compared to Goldman Sachs. Figure 3 shows monthly tip rate for trips near these companies.

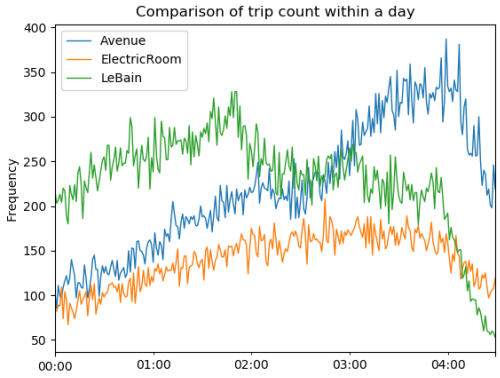
Figure 6. Comparison of monthly tip rate



Tip rates for trips near Citigroup building were consistently higher than the trips near Goldman Sachs building over the period January 2015 to June 2016. Also, two tip rates had similar trend suggesting people in each group are affected by similar macroeconomic factors when making decision how much to tip.

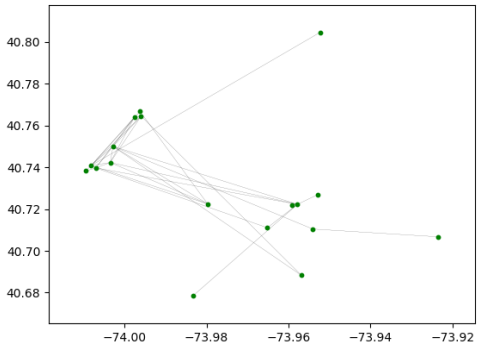
Figure 4 shows number of weekend (Saturday, Sunday, Monday), late night (12 a.m. to 4:30 a.m.) taxi trips near three nightclubs (Avenue, Electric Room, Le Bain).

Figure 7. Comparison of trip count within a day



Three nightclubs had similar average tip rate (Avenue: 0.2052, Electric Room: 0.2064, Le Bain: 0.2058), but the pick times were different (Avenue: 3:59 a.m., Electric Room: 2:45 a.m., Le Bain: 1:50 a.m.). Then we did network analysis between nightclubs to find which network is in the center of the network. Figure 5 shows the network graph between nightclubs.

Figure 8. Network graph of night clubs (weekend, late night trips)



It is not clear which nightclub is most popular, but it suggests there are more trips between Manhattan clubs than Brooklyn clubs.

Then we ran MapReduce to find top pick up, drop off, and tip amount pair with tip rate greater than 50% to find frequent user of Yellow taxi. Although it was hard to tell if the trips were from one person, there was one interesting trip happening on weekdays afternoon in New York City. Table 1 shows the trip records of this individual.

Table 4. Trip records of individual with $25 tip

	tpep_pickup_datetime	dropoff_time	day_of_week	passenger_count	trip_distance	fare_amount	tip_amount
0	2014-11-10 16:34:28	16:46:59	Monday	1	2.4	10.5	25.0
1	2014-11-12 17:45:44	18:10:44	Wednesday	1	2.3	16.0	25.0
2	2015-01-28 17:54:49	18:08:31	Wednesday	2	2.4	10.5	25.0
3	2015-02-24 18:22:30	18:42:54	Tuesday	1	2.4	13.5	25.0
4	2015-03-04 17:09:57	17:26:25	Wednesday	1	2.4	12.5	25.0
5	2015-03-11 18:18:47	18:44:31	Wednesday	1	2.4	16.5	25.0
6	2015-03-25 17:21:51	17:39:51	Wednesday	2	2.4	13.0	25.0
7	2015-04-21 17:32:25	17:48:30	Tuesday	2	2.4	12.0	25.0
8	2015-07-27 16:44:05	17:03:15	Monday	1	2.4	13.5	25.0
9	2015-07-29 16:36:17	16:57:09	Wednesday	2	2.3	14.0	25.0
10	2015-09-21 16:18:55	16:38:44	Monday	1	2.4	13.5	25.0
11	2015-10-29 16:14:21	16:30:36	Thursday	1	2.3	12.5	25.0
12	2015-11-02 16:13:03	16:24:31	Monday	1	2.4	10.0	25.0
13	2016-01-27 18:18:07	18:43:20	Wednesday	1	2.3	15.5	25.0
14	2016-01-28 16:57:12	17:19:45	Thursday	1	2.3	15.0	25.0
15	2016-03-10 16:16:49	16:33:42	Thursday	1	2.3	12.0	25.0
16	2016-03-14 17:00:17	17:18:51	Monday	1	2.3	12.5	25.0
17	2016-03-16 17:44:56	18:07:18	Wednesday	1	2.4	15.0	25.0
18	2016-03-23 16:33:02	16:51:13	Wednesday	1	2.3	13.0	25.0
19	2016-03-31 17:00:00	17:17:03	Thursday	2	2.4	12.0	25.0
20	2016-04-12 16:58:01	17:13:28	Tuesday	1	2.3	11.5	25.0
21	2016-04-25 17:32:17	17:48:33	Monday	1	2.4	12.0	25.0
22	2016-04-26 17:28:59	17:45:30	Tuesday	1	2.4	12.5	25.0
23	2016-04-28 18:18:09	18:44:56	Thursday	1	2.3	17.0	25.0
24	2016-05-17 17:48:50	18:09:43	Tuesday	1	2.3	14.0	25.0

Locations were intentionally left out to protect the innocent, but it is clear that these trip records are coming from single person (sometimes with a partner).

Our last step was trying to find a pair of trips within a day that had same tip amount but with a pick up and drop off location switched after rounding to the third digit. However, there was not a single trip within our labeled sample trip from January 2015 to June 2016.

In conclusion, we were not able to de-anonymize as much as we wanted at first due to the crowd nature of New York City, but were able to do some analysis of businesses. Privacy

concerns could be larger for when taxi trips are published by less populated city. However, we were able to identify an individual, though this person had highly unusual tipping behavior. Therefore, we argue that privacy should always be considered when publishin

VII. Conclusion

The result from the project is not so good as expected. One possible reason might be the sample size we use to do prediction in matching pair is really small. However, we trust when we get larger processing power to deal with true big data, it must outperform simple machine learning methods like multiple linear regression.