

Predicting NYC taxi trip durations for yellow taxi rides in 2016

Contents

| | |
|--|----|
| Version changes | 1 |
| 1. Introduction | 1 |
| 2. NYC trips data | 1 |
| 2.1 Extracting the data from city of New York's website | 1 |
| Data | 1 |
| Data properties | 2 |
| Data cleaning | 2 |
| 2.2 Adding features | 3 |
| Datetime features | 3 |
| Trip duration statistics | 3 |
| Distance statistics | 3 |
| Average speed statistics | 4 |
| 2.3 EDA | 4 |
| Trips on latitude vs longitude map | 5 |
| Plotting the rides across datetime features | 6 |
| Checking if <i>vendor_id</i> and <i>store_and_fwd_flag</i> matters | 7 |
| Distribution of pickup & dropoff latitudes and longitudes, and Distance | 9 |
| Regression plot trip_duration vs. the estimated distance | 10 |
| 3. Adding NYC taxi zones data (shapefile) | 11 |
| 3.1 Mapping the pickup and dropoff lat lon locations onto the taxi zones | 11 |
| 3.2. More EDA | 12 |
| 4. Adding trip direction | 15 |
| Trip direction distributions | 15 |
| 5. Clustering similar zones | 17 |
| 6. Machine Learning models | 1 |
| 6.1 Summary | 1 |

| | |
|---|---|
| 6.2 Variables to use..... | 1 |
| 6.2 Baseline to compare all models with | 1 |
| 6.3 Elastic Net Linear Regression | 2 |
| 6.4 Random Forest..... | 4 |
| Checking feature importance by permutation | 6 |
| 6.5 XGBoost | 1 |
| Checking feature importance by permutation | 3 |
| Using the trained XGBoost model on the entire dataset (~4M taxi rides)..... | 3 |
| 7. Comparing the Models..... | 1 |
| What the errors mean? | 1 |
| Checking which areas had low and high errors | 2 |
| Distribution of residuals categorized by NYC boroughs¶..... | 3 |
| 8. Future Work | 3 |

Version changes

| Date revised | Revisions | Made by |
|--------------|------------------------------|---------|
| 3/15/20 | Created the word doc report. | Pratha |
| | | |
| | | |
| | | |

1. Introduction

This project is inspired by the [NYC taxi trip duration](#) challenge hosted on Kaggle. But instead of using the train and test data from Kaggle, taxi data was queried directly from the city of NY's website. Only Yellow taxi trips were considered for this project. More information about different taxi options available in NYC can be found [here](#). The functions used and the detailed EDA and ML notebooks can be found [here](#).

The motive of the project is to identify the main factors influencing the daily taxi trips of New Yorkers. The taxi trips data is taken from the NYC Taxi and Limousine Commission (TLC), and it includes pickup time, pickup and dropoff geo-coordinates, number of passengers, and several other variables. The taxi trips considered in this project are only for the year 2016 and only those trips will be considered whose exact pickup and dropoff geo-coordinates are available.

Policy researchers at the TLC and NYC can use the project observations and ML models to observe changing trends in the industry and make informed decisions regarding transportation planning within the city.

Below is the overall process flow chart for this project.

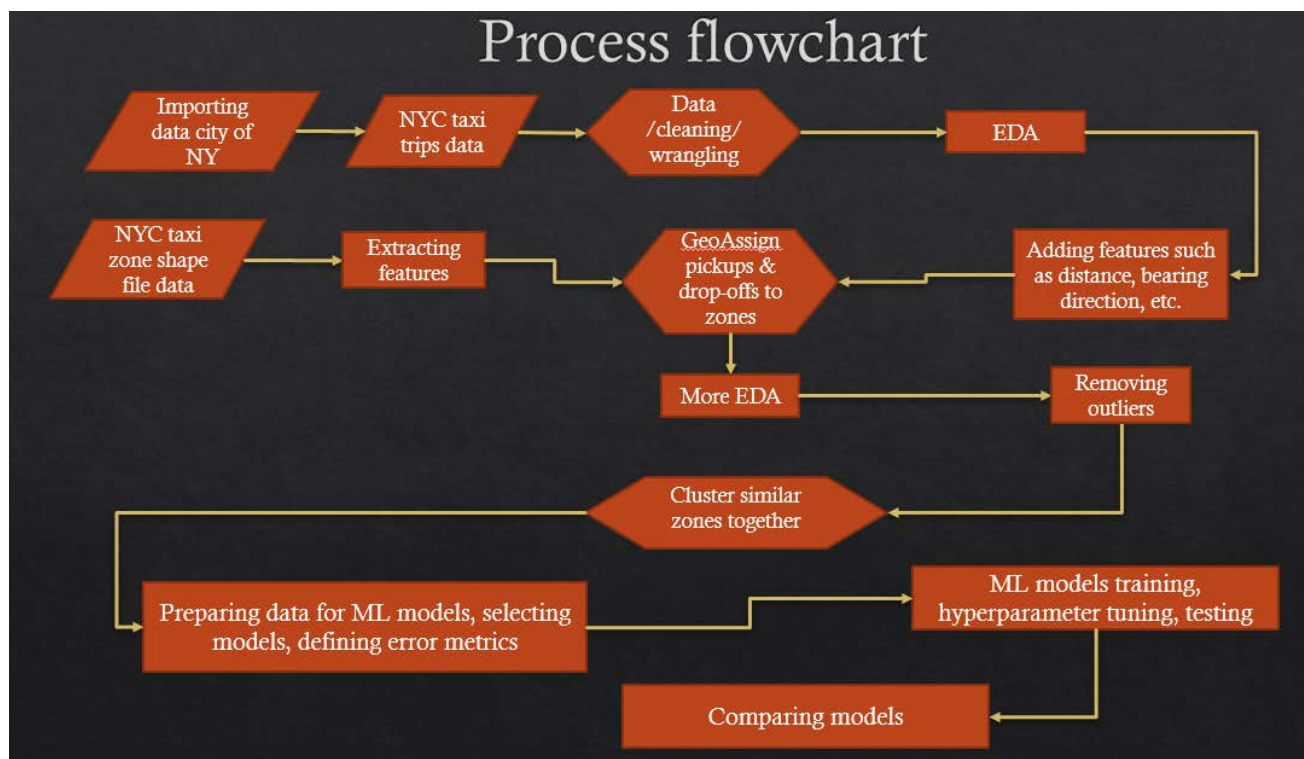


Figure 1 Process flowchart

2. NYC trips data

2.1 Extracting the data from city of New York's website

Data

The data was downloaded from city of New York's website using SQL queries enabled by [Socrata's API](#). More information on how to extract the data from city of New York's site can be found [here](#).

The total number of yellow taxi rides for the year of 2016 was around 131M but for the sake of working on a local machine only a fraction of the data was extracted (~4.2M).

To be more precise, 700,000 trip records from each month from January to June of 2016 were extracted randomly. Only the first 6 months were extracted because only the first 6 months have the exact pickup and dropoff latitude longitude locations.

Let's check the distribution of the NYC taxi trips per day of the month and per hour of the day for the above extracted data.

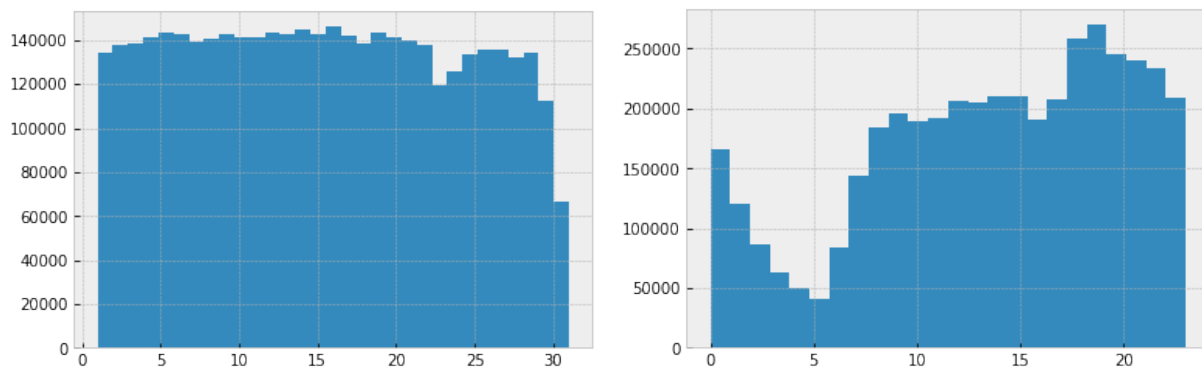


Figure 2 Taxi trips distribution on any given day of a month (added over all the months) shown on left and the same distribution shown for each hour of the day on the right

From the above two plots it is evident that the data pull was successful, and the trips were extracted uniformly across each day and hour of the day.

Data properties

Below are the variables from the data:

| | count | mean | std | min | 25% | 50% | 75% | max |
|-------------------|-----------|------------|-------------|-------------|------------|------------|------------|--------------|
| vendorid | 4200000.0 | 1.530776 | 0.499052 | 1.000000 | 1.000000 | 2.000000 | 2.000000 | 2.000000e+00 |
| passenger_count | 4200000.0 | 1.661372 | 1.310862 | 0.000000 | 1.000000 | 1.000000 | 2.000000 | 9.000000e+00 |
| trip_distance | 4200000.0 | 4.963236 | 2770.393796 | 0.000000 | 1.000000 | 1.700000 | 3.170000 | 5.361520e+06 |
| pickup_longitude | 4200000.0 | -72.920719 | 8.762610 | -131.818970 | -73.991699 | -73.981461 | -73.966309 | 0.000000e+00 |
| pickup_latitude | 4200000.0 | 40.170876 | 4.827042 | 0.000000 | 40.736206 | 40.753475 | 40.768158 | 5.124434e+01 |
| dropoff_longitude | 4200000.0 | -72.983583 | 8.500235 | -121.933372 | -73.991203 | -73.979462 | -73.962013 | 0.000000e+00 |
| dropoff_latitude | 4200000.0 | 40.206430 | 4.682630 | 0.000000 | 40.734600 | 40.753967 | 40.769550 | 4.918594e+01 |
| pulocationid | 0.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| dolocationid | 0.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

Figure 3 Numerical description of each variable in the data-frame

Luckily, we don't have any null values except the locationid columns which are just the zone ids, and we are going to remove them for now because we have the exact pickup and dropoff locations anyways. The first 6 months of the data on the city of New York's website has only the pickup and dropoff coordinates and the next 6 months only have the locationid values. The exact latitude longitude locations were removed post June 2016 because of data privacy concerns which are valid, and the dataset showed here should only be used for learning purposes.

Other variables present in the data include *pickup_datetime*, *dropoff_datetime*, *store_and_fwd_flag*. For more description on the data variables please refer to the [readme file](#).

Data cleaning

Next we will add some features to the dataset and at the same time also remove the outliers, which I have defined as below:

- *trip_duration* < 60 seconds OR *trip_duration* > 99.8th percentile value of the trip duration
- The *pickup* and *dropoff* latitudes and longitudes should be within the NYC boundary limits defined as,
nyc_long_limits = (-74.257159, -73.699215), *nyc_lat_limits* = (40.471021, 40.987326))
- *haversine_distance*¹ between the pickup and dropoff points is > 0 and < its 99.8th percentile value.

¹ Calculates the haversine distance between pairs of (lat, lon) points using the haversine distance formula. Refer to <https://janakiev.com/blog/gps-points-distance-python/>. For more accurate distance we should use the geopy.geodesic distance which calculates the shortest distance between two points on the earth's surface taking into account the ellipsoid nature of the earth's shape. But the geopy.geodesic calculations take time and for a small area like NYC where maximum of the taxi trip durations are below 30 miles we can use the haversine function to save considerable amount of time and not lose much of the accuracy. I compared the values from both the functions and 99.8% of the times the difference in those values is less than 0.37 miles. So, using haversine here.

- We will also add new features and remove outliers after some EDA based on our observations.

2.2 Adding features

Datetime features

Adding features based on the pickup datetime. Since, we won't have the dropoff datetimes for the test set we will use only the *pickup datetime* to extract the-

- *day, hour, month, holiday and weekday* variables.

Trip duration statistics

```
Total number of trips = 4,086,440 (after cleaning)
1% of the trips were below 1.83 minutes
25% of the trips were below 6.68 minutes
50% of the trips were below 11.08 minutes
75% of the trips were below 17.95 minutes
95% of the trips were below 34.70 minutes
99% of the trips were below 54.60 minutes
99.8% of the trips were below 73.42 minutes
```

We see that most of the trips were below 18 minutes and only a few were above 1 hour. That means most of the NYC yellow taxi trips are short trips, maybe people travelling within the same borough or from subways to their homes and offices or vice versa.

Distance statistics

```
Total number of trips = 4,086,440
1% of the trips were below 0.21 miles
25% of the trips were below 0.78 miles
50% of the trips were below 1.31 miles
75% of the trips were below 2.42 miles
95% of the trips were below 6.80 miles
99% of the trips were below 12.88 miles
99.8% of the trips were below 14.07 miles
```

The original dataframe did have a column for *trip_distance* which was removed during the cleaning because we won't have this variable for the test set anyways and using it will result in data leakage across the test set into the training set. The actual *trip_distance* and the calculated distance values were compared, and it was observed that,

- the calculated haversine distance values were a bit lower than the actual trip distance values, this is because the haversine distance calculates the distance along the curvature of the earth on a straight path which will mostly be shorter than the actual path. But it's a good approximation nevertheless as we cannot use the actual trip distance values.

- But a useful observation can be seen here that 75% of the trips are below 3 miles, so most of the trips were short distance trips within neighboring taxi zones. This is true for the yellow taxi rides within NYC, especially the ones within the busy areas of Manhattan.

Average speed statistics

The average speed was calculated by dividing the *distance_haversine* by the *trip_duration* and converting it to miles per hour (mph).

Total number of trips = 4,086,440

1% of the trips were below 1.87 mph

25% of the trips were below 5.73 mph

50% of the trips were below 7.98 mph

75% of the trips were below 11.09 mph

95% of the trips were below 18.18 mph

99% of the trips were below 25.00 mph

99.8% of the trips were below 30.45 mph

- The lower average speed values seem appropriate for a busy city like NYC.
- We need to check the speeds above the 99.8% value and see if those trips were really that fast or they are indeed outliers (removed later).

2.3 EDA

After cleaning and wrangling the data, EDA was done on the dataframe to observe any patterns, or outliers, or relationships within the data.

The plots are shown below with descriptions and observations below the plot.

Plotting the trip duration bin count.

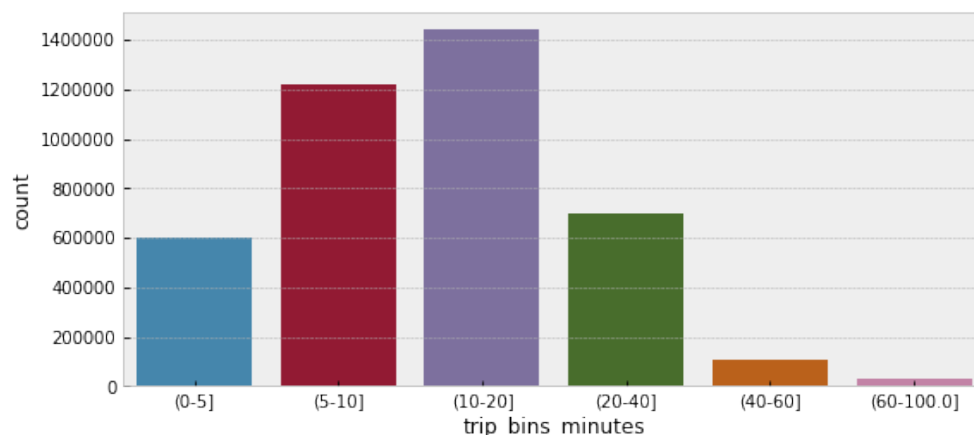


Figure 4 Distribution of the trip duration (y variable)

- As seen before from the stats, most of the trips were less than 1-hour trips and within that range most of the trips were between 5 to 20 minutes. The longer duration trips can mean heavy traffic, or they can represent airport trips.

Trips on latitude vs longitude map

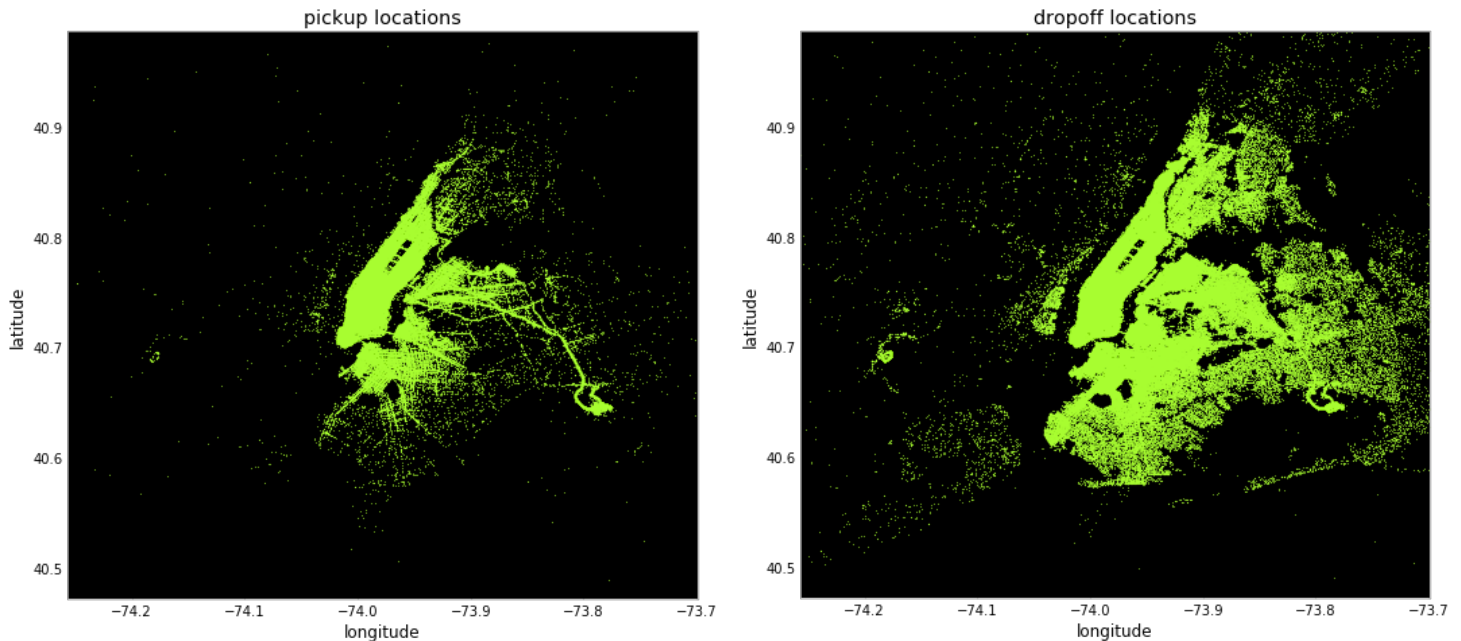


Figure 5 NYC yellow taxi pickups and dropoffs (2016) plotted on the latitude vs longitude matrix

- The scattered points give out the overall layout of the NYC boroughs and we can see that the density is highest in the Manhattan area, especially for the pickups. The dropoff locations are more spread out.
- Pickups and Dropoffs are denser in the John F. Kennedy International Airport area (south east corner, in Queens) whereas only dropoffs are denser in the Newark Liberty International Airport area (mid-west) because the Newark airport is served majorly by the Newark taxis and not by NYC taxis.
- LaGuardia Airport (east of Manhattan, in Queens) also seems to be served well by the yellow taxis.
- Also, since the yellow (medallion) taxis are able to pick up passengers only in the five boroughs of NYC, the pickup locations are mostly within NYC but the dropoffs can be outside the limits as seen from the plots.

Plotting the rides across datetime features

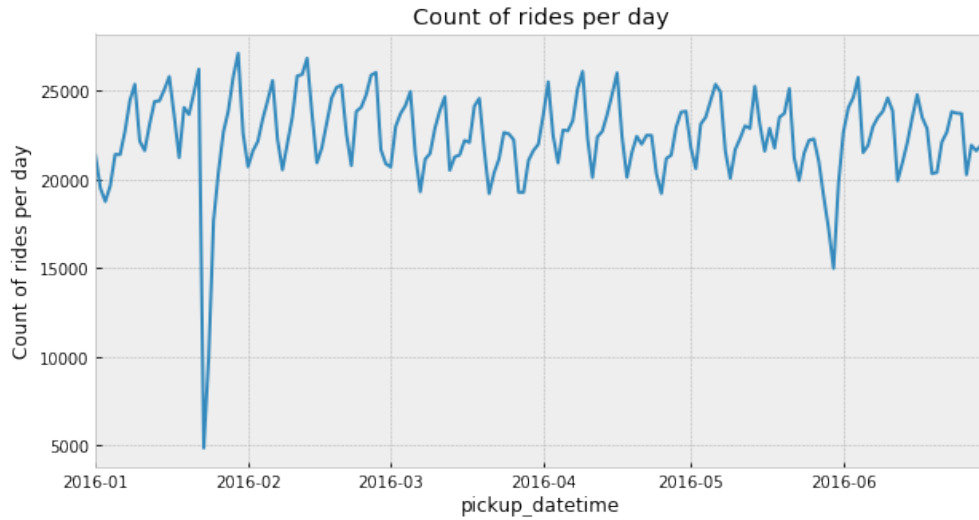


Figure 6 Count of rides per day

- The number of rides per day is cyclical with a dip in the rides during the nighttime.
- The sudden dip in the taxi rides between the 20th and 26th Jan was because of the heavy snow that was observed during that period.

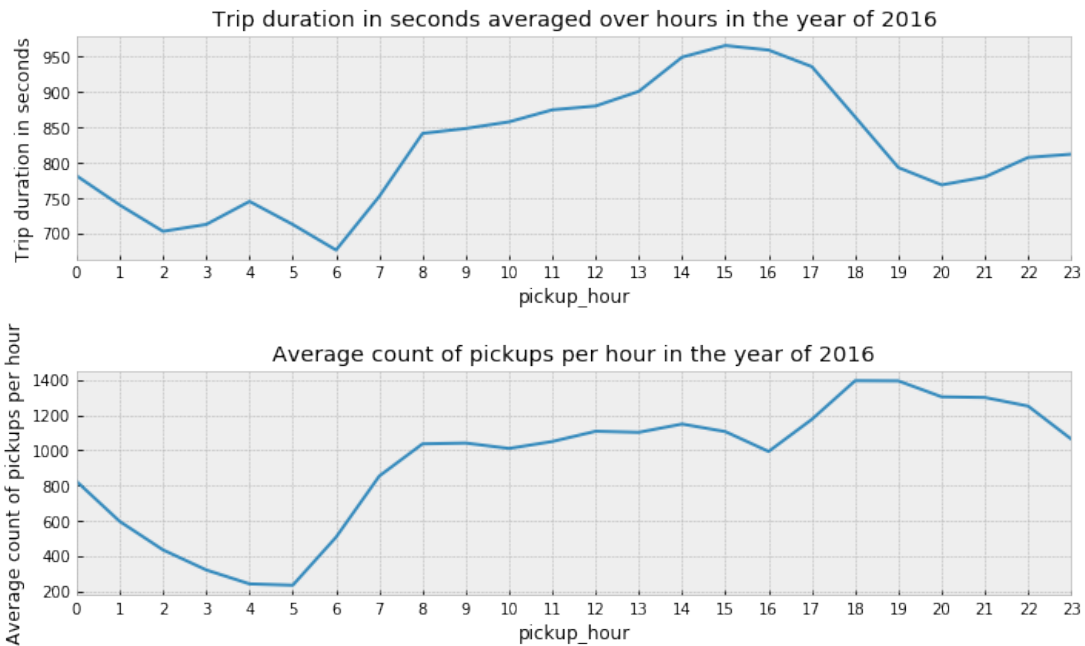


Figure 7 (Top) Trip duration in seconds averaged over each hour and (Bottom) Average count of pickups per hour

Note: Please notice that the y-axis doesn't start at 0.

- As one would expect the trip duration and the number of rides are higher in the evening time. As the number of rides go up in a certain area one can expect the resulting traffic to slow down the traffic increasing the trip duration time.

- From the 2nd plot we can see how the pickup rides increase steadily from 5am to 8am only to flatten out between 8am to 4pm and then again steeply increase from 4pm to 6pm to fall down later.
- The average trip duration is around 15 minutes during the evening time.

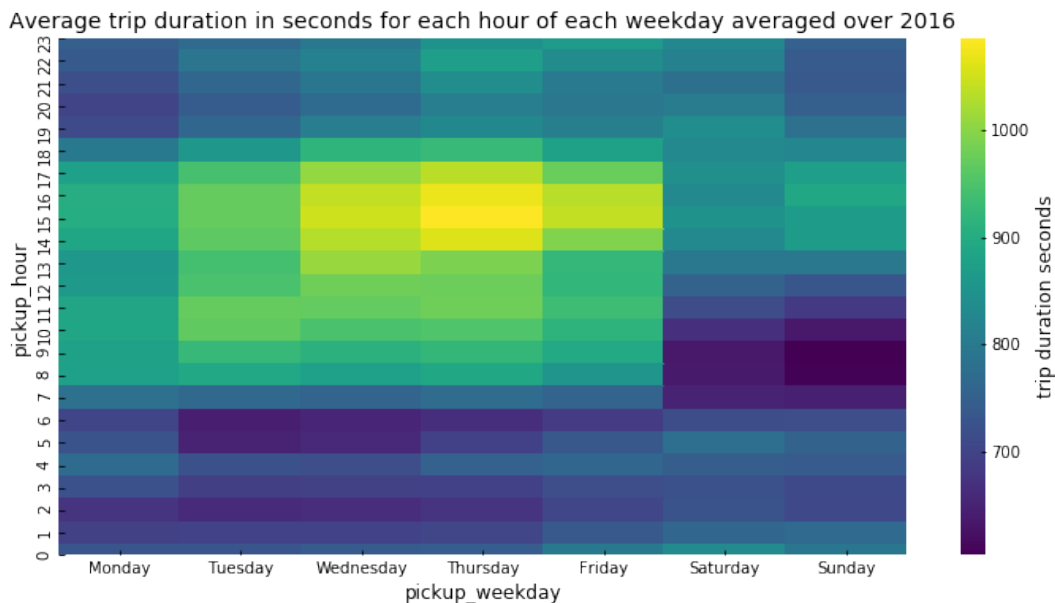


Figure 8 Average trip duration in seconds for each hour of each weekday

Checking if *vendor_id* and *store_and_fwd_flag* matters

vendor_id is a code indicating the TPEP provider that provided the record.

- 1= Creative Mobile Technologies, LLC;
- 2= VeriFone Inc.

store_and_fwd_flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server.

- Y= store and forward trip;
- N= not a store and forward trip.

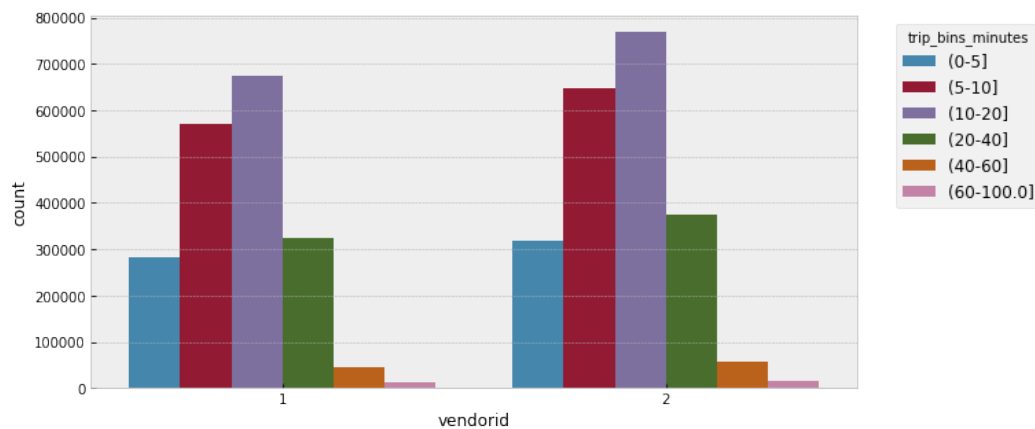


Figure 9 Distribution of vendor_id across different trip duration bins

- *vendor_id* doesn't seem to have any difference on the *trip_duration* and logically it seems right because it's just the provider which provides the record, and doesn't necessarily have any physical influence on the trip duration of a trip, except any margin for data reporting errors.

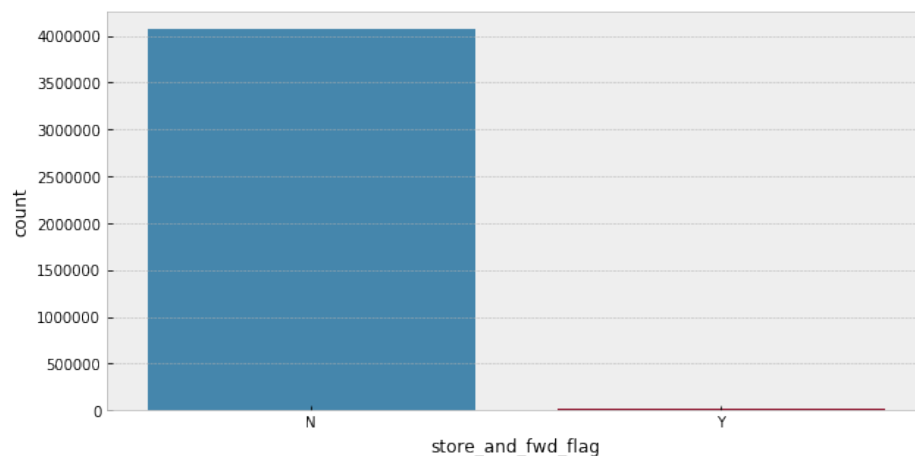


Figure 10 Distribution of store_and_fwd_flag variable

- *store_and_fwd_flag* has only 0.55% of the trips marked as Y and rest of the trips are marked as N. So, this variable shouldn't have any effect on the *trip_duration* as well.

Distribution of pickup & dropoff latitudes and longitudes, and Distance

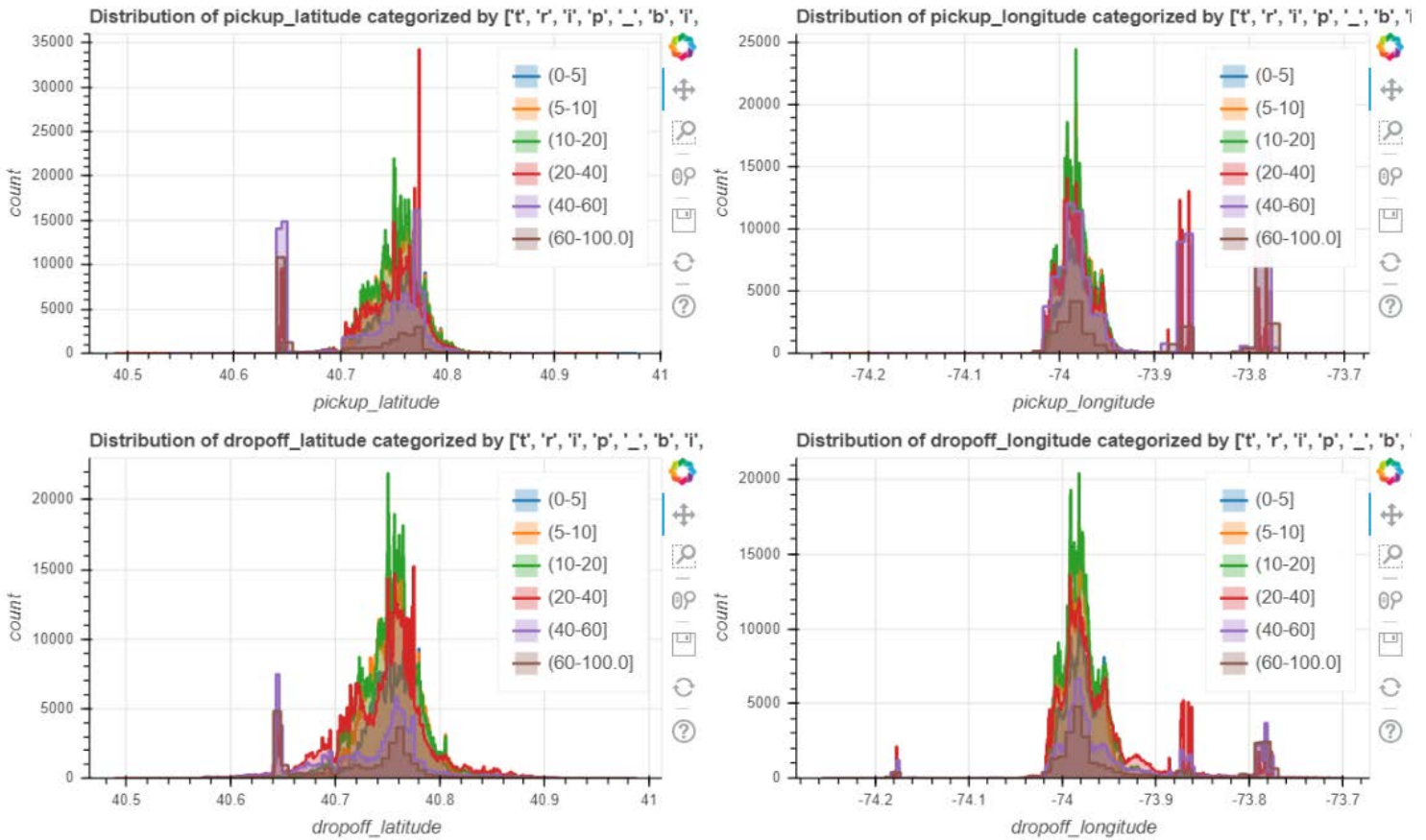


Figure 11 Distribution of pickup & dropoff lats lons categorized by trip duration bins

- Most of the trips that were 0 to 20 minutes long lie between the 40.68 and 40.82 latitude range and the -74.02 to -73.94 longitude range, which mostly covers the Manhattan region. So, we can say that most of the trips with shorter trip durations were located within the Manhattan borough.

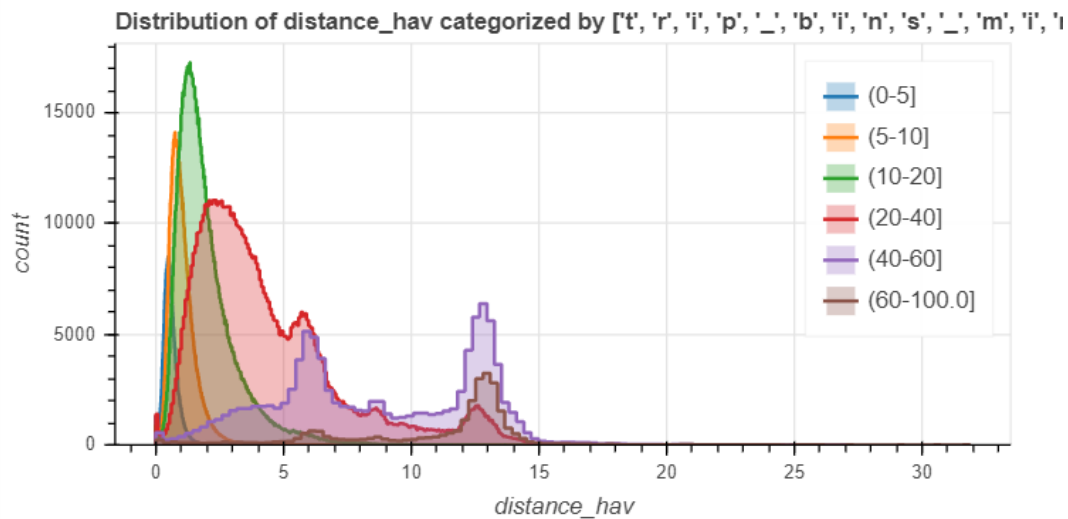
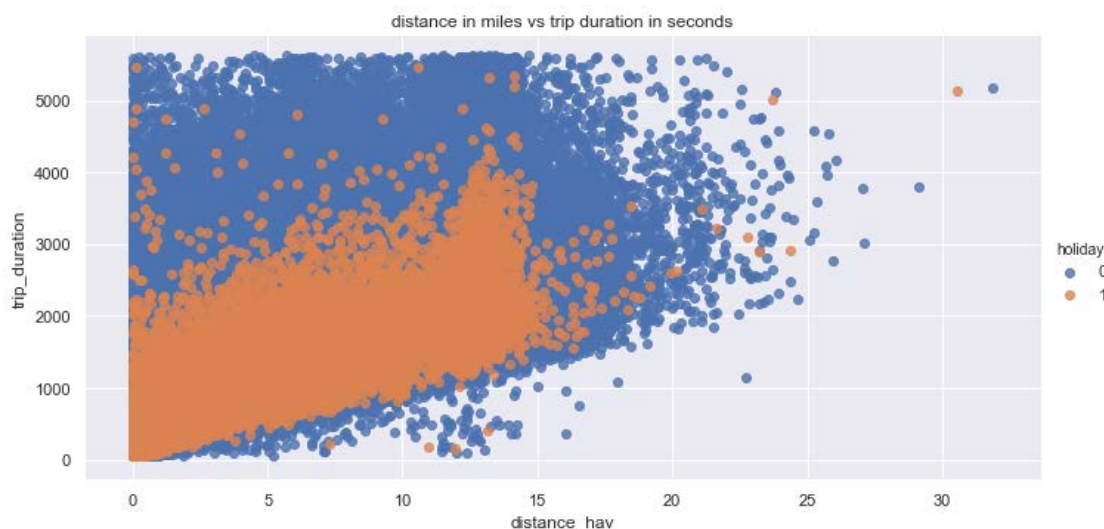


Figure 12 Distribution of estimated distance categorized by the trip duration bins

- The distribution of distance is narrow and located between 0 to 7 miles for the shorter trips from 0 to 20 minutes as expected.
- As the *trip_duration* bins go higher the distance distributions shift to the right because trip duration is directly proportional to the distance travelled.
- The >20 and <60 trip duration bins have a wide spread of distance values with multiple peaks, whereas the 60-100 bin has the distance distribution between 10 to 15 miles indicating long trips. The two peaks in the 20-40 and 40-60 *trip_duration* bins can indicate that there are other factors that affect the trip duration, but it also can mean that our estimated distance values are lower than the real values especially for the lower peaks.

Regression plot trip_duration vs. the estimated distance



- There seems to be a strong correlation between the distance and the trip duration as expected.

- And the spread of the trip duration is narrower during the holidays as compared to regular working days.
- We can also see the correlations using the `scipy.stats.pearsonr` function. The `scipy.stats.pearsonr` function returns the Pearson coefficient and the pvalue of observing such coefficient if we were to assume that there was no correlation between the x and y data sets. Let's select a significance level of 5%, and so if the pvalue is <5%, we will assume that the correlation coefficient returned is significant. The correlation coefficient came out to be 0.7 at a p-value of ~0.0.
- We can assume that the haversine distance, though not an exact ride distance, is a useful variable in predicting the trip duration and it also seems logical because keeping other factors more or less constant the trip duration of a trip is directly proportional to the distance travelled.

To better visualize the trips within each NYC Taxi zone and the 5 boroughs lets combine the taxi trips dataset with the NYC taxi zones shape files

3. Adding NYC taxi zones data (shapefile)

To better visualize the trips within each NYC Taxi zone and the 5 boroughs lets combine the taxi trips dataset with the NYC taxi zones shape files.

3.1 Mapping the pickup and dropoff lat lon locations onto the taxi zones

The NYC shape files were downloaded from [here](#).

| | zone | LocationID | borough | geometry |
|---|-------------------------|------------|---------------|---|
| 0 | Newark Airport | 1 | EWB | POLYGON ((-74.18445450453645 40.69500424662989... |
| 1 | Jamaica Bay | 2 | Queens | (POLYGON ((-73.82337735819013 40.6389952954719... |
| 2 | Allerton/Pelham Gardens | 3 | Bronx | POLYGON ((-73.84792754846802 40.8713505269578,... |
| 3 | Alphabet City | 4 | Manhattan | POLYGON ((-73.97177554799167 40.72582954148977... |
| 4 | Arden Heights | 5 | Staten Island | POLYGON ((-74.17421887421997 40.56257630700625... |

Figure 13 NYC shape file dataframe

- The nyc shape file data includes the zone name, the zone id as LocationID, borough name and the geometry that defines each zone. It's a geodataframe and has geo features such as we can find the centroid of the above geometries and also check if any lat, lon points fall within each zone.
- There are about 263 taxi zones and the 5 boroughs are 'EWB', 'Queens', 'Bronx', 'Manhattan', 'Staten Island', 'Brooklyn'.

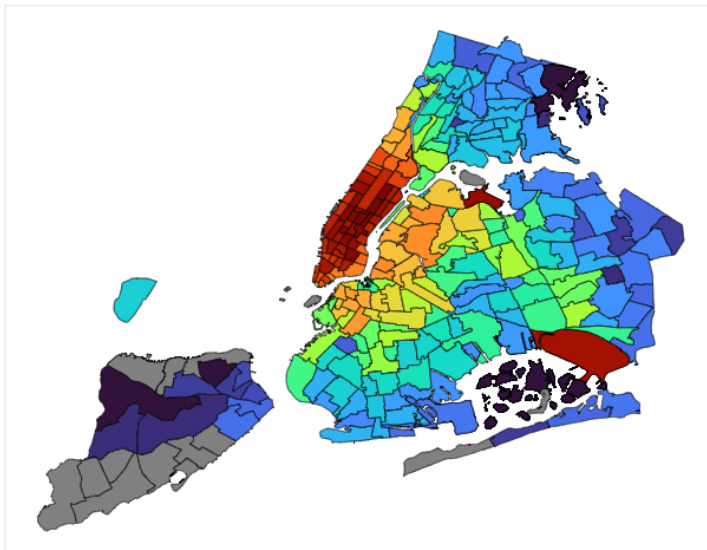
The pickup and dropoff latitude, longitude pairs were mapped onto the above zone dataset and each taxi trip was assigned a pickup zone and a dropoff zone accordingly. For more details on how the assigning was achieved please refer to section 3.1 of the [EDA notebook](#).

- After the assigning the zones it was observed that around 0.18% of the 4.2M trips were outside the defined taxi zones.
- Plotting the above points on the google maps using [bokeh](#) showed that most of the points were very far away from NYC or in waters. Since, these points represent only 0.18% of the data, they were removed from the dataset.

3.2. More EDA

Trips within each zone showing their total count and also the mean trip duration.

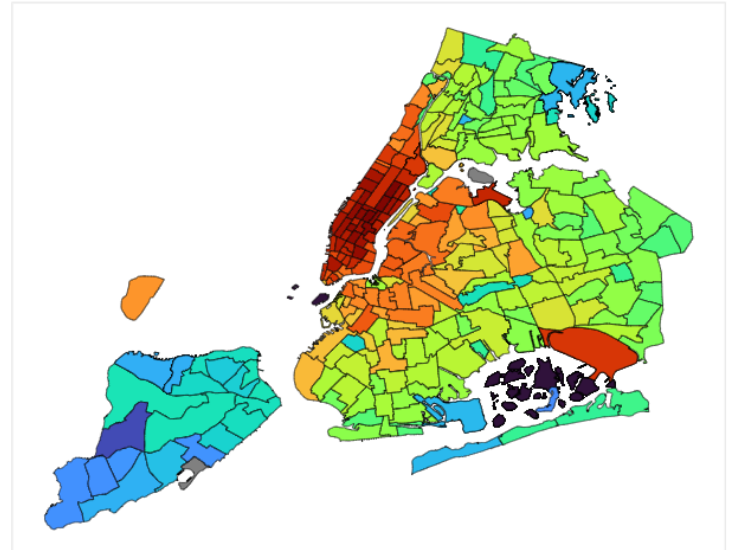
NYC Taxi pickups count map



Total number of pickups



NYC Taxi dropoffs count map



Total number of dropoffs

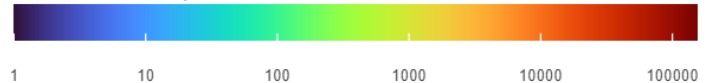
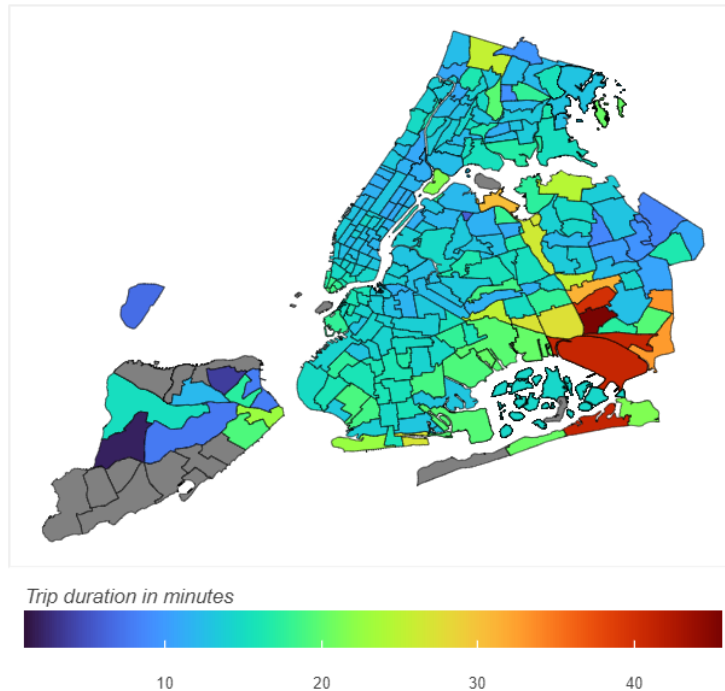


Figure 14 Total trips picked up and dropped off within each zone

- Around 100,000 or more trips originate and end in the Midtown and Lower Manhattan areas and also the Queens' airports- JFK and La Guardia.
- The dropoffs in these areas except the airports are comparably of the same magnitude. The airports have almost half the number of dropoffs as pickups.

NYC Taxi pickups trip_duration map



NYC Taxi dropoffs trip_duration map

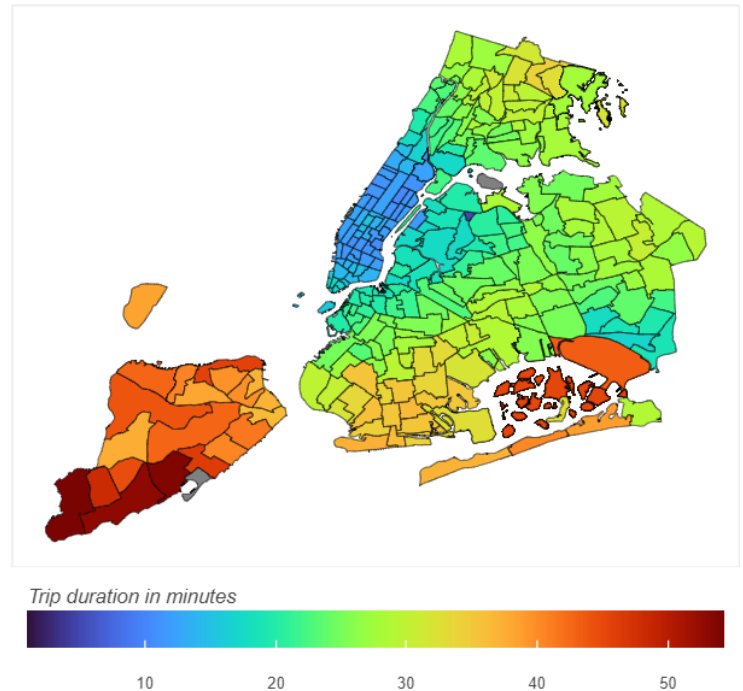


Figure 15 Average trip duration of trips picked up and dropped at each zone

- The average trip duration for rides picked up at the JFK airport is ~40 mts. and it takes on an average around 43 mts. for people from within NYC to reach the JFK airport. The same values for the La Guardia airport are 28 and 30 mts respectively.

Manhattan pickup and dropoff points were plotted on google maps and it was observed that most of the clusters, except the airports, are around the subway stations in Manhattan like the Penn station and also around popular areas like Times square.

Plotting the pickup and dropoff points side by side for trips that either started or ended at the JFK airport (zone id = 132).

This is an interactive visualization where you can change the hour of the day and weekday in the plot and the plot updates the data accordingly based on your input. You can select any weekday value on the checkbox or slide across the hours to realize the flow of traffic to and from the airport.

The left plot shows all the pickup points for trips that ended at JFK airport and similarly the right plot shows all the dropoff points for trips that started at the JFK airport.

Note: If instead of running a copy of the EDA notebook, you are just viewing it on Github or nbviewer the above plot with callback functions associated with it won't be visible. So, please find below the same plots in static mode for 7am and 4pm on a Monday.

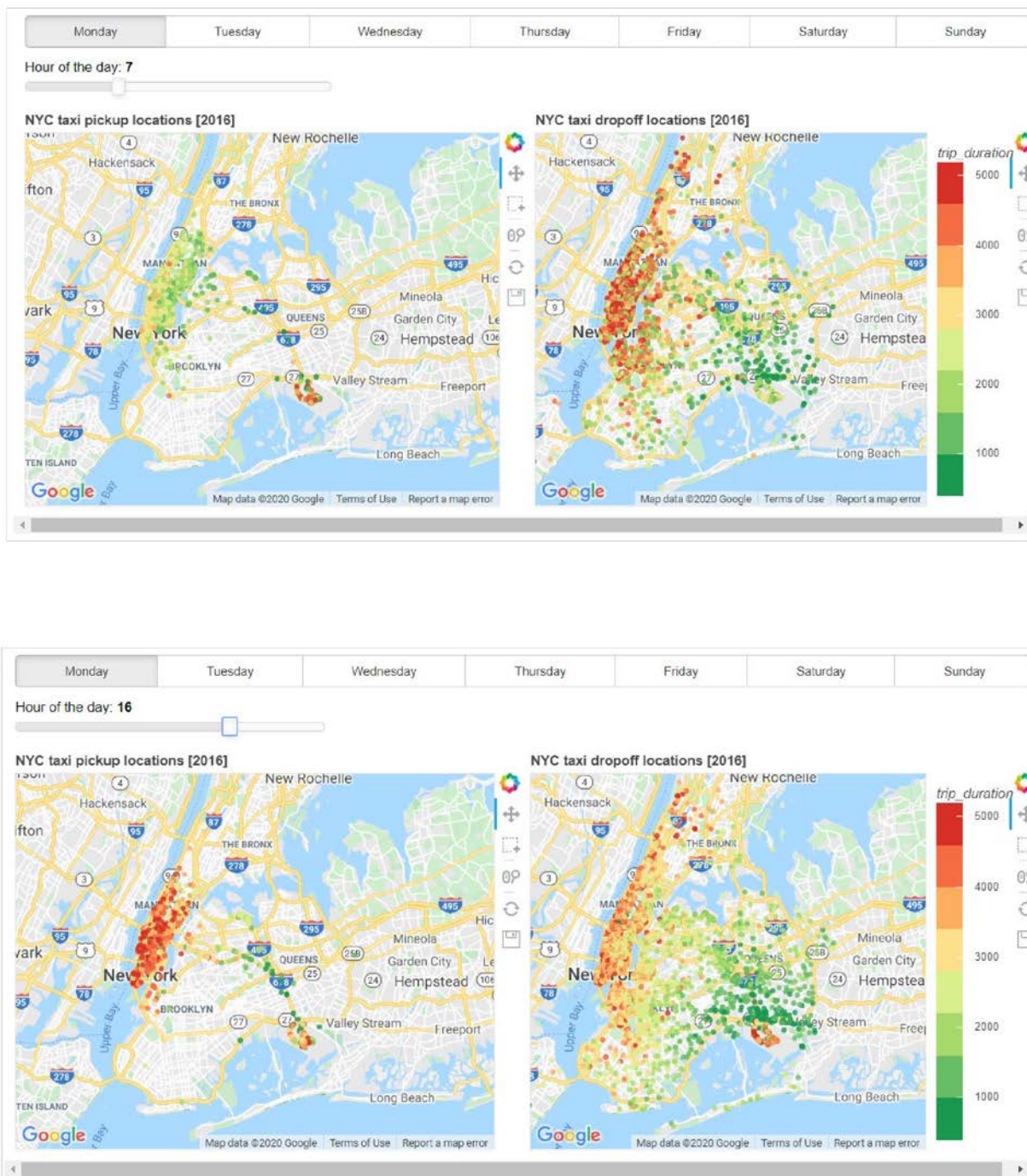


Figure 16 Pickup and dropoff locations for JFK airport. (Top) 7am on Monday, (Bottom) 4pm on Monday. (Left side) All the pickups that ended in JFK and (Right side) all the dropoffs from the JFK

- The travel time from the airport into the city (Manhattan, Queens, Brooklyn) and from the city to the airport ranges from 20 to 40 minutes from the midnight to early morning.
- The same time between the city and the airport increases significantly towards the higher range of 40 to 80 minutes during the day, peaking during the evening hours.

- One important observation is that, in the morning hours (6 am onwards) the traffic from the airport towards Manhattan is pretty slow compared to the traffic from Manhattan to the airport. And between 2 pm to 6 pm the traffic keeps getting slower in the Manhattan to Airport direction and in this time range it is slower than the opposite traffic. This can be attributed to the fact that many people travel for work to NYC from outer boroughs in the morning increasing the traffic in that direction whereas many taxis head out of Manhattan to outer boroughs in the direction of the airport in the evening time making the traffic slower in that direction. *NOTE: You can also select a single or multiple pickup or dropoff points using the box-select tool and the plot will highlight the corresponding dropoff or pickup points in the adjacent map respectively.

Removing outliers based on calculated speed

- The slow trips were defined as anything below 0.5 mph and faster trips were defined as anything above 50 mph. These values were chosen based on the distribution of the speed variable and by using the above geomap interactive graph function to analyze the pickup and dropoff points. They represent about 0.32% of the data.
- The slower trips were found to be too slow, and their travel distance was less than one mile with trip durations reaching up to 60 minutes. And the faster trips were long distance trips, but the time taken for them was surprisingly low.
- Since these trips seem as outliers, they were removed from the dataset.

4. Adding trip direction

It was observed, especially for the airport trips, that the direction of the trip also has some effect on the trip duration. And there are many one-way streets in Manhattan forming a grid structure as seen in the maps. So, for example, someone at Penn station going north towards Central park will try to hail a cab from the road catering traffic flowing to north.

Let's add the bearing of each trip which simply means the overall direction in which the taxi travelled from the pickup point to the dropoff point.

The convention followed here is such that the North is denoted as 0 degrees, East as 90 degrees, South as 180 degrees and circle back to North as 360 degrees.

Trip direction distributions

Trip directions for all boroughs except Manhattan

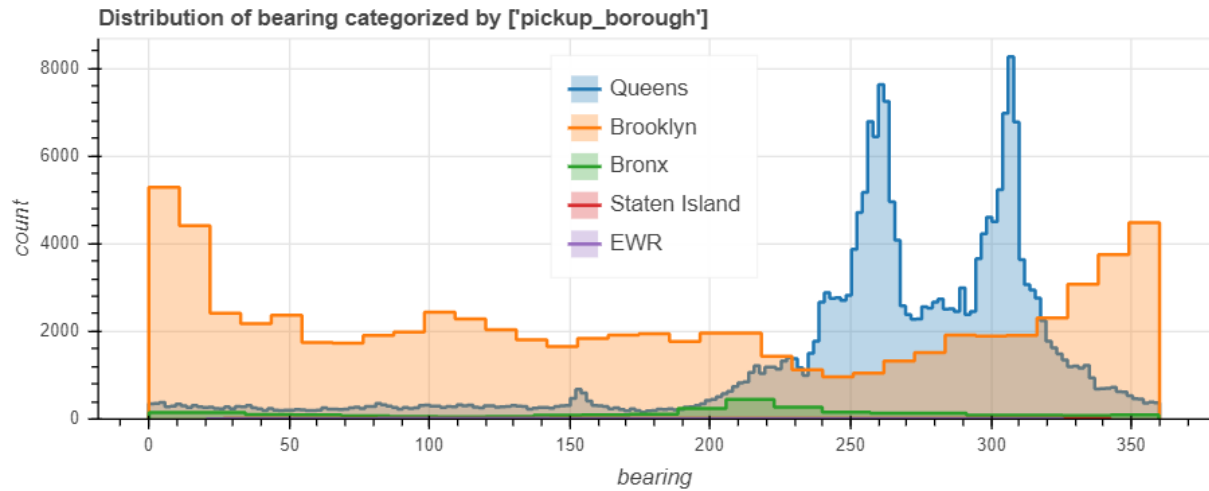


Figure 17 Trip bearing (direction) of travel from pickup to dropoff for all boroughs except Manhattan

- Most of the trips from Queens are directed towards West and North-West. This kind of makes sense because La Guardia airport lies in the North-West (of Queens) and also if a taxi has to travel from Queens to Manhattan it has to travel in West direction.
- Trips from Brooklyn seem to be spread out uniformly across all directions, but most common direction is North (0 Deg and 360 Deg) which indicates travel to Manhattan.
- Trips from Bronx are also directed mostly in the South-West direction indicating travel to Manhattan.

Trip direction of all boroughs categorized by *speed_mph*

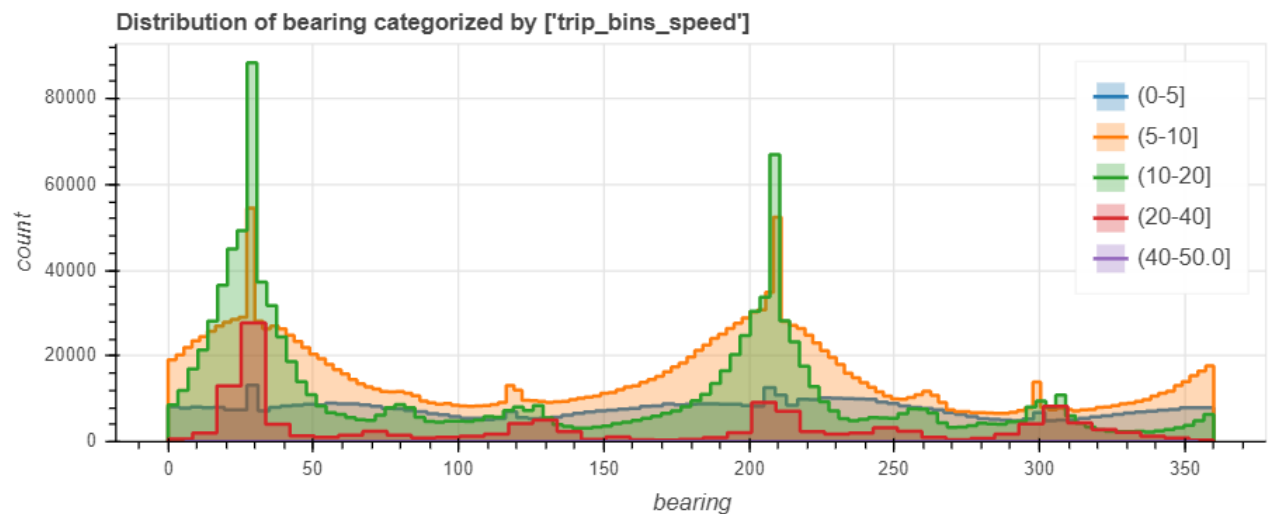


Figure 18 Trip bearing (direction) from pickup to dropoff for all boroughs categorized by average speed bins

Note: This graph includes trips originating from Manhattan unlike the previous graph which showed the other boroughs.

- We can see distinct peaks in the North-East and South-East direction which wasn't very dominant in the graph for other boroughs. So, these trips represent mostly the trips originating from Manhattan and the direction of travel makes sense given the stretch of Manhattan in the North-East to South-West direction.
- The slower trips (0-5 mph) are spread uniformly across all directions indicating short distance travel within the boroughs. Whereas the faster trips (5-10 mph and 10-20 mph) seem to spread in the North-East and South-West direction which can be travel within Manhattan and also travel from other boroughs to Manhattan.
- We can see other peaks for the faster trips (20-40 mph) in the South-East and North-West direction which can be the travel to and for Manhattan to JFK airport.

5. Clustering similar zones

We have the NYC map divided into taxi zones, but the location id of these zones don't follow any pattern and also there are total 263 zones, so, to make the locations more useful and reduce the total number of clusters we will re-cluster the zones into larger clusters.

The attributes that will be used to do the clustering are the centroid latitude and longitude of each zone and the traffic density in each zone at each hour. Traffic density is defined as the number of dropoffs in a zone at each hour minus the number of pickups in that zone for the corresponding hour, kind of indicating how busy the zones are at any particular hour.

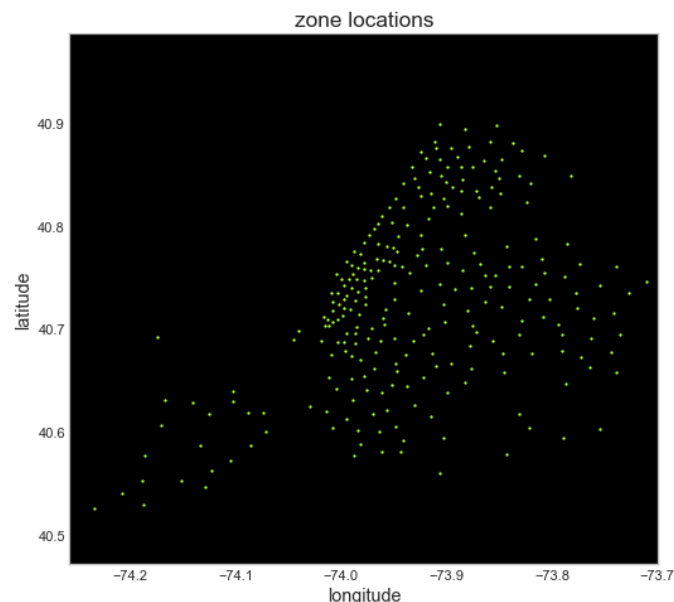


Figure 19 Centroids of the taxi zones in NYC

| | borough_num | zone_lat | zone_lon | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... |
|------------|-------------|-----------|------------|----------|----------|-----|----------|----------|----------|----------|-----|
| LocationID | | | | | | | | | | | |
| 1 | 2 | 40.691839 | -74.174002 | 1.000000 | 1.000000 | 1.0 | 1.384615 | 1.302158 | 2.636943 | 1.867089 | ... |
| 2 | 4 | 40.616754 | -73.831300 | 0.000000 | 0.000000 | 0.0 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | ... |
| 3 | 0 | 40.864482 | -73.847424 | 1.095238 | 1.111111 | 0.0 | 0.153846 | 0.000000 | 1.000000 | 1.000000 | ... |

Figure 20 Dataframe where each row indicates information about a single taxi zone

A dataframe where each row represents a single taxi zone (1 to 263) was created by merging the NYC taxi trip data and the NYC shape file. Then the average pickup count at each hour was subtracted from the average dropoff count at that corresponding hour for each zone. So, the variables for each zone were – *borough_num* (borough name label encoded), *zone_lat* and *zone_lon* (centroid lat lon of each zone) and the average dropoff minus pickup count for each hour 0 to 23.

We will use the Agglomerative Hierarchical Clustering method here for the sake of simplicity and interpretability.

More details on the hierarchical clustering can be found [here](#).

There are many methods that can be used to form the clusters including the single linkage, complete linkage, average linkage, centroids, ward's method, etc. Ward's or minimal increase of sum-of-squares method was used here for the clustering. Basically, when joining two clusters this method evaluates how does the joining of two clusters affect the sum of squared distances of the individual points/clusters from the centroids. Keep in mind that hierarchical clustering is a very space and time-consuming method to train a large dataset but here we already have taxi zones defined and we are going to cluster only the taxi zones together. We can then use the taxi zone clusters to cluster the individual pickup and dropoff locations. So, we will only be clustering 263 rows here, which doesn't take more than a few seconds. Also, the variables were normalized before feeding the dataset to the clustering algorithm.

We visualized the clusters using a dendrogram which is a tree-like diagram that records the sequences of merges or splits. The X-axis represents the rows of the dataframe that are clustered together, and Y-axis displays the proximity; the higher you go along the Y-axis the proximity gets weaker. We can either choose a threshold on y-axis such that the clusters only up to that threshold will be considered. For example, if you use a threshold of 0.25, the number of clusters comes to around 50. We can also directly feed in the number of clusters we want from the clustering model. After some back and forth with the scatter plot I have chosen 50 here. It divides the city and outskirt areas along with the airports pretty well. Please see EDA notebook for the dendrogram plot.

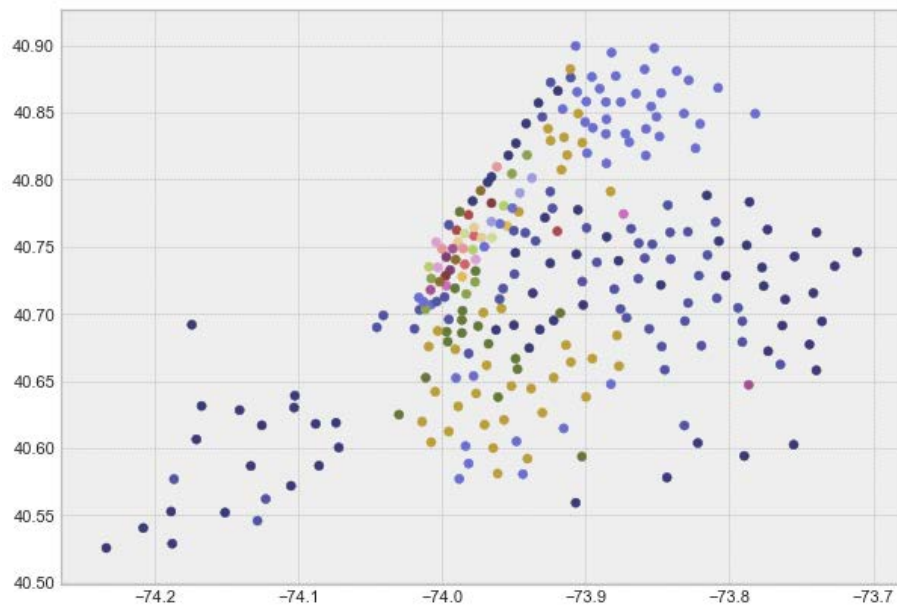


Figure 21 NYC taxi zones clustered together

- We can see from the above plot that the total number of clusters has been reduced from 263 to a mere 50 but the overall proximity and taxi trips density similarities has been retained among the clusters.
- The above cluster labels were then used to cluster the entire 4.2M rides into 50 clusters based on their pickup and dropoff taxi zones.

6. Machine Learning models

6.1 Summary

Linear regression (with only few variables), Random Forest and XGBoost were fit on the NYC taxi trip data with additional features added from the EDA notebook. For training, only a small sample (500,000) of the ~4.2M trips was used but the best model was tested on the entire dataset to verify that the training sample was selected randomly and represents the entire dataset indeed.

XGBoost with some hyperparameter tuning gave the best result of RMSLE = 0.32.

Note: RMSLE (root mean squared log error) was used here as the determining metric because we didn't want large errors to be significantly more penalized than smaller ones. RMSLE penalizes underestimates more than overestimates. Other metrics such as Root mean square error (RMSE), Mean absolute error (MAE), R2 score and Mean absolute percentage error (MAPE) were also calculated and the best XGBoost model gave the best results across all these metrics.

6.2 Variables to use

We cannot use all the columns from the dataframe because some of them are datetime and some of them were calculated based on the y variable such as *speed_mph*.

Also, the data from the pickup and dropoff taxi zone ids and the boroughs is already captured in the *zone_cluster* columns so we remove those variables as well.

So, we will use only the following columns:

```
cols_to_use = ['vendorid', 'passenger_count', 'pickup_longitude', 'pickup_latitude',  
'dropoff_longitude', 'dropoff_latitude', 'pickup_month', 'pickup_day', 'pickup_hour',  
'pickup_weekday', 'holiday', 'distance_hav', 'bearing', 'pickup_zone_cluster',  
'dropoff_zone_cluster', 'trip_duration']
```

The training set consists of 70% of the data and the test set thus contains the remaining 30% of the data.

6.2 Baseline to compare all models with

We will treat baseline as the average trip duration for all the trips in the training data.

The error metrics of the baseline model prediction of the test set are:

RMSE or Root mean squared error: 639.34


```
RMSLE or Root mean squared log error: 0.77  
Variance score: -0.00  
Mean Absolute Error: 460.26  
Mean Absolute Percentage Error: 94.37 %
```

To be considered as good models, our models should be much better than the baseline.

6.3 Elastic Net Linear Regression

We have many variables such as the day of the month, weekday, hour of the day, bearing direction, zone_cluster, etc. which are not linear variables and can be difficult for a linear regression (LR) algorithm to model without first converting these variables to appropriate forms that can be fed to the LR models and understood by it. And, we also need to scale the data to prevent the dominance of larger magnitude variables in LR models.

Another alternative is we can use non-linear methods such as decision trees to fit the data. Decision tree doesn't require us to convert the variables because it can split across any values of the variables and also, we don't need to scale the data when using trees because each variable is considered separately for calculating the gain at each branching.

But before moving onto decision trees or random forest let's check the performance of LR using only few significant variables such as the distance, hour and weekday. Since the hour and weekday are cyclical variables they were first transformed into sine and cosine terms using Fourier transform. Now, the LR can figure out that in reality 0th hour and 23rd hour are actually as much closer to each other as the 0th hour is to the 1st hour; same with the weekday. And we chose distance because based on the EDA and common intuition it can be assumed to be the most important variable in predicting the trip duration.

The hyperparameters that were tuned for the Elastic net model were '*l1_ratio*' and '*alpha*'.

- *l1_ratio*: The ElasticNet mixing parameter, with $0 \leq l1_ratio \leq 1$. For *l1_ratio* = 0 the penalty is an *L2* penalty. For *l1_ratio* = 1 it is an *L1* penalty. For $0 < l1_ratio < 1$, the penalty is a combination of *L1* and *L2*.
- *alpha*: Constant that multiplies the *L1* and *L2* penalty terms. *alpha* = 0 is equivalent to an ordinary least square without any regularization.

The error metrics for the tuned Elastic Net LR using only the distance, hour and weekday variables are as below:

```
Error metrics for model Tuned Elastic Net (w/ only dist, hr, wday)  
RMSE or Root mean squared error: 383.49  
RMSLE or Root mean squared log error: 0.49  
Variance score: 0.64  
Mean Absolute Error: 266.99  
Mean Absolute Percentage Error: 47.76 %
```

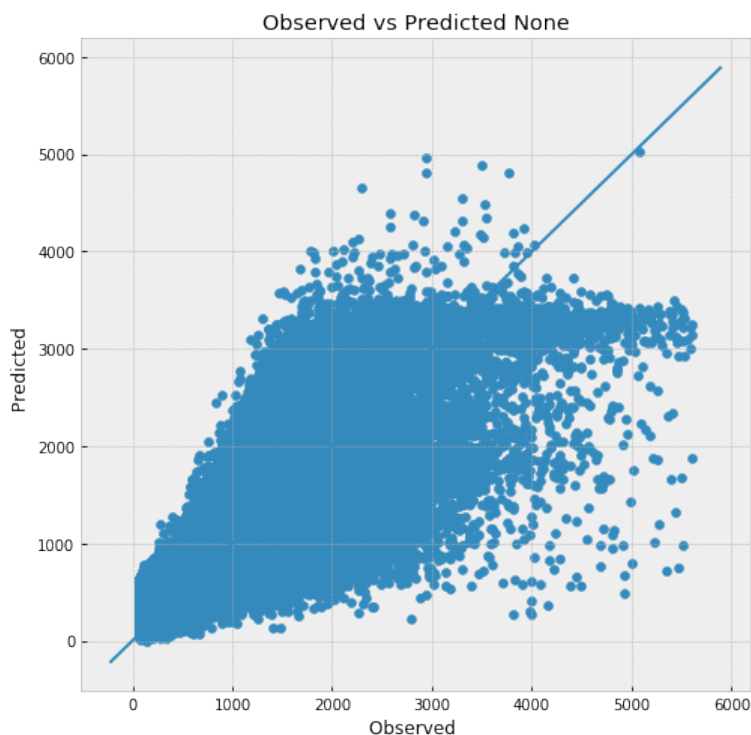


Figure 22 Observed vs Predicted trip duration in seconds for the Elastic Net LR

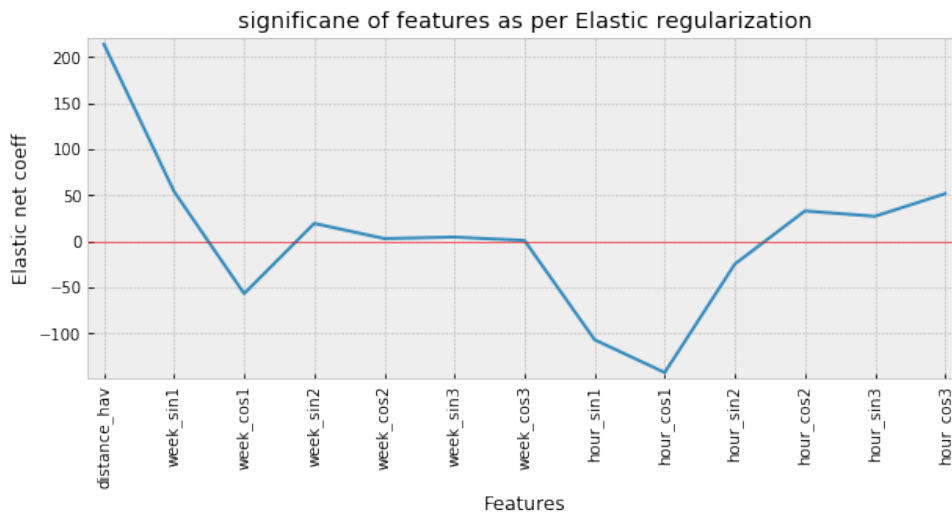


Figure 23 Elastic Net LR coefficients

We can see that the RMSLE has improved from the baseline of 0.77 to 0.49 by using a simple linear regression model with only a few variables – distance, hour, weekday. Also, the model was tuned and fit onto the dataset very quickly around 10 seconds.

Note: The top rank RMSLE for the kaggle competition is 0.28 and the top 20 groups out of some 1257 total groups are within 0.33 RMSLE. I am not using the exact same data as the kaggle competition but the above RMSLE values are a good benchmark to improve the model on. And the best models have used external datasets like the [OSRM](#) to calculate the shortest path between two points. We will try to avoid that in this project and see if the score improves without that (it does!!).

6.4 Random Forest

As discussed in the Elastic net regression section, Random Forest (RF) based on decision trees can handle non-linear and unscaled data.

All the variables defined above in the `columns_to_use` list were used to fit the RF model. The hyperparameters used to tune the model were:

- *n_estimators*: number of trees in the forest
- *max_features*: max number of features considered for splitting a node
- *max_depth*: max number of levels in each decision tree
- *min_samples_split*: min number of data points placed in a node before the node is split

The error metrics for the tuned RF using all the variables are as below:

```
Error metrics for model Tuned RF
RMSE or Root mean squared error: 334.61
RMSLE or Root mean squared log error: 0.39
Variance score: 0.73
Mean Absolute Error: 225.01
Mean Absolute Percentage Error: 34.82 %
```

The training R2 score was 0.73 as well so we can say that the model is not overfitting the data.

Again, we can see considerable decrease in the error terms from the baseline as well as the Elastic Net LR model.

But the gain in accuracy comes with a loss in time complexity. It took around 75.7min minutes to tune the model and around 10 minutes to fit the best model on to the data set which is much slower than the Elastic Net LR (which only used 3 variables, but even when using all the variables it took only a few minutes to tune and fit).

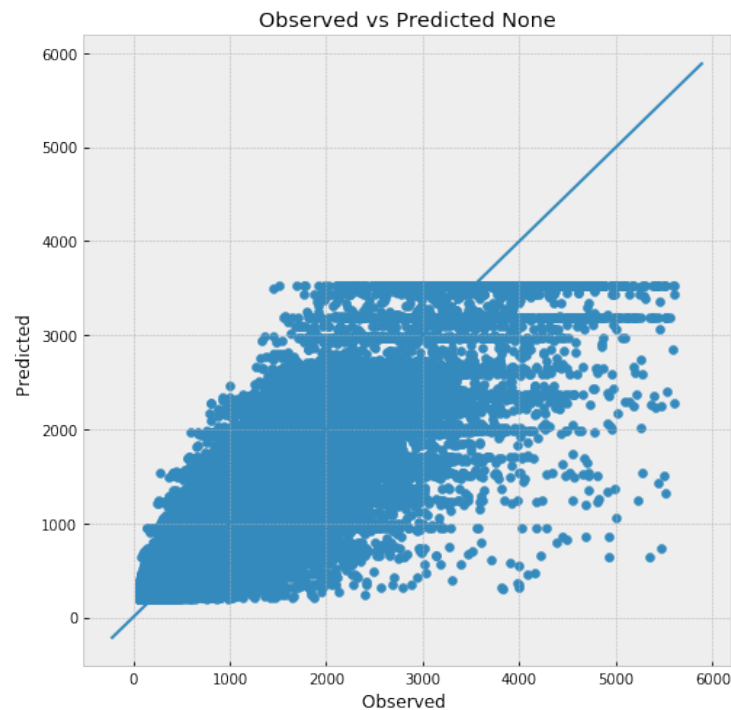


Figure 24 Observed vs Predicted trip duration in seconds Tuned Random Forest

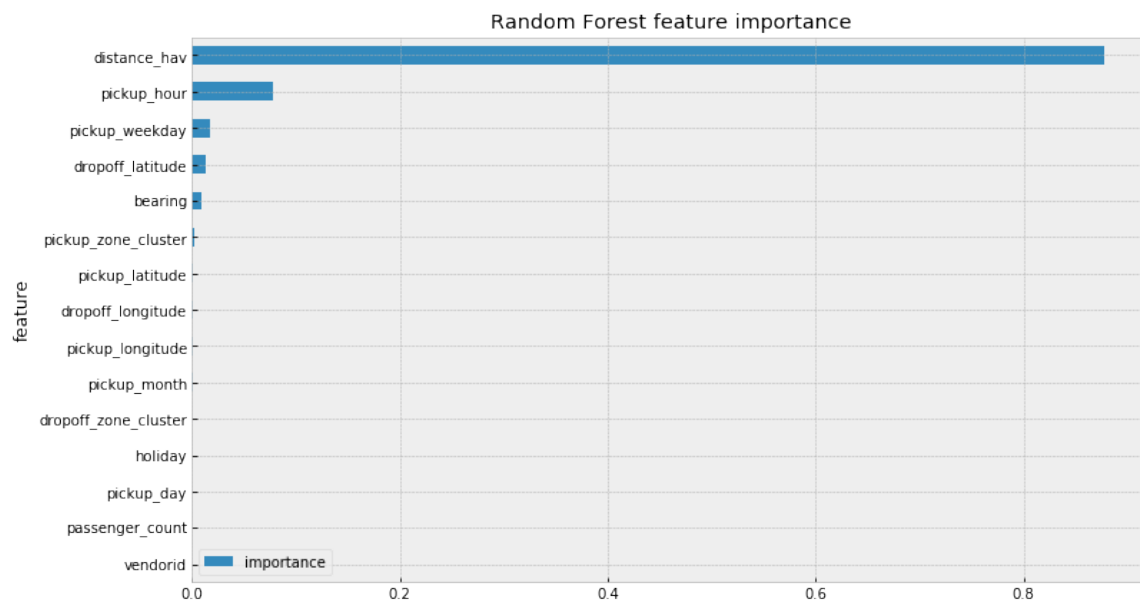


Figure 25 Feature importance using the Random Forest model

The flattening in the Observed vs Predicted seen above, is due to the fact that RF cannot predict the values beyond the scope of training data range because they cannot extrapolate well. A regression tree consists of a hierarchy of nodes, where each node specifies a test to be carried out on an attribute value and each leaf specifies a rule to calculate a predicted output, which are then averaged by random forest. For any data, that a Random Forest has not seen

before, at best, it can predict an average of training values that it has seen before. If the Validation set consists of data points that are greater or less than the training data points, a Random Forest will provide us with Average results as it is not able to Extrapolate and understand the growing/decreasing trend in our data. Also, by limiting the max_depth up to which the tree can grow the terminal nodes (leaves) end up averaging a larger range of the trip duration values together and hence the extreme values are predicted as lower values.

This can be improved by increasing the max-depth but then there is a possibility of over fitting. Let's see if we can solve this problem by using XGBoost.

Note: The above assumption was verified by running the RF model on a larger range of max_depth while using the best parameters found from above. The model was run in a separate notebook ("Further tuning RF and XGBoost.ipynb") located in the same folder. As expected, as we allow the trees to grow in depth, the leaf values are able to capture a larger range of the y variable because now each leaf is averaging over a small range as compared to the leaves at smaller depths. But, again as expected, this results in overfitting as the difference between the train and test scores increases significantly. That's why in real world scenarios, usually the max_depth parameter used falls in the range of 3 to 8.

Checking feature importance by permutation

Let's try another method for checking the feature importance based on our ML model.

A decision tree has to evaluate at every node how to split the values in a single feature so that most similar values from the dependent variable end up in the same leaf after the split. This condition is based on impurity, which, in case of regression trees, is variance. So, scikit learn calculates the feature importance by computing how much each feature contributes to decreasing the impurity over trees. In case of Random forests, this means averaging the impurity decrease over all the trees.

This results in a fast calculation, but it can also be biased, and it actually has a tendency to inflate the importance of continuous or high-cardinality categorical features. For example, our model can incorrectly assign a high importance to a randomly generated continuous variable.

To counter that we can try another feature importance evaluation method known as Permutation feature importance. The basic methodology of this approach is to randomly shuffle each predictor variable and check its effect on the overall performance on the training set.

Note: Ideally you should train the model on the re-shuffled training data and record the scores on the OOB (out of bag) samples, thus treating the OOB samples as a validation set. But here, for simplicity sake I am testing the model on the training set itself.

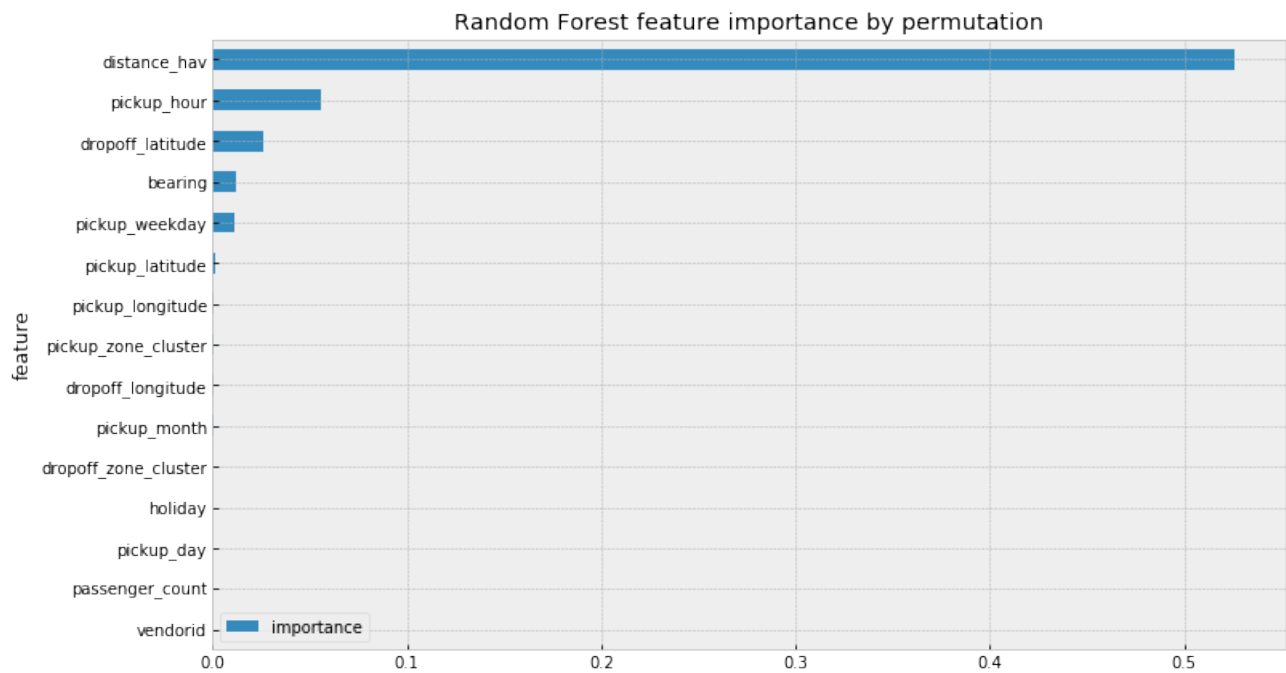


Figure 26 Random forest feature importance using feature permutation (improvement in RMSLE per feature)

The feature importance graphs based on the RF model show that the most important variable is the estimated distance between the pickup and dropoff locations, which kind of makes sense since the trip duration largely depends on the distance the taxi is eventually is going to travel. Other important variables are the pickup hour of the day, day of the week, the latitude and longitude of the pickup and dropoff locations and the bearing.

6.5 XGBoost

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is an optimized distributed gradient boosting library.

XGBoost is well known to provide better solutions than other machine learning algorithms. Let's see if we get improved performance compared to the RF model and if we can get rid of the predictions getting flattened out. We should expect to see an improvement because XGBoost makes splits up to the `max_depth` specified and then starts pruning the tree backwards and remove splits beyond which there is no positive gain.

All the variables defined above in the `columns_to_use` list were used to fit the XGBoost model. The hyperparameters used to tune the model were:

`learning_rate`: Makes the model more robust by shrinking the weights on each step. Typical final values to be used: 0.01-0.2

`n_estimators`: number of trees in the forest

`max_depth`: max number of levels in each decision tree

`colsample_bytree`: Denotes the fraction of columns to be randomly sampled for each tree. Typical values: 0.5-1.

`min_child_weight`: Defines the minimum sum of weights of all observations required in a child.

`gamma`: A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split. Makes the algorithm conservative. `subsample`: Denotes the fraction of observations to be randomly samples for each tree. Lower values make the algorithm more conservative. Typical value 0.5-1.

`reg_alpha`: L1 regularization term on weight (analogous to Lasso regression)

It took around ~30 minutes for the model to tune and fit on the training data, big improvement from the RF model. The error metrics for the tuned XGBoost using all the variables are as below:

```
Error metrics for model Tuned XGBoost
RMSE or Root mean squared error: 272.96
RMSLE or Root mean squared log error: 0.32
Variance score: 0.82
Mean Absolute Error: 178.24
Mean Absolute Percentage Error: 26.38 %
```

The train score was 0.83, so our model isn't overfit.

We can see considerable decrease in the error terms from the baseline as well as the Elastic Net LR and RF models. Also, the predicted values are not limited to any trip duration range and the higher and lower range values are also predicted well as can be seen from below figures.

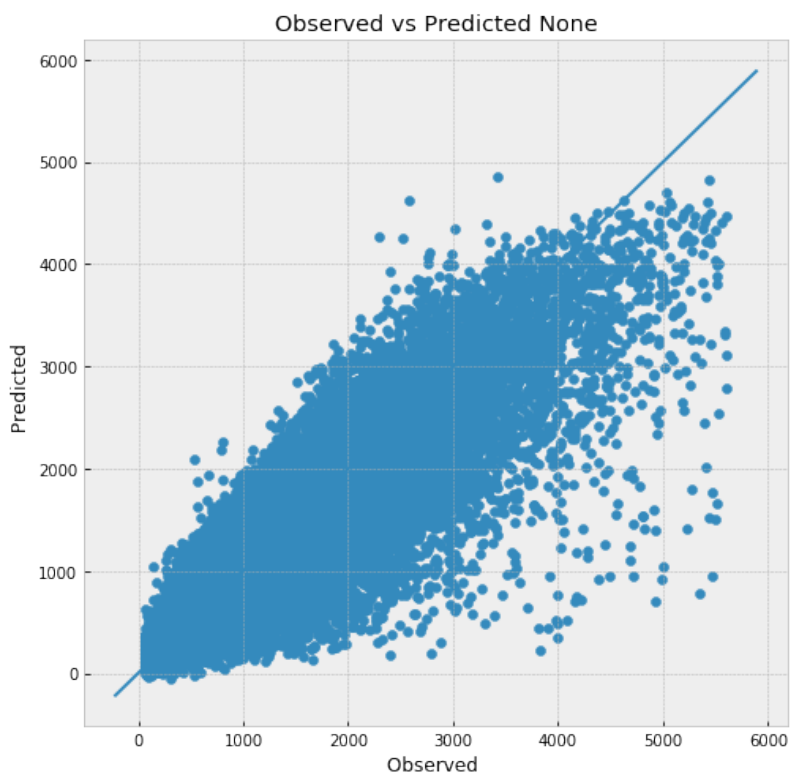


Figure 27 Observed vs predicted trip duration seconds for tuned XGBoost

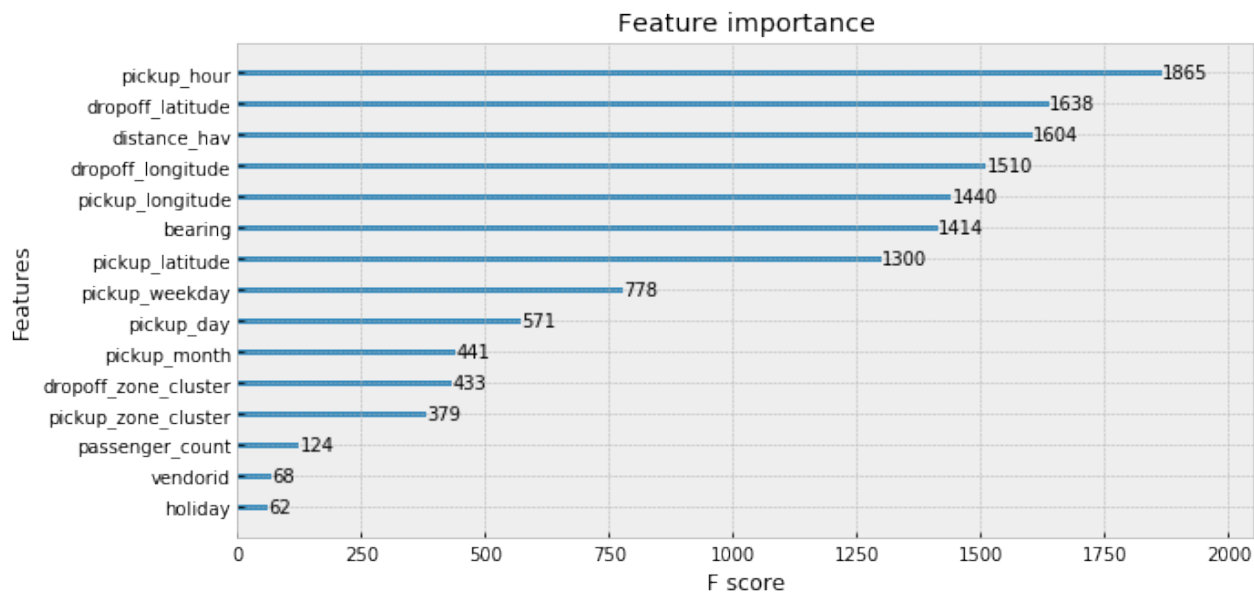


Figure 28 Feature importance using the XGBoost model

The feature importance graph assigns high importance to the hour, pickup and dropoff lat lons, distance, bearing, weekday, day, month pickup and dropoff zone clusters. Let's check the permutation feature importance.

Checking feature importance by permutation

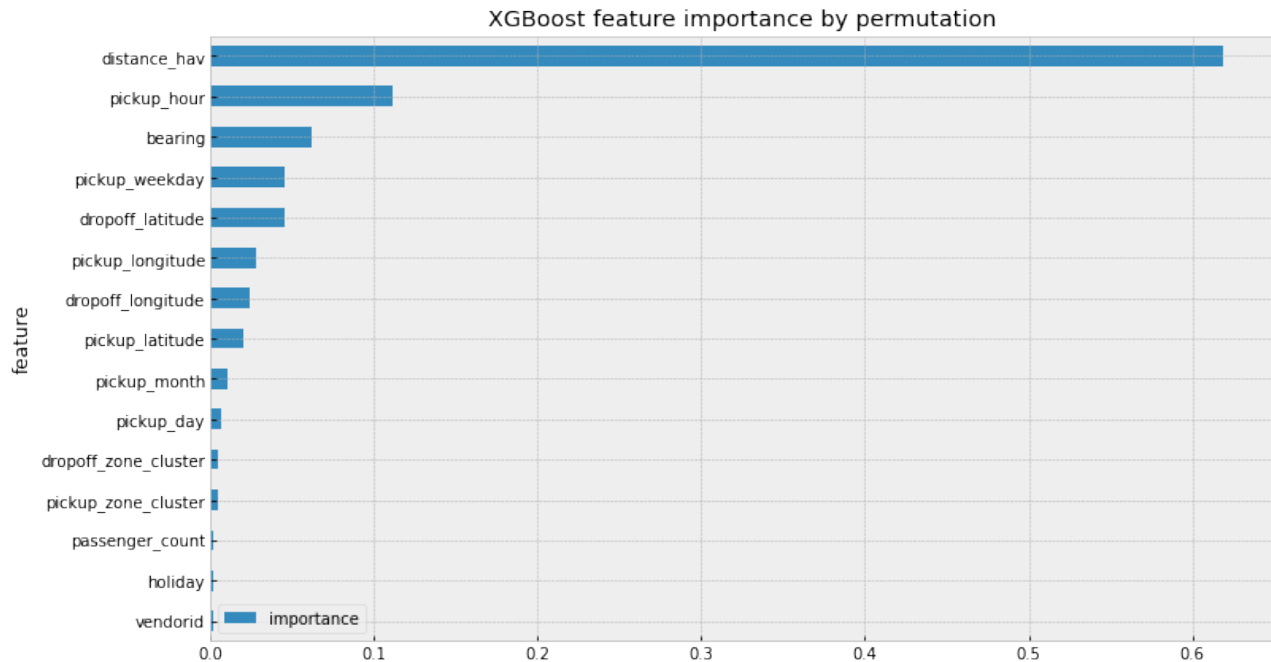


Figure 29 XGBoost feature importance using feature permutation (improvement in RMSLE per feature)

We can see that the feature permutation evaluation gives us more robust results. It puts the distance as the most important variable and also the vendor_id variable which more or less can be considered as a random variable since it has no effect on the trip duration was rated higher than holiday in the XGBoost's default feature importance graph. We can see in the feature importance permutation graph how the vendor_id is correctly rated as the least important feature.

We can see that the distance, hour, trip direction (bearing), weekday, pickup and dropoff lat longs are the most important variables.

Using the trained XGBoost model on the entire dataset (~4M taxi rides)

```
Error metrics for model Tuned XGBoost on entire dataset
RMSE or Root mean squared error: 272.42
RMSLE or Root mean squared log error: 0.33
Variance score: 0.82
Mean Absolute Error: 178.12
Mean Absolute Percentage Error: 26.34 %
```

We see that the model generalizes well over the entire dataset. Having been trained only on 500k taxi rides it predicts the trip duration of the entire 4M dataset very well and comparatively similar to our previous test results.

7. Comparing the Models

Let's compare the error metrics from all the models.

| | train_test | RMSE | RMSLE | R2 | MAE | MAPE |
|--|------------|--------|-------|-------|--------|-------|
| model | | | | | | |
| Tuned XGBoost | test | 272.96 | 0.32 | 0.82 | 178.24 | 26.38 |
| Tuned XGBoost on entire dataset | test | 272.42 | 0.33 | 0.82 | 178.12 | 26.34 |
| Tuned RF | test | 334.61 | 0.39 | 0.73 | 225.01 | 34.82 |
| Tuned Elastic Net (w/ only dist, hr, wday) | test | 383.49 | 0.49 | 0.64 | 266.99 | 47.76 |
| Baseline average duration | test | 639.34 | 0.77 | -0.00 | 460.26 | 94.37 |

Figure 30 Comparing error metrics across different models sorted with best model at the top

- Only a small sample of 500k rows gave us the above performance. The best model (XGBoost) is performing comparably well on the entire dataset of ~4M rows as well, so we can justify using a smaller dataset for training purposes, especially because it was much faster.
- Initially, 100k and 200k rows were used to fit the model and the model performance (all the error metrics above) improved significantly from using 100k to 200k rows and from using 200k to 500k rows. But the improvement from 100k to 200k was more significant than that from 200k to 500k.

What the errors mean?

```
trip_bins_minutes
(0-5]          99.659802
(10-20]        164.199907
(20-40]        297.363369
(40-60]        553.548500
(5-10]         116.300700
(60-100.0]     989.549678
Name: residuals, dtype: float64
```

Figure 31 Mean absolute errors (trip duration seconds) averaged over the trip duration bins (minutes)

- The median trip duration for the original dataset (4M rows) and the sampled dataset used for modeling is ~664 seconds. We see from the error metrics that the percentage error in predicting the trip duration is 26.38%. That is, our model predicts the trip duration as 178.24 seconds more or less than the actual on an average. This is just 3 minutes and given a busy city like NYC that's not bad at all. And we have used only the variables that were readily available to us.

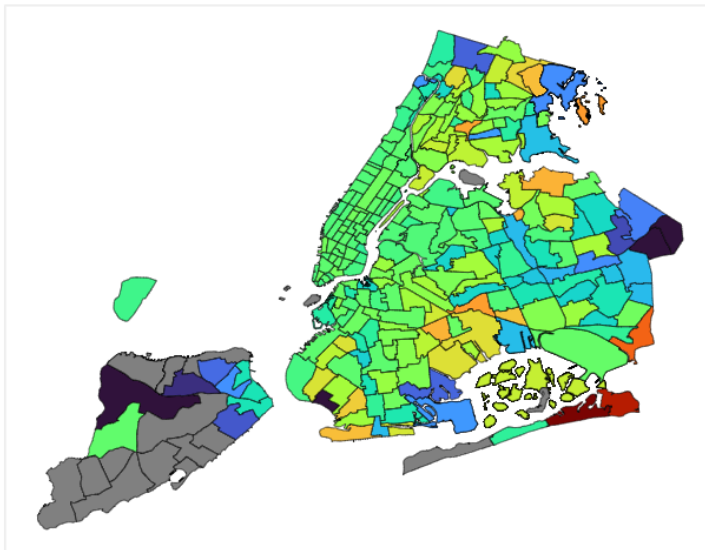
- If we further analyze the errors based on the trip duration bins then we observe that the shorter duration trips (0-5, 5-10, 10-20 and 20-40) are predicted very well within a margin of ± 1.65 to ± 5 minutes max whereas the longer duration trips such as 40-60 and 60-100 are predicted with a greater error margin of ± 8 to ± 16 minutes. Given that the 97% of the data lies below the 40 minutes trip duration, our model is performing very well on the majority of the dataset. We can check the longer duration trips further to find ways to improve their performance if desired as well.

And, the most important variables other than the distance between the pickup and dropoff locations, are:

- hour of the day
- weekday
- pickup and dropoff lat lons
- bearing (trip direction)
- even the day of the month and pickup and dropoff zone clusters seem to have some importance in the predictions.

Checking which areas had low and high errors

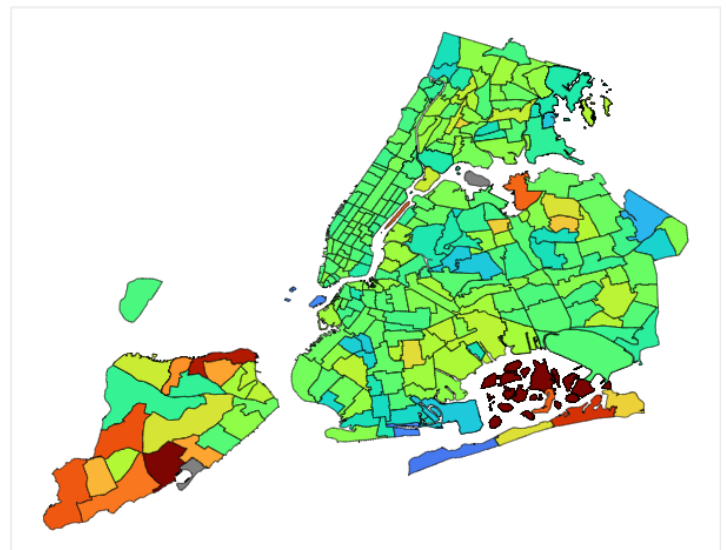
NYC Taxi pickups residuals map



residuals mts



NYC Taxi dropoffs residuals map



residuals mts



Figure 32 Residual errors (observed - predicted) of trip duration in minutes based on the pickup and dropoff zones

Distribution of residuals categorized by NYC boroughs¶

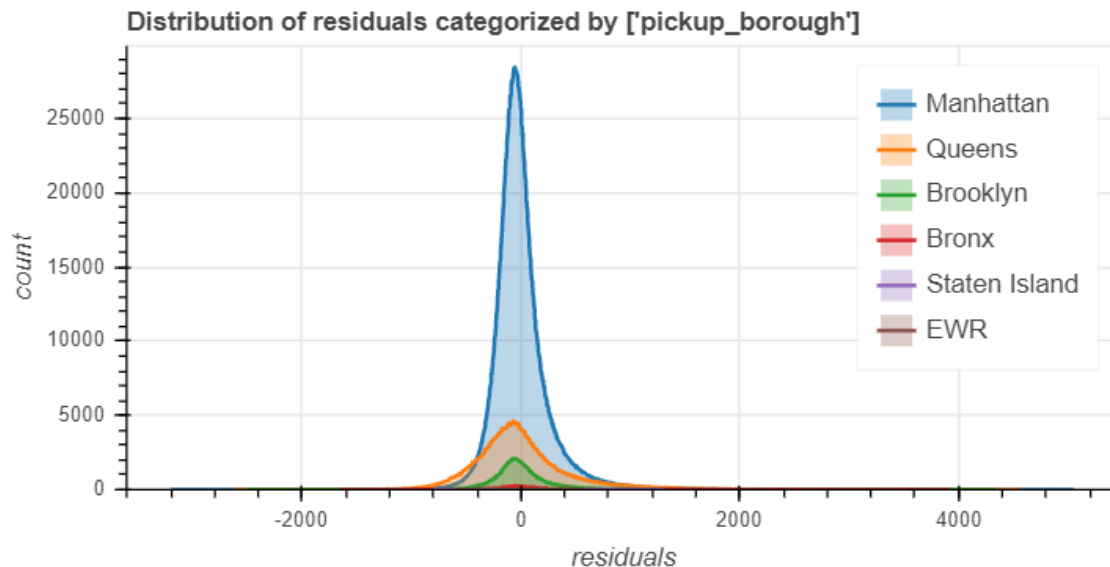


Figure 33 Distribution of residuals (observed - predicted) trip duration in seconds

The negative values of the residuals are over predictions and positive values are the under predicted values.

It seems that the errors are low for short to mid-range duration trips originating and ending in Manhattan, which kind of makes sense because majority of the dataset included Manhattan trips. The geo plots show the residuals in minutes. Technically the trip duration of taxi rides from and to all the important areas including the 5 boroughs (except Staten island) and the airports were predicted within -2 to +2 minutes error by the model. This is also because other areas (whose errors are high, shown red and blue), had only a few number of trips (~2 to 3 % of the entire dataset) and hence the model must have treated them as outliers. If we can get more trips data in those locations, then I guess we can improve our model's performance for those areas too.

The median error in trip duration minutes is -0.595.

8.Future Work

- Add zone to zone average speed as a feature. The average zone to zone speed can be added as a variable along with the distance variable.
- Add weather variables to see the effect of extreme weather events and temperature on the trip duration. The extreme events like heavy snow in Jan of 2016 were observed to cause the taxi trips to decline significantly but otherwise no significant effect of the weather in general on the taxi trip duration was seen. Temperature can have some

effect, for example high temperatures in summer or very low temperatures in winter can cause people to use more taxis, thus increasing the traffic overall.

- Implement deep learning models to fit the data. Since we have millions of taxi trips in this data we can use multi-layer neural network models to see if they give better performance compared to the other models tried in this notebook.
- Build a simple web application such that a user can input the pickup and dropoff latitudes and longitudes and the web app will throw out the estimated trip duration (using our best model).

(From EDA notebook) **Potential additions to the EDA and general visualizations:**

- Subway stations and other important information about potential busy areas can be added into the data and traffic trend around these areas of interest can be verified.
- A network graph to check how the zones are connected to each other based on the pickups and dropoffs.
 - Betweenness centrality can be added for each node in the graph.
 - Shortest path for any ride can be found and the nodes/edges through which the taxi has to pass to reach the destination can be determined. Then an average speed score can be assigned to that path based on the speed score of each edge the taxi is traversing through. This can be used as a new feature.

End of document
