DATA SCIENCE FOR BUSINESS ANALYTICS

SPRING 2020 – Course Project

“Predicting New York City Yellow Taxi Demand”

Team 13: Avantika Singh, Daffney Deepa Viswanath, Hardik Rokad

Table of Contents

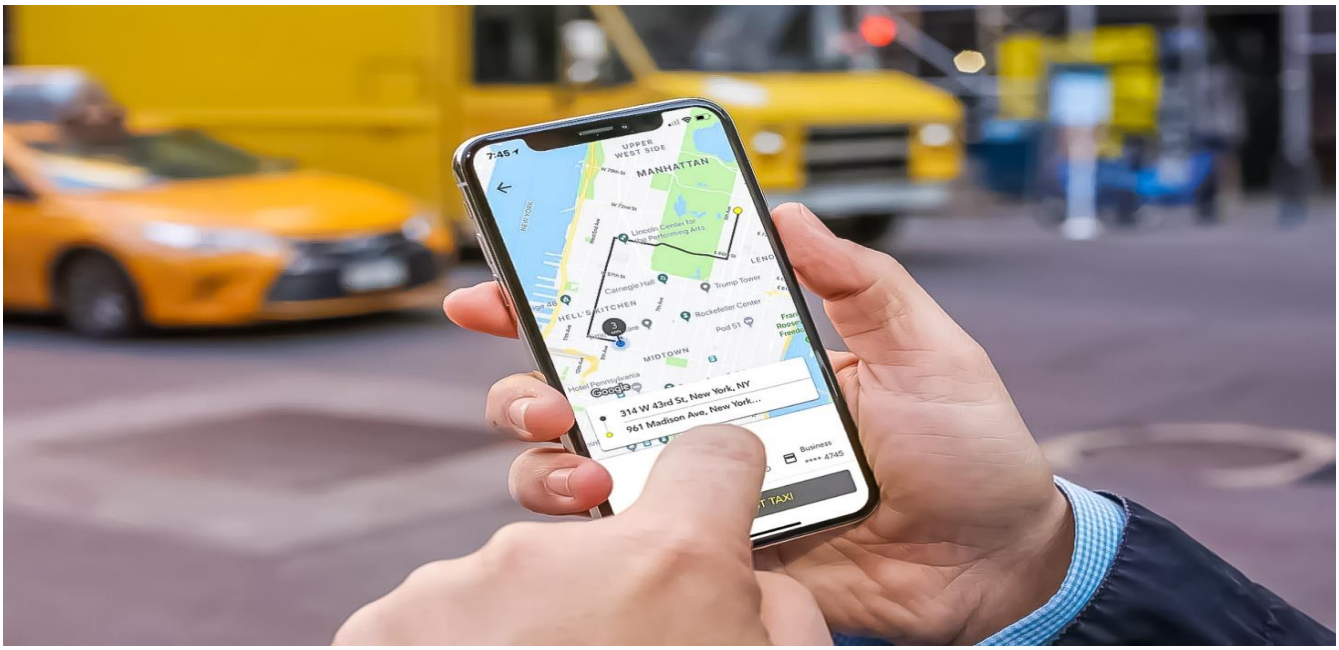
I. BUSINESS UNDERSTANDING

Medallion (yellow) cabs are concentrated in the borough of Manhattan but can be hailed anywhere throughout the five boroughs of New York City with a raised hand or from a taxi stand.

In the New York city, people use taxis at a much higher frequency than most places. Instead of booking customers by phone ahead of time, there is still a majority of New York taxi drivers that pick-up passengers on street. Hailing a cab is as simple as stepping off the curb and holding out your arm out.

Taxicabs drive through the streets of Manhattan near 5th Avenue in New York City in 1970's. This picture is from 1972. The vehicles' signature yellow livery didn't become law until 1967, when the city ordered that all licensed *"medallion taxis"* be painted the same color, in order to cut down on unofficial drivers and make the cabs more recognizable.

In 2011, Uber announced it was launching its ride sharing service in New York City, ushering in a new era of disruption in the taxi service industry.



By allowing anyone to enroll as a taxi driver with their own private vehicle and making it as easy as a click of a cell phone app to book a ride, Uber began establishing a new network of cabs outside of the medallion system. Drivers were plucked from the existing livery cab business, and vehicles didn't have to be of any particular color. Uber has opened the market for other competitors like Lyft, Via etc.

As per recent studies, Taxi patronage has considerably declined since 2011 due to competition from these rideshare services. With technology facilitated by universal smartphone penetration, decimating the existing yellow taxi industry. What if there existed a way to predict taxi ridership that gives valuable insights to taxi dispatchers – as in how to position cabs where they are most needed, how many taxis to dispatch, and how ridership varies over time. Such prediction will help dispatchers immensely in making important decisions that could revive their profit margin.

Our project focuses on predicting the number of taxi pickups given a one-hour time window and a location within New York City.

How will our data mining solution address the business problem?

For any given location in New York City, our goal is to predict the number of pickups in that given location at a particular time interval. Some location requires more taxis during a particular timeframe than other locations due to their proximity to offices, schools, hospitals etc. We have picked the data(explain). We intend to understand from the past data to build a model which would help in determining the features would be helpful in predicting the demand. With the competition that the Yellow Taxi drivers are facing due to the advent of apps like Uber and Lyft, we propose a solution that will help them understand the taxi demand in a particular area at any particular time. Exploiting an understanding of the taxi supply and demand could

increase the efficiency of the city's taxi system. As a future enhancement, we plan to build add a learning component to this system to update the model periodically or in real time and transfer the result to the taxi drivers who are near a place where there is more demand via their Mobile, thru text message or an App and they can subsequently move to the locations where the predicted pickups are higher and so they can profit more than they normally would.

II. DATA UNDERSTANDING

For solving any business problem, it is essential to understand the strengths and limitations of the data from which the solution will be built. We've decided to use the TLC Trip Record Data for our data mining process. The data used for building our solution was collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP). Each row corresponds to one trip. For the purpose of our modelling we'll be using the data from January 2015.Data Source: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page quick glance at the features in our data set :

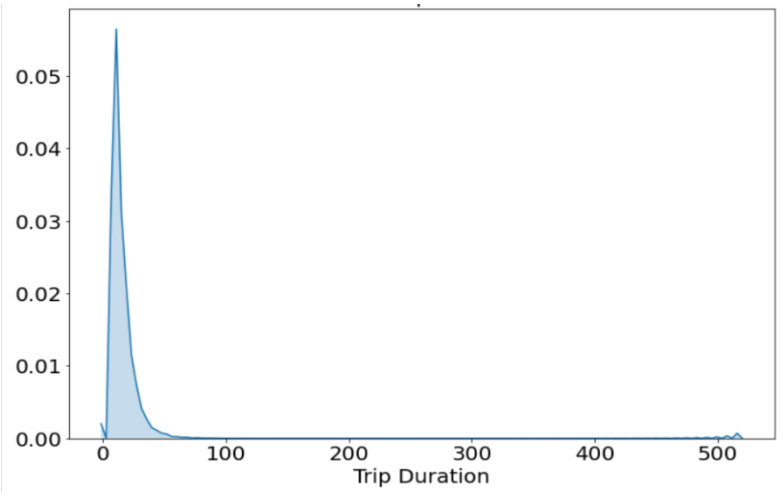
Features in the data

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1. Creative Mobile Technologies 2. VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
Pickup_longitude	Longitude where the meter was engaged.
Pickup_latitude	Latitude where the meter was engaged.
RateCodeID	The final rate code in effect at the end of the trip. 1. Standard rate 2. JFK 3. Newark 4. Nassau or Westchester 5. Negotiated fare 6. Group ride
Store_and_fwd_flag	This 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
Dropoff_longitude	Longitude where the meter was disengaged.
Dropoff_ latitude	Latitude where the meter was disengaged.
Payment_type	A numeric code signifying how the passenger paid for the trip. 1. Credit card 2. Cash 3. No charge 4. Dispute 5. Unknown 6. Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes. the $0.50 and $1 rush hour and overnight charges.
MTA_tax	0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	0.30 improvement surcharge assessed trips at the flag drop. the improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips.Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.

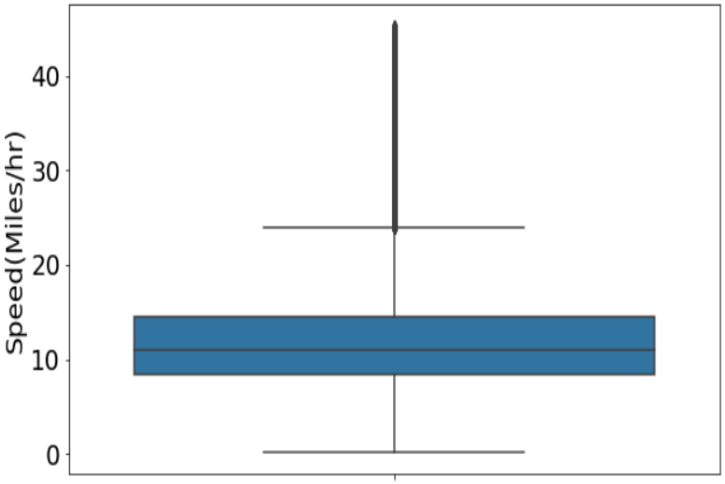
III. DATA CLEANING:

We start our data preprocessing by running a couple of essential checks on the dataset. We performed univariate analysis of the data wherein all erroneous data was removed.

- Dimensionality reduction - Checking for extraneous attributes that do not influence our target variable and removing them. (such as RateCode_ID, Store_and_fwd_flag etc).
- Checking for any null values in the dataset.
- Validating that the trip distance is not less than 0.1 miles.
- Performing a check for the uniqueness of the Vendor_ID.
- Removing any erroneous records where the drop off time is earlier than pick up time.
- Removing any erroneous records where the passenger count is less than 0 as a trip cannot happen without a passenger.
- Validating the payment types.
- As a part of our analysis, we calculate the trip distance and append it as a feature in our existing data frame.
- According to the NYC Taxi and Limousine Commission regulations the maximum allowed trip duration in a 24 hours interval is 12 hours and so we performed a check for any such outliers and removed them.

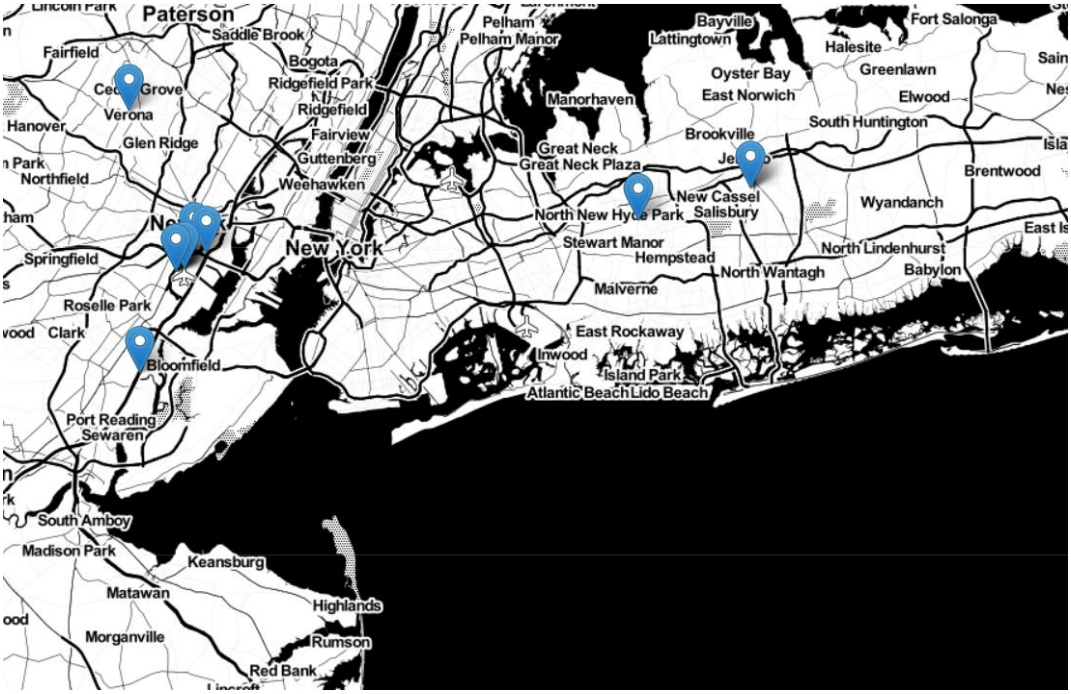


- As another essential feature, we calculate the speed of the Taxi and add it to our list of features.
- As we calculate the average speed of Taxi's in New York city to be around 12.8 miles/hour , so on an estimate a cab can drive 2 miles per 10 min on average.
- We perform additional checks on the existing feature values and remove any outliers.

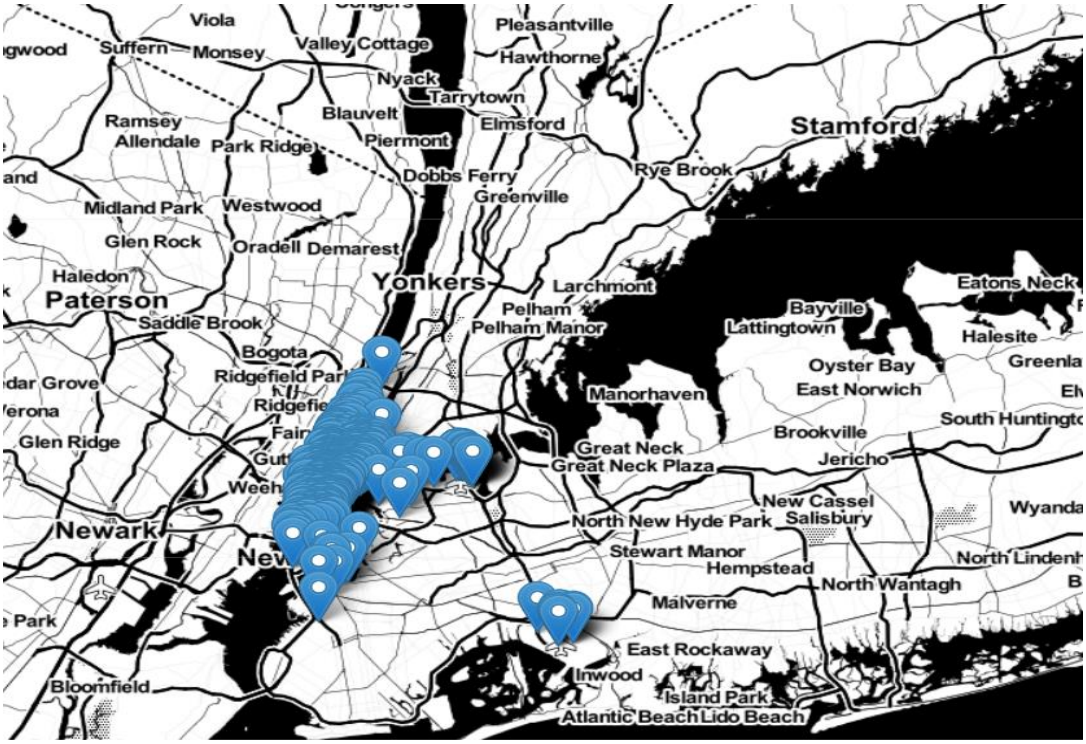


Moreover, as a part of the data cleaning phase we check if the pick-up latitude and longitude fall within NYC and remove the ones that don't.

Pick-Up Map Before Cleaning:



Pick up Map After Cleaning

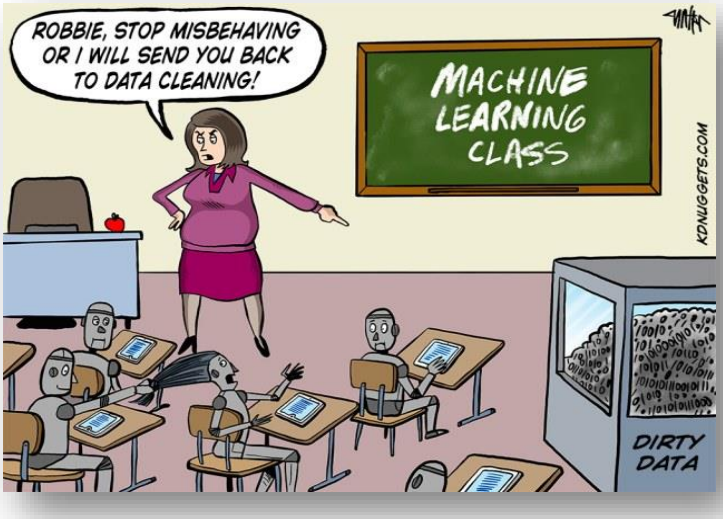


We can observe that the pick-ups are concentrated in and around NYC.

IV. DATA PREPARATION

In the data preparation phase, we focus on cleaning the data and transforming it into a format that would help us train the model better.

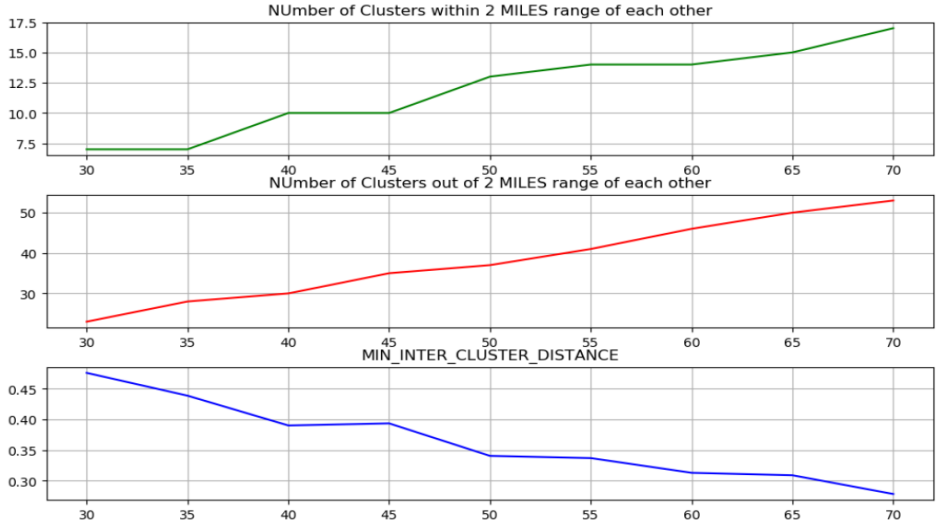
Good data preparation allows for efficient analysis and limits the errors and inaccuracies that occur due to erroneous data.



CLUSTERING AND SEGMENTATION

Since a part of our solution is to predict the pickups in a region within some time interval, we need to derive regions and time intervals from data.

- We begin with dividing NYC into K clusters using the latitude and longitude of pickups.
- Using K-Means clustering, we cluster the pick-up points into different regions.
- Since K-Means creates clusters of roughly same sizes (clusters of same number of points), the regions with more number of pickups form smaller and dense clusters whereas regions with lesser number of pickups get connected into larger and loose/sparse clusters.
- From the data, we observe that a taxi can cover upto 2 miles in 10 minutes. Therefore, we want the inner cluster distance to be less than 2 miles but greater than 0.5 miles. The optimal K value must meet this constraint.

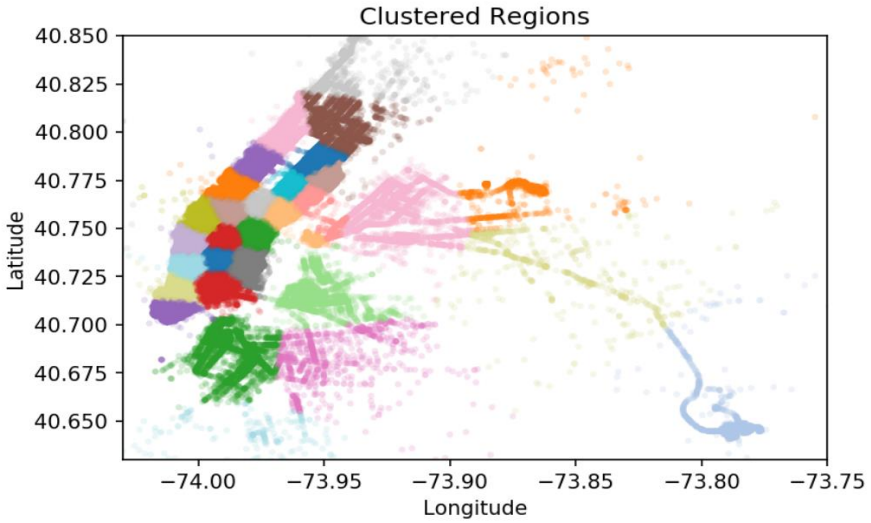


We tried a range of different cluster values from 30 to 70 clusters. After trying a different range of clusters, we observed that when the number of clusters is 30,

- The number of clusters within the 2-mile radius : 7.0
- The number of clusters outside the 2-mile radius : 23.0
- The minimum inter cluster distance : 0.475971

As we can see from the graphs that the above condition is met, we choose that as the optimal value for k as 30.

Visualizing the clusters on a map:



TIME BINNING

In order to predict pickups, we cluster NYC into regions and set a time interval within which the prediction can be performed. Let's say, we pick an interval of 10 minutes. Every region is split into 10-minute interval, which corresponds to one-time bin, i.e. each time bin has 10 minutes. However, in the data we have time in the format "YYYY-MM-DD HH:MM:SS which are converted to Unix time format so as to retrieve time in minute/hour format. We have made this project in New York (EDT). Thus, the function returns Unix time stamp from the reference of local time, divide the timestamp by 600 to convert it into 10minute bin and add 33 to convert it from GMT to EST.

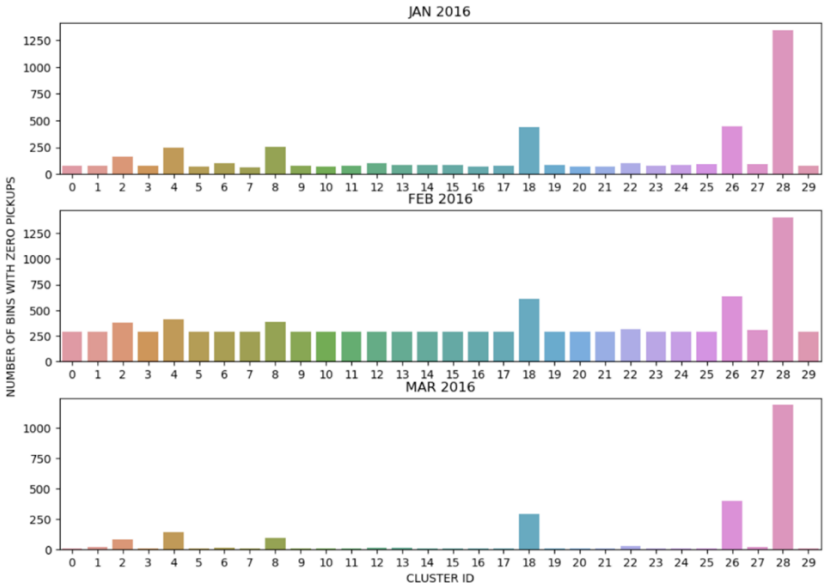
- Number of possible 10 min interval bins : 4464
- Total number of pickup_bins from clusters : 130567
- Number of possible pickup_bins from all clusters : 133920

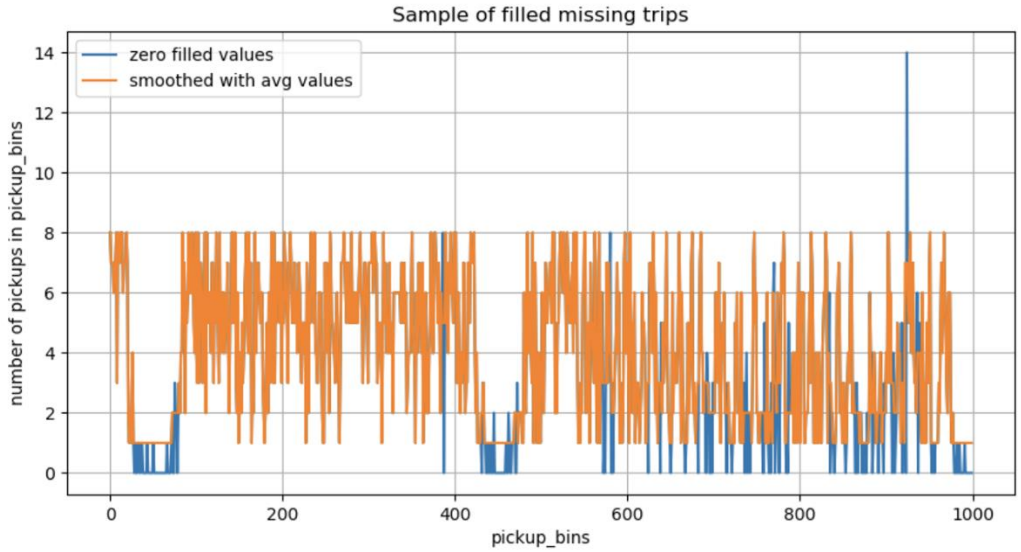
Now we have both cluster-ID and 10min time bins. With any region-ID and 10 min time bin, we can predict the number of pickups.

SMOOTHING THE DATA:

Now that the data is divided into 10-minute interval bins, each bin should contain at least one pickup. There are chances that a time bin may contain zero pick-ups leading to ratio feature error. In order to smooth this outlier:

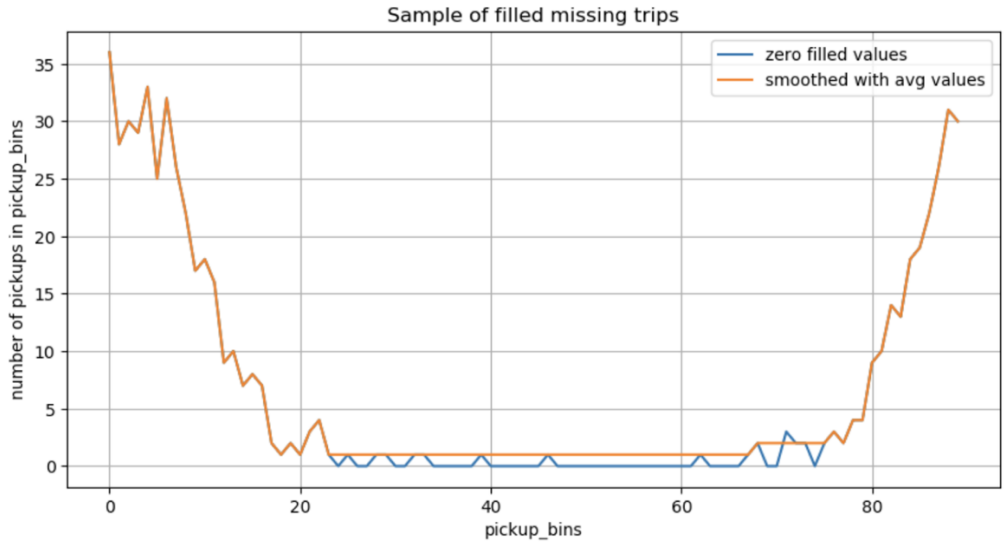
- We count the total number of pickups in each bin. There would be no value if there were no pickups in a time bin.
- Fill the missing pickup time bins with 0.
- Then, we add a few pickups from neighboring bins to the bin which contains zero pickups in order to make all the neighboring bin pickups equal.





Why smooth the data this way?

Consider FOUR 10 min intervals in which the number of pickups are 10, , , 20 respectively. By using zero filling, these values become 10, 0, 0, 20. When Smoothing, the values are distributed over the range with equal value. ie,. (30/4 ~= 7) ==> 7, 7, 7, 7. Thus, the number of pickups over the 40 min interval is same in both the cases.



If we observe, we are looking at the future values, i.e future number of pickups. These future values can cause a data leakage which is avoided by smoothing the data.

TIME SERIES AND FOURIER TRANSFORMS

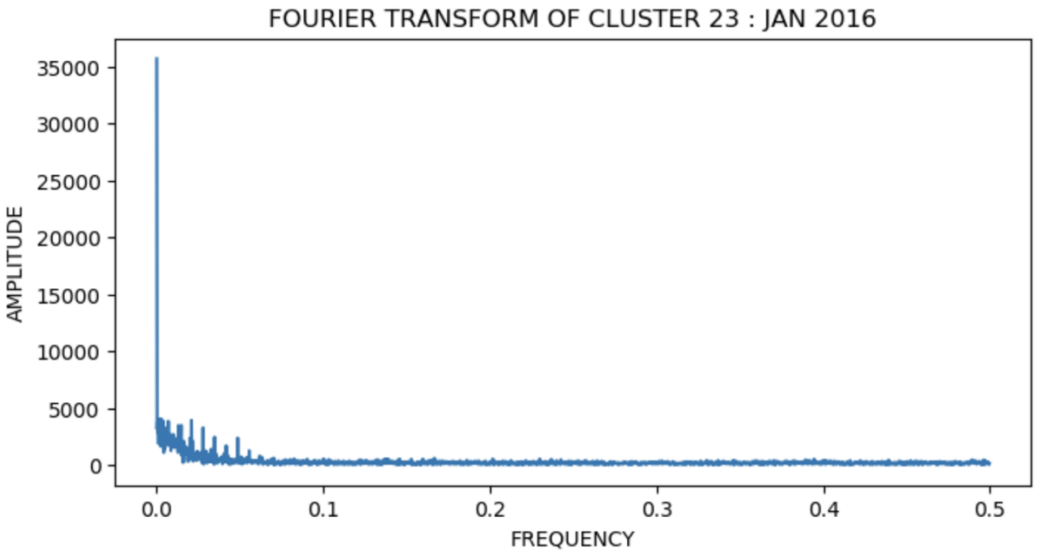
In theory, any waveform can be represented as the sum of infinite sine waves. Each sine wave has some amplitude and frequency. We can see that the number of pickups in a month in every cluster form a repeating pattern. We do not know the frequency of the repeating pattern. Our pattern cannot be represented by a single frequency as it's aperiodic. Instead it is composed of infinite sinewave with each sinewave having a frequency. Fourier transform lets

us represent our pattern from time domain (number of pickups per time) to frequency domain(can be viewed as number pickup bins with highest number of pickups).

For each cluster there exists a pattern and using the Fourier transform we can deduce the top frequencies and amplitudes of sine waves which compose our pattern from cluster and use them as features. The frequencies and amplitudes of a cluster are indicative of demand in that cluster. So they can be fed into the model for prediction of number of pickups.

Plotting the Amplitude and frequency of the sinusoidal components, we see that

1. The pattern whose repetition is very high will have a high frequency component and vice versa.
2. There are high frequency in morning and evening time durations as the pick-ups are high during peak hours. The same applies to day and night but with less frequencies.



We see that there is very high amplitude at f=0. As the frequency of a sine wave approaches zero, the sine wave tends to be axis-parallel/linear, appearing as a DC component. Since, the wave which we have Fourier transformed is a repeating wave, the DC component looks ambiguous as it is capturing the information of the previous part of the wave. Hence, we will not consider its amplitude and frequency.

V. MODELING

BASELINE MODELS

The first giant step to solving our business problem begins with baseline model. We will walk you through the process of discovering the right baseline model which gave us the best result (least errors!).

We use these models with two variations

1. Using Ratios of the 2016 data to the 2015 data i.e. Ratio= P_2016/P_2015
2. Using Previous known values of the 2016 data itself to predict the future values

P_2016 and P_2015 are the pickup densities for their respective years.
This is our data frame with pickup densities of 2015 and 2016 and their ratios.

Simple Moving Averages

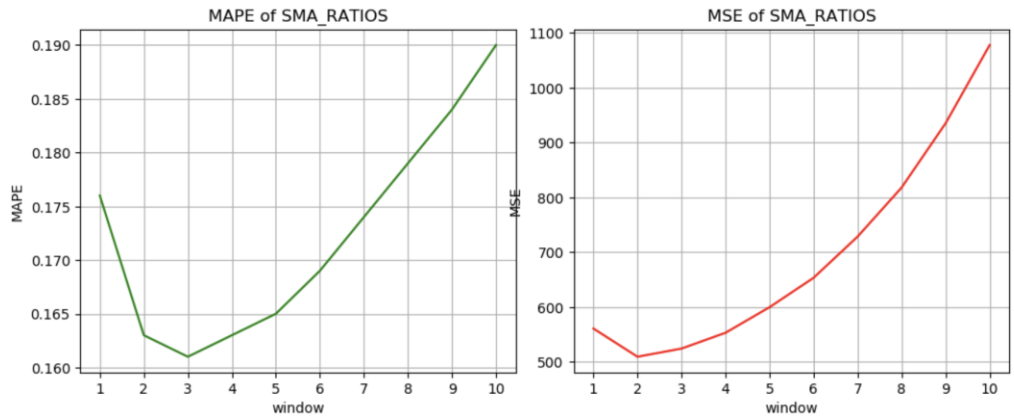
Our first choice was the Simple moving averages model which is used in the Moving Averages Model. It uses the previous n values in order to predict the next value.

SMA on Ratios:

Using the previous n ratios to predict the next one.

$$R(t)=R(t-1) +R(t-2) +….+R(t-n) / n,$$

where n is the hyperparameter tuned.
We found that n=3 (MAPE is low) window size is optimal for getting the best results.

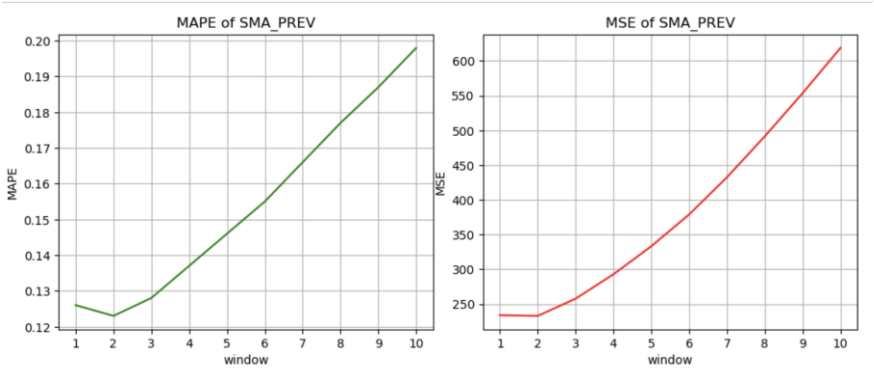


SMA on Predictions:

Use the 2016 values itself to predict the future values.

$$P(t)= P(t-1) +P(t-2) +…+P(t-n) / n$$

We found that n=2 is the optimal window size after tuning



	P_2015	P_2016	RATIOS	SMA_RATIOS_PRED	SMA_PREV_PRED
0	68	0	0.000000	0	0
1	68	106	1.558824	0	0
2	198	181	0.914141	154	53
3	281	263	0.935943	231	143
4	256	228	0.890625	290	222
5	252	241	0.956349	230	245
6	221	187	0.846154	205	234
7	201	147	0.731343	180	214
8	159	150	0.943396	134	167
9	134	158	1.179104	112	148

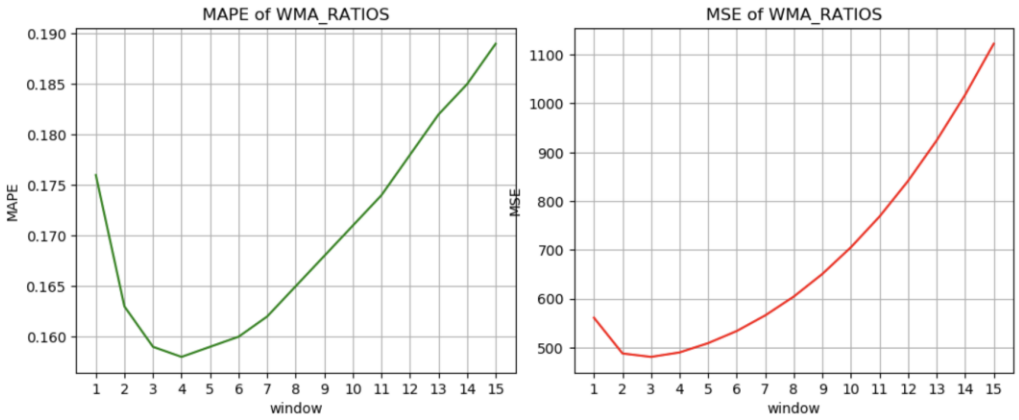
Weighted moving averages

SMA method gives equal weight to all the previous ratios and previous pickup densities, but we know intuitively that the future is more likely to be similar to the latest values and less similar to the older values. Weighted Averages converts this analogy into a mathematical relationship giving the highest weight while computing the averages to the latest previous value and decreasing weights to the subsequent older ones.

- **WMA on Ratios**

$$Rt = N \cdot Rt-1 + (N-1) \cdot Rt-2 + (N-2) \cdot Rt-3 + \cdots + 1 \cdot Rt-n \;/\; N \cdot (N+1)/2$$

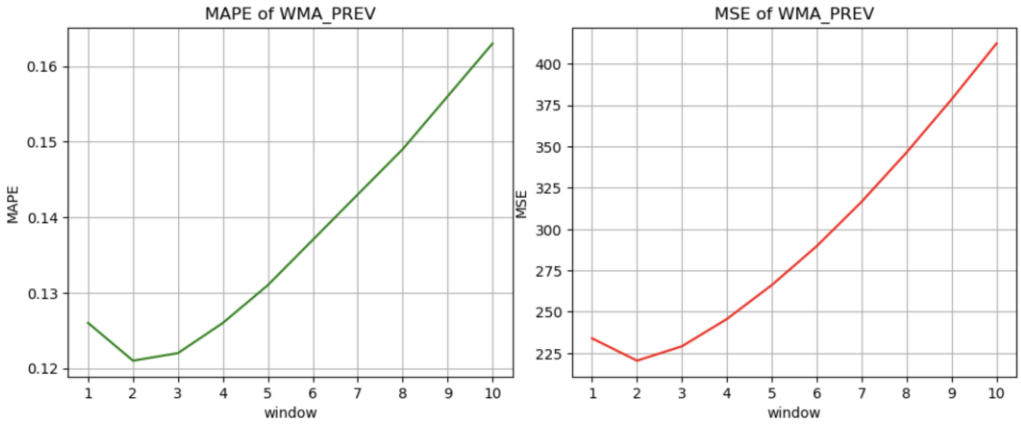
We tune to get N=4 window size to be optimal.



- **WMA on previous pickup densities.**

$$Pt = N \cdot Pt-1 + (N-1) \cdot Pt-2 + (N-2) \cdot Pt-3 \ldots . 1 * Pt-n \; / \; N \cdot (N+1)/2$$

After tuning the window size, we find n=2 to be optimal.



Contents of our data frame after adding the weighted moving average values on ratio and previous pickup densities.

	P_2015	P_2016	RATIOS	SMA_RATIOS_PRED	SMA_PREV_PRED	WMA_RATIOS_PRED	WMA_PREV_PRED
0	68	0	0.000000	0	0	0	0
1	68	106	1.558824	0	0	0	0
2	198	181	0.914141	154	53	205	70
3	281	263	0.935943	231	143	274	156
4	256	228	0.890625	290	222	245	235
5	252	241	0.956349	230	245	245	239
6	221	187	0.846154	205	234	205	236
7	201	147	0.731343	180	214	180	205
8	159	150	0.943396	134	167	131	160
9	134	158	1.179104	112	148	115	149

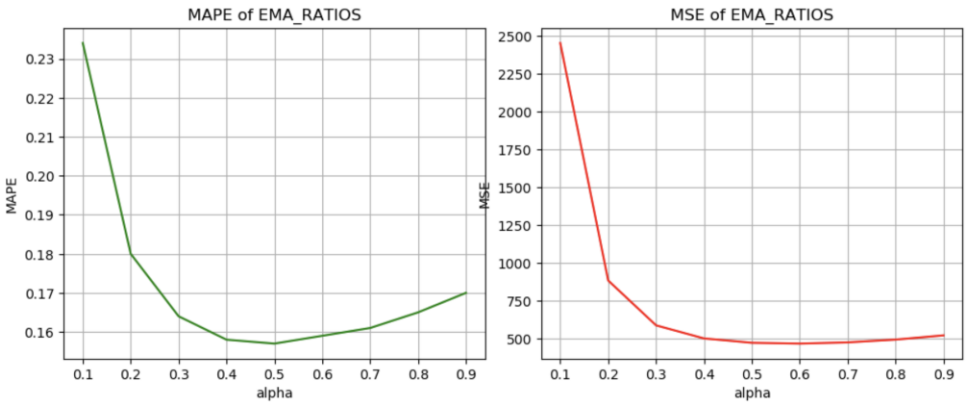
Exponential Weighted Moving Average

- Through weighted averaged we have satisfied the analogy of giving higher weights to the latest value and decreasing weights to the subsequent ones.
- But we still do not know which is the correct weighting scheme as there are infinitely many possibilities in which we can assign weights in a non-increasing order and tune the hyperparameter window-size.
- To simplify this process, we use Exponential Moving Averages which is a more logical way towards assigning weights and at the same time also using an optimal window-size.
  a. In exponential moving averages we use a single hyperparameter alpha α which is a value between 0 & 1 and based on the value of the hyperparameter alpha the weights and the window sizes are configured.
  b. For example, if α=0.9 then the number of days on which the value of the current iteration is based is ~ $1/(1-\alpha)$ =10 i.e. we consider values 10 days prior before we predict the value for the current iteration. Also, the weights are assigned using $2/(N+1)$ =0.18, where N = number of prior values being considered, hence from this it is implied that the first or latest value is assigned a weight of 0.18 which keeps exponentially decreasing for the subsequent values.

- **EMA on Ratios**

$$R_t' = \alpha . R_{t-1} + (1- \alpha) . R'_{t-1}$$

After tuning the alpha parameter, we get alpha = 0.5 to be optimal.

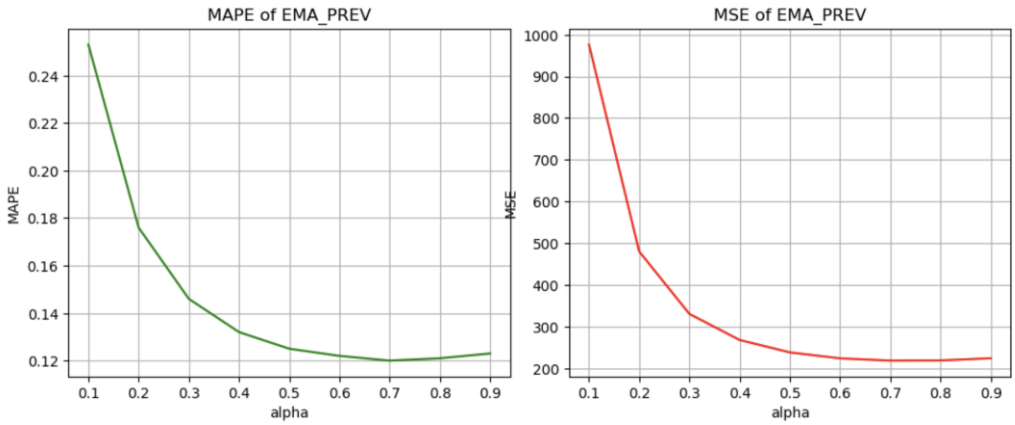


- **EMA on Previous Pickup Densities**

$$P_t' = \alpha . P_{t-1}' + (1- \alpha) . P'_{t-1}$$

After tuning the values of alpha, we use alpha = 0.7 to be optimal.

Below is the data frame after doing all the above baseline models



Contents of our data frame after adding the exponential moving average values on ratio and previous pickup densities.

	P_2015	P_2016	RATIOS	SMA_RATIOS_PRED	SMA_PREV_PRED	WMA_RATIOS_PRED	WMA_PREV_PRED	EMA_RATIOS_PRED	EMA_PREV_PRED
0	68	0	0.000000	0	0	0	0	0	0
1	68	106	1.558824	0	0	0	0	0	0
2	198	181	0.914141	154	53	205	70	154	74
3	281	263	0.935943	231	143	274	156	237	148
4	256	228	0.890625	290	222	245	235	228	228
5	252	241	0.956349	230	245	245	239	224	228
6	221	187	0.846154	205	234	205	236	204	237
7	201	147	0.731343	180	214	180	205	177	202
8	159	150	0.943396	134	167	131	160	128	163
9	134	158	1.179104	112	148	115	149	117	154

Comparison between different baseline models

After comparing all the baseline using MAPE (Mean Absolute Percentage Error) as our error metric to know how good the model is with predictions and MSE (Mean Squared Error) to know how well our forecasting model performs with outliers so that we make sure that there is not much of an error margin between our prediction and the actual value.

	MAPE	MSE
EMA_PREV	0.1205	218.6057
WMA_PREV	0.1206	220.4236
SMA_PREV	0.1231	232.9855
EMA_RATIOS	0.1574	473.4599
WMA_RATIOS	0.1580	489.5664
SMA_RATIOS	0.1613	524.1292

From the below results we see that Exponential Weighted Moving Average using previous pickup densities. (EMA_PREV) is the best forecasting model for our predictions.

FEATURE ENGINEERING

In order to build a good predication model, we need to ensure all the necessary features are present. After doing all the data cleaning and preparation and baseline modelling, we now have an arsenal of features which would help build a good predication model.
We saw from baseline modelling how Exponential weighted moving averages gave us the best forecasting among the rest. We will use this as a feature while building the regression model along with others we got from data preparation stage.

Below are the list of 14 features we use for modelling,

- Latitude and Longitude of the cluster - 2
- Top three amplitudes with respective frequencies from fourier transform - 6
- Five previous pickup densities from the cluster where the pickup density is to be predicted - 5
- The pickup density predicted by the EMA_PREV baseline model - 1

The feature vector we use as training data for our model.

	LAT	LON	AMP1	AMP2	AMP3	FREQ1	FREQ2	FREQ3	P5	P4	P3	P2	P1	EMA_PRED
2	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	0.0	0.0	0.0	0.0	106.0	74.0
3	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	0.0	0.0	0.0	106.0	181.0	148.0
4	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	0.0	0.0	106.0	181.0	263.0	228.0
5	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	0.0	106.0	181.0	263.0	228.0	228.0
6	40.733543	-73.991486	130399.585938	62330.636719	45587.714844	0.006944	0.013889	0.012993	106.0	181.0	263.0	228.0	241.0	237.0

REGRESSION MODELS:

Test and train split:

For test and train split by time, we take 3 months of 2016 pickup data and split it such that for every region we have 70% data in train and 30% in the test. As this is a time-series problem, we have to split our train and test data on the basis of time.

LINEAR REGRESSION:

To begin with, we standardize the test and train data by removing the mean and scaling to unit variance. Then, we apply the model. We have used SDGRegressor as it is well suited for regression problems with large number of training and testing data. Also that it is much faster as it has lesser data to manipulate at every iteration.i.e picks a random instance in the training set at every step and computes the gradients based only on that single instance.

```
linear_reg_gridsearch = GridSearchCV(SGDRegressor(),
                                     param_grid={'alpha' : np.logspace(-6, 3, 10)},
                                     scoring='neg_mean_absolute_error',
                                     cv=3)

linear_reg_gridsearch.fit(train_2016_features_std, train_labels)
```

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsilon=0.1,
       eta0=0.01, fit_intercept=True, l1_ratio=0.15,
       learning_rate='invscaling', loss='squared_loss', max_iter=None,
       n_iter=None, n_iter_no_change=5, penalty='l2', power_t=0.25,
       random_state=None, shuffle=True, tol=None, validation_fraction=0.1,
       verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-06, 1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+0
1,
       1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_absolute_error', verbose=0)
```

Tuning of Hyper Parameter is done using GridSearchCV() in order to find the best combination of parameter values,.i.e optimal parameter. We also do 3-fold cross validation to improve generalization performance.

Measuring Performance:

Finally, we calculated the Mean Absolute Percentage Error and the Mean Squared Error for training data of 2016. We have used the below function to calculate the Error Metrics.

```
def mean_abs_per_error(y_true, y_pred):
    return np.round((np.sum(np.abs(y_true - y_pred)) / np.sum(y_true)), 4)

def mean_squared_error(y_true, y_pred):
    return np.round(np.mean(np.square(y_true - y_pred)), 4)
```

```
print(f" TRAIN MAPE : {mean_abs_per_error(train_labels, linear_reg.predict(train_2016_features_st
d))}")
print(f" TRAIN MSE  : {mean_squared_error(train_labels, linear_reg.predict(train_2016_features_st
d))}")
print()
print(f" TEST MAPE : {lin_reg_mape}")
print(f" TEST MSE  : {lin_reg_mse}")
```

```
TRAIN MAPE : 0.1175
TRAIN MSE  : 238.5128

TEST MAPE : 0.1165
TEST MSE  : 218.275
```

RANDOM FOREST REGRESSION:

Our next choice of model was Random Forest Regression. Again, Random forest is great for maintaining accuracy for a large proportion of data and doesn't allow overfitting if there are too many trees.

```
rf_random_cv_clf = RandomizedSearchCV(RandomForestRegressor(n_jobs=-1),
                                      param_distributions=parameters,
                                      scoring='neg_mean_absolute_error',
                                      cv=5)
rf_random_cv_clf.fit(train_2016_features, train_labels)
```

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
          estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'n_estimators': [10, 20, 40, 80], 'max_depth': [2, 3, 4], 'min_sam
ples_leaf': [4, 8, 12], 'min_samples_split': [4, 8, 12]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

We have used RandomizedSearchCV() as it tries out a fixed number of parameter settings is sampled from the specified distributions, unlike gridsearchcv(), where all parameters are tried out. We also do 5-fold cross validation to improve generalization performance.

Measuring Performance:

As in the previous model, we calculated the Mean Absolute Percentage Error and the Mean Squared Error for training data of 2016.

```
TRAIN MAPE : 0.1266
TRAIN MSE  : 272.4468

TEST MAPE : 0.1261
TEST MSE  : 250.1323
```

XGBOOST REGRESSOR

Our next choice is XGBoost Regressor which is an implementation of gradient boosted decision trees designed for speed and performance. Training a hyper-parameter tuned Xg-Boost regressor on our train data.

```
xgb_random_cv = RandomizedSearchCV(xgb.XGBRegressor(),
                                   param_distributions=parameters,
                                   scoring='neg_mean_absolute_error',
                                   cv=3)
xgb_random_cv.fit(train_2016_features, train_labels)
```

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'n_estimators': [10, 20, 40, 80, 150], 'max_depth': [2, 3, 4], 'mi
n_child_weight': [2, 3, 4, 5, 6], 'learning_rate': [0.1, 0.3, 0.5, 0.7, 0.9], 'colsample_bytree':
[0.5, 0.75, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

Measuring Performance:

As in the previous model, we calculated the Mean Absolute Percentage Error and the Mean Squared Error for training data of 2016.

```
TRAIN MAPE : 0.1151
TRAIN MSE  : 225.9773

TEST MAPE : 0.1158
TEST MSE  : 213.0978
```

VI. MODEL EVALUATION

We have built a total of nine regression models using the pickup_densities. Some models used the time series property while the other models ignored it and was treated as a proper regression problem.

We can see that XGBoost performed based on TEST_MAPE was better than other models. Linear Regression was able to achieve similar values as XGBoost.

	TEST_MAPE	TEST_MSE
XGBOOST_REG	0.1158	213.0978
LINEAR_REGRESSION	0.1165	218.2750
EMA_PREV	0.1205	218.6057
WMA_PREV	0.1206	220.4236
SMA_PREV	0.1231	232.9855
RANDOM_FORESTS_REG	0.1261	250.1323
EMA_RATIOS	0.1574	473.4599
WMA_RATIOS	0.1580	489.5664
SMA_RATIOS	0.1613	524.1292

Some of the more important features used by these models are pickupdensity(P(t-1)) followed by EMA Model using the previous pickup densities.

Using these predictions, the Yellow taxi owners can decide where to station themselves during their hours of least pickup.

Some of the risks associated with this plan is that the Yellow taxi fleet itself might decline in the years to come with drivers switching to Uber/Lyft, leaving not much data for our model to predict the pickup density.

A compelling business case would be to do a pilot project on the yellow taxi for a year and see if this model really helps increase the revenue. If this works out, the yellow taxis will be going back to their glory days when they dominated the taxi ridership industry and be less impacted by Uber/Lyft.

VII. DEPLOYMENT

Taxi services are a central transportation method in almost all urban areas. Like so many other fields today the taxi business is undergoing a rapid digital transformation with new actors like Uber taking market shares with innovative digital products. The most important objective for any taxi company and driver is to minimize the vacant driving time, and a very important aspect of this is to know where to find the passengers. Predicting where passengers is a problem ideally suited for a machine learning approach.

Exploiting an understanding of taxi supply and demand could increase the efficiency of the city's taxi system. Overall, our models for predicting taxi pickups in New York City performed well. The xgboost regression model performed best, likely due to its unique ability to capture complex feature dependencies. We propose to develop a mobile application that would essentially use the insights from our model to position taxicabs with more efficiency. Our model could be used by both taxi drivers/dispatchers as well as city planners.

Some of the issues associated with the model's use could be that we have used smoothing to average out the bins where pickups would be zero which impacts the accuracy of the model.

One of the major risks associated with our proposed solution is that sometimes there may be latency in Information. Given a location and current time of a taxi driver, as a taxi driver, he/she excepts to get the predicted pickups in his/her region and the adjoining regions. Therefore, it is essential to ensure minimal latency in information.

Some of the risks associated with this plan is that the Yellow taxi fleet itself might reduce year on year with drivers switching to Uber/Lyft hence our model might not have as much data to predict the pickup density accurately in the future.

REFERENCES:

1. Data source - https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
2. https://python-visualization.github.io/folium/docs-v0.6.0/quickstart.html#Getting-Started
3. https://github.com/tkrajina/gpxpy/blob/master/gpxpy/geo.py
4. http://dask.pydata.org/en/latest/dataframe.html
5. http://scikit-learn.org/stable/
6. https://xgboost.readthedocs.io/en/latest/
7. https://pandas.pydata.org/
8. http://robertmitchellv.com/blog-bar-chart-annotations-pandas-mpl.html
9. https://matplotlib.org/
10. https://docs.scipy.org/doc/numpy/reference/