**NYC Yellow Cab Fare Prediction Final Report**

[GitHub Project Link](https://github.com/CeyhunSahinkaya/NYC-Yellow-Cab-Fare-Prediction)

# Table of Contents

[**Table of Contents**](#_heading=h.30j0zll) **1**

**Project** [**Statement**](#_heading=h.1fob9te) **2**

[**Data Collection**](#_heading=h.3dy6vkm) **2**

[**Data Wrangling**](#_heading=h.17dp8vu) **3**

**Feature Engineering 4**

Handling Time Variables  **4**

Handling Coordinate Variables  **4**

Handling Distance Variable **4**

[**Exploratory Data Analysis**](#_heading=h.1ksv4uv) **5**

Fare Amount Distribution **5**

Trip Distance vs. Fare Amount  **6**

Passenger Number vs. Fare Amount  **7**

Datetime Variables vs. Fare Amount  **8**  
 Correlation with Fare Amount **14**

**Model** [**Prediction Analysis**](#_heading=h.2xcytpi) **16**

Linear Regression  **16**

Random Forest Regression  **17** Xgboost Regression  **19**  
 LightGBM Regression **20**

**Potential Improvements**   **22**

[**Conclusion**](#_heading=h.2u6wntf) **23**

# Project Statement

This project is inspired by a Kaggle Competition that challenges to build a model to predict NYC yellow cab fare amount. Subject Kaggle Competition’s link can be found in the Data Collection Section.

Main purpose of the project is to build a model predicting the fare amount (inclusive of tolls) for a taxi ride in New York City given the pickup and dropoff locations. The project can be applied to business models such as a mobile application that gives an estimated fare amount based on riders pick up and destination location, time and some other features for yellow cabs in NYC. The project also includes some exploratory data analysis that shows the correlations between features with graphics and charts.

# Data Collection

This dataset contains a csv file of 55 million rows and 8 columns. Only 8 million rows out of 55 million columns are used in the project due to computing limitations. Below is the feature list:

## Content

There are 8 columns:

* key - Unique string identifying each row.
* pickup\_datetime - timestamp value indicating when the taxi ride started.
* pickup\_longitude - float for longitude coordinate of where the taxi ride started.
* pickup\_latitude - float for latitude coordinate of where the taxi ride started.
* dropoff\_longitude - float for longitude coordinate of where the taxi ride ended.
* dropoff\_latitude - float for latitude coordinate of where the taxi ride ended.
* passenger\_count - integer indicating the number of passengers in the taxi ride.

The dataset is available at the below:

<https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/data>

# 

# Data Wrangling

Dataset is read to a dataframe with 8 million rows since it requires massive computing power to load and manipulate 55 million rows of data. The key column is dropped from the dataframe since it has the same variables with pickup\_datetime column.

First, the dataframe is checked for missing data and 49 rows are found with missing data. Since it is a really small number, we decided to drop the rows that have the missing data instead of using some other missing data replacement techniques.

On the describe table, there were some negative fare amounts in the dataset. We removed them from the dataset. The fare amounts greater than $200 are removed as well since the most expensive cab ride you can ever in New York City shouldn’t be more expensive than $200.

In NYC, the initial charge is $2.5 plus a 50 cent surcharge is added to the fare at the end of the cab ride. Therefore, the cheapest cab ride can’t be less than $3. The fare amounts that are less than $3 are removed.

By law, NYC taxi cabs are not allowed to carry more than four people. Therefore, the rows that have more passengers than four in passenger\_count are removed.

NY coordinates are ( 40.730610, -73.935242), so the coordinates that don't fall between these numbers are removed. However, this is only applied to pick up coordinates since, by law, NYC cabs can drop off passengers to other states, but can only pick them up from NY. We want to include the rides that were picked up from New York to close states like New Jersey, Connecticut and Pennsylvania. Also, the rows that have the same pickup and drop off coordinates are removed from the dataset as well.

After all data cleaning, we successfully removed 1945783 rows that are outliers or don’t make sense to the dataset.

# 

# Feature Engineering

## Handling Time Variables

# First, the pickup\_datetime column is converted to datetime. Then, below columns are created from pickup\_datetime by using the datetime module. New features are used in both EDA and model building. New created datetime columns are:

Pickup\_day: - day of the month indicating when the taxi ride started.

Pickup\_hour: - time indicating when the taxi ride started.

Pickup\_day\_of\_week: - day of the week indicating when the taxi ride started.

Pickup\_month: - month indicating when the taxi ride started.

Pickup\_year: - year indicating when the taxi ride started.

## Handling Coordinate Variables

The absolute difference between longitude and latitude coordinates are calculated and below columns added to the dataset. This is not the actual difference in miles/km but it gives us a good idea to compare each ride.

Latitude\_difference - absolute difference between pickup latitude and dropoff latitude

Longitude\_difference - absolute difference between pickup longitude and dropoff longitude

## Handling Distance Variable

The distance for each cab ride is measured by using the Haversine distance formula. Haversine distance measures the distance between two points by using coordinates and taking into account the spherical shape of the Earth. The code below to calculate Haversine distance is from a Stack Overflow answer that you can see the link below as well as added column to the dataset:

Distance\_in\_km - the great circle distance between two points on the earth

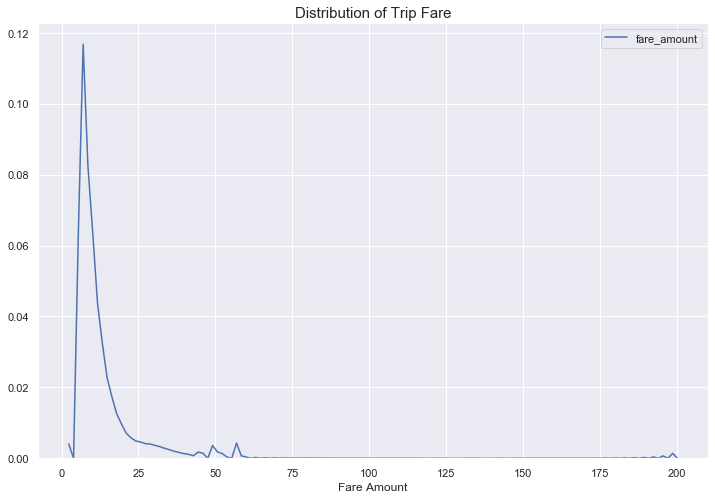
Referenced Stack Overflow link: <https://stackoverflow.com/a/29546836>

# Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a good way to start with your analysis and learn about the dataset. Now, we will look into some features and check how they are correlated with some other features. Also, we will create some charts and plots to visualize the data to make more sense about it. Visualization helps us to ask relevant questions to build our model.

## Fare Amount Distribution

Most of the rides cost less than 25 dollars, average fare amount is $11.36 and median is $8.5.



## 

## Trip Distance vs. Fare Amount

## 

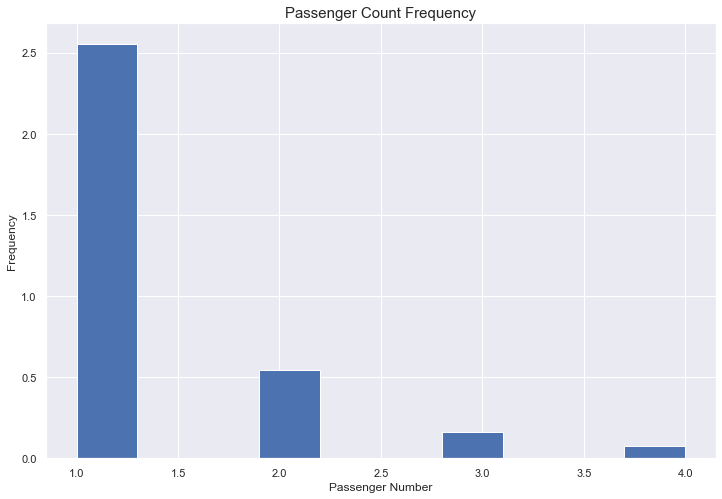
## Above plot shows the relationship between distance and fare amount. There is a linear correlation between them.Fare amount increases as the distance traveled increases as well for the most of the rides. There is a good amount of rides with longer distance and less than $50. It could be the airport rides since there is a flat rate policy to/from NY airports.

## The rides that are very expensive and have very short distance are dropped as outliers since it doesn’t make sense. Also, the rides take more than 50 km and cost less than $10 are removed for the same reason. After these steps, the total row number decreases to 6039637 rows.

## 

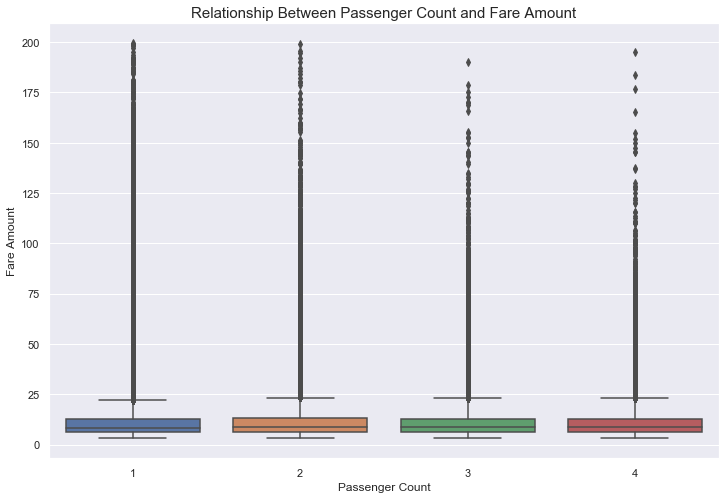
## 

## Passenger Number vs. Fare Amount



Above is the bar plot showing passenger number distribution of total cab rides. 1 passenger is the most frequent and 4 passenger is least frequent.

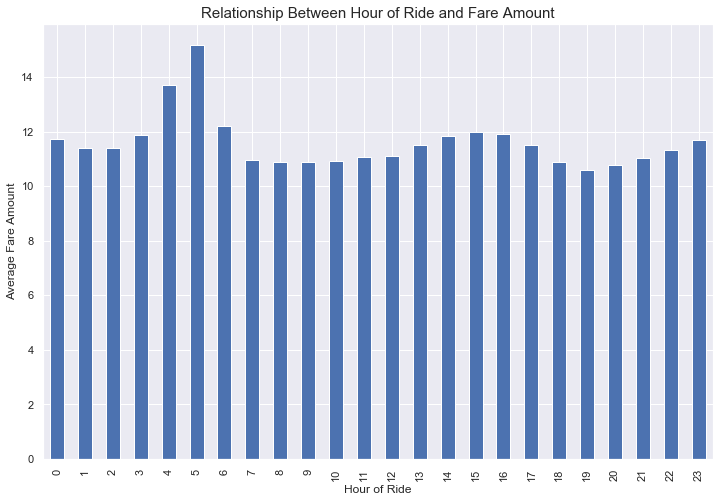
Another 23639 rows were removed from the dataframe since there were 0 passengers on those rides. A cab ride without a passenger is not a cab ride at all!



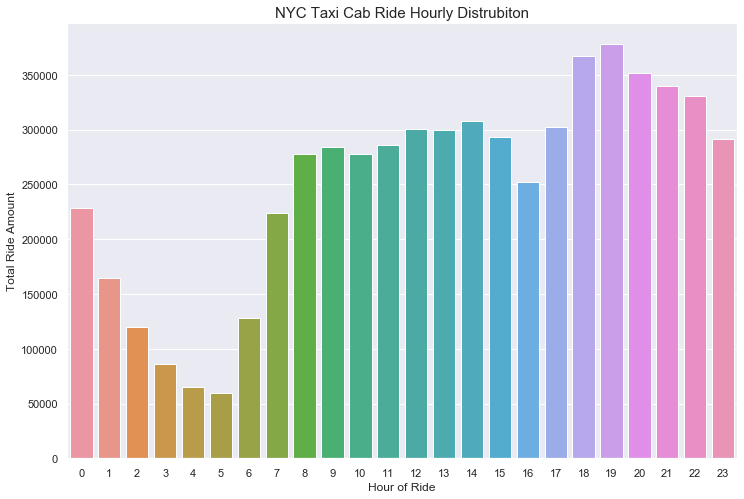
Second graphic is a boxplot showing the relationship between passenger count and fare amount. The box for each passenger count looks similar. They have the same median and whiskers, which indicates that fare amount and passenger count is not correlated with each other.

## **Datetime Variables vs. Fare** Amount

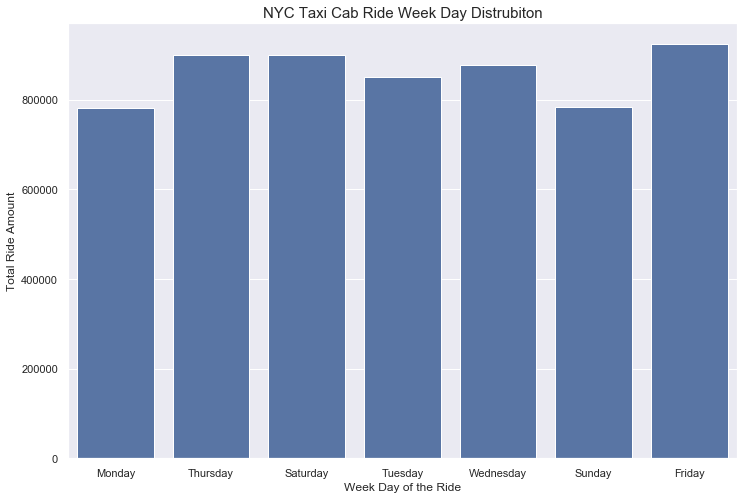
Plots are created to check if the fare amount changes depending on the time, date, month or the year. Let's start with time, we will check if there is a pattern we can see.



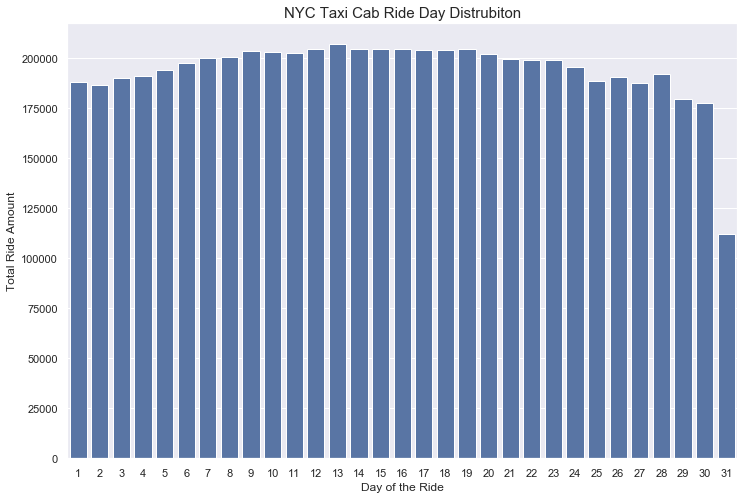
Average fare amount is similar except the rides take place at 4 AM and 5 AM. Average rate is the highest at 5 AM.

****

The least busy time of the day for the taxi cabs in NYC is 4 AM and 5 AM, the busiest time is between 6 PM to 7 PM which makes sense since it is rush hour. The amount of the rides increases from morning till 7 PM, then it decreases till it reaches the lowest level at 7 AM.

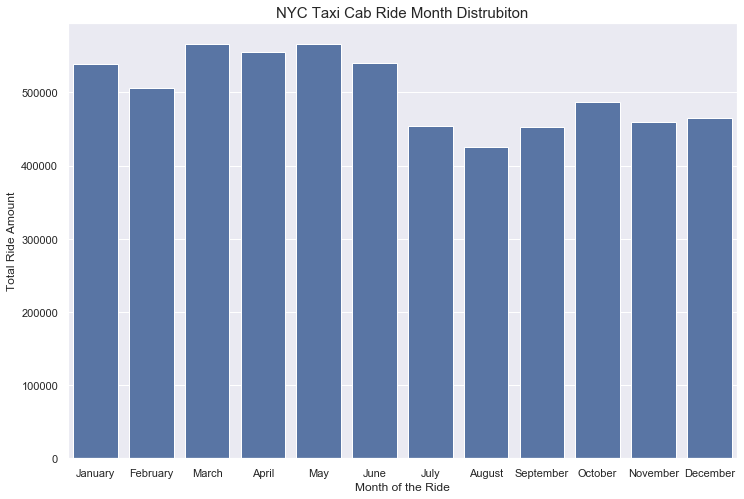
****

Sundays and Mondays are the least busiest days of the week. Total number of rides increases from Monday through Friday. Friday is the busiest day.

****

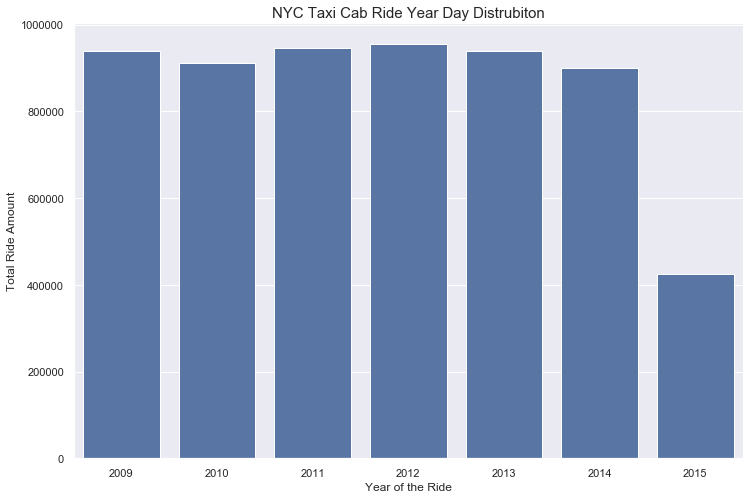
The ride amount goes up from the first day of the month till the 20th day of the month, then goes down and hits the lowest on the last day of the month.

## 

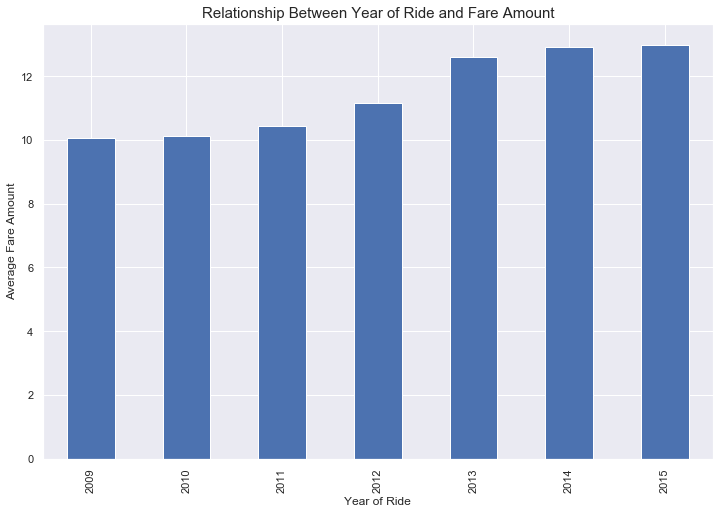


The ride amount hits the lowest in July and August. This makes sense because NYC gets empty during summer. And the busiest season is Spring (March, April, May).

## 

****

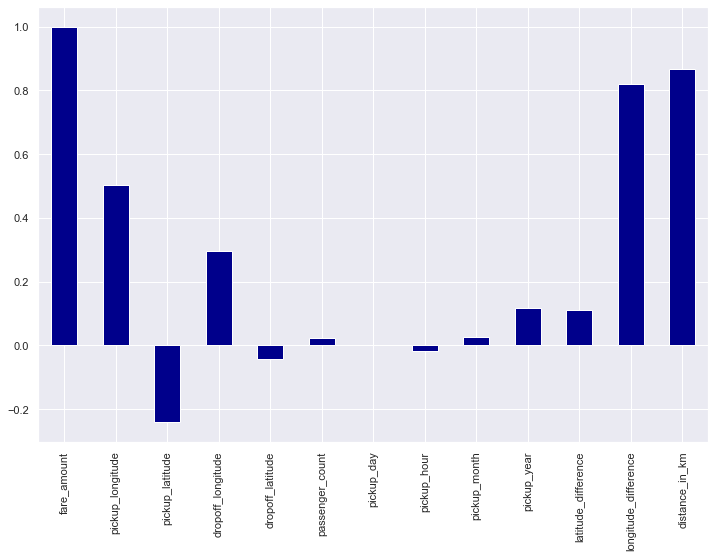
The 2015 ride amount is less than half from other years due to unknown reasons.



The average fare amount increases by the year significantly.

## Correlation with Fare Amount

we can create a plot to see the correlation between the features and the fare amount. The correlation coefficient measures the strength and direction of a linear relationship.



Distance has a positive linear relationship with fare amount. When distance increases the fare amount increases as well. It seems like date time features except pick up year doesn't have a strong relationship with the fare amount.

# 

Above heatmap shows all variable’s correlation between each other. Distance in km is the most correlated variable with fare amount. Most of the variables have little or no correlation with fare amount.

# 

# 

# Model Prediction Analysis

We started building our models to predict the fare amount with our dataset. We started with the simple algorithm, we tried to improve our accuracy by training more complex models with different features and tuning. We started with Linear Regression since there is a strong linear correlation between some features and fare amount.

## Defining Features and Target Value

Before jumping the regression models, we need to define our target value and features. Our features will X and our target value will be y as per below,non numeric columns like pickup\_datetime and pickup\_day\_of\_week are also dropped:

X = All features except fare\_amoun', pickup\_datetime and pickup\_day\_of\_week (predictors)

y = fare\_amount column (target value)

**Training and Test Datasets**

We set up our datasets, we split our sets into training data and test data. The reason behind is that we would like to build a strong model that predicts well. By splitting our data into test and training data, we can use cross validation on our model later. We split our data as 80% training and 20% test data since we have a huge dataset.

## Linear Regression

We fitted features that are highly correlated with fare amount and not correlated with each other on the first linear Regression model.Pickup\_year, distance\_in\_km, and longitude\_difference were selected for this Linear Regression model.

The model's accuracy is measured by using Root mean squared error (RMSE). RMSE is the square root of the average of squared differences between prediction and actual observation. It is one of the most common metrics used in measuring continuous variables. Since RMSE shows the error of the prediction, a lower score is better. Also, when we tested the RMSE score, only 10000 rows were selected to avoid long computing time.

The RMSE score of the first Linear Regression model is 4.515.

All twelve features were fitted to the model for the second time with Linear Regression model and the RMSE score is 4.488. The RMSE score slightly improved with training more feature.

There are a couple more things we can try to improve our score. Since we can't tune any parameters in simple Linear Regression, we can add more variables/observations. Also, we can work building new features. Last, we can try more complex models to get a better RMSE score.

## Random Forest Regression

The second algorithm that is used in this project is Random Forest Regression. Random forest is a good algorithm to avoid overfitting problems and it can be tuned to get better scores even it's default hyperparameters return pretty good results.

First, the sklearn default random forest hyperparameters are used in the model and only 50K rows were fitted to the model to avoid long computing time.

The first RMSE score with Random Forest is 3.631, way better than what we have with Linear Regression. This explains our dataset fits better on non-linear algorithms.

Couple things can be done to get an even better score with Random Forest. We can use cross validation and grid search to tune the hyperparameters and get the best combination tried in different folds.

For the second run with Random Forest, we adjusted some of the most important parameters such as number of trees used in the model, max number of levels in each decision tree and more. Below is the parameter that was used in GridSearch:

n\_estimators: - 100, 200, 300, 400, 500.

Max\_features: - 'auto', 'sqrt'

Max\_depth: - 10, 20, 30, 40, 50, None

min\_samples\_split: - 2, 5, 10

min\_samples\_leaf: - 1, 2, 4

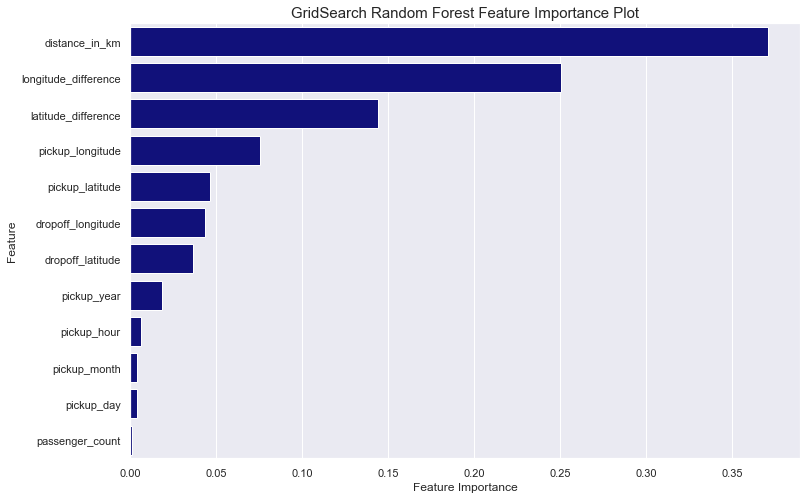
bootstrap: - True, False

Based on GridSearch best parameters, we used below parameters on the second run with Random Forest.

bootstrap=True, max\_depth=80, max\_features=3, min\_samples\_leaf=5, min\_samples\_split=12, n\_estimators=100

The second RMSE score is 3.471 and We increased RMSE score 4.8% by using GridSearch to find the best parameters and using them in the model. Idea to improve RMSE: we only fitted the first 50K rows to our model. We can fit more rows to get a better score even though it takes more time to compute.

Let's check the most important features for the Random Forest Model.



Most important feature for random forest is distance\_in\_km, and there are some feature variables that are not important for the model.

Ideas to Improve: Since the distance is the most important feature for this model, we can look into using a different distance measurement ( euclidean or manhattan) to improve score. Also, we can remove the features that have little or no importance to improve the score. Last, we can try some other complex algorithms to see if we can score a better RMSE score.

## XGBoost Regression

Xgboost regression algorithm is one of the most popular algorithms in the modern machine learning world due to its computing speed and performance. We will try our dataset on this algorithm since our data fits better on non-linear complex algorithms.

All features and 50000 rows were trained on the training test with XGBoost Regression. The RMSE score without tuning is 3.404 which is the best score so far.

For the second run, below grid parameter was used in GridSearch:

Learning\_rate - 0.07, 0.1, 0.3

max\_depth: - 3, 5, 7

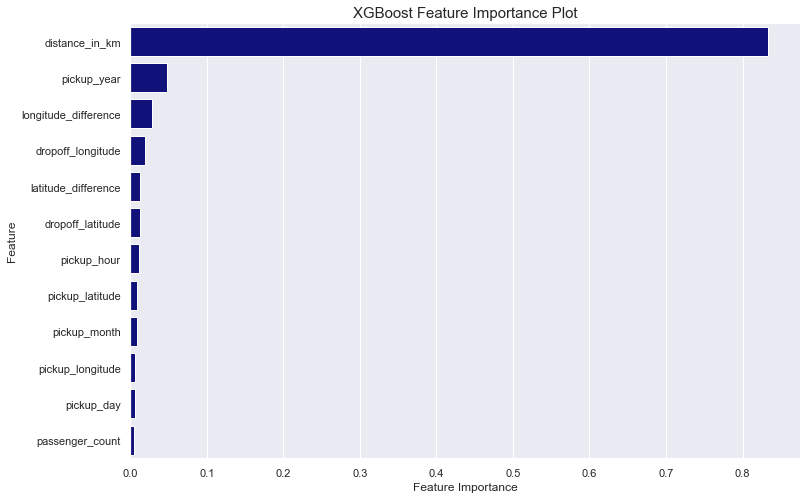
N\_estimators - 200, 400, 500

Based on GridSearch best parameters, we used below parameters on the second run with XGBoost.

learning\_rate=0.1, max\_depth=7, n\_estimators=400

The second RMSE score is 3.408 and we increased RMSE score 3.3% by using GridSearch to find the best parameters and using them in the model.

Idea to improve: Since we improved RMSE with parameter tuning with GridSearch, there are more hyperparameters that can be tuned such as colsample\_bytree,gamma. We can add these parameters to Gridsearch (or RandomSearch) to improve RMSE.



One and only important feature here is distance in km. Therefore, trying different measurement calculations using coordinates may make a significant difference on this model's prediction power.

## LightGBM Regression

The last algorithm we will try on our dataset is LightGBM Regression algorithm. Since LightGBM can handle the large size of data and takes lower memory to run, it is a good option for our dataset.

All features and 50000 rows were trained on the training test with LightGBM Regression this time.The RMSE score without tuning is 3.413 which is slightly worse than what we acquired with XGBoost.

For the second run, below grid parameter was used in GridSearch:

N\_estimators - 400, 700, 1000

Colsample\_bytree - 0.7, 0.8

Max\_depth - 15, 20, 25

Num\_leaves - 50, 100, 200

Reg\_alpha': - 1.1, 1.2, 1.3

Reg\_lambda' - 1.1, 1.2, 1.3

min\_split\_gain': - 0.3, 0.4

'subsample': [0.7, 0.8, 0.9],

'subsample\_freq': [20]

Based on GridSearch best parameters, we used below parameters on the second run with LightGBM Regression.

learning\_rate=0.07,

boosting\_type='gbdt',

n\_estimators=400,

max\_depth=15,

min\_split\_gain=0.3,

num\_leaves=50,

reg\_alpha=1.3,

reg\_lambda=1.1,

subsample=0.7,

subsample\_freq=20,

colsample\_bytree=0.7

The second RMSE score is 3.408. The score has slightly increased but RMSE with tuned XGBoost is the best score so far.

## 

## 

## Potential Improvements

There are still couple things that we can work on to improve RMSE score for each model:

* We can fit more observations/ variables to improve models' scores. Since we have a huge dataset and can't load the whole dataset, we can work on reducing memory consumption.
* We can apply StandartScaler or any other Scaler to scale the dataset and re-run the models and check if the scores are improved.

## 

# 

# 

# Conclusion

This project includes complex models that predict NYC cab fare amount with RMSE $ 3.37 . The project can easily be applied to a business model. We can create an app telling the customers how much their cab ride will be based on their location, time and passenger count.

Based on our analysis:

- Most of the rides cost less than 25 dollars, average fare amount is $ 11.36.

- The least busy time of the day for the taxi cabs in NYC is 4 AM and 5 AM, the busiest time is between 6 PM to 7 PM.

- Most of the rides are with one passenger.

- The busiest season is Spring for NYC yellow cabs and the ride amounts hit the bottom in July and August.

- Average fare amount is increasing every year.

### 

Best RMSE scores with 4 algorithms:

- Linear Regression: 4.488

- Random Forest: 3.470

- Xgboost Regression: 3.371

- Light GBM Regression: 3.408