Cab Fare Prediction Model

A dissertation submitted to

JAIN UNIVERSITY

by

Shashank Pandey

in partial fulfillment of the requirements

for the degree of

BACHELOR OF SCIENCE

in

DATA SCIENCE AND ANALYTICS

DEPARTMENT OF DATA SCIENCE

JAIN UNIVERSITY

BANGALORE -- 560027

MAY – 2021

CERTIFICATE

This is to certify that the project entitled Cab Fare Prediction Model is a bona-fide record of work done by ***Shashank Pandey, 18BSR18004,*** *during the academic year 2018-2021, in partial fulfillment of the requirements for award of the degree of Bachelor of Science in Data Science and Analytics.*

Project Supervisor

Arun Selvi

This report is submitted for B.Sc. Degree Examination held on 3,June, 2021

CONTENTS

Abstract:

The cab service industry is booming for the last couple of years and it is expected to grow in the near future. Taxi drivers need to decide where to wait for passengers as they can pick up someone as soon as possible. Passengers also prefer a quick taxi service whenever needed. The control center of the taxi service decides the busy area to be concentrated. In the existing system, sometimes the taxis were scattered across the larger area missing the time-based busy area like Airport, Business area, school area, Train stations etc. Effective taxi allocation can help both drivers and passengers to minimize the wait-time to find each other. In the proposed system, the future demand can be predicted using Multiple Linear Regression based model that can be trained with given historical data. It can serve more customers in a short time by organizing the availability of taxi. The data set includes GPS location and other properties of the taxi like drop point, pickup point etc. This model is used to predict the Fare for a particular train dataset.

Key Words: Cab Fare Prediction, Multiple Linear Regression, Bounding Box

Introduction:

PROBLEM STATEMENT: The project is about a cab company who has done its pilot project and now they are looking to predict the fare for their future transactional cases. As, nowadays there are number of cab companies like Uber, Ola, Meru Cabs etc. And these cab companies deliver services to lakhs of customers daily. Now it becomes really important to manage their data properly to come up with new business ideas to get best results. In this case, earn most revenues. So, it becomes really important to estimate the fare prices accurately. The goal of this challenge is to predict the fare of a taxi trip given information about the pickup and drop off locations, the pickup date time and number of passengers travelling.

In any analytics project 80% of the time and effort is spent on data cleaning, exploratory analysis and deriving new features. In this project, we aim to clean the data, visualize the relationship between variables and also figure out new features that are better predictors of taxi fare. Taxi drivers need to decide where to wait for the passengers so that they can pick someone quickly. Similarly, passengers also need to find their cabs quickly. Dispatching the taxi efficiently help both the customers and drivers. Effective dispatching of taxi helps to reduce waiting time for customers, as well as drivers. A driver will not have enough information about where to wait in order to get passengers quickly. A taxi centre can organize and send the required number of taxis to the area based on the historical data. The historical data uses Global Positioning System (GPS) and predict future demand

The taxi fare rate prediction challenge was organized by the 2017 EPIA conference and proposed as a Kaggle Competition. The challenge consisted of taxi fare prediction based on various features and selected meta-information associated with each ride. Such a prediction model can aid the user and the taxi companies in predicting the fare rate based on derived features rather than use of the end location. The training dataset contains 1.4 million data points. Each sample corresponds to one complete trip and contains the following attributes:

- ID (integer): Unique identifier for each trip.
- TAXI_ID (integer): Unique identifier for each vehicle.
- TIMESTAMP (float): Julian timestamp that identifies the trip start.
- STARTING_LATITUTE (float): Latitude coordinate of the trip start location.
- STARTING_LONGITUDE (float): Longitude coordinate of the trip start location.

In the system, the multiple linear regression is used and Python language is preferred because it has a vast collection of machine learning libraries. The data set might contain empty values, negative values or error. Data set is cleaned in the pre-processing. The pre-processing methods involve removing records which are not complete. Once the cleaned data set is available it is prepared to be fed to the machine learning algorithm.
Machine learning (ML) is closely related to computational statistics, which focuses on making predictions using computers. Data mining (DM) is a field of study within ML and focuses on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Machine learning tasks are classified into several broad categories. In supervised learning, the algorithm builds a mathematical model from a set of data that contains both the inputs and the desired outputs. Classification algorithms and regression algorithms are examples of supervised learning Regression algorithms are named for their continuous outputs, meaning they may have any value within a range [1]. In unsupervised learning, the algorithm builds a mathematical model from a set of data that contains only inputs and no desired output labels. Unsupervised learning algorithms are used to find structure in the data, like grouping or clustering of data points. Unsupervised learning can discover patterns in the data and can group the inputs into categories, as in feature learning. Dimensionality reduction is the process of reducing the number of "features", or inputs, in a set of data. Machine learning and data mining often employ the same methods and overlap significantly, but while ML focuses on prediction, based on known properties learned from the training data, data mining focuses on the discovery of (previously) unknown properties in the data. This is the analysis step of knowledge discovery in databases (KDD) [1]. Data Mining uses many Machine Learning methods, but with different goals; on the other hand, ML also employs data mining methods as "unsupervised learning" or as a pre-processing step to improve learner accuracy.

Methodology

Pre-Processing

The work starts with data pre-processing, means looking at the data to get insights. However, in data mining termslooking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis.To start this process we will first try and look at all the distributions of the variables . Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the distributions of the variable by QQ-normality graph or histogram. Histogram is the best chart to represent the data distribution, later the data is normalized or standardized, according to the model needs. And we check for distribution of the target variable with respect to its independent features. Data pre-processing is the tedious task, we need to focus more on this part to reduce the model complexity. Before feeding the data to model, we must pre-process the data, which has various stages like missing value analysis, impute the missing values, outlier check, normality check, sampling, we at her the data set is imbalanced are not. Then the data is subjected to model training and prediction. After completing the model prediction, the machine learning prediction system is ready for deployment. Project deployment refers to, preparing the machine learning environment to the front end users.

The Incorrect information.

Exploring the data, which means looking at all the features and knowing their characters. According to our problem statement. We are analysing the data of cab rental startup, whose features are comprised off are amount, passenger count and geometrical coordinates like pickup latitude, longitude and drop off longitude and latitude. So keeping in mind that these variables play an important role in the analysis, we keep checking the minimum and maximum values in this features. There were a lot of incorrect information in this feature, like observation showed up passenger count as 0 and 208 and few showed decimal values 1.3 And fare amount ranged from 1 to 500 and its normal, few showed high as 4000 and 5000...etc, which is not possible. Latitudes range from-90 to 90. Longitudes range from -180 to 180. So Removing those features, which does not satisfy these ranges. Many showed 0. So these residual information must be removed from dataset, this is known as data cleaning. Many observations nearly 15628 has this wrong information, so considering this as OUTLIERS in the data and removing it. Deriving the new features year, weekday, hour by time stamp variable pickup _datetime. Deriving the new feature distance from the given coordinates with the help of haversine function. After this process we will move forward to deploy the model and test the accuracy.

Literature Survey

Fare and Duration Prediction: A Study of New York City Taxi Rides

Christophoros Antoniades, Delara Fadavi, Antoine Foba Amon Jr.

December 16, 2016

1 Introduction

New York City taxi rides paint a vibrant picture of life in the city. The millions of rides taken each month can provide insight into traffic patterns, road blockage, or large-scale events that attract many New Yorkers. With ridesharing apps gaining popularity, it is increasingly important for taxi companies to provide visibility to their estimated fare and ride duration, since the competing apps provide these metrics upfront. Predicting fare and duration of a ride can help passengers decide when is the optimal time to start their commute, or help drivers decide which of two potential rides will be more profitable, for example. Furthermore, this visibility into fare will attract customers during times when ridesharing services are implementing surge pricing.

In order to predict duration and fare, only data which would be available at the beginning of a ride was used. This includes pickup and dropoff coordinates, trip distance, start time, number of passengers, and a rate code detailing whether the standard rate or the airport rate was applied. Linear regression with model selection, lasso, and random forest models were used to predict duration and fare amount.

2 Related Work

The fare of a taxi ride is function of the mileage and the duration of the ride (sum of drop charge, distance charge and time charge). The drop charge is constant and the distance can easily be estimated but evaluating the duration is not a trivial task. It is the result of complex traffic processes that are nonlinear.

One way to predict duration is by doing short term prediction with the help of real time data collection. In [1] the authors tackle the problem by using data from from buses (GPS) and an algorithm based on Kalman filters. Using a similar approach, [2] uses real time data from smartphone placed inside vehicles.
Estimating travel time for highways yields better results than in the cities. This allows for more accurate predictions. In [3] the authors use a combination of traffic modelling, real time data analysis and traffic history to predict travel time in congested freeways. They try to overcome the assumption that real time analysis communication is instantaneous. A lot of other papers also work on freeways. In [4] the prediction is done using Support Vector Regression (SVR) while in [5] Neural Networks (SSNN) are used.

Predictive estimates of future transit times is a feature that was released in 2015 in the Google Maps API [6]. This shows the importance of being able to predict time travel without having real time data of traffic.

We are trying to solve a similar problem: estimating ride duration without real time data, by analysing data collected from taxis. Being able to do such estimation would help making better future predictions.

3 Data

The data used in this study are all subsets of New York City Taxi and Limousine Commission's trip data, which contains observations on around 1 billion taxi rides in New York City between 2009 and 2016. The total data is split between yellow taxis, which operate mostly in Manhattan, and green taxis, which operate mostly in the outer areas of the city. For the main analyses of this study, the data for yellow taxi rides during the month of May 2016 were used, although the models were validated on additional data. Since each month consists of about 12 million observations, and there were computational limitations, subsets of the monthly data were used for model building, and other subsets were used for validation. To build the models, a random subset of 10,000 observations from May 2016 were used, of which 8,000 were used for training and 2,000 for validation.

The original dataset contains features as pickup and dropoff locations, as longitude and latitude coordinates, time and date of pickup and dropoff, ride fare, tip amount, payment type, trip distance and passenger count (as well as other, for this

study, less relevant variables). The data was processed to extract separate features for year, month, day, weekday, hour and minute from the date and time of each ride, as well as trip duration as the difference between dropoff and pickup time. Furthermore, with the objective to model and account for traffic in the predictions, two more features were calculated from the data; rides in an hour and average speed during the hour. Rides in hour represents the number of started rides within the hour of each observation, and the average speed represents the average speed of all those rides.

Figure 1 show the distributions of ride duration and fare amount, which are clearly similar (the spike at $52 represents rides to JFK International airport). The objective of this study has been to predict both, although as the results and models chosen are very similar, the illustrations and results have been focused on the prediction of trip duration.

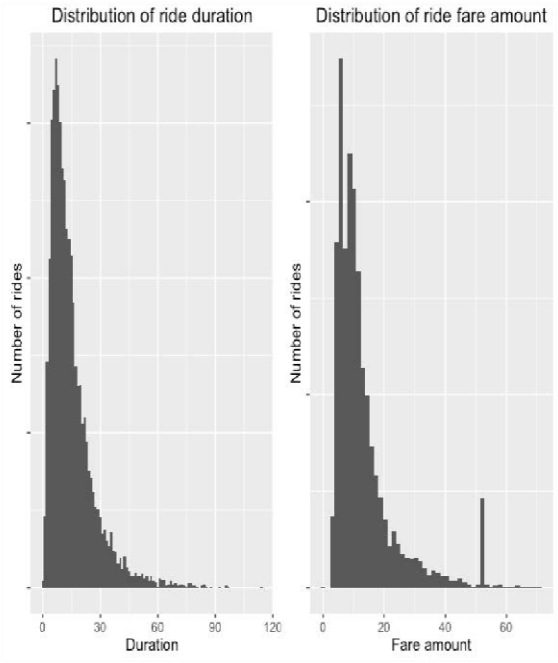


Figure 1: Duration and Fare Amount Distribution

4 Models and Methodology

4.1 Linear Regression

As a baseline prediction, the mean duration and fare from the training set were used to predict a constant value for the validation set.

Model Selection

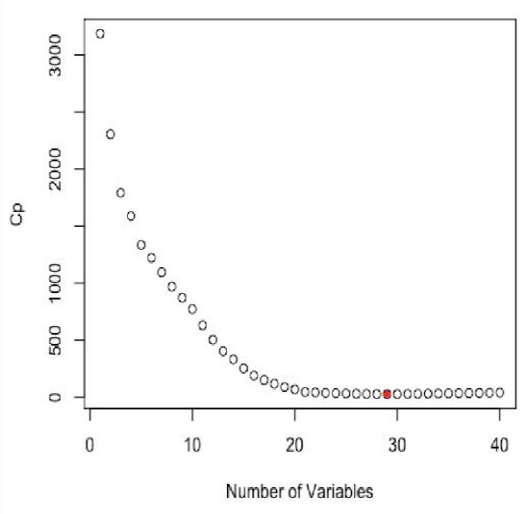


Figure 2: Forward Selection for Linear Regression

To avoid selecting a sub-optimal model by selecting covariates by hand, forward selection was used to identify which subset of covariates would be best to use. When iteratively adding the variables that minimizes the RSS one at a time it is evident that selecting a model with all covariates does not improve the C_p score (proxy for test error) over a 20-covariate model (figure 2). Therefore, for simplicity reasons, the smallest model was selected for linear regression.

Lasso

To further confirm the best set of covariates to use, the lasso method was used to shrink coefficients. Lasso results in a sparser model, which makes interpreting the model easier.

$$\sum_{i=1}^{m}(y^{(i)}-\theta_0-\sum_{j=1}^{n}\theta_j x_j^{(i)})^2+\lambda\sum_{j=1}^{n}|\theta_j| = RSS+\lambda\sum_{j=1}^{n}|\theta_j| \quad (1)$$

For both the fare and duration prediction models, lasso was run using a range of values for the penalizing parameter, λ. Cross validation was used to find the lasso model with the lowest error and select the value of λ to use. In both cases, the λ values that gave the lowest cross validation error were close to zero. For example, the optimal λ parameter for the fare regression was $1.19 \cdot 10^{-5}$. Because of these results, variables were not penalized using lasso, and the linear regression model selected with forward selection was used.

Linear Regression Model

From feature selection, the linear regression with all covariates available at the pickup time was predicted to be the best for both duration and fare prediction. The linear regression model finds the set of θ coefficients that minimize the sum of squared errors

$$y^{(i)} = \theta_0 + \sum_j \theta_j x_j^{(i)} \quad (2)$$

Interaction and Higher Order Terms

Because the available covariates alone cannot model the nonlinear effects of traffic, interaction and higher order terms are considered to allow the model to fit these effects more closely. Plotting the covariates used in the linear models against each other shows that there are no strong correlations between them, which suggests that interaction terms should not be included in the model.

However, second order terms do make logical sense to be used in the model. Since the nonlinearities arise out of traffic patterns, it can be assumed that longer distance trips experience more instances of traffic. Therefore, adding squared trip distance to the model would increase the trip distance's importance in the duration and price prediction.

4.2 Random Forest

As traffic is clustered and aggregated more densely to different locations at different times, the location of the ride will clearly have an affect on the trip duration. Although there is no straightforward way of considering all locations between the start and end points of a ride, the pickup and dropoff locations are available in the dataset and can be used to model some of the effect of traffic and conjunctions. In the linear regressions, the locations' effect on trip duration is modeled simply by the magnitude of the longitude and latitude coordinates. As traffic is clearly not varying solely based on the magnitude of the coordinates, the linear models fail to account for the nonlinear effect the locations have on traffic and hence trip duration (and also fare amount). An algorithm that can better account for these nonlinearities is the random forest.

The random forest algorithm aggregates many decision trees built on bootstrapped samples of the training data in order to reduce the high variance of a single decision tree and improve prediction accuracy [7][8]. Each of these decision trees aims to divide the predictor space, i.e. the set of all possible values for the features $x_1, x_2, ..., x_n$, in J distinct and non-overlapping regions $R_1, R_2, ..., R_J$. The predictor space is divided into high-dimensional rectangles, with the goal to find rectangles $R_1, R_2, ..., R_J$ that minimize the RSS,

$$\sum_{j=1}^{J} \sum_{i \in R_j} \left(y^{(i)} - \hat{y}_{R_j} \right)^2 \quad (3)$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth rectangle. When building each tree, a top-down approach is taken. Beginning with all points in the same region, the algorithm successively splits the predictor space into two halves, stopping when there are no more than five points in a region. At each split, a prediction x_j and cutpoint s are chosen such that splitting the predictor space into the regions $\{x \mid x_j < s\}$ and $\{x \mid x_j \geq s\}$ leads to the biggest reduction in RSS. Defining the pair of halves as $R_1(j, s)$ and $R_2(j, s)$, at each split we seek to find j and s that minimize the equation

$$\sum_{i:\, x^{(i)} \in R_1(j,\, s)} \left(y^{(i)} - \hat{y}_{R_j} \right)^2 + \sum_{i:\, x^{(i)} \in R_2(j,\, s)} \left(y^{(i)} - \hat{y}_{R_j} \right)^2 \quad (4)$$

Once the regions are defined, a prediction by a single tree is made by averaging the responses of the training observations in the region to which the test observation belongs. In the random forest, a large number of trees are fit, each using a bootstrap sample from the training data, and a prediction of a new observation is made using the mean of the predictions by all the trees. At each split, only m of the total n predictors are randomly chosen to be considered. This approach is taken to decorrelate the trees, as considering all predictors might yield very similar trees when one or a few predictors are particularly strong. As averaging many uncorrelated trees leads to a larger reduction in variance, this approach often yields better prediction results. As can be seen in figure 3, the model performs better for a smaller choice of m. Also, averaging over a larger number of trees yields a better results, although the effect is flattening out after a few hundred trees. To optimize prediction accuracy, $m = \sqrt{n}$ and 500 trees were used.

3

Conclusion of Literature survey

Considering what is and what is not accounted for in the models built in this study, their predicting results are fairly accurate. To further improve the prediction accuracy, more variabilities need to be considered and modelled. Although the rides in hour and average speed in hour work as proxies for traffic, more modelling on the effect of location is needed. These quantities could be calculated for different areas to further model local effects of traffic. Also, modelling traffic and the effect of location in between pickup and dropoff points should be considered as well as difference in driver's speed. These further steps could be taken both by analysing larger sets of the data to infer relationships and effects of location and traffic at different times, as well as aggregation with other datasets, as data on

traffic, speed limitations, etc. As we can see they have tried to apply different kind of regression and prediction models to get the best accuracy for their model.

The DATA

The data for this problem can be found on Kaggle. For purposes of this analysis, I have imported 1 million rows out of the 1.4 million rows from the training data. The fields that are present in the data are as below:

Feature Name	Feature Description	Feature Data Type
Key	This is the unique identifier. This is combination of pickup datetime and an unique identifier	string
pickup_datetime	Date time when trip started	timestamp
pickup_longitude	longitude coordinate of where trip started	float
pickup_latitude	latitude coordinate of where the trip started	float
dropoff_longitude	longitude coordinate of where trip ended	float
dropoff_latitude	latitude coordinate of where the trip ended	float
passenger_count	number of passengers in taxi ride	integer
fare_amount	cost of the taxi ride in dollars. This is the value to be predicted. It is not present in the test data	float

Following table explains how the variables are categorized.

Independent Variables
pickup_datetime
pickup_longitude
pickup_latitude
dropoff_longitude
dropoff_latitude
passenger_count

Independent Variables

Dependent Variables
Fare_amount

Dependent/Target Variable

From the given train data, it is understood that, we have to predict fare amount, and other variables will help me achieve that, here pickup_latitude/longitude, dropoff_latitude/longitude this data are signifying the location of pick up and drop off.

It is explaining starting point and end point of the ride. So, these variables are crucial for us. Passenger_count is another variable, that explains about how many people or passenger boarded the ride, between the pickup and drop off locations. And pick up date time gives information about the time the passenger is picked up and ride has started. But unlike pick up and drop off locations has start and end details both in given data. The time data has only start details and no time value or time related information of end of ride. So, during pre-processing of data we will drop this variable. As it seems the information of time is incomplete.

Also, there is a separate test data given, in the format of CSV file containing 9514 observations and 6 variables. All of them are the independent variables. And in these data at the end, we have to predict the fare or the target variable.

Train Data

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.84161	40.712278	1
16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1
5.7	2011-08-18 00:35:00 UTC	-73.982738	40.76127	-73.991242	40.750562	2
7.7	2012-04-21 04:30:42 UTC	-73.98713	40.733143	-73.991567	40.758092	1
5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1
12.1	2011-01-06 09:50:45 UTC	-74.000964	40.73163	-73.972892	40.758233	1

Test Data

pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
2015-01-27 13:08:24 UTC	-73.97332	40.7638054	-73.9814301	40.7438355	1
2015-01-27 13:08:24 UTC	-73.9868622	40.7193832	-73.9988861	40.7392006	1
2011-10-08 11:53:44 UTC	-73.982524	40.75126	-73.979654	40.746139	1

Algorithm and Techniques

Multiple Linear Regression (MLR)

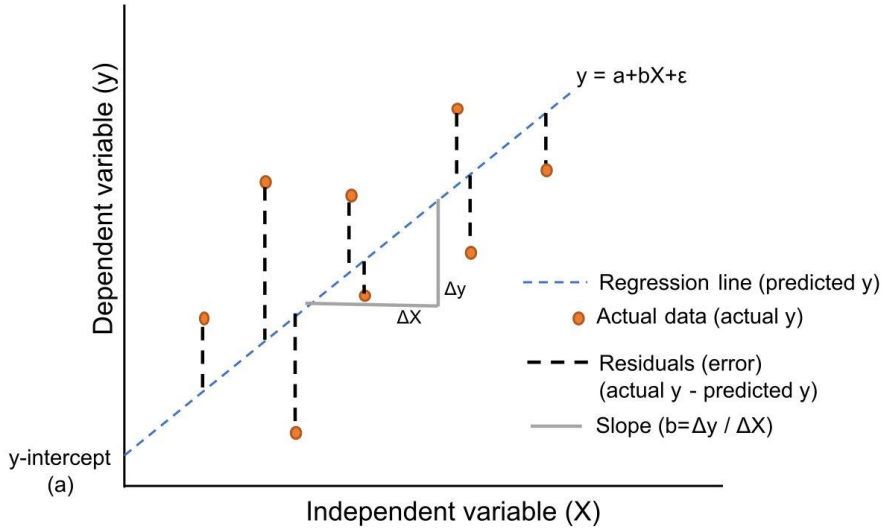
- Multiple Linear Regression (MLR), also called as Multiple Regression, models the linear relationships of one continuous dependent variable by two or more continuous or categorical independent variables. If the dependent variable is measured on an ordinal scale (e.g. Likert-type scale for severity of disease), it is better to use ordinal logistic regression (ordinal regression).
- In contrast to simple linear regression, the MLR model is useful when there are multiple independent variables, as it estimates an individual slope of the regression line for each independent variable.
- If we have n independent variables (X), the MLR model for predicting the dependent variable (y) can be represented as,

$$y=a+b1X1+b2X2+b3X3+...+bnXn+\epsilon$$

Where, a = y-intercept, b = slope of the regression line (unbiased estimate) and ϵ= error term (residuals)

- MLR makes similar assumptions as simple linear regression. In addition, there should be no multicollinearity among the independent variables.

Note: Dependent variable also called a response, outcome, regressand, criterion, or endogenous variable. Independent variable also called explanatory, covariates, predictor, regressor, or exogenous variable.



In this project we applied a multiple linear regression model with multiple features as our X and a single feature in Y

List of features which were derived during Feature Engineering.

Year – used lambda function to derive it from Pickup_datetime

Hour - used lambda function to derive it from Pickup_datetime

Weekday - used lambda function to derive it from Pickup_datetime

Distance_miles – Derived from distance function, features which were used are pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_latitude

Passenger_count – some trips were having more than 4 passengers which is not possible for a cab so we need to remove those entries with more than 4 passengers.

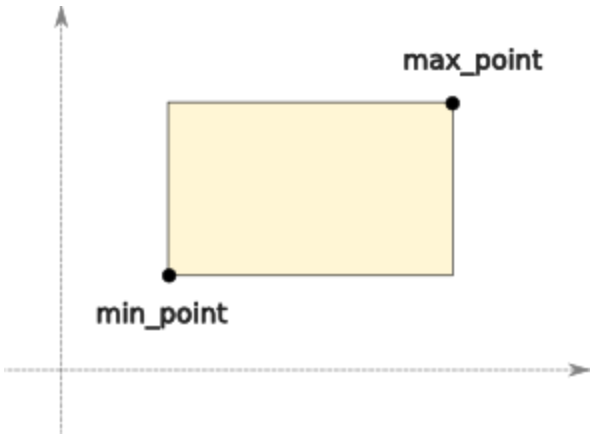
Distance_to_center – Derived from distance function with respect to New York City centre coordinates, pick_latitude and pickup_longitude.

Direction – First we derived delta_lon and delta_lat by subtracting pickup_latitude and pickup_longitude from dropoff_latitude and dropoff_longitude and then applied haversine formulae

Functions

Bounding Box

Bounding boxes are axis-aligned rectangles. They are the simplest closed shape type in planar, represented by two points containing the minimum and maximum coordinates for each axis.



Bounding boxes can be useful as standalone shapes, but they are primarily used for approximating more complex shapes to speed operations such as containment checks and intersection. To facilitate this, all finite shapes defined by planar have a bounding_box attribute which returns the smallest Bounding Box object that completely contains the shape. Bounding boxes can also be constructed directly from an arbitrary sequence of points.

There were many values in the longitude and latitude features which does not satisfy the range of longitude and latitude as we know Latitudes range from-90 to 90. Longitudes range from -180 to 180.

So we have put the longitude and latitude of New York City in our bounding box function so that our model could not take any values which are outside of New York City or which does not come in the ranges of latitude and longitude.

Haversine formula

The **Haversine** formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface. It is important for use in navigation. The haversine can be expressed in trigonometric function as:

$$haversine(\theta) = sin^2\left(\frac{\theta}{2}\right)$$

The haversine of the central angle (which is d/r) is calculated by the following formula:

$$\left(\frac{d}{r}\right) = haversine(\Phi_2 - \Phi_1) + cos(\Phi_1)cos(\Phi_2)haversine(\lambda_2 - \lambda_1)$$

where r is the radius of the earth (6371 km), d is the distance between two points, ϕ_1 , ϕ_2 is the latitude of the two points, and λ_1, λ_2 is the longitude of the two points respectively. Solving d by applying the inverse haversine or by using the inverse sine function, we get:

$$d = rhav^{-1}(h) = 2rsin^{-1}(\sqrt{h})$$

or

$$d = 2rsin^{-1}\left(\sqrt{sin^2\left(\frac{\Phi_2-\Phi_1}{2}\right) + cos(\Phi_1)cos(\Phi_2)sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$$

The distance between Big Ben in London (51.5007° N, 0.1246° W) and The Statue of Liberty in New York (40.6892° N, 74.0445° W) is 5574.8 km. This is not the exact measurement because the formula assumes that the Earth is a perfect sphere when in fact it is an oblate spheroid.

We used haversine formula in this project to calculate the trip distance.

Python-Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
pd.set_option('display.max_rows',500)
%matplotlib inline
```

```
sns.set_style('whitegrid')
```

Train Data

```
df_train = pd.read_csv('D:/Board Infinity/Project/New york taxi fare data/train.csv' , nrows=1000000, parse_dates = ["p
```

Total libraries and functions which have been used in the project.

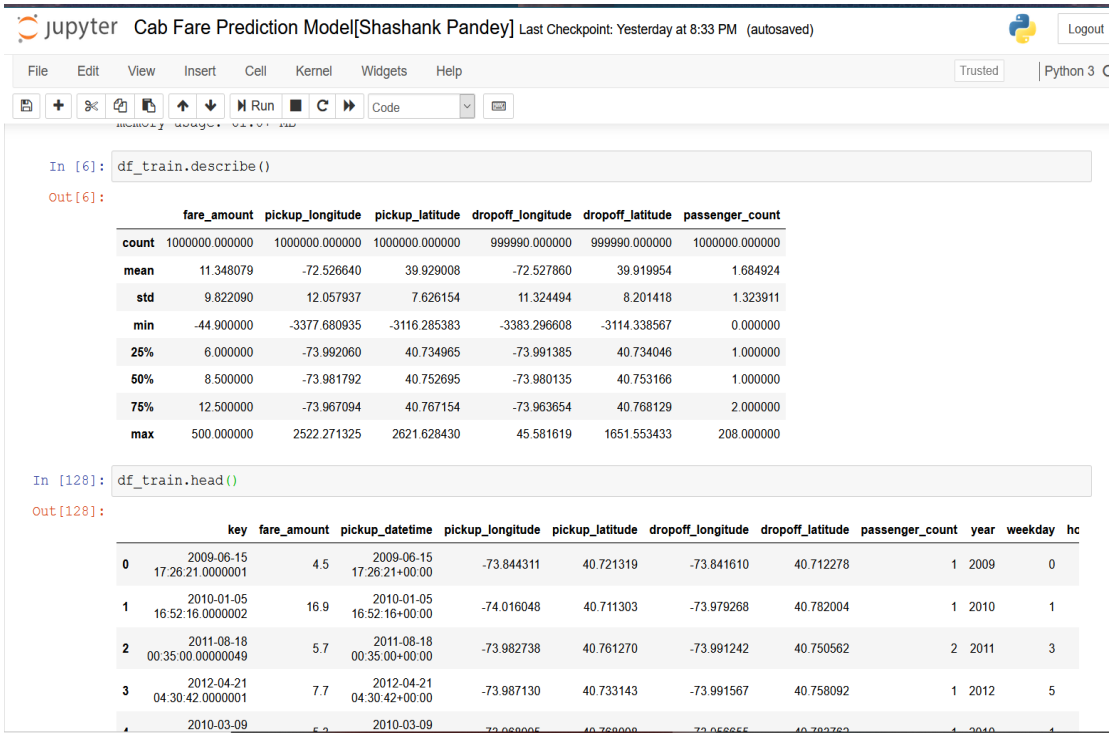
Train Data

```
: df_train = pd.read_csv('D:/Board Infinity/Project/New york taxi fare data/train.csv' , nrows=1000000, parse_dates = ["p
```

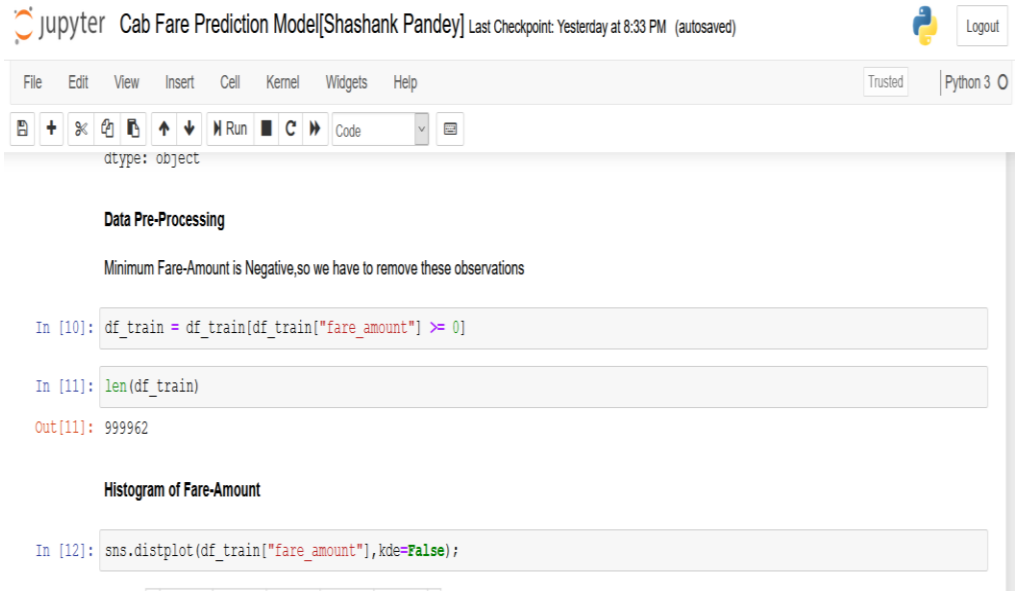
EDA(Exploratory Data Analysis)

```
: df_train.info()
```

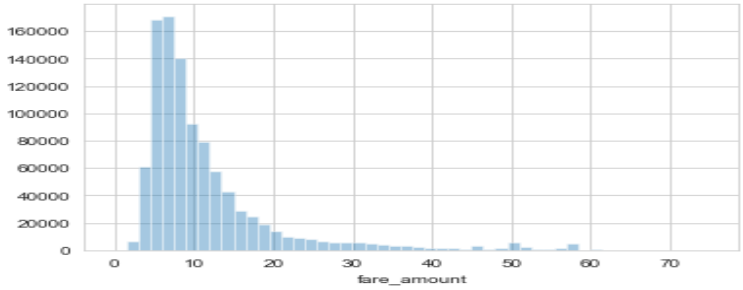
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
key                  1000000 non-null object
fare_amount          1000000 non-null float64
pickup_datetime      1000000 non-null datetime64[ns, UTC]
pickup_longitude     1000000 non-null float64
pickup_latitude      1000000 non-null float64
dropoff_longitude    999990 non-null float64
dropoff_latitude     999990 non-null float64
passenger_count      1000000 non-null int64
dtypes: datetime64[ns, UTC](1), float64(5), int64(1), object(1)
memory usage: 61.0+ MB
```



As we can in df.describe () function some fare amounts are in negative values and some passenger count is 0 or as highest as 208 which is not possible so we need to remove these values.



sns.distplot(df_train[df_train["fare_amount"]<75]["fare_amount"],kde=False);



So in the Data Pre-Processing part we remove the unwanted observations from our data and the plotted a histogram of fare amount to check the range of maximum fares.

The histogram shows us that the maximum number of fares are in between 0 to 60.

Removing Missing Data

```
print(df_train.isnull().sum())
```

```
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    10
dropoff_latitude     10
passenger_count      0
dtype: int64
```

```
df_train = df_train.dropna(how = 'any',axis = 'rows')
```

```
len(df_train)
```

```
999952
```

Test Data

```
df_test = pd.read_csv('D:/Board Infinity/Project/New york taxi fare data/test.csv')
```

```
df_test.head()
```

	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24.0000002	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24.0000003	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44.0000002	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12.0000002	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12.0000003	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

As we can see there are total 20 null values in dropoff_longitude and dropoff_latitude which we need to remove from the dataset in order to achieve better accuracy.

And we can also see our test dataset which has approx.10 thousand rows.

Function for selecting the bounding box.

```
def select_within_boundingbox(df, BB):
    return (df.pickup_longitude >= BB[0]) & (df.pickup_longitude <= BB[1]) & \
           (df.pickup_latitude >= BB[2]) & (df.pickup_latitude <= BB[3]) & \
           (df.dropoff_longitude >= BB[0]) & (df.dropoff_longitude <= BB[1]) & \
           (df.dropoff_latitude >= BB[2]) & (df.dropoff_latitude <= BB[3])
```

Load Image of NYC Map for Visualization

```
BB = (-74.3,-73.0,40.6,41.7)
```

```
nyc_map = plt.imread('https://aiblog.nl/download/nyc_-74.5_-72.8_40.5_41.8.png')
```

```
df_train = df_train[select_within_boundingbox(df_train,BB)]
```

```
print("New size {}".format(len(df_train)))
```

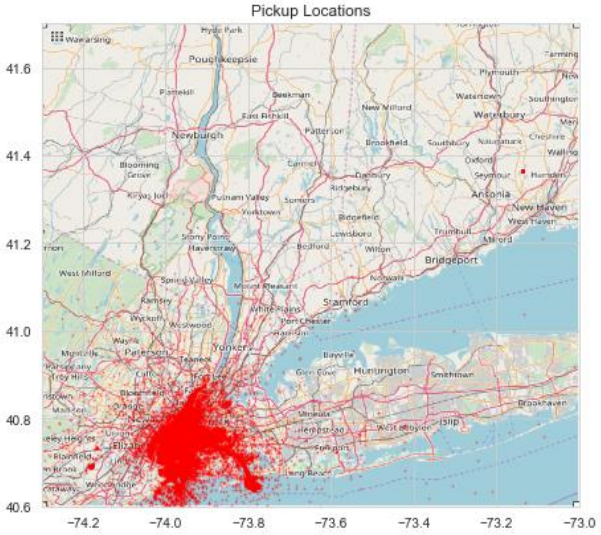
```
New size 978061
```

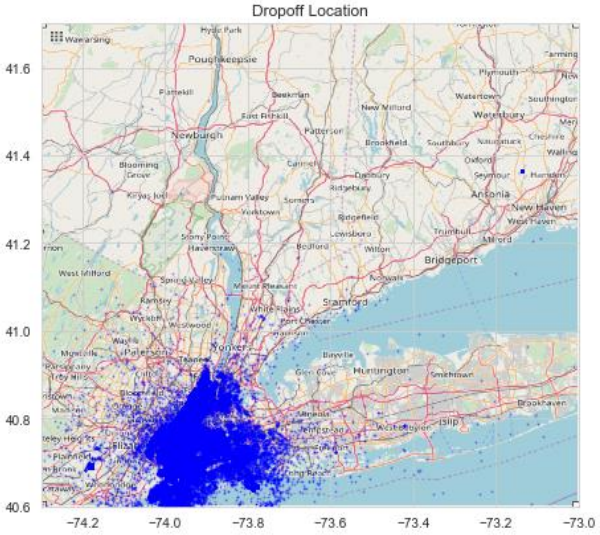
Function will be used for plotting data on NYC Map

```
def plot_on_map(df, BB, nyc_map, s = 10, alpha = 0.2):
    fig, axs = plt.subplots(1,2,figsize = (16, 10))
    axs[0].scatter(df["pickup_longitude"], df["pickup_latitude"], alpha = alpha , c = 'r', s=s)
    axs[0].set_xlim((BB[0], BB[1]))
    axs[0].set_ylim((BB[2],BB[3]))
    axs[0].set_title('Pickup Locations')
    axs[0].imshow(nyc_map,extent = BB)


    axs[1].scatter(df["dropoff_longitude"], df["dropoff_latitude"], alpha = alpha , c = 'b', s=s)
    axs[1].set_xlim((BB[0], BB[1]))
    axs[1].set_ylim((BB[2],BB[3]))
    axs[1].set_title('Dropoff Location')
    axs[1].imshow(nyc_map,extent = BB)
```

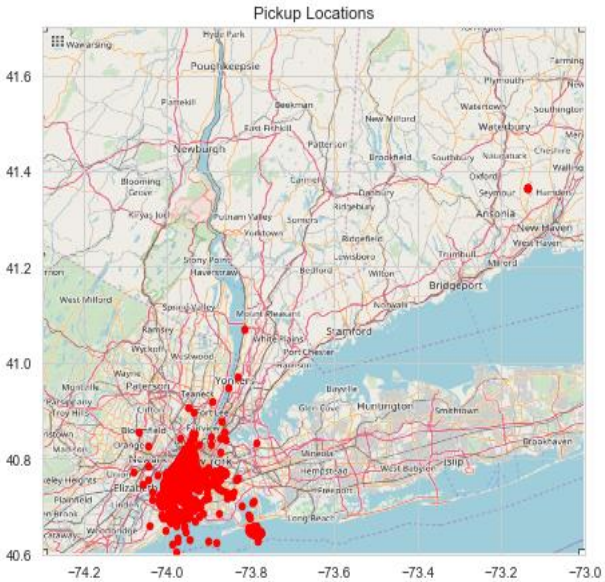
In the above image we can see the function which we used to define our bounding box and with bounding box function we will create a scatter plot to observe the distribution of rides in New York City map.

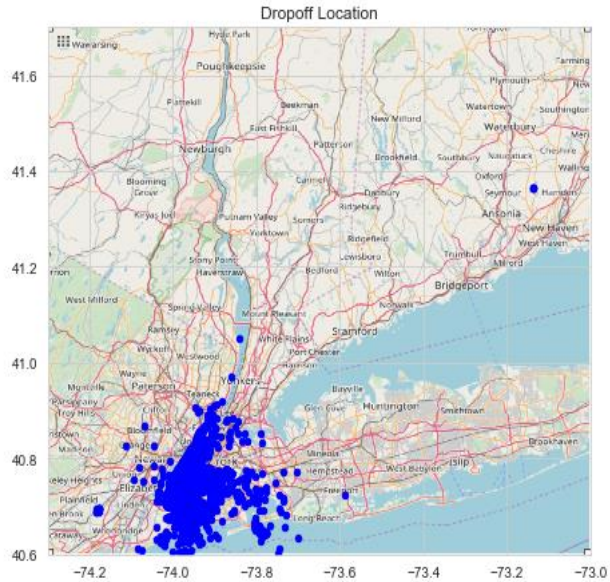
Plot on Training Data





Plot on Testing Data

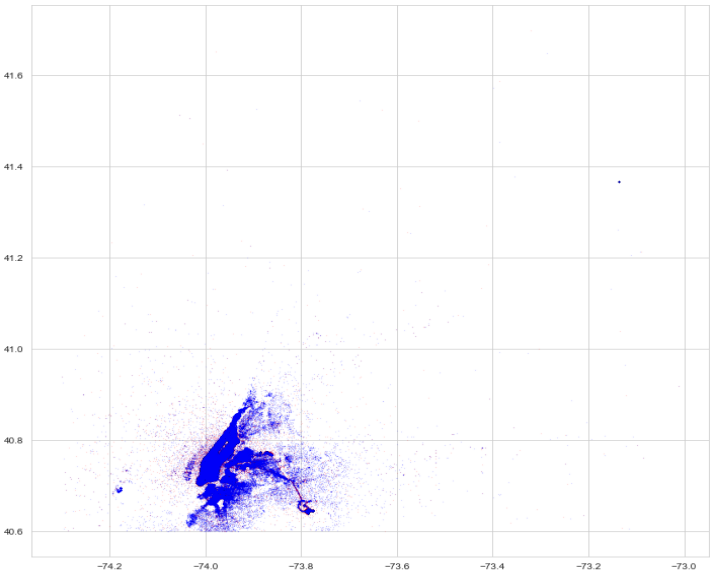




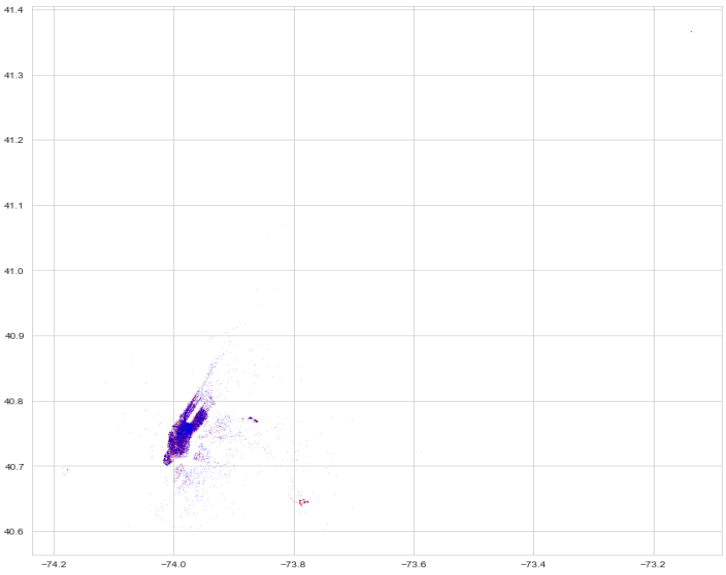
We can see on the training data some points are in Water region and some are far away on the right top side we can consider them as outliers and that's why we have used the bounding box.

The next plot will show us the distribution without the map so that we can have a clear look.

Plot on Training Data



Plot on Testing Data



```
# Add time information
df_train['year'] = df_train["pickup_datetime"].apply(lambda t:t.year)
df_train['weekday'] = df_train["pickup_datetime"].apply(lambda t:t.weekday())
df_train['hour'] = df_train["pickup_datetime"].apply(lambda t:t.hour)
```

Distance and Time Visualization

1.The longer the distance between pickup and dropoff locations, the higher the fare.

2.Some trips,like to/from an airport are fixed fee.

3.Fare at night is different from the day time.

Formula to be used for calculating the distance between latitude and longitude.

```
def distance(lat1, lon1, lat2, lon2):
    p = 0.017453292519943295  #Pi/180
    a = 0.5 - np.cos((lat2 - lat1)*p)/2 + np.cos(lat1 * p) * np.cos(lat2*p) * (1 - np.cos((lon2-lon1)*p))/2
    return 0.6213712 * 12742 * np.arcsin(np.sqrt(a))
```

The longer the distance between pickup and dropoff location,higher the fare.Adding new column to dataframe with distance in miles

```
df_train["distance_miles"] = distance(df_train["pickup_latitude"], df_train["pickup_longitude"],
                                      df_train["dropoff_latitude"], df_train["dropoff_longitude"])
```

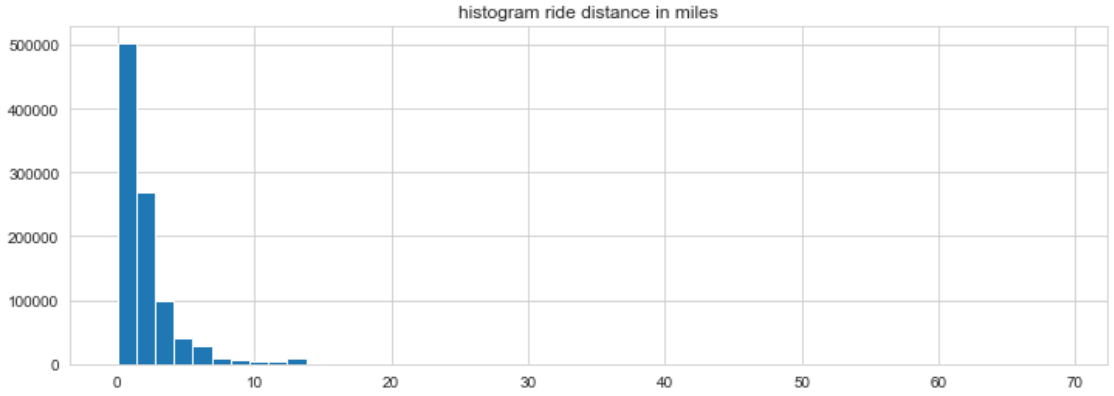
```
df_train.head(2)
```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year	weekday	hour
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.841610	40.712278	1	2009	0	17

In the above image we can see one of our feature engeniring part in which we have split the datetime stamp into year, weekday, hour and distance in miles.

And below that is our distance function which has used the haversine formula to calculate the distance.

After that we will plot a histogram of distance in miles to check that major rides cover how much distance.



By looking at the histogram we can see that the major rides are happening between 0 to 13 miles only. It seems most rides are just short rides, with a small peak at 13 miles. This peak could be due to airport drives.

```
df_train["distance_miles"].describe()
```

```
count     978061.000000
mean           2.061043
std            2.320323
min            0.000000
25%            0.779002
50%            1.336635
75%            2.430157
max           68.868482
Name: distance_miles, dtype: float64
```

- It seems most rides are just short rides, with a small peak at~13 miles.This peak could be due to airport drives.

```
df_train.groupby('passenger_count')['distance_miles','fare_amount'].mean()
```

passenger_count	distance_miles	fare_amount
0	1.711269	8.707828
1	2.034222	11.178548
2	2.168741	11.764392
3	2.086411	11.469143
4	2.117401	11.674912
5	2.059920	11.167977
6	2.134918	12.243306

Scatter Plot distance vs Fare

```
fig, axs = plt.subplots(1, 2, figsize =(16,6))
axs[0].scatter(df_train["distance_miles"],df_train["fare_amount"],alpha = 0.2)
axs[0].set_xlabel("distance mile")
```

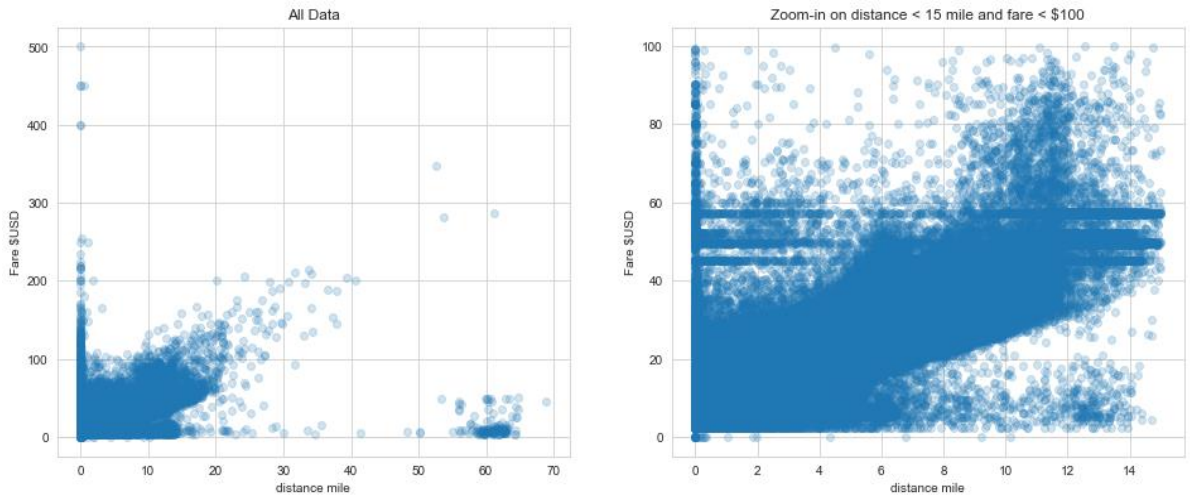
Looking at the distance_miles.describe (). We can see that min is 0 which is again not possible if you take a ride and travel 0 miles so we have to remove these points also.

Now if we look at the output of

df_train.groupby('passenger_count')['distance_miles','fare_amount'].mean()

We can observe that passenger count is 0 and there is certain fare amount charged for those rides also so we are going to put a filter for there kind of observations.

After that we are going to plot a scatter plot of distance in miles and fare amount to check what fare amount has been charged for what distance.



Now by looking at the All Data plot we can see there are mainy isntances where distance traveled is 0 and fare amount has been charged as high as $ 500 so we have to remove these from our model because we have a large taining data and a very small testing data and it

would be more accurate if it only focuses on that and also we don't have any other features like cancellation charges.

But when we look at the Zoom-in distance plot of only 15 miles and fare amount $ 100 we can see that maximum number of rides are there and between fare amount $40 to $60 the graph is showing constant relation.which can be because of airport rides.

Few Observations-

1.There are trips with zero distance but with a non-zero fare.could this be trips from and to the same location? Predicting these dare will be difficult as there is likely not sufficient information in the dataset.

2.There are some trips with >50 miles travel distance but low fare. Perhaps these are discounted trips.

3.The horizontal lines in the right plot might indicate again the fixed fare trips to/from JFK airport. Overall, there seems to be a (linear) relation between distance and fare with an average rate of +/- 100/20 = $5/mile.

```
#remove datapoints with distance <0.05 miles
idx = (df_train["distance_miles"] >= 0.05)
print('Old size: %d' % len(df_train))
df_train = df_train[idx]
print('New size: %d' % len(df_train))
```

```
Old size: 978061
New size: 962433
```

JFK airport coordinates - https://www.travelmath.com/airport/JFK

```
jfk = (-73.7822222222,40.6441666667)
nyc = (-74.0063889,40.7141667)
```

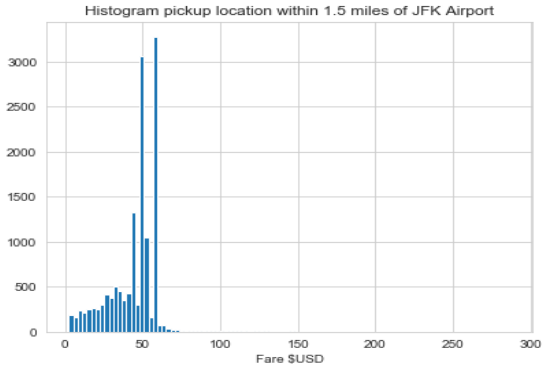
```
def plot_location_fare(loc, name, range = 1.5):
    #select all datapoints with dropoff location within range of airport
    fig, axs = plt.subplots(1,2, figsize = (14,5))
    idx = (distance(df_train["pickup_latitude"], df_train["pickup_longitude"],loc[1],loc[0])< range)
    df_train[idx]["fare_amount"].hist(bins = 100,ax = axs[0])
    axs[0].set_xlabel("Fare $USD")
    axs[0].set_title("Histogram pickup location within {} miles of {}".format(range,name))

    idx = (distance(df_train["dropoff_latitude"], df_train["dropoff_longitude"],loc[1],loc[0])< range)
    df_train[idx]["fare_amount"].hist(bins = 100,ax = axs[1])
    axs[1].set_xlabel("Fare $USD")
    axs[1].set_title("Histogram dropoff location within {} miles of {}".format(range,name))
```

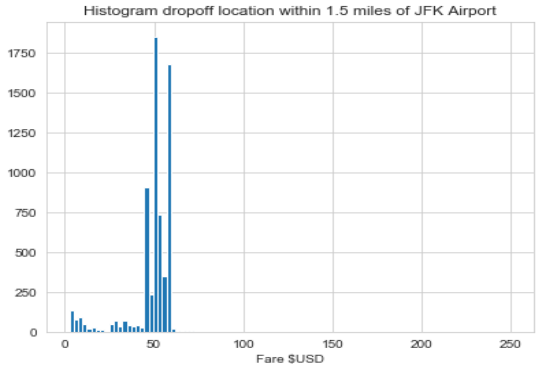
```
plot_location_fare(jfk,'JFK Airport')
```

In the above image we are removing data points with distance less than 0.05 miles

And we are going to compare the fares of some important locations like airports with their coordinates

JFK Airport Plot

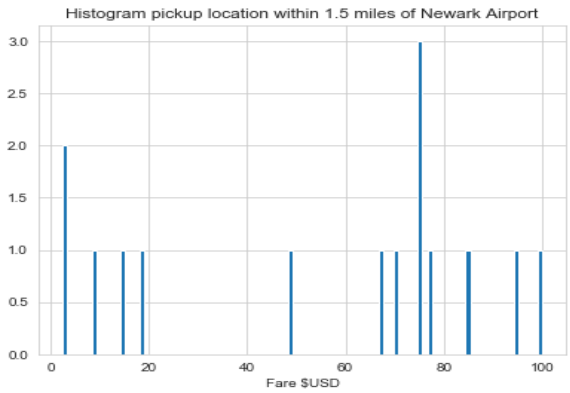


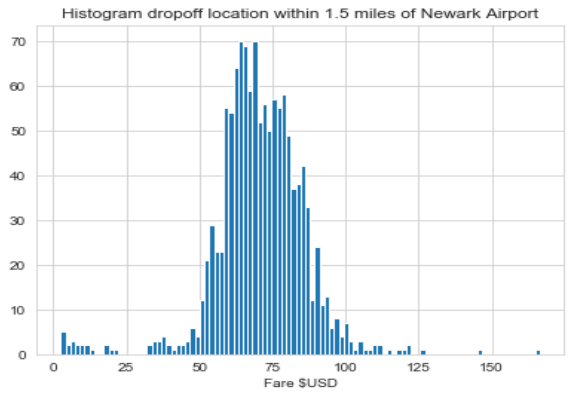


These graphs show the pickup and dropoff near JFK Airport as we can see certaing spikes between $ 50 to $ 60 this kind of certifys the assumption of a hotspot location for cab trips.

Some other famous locations

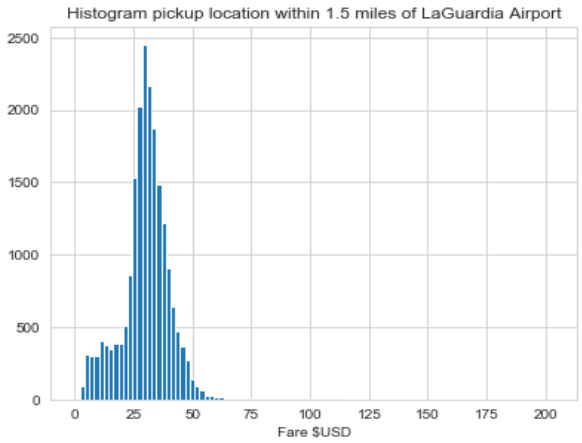
Newark Airport

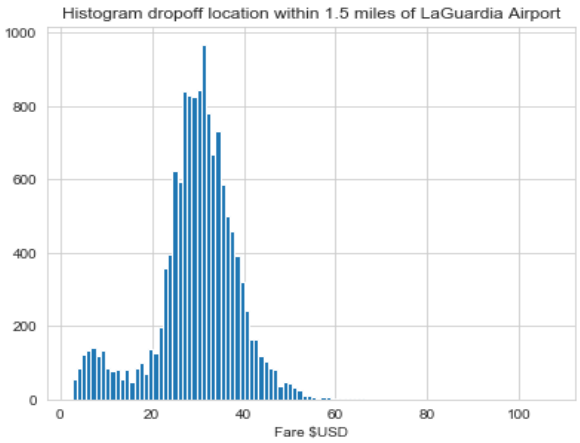




This graph shows that there are very less pickup at this airport but there are so many dropoffs at this airport.

LaGuardia Airport





This airport is like a domestic airport so there are many pickups and drop-offs mostly between $ 20 to $ 60.

Fare at night is different from the day time.

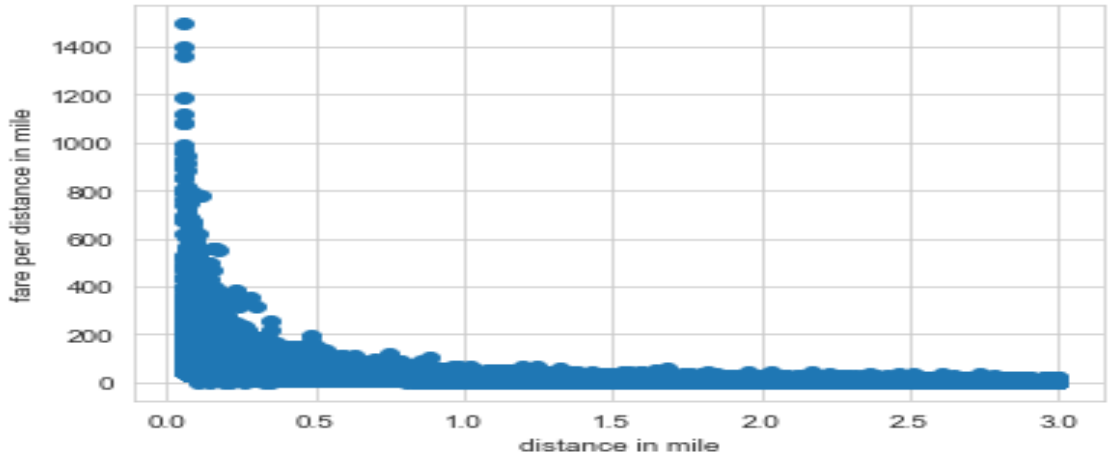
```
df_train["fare_per_mile"] = df_train["fare_amount"]/df_train["distance_miles"]
```

```
df_train["fare_per_mile"].describe()
```

```
count    962433.000000
mean          7.421546
std          12.721752
min           0.000000
25%           4.771449
50%           6.135104
75%           8.085859
max        3812.571628
Name: fare_per_mile, dtype: float64
```

```
idx = (df_train["distance_miles"]< 3) & (df_train["fare_amount"]<100)
plt.scatter(df_train[idx]["distance_miles"],df_train[idx]["fare_per_mile"])
plt.xlabel("distance in mile")
plt.ylabel("fare per distance in mile")
```

```
Text(0, 0.5, 'fare per distance in mile')
```



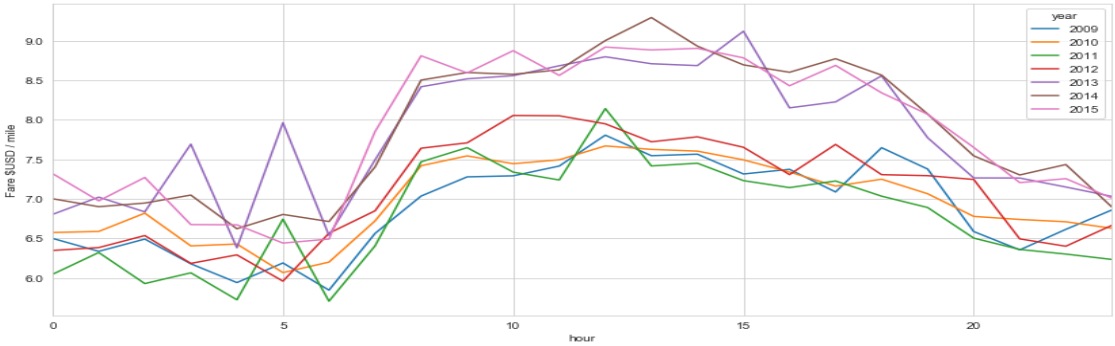
Here we tried to build another feature of fare per mile and tried to observe it with a graph but if we notice in shorter distance have significantly higher fare and as the distance increases the fare becomes constant but we cannot take this feature for our prediction as it is not available on our testing data.

Pivot Table

```
df_train.pivot_table("fare_per_mile",index = "hour", columns = "year")
```

year hour	2009	2010	2011	2012	2013	2014	2015
0	6.499397	6.574324	6.046723	6.347752	6.804020	7.001473	7.318776
1	6.336259	6.587778	6.321370	6.383975	7.021929	6.900875	6.974407
2	6.491376	6.819194	5.928391	6.535747	6.835403	6.946912	7.270579
3	6.178696	6.405068	6.064605	6.183963	7.692000	7.047893	6.674571
4	5.941006	6.428828	5.723230	6.291553	6.380988	6.621876	6.671974
5	6.189194	6.067733	6.742914	5.958988	7.963065	6.802803	6.440684
6	5.845517	6.199944	5.705709	6.565798	6.537331	6.713167	6.491548
7	6.562564	6.721096	6.405061	6.848710	7.492602	7.403542	7.850845
8	7.035261	7.418894	7.468816	7.640458	8.417914	8.502368	8.809942
9	7.278168	7.543578	7.647594	7.708457	8.517895	8.597979	8.593047
10	7.291380	7.444602	7.338167	8.055012	8.558028	8.575852	8.874432
11	7.414921	7.494226	7.238782	8.050909	8.683661	8.631433	8.561388
12	7.805850	7.669330	8.141671	7.948765	8.797059	9.001706	8.920133
13	7.545854	7.625559	7.416197	7.721714	8.708989	9.293393	8.883700
14	7.565213	7.603563	7.450513	7.784844	8.685488	8.931433	8.903522
15	7.314500	7.493269	7.230718	7.652271	9.120416	8.694742	8.783262
16	7.373671	7.339379	7.142134	7.306803	8.151143	8.601068	8.430501
17	7.087076	7.161012	7.226114	7.688501	8.226106	8.772463	8.688070
18	7.647091	7.248898	7.034009	7.306104	8.556521	8.564641	8.340938
19	7.376881	7.064368	6.890245	7.293315	7.772822	8.072091	8.070096
20	6.586457	6.778068	6.503484	7.245539	7.264707	7.544221	7.650078

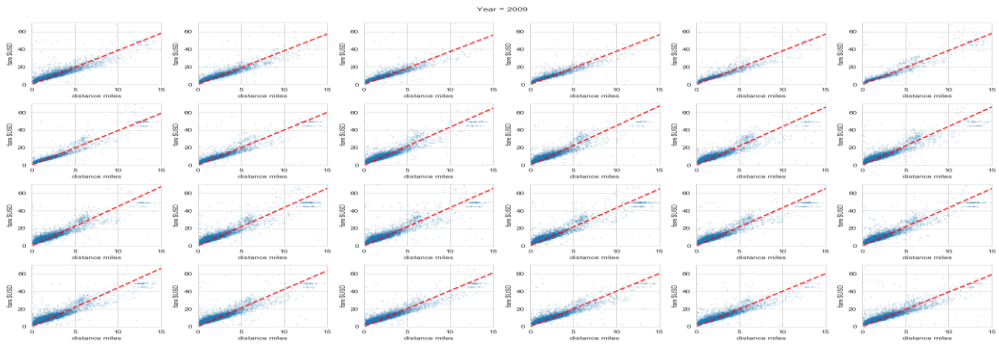
Next, we tried to create a pivot table with index as hour and fare amount of all the years then we will create a line plot of all the years and fare_per_mile for 24 hours.

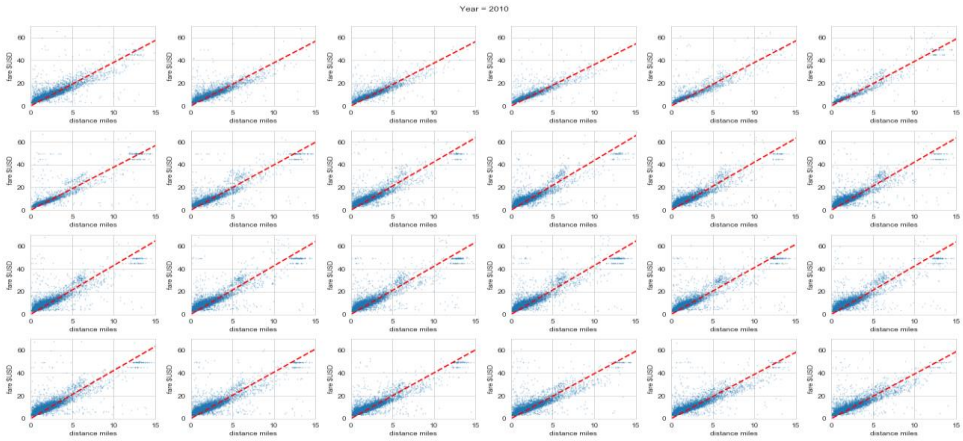


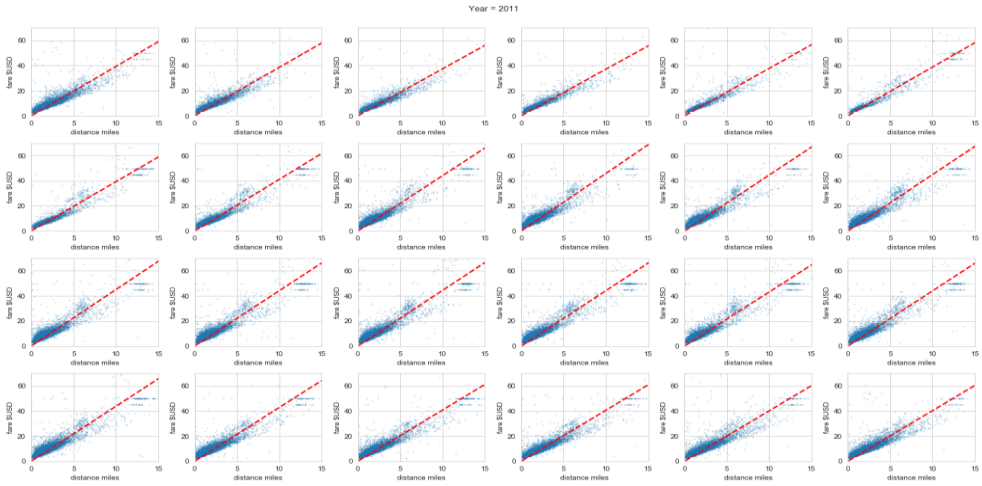
Now we can see that that there is a peak between 9-12 and 15-18 hours which justifies the office peak hours. This gives us the idea to put hour and year as feature to calculate the Fare amount.

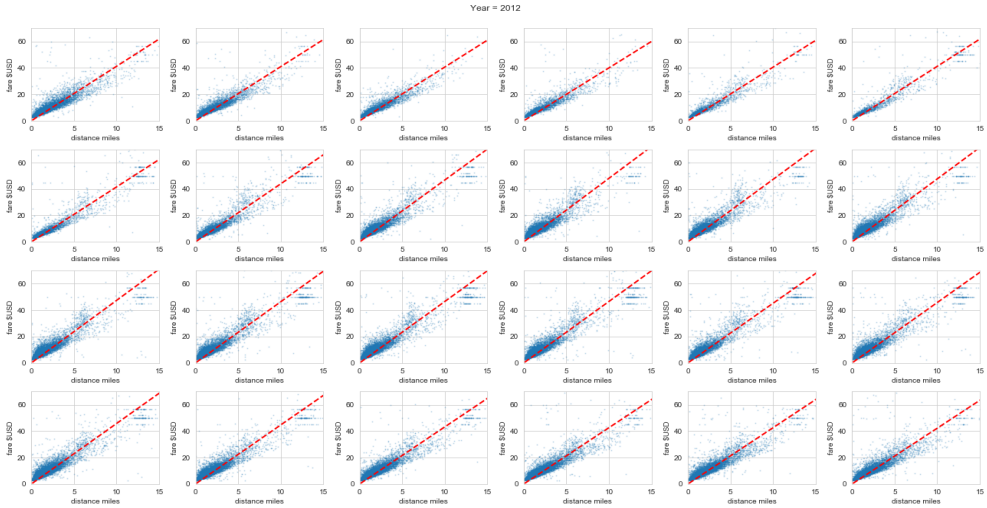
```
# plot all years
for year in df_train["year"].unique():

    # create figure
    fig, axs = plt.subplots(4, 6, figsize=(18, 10))
    axs = axs.ravel()

    # plot for all hours

    for h in range(24):
        idx = (df_train["distance_miles"] < 15) & (df_train["fare_amount"] < 100) & (df_train["hour"] == h) & \
              (df_train["year"] == year)
        axs[h].scatter(df_train[idx]["distance_miles"], df_train[idx]["fare_amount"], alpha=0.2, s=1)
        axs[h].set_xlabel('distance miles')
        axs[h].set_ylabel('fare $USD')
        axs[h].set_xlim((0, 15))
        axs[h].set_ylim((0, 80))

        model = LinearRegression(fit_intercept=False)

        X, y = df_train[idx]["distance_miles"].values.reshape(-1,1), df_train[idx]["fare_amount"].values
        model.fit(X, y)
        xx = np.linspace(0.1, 25, 100)
        axs[h].plot(xx, model.predict(xx.reshape(-1,1)), '--', c='r', lw=2)

    plt.suptitle("Year = {}".format(year))
    plt.tight_layout(rect=[0, 0, 1, 0.95]);
```

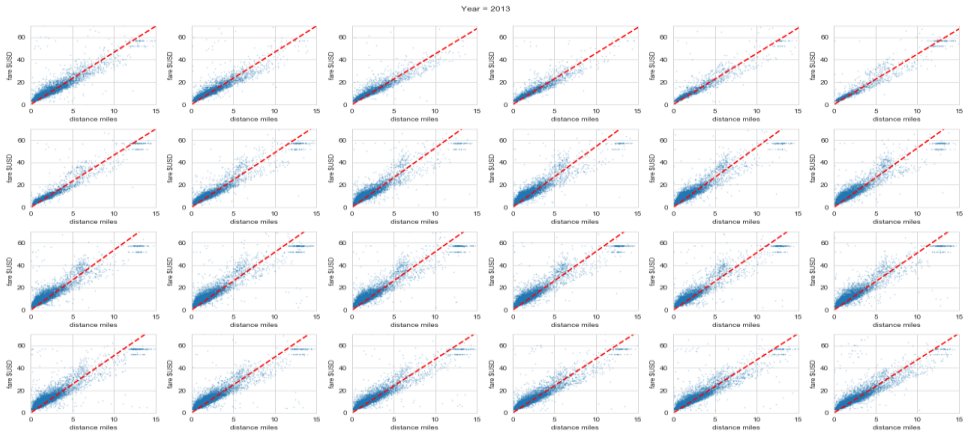
Next, we will create a regression plot of hour, year and fare amount for all the years and 24 hours

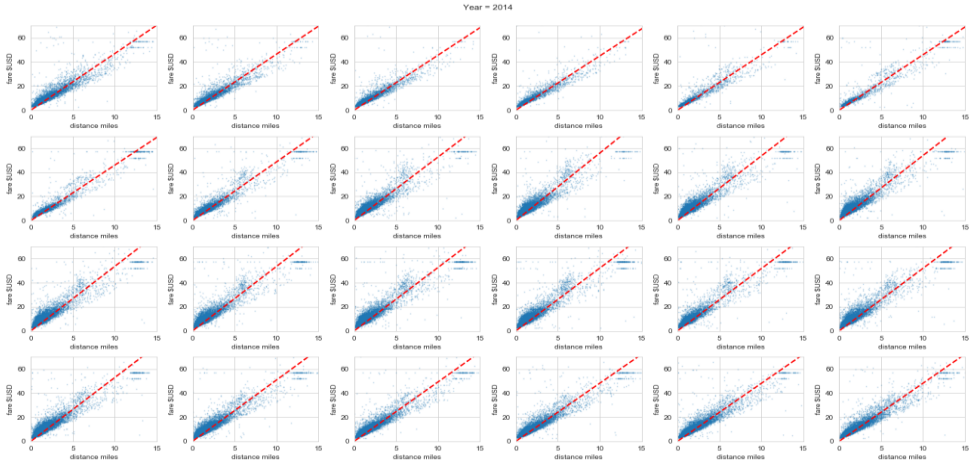


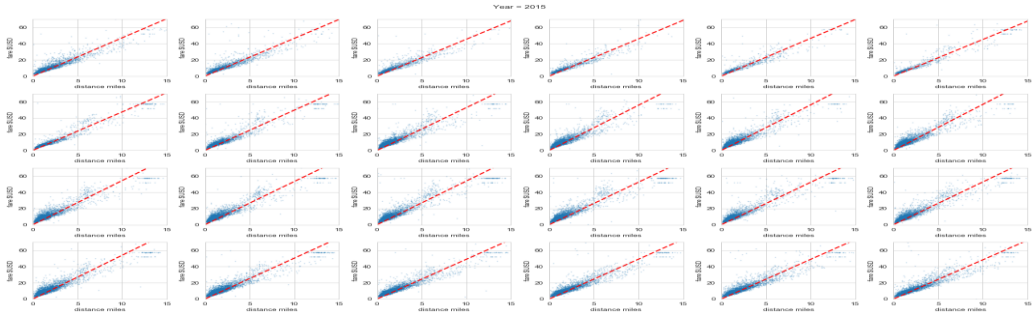

Year = 2010
fare $USD
distance miles


Year = 2011
fare $USD
distance miles


Year = 2012
fare $USD
distance miles







These regression plots show us increasing trend over the years so by this we are sure that taking hour and year for prediction will be very effective for this model.

Relevance of direction for calculation of fare amount

```
df_train["delta_lon"] = df_train["pickup_longitude"] - df_train["dropoff_longitude"]
df_train["delta_lat"] = df_train["pickup_latitude"] - df_train["dropoff_latitude"]
```

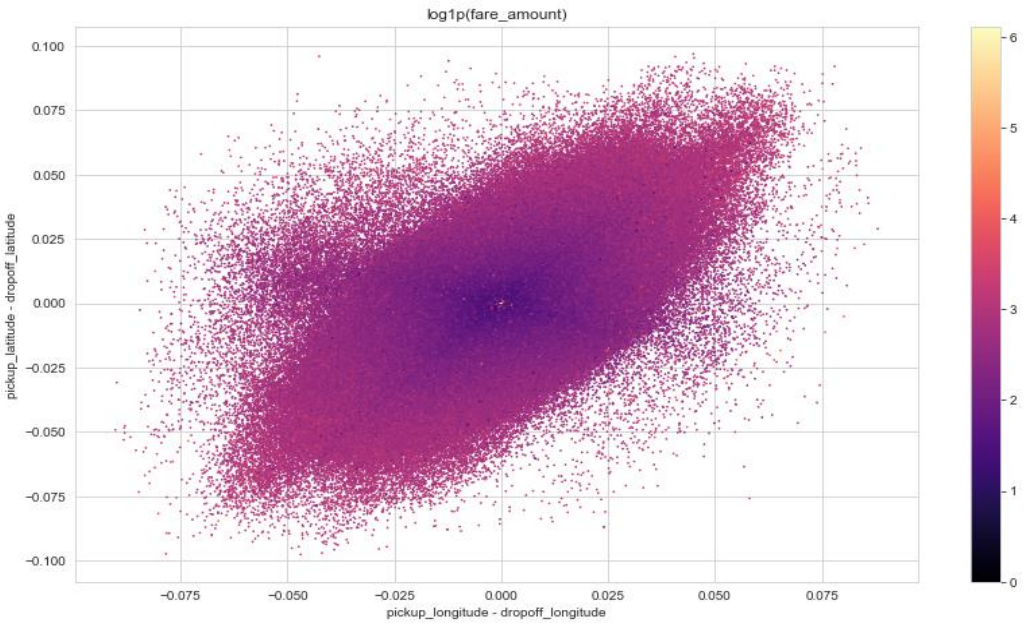
Trips in Manhattan

```
BB_manhattan = (-74.025,-73.925,40.7,40.8)
idx_manhattan = select_within_boundingbox(df_train,BB_manhattan)
```

```
plt.figure(figsize= (14,8))
plt.scatter(df_train[idx_manhattan]["delta_lon"],df_train[idx_manhattan]["delta_lat"],s = 0.5, alpha = 1.0,
           c=np.log1p(df_train[idx_manhattan]["fare_amount"]),
           cmap = "magma")
plt.colorbar()
plt.xlabel('pickup_longitude - dropoff_longitude')
plt.ylabel('pickup_latitude - dropoff_latitude')
plt.title('log1p(fare_amount)')
```

```
Text(0.5, 1.0, 'log1p(fare_amount)')
```

One of our feature engineering that is getting the direction so for observing direction we have taken Manhattan coordinates and put it in our bounding box function and tried to proof that direction matters by creating a scatter plot.



The centre of the plot is 0 as we can see if we are moving in particular direction the colour gradient is getting varied so this means if we move from any direction the price changes so we are going to apply haversine formula to get direction as our feature

The above plot shows that direction seems to matter.Direction of a trip,from 180 to -180 degrees.Horizontal axes = 0 degrees

```
def calculate_direction(d_lon, d_lat):
    result = np.zeros(len(d_lon))
    l = np.sqrt(d_lon**2 + d_lat**2)
    result[d_lon>0] = (180/np.pi)*np.arcsin(d_lat[d_lon>0]/l[d_lon>0])
    idx = (d_lon<0) & (d_lat>0)
    result[idx] = 180 - (180/np.pi)*np.arcsin(d_lat[idx]/l[idx])
    idx = (d_lon<0) & (d_lat<0)
    result[idx] = -180 - (180/np.pi)*np.arcsin(d_lat[idx]/l[idx])
    return result
```

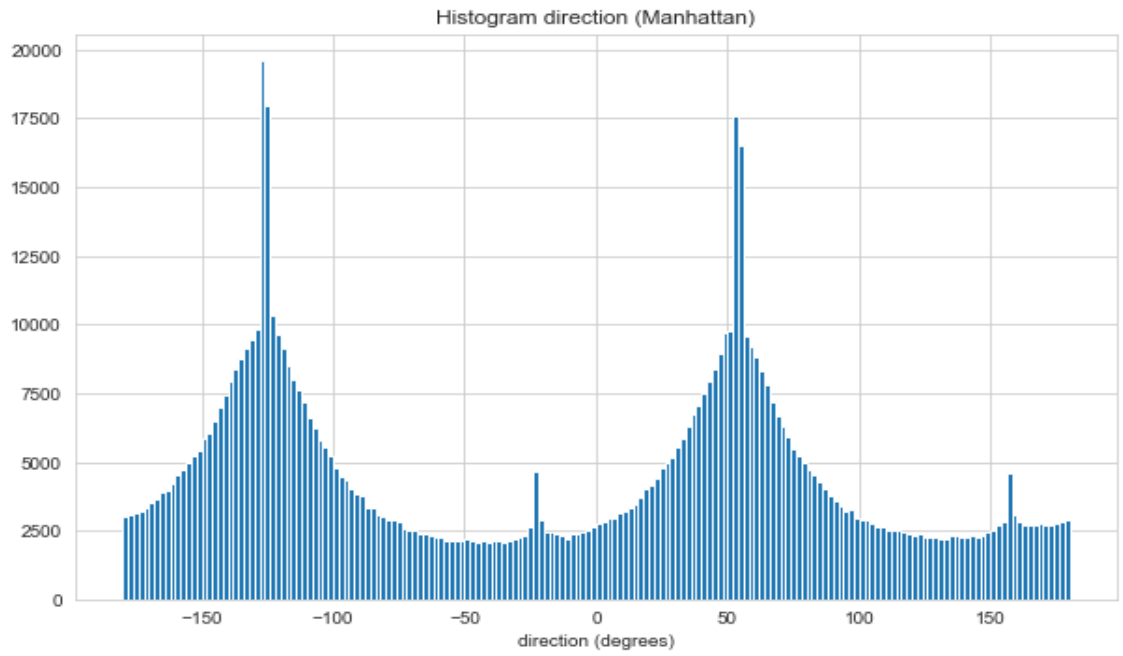
```
df_train['direction'] = calculate_direction(df_train.delta_lon,df_train.delta_lat)
```

```
df_train.head(1)
```

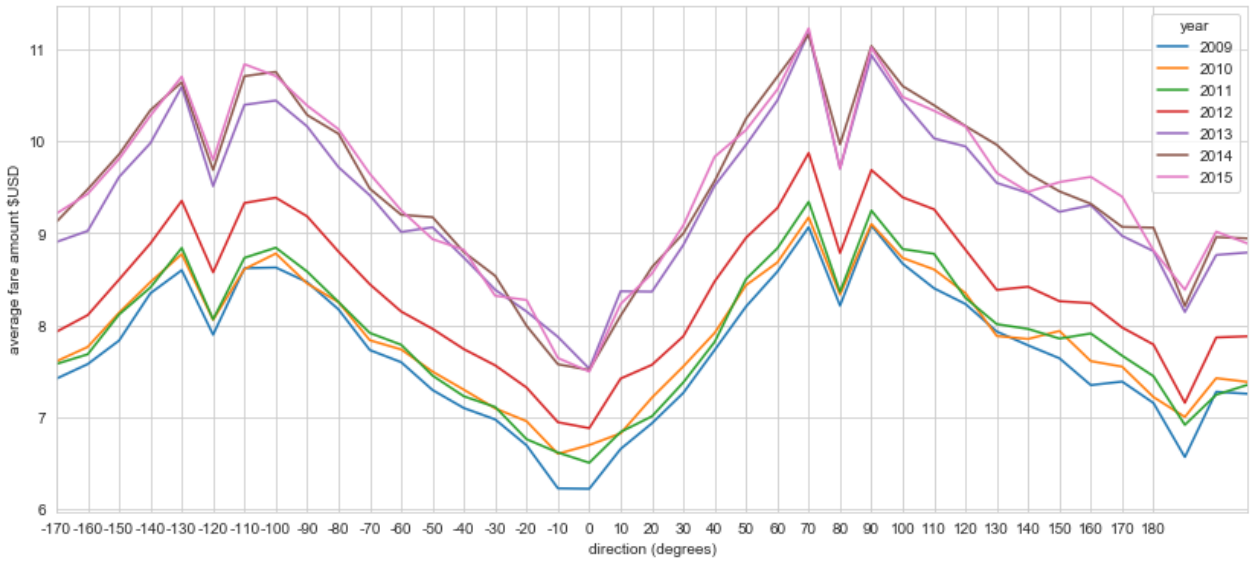
	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year	weekday	hour	d
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.84161	40.712278	1	2009	0	17	

```
# plot histogram of directions
plt.figure(figsize=(10,6))
df_train[idx_manhattan].direction.hist(bins=180)
plt.xlabel('direction (degrees)')
plt.title('Histogram direction (Manhattan)')

# plot direction vs average fare amount
fig, ax = plt.subplots(1, 1, figsize=(14,6))
direc = pd.cut(df_train[idx_manhattan]['direction'], np.linspace(-180, 180, 40))

df_train[idx_manhattan].pivot_table('fare_amount', index=[direc], columns='year', aggfunc='mean').plot(ax=ax)

plt.xlabel('direction (degrees)')
plt.xticks(range(36), np.arange(-170, 190, 10))
plt.ylabel('average fare amount $USD');
```

Now we are going to create a direction histogram with respect to fare amount.



This shows that if we move in a particular direction the number of observations increase now we will compare it with the fare amount.



Now if we see at the line graph it shows that moving to a particular direction increases fare amount so this proves that direction can also be taken in feature selection of the final model.

jupyter Cab Fare Prediction Model[Shashank Pandey] Last Checkpoint: Last Monday at 8:33 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

In [67]:
```
# Add a new column to df_train with distance in mile
df_train['distance_to_center'] = distance(nyc[1],nyc[0],df_train["pickup_latitude"], df_train["pickup_longitude"])
```

In [68]:
```
df_train.head()
```

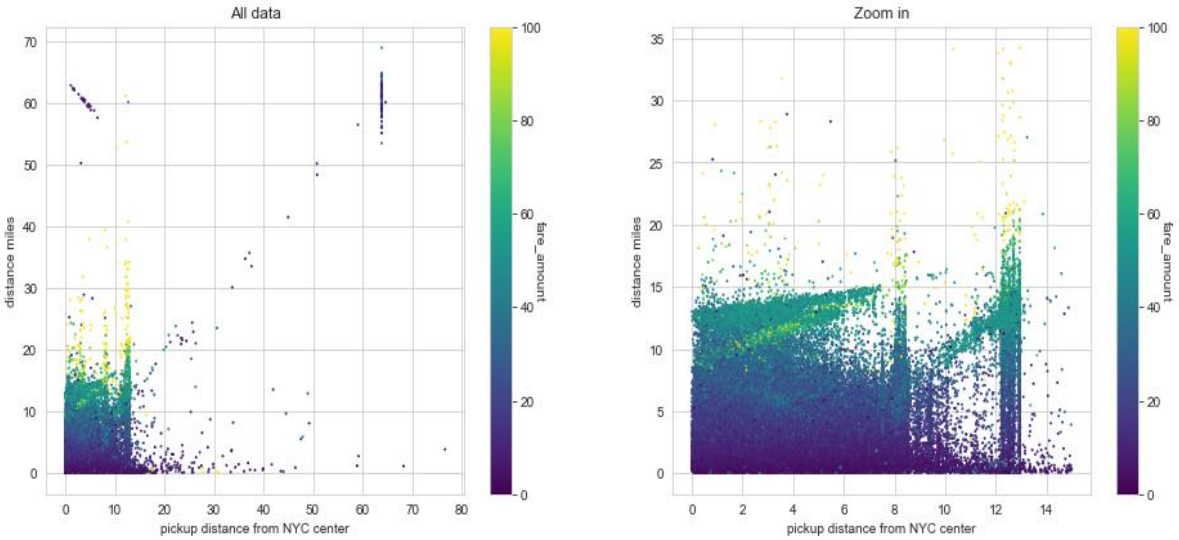
Out[68]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year	weekday	hour
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.841610	40.712278	1	2009	0	17
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.979268	40.782004	1	2010	1	16
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.991242	40.750562	2	2011	3	0
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.991567	40.758092	1	2012	5	4
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.956655	40.783762	1	2010	1	7

In [69]:
```
fig, axs = plt.subplots(1, 2, figsize=(16,6))
im = axs[0].scatter(df_train["distance_to_center"], df_train["distance_miles"], c=np.clip(df_train["fare_amount"], 0, 1
                    cmap='viridis', alpha=1.0, s=1)

axs[0].set_xlabel('pickup distance from NYC center')
axs[0].set_ylabel('distance miles')
axs[0].set_title('All data')

cbar = fig.colorbar(im, ax=axs[0])
cbar.ax.set_ylabel('fare_amount', rotation=270)

#zoom-in plot
idx = (df_train["distance_to_center"] < 15) & (df_train["distance_miles"] < 35)
im = axs[1].scatter(df_train[idx]["distance_to_center"], df_train[idx]["distance_miles"],
                    c=np.clip(df_train[idx]["fare_amount"], 0, 100), cmap='viridis', alpha=1.0, s=1)
axs[1].set_xlabel('pickup distance from NYC center')
axs[1].set_ylabel('distance miles')
axs[1].set_title('Zoom in')
cbar = fig.colorbar(im, ax=axs[1])
cbar.ax.set_ylabel('fare_amount', rotation=270);
```

All data Zoom in

Last part of our feature engineering is our distance to centre which will help us identify the fares from centre of New York City to other places.



The above plot shows us distance from NYC center, distance miles and price as we can see in the zoom-in plot. A lot of 'green' dots, which is about 50 to 60 fare amount near 13 miles distance of NYC center of distance of trip. This could be due to trips from/to JFK airport.

With this our feature engineering part is done now we will implement it in a regression model.

Baseline Model and Submission.

```
# add new column to dataframe with distance in km
df_test['distance_miles'] = distance(df_test["pickup_latitude"], df_test["pickup_longitude"],
                                     df_test["dropoff_latitude"], df_test["dropoff_longitude"])
df_test['distance_to_center'] = distance(nyc[1], nyc[0], df_test["dropoff_latitude"], df_test["dropoff_longitude"])
df_test['hour'] = df_test["pickup_datetime"].apply(lambda t: pd.to_datetime(t).hour)
df_test['year'] = df_test["pickup_datetime"].apply(lambda t: pd.to_datetime(t).year)
df_test['weekday'] = df_test["pickup_datetime"].apply(lambda t: pd.to_datetime(t).weekday())
```

```
df_train.head()
```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year	weekday	hour
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.841610	40.712278	1	2009	0	17
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.979268	40.782004	1	2010	1	16
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.991242	40.750562	2	2011	3	0
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.991567	40.758092	1	2012	5	4
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.956655	40.783762	1	2010	1	7

```
len(df_train)
```

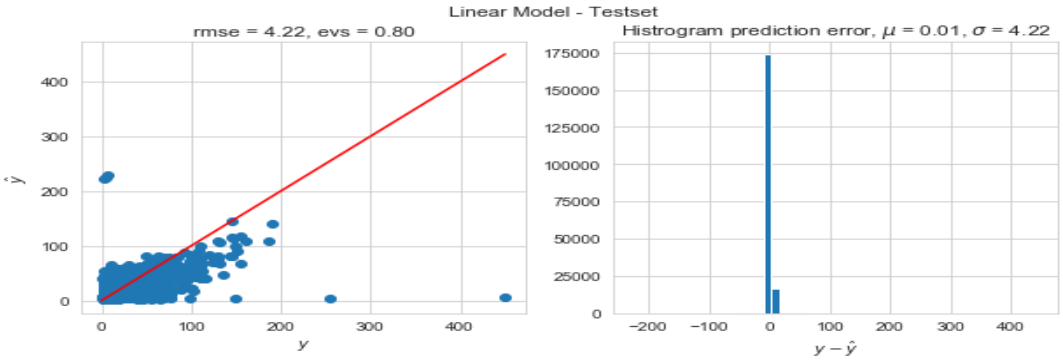
```
962433
```

```python
# define some handy analysis support function
def plot_prediction_analysis(y, y_pred, figsize=(10,4), title=''):
    fig, axs = plt.subplots(1, 2, figsize=figsize)
    axs[0].scatter(y, y_pred)
    mn = min(np.min(y), np.min(y_pred))
    mx = max(np.max(y), np.max(y_pred))
    axs[0].plot([mn, mx], [mn, mx], c='red')
    axs[0].set_xlabel('$y$')
    axs[0].set_ylabel('$\hat{y}$')
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    evs = explained_variance_score(y, y_pred)
    axs[0].set_title('rmse = {:.2f}, evs = {:.2f}'.format(rmse, evs))

    axs[1].hist(y-y_pred, bins=50)
    avg = np.mean(y-y_pred)
    std = np.std(y-y_pred)
    axs[1].set_xlabel('$y - \hat{y}$')
    axs[1].set_title('Histrogram prediction error, $\mu$ = {:.2f}, $\sigma$ = {:.2f}'.format(avg, std))

    if title!='':
        fig.suptitle(title)
```

```python
# create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 0)
```

```python
model_lin = Pipeline((
        ("standard_scaler", StandardScaler()),
        ("lin_reg", LinearRegression()),
    ))

model_lin.fit(X_train, y_train)


y_train_pred = model_lin.predict(X_train)

plot_prediction_analysis(y_train, y_train_pred, title='Linear Model - Trainingset')

y_test_pred = model_lin.predict(X_test)
plot_prediction_analysis(y_test, y_test_pred, title='Linear Model - Testset')
```

As we can 20% of our data set is for testing and the remaining 80% is for training And then we have created a pipeline and put standard scaler to bring each and every feature in to one scale and then linear regression for our model and then we are trying to plot a prediction graph.



As we can see the prediction error is very less in the test set linear model.

```
In [123]: sklearn.metrics.r2_score(y_test, y_test_pred)

Out[123]: 0.7991367652412371

In [ ]:
```

Performance analysis

The model got an accuracy of almost 80 % which can be considered as a good model. Surely, we can use some other models like xg boosting or random forest to improve the accuracy but this model will fairly work.

Conclusion and Future Work

We attempted to use Multiple Linear Regression model for this task, and the result was not optimal, but it was acceptable. We did a lot of EDA for better feature selection.

In the future, I plan to work on extraction of additional features and test other parameters for the same dataset in deep learning. I believe that deep learning will produce good results if more features and data are extracted.

References

1. https://medium.com/analytics-vidhya/machine-learning-to-predict-taxi-fare-part-one-exploratory-analysis-6b7e6b1fbc78
2. https://www.irjet.net/archives/V6/i3/IRJET-V6I3768.pdf
3. https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/
4. https://www.youtube.com/watch?v=BmGJGEwOUrU&list=LL&index=2&t=2094s
5. https://www.youtube.com/watch?v=Q73ADVZCqSU&list=LL&index=1&t=37s
6. https://www.researchgate.net/publication/324706525_Taxi_Fare_Rate_Classification_Using_Deep_Networks
7. https://pythonhosted.org/planar/bbox.html
8. https://www.reneshbedre.com/blog/multiple-linear-regression.html
9. https://www.investopedia.com/terms/m/mlr.asp