Chapter 1: Introduction

1.1 Problem Statement

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

1.2 Data

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.500	2009-06-15 17:26:21	-73.844	40.721	-73.842	40.712	1
1	16.900	2010-01-05 16:52:16	-74.016	40.711	-73.979	40.782	1
2	5.700	2011-08-18 00:35:00	-73.983	40.761	-73.991	40.751	2
3	7.700	2012-04-21 04:30:42	-73.987	40.733	-73.992	40.758	1
4	5.300	2010-03-09 07:51:00	-73.968	40.768	-73.957	40.784	1

Table 1.1 Cab Fare Sample Data

As you can see in the table below we have the following 7 variables, using which we have to correctly predict the fare of cab:

Sl.No	Variables
1	Pickup_datetime
2	pickup_longitude
3	pickup_latitude
4	dropoff_longitude
5	dropoff_latitude
6	passenger_count

Table 1.2: Predictor variables

In [14]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	15988.000000	15988.000000	15988.000000	15988.000000	15988.000000	15988.000000
mean	15.056187	-72.464541	39.915681	-72.464099	39.898780	2.623030
std	431.199461	10.572946	6.828608	10.569608	6.185996	60.888226
min	-3.000000	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	6.000000	-73.992143	40.734931	-73.991182	40.734655	1.000000
50%	8.500000	-73.981691	40.752603	-73.980168	40.753560	1.000000
75%	12.500000	-73.966822	40.767356	-73.963645	40.768006	2.000000
max	54343.000000	40.766125	401.083332	40.802437	41.366138	5345.000000

When we look at statistic summary, we have several discoveries:

- The minimum fare amount is negative.
- Minimum and Maximum longitude and latitude look unreal.
- Minimum passenger count is 0.

We are going to fix them.

- New York city longitudes are around -74 and latitudes are around 41.
- Remove 0 passenger count.
- The taxi fare initial charge is $2.5, so we are removing fare amount smaller than this amount.

Then we check statistical summary again.

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	15590.000000	15590.000000	15590.000000	15590.000000	15590.000000	15590.000000
mean	15.153325	-73.974823	40.750913	-73.973851	40.751408	2.636369
std	436.665087	0.041530	0.038016	0.039371	0.039676	61.630264
min	2.500000	-74.438233	39.603178	-74.429332	39.604972	1.000000
25%	6.000000	-73.992374	40.736588	-73.991371	40.736332	1.000000
50%	8.500000	-73.982053	40.753333	-73.980563	40.754232	1.000000
75%	12.500000	-73.968104	40.767803	-73.965441	40.768314	2.000000
max	54343.000000	-73.137393	41.366138	-73.137393	41.366138	5345.000000

Chapter 2: Methodology

2.1 Pre-Processing

A predictive model requires that we look at the data before we start to create a model. However, in data mining, looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is known as Exploratory Data Analysis.

Hypothesis Generation

The next step to solve any analytics problems is to list down a set of hypothesis, which in our case are factors that will affect the cost of a taxi trip.

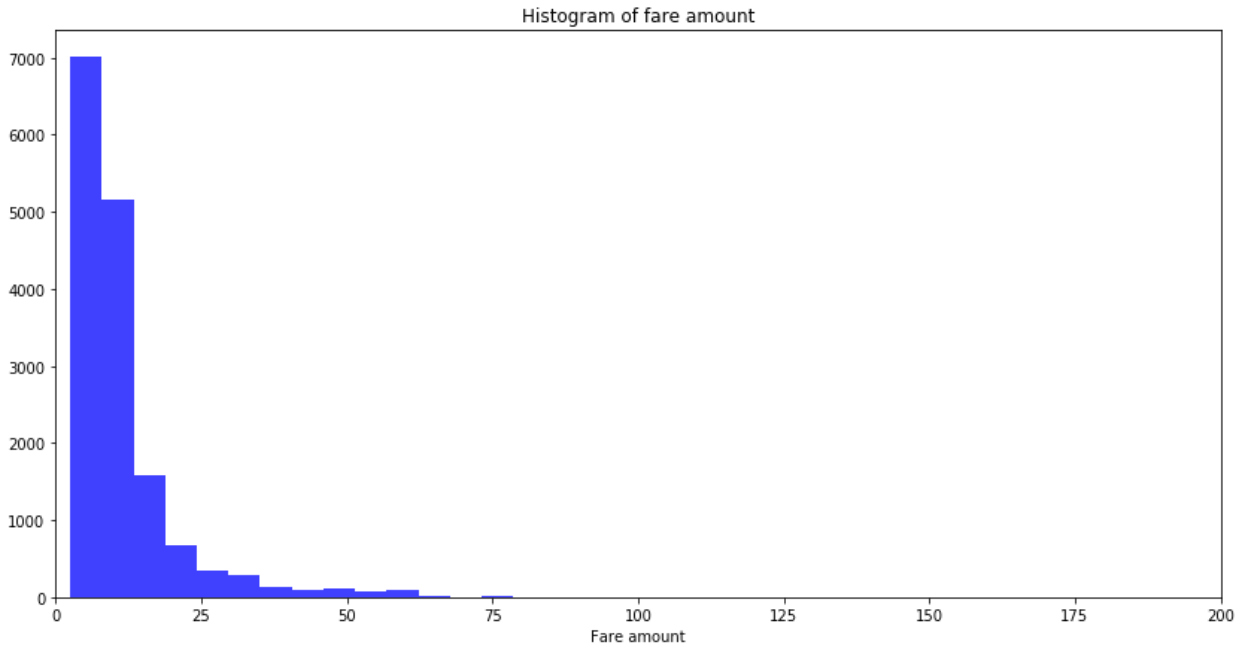
1. **Trip distance** : If the distance to be traveled is more, then fare should be higher.
2. **Time of Travel** : During peak traffic hours, the taxi fare may be higher.
3. **Day of Travel** : Fare amount may differ on weekday and weekends
4. **Is it a trip to/from airport** : Trips to/from airport generally have a fixed fare.

Data Cleaning and Exploration

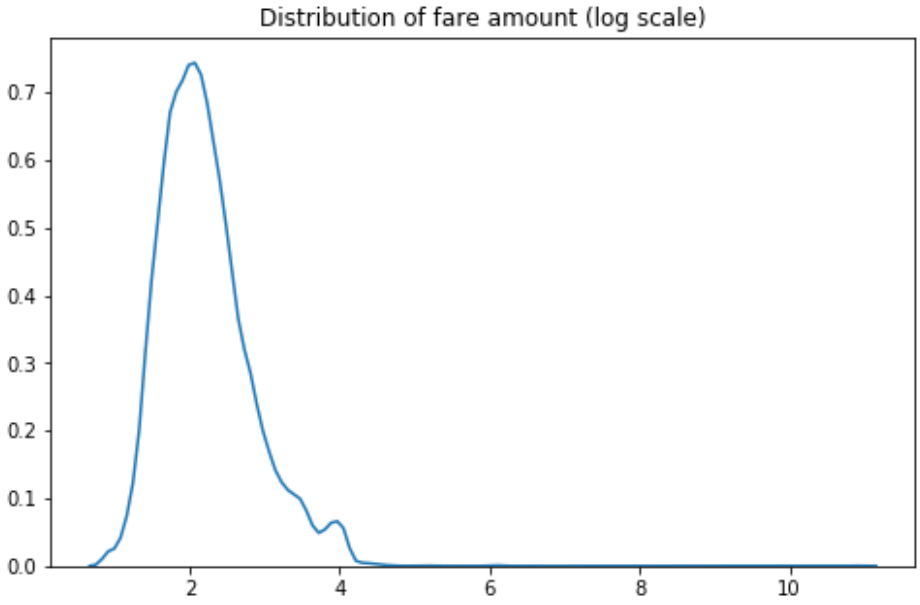
In this section, we will discuss various steps used to clean the data and understand the relationship between variables and use this understanding to create better features.

1. **Distribution of fare amount**

We first looked at the distribution of fare amount and found that there were some records where the fare was negative. Since, cost of a trip cannot be negative we removed such instances from the data.After removing the values we have plot the histogram of Fare Amount and found most of the fare is between 2.5 to 75 .

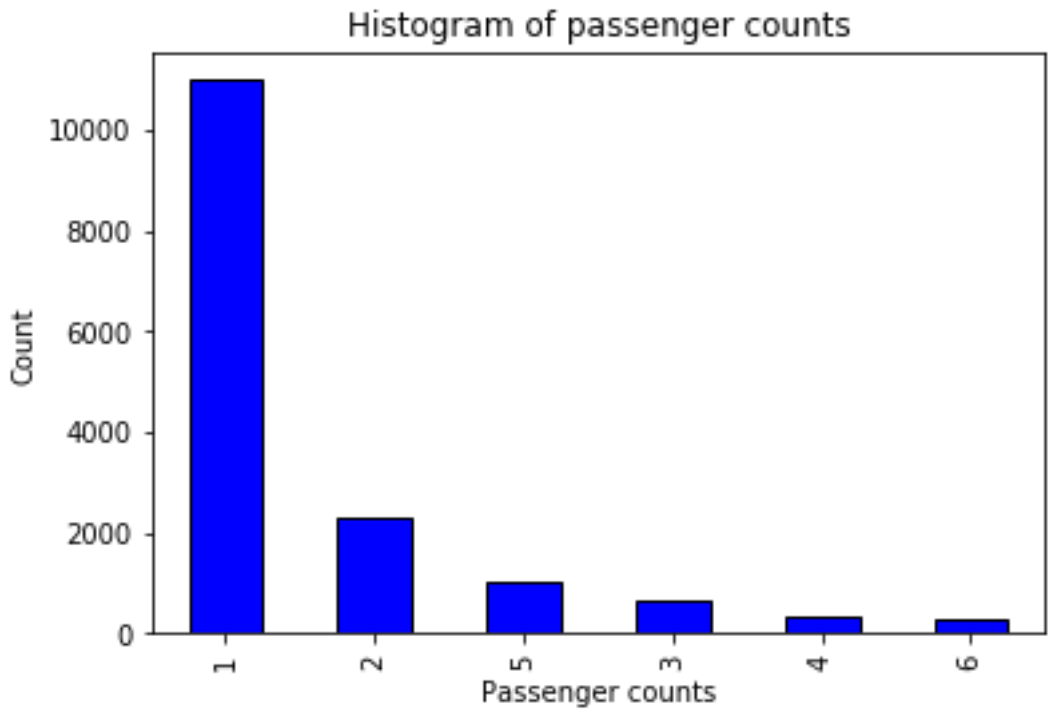


Also, fare amount follows long tail distribution.To understand the distribution of fare amount better we take a log transformation after removing the negative fares- this makes the distribution close to normal. Since we saw above that fare amount is highly skewed,let us take log transformation of the fare amount and plot the distribution.



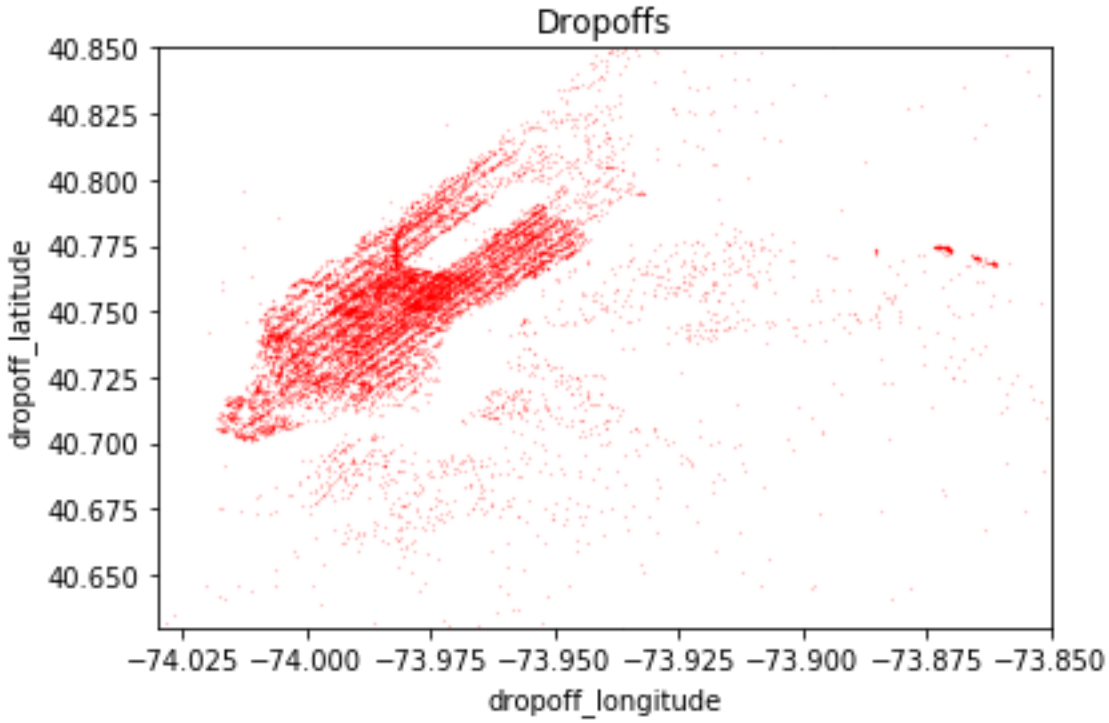
2 Distribution of Passenger Count

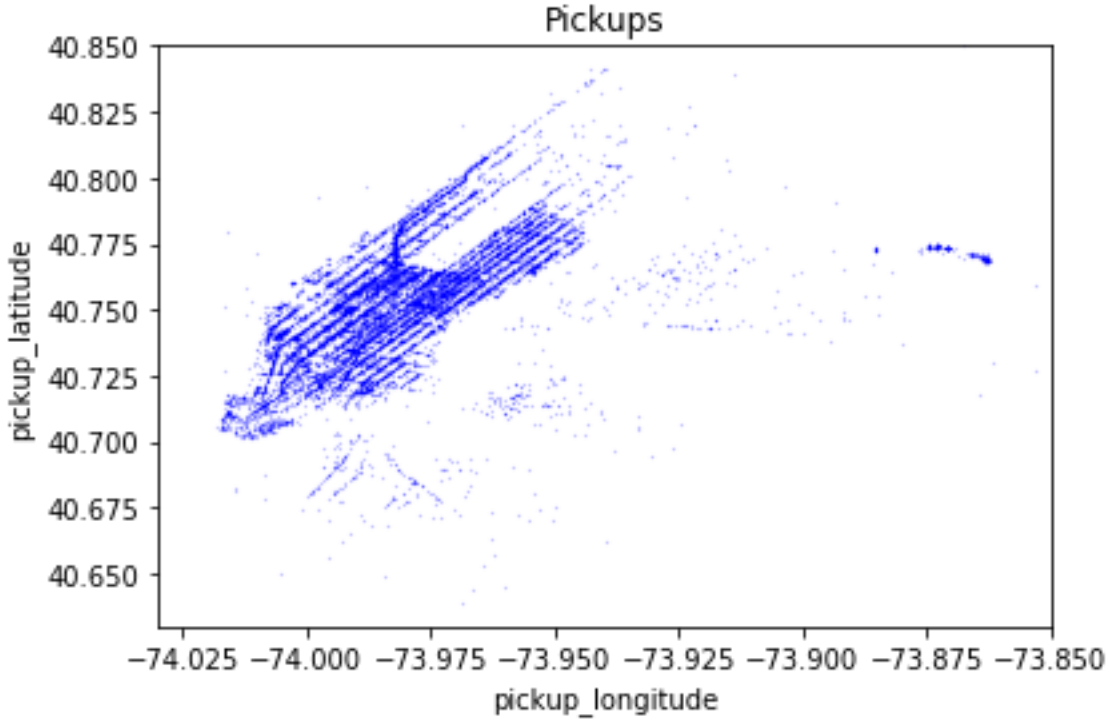
We have seen from the data that we have passenger count more then 6 also which is not possible so we have set the max passenger count to 6 and plot the histogram for passenger count and found we hve maximum passenger as 1 then 2 and so on.



3. **Distribution of Geographical Features**

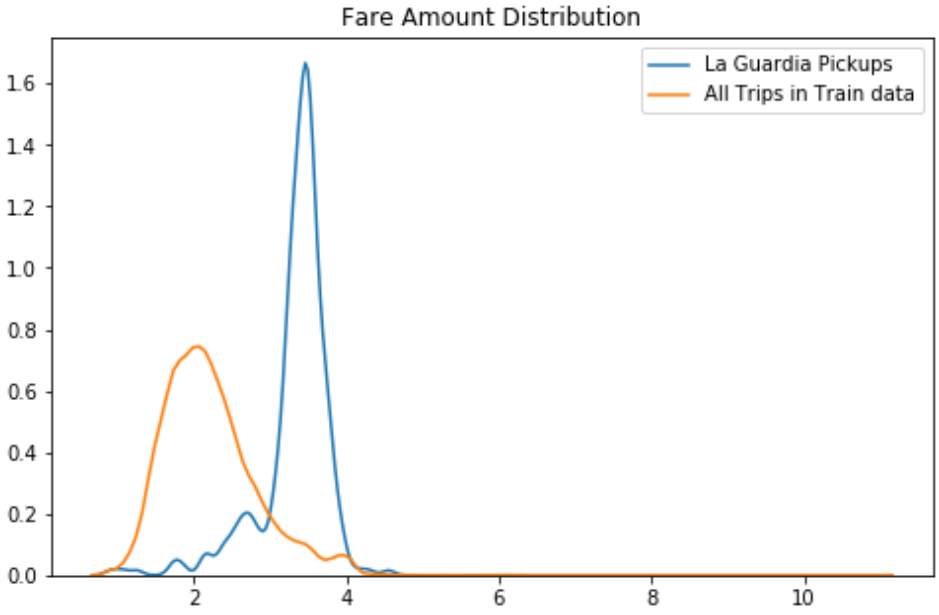
The range of latitudes and longitudes are between -90 to 90 and -180 to 180 respectively. But in the training data set we observed latitudes and longitudes in range of (-3488.079513, 3344.459268) which is not possible. On further exploration, we also identified a set of records which had both pickup and drop-off coordinates at the Equator. Since, this data is for taxi rides in New York, we remove these rows from our analysis. Such anomalies where not found in the test data.

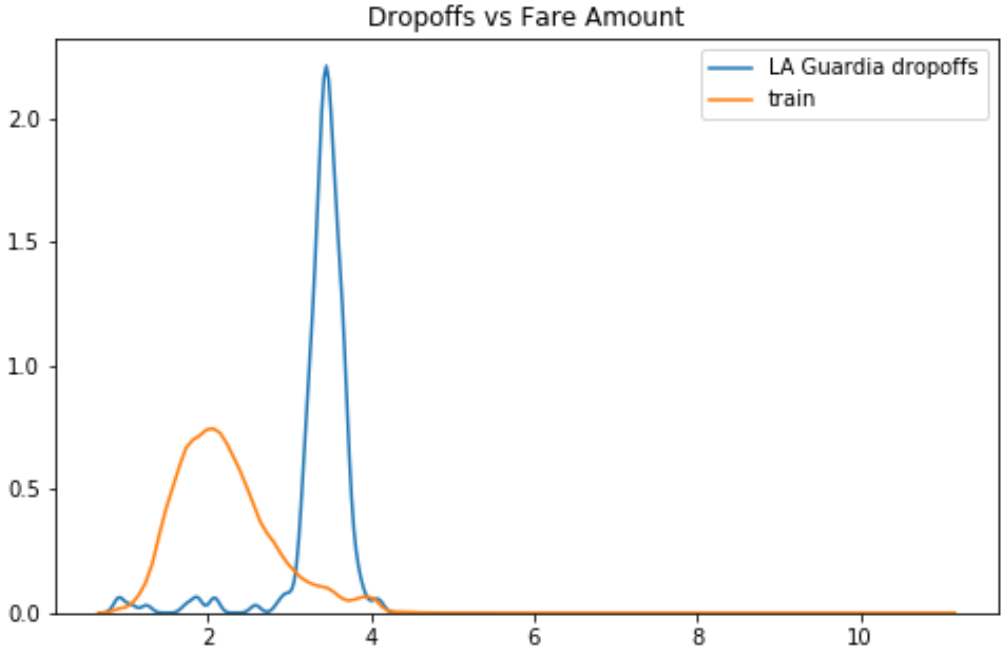

Dropoffs
40.850
40.825
40.800
40.775
40.750
40.725
40.700
40.675
40.650
dropoff_latitude
−74.025
−74.000
−73.975
−73.950
−73.925
−73.900
−73.875
−73.850
dropoff_longitude


Pickups
40.850
40.825
40.800
40.775
40.750
40.725
40.700
40.675
40.650
pickup_latitude
−74.025
−74.000
−73.975
−73.950
−73.925
−73.900
−73.875
−73.850
pickup_longitude

We can see that there is a cluster of pickups near La Guardia Airport.We then looked at what is the average fare amount for pickups and drop offs to La Guardia Airport, compared to all trips in the train data .

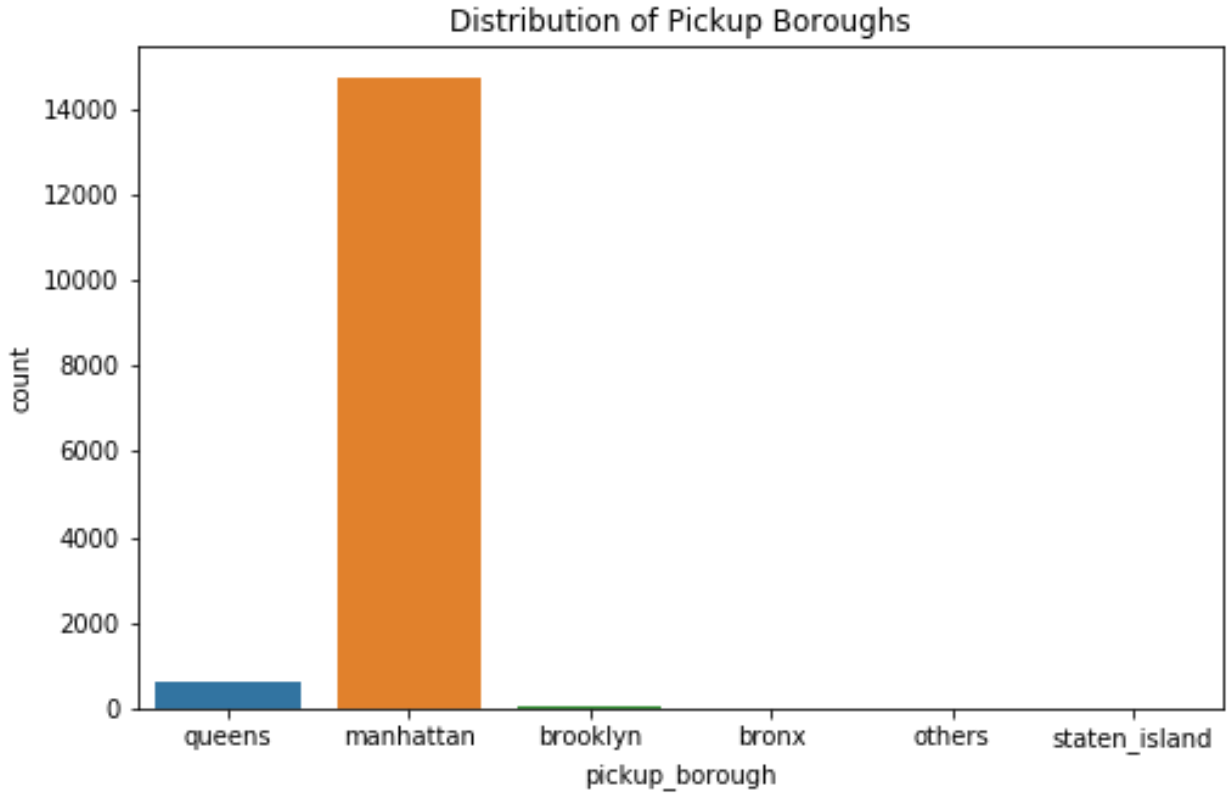
Number of Trips with Pickups from LA Guardia 324
Number of Trips with Dropoffs to LA Guardia 206

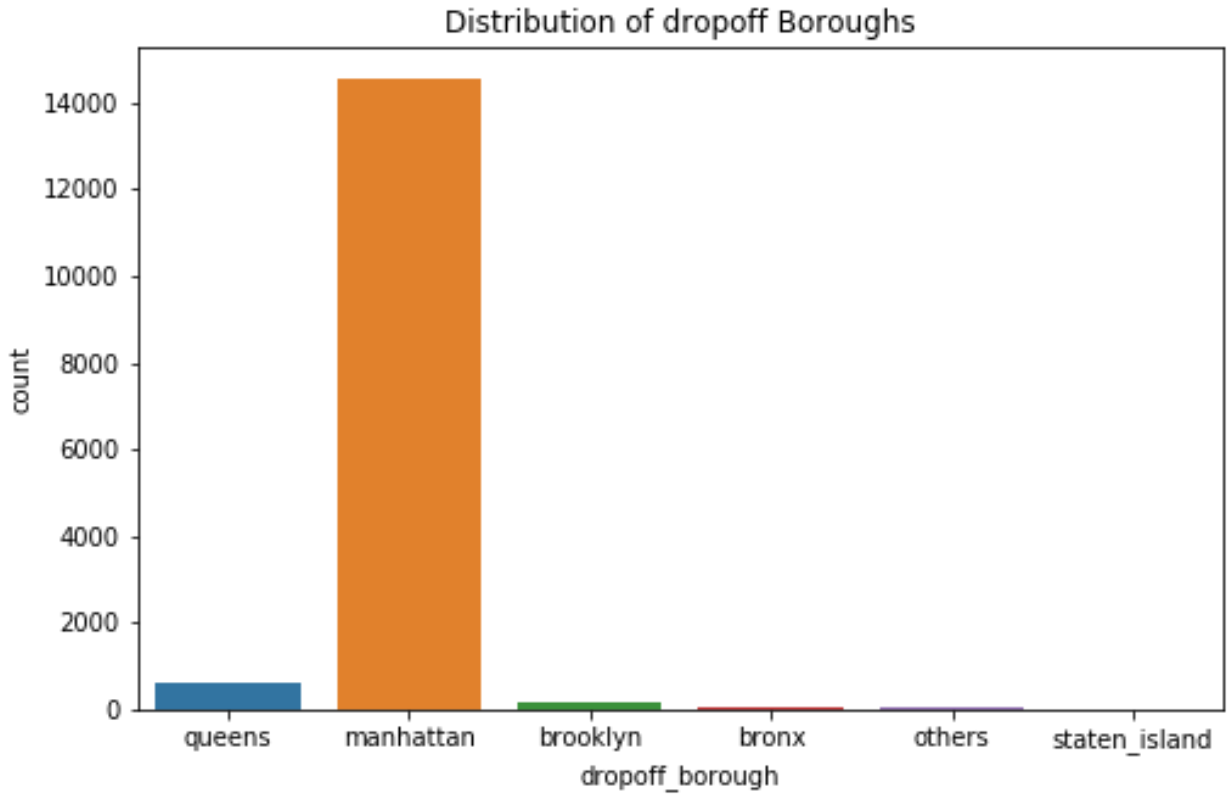


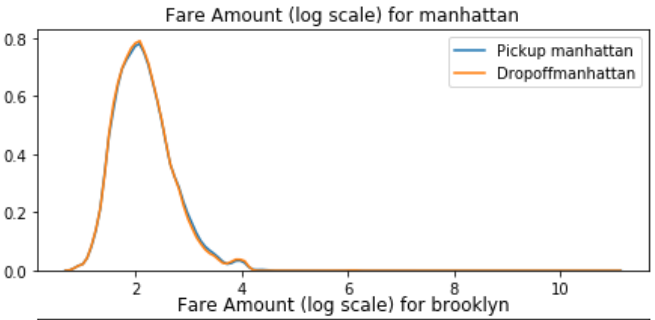


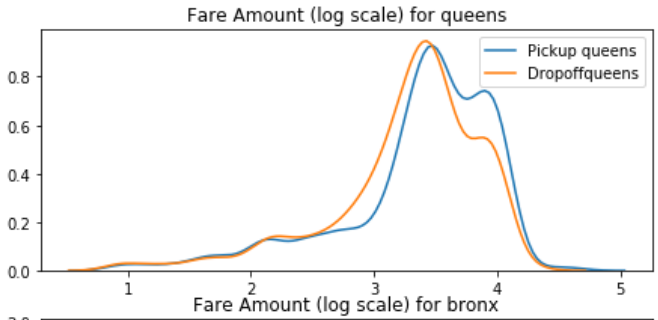
So after plotting the same we observed that fare was higher for airport trips. Based on this observations we created features to check whether a pickup or a drop-off was to airport in LaGuardia

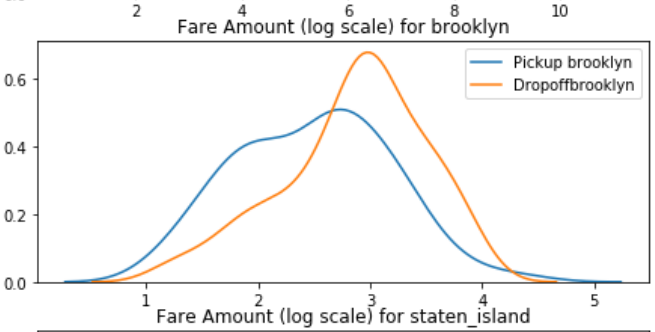
The next step was to check whether our hypothesis of fare from certain neighborhoods are higher than the rest, based on the 5 Boroughs New York city is divided — Manhattan, Queens, Brooklyn, Staten Island and Bronx, each pickup and drop off location was grouped into these 5 neighborhoods.

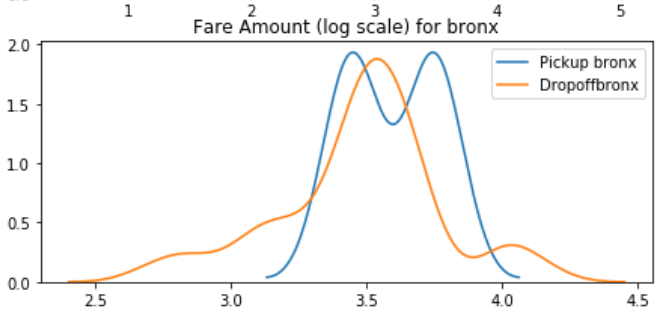


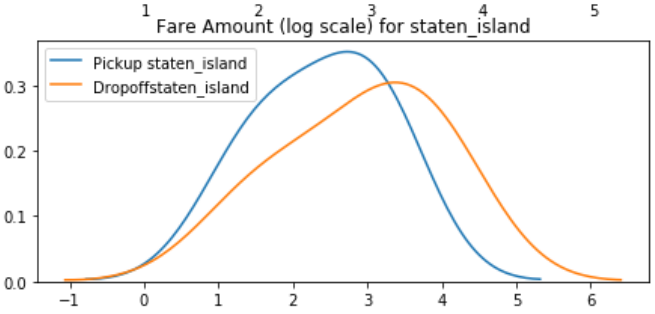

Distribution of dropoff Boroughs
count
dropoff_borough
queens
manhattan
brooklyn
bronx
others
staten_island


Fare Amount (log scale) for manhattan
Pickup manhattan
Dropoffmanhattan

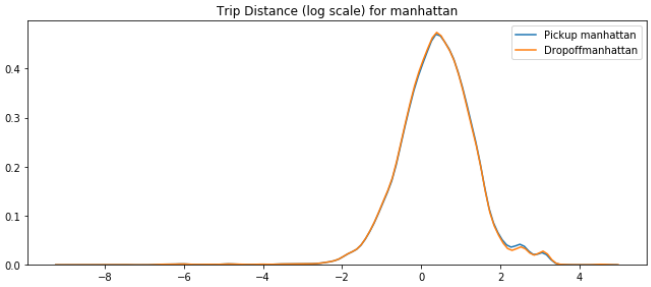

Fare Amount (log scale) for queens
Pickup queens
Dropoffqueens

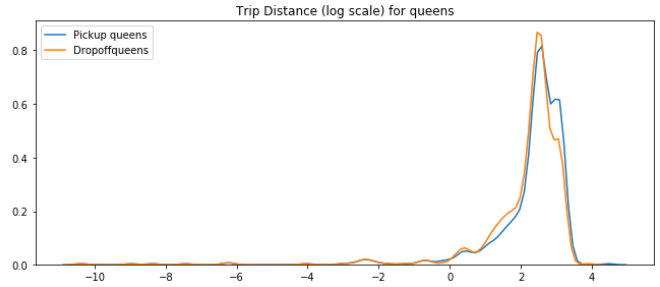

Fare Amount (log scale) for brooklyn
Pickup brooklyn
Dropoffbrooklyn

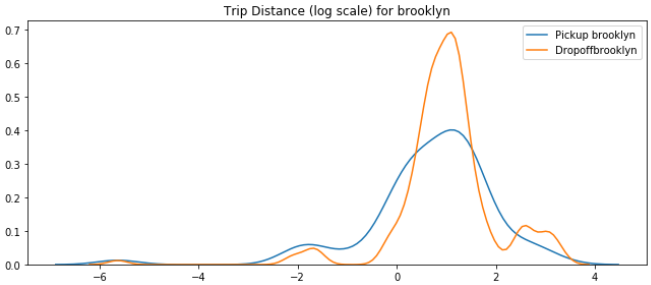

Fare Amount (log scale) for bronx
Pickup bronx
Dropoffbronx

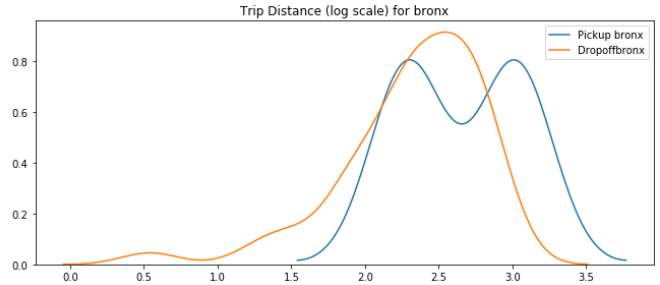

Fare Amount (log scale) for staten_island
Pickup staten_island
Dropoffstaten_island

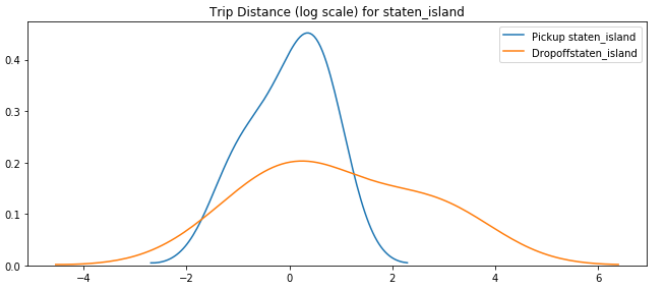
. And yes our hypothesis was right- except for Manhattan which had most of the pickups and drop offs, for every other neighborhood, there was a difference in the pickup and drop off fare distribution. Also, Queens had a higher mean pickup fare and dropoff fare compared to other neighborhoods. We can see pickups from Queens is expensive compared to pickups from other Buroughs.Very high difference in pickup and dropoff prices for Brooklyn







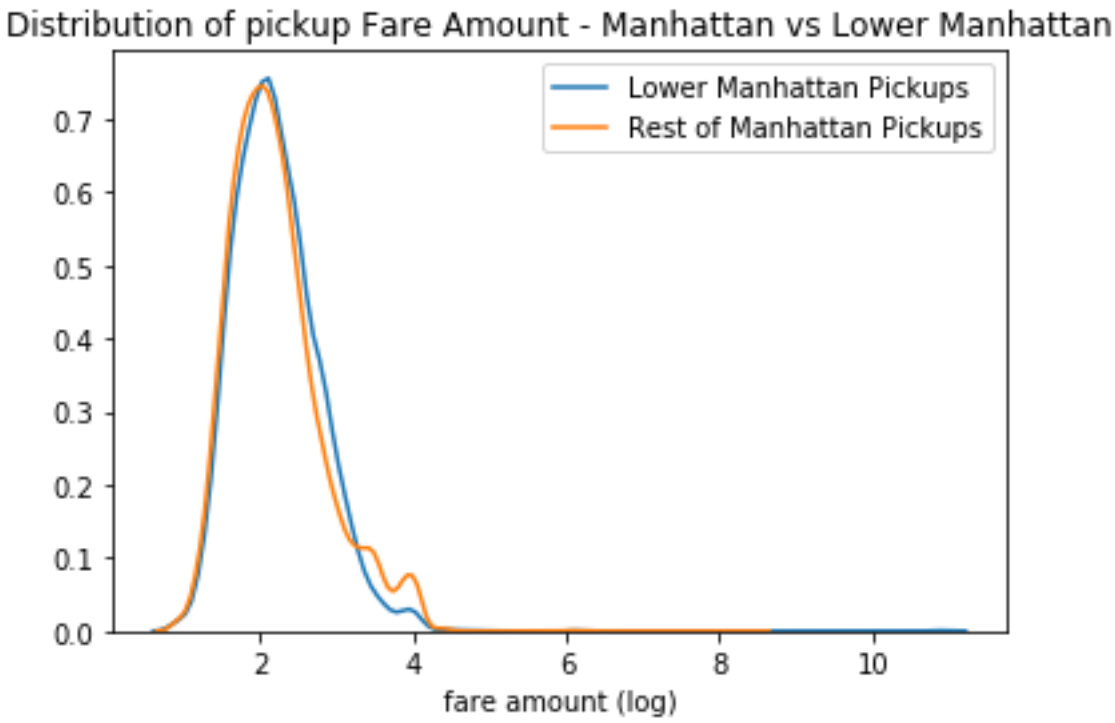


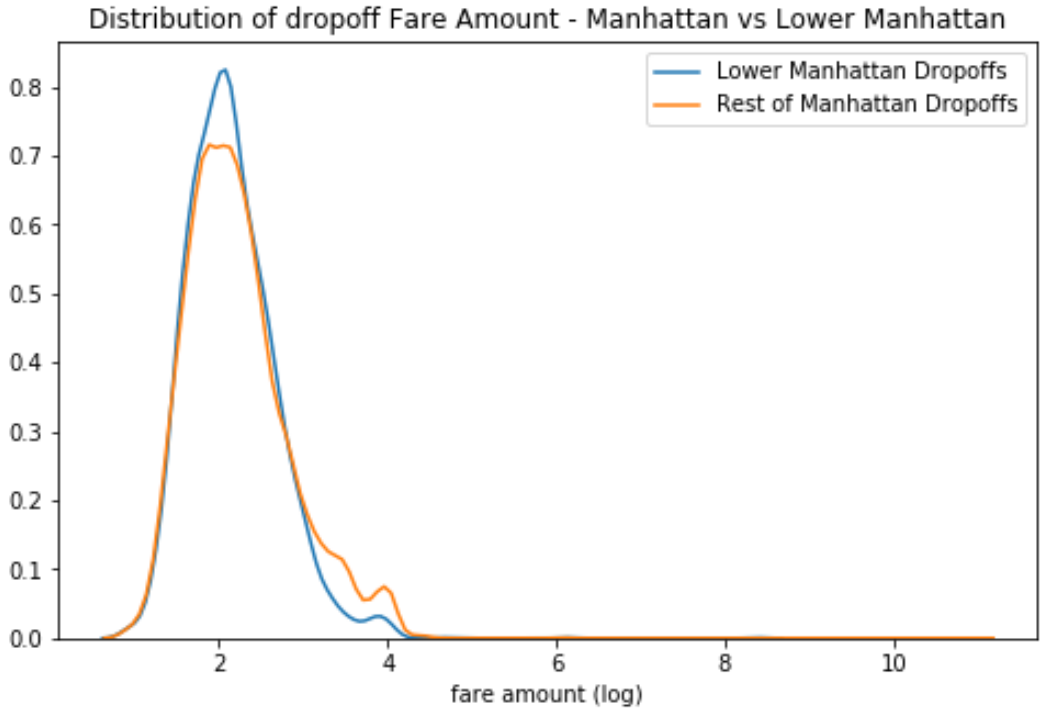


Dropoffs to Bronx and queens are long trips. In Manhattan the pickup and dropoffs fare amount has similar distribution. Let us add a field, is_lower_manhattan as we had seen above that dropoffs to lower manhattan had higher trip distance but lower fare

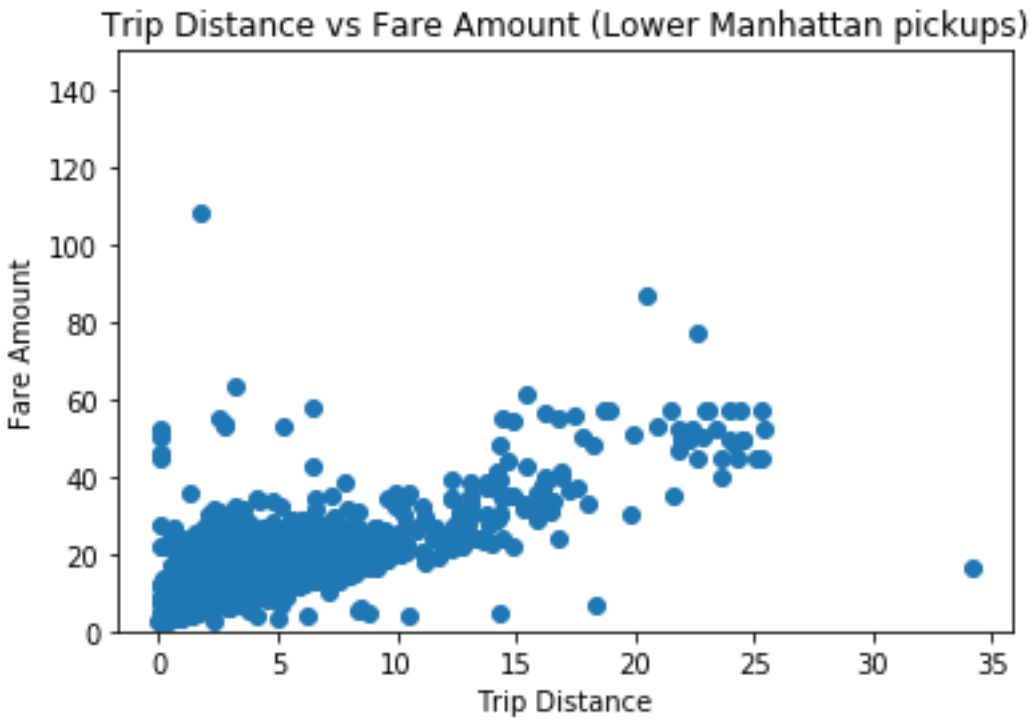
So After adding fields as is_pickup_lower_manhattan and is_dropoff_lower_manhattan ,

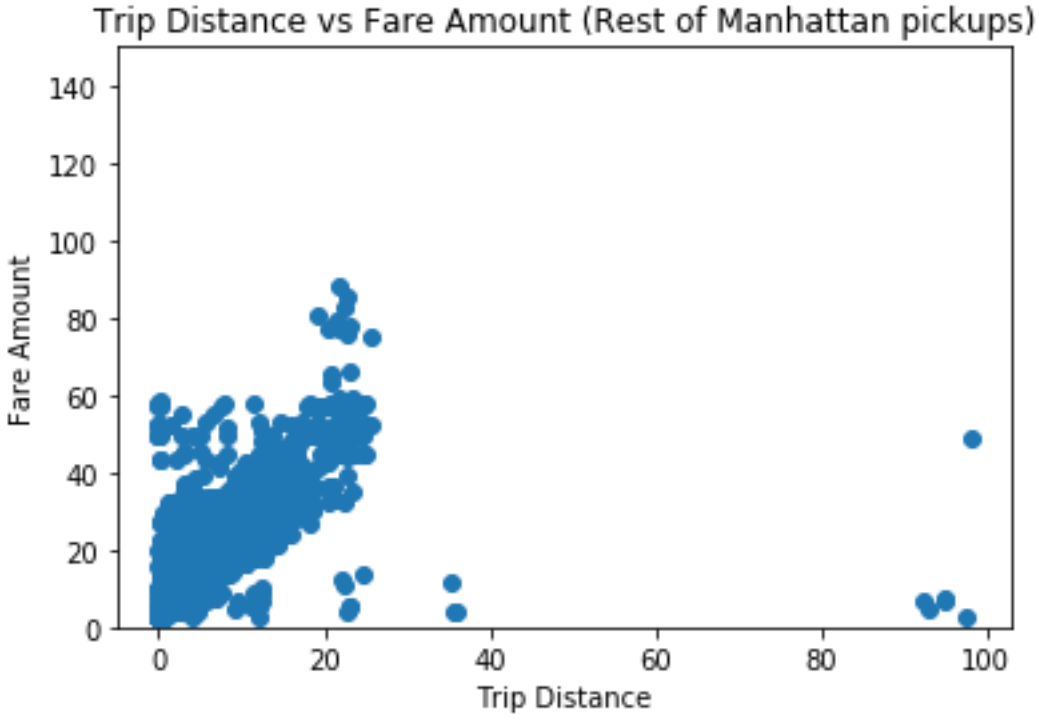
We have seen the variations in pickup fare amount for manhattan and lower manhattan.

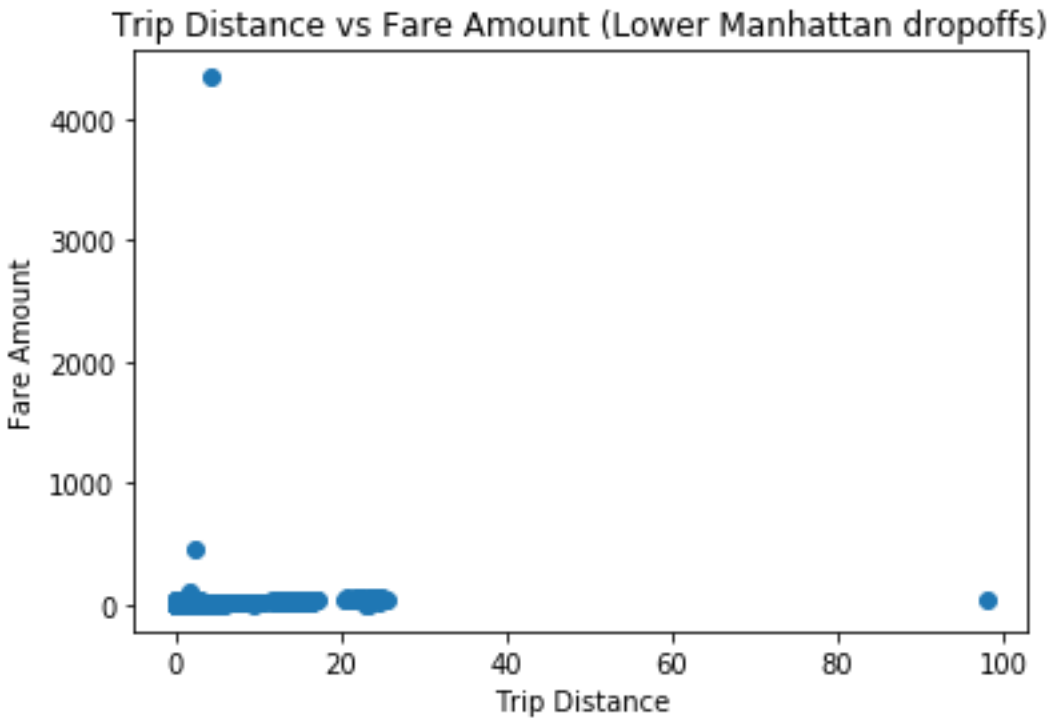


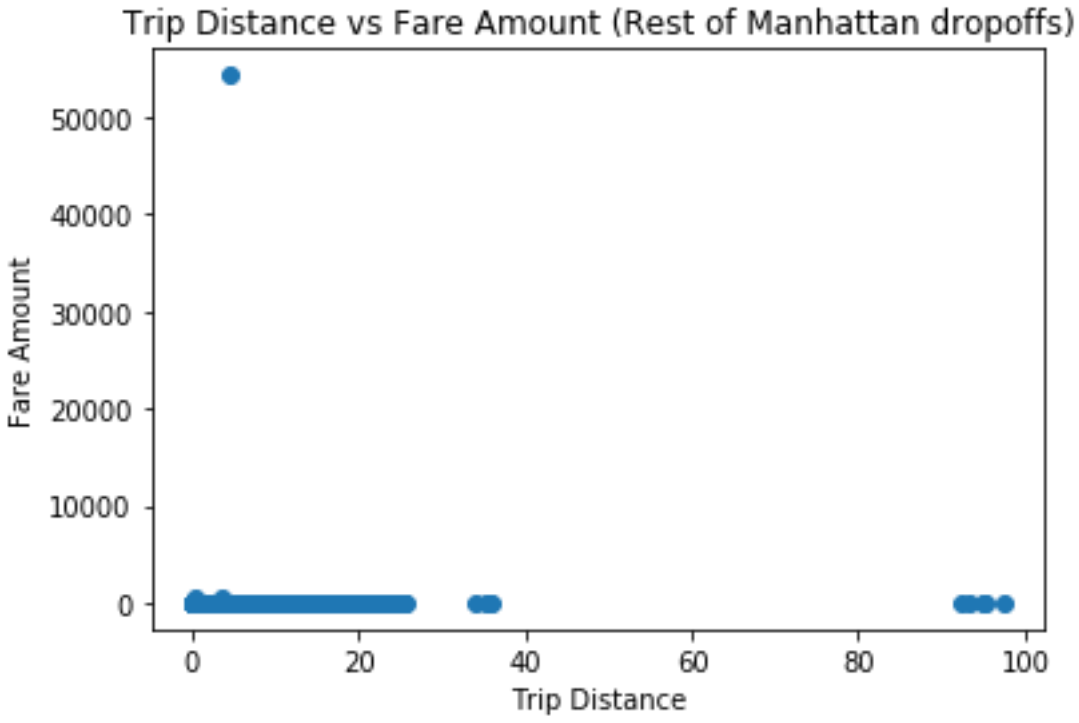


And from above we found that there is difference in dropoff fare amount for manhattan and lower manhattan but same for manhattan and lower manhattan in case of pickup fare amount .


Trip Distance vs Fare Amount (Lower Manhattan pickups)
Fare Amount
Trip Distance
0
20
40
60
80
100
120
140
0
5
10
15
20
25
30
35


Trip Distance vs Fare Amount (Rest of Manhattan pickups)
Fare Amount
Trip Distance
0
20
40
60
80
100
120
140
0
20
40
60
80
100

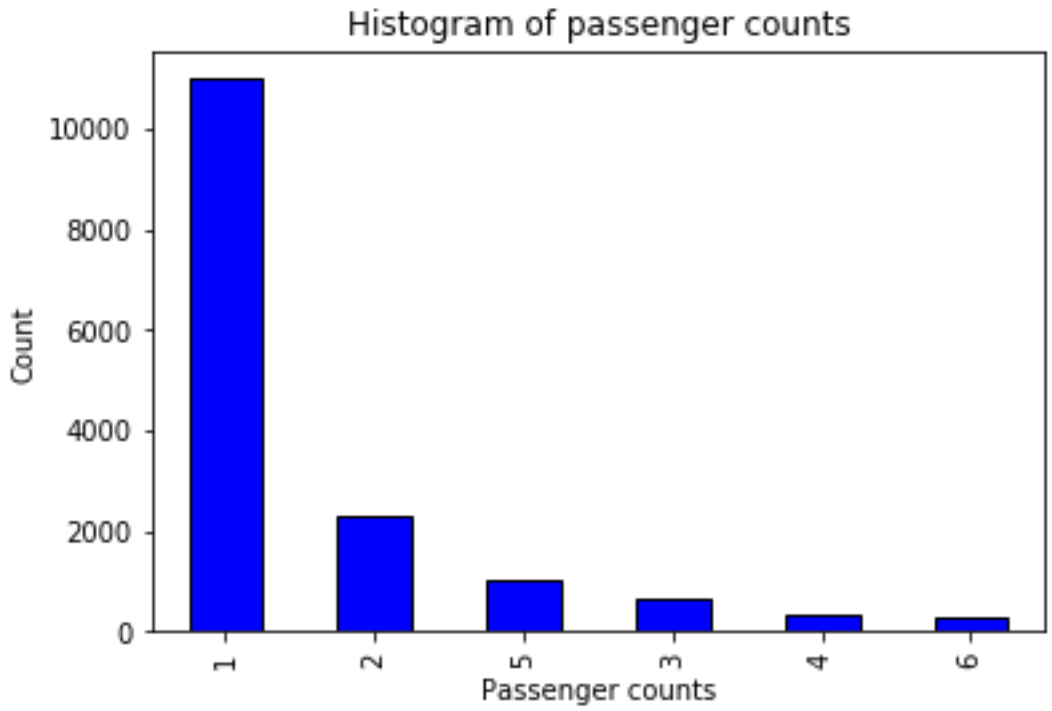


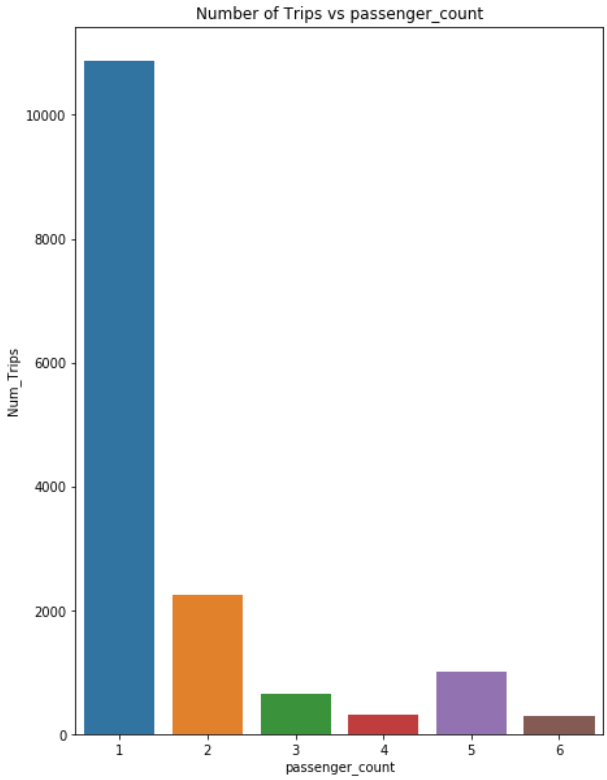


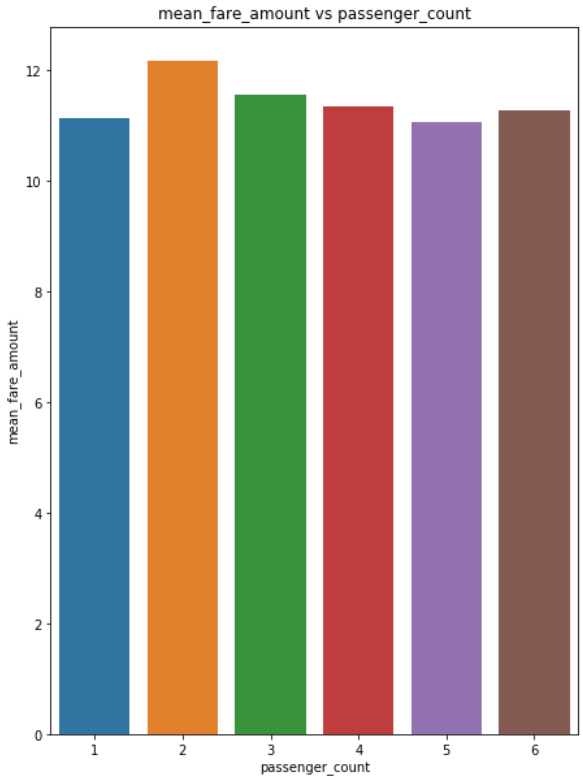
The distribution of trip distance and fare amount for Lower Manhattan pickups and dropoffs is very different. Also, slope of linear realtionship for pickups for Lower Manhattan is higher than that for Rest of Manhattan

Distribution of Passenger Count

We have seen from the data that we have passenger count more then 6 also which is not possible so we have set the max passenger count to 6 and plot the histogram for passenger count and found we hve maximum passenger as 1 then 2 and so on.





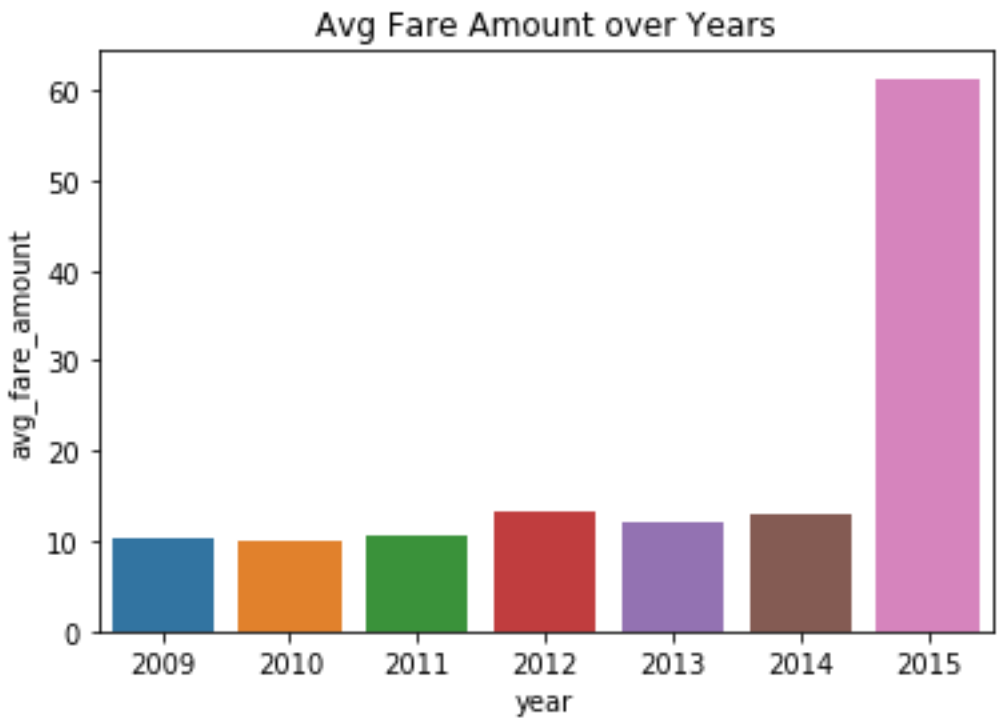


4. Distribution of Pickup date time

The first step to analyse how the fares have changed over time, is to create features like hour, day of the week, day, month, year from pickup datetime.

After creating the variables like hour,weekday,day,month,year.

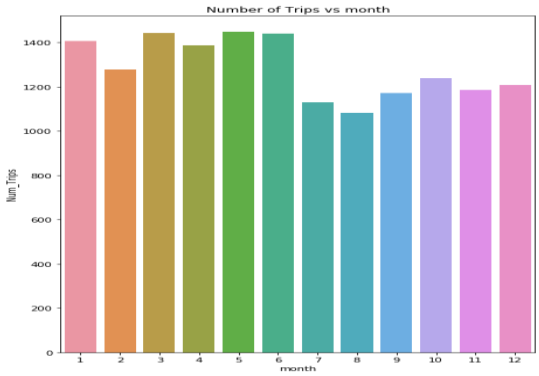
We have drawn the bar chart between average fare amount and consecutive years.

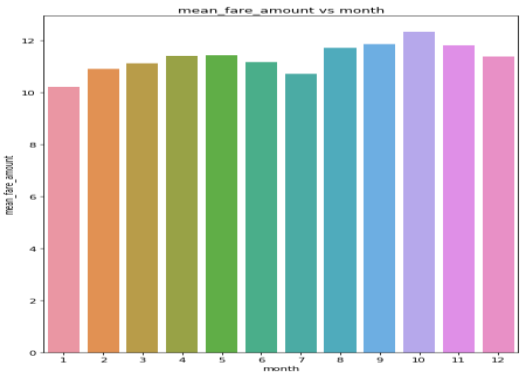


As expected, over years the average taxi fare has increased.

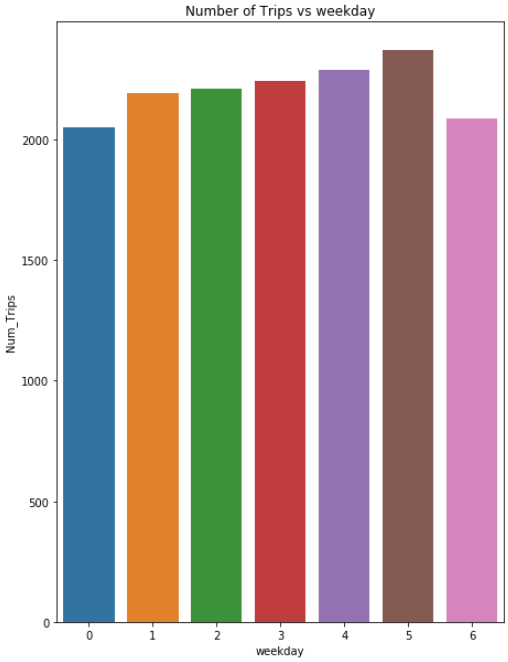
Now we have checked the relationship between Num of trips and fare amount with consecutive months

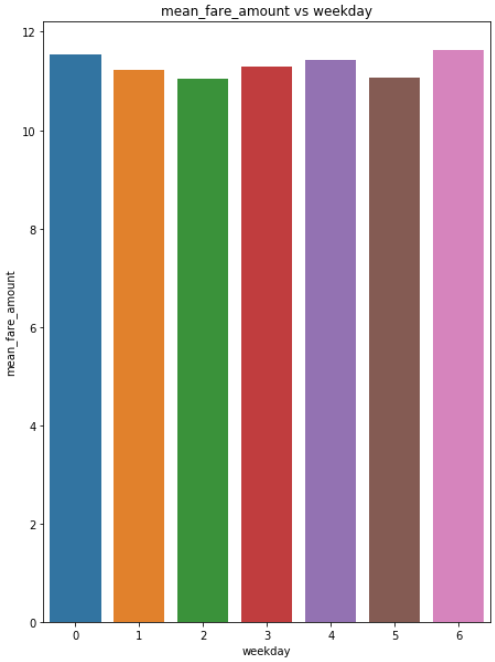
And found :-



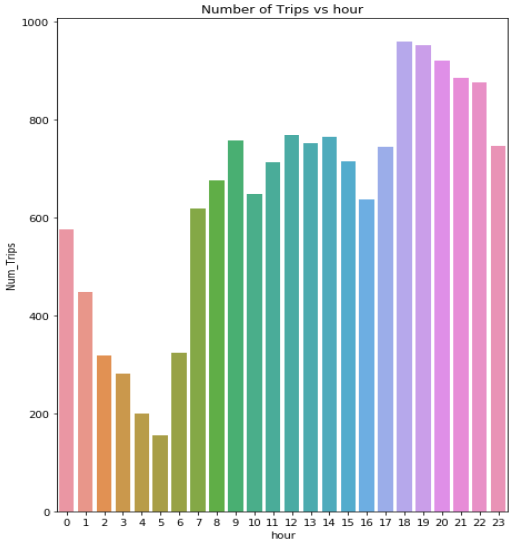


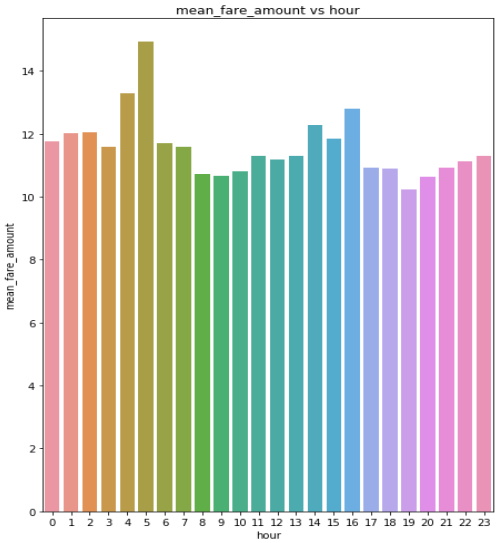
Over months, though there have been fewer pickups from July to December, the average fare is almost constant across months.





We observed that though the number of pickups are higher on Saturday, the average fare amount is lower. On Sunday and Monday though the number of trips are lower, avg fare amount is higher





The average fare amount at 5 am is the highest while the number of trips at 5 am are the least. This is because, at 5 AM 83% of the trips are to the airport.The number of trips are highest in 18 and 19 hours

Chapter 3: Modelling

3.1 Model Selection

The dependent variable in our model is a continuous variable i.e. fare_amount. Hence the models that we choose are Linear Regression, Decision Tree and Random Forest. The error metric chosen for the problem statement is Mean Absolute Error (MAE). And Mean absolute percentage error and Root Mean Square Error(RMSE)

3.1 Multiple Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables. The independent variables can be continuous or categorical.

Reason for failure of logistic regression model, is that it tries to fit a linear line between the variables and the targer. But, as we saw in the Exploratory analysis phase this is not true.

As you can see the Adjusted R-squared value, we can explain 87.13% of the data using our multiple linear regression model. By looking at the F-statistic and combined p-value we can reject the null hypothesis that target variable does not depend on any of the predictor variables. This model explains the data very well and is considered to be good.

Even after removing the non-significant variables, the accuracy, Adjusted R-squared and F- statistic do not change by much, hence the accuracy of this model is chosen to be final.
Mean Absolute Error (MAE) is calculated and found to be 3.
MAPE of this multiple linear regression model is 25.82%. Hence the accuracy of this model is 74.18%. This model performs very well for this test data.
RMSE for Linear regression is 7.53

3.2 Decision Tree:

A decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

Using decision tree, we can predict the value of bike count. MAE for this model is 3.4. T he MAPE for this decision tree is 26.78%. Hence the accuracy for this model is 74.28%. RMSE for decision tree 5.88

3.1 Random Forest:

Random Forest has reduced the RMSE considerably as compared to Linear Regression. Random Forest works on the principle of Bagging The idea behind this is very intutive- when we want to make a decision about a field we do not know about we take advice from a people and then we decide based on the majority opinion. This is the idea behind Random Forest - multiple decision trees are created and output from each of the trees are averaged to predict the value. Random Forest are not susceptible to overfitting and since in Random Forest, each tree is trained independently of the other, it is more robust.

Using Classification for prediction analysis in this case is not normal, though it can be done. The number of decision trees used for prediction in the forest is 1000. MAE for this model is 2.18. Using random forest, the MAPE was found to be 23.07%. Hence the accuracy is 77.97%.
RMSE for decision tree 4.59.

Chapter 4: Conclusion

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models.
We can compare the models using any of the following criteria:

1. Predictive Performance
2. Interpretability
3. Computational Efficiency

In our case of Cab fare prediction Data, Interpretability and Computation Efficiency, do not hold much significance. Therefore, we will use Predictive performance as the criteria to compare and evaluate models.

Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

4.1 Mean Absolute Error (MAE)

MAE is one of the error measures used to calculate the predictive performance of the model. We will apply this measure to our models that we have generated in the previous section.

```
MAE <- function (actual, pred)
{
  print(mean (abs (actual - pred)))
}
```

LinearRegressionModel:
MAE = 3.18
Decision Tree:MAE=3.42.
Random Forest: MAE = 2.18

4.2 Root Mean Square Error (RMSE)

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.

The formula is:

$$RMSE = \sqrt{\overline{(f-o)^2}}$$

Where:

- f = forecasts (expected values or unknown results),
- o = observed values (known results).

LinearRegressionModel:s
MAE = 7.53
Decision Tree:MAE=5.88.
Random Forest: MAE = 4.598.

Based on the above error metrics, Random Forest is the better model for our analysis. Hence Random Forest is chosen as the model for prediction of bike rental count.

Chapter 5: Python Code

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from math import radians, cos, sin, asin, sqrt

import folium
from folium import FeatureGroup, LayerControl, Map, Marker
from folium.plugins import HeatMap
from folium.plugins import TimestampedGeoJson
from folium.plugins import MarkerCluster
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.tree import DecisionTreeRegressor

from sklearn.preprocessing import LabelEncoder

import statsmodels.api as sm


os.chdir(r"C:\Users\Bhavesh\Desktop\Data scientist\Project\Cab Fare")

os.getcwd()

dataset = pd.read_csv('train_cab.csv', encoding = "ISO-8859-1",sep=',')

dataset.head(5)


dataset['fare_amount'] =dataset['fare_amount'].astype(float)

dataset['pickup_datetime']=pd.to_datetime(dataset['pickup_datetime'],format='%Y-%m-
%d %H:%M:%S UTC')

dataset['passenger_count'] = pd.to_numeric(dataset['passenger_count'],
errors='coerce')

dataset['passenger_count'] =dataset['passenger_count'].astype(int)


dataset[dataset['passenger_count'].isnull()]

dataset[dataset['fare_amount'].isnull()]

dataset = dataset.dropna(subset=['passenger_count'])

dataset = dataset.dropna(subset=['fare_amount'])


dataset.describe()

dataset.dtypes


dataset1 = dataset[((dataset['pickup_longitude'] > -78) & (dataset['pickup_longitude'] <
-70)) & ((dataset['dropoff_longitude'] > -78) & (dataset['dropoff_longitude'] < -70)) &
((dataset['pickup_latitude'] > 37) & (dataset['pickup_latitude'] < 45)) &
((dataset['dropoff_latitude'] > 37) & (dataset['dropoff_latitude'] < 45)) &
(dataset['passenger_count'] > 0) & (dataset['fare_amount'] >= 2.5)]


dataset1.shape
```

```
# plotting histogram for fare amount


plt.figure(figsize = (14, 7))
 n, bins, patches = plt.hist(dataset1['fare_amount'], 10000,facecolor='blue', alpha=0.75)
 plt.xlabel('Fare amount')
 plt.title('Histogram of fare amount')
 plt.xlim(0, 200)
 plt.show();



 # plotting KDE plot for fare amount


plt.figure(figsize=(8,5))
sns.kdeplot(np.log(dataset1['fare_amount'].values)).set_title("Distribution of fare amount (log scale)")


# plotting histogram for passenger count
dataset1.groupby('passenger_count').size()
dataset1 = dataset1.loc[dataset1['passenger_count'] <= 6]
dataset1['passenger_count'].value_counts().plot.bar(color = 'b', edgecolor = 'k');
plt.title('Histogram of passenger counts'); plt.xlabel('Passenger counts'); plt.ylabel('Count');


# creating features(varibles) as below from pickup_datetime
dataset1['year'] = dataset.pickup_datetime.dt.year
dataset1['month'] = dataset.pickup_datetime.dt.month
dataset1['day'] = dataset.pickup_datetime.dt.day
dataset1['weekday'] = dataset.pickup_datetime.dt.weekday
```

```
dataset1['hour'] = dataset.pickup_datetime.dt.hour
dataset1.head(5)

# calculating the distance of rides and creating feature names as distance
def haversine_np(lon1, lat1, lon2, lat2):



    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

    dlon = lon2 - lon1
    dlat = lat2 - lat1


    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2


    c = 2 * np.arcsin(np.sqrt(a))
    km = 6371 * c  # 6371 is Radius of earth in kilometers. Use 3956 for miles
    return km



dataset1['distance'] =  haversine_np(dataset1['pickup_latitude'],
dataset1['pickup_longitude'], dataset1['dropoff_latitude' ], dataset1['dropoff_longitude'])

# plotting the scatter plot for fare amount VS trip distance
yaxix = (0, 500)
```

```
plt.scatter(x=dataset1['distance'],y=dataset1['fare_amount'])
plt.xlabel("Trip Distance")
plt.ylabel("Fare Amount")
#dataset1.drop(['distance'], axis=1)
plt.ylim(yaxix)

# plotting the hostogram for ride distance
plt.figure(figsize = (14, 4))
n, bins, patches = plt.hist(dataset1.distance, 1000, facecolor='blue', alpha=0.75)
plt.xlabel('distance')
plt.title('Histogram of ride distance')
plt.show();

# plotting the scatter plot for dropoff's longitude and latitude
city_long_border = (-74.03, -73.85)
city_lat_border = (40.63, 40.85)
dataset1.plot(kind='scatter', x='dropoff_longitude', y='dropoff_latitude',color='red',
s=.02, alpha=.6)
plt.title("Dropoffs")
plt.ylim(city_lat_border)
plt.xlim(city_long_border)

# plotting the scatter plot for pickup's longitude and latitude
dataset1.plot(kind='scatter', x='pickup_longitude', y='pickup_latitude',color='blue',
s=.02, alpha=.6)
plt.title("Pickups")
plt.ylim(city_lat_border)
plt.xlim(city_long_border)
```

```
# setting the LA Guardia coordinates finding out the number of pickups and dropoffs for La Guardia
LAGuardia={'min_lng':-73.8895,
    'min_lat':40.7664,
    'max_lng':-73.8550,
    'max_lat':40.7931}
LA_center=[40.763599,-73.863029]
# Get all pickups to JFK
LA_data=dataset1.loc[(dataset1.pickup_latitude>=LAGuardia['min_lat']) & (dataset1.pickup_latitude<=LAGuardia['max_lat'])]
LA_data=LA_data.loc[(dataset1.pickup_longitude>=LAGuardia['min_lng']) & (dataset1.pickup_longitude<=LAGuardia['max_lng'])]
print("Number of Trips with Pickups from LA Guardia",LA_data.shape[0])
LA_dropoff=dataset1.loc[(dataset1.dropoff_latitude>=LAGuardia['min_lat']) & (dataset1.dropoff_latitude<=LAGuardia['max_lat'])]
LA_dropoff=LA_dropoff.loc[(dataset1.dropoff_longitude>=LAGuardia['min_lng']) & (dataset1.dropoff_longitude<=LAGuardia['max_lng'])]
print("Number of Trips with Dropoffs to LA Guardia",LA_dropoff.shape[0])


# plotting the KDE plot for fare amount for La Guardia pickups and other pickups
plt.figure(figsize=(8,5))
sns.kdeplot(np.log(LA_data['fare_amount'].values),label='La Guardia Pickups')
#sns.kdeplot(np.log(JFK_dropoff['fare_amount'].values),label='JFK Dropoff')
sns.kdeplot(np.log(dataset1['fare_amount'].values),label='All Trips in Train data')
plt.title("Fare Amount Distribution")


# plotting the KDE plot for fare amount for La Guardia dropoffs and other dropoffs
plt.figure(figsize=(8,5))
sns.kdeplot(np.log(LA_dropoff['fare_amount'].values),label='LA Guardia dropoffs')
sns.kdeplot(np.log(dataset1['fare_amount'].values),label='train')
```

```
plt.title("Dropoffs vs Fare Amount")


# creating functions for LA Guardia pickups and dropoffs rides and creating features is_pickup_la_guardia and is_dropoff_la_guardia
def isAirport(latitude,longitude,airport_name='LA Guardia'):
    if airport_name=='la guardia':
        boundary={'min_lng':-73.8895,
                  'min_lat':40.7664,
                  'max_lng':-73.8550,
                  'max_lat':40.7931
                 }
    if latitude>=boundary['min_lat'] and latitude<=boundary['max_lat']:
        if longitude>=boundary['min_lng'] and longitude<=boundary['max_lng']:
            return 1
    else:
        return 0


nyc_airports={
    'LaGuardia':{'min_lng':-73.8895,
                 'min_lat':40.7664,
                 'max_lng':-73.8550,
                 'max_lat':40.7931


    }


}


def isAirport(latitude,longitude,airport_name='LA Guardia'):
```

```
    if latitude>=nyc_airports[airport_name]['min_lat'] and
latitude<=nyc_airports[airport_name]['max_lat'] and
longitude>=nyc_airports[airport_name]['min_lng'] and
longitude<=nyc_airports[airport_name]['max_lng']:
        return 1
    else:
        return 0


dataset1['is_pickup_la_guardia']=dataset1.apply(lambda
row:isAirport(row['pickup_latitude'],row['pickup_longitude'],'LaGuardia'),axis=1)
dataset1['is_dropoff_la_guardia']=dataset1.apply(lambda
row:isAirport(row['dropoff_latitude'],row['dropoff_longitude'],'LaGuardia'),axis=1)



# checking the non airports rides
non_airport=dataset1.loc[(dataset1['is_dropoff_la_guardia']==0)]
non_airport=non_airport.loc[(non_airport['is_pickup_la_guardia']==0)]
non_airport.shape


# creating array coordinates for all boroughs in new york


nyc_boroughs={
    'manhattan':{
        'min_lng':-74.0479,
        'min_lat':40.6829,
        'max_lng':-73.9067,
        'max_lat':40.8820
    },


    'queens':{
```

```
    'min_lng':-73.9630,
    'min_lat':40.5431,
    'max_lng':-73.7004,
    'max_lat':40.8007

},

'brooklyn':{
    'min_lng':-74.0421,
    'min_lat':40.5707,
    'max_lng':-73.8334,
    'max_lat':40.7395

},

'bronx':{
    'min_lng':-73.9339,
    'min_lat':40.7855,
    'max_lng':-73.7654,
    'max_lat':40.9176

},

'staten_island':{
    'min_lng':-74.2558,
    'min_lat':40.4960,
    'max_lng':-74.0522,
    'max_lat':40.6490
}
```

```
}


def getBorough(lat,lng):


    locs=nyc_boroughs.keys()
    for loc in locs:
        if lat>=nyc_boroughs[loc]['min_lat'] and lat<=nyc_boroughs[loc]['max_lat'] and
lng>=nyc_boroughs[loc]['min_lng'] and lng<=nyc_boroughs[loc]['max_lng']:
            return loc
    return 'others'


# Creating feature for borough as pickup_borough and dropoff_borough


dataset1['pickup_borough']=dataset1.apply(lambda
row:getBorough(row['pickup_latitude'],row['pickup_longitude']),axis=1)
dataset1['dropoff_borough']=dataset1.apply(lambda
row:getBorough(row['dropoff_latitude'],row['dropoff_longitude']),axis=1)


# Creating countplot for pickup and dropoffs boroughs
plt.figure(figsize=(8,5))
sns.countplot(x=dataset1['pickup_borough'])
plt.title("Distribution of Pickup Boroughs")'


plt.figure(figsize=(8,5))
sns.countplot(x=dataset1['dropoff_borough'])
plt.title("Distribution of dropoff Boroughs")'



# creating KDE plot Distribution of Fare Amount Across Buroughs
plt.figure(figsize=(16,10))
```

```
plt.title("Distribution of Fare Amount Across Buroughs")
i=1
for key in nyc_boroughs.keys():
    plt.subplot(3,2,i)

sns.kdeplot(np.log(dataset1.loc[dataset1['pickup_borough']==key,'fare_amount'].values
),label='Pickup '+ key)

sns.kdeplot(np.log(dataset1.loc[dataset1['dropoff_borough']==key,'fare_amount'].value
s),label='Dropoff'+ key).set_title("Fare Amount (log scale) for "+key)

    i=i+1

 # creating KDE plot Distribution of trip distance Across Buroughs

plt.figure(figsize=(24,15))
plt.title("Distribution of Trip Distances Across Buroughs")
i=1
for key in nyc_boroughs.keys():
    plt.subplot(3,2,i)

sns.kdeplot(np.log(dataset1.loc[dataset1['pickup_borough']==key,'distance'].values),lab
el='Pickup '+ key)

sns.kdeplot(np.log(dataset1.loc[dataset1['dropoff_borough']==key,'distance'].values),la
bel='Dropoff'+ key).set_title("Trip Distance (log scale) for "+key)
    i=i+1

# creating fucting for rides in lower Manhattan
lower_manhattan_boundary={'min_lng': -74.0194,
```

```
                    'min_lat':40.6997,
                    'max_lng':-73.9716,
                    'max_lat':40.7427}


def isLowerManhattan(lat,lng):
    if lat>=lower_manhattan_boundary['min_lat'] and
lat<=lower_manhattan_boundary['max_lat'] and
lng>=lower_manhattan_boundary['min_lng'] and
lng<=lower_manhattan_boundary['max_lng']:
        return 1
    else:
        return 0


# Creating feature as is_pickup_lower_manhattan  and is_dropoff_lower_manhattan
dataset1['is_pickup_lower_manhattan']=dataset1.apply(lambda
row:isLowerManhattan(row['pickup_latitude'],row['pickup_longitude']),axis=1)
dataset1['is_dropoff_lower_manhattan']=dataset1.apply(lambda
row:isLowerManhattan(row['dropoff_latitude'],row['dropoff_longitude']),axis=1)


#checking the rides in manhattan only
manhattan=dataset1.loc[(dataset1['pickup_borough']=='manhattan') |
(dataset1['dropoff_borough']=='manhattan')]
manhattan.shape


# plotting a Kde plot for  Distribution of pickup Fare Amount - Manhattan vs Lower
Manhattan
sns.kdeplot(np.log(manhattan.loc[manhattan['is_pickup_lower_manhattan']==1,'fare_a
mount'].values),label='Lower Manhattan Pickups')
sns.kdeplot(np.log(manhattan.loc[manhattan['is_pickup_lower_manhattan']==0,'fare_a
mount'].values),label='Rest of Manhattan Pickups')
plt.xlabel("fare amount (log)")
plt.title("Distribution of pickup Fare Amount - Manhattan vs Lower Manhattan")
```

```
# plotting a Kde plot for  Distribution of dropoffs Fare Amount - Manhattan vs Lower Manhattan


plt.figure(figsize=(8,5))

sns.kdeplot(np.log(manhattan.loc[manhattan['is_dropoff_lower_manhattan']==1,'fare_amount'].values),label='Lower Manhattan Dropoffs')

sns.kdeplot(np.log(manhattan.loc[manhattan['is_dropoff_lower_manhattan']==0,'fare_amount'].values),label='Rest of Manhattan Dropoffs')

plt.xlabel("fare amount (log)")

plt.title("Distribution of dropoff Fare Amount - Manhattan vs Lower Manhattan")


# plotting the scatter for Trip Distance vs Fare Amount (Lower Manhattan pickups)

yaxix = (0, 150)

plt.scatter(x=manhattan.loc[manhattan['is_pickup_lower_manhattan']==1,'distance'].values,y=manhattan.loc[manhattan['is_pickup_lower_manhattan']==1,'fare_amount'].values)

plt.xlabel("Trip Distance")

plt.ylabel("Fare Amount")

plt.title("Trip Distance vs Fare Amount (Lower Manhattan pickups)")

plt.ylim(yaxix)


# plotting the scatter for Trip Distance vs Fare Amount (Rest of  Manhattan pickups)

yaxix = (0, 150)

plt.scatter(x=manhattan.loc[manhattan['is_pickup_lower_manhattan']==0,'distance'].values,y=manhattan.loc[manhattan['is_pickup_lower_manhattan']==0,'fare_amount'].values)

plt.xlabel("Trip Distance")

plt.ylabel("Fare Amount")

plt.title("Trip Distance vs Fare Amount (Rest of Manhattan pickups)")

plt.ylim(yaxix)
```

```
# plotting the scatter for Trip Distance vs Fare Amount (Lower Manhattan dropoffs)

plt.scatter(x=manhattan.loc[manhattan['is_dropoff_lower_manhattan']==1,'distance'].values,y=manhattan.loc[manhattan['is_dropoff_lower_manhattan']==1,'fare_amount'].values)

plt.xlabel("Trip Distance")

plt.ylabel("Fare Amount")

plt.title("Trip Distance vs Fare Amount (Lower Manhattan dropoffs)")


# plotting the scatter for Trip Distance vs Fare Amount (Rest of  Manhattan dropoffs)

plt.scatter(x=manhattan.loc[manhattan['is_dropoff_lower_manhattan']==0,'distance'].values,y=manhattan.loc[manhattan['is_dropoff_lower_manhattan']==0,'fare_amount'].values)

plt.xlabel("Trip Distance")

plt.ylabel("Fare Amount")

plt.title("Trip Distance vs Fare Amount (Rest of Manhattan dropoffs)")


# creating a bar plot for num of trips VS year

dataset1['key'] = dataset1['fare_amount']

trips_year=dataset1.groupby(['year'])['key'].count().reset_index().rename(columns={'key':'Num_Trips'})

trips_year.head()

sns.barplot(x='year',y='Num_Trips',data=trips_year)


# creating a bar plot for num of fare amount VS year

trips_year_fareamount=dataset1.groupby(['year'])['fare_amount'].mean().reset_index().rename(columns={'fare_amount':'avg_fare_amount'})

sns.barplot(x='year',y='avg_fare_amount',data=trips_year_fareamount).set_title("Avg Fare Amount over Years")

dataset1=dataset1[dataset1['fare_amount']<=200]
```

```
# creating a bar plot for num of trips VS month   and  fare amount VS month
def groupandplot(data,groupby_key,value,aggregate='mean'):
    plt.figure(figsize=(16,10))

agg_data=data.groupby([groupby_key])[value].agg(aggregate).reset_index().rename(columns={value:aggregate+'_'+value})
    plt.subplot(1,2,1)

count_data=dataset1.groupby([groupby_key])['key'].count().reset_index().rename(columns={'key':'Num_Trips'})
    sns.barplot(x=groupby_key,y='Num_Trips',data=count_data).set_title("Number of Trips vs "+groupby_key)


    plt.subplot(1,2,2)

sns.barplot(x=groupby_key,y=aggregate+'_'+value,data=agg_data).set_title(aggregate+'_'+value+" vs "+groupby_key)



 # creating a bar plot for num of trips VS weekday   and  fare amount VS weekday
groupandplot(dataset1,'weekday','fare_amount')

 # creating a bar plot for num of trips VS hour   and  fare amount VS hour
groupandplot(dataset1,'hour','fare_amount')

 # creating a bar plot for num of trips VS passenger count   and  fare amount VS passenger count
dataset1=dataset1[dataset1['passenger_count']<=6]
groupandplot(dataset1,'passenger_count','fare_amount')
dataset1= dataset1.drop(['key'], axis=1)
```

```
# saving the cleaned data into train_cleaned file
dataset1.to_csv("train_cleaned.csv",index=False)

# Liner Regression Model

y = dataset1['fare_amount']
X = dataset1.drop(columns=['fare_amount','pickup_datetime'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
lr = LinearRegression()

le = LabelEncoder()
le.fit(X_train['pickup_borough'].astype(str))
X_train['pickup_borough'] = le.transform(X_train['pickup_borough'].astype(str))
X_test['pickup_borough'] = le.transform(X_test['pickup_borough'].astype(str))



le.fit(X_train['dropoff_borough'].astype(str))
X_train['dropoff_borough'] = le.transform(X_train['dropoff_borough'].astype(str))
X_test['dropoff_borough'] = le.transform(X_test['dropoff_borough'].astype(str))
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
lm_rmse=np.sqrt(mean_squared_error(y_pred, y_test))
print("RMSE for Linear Regression is ",lm_rmse)

def MAPE(y_true,y_pred):
    mape = np.mean(np.abs((y_true - y_pred)/ y_true))
    return mape
```

```
def MAE(y_true,y_pred):
        mae = np.mean(np.abs((y_true - y_pred)))
        return mae

MAPE(y_test,y_pred)
MAE(y_test,y_pred)
model = sm.OLS(y_train,X_train.astype(float)).fit()
model.summary()

# Decision tree Model

dt  = DecisionTreeRegressor(max_depth=2)
dt.fit(X_train,y_train)

dt_pred= dt.predict(X_test)
dt_rmse=np.sqrt(mean_squared_error(dt_pred, y_test))
print("RMSE for Random Forest is ",dt_rmse)

MAPE(y_test,dt_pred)
MAE(y_test,dt_pred)

# Random Forest Model

rf = RandomForestRegressor(n_estimators = 1000, random_state = 883,n_jobs=-1)
rf.fit(X_train,y_train)

rf_pred= rf.predict(X_test)
rf_rmse=np.sqrt(mean_squared_error(rf_pred, y_test))
print("RMSE for Random Forest is ",rf_rmse)
```

```
MAPE(y_test,rf_pred)
MAE(y_test,rf_pred)
```

Chapter 7: R Code

```
rm(list = ls())
setwd("C:/Users/Bhavesh/Desktop/Data scientist/Project/Cab Fare ")
# #loading Libraries
x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
      "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','stats')
#load Packages
lapply(x, require, character.only = TRUE)
rm(x)



# The details of data attributes in the dataset are as follows:
# pickup_datetime - timestamp value indicating when the cab ride started.
# pickup_longitude - float for longitude coordinate of where the cab ride started.
# pickup_latitude - float for latitude coordinate of where the cab ride started.
# dropoff_longitude - float for longitude coordinate of where the cab ride ended.
# dropoff_latitude - float for latitude coordinate of where the cab ride ended.
# passenger_count - an integer indicating the number of passengers in the cab ride.



# loading datasets
```

```
train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
test = read.csv("test.csv")
test_pickup_datetime = test["pickup_datetime"]
# Structure of data
str(train)
str(test)
summary(train)
summary(test)
head(train,5)
head(test,5)



############                    Exploratory Data Analysis
########################
# Changing the data types of variables
train$fare_amount = as.numeric(as.character(train$fare_amount))
train$passenger_count=round(train$passenger_count)
```

Removing values which are not within desired range(outlier) depending upon basic understanding of dataset.

1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve and also cannot be 0. So we will remove these fields.

```
train[which(train$fare_amount < 0 ),]
nrow(train[which(train$fare_amount < 0 ),])
train = train[-which(train$fare_amount < 0 ),]



#2.Passenger_count variable
for (i in seq(4,11,by=1)){
  print(paste('passenger_count above ' ,i,nrow(train[which(train$passenger_count > i
),])))
```

```
}
# so 20 observations of passenger_count is consistenly above from 6,7,8,9,10 passenger_counts, let's check them.
train[which(train$passenger_count > 6 ),]
# Also we need to see if there are any passenger_count==0
train[which(train$passenger_count <1 ),]
nrow(train[which(train$passenger_count <1 ),])
# We will remove these 58 observations and 20 observation which are above 6 value because a cab cannot hold these number of passengers.
train = train[-which(train$passenger_count < 1 ),]
train = train[-which(train$passenger_count > 6),]
# 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.Removing which does not satisfy these ranges
print(paste('pickup_longitude above 180=',nrow(train[which(train$pickup_longitude >180 ),])))
print(paste('pickup_longitude above -180=',nrow(train[which(train$pickup_longitude < -180 ),])))
print(paste('pickup_latitude above 90=',nrow(train[which(train$pickup_latitude > 90 ),])))
print(paste('pickup_latitude above -90=',nrow(train[which(train$pickup_latitude < -90 ),])))
print(paste('dropoff_longitude above 180=',nrow(train[which(train$dropoff_longitude > 180 ),])))
print(paste('dropoff_longitude above -180=',nrow(train[which(train$dropoff_longitude < -180 ),])))
print(paste('dropoff_latitude above -90=',nrow(train[which(train$dropoff_latitude < -90 ),])))
print(paste('dropoff_latitude above 90=',nrow(train[which(train$dropoff_latitude > 90 ),])))
# There's only one outlier which is in variable pickup_latitude.So we will remove it with nan.
# Also we will see if there are any values equal to 0.
nrow(train[which(train$pickup_longitude == 0 ),])
```

```
 nrow(train[which(train$pickup_latitude == 0 ),])
 nrow(train[which(train$dropoff_longitude == 0 ),])
 nrow(train[which(train$pickup_latitude == 0 ),])
 # there are values which are equal to 0. we will remove them.
 train = train[-which(train$pickup_latitude > 90),]
 train = train[-which(train$pickup_longitude == 0),]
 train = train[-which(train$dropoff_longitude == 0),]

 # Make a copy
 df=train
 # train=df
############## Missing Value Analysis ##############
 missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
 missing_val$Columns = row.names(missing_val)
 names(missing_val)[1] =  "Missing_percentage"
 missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(train)) *
 100
 missing_val = missing_val[order(-missing_val$Missing_percentage),]
 row.names(missing_val) = NULL
 missing_val = missing_val[,c(2,1)]
 missing_val
 unique(train$passenger_count)
 unique(test$passenger_count)
 train[,'passenger_count'] = factor(train[,'passenger_count'], labels=(1:6))
test[,'passenger_count'] = factor(test[,'passenger_count'], labels=(1:6))
 # 1.For Passenger_count:
 # KNN = 1
 train$passenger_count[1000]
 train$passenger_count[1000] = NA
```

```
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}
# kNN Imputation
train = knnImputation(train, k = 181)
train$fare_amount[1000]
train$passenger_count[1000]
sapply(train, sd, na.rm = TRUE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.661952       4635.700531          2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
# 2.613305          2.710835          2.632400
# passenger_count
# 1.263859
sum(is.na(train))
str(train)
summary(train)
df1=train
# train=df1
#######################                    Outlier Analysis
###################
```

We Will do Outlier Analysis only on Fare_amount just for now and we will do outlier analysis after feature engineering laitudes and longitudes.

```
# Boxplot for fare_amount
pl1 = ggplot(train,aes(x = factor(passenger_count),y = fare_amount))
pl1 + geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1,
notch=FALSE)+ylim(0,100)
```

```
# Replace all outliers with NA and impute
 vals = train[,"fare_amount"] %in% boxplot.stats(train[,"fare_amount"])$out
 train[which(vals),"fare_amount"] = NA
 #lets check the NA's
 sum(is.na(train$fare_amount))
 #Imputing with KNN
 train = knnImputation(train,k=3)
# lets check the missing values
 sum(is.na(train$fare_amount))
 str(train)
 df2=train
 # train=df2
 ###################              Feature Engineering
 ############################
 # 1.Feature Engineering for timestamp variable
# we will derive new features from pickup_datetime variable
 # new features will be year,month,day_of_week,hour
 #Convert pickup_datetime from factor to date time
 train$pickup_date = as.Date(as.character(train$pickup_datetime))
 train$pickup_weekday = as.factor(format(train$pickup_date,"%u"))# Monday = 1
 train$pickup_mnth = as.factor(format(train$pickup_date,"%m"))
 train$pickup_yr = as.factor(format(train$pickup_date,"%Y"))
 pickup_time = strptime(train$pickup_datetime,"%Y-%m-%d %H:%M:%S")
train$pickup_hour = as.factor(format(pickup_time,"%H"))
#Add same features to test set
 test$pickup_date = as.Date(as.character(test$pickup_datetime))
 test$pickup_weekday = as.factor(format(test$pickup_date,"%u"))# Monday = 1
 test$pickup_mnth = as.factor(format(test$pickup_date,"%m"))
```

```
 test$pickup_yr = as.factor(format(test$pickup_date,"%Y"))
 pickup_time = strptime(test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
 test$pickup_hour = as.factor(format(pickup_time,"%H"))


sum(is.na(train))# there was 1 'na' in pickup_datetime which created na's in above feature engineered variables.
train = na.omit(train) # we will remove that 1 row of na's


train = subset(train,select = -c(pickup_datetime,pickup_date))
 test = subset(test,select = -c(pickup_datetime,pickup_date))


 # 2.Calculate the distance travelled using longitude and latitude
 deg_to_rad = function(deg){
   (deg * pi) / 180
 }
 haversine = function(long1,lat1,long2,lat2){
   #long1rad = deg_to_rad(long1)
   phi1 = deg_to_rad(lat1)
   #long2rad = deg_to_rad(long2)
   phi2 = deg_to_rad(lat2)
   delphi = deg_to_rad(lat2 - lat1)
   dellamda = deg_to_rad(long2 - long1)

   a = sin(delphi/2) * sin(delphi/2) + cos(phi1) * cos(phi2) *
     sin(dellamda/2) * sin(dellamda/2)

   c = 2 * atan2(sqrt(a),sqrt(1-a))
   R = 6371e3
   R * c / 1000 #1000 is used to convert to meters
```

```
 }
 # Using haversine formula to calculate distance fr both train and test
 train$dist =
 haversine(train$pickup_longitude,train$pickup_latitude,train$dropoff_longitude,train$dr
 opoff_latitude)
 test$dist =
 haversine(test$pickup_longitude,test$pickup_latitude,test$dropoff_longitude,test$drop
 off_latitude)


# We will remove the variables which were used to feature engineer new variables
 train = subset(train,select = -
 c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
 test = subset(test,select = -
 c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))


str(train)
 summary(train)


################                    Feature selection
####################
 numeric_index = sapply(train,is.numeric) #selecting only numeric


numeric_data = train[,numeric_index]
cnames = colnames(numeric_data)
 #Correlation analysis for numeric variables
 corrgram(train[,numeric_index],upper.panel=panel.pie, main = "Correlation Plot")



#ANOVA for categorical variables with target numeric variable
```

```
#aov_results = aov(fare_amount ~ passenger_count * pickup_hour *
pickup_weekday,data = train)

 aov_results = aov(fare_amount ~ passenger_count + pickup_hour + pickup_weekday
 + pickup_mnth + pickup_yr,data = train)

summary(aov_results)


# pickup_weekdat has p value greater than 0.05

 train = subset(train,select=-pickup_weekday)


#remove from test set

 test = subset(test,select=-pickup_weekday)


#####################################          Feature Scaling
#################################################

 #Normality check

 # qqnorm(train$fare_amount)

 # histogram(train$fare_amount)

 library(car)

 # dev.off()

 par(mfrow=c(1,2))

 qqPlot(train$fare_amount)                    # qqPlot, it has a x values derived from
gaussian distribution, if data is distributed normally then the sorted data points should
lie very close to the solid reference line

 truehist(train$fare_amount)                  # truehist() scales the counts to give an
estimate of the probability density.

 lines(density(train$fare_amount))  # Right skewed      # lines() and density() functions
 to overlay a density plot on histogram



#Normalisation
```

```
print('dist')
 train[,'dist'] = (train[,'dist'] - min(train[,'dist']))/
   (max(train[,'dist'] - min(train[,'dist'])))


# #check multicollearity
 # library(usdm)
 # vif(train[,-1])
 #
 # vifcor(train[,-1], th = 0.9)


###################### Splitting train into train and validation subsets
####################
 set.seed(1000)
 tr.idx = createDataPartition(train$fare_amount,p=0.75,list = FALSE) # 75% in trainin
 and 25% in Validation Datasets
 train_data = train[tr.idx,]
 test_data = train[-tr.idx,]


rmExcept(c("test","train","df",'df1','df2','df3','test_data','train_data','test_pickup_datetime')
)
 ####################Model Selection#################
 #Error metric used to select model is RMSE


##############          Linear regression          ##################
 lm_model = lm(fare_amount ~.,data=train_data)
summary(lm_model)
str(train_data)
 plot(lm_model$fitted.values,rstandard(lm_model),main = "Residual plot",
      xlab = "Predicted values of fare_amount",
      ylab = "standardized residuals")
```

```
lm_predictions = predict(lm_model,test_data[,2:6])


qplot(x = test_data[,1], y = lm_predictions, data = test_data, color = I("blue"), geom =
"point")

regr.eval(test_data[,1],lm_predictions)

 # mae        mse       rmse       mape

 # 3.5303114 19.3079726  4.3940838  0.4510407



##############                    Decision Tree          ######################


Dt_model = rpart(fare_amount ~ ., data = train_data, method = "anova")


summary(Dt_model)

 #Predict for new test cases

 predictions_DT = predict(Dt_model, test_data[,2:6])


qplot(x = test_data[,1], y = predictions_DT, data = test_data, color = I("blue"), geom =
"point")


regr.eval(test_data[,1],predictions_DT)

 # mae       mse      rmse      mape

 # 1.8981592 6.7034713 2.5891063 0.2241461



##############                    Random forest          ######################
```

```
 rf_model = randomForest(fare_amount ~.,data=train_data)

summary(rf_model)

rf_predictions = predict(rf_model,test_data[,2:6])

qplot(x = test_data[,1], y = rf_predictions, data = test_data, color = I("blue"), geom =
"point")

regr.eval(test_data[,1],rf_predictions)

 # mae        mse       rmse       mape

 # 1.9053850 6.3682283 2.5235349 0.2335395
```