**Chapter 1: Introduction**

* 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**

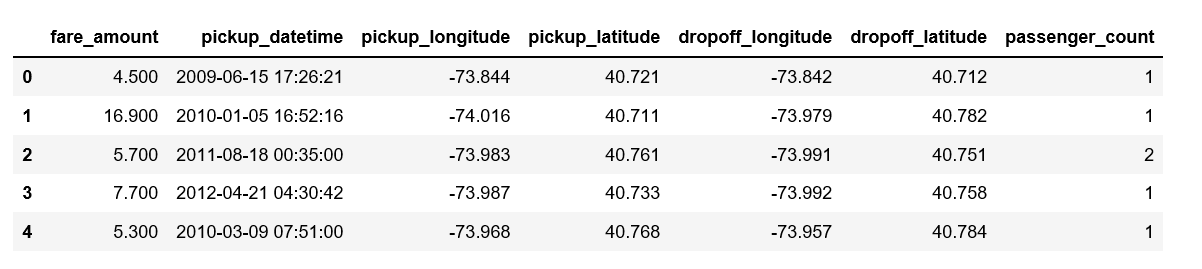


Table 1.1 Cab Fare Sample Data

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

### 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**

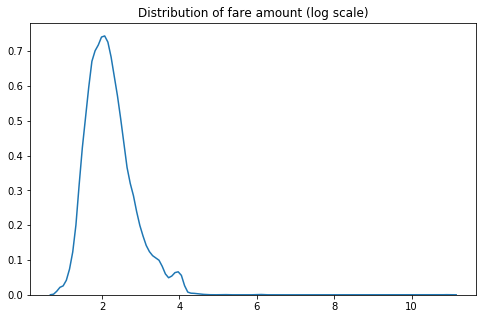
In this section, we will discuss various steps used to clean the data and understand the relationship between variables and use this understanding to create better features.

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 .

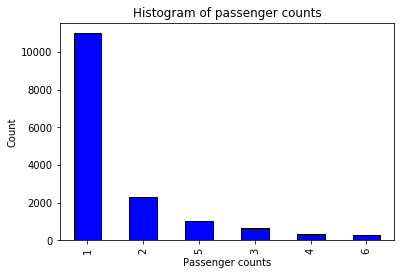


Also, fare amount follows long tail distribution.To understand the distribution of fare amount better we take a log transformation after removing the negative fares- this makes the distribution close to normal. Since we saw above that fare amount is highly skewed,let us take log transformation of the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



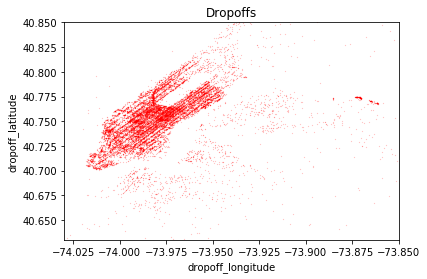
**2 Distribution of Passenger Count**

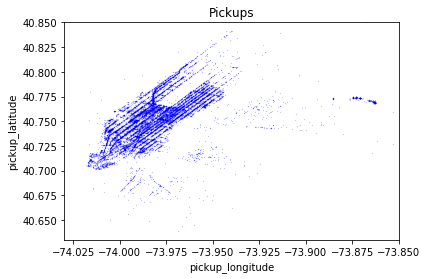
We have seen from the data that we have passenger count more then 6 also which is not possible so we have set the max passenger count to 6 and plot the histogram for passenger count and found we hve maximum passenger as 1 then 2 and so on.



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.

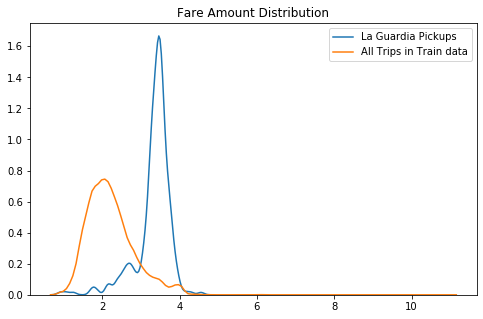


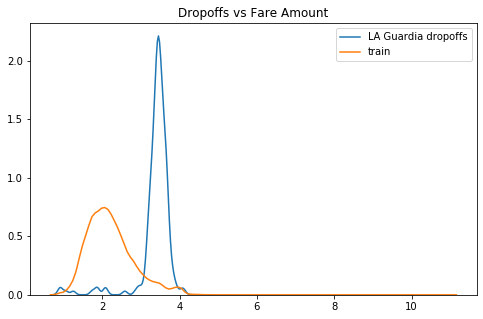


We can see that there is a cluster of pickups near La Guardia Airport.We then looked at what is the average fare amount for pickups and drop offs to La Guardia Airport, compared to all trips in the train data .

Number of Trips with Pickups from LA Guardia 324

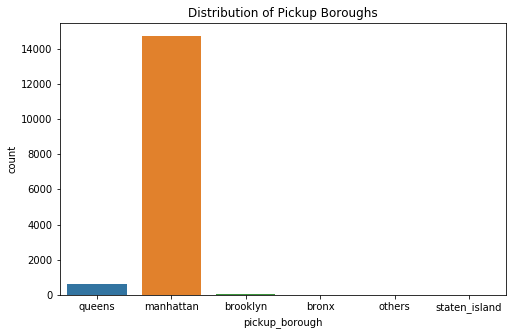
Number of Trips with Dropoffs to LA Guardia 206

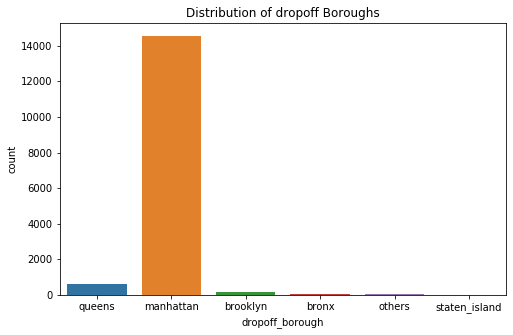


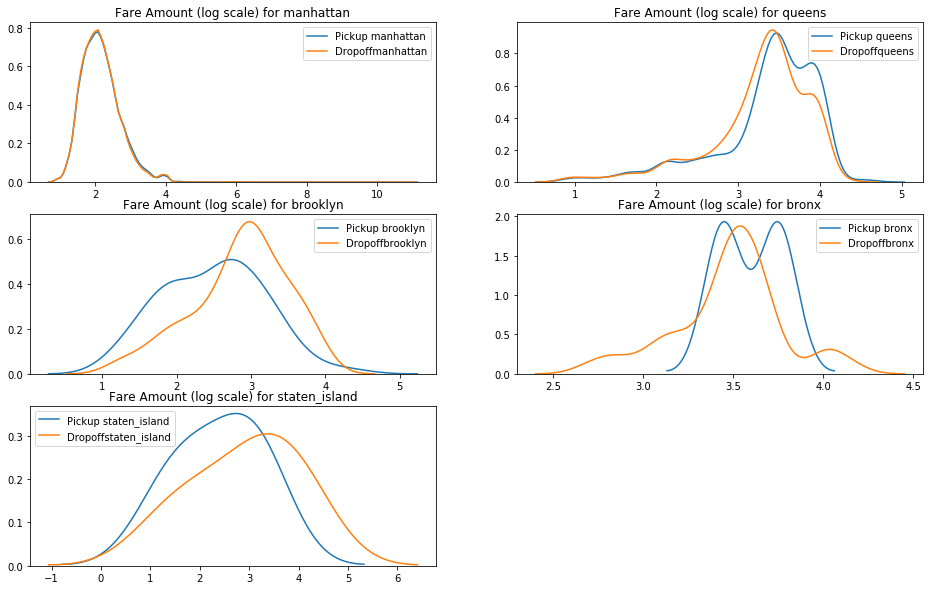


So after plotting the same we observed that fare was higher for airport trips. Based on this observations we created features to check whether a pickup or a drop-off was to airport in LaGuardia

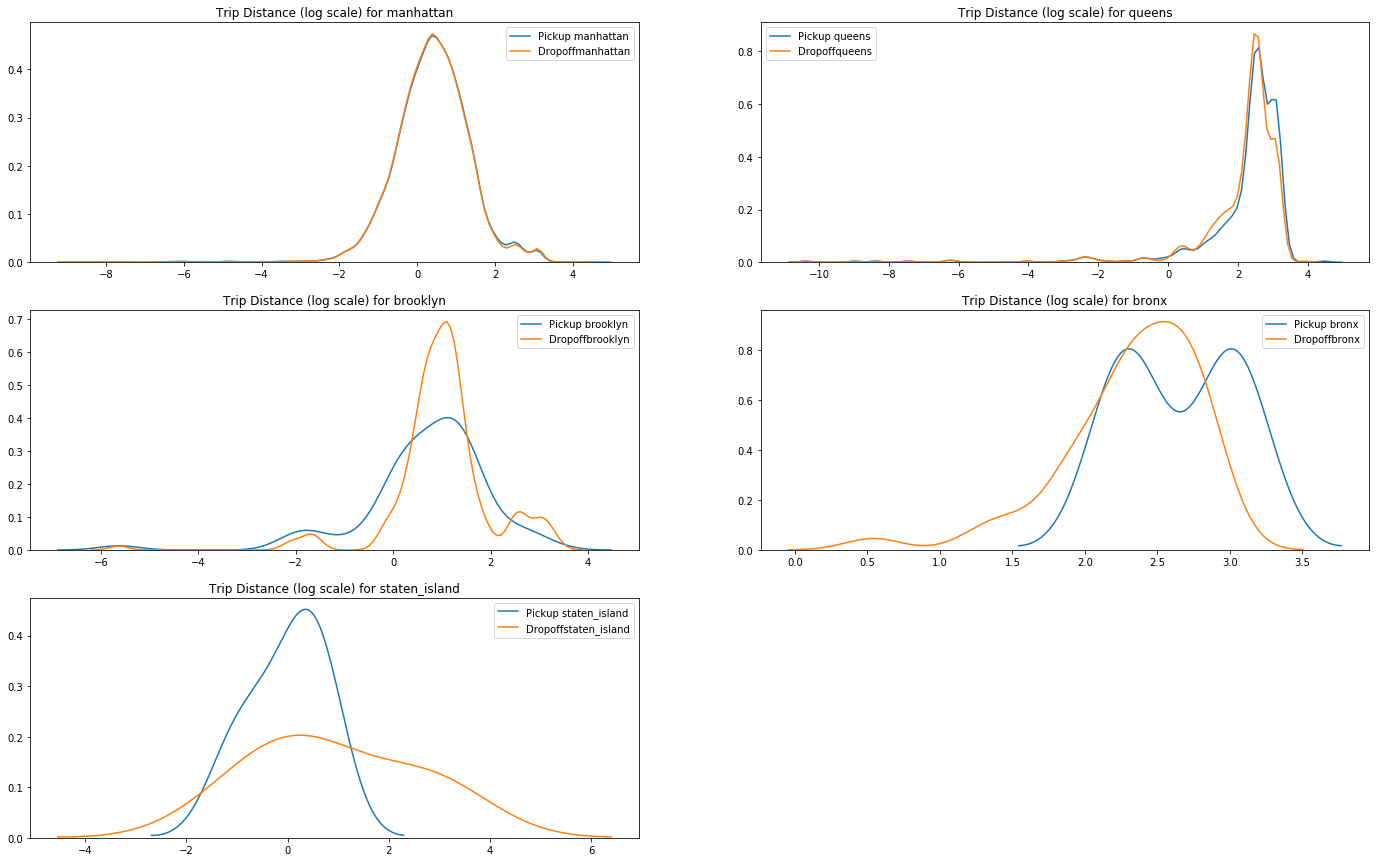
The next step was to check whether our hypothesis of fare from certain neighborhoods are higher than the rest, based on the 5 Boroughs New York city is divided — Manhattan, Queens, Brooklyn, Staten Island and Bronx, each pickup and drop off location was grouped into these 5 neighborhoods.







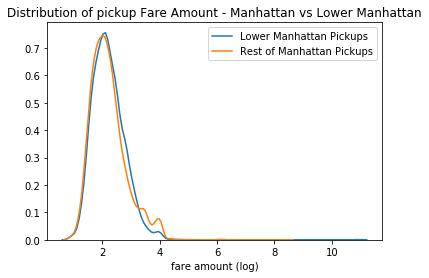
. And yes our hypothesis was right- except for Manhattan which had most of the pickups and drop offs, for every other neighborhood, there was a difference in the pickup and drop off fare distribution. Also, Queens had a higher mean pickup fare and dropoff fare compared to other neighborhoods. We can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

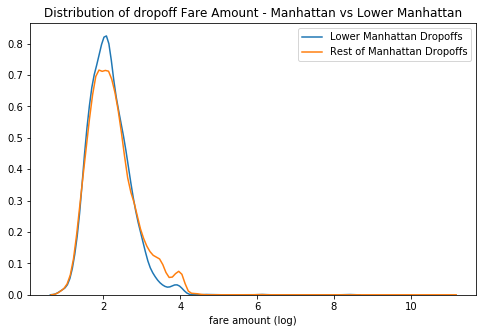


Dropoffs to Bronx and queens are long trips. In Manhattan the pickup and dropoffs fare amount has similar distribution. Let us add a field, is\_lower\_manhattan as we had seen above that dropoffs to lower manhattan had higher trip distance but lower fare

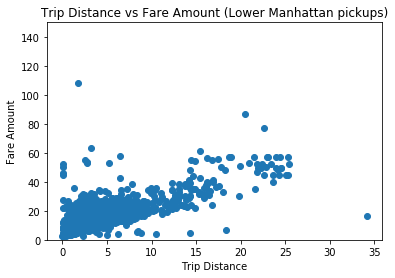
So After adding fields as is\_pickup\_lower\_manhattan and is\_dropoff\_lower\_manhattan ,

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 .





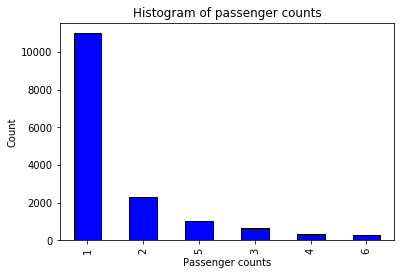


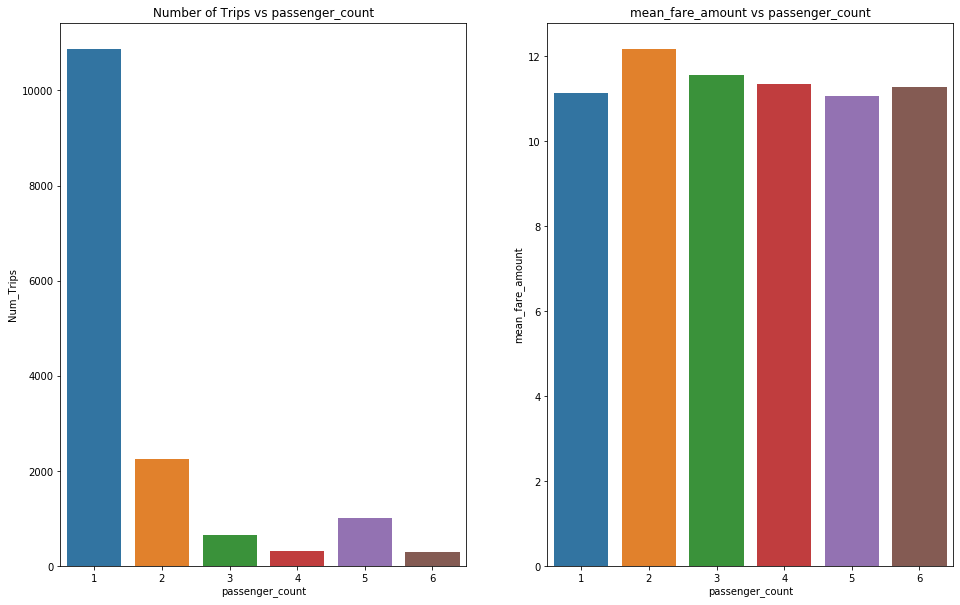


The distribution of trip distance and fare amount for Lower Manhattan pickups and dropoffs is very different. Also, slope of linear realtionship for pickups for Lower Manhattan is higher than that for Rest of Manhattan

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.



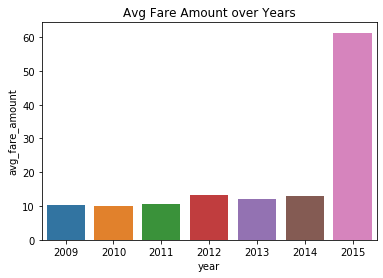


**4. Distribution of Pickup date time**

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.

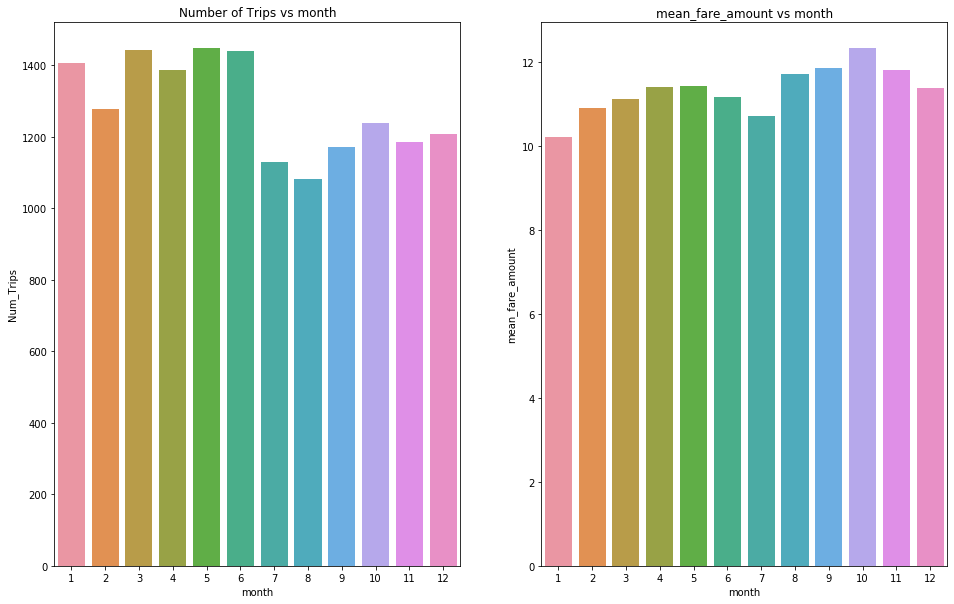
We have drawn the bar chart between average fare amount and consecutive years.



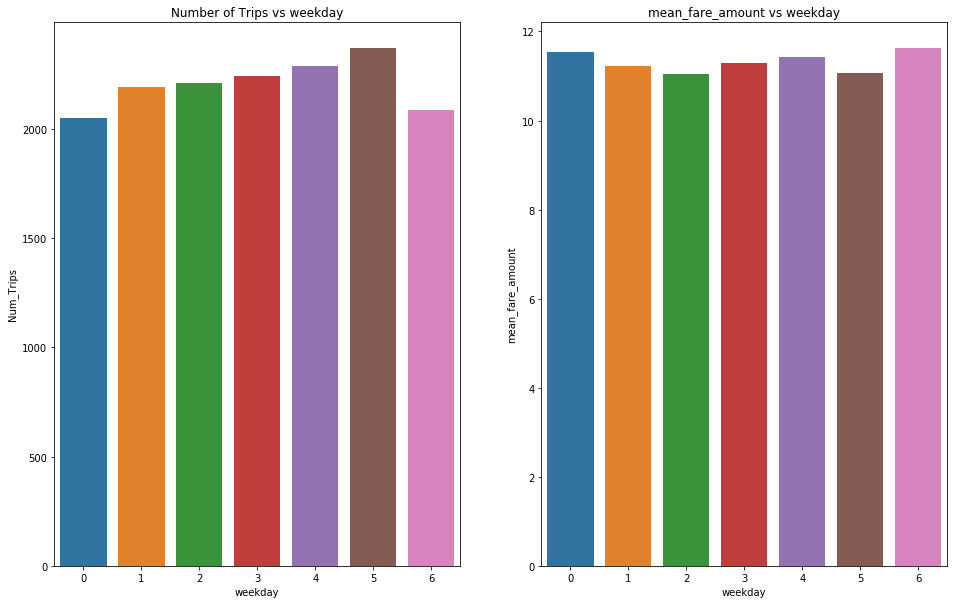
As expected, over years the average taxi fare has increased.

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

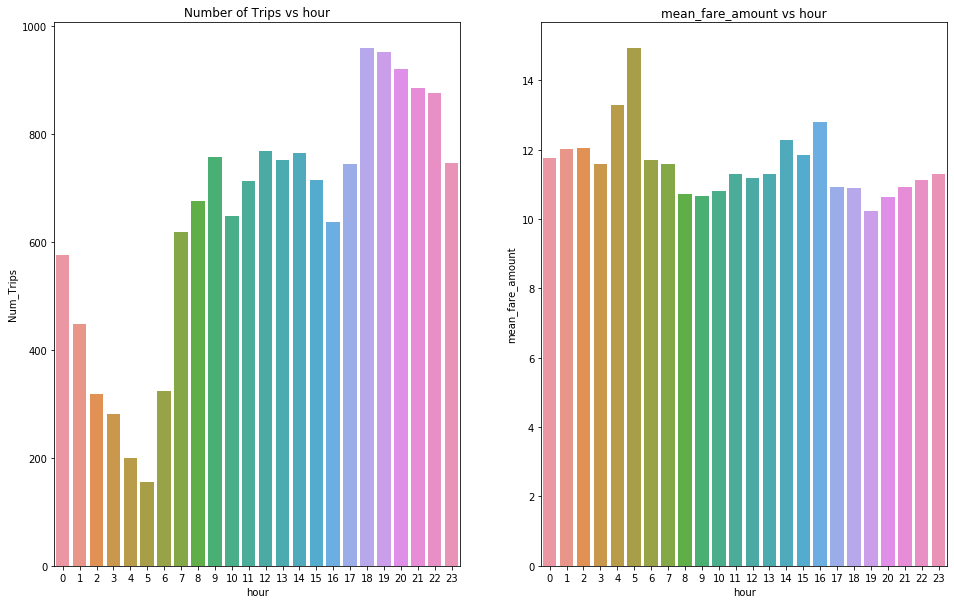
And found :-



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)

### 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

### 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. The MAPE for this decision tree is 26.78%. Hence the accuracy for this model is 74.28%.

RMSE for decision tree 5.88

### 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](https://www.statisticshowto.datasciencecentral.com/residual/) ([prediction errors](https://www.statisticshowto.datasciencecentral.com/prediction-error-definition/)). 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](https://www.statisticshowto.datasciencecentral.com/line-of-best-fit/). Root mean square error is commonly used in climatology, forecasting, and [regression analysis](https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/regression-analysis/) to verify experimental results.

The formula is:  
[rmse](https://www.statisticshowto.datasciencecentral.com/wp-content/uploads/2016/10/rmse.png)  
**Where**:

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

### LinearRegressionModel:sMAE = 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'].values),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),label='Pickup '+ key)

sns.kdeplot(np.log(dataset1.loc[dataset1['dropoff\_borough']==key,'distance'].values),label='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\_amount'].values),label='Lower Manhattan Pickups')

sns.kdeplot(np.log(manhattan.loc[manhattan['is\_pickup\_lower\_manhattan']==0,'fare\_amount'].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$dropoff\_latitude)

test$dist = haversine(test$pickup\_longitude,test$pickup\_latitude,test$dropoff\_longitude,test$dropoff\_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