**Project Name – Cab Fare Prediction**

**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.

**Data Set :**

1) train\_cab.zip

2) test.zip

**Missing Values** : Yes

**Chapter 1**

**Introduction**

With Cab rental services growing exponentially in the past decade, the ease of the public over using the cab services provide a greater value for using these rental services.

**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**

Understanding the data is important for solving the business problem and applying scientific models to predict the data

|  |  |
| --- | --- |
| **Variables** | **Description** |
| **fare\_amount** | Fare amount |
| **pickup\_datetime** | Cab pickup date with time |
| **pickup\_longitude** | Pickup location longitude |
| **pickup\_latitude** | Pickup location latitude |
| **dropoff\_longitude** | Drop location longitude |
| **dropoff\_latitude** | Drop location latitude |
| **passenger\_count** | Number of passengers sitting in the cab |

|  |  |
| --- | --- |
| Missing Values: Yes |  |

Below code is used to copy the train\_data and verify the data provided for the problem statement.

INSERT PYTHON CODE SNIPPET

From the above analysis we find that we have,

* 7 Variables
* 16067 Observations

Among these 7 Variables 6 are independent variables and 1 is dependent variable which is ***fare\_amount***

**Notes: Fare\_amount is a continuous variable.**

**Chapter 2**

**Data Pre-Processing**

Before building a model to predict the fare amount from historical data that we have , it is important for us to do the Exploratory Data Analysis so that we can transform the raw data into an understandable format. Raw data cannot be fed directly into the model as it would cause errors in the prediction. Hence it is important we pre process the data before sending it through a model.

Below are the steps we have to perform first in Data Preprocessing

Here are the steps that are being followed for the given data.

1. Data cleaning and exploration

2. Missing value analysis

3. Outlier Analysis

4. Feature selection

5. Feature scaling

3.1 **Data exploration and Cleaning**

Below observations can be made from the given data,

* The Variable **pickup\_datetime** contains date and time of the passenger in the cab. Hence this can be further broken down to extract more variables from this as stated below.
  + 1. Year
    2. Month
    3. Day
    4. Hour
    5. Minute
    6. Seconds

With this approach 6 variables are created which are **Year, Month, Date, Hour, Minute** and **Seconds.**

Also we will be removing the **pickup\_datetime** variable as we no longer need this

* We have the four variables, **pickup\_longitude, pickup\_latitude , dropoff\_longitude, dropoff\_latitude** . Using these 4 variables we can combine into a one variable called distance , by calculating the difference between pickup location and drop off location.

The Haversine ('half-versed-sine') formula was published by R.W. Sinnott in 1984, although it has been known for much longer. At that time computational precision was lower than today (15 digits precision). With current precision, the spherical law of cosines formula appears to give equally good results down to very small distances.

Since this project is related to cab pick up point of a passenger within a city hence the distance will be smaller and we are going to use this function using geosphere package available in R

* From the data we have obtained so far we are going to further pre process the data.

From the previous section we can see that the fare\_amount and distance.

Once we change the data type we now move on to find the missing values and impute those values.

**Missing Values**

First we calculate the no of NA values in the data\_set. We find that only 85 is the missing values out of 16067 observations. Since the no of NA values is 85 only, it is very negligible compared to the 16067 observations, hence we are not going to impute the missing values and we will be ignoring them.

**Outlier Analysis**

In order to find the Outliers in a data, we first plot the boxplot for the dataset to find the outliers.

From the above diagram we see that there are few outliers present in the data set for some of the variables i.e. **fare\_amount** and **distance**. For the remaining variables of time like **Year**, **Month**,**Date**, **Hour**, **Minute** and **Seconds** there are not outliers present.

Hence we go for removing these outliers and replacing them with the value of NA’s.

From the co relation plot we find that the fare\_amount and distance variables are co related which is true because as the distance increases for the cab ride, the fare for the ride also increases. Since we need both the distance and fare\_amount variables we will retain the distance variable to predict the fare\_amount for the test data.

* Feature Scaling
* We first find how the data is distributed across different variables that we have like fare\_amount, passenger\_count, distance

From the histogram plotted we can verify the data is not normally distributed hence we go for normalisation.

**Chapter 3**

**Modelling**

Since our target variable to predict fare of the cab (fare\_amount) is continous variable we will be evaluating the model using regression algorithms to predict the data which are as following,

Linear Regression

Decision Tree

Random Forest

First we split the data into train and test using sampling method and apply the regression models for the test data to predict the values and compare with the actual values.

Below is the comparison model we have arrived at after evaluating all the model and its parameters

|  |  |  |
| --- | --- | --- |
| Model Name | MAPE | Accuracy |
| Linear Regression | 18.42129 | 81.578706 |
| Decision Tree | 20.96588 | 79.034118 |
| Random Forest | 18.59785 | 81.402151 |

For the Error Metrics we have calcualted MAPE only because RMSE is widely used in time series data.

From the above table we can see that the Random\_Forest Model and Linear Regression is having almost similar Accuracy rate hence we will be finalizing one of them, and I have chosen the Random Forest Model for predicting the cab fare for new test data.

Below is the code written for arriving at the decision of Model selection along with Data Exploratory Analysis.

In [1]:

#Set working directory

import os

os.chdir("C:/Users/VB018797/Documents/Cab\_Fare\_Python")

In [2]:

#Get the current working directory##

os.getcwd()

Out[2]:

'C:\\Users\\VB018797\\Documents\\Cab\_Fare\_Python'

In [3]:

#Import libraries

import os

import pandas as pd

import numpy as np

In [4]:

#Load the data from csv

cab\_train = pd.read\_csv("train\_cab.csv", sep=',')

In [5]:

cab\_train.head(5)

Out[5]:

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 |
| **1** | 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 |
| **2** | 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 |
| **3** | 7.7 | 2012-04-21 04:30:42 UTC | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 |
| **4** | 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 |

In [6]:

######Find the shape of the data

cab\_train.shape

Out[6]:

(16067, 7)

In [7]:

#Data types of the variables

cab\_train.dtypes

Out[7]:

fare\_amount object

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count float64

dtype: object

In [8]:

####Create new variables from pickup\_datetime for reading Year, Date, Month, Hour, Minute, Seconds and use str function

cab\_train['Year']= cab\_train.pickup\_datetime.str[0:4]

cab\_train['Month']= cab\_train.pickup\_datetime.str[5:7]

cab\_train['Date']= cab\_train.pickup\_datetime.str[8:10]

cab\_train['Hour']= cab\_train.pickup\_datetime.str[11:13]

cab\_train['Minute']= cab\_train.pickup\_datetime.str[14:16]

cab\_train['Seconds'] = cab\_train.pickup\_datetime.str[17:19]

In [9]:

###Verify the shape

cab\_train.dtypes

Out[9]:

fare\_amount object

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count float64

Year object

Month object

Date object

Hour object

Minute object

Seconds object

dtype: object

In [10]:

###Convert new variables created to type numeric since they are of type object

cab\_train['Year'] = pd.to\_numeric(cab\_train['Year'],errors = "coerce")

cab\_train['Month'] = pd.to\_numeric(cab\_train['Month'],errors = "coerce")

cab\_train['Date'] = pd.to\_numeric(cab\_train['Date'],errors = "coerce")

cab\_train['Hour'] = pd.to\_numeric(cab\_train['Hour'],errors = "coerce")

cab\_train['Minute'] = pd.to\_numeric(cab\_train['Minute'],errors = "coerce")

cab\_train['Seconds'] = pd.to\_numeric(cab\_train['Seconds'],errors = "coerce")

In [11]:

cab\_train.dtypes

Out[11]:

fare\_amount object

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count float64

Year int64

Month float64

Date float64

Hour float64

Minute float64

Seconds float64

dtype: object

In [12]:

####Convert fare\_amount variable to numeric

cab\_train['fare\_amount'] = pd.to\_numeric(cab\_train['fare\_amount'],errors = "coerce")

In [13]:

#Remove pickup\_datetime variable as we do not need it

cab\_train.drop(['pickup\_datetime'], axis =1)

Out[13]:

|  | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 | 2009 | 6.0 | 15.0 | 17.0 | 26.0 | 21.0 |
| **1** | 16.9 | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 | 2010 | 1.0 | 5.0 | 16.0 | 52.0 | 16.0 |
| **2** | 5.7 | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 | 2011 | 8.0 | 18.0 | 0.0 | 35.0 | 0.0 |
| **3** | 7.7 | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 | 2012 | 4.0 | 21.0 | 4.0 | 30.0 | 42.0 |
| **4** | 5.3 | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 | 2010 | 3.0 | 9.0 | 7.0 | 51.0 | 0.0 |
| **5** | 12.1 | -74.000964 | 40.731630 | -73.972892 | 40.758233 | 1.0 | 2011 | 1.0 | 6.0 | 9.0 | 50.0 | 45.0 |
| **6** | 7.5 | -73.980002 | 40.751662 | -73.973802 | 40.764842 | 1.0 | 2012 | 11.0 | 20.0 | 20.0 | 35.0 | 0.0 |
| **7** | 16.5 | -73.951300 | 40.774138 | -73.990095 | 40.751048 | 1.0 | 2012 | 1.0 | 4.0 | 17.0 | 22.0 | 0.0 |
| **8** | NaN | -74.006462 | 40.726713 | -73.993078 | 40.731628 | 1.0 | 2012 | 12.0 | 3.0 | 13.0 | 10.0 | 0.0 |
| **9** | 8.9 | -73.980658 | 40.733873 | -73.991540 | 40.758138 | 2.0 | 2009 | 9.0 | 2.0 | 1.0 | 11.0 | 0.0 |
| **10** | 5.3 | -73.996335 | 40.737142 | -73.980721 | 40.733559 | 1.0 | 2012 | 4.0 | 8.0 | 7.0 | 30.0 | 50.0 |
| **11** | 5.5 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 3.0 | 2012 | 12.0 | 24.0 | 11.0 | 24.0 | 0.0 |
| **12** | 4.1 | -73.991601 | 40.744712 | -73.983081 | 40.744682 | 2.0 | 2009 | 11.0 | 6.0 | 1.0 | 4.0 | 3.0 |
| **13** | 7.0 | -74.005360 | 40.728867 | -74.008913 | 40.710907 | 1.0 | 2013 | 7.0 | 2.0 | 19.0 | 54.0 | 0.0 |
| **14** | 7.7 | -74.001821 | 40.737547 | -73.998060 | 40.722788 | 2.0 | 2011 | 4.0 | 5.0 | 17.0 | 11.0 | 5.0 |
| **15** | 5.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.0 | 2013 | 11.0 | 23.0 | 12.0 | 57.0 | 0.0 |
| **16** | 12.5 | -73.986430 | 40.760465 | -73.988990 | 40.737075 | 1.0 | 2014 | 2.0 | 19.0 | 7.0 | 22.0 | 0.0 |
| **17** | 5.3 | -73.981060 | 40.737690 | -73.994177 | 40.728412 | 1.0 | 2009 | 7.0 | 22.0 | 16.0 | 8.0 | 0.0 |
| **18** | 5.3 | -73.969505 | 40.784843 | -73.958732 | 40.783357 | 1.0 | 2010 | 7.0 | 7.0 | 14.0 | 52.0 | 0.0 |
| **19** | 4.0 | -73.979815 | 40.751902 | -73.979446 | 40.755481 | 1.0 | 2014 | 12.0 | 6.0 | 20.0 | 36.0 | 22.0 |
| **20** | 10.5 | -73.985382 | 40.747858 | -73.978377 | 40.762070 | 1.0 | 2010 | 9.0 | 7.0 | 13.0 | 18.0 | 0.0 |
| **21** | 11.5 | -73.957954 | 40.779252 | -73.961250 | 40.758787 | 1.0 | 2013 | 2.0 | 12.0 | 12.0 | 15.0 | 46.0 |
| **22** | 4.5 | -73.991707 | 40.770505 | -73.985459 | 40.763671 | 1.0 | 2009 | 8.0 | 6.0 | 18.0 | 17.0 | 23.0 |
| **23** | 4.9 | -74.000632 | 40.747473 | -73.986672 | 40.740577 | 1.0 | 2010 | 12.0 | 6.0 | 12.0 | 29.0 | 0.0 |
| **24** | 6.1 | -73.969622 | 40.756973 | -73.981152 | 40.759712 | 1.0 | 2009 | 12.0 | 10.0 | 15.0 | 37.0 | 0.0 |
| **25** | 7.3 | -73.991875 | 40.754437 | -73.977230 | 40.774323 | 3.0 | 2011 | 6.0 | 21.0 | 16.0 | 15.0 | 0.0 |
| **26** | NaN | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.0 | 2011 | 2.0 | 7.0 | 20.0 | 1.0 | 0.0 |
| **27** | 4.5 | -73.988893 | 40.760160 | -73.986445 | 40.757857 | 3.0 | 2011 | 6.0 | 28.0 | 19.0 | 47.0 | 0.0 |
| **28** | 9.3 | -73.989258 | 40.690835 | -74.004133 | 40.725690 | 1.0 | 2012 | 5.0 | 4.0 | 6.0 | 11.0 | 20.0 |
| **29** | 4.5 | -73.981020 | 40.737760 | -73.980668 | 40.730497 | 2.0 | 2013 | 8.0 | 11.0 | 0.0 | 52.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **16037** | 6.5 | -73.992618 | 40.723878 | -73.977073 | 40.733778 | 1.0 | 2012 | 2.0 | 27.0 | 21.0 | 40.0 | 50.0 |
| **16038** | 5.7 | -73.990336 | 40.718973 | -73.956060 | 40.713974 | 1.0 | 2010 | 8.0 | 31.0 | 10.0 | 43.0 | 42.0 |
| **16039** | 12.9 | -73.936462 | 40.794292 | -73.948747 | 40.779097 | 5.0 | 2010 | 12.0 | 11.0 | 16.0 | 25.0 | 0.0 |
| **16040** | 6.5 | -73.980597 | 40.744267 | -73.979330 | 40.731205 | 1.0 | 2014 | 6.0 | 16.0 | 0.0 | 5.0 | 19.0 |
| **16041** | 11.0 | -73.983610 | 40.747090 | -73.961310 | 40.770980 | 1.0 | 2014 | 11.0 | 17.0 | 21.0 | 53.0 | 0.0 |
| **16042** | 8.5 | -73.991425 | 40.749832 | -74.000107 | 40.727898 | 2.0 | 2015 | 4.0 | 6.0 | 21.0 | 53.0 | 6.0 |
| **16043** | 8.5 | -73.973961 | 40.764055 | -73.986807 | 40.751617 | 2.0 | 2011 | 11.0 | 17.0 | 10.0 | 58.0 | 5.0 |
| **16044** | 16.5 | -73.982785 | 40.731421 | -74.011358 | 40.713788 | 1.0 | 2013 | 4.0 | 29.0 | 3.0 | 5.0 | 45.0 |
| **16045** | 6.5 | -73.995227 | 40.733475 | -73.984030 | 40.743287 | 2.0 | 2013 | 9.0 | 19.0 | 23.0 | 56.0 | 0.0 |
| **16046** | 6.0 | -73.976298 | 40.753948 | -73.993062 | 40.744550 | 1.0 | 2014 | 4.0 | 24.0 | 1.0 | 48.0 | 40.0 |
| **16047** | 6.1 | -73.970733 | 40.758193 | -73.979457 | 40.755830 | 1.0 | 2010 | 3.0 | 18.0 | 11.0 | 9.0 | 0.0 |
| **16048** | 9.7 | -73.988040 | 40.774902 | -74.005265 | 40.744157 | 1.0 | 2012 | 7.0 | 10.0 | 17.0 | 32.0 | 0.0 |
| **16049** | 15.7 | -74.008657 | 40.715975 | -73.975653 | 40.751233 | 4.0 | 2012 | 7.0 | 31.0 | 12.0 | 27.0 | 0.0 |
| **16050** | 8.5 | -73.996715 | 40.742504 | -73.977987 | 40.751805 | 1.0 | 2013 | 1.0 | 23.0 | 7.0 | 36.0 | 49.0 |
| **16051** | 11.5 | -73.975540 | 40.755590 | -73.944780 | 40.780050 | 2.0 | 2014 | 10.0 | 1.0 | 20.0 | 5.0 | 0.0 |
| **16052** | 10.0 | -73.987298 | 40.722007 | -74.000267 | 40.730342 | 5.0 | 2014 | 10.0 | 3.0 | 22.0 | 24.0 | 0.0 |
| **16053** | 4.0 | -73.954977 | 40.788582 | -73.964227 | 40.792305 | 1.0 | 2014 | 9.0 | 23.0 | 9.0 | 49.0 | 0.0 |
| **16054** | 5.3 | -73.993929 | 40.756944 | -73.993044 | 40.744088 | 1.0 | 2009 | 11.0 | 28.0 | 15.0 | 58.0 | 2.0 |
| **16055** | 48.3 | -73.994077 | 40.741242 | -73.830257 | 40.763645 | 1.0 | 2012 | 9.0 | 5.0 | 17.0 | 34.0 | 0.0 |
| **16056** | 38.3 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.0 | 2012 | 12.0 | 17.0 | 14.0 | 59.0 | 16.0 |
| **16057** | 5.0 | -73.963582 | 40.774242 | -73.956525 | 40.783952 | 6.0 | 2013 | 1.0 | 31.0 | 15.0 | 46.0 | 0.0 |
| **16058** | 5.5 | -73.974265 | 40.756048 | -73.980885 | 40.746838 | 2.0 | 2014 | 4.0 | 19.0 | 14.0 | 58.0 | 57.0 |
| **16059** | 5.3 | -73.973297 | 40.743768 | -73.986060 | 40.730768 | 3.0 | 2010 | 1.0 | 3.0 | 18.0 | 26.0 | 0.0 |
| **16060** | 22.0 | -73.954582 | 40.778047 | -74.005982 | 40.742117 | 1.0 | 2014 | 10.0 | 1.0 | 9.0 | 15.0 | 0.0 |
| **16061** | 10.9 | -73.994191 | 40.751138 | -73.962769 | 40.769719 | 1.0 | 2009 | 5.0 | 20.0 | 18.0 | 56.0 | 42.0 |
| **16062** | 6.5 | -74.008820 | 40.718757 | -73.998865 | 40.719987 | 1.0 | 2014 | 12.0 | 12.0 | 7.0 | 41.0 | 0.0 |
| **16063** | 16.1 | -73.981310 | 40.781695 | -74.014392 | 40.715527 | 2.0 | 2009 | 7.0 | 13.0 | 7.0 | 58.0 | 0.0 |
| **16064** | 8.5 | -73.972507 | 40.753417 | -73.979577 | 40.765495 | 1.0 | 2009 | 11.0 | 11.0 | 11.0 | 19.0 | 7.0 |
| **16065** | 8.1 | -73.957027 | 40.765945 | -73.981983 | 40.779560 | 1.0 | 2010 | 5.0 | 11.0 | 23.0 | 53.0 | 0.0 |
| **16066** | 8.5 | -74.002111 | 40.729755 | -73.983877 | 40.761975 | NaN | 2011 | 12.0 | 14.0 | 6.0 | 24.0 | 33.0 |

16067 rows × 12 columns

In [14]:

cab\_train.shape

Out[14]:

(16067, 13)

In [15]:

cab\_train.dtypes

Out[15]:

fare\_amount float64

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count float64

Year int64

Month float64

Date float64

Hour float64

Minute float64

Seconds float64

dtype: object

In [16]:

cab\_train.head(5)

Out[16]:

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 | 2009 | 6.0 | 15.0 | 17.0 | 26.0 | 21.0 |
| **1** | 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 | 2010 | 1.0 | 5.0 | 16.0 | 52.0 | 16.0 |
| **2** | 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 | 2011 | 8.0 | 18.0 | 0.0 | 35.0 | 0.0 |
| **3** | 7.7 | 2012-04-21 04:30:42 UTC | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 | 2012 | 4.0 | 21.0 | 4.0 | 30.0 | 42.0 |
| **4** | 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 | 2010 | 3.0 | 9.0 | 7.0 | 51.0 | 0.0 |

In [17]:

###To calculate the distance using pickup\_laongitude, pickup\_latitude, dropff\_longitude and dropoff\_latitude we will be using haversine formula

## First define a function which calculates the distance based on these parameters

from math import radians, cos, sin, atan2, sqrt

def get\_distance(x):

###Read the pandas series data into the function

long1=x[0]

lat1=x[1]

long2=x[2]

lat2=x[3]

r = 6371 #Radius of earth in kilometers

##Since the data is in degrees first convert them to radians

long1 = radians(long1)

lat1 = radians(lat1)

long2 = radians(long2)

lat2 = radians(lat2)

##Applying Haversine formula

diff\_long = long2 - long1

diff\_lat = lat2 - lat1

a = sin(diff\_lat/2)\*\*2 + cos(lat1) \* cos(lat2) \* sin(diff\_long/2)\*\*2

c = 2 \* atan2(sqrt(a), sqrt(1-a))

d = c \* r

return d

In [18]:

cab\_train['distance'] = cab\_train[['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']].apply(get\_distance,axis=1)

In [19]:

cab\_train.head(5)

Out[19]:

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 | 2009 | 6.0 | 15.0 | 17.0 | 26.0 | 21.0 | 1.030764 |
| **1** | 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 | 2010 | 1.0 | 5.0 | 16.0 | 52.0 | 16.0 | 8.450134 |
| **2** | 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 | 2011 | 8.0 | 18.0 | 0.0 | 35.0 | 0.0 | 1.389525 |
| **3** | 7.7 | 2012-04-21 04:30:42 UTC | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 | 2012 | 4.0 | 21.0 | 4.0 | 30.0 | 42.0 | 2.799270 |
| **4** | 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 | 2010 | 3.0 | 9.0 | 7.0 | 51.0 | 0.0 | 1.999157 |

In [20]:

###Remove the pickup\_longitude, pickup\_latitude, dropoff\_longitude , dropoff\_latitude, pickup\_datetime variables as they are not needed

cab\_train.drop(['pickup\_longitude','pickup\_latitude','dropoff\_longitude', 'dropoff\_latitude','pickup\_datetime'], axis=1, inplace=True)

In [21]:

cab\_train.head(5)

Out[21]:

|  | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 1.0 | 2009 | 6.0 | 15.0 | 17.0 | 26.0 | 21.0 | 1.030764 |
| **1** | 16.9 | 1.0 | 2010 | 1.0 | 5.0 | 16.0 | 52.0 | 16.0 | 8.450134 |
| **2** | 5.7 | 2.0 | 2011 | 8.0 | 18.0 | 0.0 | 35.0 | 0.0 | 1.389525 |
| **3** | 7.7 | 1.0 | 2012 | 4.0 | 21.0 | 4.0 | 30.0 | 42.0 | 2.799270 |
| **4** | 5.3 | 1.0 | 2010 | 3.0 | 9.0 | 7.0 | 51.0 | 0.0 | 1.999157 |

In [22]:

cab\_train.shape

Out[22]:

(16067, 9)

In [23]:

###Missing value analysis###

missing\_val = pd.DataFrame(cab\_train.isnull().sum())

In [24]:

missing\_val

Out[24]:

|  | **0** |
| --- | --- |
| **fare\_amount** | 25 |
| **passenger\_count** | 55 |
| **Year** | 0 |
| **Month** | 1 |
| **Date** | 1 |
| **Hour** | 1 |
| **Minute** | 1 |
| **Seconds** | 1 |
| **distance** | 0 |

In [25]:

####Reset\_index

missing\_val = missing\_val.reset\_index()

In [26]:

missing\_val

Out[26]:

|  | **index** | **0** |
| --- | --- | --- |
| **0** | fare\_amount | 25 |
| **1** | passenger\_count | 55 |
| **2** | Year | 0 |
| **3** | Month | 1 |
| **4** | Date | 1 |
| **5** | Hour | 1 |
| **6** | Minute | 1 |
| **7** | Seconds | 1 |
| **8** | distance | 0 |

In [27]:

#Rename the variable to proper name

missing\_val= missing\_val.rename(columns={'index': 'Variables', 0:'Missing\_Percentage'})

In [28]:

missing\_val

Out[28]:

|  | **Variables** | **Missing\_Percentage** |
| --- | --- | --- |
| **0** | fare\_amount | 25 |
| **1** | passenger\_count | 55 |
| **2** | Year | 0 |
| **3** | Month | 1 |
| **4** | Date | 1 |
| **5** | Hour | 1 |
| **6** | Minute | 1 |
| **7** | Seconds | 1 |
| **8** | distance | 0 |

In [29]:

###Calculate Percentage

missing\_val['Missing\_Percentage'] = (missing\_val['Missing\_Percentage']/len(cab\_train))\*100

In [30]:

missing\_val

Out[30]:

|  | **Variables** | **Missing\_Percentage** |
| --- | --- | --- |
| **0** | fare\_amount | 0.155598 |
| **1** | passenger\_count | 0.342317 |
| **2** | Year | 0.000000 |
| **3** | Month | 0.006224 |
| **4** | Date | 0.006224 |
| **5** | Hour | 0.006224 |
| **6** | Minute | 0.006224 |
| **7** | Seconds | 0.006224 |
| **8** | distance | 0.000000 |

In [31]:

####Since the missing values is too less in comparison hence we would be going ahead and ignoring those values by deleting

cab\_train.shape

Out[31]:

(16067, 9)

In [32]:

#####Remove Missing values from all the variables

cab\_train = cab\_train.drop(cab\_train[cab\_train['passenger\_count'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['fare\_amount'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['Month'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['Date'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['Minute'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['Seconds'].isnull()].index, axis=0)

In [33]:

cab\_train.shape

Out[33]:

(15986, 9)

In [34]:

###Calculate the missing value again to find out that it is removed

missing\_val = pd.DataFrame(cab\_train.isnull().sum())

print(missing\_val)

0

fare\_amount 0

passenger\_count 0

Year 0

Month 0

Date 0

Hour 0

Minute 0

Seconds 0

distance 0

In [35]:

##Check for negative values in fare\_amount and passanger count

cab\_train["fare\_amount"].sort\_values(ascending=True)

Out[35]:

13032 -3.00

2039 -2.90

2486 -2.50

10002 0.00

2780 0.01

1427 1.14

4321 2.50

13221 2.50

15257 2.50

4367 2.50

14633 2.50

11222 2.50

13571 2.50

8795 2.50

13488 2.50

11062 2.50

14574 2.50

657 2.50

9177 2.50

2306 2.50

11153 2.50

4539 2.50

14304 2.50

4084 2.50

6297 2.50

12343 2.50

12598 2.50

3168 2.50

3427 2.50

12178 2.50

...

649 66.30

4118 69.70

1494 70.00

15023 73.30

13615 75.00

11019 75.33

10524 75.80

8363 76.00

6668 76.80

2013 77.00

13962 77.15

4013 77.70

2639 79.00

12437 80.75

14519 82.50

4620 85.50

12614 87.00

10077 87.30

9431 88.00

7810 95.00

12915 96.00

12349 104.67

14142 108.00

6630 128.83

1483 165.00

1335 180.00

980 434.00

607 453.00

1072 4343.00

1015 54343.00

Name: fare\_amount, Length: 15986, dtype: float64

In [36]:

#Remove the negative values from the fare\_amount variable since fare cant be negative

cab\_train = cab\_train.drop(cab\_train[cab\_train['fare\_amount']<=0].index, axis=0)

In [37]:

#Similarily check for passenger\_count

cab\_train["passenger\_count"].sort\_values(ascending=True)

Out[37]:

10711 0.0

5150 0.0

6575 0.0

566 0.0

15286 0.0

15554 0.0

3489 0.0

11462 0.0

15919 0.0

4114 0.0

8971 0.0

3481 0.0

5517 0.0

12611 0.0

15514 0.0

5914 0.0

13742 0.0

14196 0.0

10642 0.0

3034 0.0

8916 0.0

4344 0.0

6881 0.0

678 0.0

5277 0.0

13379 0.0

1935 0.0

7640 0.0

3599 0.0

6036 0.0

...

4048 6.0

4595 6.0

7973 6.0

1110 6.0

685 6.0

11420 6.0

5976 6.0

10404 6.0

11415 6.0

1407 6.0

12838 6.0

1043 35.0

1242 43.0

8631 43.0

1007 53.0

8406 53.0

8445 58.0

8571 87.0

233 236.0

1107 345.0

386 354.0

263 456.0

8715 531.2

356 535.0

1200 536.0

8506 537.0

971 554.0

8985 557.0

293 5334.0

1146 5345.0

Name: passenger\_count, Length: 15982, dtype: float64

In [38]:

#####Since passanger count cannot be zero hence we remove those values as well

cab\_train = cab\_train.drop(cab\_train[cab\_train['passenger\_count']<1].index, axis=0)

In [39]:

cab\_train.shape

Out[39]:

(15924, 9)

In [40]:

#Similarily check for distance

cab\_train["distance"].sort\_values(ascending=True)

Out[40]:

3128 0.000000

12541 0.000000

12498 0.000000

4799 0.000000

872 0.000000

12478 0.000000

4839 0.000000

881 0.000000

12420 0.000000

2722 0.000000

12412 0.000000

887 0.000000

4858 0.000000

12300 0.000000

4872 0.000000

12296 0.000000

12276 0.000000

4888 0.000000

4889 0.000000

4893 0.000000

4793 0.000000

12581 0.000000

4777 0.000000

12598 0.000000

12914 0.000000

4600 0.000000

4606 0.000000

12882 0.000000

12847 0.000000

2763 0.000000

...

3075 97.985088

1684 99.771579

5663 101.094619

12228 123.561157

11619 127.509261

14536 129.560455

10710 129.950482

7014 4447.086698

5864 5420.988959

2280 6026.494216

15749 6028.926779

15783 8656.714168

14197 8657.136619

12705 8661.362152

6302 8663.039123

12983 8664.131808

6188 8664.191488

4278 8665.223767

1260 8665.268588

10488 8665.555634

10672 8665.702390

10458 8665.976222

4597 8666.566030

10215 8666.584706

13340 8666.613646

11653 8666.701504

472 8667.304968

2397 8667.454421

8647 8667.497512

9147 8667.542104

Name: distance, Length: 15924, dtype: float64

In [41]:

#####Since distance in kms cannot be zero hence we remove those values as well

cab\_train = cab\_train.drop(cab\_train[cab\_train['distance']<=0].index, axis=0)

In [42]:

cab\_train.shape

Out[42]:

(15468, 9)

In [43]:

cab\_train.dtypes

Out[43]:

fare\_amount float64

passenger\_count float64

Year int64

Month float64

Date float64

Hour float64

Minute float64

Seconds float64

distance float64

dtype: object

In [44]:

#Convert the variables to of type float to int except fare\_amount and distance variables since they contain decimal values

cab\_train['passenger\_count'] = cab\_train['passenger\_count'].astype('int64')

cab\_train['Seconds'] = cab\_train['Seconds'].astype('int64')

cab\_train['Month'] = cab\_train['Month'].astype('int64')

cab\_train['Date'] = cab\_train['Date'].astype('int64')

cab\_train['Minute'] = cab\_train['Minute'].astype('int64')

cab\_train['Hour'] = cab\_train['Hour'].astype('int64')

In [45]:

cab\_train.dtypes

Out[45]:

fare\_amount float64

passenger\_count int64

Year int64

Month int64

Date int64

Hour int64

Minute int64

Seconds int64

distance float64

dtype: object

In [46]:

#Outlier Analysis

#First plot boxplot for visualization

from matplotlib import pyplot as plt

plt.boxplot(cab\_train['passenger\_count'])

Out[46]:

{'whiskers': [<matplotlib.lines.Line2D at 0x2360b3aaeb8>,

<matplotlib.lines.Line2D at 0x2360b76e630>],

'caps': [<matplotlib.lines.Line2D at 0x2360b76e978>,

<matplotlib.lines.Line2D at 0x2360b76ecc0>],

'boxes': [<matplotlib.lines.Line2D at 0x2360b757ef0>],

'medians': [<matplotlib.lines.Line2D at 0x2360b76eda0>],

'fliers': [<matplotlib.lines.Line2D at 0x2360b77d390>],

'means': []}

In [47]:

plt.boxplot(cab\_train['distance'])

Out[47]:

{'whiskers': [<matplotlib.lines.Line2D at 0x2360b929cf8>,

<matplotlib.lines.Line2D at 0x2360b9396a0>],

'caps': [<matplotlib.lines.Line2D at 0x2360b9399e8>,

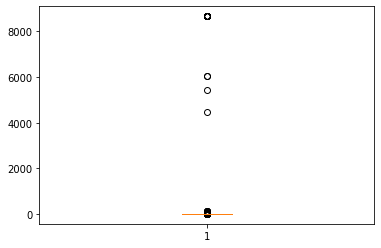
<matplotlib.lines.Line2D at 0x2360b939d30>],

'boxes': [<matplotlib.lines.Line2D at 0x2360b929f98>],

'medians': [<matplotlib.lines.Line2D at 0x2360b939e10>],

'fliers': [<matplotlib.lines.Line2D at 0x2360b945400>],

'means': []}



In [48]:

plt.boxplot(cab\_train['fare\_amount'])

Out[48]:

{'whiskers': [<matplotlib.lines.Line2D at 0x2360b9aaef0>,

<matplotlib.lines.Line2D at 0x2360b9aaf60>],

'caps': [<matplotlib.lines.Line2D at 0x2360b9b8630>,

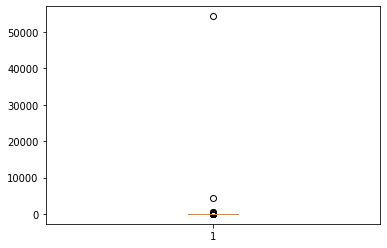
<matplotlib.lines.Line2D at 0x2360b9b8978>],

'boxes': [<matplotlib.lines.Line2D at 0x2360b9aab70>],

'medians': [<matplotlib.lines.Line2D at 0x2360b9b8cc0>],

'fliers': [<matplotlib.lines.Line2D at 0x2360b9b8da0>],

'means': []}



In [49]:

cab\_train.shape

Out[49]:

(15468, 9)

In [50]:

#####Detect and remove outliers##

column\_names = ["passenger\_count","fare\_amount","distance"]

for i in column\_names :

q75, q25 = np.percentile(cab\_train.loc[:,i], [75,25])

iqr = q75 - q25

min1 = q25 - (iqr\*1.5)

max1 = q75 + (iqr\*1.5)

cab\_train = cab\_train.drop(cab\_train[cab\_train.loc[:,i] < min1].index)

cab\_train = cab\_train.drop(cab\_train[cab\_train.loc[:,i] > max1].index)

In [51]:

cab\_train.shape

Out[51]:

(12038, 9)

In [52]:

cab\_train.head()

Out[52]:

|  | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 1 | 2009 | 6 | 15 | 17 | 26 | 21 | 1.030764 |
| **2** | 5.7 | 2 | 2011 | 8 | 18 | 0 | 35 | 0 | 1.389525 |
| **3** | 7.7 | 1 | 2012 | 4 | 21 | 4 | 30 | 42 | 2.799270 |
| **4** | 5.3 | 1 | 2010 | 3 | 9 | 7 | 51 | 0 | 1.999157 |
| **5** | 12.1 | 1 | 2011 | 1 | 6 | 9 | 50 | 45 | 3.787239 |

In [53]:

###Feature Selection

## Correlation analysis

import seaborn as sns

f, ax = plt.subplots(figsize=(10,8))

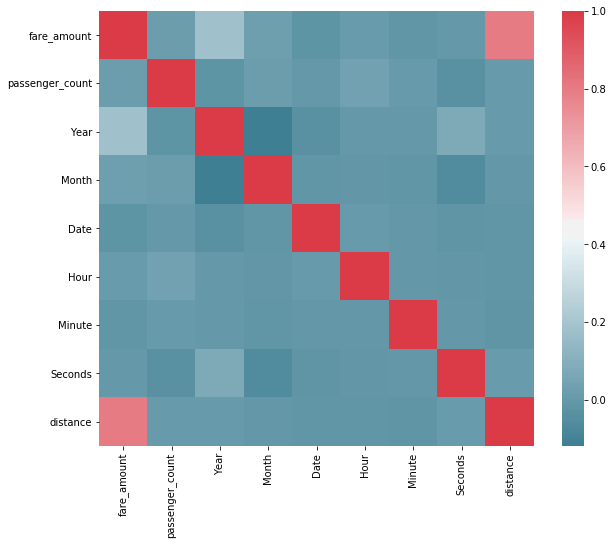
corr = cab\_train.corr()

sns.heatmap(corr,mask=np.zeros\_like(corr,dtype=np.bool), cmap = sns.diverging\_palette(220, 10, as\_cmap =True),

square=True, ax=ax)

Out[53]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x2360c7b0b00>



In [54]:

##Since all variables are dependent on each other hence we will not be deleting any of the variables.

#Next we go for Feature scaling to transform the data on same units

In [55]:

#Feature Scaling

# First check for Normality check

import matplotlib.pyplot as plt

plt.hist(cab\_train['fare\_amount'], bins= 10)

Out[55]:

(array([1.000e+00, 8.620e+02, 3.728e+03, 2.672e+03, 2.231e+03, 1.120e+03,

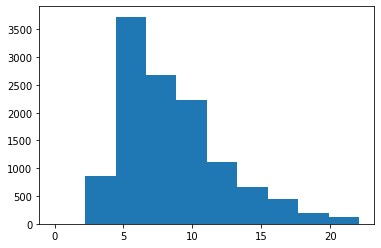
6.660e+02, 4.470e+02, 1.860e+02, 1.250e+02]),

array([1.0000e-02, 2.2190e+00, 4.4280e+00, 6.6370e+00, 8.8460e+00,

1.1055e+01, 1.3264e+01, 1.5473e+01, 1.7682e+01, 1.9891e+01,

2.2100e+01]),

<a list of 10 Patch objects>)



In [56]:

plt.hist(cab\_train['passenger\_count'], bins= 10)

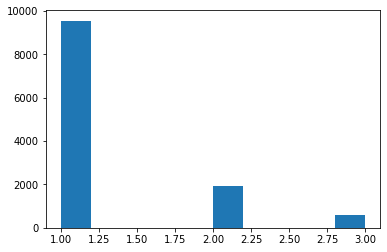
Out[56]:

(array([9541., 0., 0., 0., 0., 1926., 0., 0., 0.,

571.]),

array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. ]),

<a list of 10 Patch objects>)



In [57]:

plt.hist(cab\_train['distance'], bins= 10)

Out[57]:

(array([ 780., 2718., 2616., 1878., 1334., 917., 665., 496., 342.,

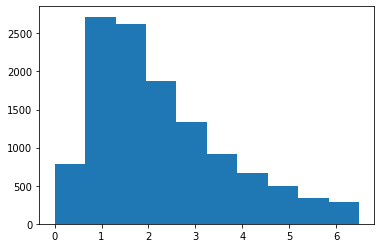
292.]),

array([1.11194926e-04, 6.49163049e-01, 1.29821490e+00, 1.94726676e+00,

2.59631861e+00, 3.24537047e+00, 3.89442232e+00, 4.54347417e+00,

5.19252603e+00, 5.84157788e+00, 6.49062974e+00]),

<a list of 10 Patch objects>)



In [58]:

#From the histogram we can see that the data is skewed and not normally distributed

#Hence we will be going for normalisation for all the variables since all are continuous variables in our data set

In [59]:

#Normalisation

cab\_train\_columns = list(cab\_train.columns.values)

for i in cab\_train\_columns:

cab\_train[i] = (cab\_train[i] - min(cab\_train[i]))/(max(cab\_train[i]) - min(cab\_train[i]))

In [60]:

cab\_train.head(5)

Out[60]:

|  | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.203259 | 0.0 | 0.000000 | 0.454545 | 0.466667 | 0.739130 | 0.440678 | 0.355932 | 0.158794 |
| **2** | 0.257583 | 0.5 | 0.333333 | 0.636364 | 0.566667 | 0.000000 | 0.593220 | 0.000000 | 0.214068 |
| **3** | 0.348121 | 0.0 | 0.500000 | 0.272727 | 0.666667 | 0.173913 | 0.508475 | 0.711864 | 0.431269 |
| **4** | 0.239475 | 0.0 | 0.166667 | 0.181818 | 0.266667 | 0.304348 | 0.864407 | 0.000000 | 0.307995 |
| **5** | 0.547306 | 0.0 | 0.333333 | 0.000000 | 0.166667 | 0.391304 | 0.847458 | 0.762712 | 0.583486 |

In [61]:

cab\_train = cab\_train.reset\_index()

In [62]:

cab\_train.head(5)

Out[62]:

|  | **index** | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0.203259 | 0.0 | 0.000000 | 0.454545 | 0.466667 | 0.739130 | 0.440678 | 0.355932 | 0.158794 |
| **1** | 2 | 0.257583 | 0.5 | 0.333333 | 0.636364 | 0.566667 | 0.000000 | 0.593220 | 0.000000 | 0.214068 |
| **2** | 3 | 0.348121 | 0.0 | 0.500000 | 0.272727 | 0.666667 | 0.173913 | 0.508475 | 0.711864 | 0.431269 |
| **3** | 4 | 0.239475 | 0.0 | 0.166667 | 0.181818 | 0.266667 | 0.304348 | 0.864407 | 0.000000 | 0.307995 |
| **4** | 5 | 0.547306 | 0.0 | 0.333333 | 0.000000 | 0.166667 | 0.391304 | 0.847458 | 0.762712 | 0.583486 |

In [63]:

cab\_train.drop(['index'], axis=1, inplace=True)

In [64]:

cab\_train.head(5)

Out[64]:

|  | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.203259 | 0.0 | 0.000000 | 0.454545 | 0.466667 | 0.739130 | 0.440678 | 0.355932 | 0.158794 |
| **1** | 0.257583 | 0.5 | 0.333333 | 0.636364 | 0.566667 | 0.000000 | 0.593220 | 0.000000 | 0.214068 |
| **2** | 0.348121 | 0.0 | 0.500000 | 0.272727 | 0.666667 | 0.173913 | 0.508475 | 0.711864 | 0.431269 |
| **3** | 0.239475 | 0.0 | 0.166667 | 0.181818 | 0.266667 | 0.304348 | 0.864407 | 0.000000 | 0.307995 |
| **4** | 0.547306 | 0.0 | 0.333333 | 0.000000 | 0.166667 | 0.391304 | 0.847458 | 0.762712 | 0.583486 |

In [65]:

###Evaluate different models

#First divide the data into train and test through sampling

from sklearn.model\_selection import train\_test\_split

model\_train, model\_test = train\_test\_split(cab\_train,test\_size=0.2)

In [66]:

#Linear regression

import statsmodels.api as sm

model\_LR = sm.OLS(model\_train.iloc[:,0], model\_train.iloc[:,1:9]).fit()

In [67]:

model\_LR.summary()

Out[67]:

|  |  |  |  |
| --- | --- | --- | --- |
| OLS Regression Results | | | |
| **Dep. Variable:** | fare\_amount | **R-squared (uncentered):** | 0.945 |
| **Model:** | OLS | **Adj. R-squared (uncentered):** | 0.945 |
| **Method:** | Least Squares | **F-statistic:** | 2.059e+04 |
| **Date:** | Thu, 19 Mar 2020 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 00:54:47 | **Log-Likelihood:** | 8617.4 |
| **No. Observations:** | 9630 | **AIC:** | -1.722e+04 |
| **Df Residuals:** | 9622 | **BIC:** | -1.716e+04 |
| **Df Model:** | 8 |  |  |
| **Covariance Type:** | nonrobust |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **coef** | **std err** | **t** | **P>|t|** | **[0.025** | **0.975]** |
| **passenger\_count** | 0.0177 | 0.004 | 4.690 | 0.000 | 0.010 | 0.025 |
| **Year** | 0.1271 | 0.003 | 41.966 | 0.000 | 0.121 | 0.133 |
| **Month** | 0.0574 | 0.003 | 19.309 | 0.000 | 0.052 | 0.063 |
| **Date** | 0.0314 | 0.003 | 9.741 | 0.000 | 0.025 | 0.038 |
| **Hour** | 0.0496 | 0.003 | 15.514 | 0.000 | 0.043 | 0.056 |
| **Minute** | 0.0356 | 0.003 | 11.276 | 0.000 | 0.029 | 0.042 |
| **Seconds** | 0.0054 | 0.003 | 1.807 | 0.071 | -0.000 | 0.011 |
| **distance** | 0.6508 | 0.004 | 152.487 | 0.000 | 0.642 | 0.659 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Omnibus:** | 2625.930 | **Durbin-Watson:** | 2.007 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 12259.674 |
| **Skew:** | 1.251 | **Prob(JB):** | 0.00 |
| **Kurtosis:** | 7.929 | **Cond. No.** | 5.55 |

Warnings:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [68]:

#Predict the test values

predictions\_LR = model\_LR.predict(model\_test.iloc[:,1:9])

In [69]:

#Calculate MAPE

def MAPE(y\_actual, y\_predict):

mape = np.mean(np.abs((y\_actual-y\_predict)/y\_actual))\*100

return mape

In [70]:

MAPE\_LR = MAPE(model\_test.iloc[:,0], predictions\_LR)

In [71]:

Accuracy\_LR = 100- MAPE\_LR

In [72]:

####Decision Tree Model###

from sklearn.tree import DecisionTreeRegressor

Model\_DTM =DecisionTreeRegressor(max\_depth=2).fit(model\_train.iloc[:,1:9], model\_train.iloc[:,0])

In [73]:

#Apply Model on test data

predictions\_DT = Model\_DTM.predict(model\_test.iloc[:,1:9])

In [74]:

MAPE\_DT = MAPE(model\_test.iloc[:,0], predictions\_DT)

In [75]:

Accuracy\_DT = 100 - MAPE\_DT

In [76]:

# Random Forest Model

from sklearn.ensemble import RandomForestRegressor

In [77]:

###Apply the model

model\_RF = RandomForestRegressor(n\_estimators = 500).fit(model\_train.iloc[:,1:9], model\_train.iloc[:,0])

In [78]:

#Apply Model on test data

predictions\_RF = model\_RF.predict(model\_test.iloc[:,1:9])

In [79]:

MAPE\_RF = MAPE(model\_test.iloc[:,0], predictions\_RF)

In [80]:

Accuracy\_RF = 100 - MAPE\_RF

In [81]:

Model\_Evaluation = [["Linear Regression", MAPE\_LR, Accuracy\_LR],["Decision Tree", MAPE\_DT, Accuracy\_DT],["Random Forest", MAPE\_RF, Accuracy\_RF]]

In [82]:

Model\_Evaluation = pd.DataFrame(Model\_Evaluation, columns = ['Model Name', 'MAPE','Accuracy'])

In [83]:

Model\_Evaluation

Out[83]:

|  | **Model Name** | **MAPE** | **Accuracy** |
| --- | --- | --- | --- |
| **0** | Linear Regression | 18.866230 | 81.133770 |
| **1** | Decision Tree | 21.444867 | 78.555133 |
| **2** | Random Forest | 18.708145 | 81.291855 |

In [ ]:

**Chapter 4**

**Results**

Based on the model selection we have done and the test data provided in the problem statement we now pre process the test data and run the prediction against the random forest model chosen for this problem and predict the cab fare.

Final code written for arriving at the results.

In [1]:

#Set working directory

import os

os.chdir("C:/Users/VB018797/Documents/Cab\_Fare\_Python")

In [2]:

#Get the current working directory##

os.getcwd()

Out[2]:

'C:\\Users\\VB018797\\Documents\\Cab\_Fare\_Python'

In [3]:

#Import libraries

import os

import pandas as pd

import numpy as np

In [4]:

#Load the data from csv

cab\_train = pd.read\_csv("train\_cab.csv", sep=',')

In [5]:

cab\_train.head(5)

Out[5]:

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 |
| **1** | 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 |
| **2** | 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 |
| **3** | 7.7 | 2012-04-21 04:30:42 UTC | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 |
| **4** | 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 |

In [6]:

######Find the shape of the data

cab\_train.shape

Out[6]:

(16067, 7)

In [7]:

#Data types of the variables

cab\_train.dtypes

Out[7]:

fare\_amount object

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count float64

dtype: object

In [8]:

####Create new variables from pickup\_datetime for reading Year, Date, Month, Hour, Minute, Seconds and use str function

cab\_train['Year']= cab\_train.pickup\_datetime.str[0:4]

cab\_train['Month']= cab\_train.pickup\_datetime.str[5:7]

cab\_train['Date']= cab\_train.pickup\_datetime.str[8:10]

cab\_train['Hour']= cab\_train.pickup\_datetime.str[11:13]

cab\_train['Minute']= cab\_train.pickup\_datetime.str[14:16]

cab\_train['Seconds'] = cab\_train.pickup\_datetime.str[17:19]

In [9]:

###Verify the shape

cab\_train.dtypes

Out[9]:

fare\_amount object

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count float64

Year object

Month object

Date object

Hour object

Minute object

Seconds object

dtype: object

In [10]:

###Convert new variables created to type numeric since they are of type object

cab\_train['Year'] = pd.to\_numeric(cab\_train['Year'],errors = "coerce")

cab\_train['Month'] = pd.to\_numeric(cab\_train['Month'],errors = "coerce")

cab\_train['Date'] = pd.to\_numeric(cab\_train['Date'],errors = "coerce")

cab\_train['Hour'] = pd.to\_numeric(cab\_train['Hour'],errors = "coerce")

cab\_train['Minute'] = pd.to\_numeric(cab\_train['Minute'],errors = "coerce")

cab\_train['Seconds'] = pd.to\_numeric(cab\_train['Seconds'],errors = "coerce")

In [11]:

cab\_train.dtypes

Out[11]:

fare\_amount object

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count float64

Year int64

Month float64

Date float64

Hour float64

Minute float64

Seconds float64

dtype: object

In [12]:

####Convert fare\_amount variable to numeric

cab\_train['fare\_amount'] = pd.to\_numeric(cab\_train['fare\_amount'],errors = "coerce")

In [13]:

#Remove pickup\_datetime variable as we do not need it

cab\_train.drop(['pickup\_datetime'], axis =1)

Out[13]:

|  | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 | 2009 | 6.0 | 15.0 | 17.0 | 26.0 | 21.0 |
| **1** | 16.9 | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 | 2010 | 1.0 | 5.0 | 16.0 | 52.0 | 16.0 |
| **2** | 5.7 | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 | 2011 | 8.0 | 18.0 | 0.0 | 35.0 | 0.0 |
| **3** | 7.7 | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 | 2012 | 4.0 | 21.0 | 4.0 | 30.0 | 42.0 |
| **4** | 5.3 | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 | 2010 | 3.0 | 9.0 | 7.0 | 51.0 | 0.0 |
| **5** | 12.1 | -74.000964 | 40.731630 | -73.972892 | 40.758233 | 1.0 | 2011 | 1.0 | 6.0 | 9.0 | 50.0 | 45.0 |
| **6** | 7.5 | -73.980002 | 40.751662 | -73.973802 | 40.764842 | 1.0 | 2012 | 11.0 | 20.0 | 20.0 | 35.0 | 0.0 |
| **7** | 16.5 | -73.951300 | 40.774138 | -73.990095 | 40.751048 | 1.0 | 2012 | 1.0 | 4.0 | 17.0 | 22.0 | 0.0 |
| **8** | NaN | -74.006462 | 40.726713 | -73.993078 | 40.731628 | 1.0 | 2012 | 12.0 | 3.0 | 13.0 | 10.0 | 0.0 |
| **9** | 8.9 | -73.980658 | 40.733873 | -73.991540 | 40.758138 | 2.0 | 2009 | 9.0 | 2.0 | 1.0 | 11.0 | 0.0 |
| **10** | 5.3 | -73.996335 | 40.737142 | -73.980721 | 40.733559 | 1.0 | 2012 | 4.0 | 8.0 | 7.0 | 30.0 | 50.0 |
| **11** | 5.5 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 3.0 | 2012 | 12.0 | 24.0 | 11.0 | 24.0 | 0.0 |
| **12** | 4.1 | -73.991601 | 40.744712 | -73.983081 | 40.744682 | 2.0 | 2009 | 11.0 | 6.0 | 1.0 | 4.0 | 3.0 |
| **13** | 7.0 | -74.005360 | 40.728867 | -74.008913 | 40.710907 | 1.0 | 2013 | 7.0 | 2.0 | 19.0 | 54.0 | 0.0 |
| **14** | 7.7 | -74.001821 | 40.737547 | -73.998060 | 40.722788 | 2.0 | 2011 | 4.0 | 5.0 | 17.0 | 11.0 | 5.0 |
| **15** | 5.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.0 | 2013 | 11.0 | 23.0 | 12.0 | 57.0 | 0.0 |
| **16** | 12.5 | -73.986430 | 40.760465 | -73.988990 | 40.737075 | 1.0 | 2014 | 2.0 | 19.0 | 7.0 | 22.0 | 0.0 |
| **17** | 5.3 | -73.981060 | 40.737690 | -73.994177 | 40.728412 | 1.0 | 2009 | 7.0 | 22.0 | 16.0 | 8.0 | 0.0 |
| **18** | 5.3 | -73.969505 | 40.784843 | -73.958732 | 40.783357 | 1.0 | 2010 | 7.0 | 7.0 | 14.0 | 52.0 | 0.0 |
| **19** | 4.0 | -73.979815 | 40.751902 | -73.979446 | 40.755481 | 1.0 | 2014 | 12.0 | 6.0 | 20.0 | 36.0 | 22.0 |
| **20** | 10.5 | -73.985382 | 40.747858 | -73.978377 | 40.762070 | 1.0 | 2010 | 9.0 | 7.0 | 13.0 | 18.0 | 0.0 |
| **21** | 11.5 | -73.957954 | 40.779252 | -73.961250 | 40.758787 | 1.0 | 2013 | 2.0 | 12.0 | 12.0 | 15.0 | 46.0 |
| **22** | 4.5 | -73.991707 | 40.770505 | -73.985459 | 40.763671 | 1.0 | 2009 | 8.0 | 6.0 | 18.0 | 17.0 | 23.0 |
| **23** | 4.9 | -74.000632 | 40.747473 | -73.986672 | 40.740577 | 1.0 | 2010 | 12.0 | 6.0 | 12.0 | 29.0 | 0.0 |
| **24** | 6.1 | -73.969622 | 40.756973 | -73.981152 | 40.759712 | 1.0 | 2009 | 12.0 | 10.0 | 15.0 | 37.0 | 0.0 |
| **25** | 7.3 | -73.991875 | 40.754437 | -73.977230 | 40.774323 | 3.0 | 2011 | 6.0 | 21.0 | 16.0 | 15.0 | 0.0 |
| **26** | NaN | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.0 | 2011 | 2.0 | 7.0 | 20.0 | 1.0 | 0.0 |
| **27** | 4.5 | -73.988893 | 40.760160 | -73.986445 | 40.757857 | 3.0 | 2011 | 6.0 | 28.0 | 19.0 | 47.0 | 0.0 |
| **28** | 9.3 | -73.989258 | 40.690835 | -74.004133 | 40.725690 | 1.0 | 2012 | 5.0 | 4.0 | 6.0 | 11.0 | 20.0 |
| **29** | 4.5 | -73.981020 | 40.737760 | -73.980668 | 40.730497 | 2.0 | 2013 | 8.0 | 11.0 | 0.0 | 52.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **16037** | 6.5 | -73.992618 | 40.723878 | -73.977073 | 40.733778 | 1.0 | 2012 | 2.0 | 27.0 | 21.0 | 40.0 | 50.0 |
| **16038** | 5.7 | -73.990336 | 40.718973 | -73.956060 | 40.713974 | 1.0 | 2010 | 8.0 | 31.0 | 10.0 | 43.0 | 42.0 |
| **16039** | 12.9 | -73.936462 | 40.794292 | -73.948747 | 40.779097 | 5.0 | 2010 | 12.0 | 11.0 | 16.0 | 25.0 | 0.0 |
| **16040** | 6.5 | -73.980597 | 40.744267 | -73.979330 | 40.731205 | 1.0 | 2014 | 6.0 | 16.0 | 0.0 | 5.0 | 19.0 |
| **16041** | 11.0 | -73.983610 | 40.747090 | -73.961310 | 40.770980 | 1.0 | 2014 | 11.0 | 17.0 | 21.0 | 53.0 | 0.0 |
| **16042** | 8.5 | -73.991425 | 40.749832 | -74.000107 | 40.727898 | 2.0 | 2015 | 4.0 | 6.0 | 21.0 | 53.0 | 6.0 |
| **16043** | 8.5 | -73.973961 | 40.764055 | -73.986807 | 40.751617 | 2.0 | 2011 | 11.0 | 17.0 | 10.0 | 58.0 | 5.0 |
| **16044** | 16.5 | -73.982785 | 40.731421 | -74.011358 | 40.713788 | 1.0 | 2013 | 4.0 | 29.0 | 3.0 | 5.0 | 45.0 |
| **16045** | 6.5 | -73.995227 | 40.733475 | -73.984030 | 40.743287 | 2.0 | 2013 | 9.0 | 19.0 | 23.0 | 56.0 | 0.0 |
| **16046** | 6.0 | -73.976298 | 40.753948 | -73.993062 | 40.744550 | 1.0 | 2014 | 4.0 | 24.0 | 1.0 | 48.0 | 40.0 |
| **16047** | 6.1 | -73.970733 | 40.758193 | -73.979457 | 40.755830 | 1.0 | 2010 | 3.0 | 18.0 | 11.0 | 9.0 | 0.0 |
| **16048** | 9.7 | -73.988040 | 40.774902 | -74.005265 | 40.744157 | 1.0 | 2012 | 7.0 | 10.0 | 17.0 | 32.0 | 0.0 |
| **16049** | 15.7 | -74.008657 | 40.715975 | -73.975653 | 40.751233 | 4.0 | 2012 | 7.0 | 31.0 | 12.0 | 27.0 | 0.0 |
| **16050** | 8.5 | -73.996715 | 40.742504 | -73.977987 | 40.751805 | 1.0 | 2013 | 1.0 | 23.0 | 7.0 | 36.0 | 49.0 |
| **16051** | 11.5 | -73.975540 | 40.755590 | -73.944780 | 40.780050 | 2.0 | 2014 | 10.0 | 1.0 | 20.0 | 5.0 | 0.0 |
| **16052** | 10.0 | -73.987298 | 40.722007 | -74.000267 | 40.730342 | 5.0 | 2014 | 10.0 | 3.0 | 22.0 | 24.0 | 0.0 |
| **16053** | 4.0 | -73.954977 | 40.788582 | -73.964227 | 40.792305 | 1.0 | 2014 | 9.0 | 23.0 | 9.0 | 49.0 | 0.0 |
| **16054** | 5.3 | -73.993929 | 40.756944 | -73.993044 | 40.744088 | 1.0 | 2009 | 11.0 | 28.0 | 15.0 | 58.0 | 2.0 |
| **16055** | 48.3 | -73.994077 | 40.741242 | -73.830257 | 40.763645 | 1.0 | 2012 | 9.0 | 5.0 | 17.0 | 34.0 | 0.0 |
| **16056** | 38.3 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.0 | 2012 | 12.0 | 17.0 | 14.0 | 59.0 | 16.0 |
| **16057** | 5.0 | -73.963582 | 40.774242 | -73.956525 | 40.783952 | 6.0 | 2013 | 1.0 | 31.0 | 15.0 | 46.0 | 0.0 |
| **16058** | 5.5 | -73.974265 | 40.756048 | -73.980885 | 40.746838 | 2.0 | 2014 | 4.0 | 19.0 | 14.0 | 58.0 | 57.0 |
| **16059** | 5.3 | -73.973297 | 40.743768 | -73.986060 | 40.730768 | 3.0 | 2010 | 1.0 | 3.0 | 18.0 | 26.0 | 0.0 |
| **16060** | 22.0 | -73.954582 | 40.778047 | -74.005982 | 40.742117 | 1.0 | 2014 | 10.0 | 1.0 | 9.0 | 15.0 | 0.0 |
| **16061** | 10.9 | -73.994191 | 40.751138 | -73.962769 | 40.769719 | 1.0 | 2009 | 5.0 | 20.0 | 18.0 | 56.0 | 42.0 |
| **16062** | 6.5 | -74.008820 | 40.718757 | -73.998865 | 40.719987 | 1.0 | 2014 | 12.0 | 12.0 | 7.0 | 41.0 | 0.0 |
| **16063** | 16.1 | -73.981310 | 40.781695 | -74.014392 | 40.715527 | 2.0 | 2009 | 7.0 | 13.0 | 7.0 | 58.0 | 0.0 |
| **16064** | 8.5 | -73.972507 | 40.753417 | -73.979577 | 40.765495 | 1.0 | 2009 | 11.0 | 11.0 | 11.0 | 19.0 | 7.0 |
| **16065** | 8.1 | -73.957027 | 40.765945 | -73.981983 | 40.779560 | 1.0 | 2010 | 5.0 | 11.0 | 23.0 | 53.0 | 0.0 |
| **16066** | 8.5 | -74.002111 | 40.729755 | -73.983877 | 40.761975 | NaN | 2011 | 12.0 | 14.0 | 6.0 | 24.0 | 33.0 |

16067 rows × 12 columns

In [14]:

cab\_train.shape

Out[14]:

(16067, 13)

In [15]:

cab\_train.dtypes

Out[15]:

fare\_amount float64

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count float64

Year int64

Month float64

Date float64

Hour float64

Minute float64

Seconds float64

dtype: object

In [16]:

cab\_train.head(5)

Out[16]:

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 | 2009 | 6.0 | 15.0 | 17.0 | 26.0 | 21.0 |
| **1** | 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 | 2010 | 1.0 | 5.0 | 16.0 | 52.0 | 16.0 |
| **2** | 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 | 2011 | 8.0 | 18.0 | 0.0 | 35.0 | 0.0 |
| **3** | 7.7 | 2012-04-21 04:30:42 UTC | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 | 2012 | 4.0 | 21.0 | 4.0 | 30.0 | 42.0 |
| **4** | 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 | 2010 | 3.0 | 9.0 | 7.0 | 51.0 | 0.0 |

In [17]:

###To calculate the distance using pickup\_laongitude, pickup\_latitude, dropff\_longitude and dropoff\_latitude we will be using haversine formula

## First define a function which calculates the distance based on these parameters

from math import radians, cos, sin, atan2, sqrt

def get\_distance(x):

###Read the pandas series data into the function

long1=x[0]

lat1=x[1]

long2=x[2]

lat2=x[3]

r = 6371 #Radius of earth in kilometers

##Since the data is in degrees first convert them to radians

long1 = radians(long1)

lat1 = radians(lat1)

long2 = radians(long2)

lat2 = radians(lat2)

##Applying Haversine formula

diff\_long = long2 - long1

diff\_lat = lat2 - lat1

a = sin(diff\_lat/2)\*\*2 + cos(lat1) \* cos(lat2) \* sin(diff\_long/2)\*\*2

c = 2 \* atan2(sqrt(a), sqrt(1-a))

d = c \* r

return d

In [18]:

cab\_train['distance'] = cab\_train[['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']].apply(get\_distance,axis=1)

In [19]:

cab\_train.head(5)

Out[19]:

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 | 2009 | 6.0 | 15.0 | 17.0 | 26.0 | 21.0 | 1.030764 |
| **1** | 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 | 2010 | 1.0 | 5.0 | 16.0 | 52.0 | 16.0 | 8.450134 |
| **2** | 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 | 2011 | 8.0 | 18.0 | 0.0 | 35.0 | 0.0 | 1.389525 |
| **3** | 7.7 | 2012-04-21 04:30:42 UTC | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 | 2012 | 4.0 | 21.0 | 4.0 | 30.0 | 42.0 | 2.799270 |
| **4** | 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 | 2010 | 3.0 | 9.0 | 7.0 | 51.0 | 0.0 | 1.999157 |

In [20]:

###Remove the pickup\_longitude, pickup\_latitude, dropoff\_longitude , dropoff\_latitude, pickup\_datetime variables as they are not needed

cab\_train.drop(['pickup\_longitude','pickup\_latitude','dropoff\_longitude', 'dropoff\_latitude','pickup\_datetime'], axis=1, inplace=True)

In [21]:

cab\_train.head(5)

Out[21]:

|  | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 1.0 | 2009 | 6.0 | 15.0 | 17.0 | 26.0 | 21.0 | 1.030764 |
| **1** | 16.9 | 1.0 | 2010 | 1.0 | 5.0 | 16.0 | 52.0 | 16.0 | 8.450134 |
| **2** | 5.7 | 2.0 | 2011 | 8.0 | 18.0 | 0.0 | 35.0 | 0.0 | 1.389525 |
| **3** | 7.7 | 1.0 | 2012 | 4.0 | 21.0 | 4.0 | 30.0 | 42.0 | 2.799270 |
| **4** | 5.3 | 1.0 | 2010 | 3.0 | 9.0 | 7.0 | 51.0 | 0.0 | 1.999157 |

In [22]:

cab\_train.shape

Out[22]:

(16067, 9)

In [23]:

###Missing value analysis###

missing\_val = pd.DataFrame(cab\_train.isnull().sum())

In [24]:

missing\_val

Out[24]:

|  | **0** |
| --- | --- |
| **fare\_amount** | 25 |
| **passenger\_count** | 55 |
| **Year** | 0 |
| **Month** | 1 |
| **Date** | 1 |
| **Hour** | 1 |
| **Minute** | 1 |
| **Seconds** | 1 |
| **distance** | 0 |

In [25]:

####Reset\_index

missing\_val = missing\_val.reset\_index()

In [26]:

missing\_val

Out[26]:

|  | **index** | **0** |
| --- | --- | --- |
| **0** | fare\_amount | 25 |
| **1** | passenger\_count | 55 |
| **2** | Year | 0 |
| **3** | Month | 1 |
| **4** | Date | 1 |
| **5** | Hour | 1 |
| **6** | Minute | 1 |
| **7** | Seconds | 1 |
| **8** | distance | 0 |

In [27]:

#Rename the variable to proper name

missing\_val= missing\_val.rename(columns={'index': 'Variables', 0:'Missing\_Percentage'})

In [28]:

missing\_val

Out[28]:

|  | **Variables** | **Missing\_Percentage** |
| --- | --- | --- |
| **0** | fare\_amount | 25 |
| **1** | passenger\_count | 55 |
| **2** | Year | 0 |
| **3** | Month | 1 |
| **4** | Date | 1 |
| **5** | Hour | 1 |
| **6** | Minute | 1 |
| **7** | Seconds | 1 |
| **8** | distance | 0 |

In [29]:

###Calculate Percentage

missing\_val['Missing\_Percentage'] = (missing\_val['Missing\_Percentage']/len(cab\_train))\*100

In [30]:

missing\_val

Out[30]:

|  | **Variables** | **Missing\_Percentage** |
| --- | --- | --- |
| **0** | fare\_amount | 0.155598 |
| **1** | passenger\_count | 0.342317 |
| **2** | Year | 0.000000 |
| **3** | Month | 0.006224 |
| **4** | Date | 0.006224 |
| **5** | Hour | 0.006224 |
| **6** | Minute | 0.006224 |
| **7** | Seconds | 0.006224 |
| **8** | distance | 0.000000 |

In [31]:

####Since the missing values is too less in comparison hence we would be going ahead and ignoring those values by deleting

cab\_train.shape

Out[31]:

(16067, 9)

In [32]:

#####Remove Missing values from all the variables

cab\_train = cab\_train.drop(cab\_train[cab\_train['passenger\_count'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['fare\_amount'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['Month'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['Date'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['Minute'].isnull()].index, axis=0)

cab\_train = cab\_train.drop(cab\_train[cab\_train['Seconds'].isnull()].index, axis=0)

In [33]:

cab\_train.shape

Out[33]:

(15986, 9)

In [34]:

###Calculate the missing value again to find out that it is removed

missing\_val = pd.DataFrame(cab\_train.isnull().sum())

print(missing\_val)

0

fare\_amount 0

passenger\_count 0

Year 0

Month 0

Date 0

Hour 0

Minute 0

Seconds 0

distance 0

In [35]:

##Check for negative values in fare\_amount and passanger count

cab\_train["fare\_amount"].sort\_values(ascending=True)

Out[35]:

13032 -3.00

2039 -2.90

2486 -2.50

10002 0.00

2780 0.01

1427 1.14

4321 2.50

13221 2.50

15257 2.50

4367 2.50

14633 2.50

11222 2.50

13571 2.50

8795 2.50

13488 2.50

11062 2.50

14574 2.50

657 2.50

9177 2.50

2306 2.50

11153 2.50

4539 2.50

14304 2.50

4084 2.50

6297 2.50

12343 2.50

12598 2.50

3168 2.50

3427 2.50

12178 2.50

...

649 66.30

4118 69.70

1494 70.00

15023 73.30

13615 75.00

11019 75.33

10524 75.80

8363 76.00

6668 76.80

2013 77.00

13962 77.15

4013 77.70

2639 79.00

12437 80.75

14519 82.50

4620 85.50

12614 87.00

10077 87.30

9431 88.00

7810 95.00

12915 96.00

12349 104.67

14142 108.00

6630 128.83

1483 165.00

1335 180.00

980 434.00

607 453.00

1072 4343.00

1015 54343.00

Name: fare\_amount, Length: 15986, dtype: float64

In [36]:

#Remove the negative values from the fare\_amount variable since fare cant be negative

cab\_train = cab\_train.drop(cab\_train[cab\_train['fare\_amount']<=0].index, axis=0)

In [37]:

#Similarily check for passenger\_count

cab\_train["passenger\_count"].sort\_values(ascending=True)

Out[37]:

10711 0.0

5150 0.0

6575 0.0

566 0.0

15286 0.0

15554 0.0

3489 0.0

11462 0.0

15919 0.0

4114 0.0

8971 0.0

3481 0.0

5517 0.0

12611 0.0

15514 0.0

5914 0.0

13742 0.0

14196 0.0

10642 0.0

3034 0.0

8916 0.0

4344 0.0

6881 0.0

678 0.0

5277 0.0

13379 0.0

1935 0.0

7640 0.0

3599 0.0

6036 0.0

...

4048 6.0

4595 6.0

7973 6.0

1110 6.0

685 6.0

11420 6.0

5976 6.0

10404 6.0

11415 6.0

1407 6.0

12838 6.0

1043 35.0

1242 43.0

8631 43.0

1007 53.0

8406 53.0

8445 58.0

8571 87.0

233 236.0

1107 345.0

386 354.0

263 456.0

8715 531.2

356 535.0

1200 536.0

8506 537.0

971 554.0

8985 557.0

293 5334.0

1146 5345.0

Name: passenger\_count, Length: 15982, dtype: float64

In [38]:

#####Since passanger count cannot be zero hence we remove those values as well

cab\_train = cab\_train.drop(cab\_train[cab\_train['passenger\_count']<1].index, axis=0)

In [39]:

cab\_train.shape

Out[39]:

(15924, 9)

In [40]:

#Similarily check for distance

cab\_train["distance"].sort\_values(ascending=True)

Out[40]:

3128 0.000000

12541 0.000000

12498 0.000000

4799 0.000000

872 0.000000

12478 0.000000

4839 0.000000

881 0.000000

12420 0.000000

2722 0.000000

12412 0.000000

887 0.000000

4858 0.000000

12300 0.000000

4872 0.000000

12296 0.000000

12276 0.000000

4888 0.000000

4889 0.000000

4893 0.000000

4793 0.000000

12581 0.000000

4777 0.000000

12598 0.000000

12914 0.000000

4600 0.000000

4606 0.000000

12882 0.000000

12847 0.000000

2763 0.000000

...

3075 97.985088

1684 99.771579

5663 101.094619

12228 123.561157

11619 127.509261

14536 129.560455

10710 129.950482

7014 4447.086698

5864 5420.988959

2280 6026.494216

15749 6028.926779

15783 8656.714168

14197 8657.136619

12705 8661.362152

6302 8663.039123

12983 8664.131808

6188 8664.191488

4278 8665.223767

1260 8665.268588

10488 8665.555634

10672 8665.702390

10458 8665.976222

4597 8666.566030

10215 8666.584706

13340 8666.613646

11653 8666.701504

472 8667.304968

2397 8667.454421

8647 8667.497512

9147 8667.542104

Name: distance, Length: 15924, dtype: float64

In [41]:

#####Since distance in kms cannot be zero hence we remove those values as well

cab\_train = cab\_train.drop(cab\_train[cab\_train['distance']<=0].index, axis=0)

In [42]:

cab\_train.shape

Out[42]:

(15468, 9)

In [43]:

cab\_train.dtypes

Out[43]:

fare\_amount float64

passenger\_count float64

Year int64

Month float64

Date float64

Hour float64

Minute float64

Seconds float64

distance float64

dtype: object

In [44]:

#Convert the variables to of type float to int except fare\_amount and distance variables since they contain decimal values

cab\_train['passenger\_count'] = cab\_train['passenger\_count'].astype('int64')

cab\_train['Seconds'] = cab\_train['Seconds'].astype('int64')

cab\_train['Month'] = cab\_train['Month'].astype('int64')

cab\_train['Date'] = cab\_train['Date'].astype('int64')

cab\_train['Minute'] = cab\_train['Minute'].astype('int64')

cab\_train['Hour'] = cab\_train['Hour'].astype('int64')

In [45]:

cab\_train.dtypes

Out[45]:

fare\_amount float64

passenger\_count int64

Year int64

Month int64

Date int64

Hour int64

Minute int64

Seconds int64

distance float64

dtype: object

In [46]:

#Outlier Analysis

#First plot boxplot for visualization

from matplotlib import pyplot as plt

plt.boxplot(cab\_train['passenger\_count'])

Out[46]:

{'whiskers': [<matplotlib.lines.Line2D at 0x218fcabc4e0>,

<matplotlib.lines.Line2D at 0x218fcabc828>],

'caps': [<matplotlib.lines.Line2D at 0x218fcabcb70>,

<matplotlib.lines.Line2D at 0x218fcabceb8>],

'boxes': [<matplotlib.lines.Line2D at 0x218fcabc0b8>],

'medians': [<matplotlib.lines.Line2D at 0x218fcabcf98>],

'fliers': [<matplotlib.lines.Line2D at 0x218fcacc588>],

'means': []}

In [47]:

plt.boxplot(cab\_train['distance'])

Out[47]:

{'whiskers': [<matplotlib.lines.Line2D at 0x218fcc78470>,

<matplotlib.lines.Line2D at 0x218fcc787b8>],

'caps': [<matplotlib.lines.Line2D at 0x218fcc78b00>,

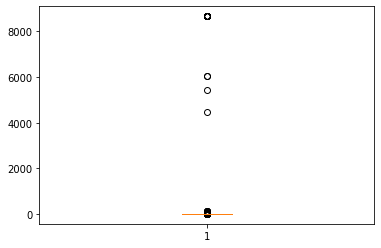
<matplotlib.lines.Line2D at 0x218fcc78e48>],

'boxes': [<matplotlib.lines.Line2D at 0x218fcc78048>],

'medians': [<matplotlib.lines.Line2D at 0x218fcc78f28>],

'fliers': [<matplotlib.lines.Line2D at 0x218fcc84518>],

'means': []}



In [48]:

plt.boxplot(cab\_train['fare\_amount'])

Out[48]:

{'whiskers': [<matplotlib.lines.Line2D at 0x218fccedf60>,

<matplotlib.lines.Line2D at 0x218fccedfd0>],

'caps': [<matplotlib.lines.Line2D at 0x218fccfa6a0>,

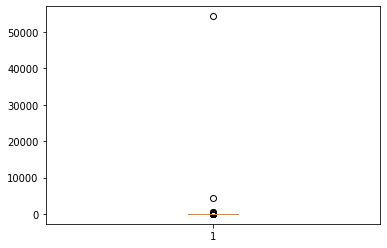
<matplotlib.lines.Line2D at 0x218fccfa9e8>],

'boxes': [<matplotlib.lines.Line2D at 0x218fccedb38>],

'medians': [<matplotlib.lines.Line2D at 0x218fccfad30>],

'fliers': [<matplotlib.lines.Line2D at 0x218fccfae10>],

'means': []}



In [49]:

cab\_train.shape

Out[49]:

(15468, 9)

In [50]:

#####Detect and remove outliers##

column\_names = ["passenger\_count","fare\_amount","distance"]

for i in column\_names :

q75, q25 = np.percentile(cab\_train.loc[:,i], [75,25])

iqr = q75 - q25

min1 = q25 - (iqr\*1.5)

max1 = q75 + (iqr\*1.5)

cab\_train = cab\_train.drop(cab\_train[cab\_train.loc[:,i] < min1].index)

cab\_train = cab\_train.drop(cab\_train[cab\_train.loc[:,i] > max1].index)

In [51]:

cab\_train.shape

Out[51]:

(12038, 9)

In [52]:

cab\_train.head()

Out[52]:

|  | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 1 | 2009 | 6 | 15 | 17 | 26 | 21 | 1.030764 |
| **2** | 5.7 | 2 | 2011 | 8 | 18 | 0 | 35 | 0 | 1.389525 |
| **3** | 7.7 | 1 | 2012 | 4 | 21 | 4 | 30 | 42 | 2.799270 |
| **4** | 5.3 | 1 | 2010 | 3 | 9 | 7 | 51 | 0 | 1.999157 |
| **5** | 12.1 | 1 | 2011 | 1 | 6 | 9 | 50 | 45 | 3.787239 |

In [53]:

###Feature Selection

## Correlation analysis

import seaborn as sns

f, ax = plt.subplots(figsize=(10,8))

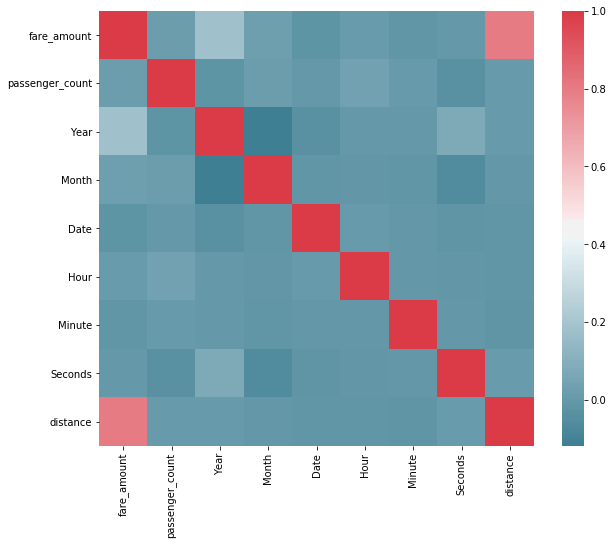
corr = cab\_train.corr()

sns.heatmap(corr,mask=np.zeros\_like(corr,dtype=np.bool), cmap = sns.diverging\_palette(220, 10, as\_cmap =True),

square=True, ax=ax)

Out[53]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x218fdaefa58>



In [54]:

##Since all variables are dependent on each other hence we will not be deleting any of the variables.

#Next we go for Feature scaling to transform the data on same units

In [55]:

#Feature Scaling

# First check for Normality check

import matplotlib.pyplot as plt

plt.hist(cab\_train['fare\_amount'], bins= 10)

Out[55]:

(array([1.000e+00, 8.620e+02, 3.728e+03, 2.672e+03, 2.231e+03, 1.120e+03,

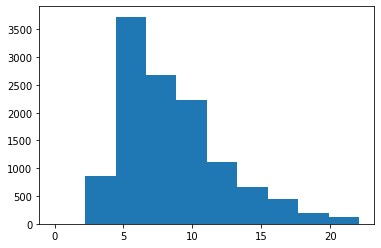
6.660e+02, 4.470e+02, 1.860e+02, 1.250e+02]),

array([1.0000e-02, 2.2190e+00, 4.4280e+00, 6.6370e+00, 8.8460e+00,

1.1055e+01, 1.3264e+01, 1.5473e+01, 1.7682e+01, 1.9891e+01,

2.2100e+01]),

<a list of 10 Patch objects>)



In [56]:

plt.hist(cab\_train['passenger\_count'], bins= 10)

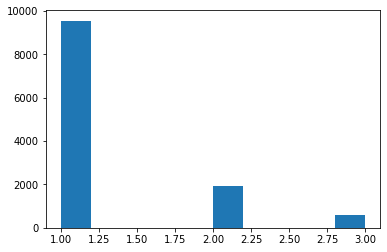
Out[56]:

(array([9541., 0., 0., 0., 0., 1926., 0., 0., 0.,

571.]),

array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. ]),

<a list of 10 Patch objects>)



In [57]:

plt.hist(cab\_train['distance'], bins= 10)

Out[57]:

(array([ 780., 2718., 2616., 1878., 1334., 917., 665., 496., 342.,

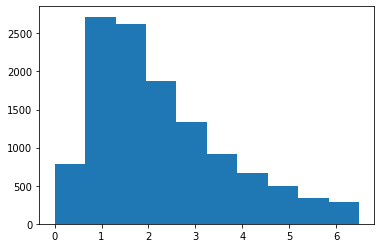
292.]),

array([1.11194926e-04, 6.49163049e-01, 1.29821490e+00, 1.94726676e+00,

2.59631861e+00, 3.24537047e+00, 3.89442232e+00, 4.54347417e+00,

5.19252603e+00, 5.84157788e+00, 6.49062974e+00]),

<a list of 10 Patch objects>)



In [58]:

#From the histogram we can see that the data is skewed and not normally distributed

#Hence we will be going for normalisation for all the variables since all are continuous variables in our data set

In [59]:

#Normalisation

cab\_train\_columns = list(cab\_train.columns.values)

for i in cab\_train\_columns:

cab\_train[i] = (cab\_train[i] - min(cab\_train[i]))/(max(cab\_train[i]) - min(cab\_train[i]))

In [60]:

cab\_train.head(5)

Out[60]:

|  | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.203259 | 0.0 | 0.000000 | 0.454545 | 0.466667 | 0.739130 | 0.440678 | 0.355932 | 0.158794 |
| **2** | 0.257583 | 0.5 | 0.333333 | 0.636364 | 0.566667 | 0.000000 | 0.593220 | 0.000000 | 0.214068 |
| **3** | 0.348121 | 0.0 | 0.500000 | 0.272727 | 0.666667 | 0.173913 | 0.508475 | 0.711864 | 0.431269 |
| **4** | 0.239475 | 0.0 | 0.166667 | 0.181818 | 0.266667 | 0.304348 | 0.864407 | 0.000000 | 0.307995 |
| **5** | 0.547306 | 0.0 | 0.333333 | 0.000000 | 0.166667 | 0.391304 | 0.847458 | 0.762712 | 0.583486 |

In [61]:

cab\_train = cab\_train.reset\_index()

In [62]:

cab\_train.head(5)

Out[62]:

|  | **index** | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0.203259 | 0.0 | 0.000000 | 0.454545 | 0.466667 | 0.739130 | 0.440678 | 0.355932 | 0.158794 |
| **1** | 2 | 0.257583 | 0.5 | 0.333333 | 0.636364 | 0.566667 | 0.000000 | 0.593220 | 0.000000 | 0.214068 |
| **2** | 3 | 0.348121 | 0.0 | 0.500000 | 0.272727 | 0.666667 | 0.173913 | 0.508475 | 0.711864 | 0.431269 |
| **3** | 4 | 0.239475 | 0.0 | 0.166667 | 0.181818 | 0.266667 | 0.304348 | 0.864407 | 0.000000 | 0.307995 |
| **4** | 5 | 0.547306 | 0.0 | 0.333333 | 0.000000 | 0.166667 | 0.391304 | 0.847458 | 0.762712 | 0.583486 |

In [63]:

cab\_train.drop(['index'], axis=1, inplace=True)

In [64]:

cab\_train.head(5)

Out[64]:

|  | **fare\_amount** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.203259 | 0.0 | 0.000000 | 0.454545 | 0.466667 | 0.739130 | 0.440678 | 0.355932 | 0.158794 |
| **1** | 0.257583 | 0.5 | 0.333333 | 0.636364 | 0.566667 | 0.000000 | 0.593220 | 0.000000 | 0.214068 |
| **2** | 0.348121 | 0.0 | 0.500000 | 0.272727 | 0.666667 | 0.173913 | 0.508475 | 0.711864 | 0.431269 |
| **3** | 0.239475 | 0.0 | 0.166667 | 0.181818 | 0.266667 | 0.304348 | 0.864407 | 0.000000 | 0.307995 |
| **4** | 0.547306 | 0.0 | 0.333333 | 0.000000 | 0.166667 | 0.391304 | 0.847458 | 0.762712 | 0.583486 |

In [65]:

###Evaluate different models

#First divide the data into train and test through sampling

from sklearn.model\_selection import train\_test\_split

model\_train, model\_test = train\_test\_split(cab\_train,test\_size=0.2)

In [66]:

#Linear regression

import statsmodels.api as sm

model\_LR = sm.OLS(model\_train.iloc[:,0], model\_train.iloc[:,1:9]).fit()

In [67]:

model\_LR.summary()

Out[67]:

|  |  |  |  |
| --- | --- | --- | --- |
| OLS Regression Results | | | |
| **Dep. Variable:** | fare\_amount | **R-squared (uncentered):** | 0.945 |
| **Model:** | OLS | **Adj. R-squared (uncentered):** | 0.945 |
| **Method:** | Least Squares | **F-statistic:** | 2.060e+04 |
| **Date:** | Thu, 19 Mar 2020 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 02:23:11 | **Log-Likelihood:** | 8626.4 |
| **No. Observations:** | 9630 | **AIC:** | -1.724e+04 |
| **Df Residuals:** | 9622 | **BIC:** | -1.718e+04 |
| **Df Model:** | 8 |  |  |
| **Covariance Type:** | nonrobust |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **coef** | **std err** | **t** | **P>|t|** | **[0.025** | **0.975]** |
| **passenger\_count** | 0.0167 | 0.004 | 4.410 | 0.000 | 0.009 | 0.024 |
| **Year** | 0.1290 | 0.003 | 42.316 | 0.000 | 0.123 | 0.135 |
| **Month** | 0.0579 | 0.003 | 19.444 | 0.000 | 0.052 | 0.064 |
| **Date** | 0.0307 | 0.003 | 9.514 | 0.000 | 0.024 | 0.037 |
| **Hour** | 0.0509 | 0.003 | 15.858 | 0.000 | 0.045 | 0.057 |
| **Minute** | 0.0335 | 0.003 | 10.605 | 0.000 | 0.027 | 0.040 |
| **Seconds** | 0.0064 | 0.003 | 2.146 | 0.032 | 0.001 | 0.012 |
| **distance** | 0.6498 | 0.004 | 151.586 | 0.000 | 0.641 | 0.658 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Omnibus:** | 2702.809 | **Durbin-Watson:** | 1.995 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 11754.113 |
| **Skew:** | 1.314 | **Prob(JB):** | 0.00 |
| **Kurtosis:** | 7.732 | **Cond. No.** | 5.59 |

Warnings:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [68]:

#Predict the test values

predictions\_LR = model\_LR.predict(model\_test.iloc[:,1:9])

In [69]:

#Calculate MAPE

def MAPE(y\_actual, y\_predict):

mape = np.mean(np.abs((y\_actual-y\_predict)/y\_actual))\*100

return mape

In [70]:

MAPE\_LR = MAPE(model\_test.iloc[:,0], predictions\_LR)

In [71]:

Accuracy\_LR = 100- MAPE\_LR

In [72]:

####Decision Tree Model###

from sklearn.tree import DecisionTreeRegressor

Model\_DTM =DecisionTreeRegressor(max\_depth=2).fit(model\_train.iloc[:,1:9], model\_train.iloc[:,0])

In [73]:

#Apply Model on test data

predictions\_DT = Model\_DTM.predict(model\_test.iloc[:,1:9])

In [74]:

MAPE\_DT = MAPE(model\_test.iloc[:,0], predictions\_DT)

In [75]:

Accuracy\_DT = 100 - MAPE\_DT

In [76]:

# Random Forest Model

from sklearn.ensemble import RandomForestRegressor

In [77]:

###Apply the model

model\_RF = RandomForestRegressor(n\_estimators = 500).fit(model\_train.iloc[:,1:9], model\_train.iloc[:,0])

In [78]:

#Apply Model on test data

predictions\_RF = model\_RF.predict(model\_test.iloc[:,1:9])

In [79]:

MAPE\_RF = MAPE(model\_test.iloc[:,0], predictions\_RF)

In [80]:

Accuracy\_RF = 100 - MAPE\_RF

In [81]:

Model\_Evaluation = [["Linear Regression", MAPE\_LR, Accuracy\_LR],["Decision Tree", MAPE\_DT, Accuracy\_DT],["Random Forest", MAPE\_RF, Accuracy\_RF]]

In [82]:

Model\_Evaluation = pd.DataFrame(Model\_Evaluation, columns = ['Model Name', 'MAPE','Accuracy'])

In [83]:

Model\_Evaluation

Out[83]:

|  | **Model Name** | **MAPE** | **Accuracy** |
| --- | --- | --- | --- |
| **0** | Linear Regression | inf | -inf |
| **1** | Decision Tree | inf | -inf |
| **2** | Random Forest | inf | -inf |

In [84]:

## Load the test data from csv

cab\_test = pd.read\_csv("test\_cab.csv", sep=',')

In [85]:

cab\_test.head(5)

Out[85]:

|  | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 2015-01-27 13:08:24 UTC | -73.973320 | 40.763805 | -73.981430 | 40.743835 | 1 |
| **1** | 2015-01-27 13:08:24 UTC | -73.986862 | 40.719383 | -73.998886 | 40.739201 | 1 |
| **2** | 2011-10-08 11:53:44 UTC | -73.982524 | 40.751260 | -73.979654 | 40.746139 | 1 |
| **3** | 2012-12-01 21:12:12 UTC | -73.981160 | 40.767807 | -73.990448 | 40.751635 | 1 |
| **4** | 2012-12-01 21:12:12 UTC | -73.966046 | 40.789775 | -73.988565 | 40.744427 | 1 |

In [86]:

######Find the shape of the data

cab\_test.shape

Out[86]:

(9914, 6)

In [87]:

cab\_test.dtypes

Out[87]:

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count int64

dtype: object

In [88]:

#Similar to train Create new variables from pickup\_datetime like Year, Date, Month, Hour, Minute, Seconds and use str function

cab\_test['Year']= cab\_test.pickup\_datetime.str[0:4]

cab\_test['Month']= cab\_test.pickup\_datetime.str[5:7]

cab\_test['Date']= cab\_test.pickup\_datetime.str[8:10]

cab\_test['Hour']= cab\_test.pickup\_datetime.str[11:13]

cab\_test['Minute']= cab\_test.pickup\_datetime.str[14:16]

cab\_test['Seconds'] = cab\_test.pickup\_datetime.str[17:19]

In [89]:

###Verify the shape

cab\_test.dtypes

Out[89]:

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count int64

Year object

Month object

Date object

Hour object

Minute object

Seconds object

dtype: object

In [90]:

cab\_test.head(5)

Out[90]:

|  | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2015-01-27 13:08:24 UTC | -73.973320 | 40.763805 | -73.981430 | 40.743835 | 1 | 2015 | 01 | 27 | 13 | 08 | 24 |
| **1** | 2015-01-27 13:08:24 UTC | -73.986862 | 40.719383 | -73.998886 | 40.739201 | 1 | 2015 | 01 | 27 | 13 | 08 | 24 |
| **2** | 2011-10-08 11:53:44 UTC | -73.982524 | 40.751260 | -73.979654 | 40.746139 | 1 | 2011 | 10 | 08 | 11 | 53 | 44 |
| **3** | 2012-12-01 21:12:12 UTC | -73.981160 | 40.767807 | -73.990448 | 40.751635 | 1 | 2012 | 12 | 01 | 21 | 12 | 12 |
| **4** | 2012-12-01 21:12:12 UTC | -73.966046 | 40.789775 | -73.988565 | 40.744427 | 1 | 2012 | 12 | 01 | 21 | 12 | 12 |

In [91]:

###Convert new variables created to type numeric since they are of type object

cab\_test['Year'] = pd.to\_numeric(cab\_test['Year'],errors = "coerce")

cab\_test['Month'] = pd.to\_numeric(cab\_test['Month'],errors = "coerce")

cab\_test['Date'] = pd.to\_numeric(cab\_test['Date'],errors = "coerce")

cab\_test['Hour'] = pd.to\_numeric(cab\_test['Hour'],errors = "coerce")

cab\_test['Minute'] = pd.to\_numeric(cab\_test['Minute'],errors = "coerce")

cab\_test['Seconds'] = pd.to\_numeric(cab\_test['Seconds'],errors = "coerce")

In [92]:

cab\_test.dtypes

Out[92]:

pickup\_datetime object

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count int64

Year int64

Month int64

Date int64

Hour int64

Minute int64

Seconds int64

dtype: object

In [93]:

#Calculate distance using Haversine formula

cab\_test['distance'] = cab\_test[['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']].apply(get\_distance,axis=1)

In [94]:

###Remove the pickup\_longitude, pickup\_latitude, dropoff\_longitude , dropoff\_latitude, pickup\_datetime variables as they are not needed

cab\_test.drop(['pickup\_longitude','pickup\_latitude','dropoff\_longitude', 'dropoff\_latitude','pickup\_datetime'], axis=1, inplace=True)

In [95]:

cab\_test.head(2)

Out[95]:

|  | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 2015 | 1 | 27 | 13 | 8 | 24 | 2.323259 |
| **1** | 1 | 2015 | 1 | 27 | 13 | 8 | 24 | 2.425353 |

In [96]:

cab\_test.shape

Out[96]:

(9914, 8)

In [97]:

missing\_val = pd.DataFrame(cab\_test.isnull().sum())

In [98]:

missing\_val

Out[98]:

|  | **0** |
| --- | --- |
| **passenger\_count** | 0 |
| **Year** | 0 |
| **Month** | 0 |
| **Date** | 0 |
| **Hour** | 0 |
| **Minute** | 0 |
| **Seconds** | 0 |
| **distance** | 0 |

In [99]:

#There are no missing values in cab\_test

In [100]:

##Check for any negative values

cab\_test["passenger\_count"].sort\_values(ascending=True)

Out[100]:

0 1

4616 1

4615 1

4614 1

4613 1

4612 1

4611 1

4610 1

4609 1

4608 1

4607 1

4606 1

4605 1

4604 1

4603 1

4602 1

4601 1

4600 1

4599 1

4598 1

4597 1

4596 1

4595 1

4594 1

4617 1

4593 1

4618 1

4620 1

4643 1

4642 1

..

9818 6

9819 6

9820 6

9821 6

9822 6

9814 6

9803 6

9802 6

9801 6

9782 6

9783 6

9784 6

9785 6

9786 6

9787 6

9788 6

9789 6

9790 6

9791 6

9792 6

9793 6

9794 6

9795 6

9796 6

9797 6

9798 6

9799 6

9800 6

9780 6

9913 6

Name: passenger\_count, Length: 9914, dtype: int64

In [101]:

##Similarily check for distance variable

cab\_test["distance"].sort\_values(ascending=True)

Out[101]:

6335 0.000000

7637 0.000000

2258 0.000000

943 0.000000

8835 0.000000

860 0.000000

9221 0.000000

6157 0.000000

2936 0.000000

2247 0.000000

1189 0.000000

318 0.000000

491 0.000000

8424 0.000000

8426 0.000000

5443 0.000000

498 0.000000

2618 0.000000

976 0.000000

6954 0.000000

7874 0.000000

1409 0.000000

2675 0.000000

981 0.000000

451 0.000000

7839 0.000000

1429 0.000000

1218 0.000000

2641 0.000000

121 0.000000

...

4018 21.738063

7857 21.782684

8944 21.785143

8876 21.811484

7000 21.845869

5523 21.880030

628 21.880433

6260 22.064573

7346 22.080922

8825 22.113966

5373 22.132067

248 22.344419

2718 22.407984

7787 22.474632

8173 22.640229

2175 23.051119

5905 23.099448

5357 23.120250

5865 23.443658

2487 25.364627

706 25.565980

4864 27.169511

9391 27.287881

5115 27.653128

3569 27.863332

4334 28.837177

7269 33.604366

5887 97.240975

8529 98.192419

4080 99.996040

Name: distance, Length: 9914, dtype: float64

In [102]:

#####Since distance in kms cannot be zero hence we remove those values as well

cab\_test = cab\_test.drop(cab\_test[cab\_test['distance']<=0].index, axis=0)

In [103]:

cab\_test.shape

Out[103]:

(9829, 8)

In [104]:

cab\_test.dtypes

Out[104]:

passenger\_count int64

Year int64

Month int64

Date int64

Hour int64

Minute int64

Seconds int64

distance float64

dtype: object

In [105]:

plt.boxplot(cab\_test['passenger\_count'])

Out[105]:

{'whiskers': [<matplotlib.lines.Line2D at 0x218ffe46fd0>,

<matplotlib.lines.Line2D at 0x21895940668>],

'caps': [<matplotlib.lines.Line2D at 0x218959409b0>,

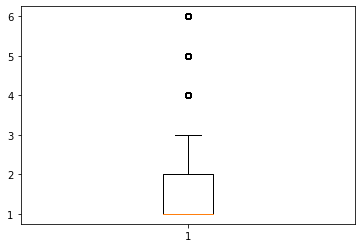
<matplotlib.lines.Line2D at 0x21895940cf8>],

'boxes': [<matplotlib.lines.Line2D at 0x218ffe46e80>],

'medians': [<matplotlib.lines.Line2D at 0x21895940dd8>],

'fliers': [<matplotlib.lines.Line2D at 0x218959503c8>],

'means': []}



In [106]:

### Since cab capacity is from 1 to 6 we wont go for any outlier detection for passenger\_count variable

In [107]:

plt.boxplot(cab\_test['distance'])

Out[107]:

{'whiskers': [<matplotlib.lines.Line2D at 0x218957a6208>,

<matplotlib.lines.Line2D at 0x21895798b38>],

'caps': [<matplotlib.lines.Line2D at 0x218957987f0>,

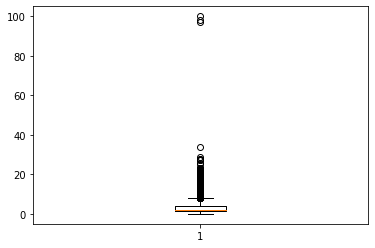
<matplotlib.lines.Line2D at 0x218957984a8>],

'boxes': [<matplotlib.lines.Line2D at 0x218957a6358>],

'medians': [<matplotlib.lines.Line2D at 0x21895798160>],

'fliers': [<matplotlib.lines.Line2D at 0x21895798080>],

'means': []}



In [108]:

cab\_test.shape

Out[108]:

(9829, 8)

In [109]:

#Detect and remove outliers for distance variable

e75, f25 = np.percentile(cab\_test['distance'], [75,25])

iqr = e75 - f25

min3 = f25 - (iqr\*1.5)

max3 = e75 + (iqr\*1.5)

cab\_test = cab\_test.drop(cab\_test[cab\_test['distance'] < min3].index, axis = 0)

cab\_test = cab\_test.drop(cab\_test[cab\_test['distance'] > max3].index, axis = 0)

In [110]:

cab\_test.shape

Out[110]:

(9013, 8)

In [111]:

#Normalisation of Data

cab\_test\_columns = list(cab\_test.columns.values)

for i in cab\_test\_columns:

cab\_test[i] = (cab\_test[i] - min(cab\_test[i]))/(max(cab\_test[i]) - min(cab\_test[i]))

In [112]:

cab\_test.head(2)

Out[112]:

|  | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.0 | 1.0 | 0.0 | 0.866667 | 0.565217 | 0.135593 | 0.40678 | 0.284350 |
| **1** | 0.0 | 1.0 | 0.0 | 0.866667 | 0.565217 | 0.135593 | 0.40678 | 0.296853 |

In [113]:

#We have cleaned the test data and completed the different exploratory analysis.

#Now Predict fare\_amount on test data provided using Random Forest Model

predictions\_RF\_test = model\_RF.predict(cab\_test)

In [114]:

###Create new variable to plot the data

cab\_test['Predicted\_fare\_amount'] = predictions\_RF\_test

In [115]:

cab\_test.head(5)

Out[115]:

|  | **passenger\_count** | **Year** | **Month** | **Date** | **Hour** | **Minute** | **Seconds** | **distance** | **Predicted\_fare\_amount** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.0 | 1.000000 | 0.000000 | 0.866667 | 0.565217 | 0.135593 | 0.406780 | 0.284350 | 0.432694 |
| **1** | 0.0 | 1.000000 | 0.000000 | 0.866667 | 0.565217 | 0.135593 | 0.406780 | 0.296853 | 0.429036 |
| **2** | 0.0 | 0.333333 | 0.818182 | 0.233333 | 0.478261 | 0.898305 | 0.745763 | 0.075582 | 0.198533 |
| **3** | 0.0 | 0.500000 | 1.000000 | 0.000000 | 0.913043 | 0.203390 | 0.203390 | 0.239988 | 0.337320 |
| **4** | 0.0 | 0.500000 | 1.000000 | 0.000000 | 0.913043 | 0.203390 | 0.203390 | 0.659606 | 0.543531 |

In [116]:

cab\_test.to\_csv('final\_results.csv')

In [ ]: