Cab Fare Prediction

Vinayak Chaturvedi

Date: 27-May-2019

Contents

Chapter 1
Introduction

1.1 Problem Statement

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

1.2 Data Set

Number of attributes:

1. pickup_datetime - timestamp value indicating when the cab ride started.
2. pickup_longitude - float for longitude coordinate of where the cab ride started.
3. pickup_latitude - float for latitude coordinate of where the cab ride started.
4. dropoff_longitude - float for longitude coordinate of where the cab ride ended.
5. dropoff_latitude - float for latitude coordinate of where the cab ride ended.
6. passenger_count - an integer indicating the number of passengers in the cab ride.

Target Variable: Fare Amount

Missing Values: Yes

Sample Data:

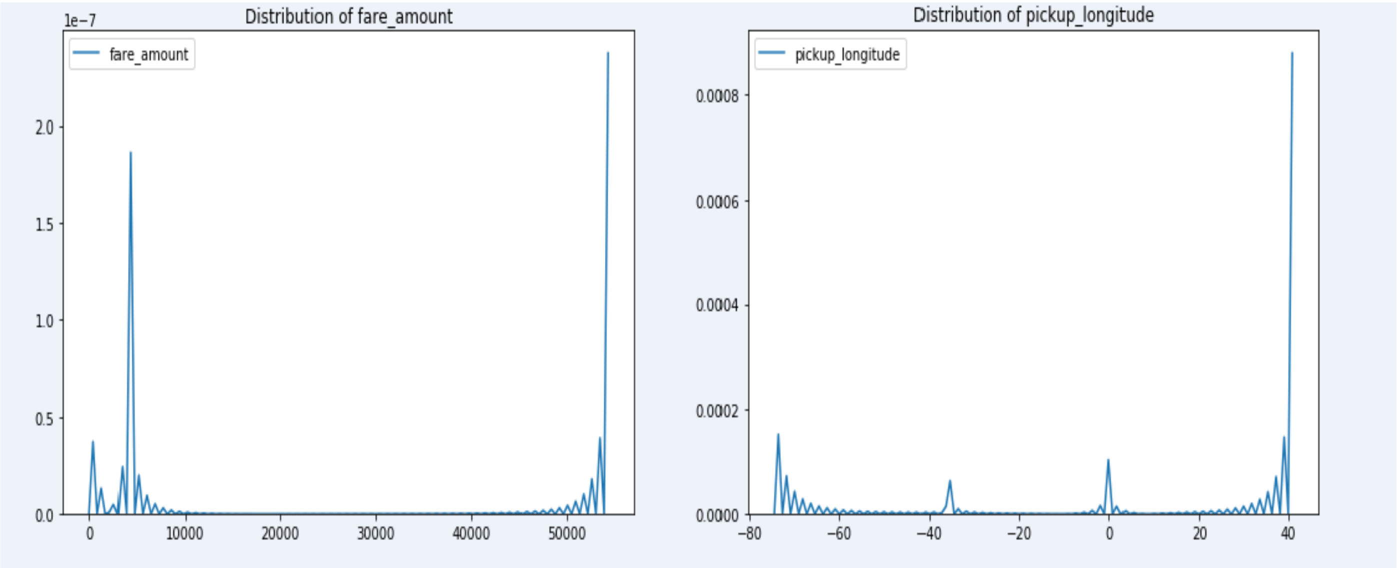
	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1.0
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.0
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	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

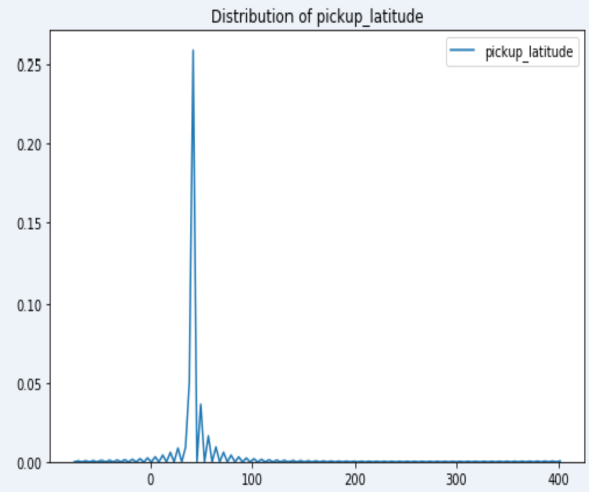
Chapter 2
Methodology

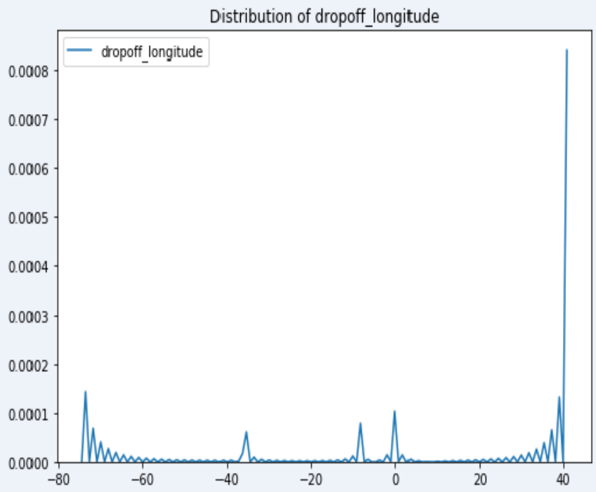
2.1 Pre Processing

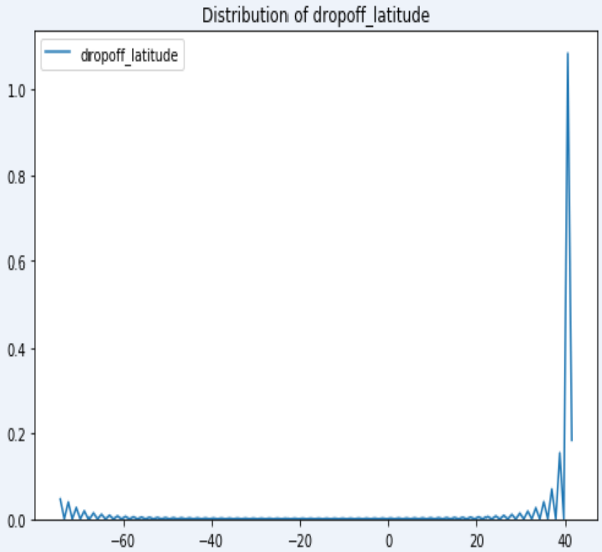
Any predictive modelling requires that we look at the data before we start modelling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process, we will first try and look at all the distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize our data and check its distribution:

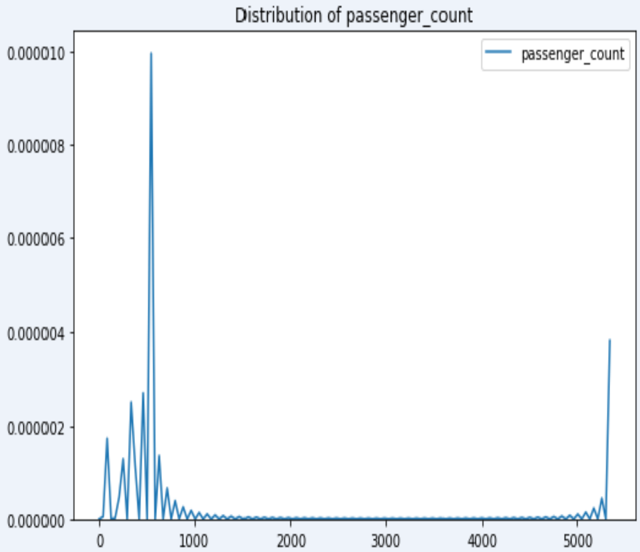
2.1.1 Data Distribution:



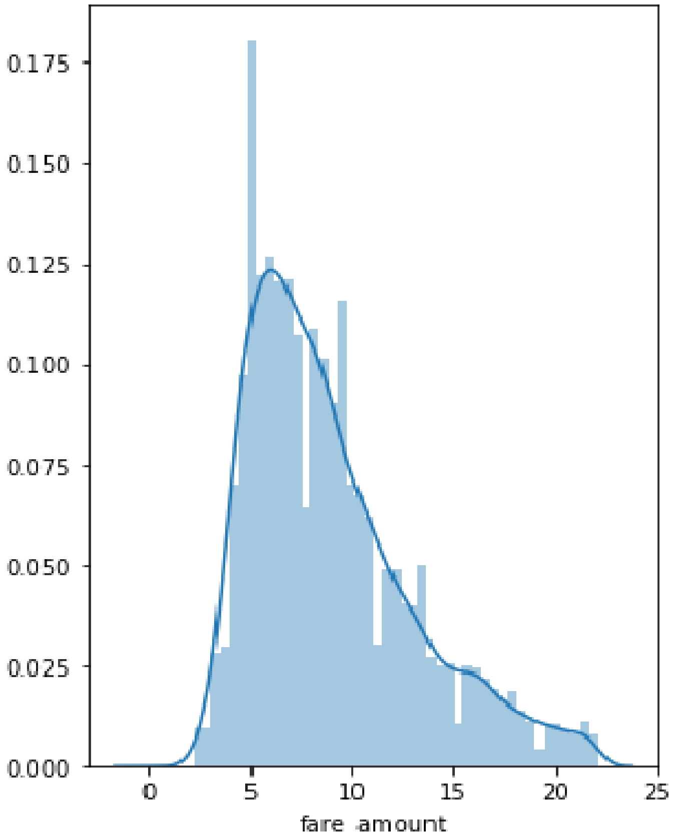

Distribution of pickup_latitude
pickup_latitude
0.00
0.05
0.10
0.15
0.20
0.25
0
100
200
300
400

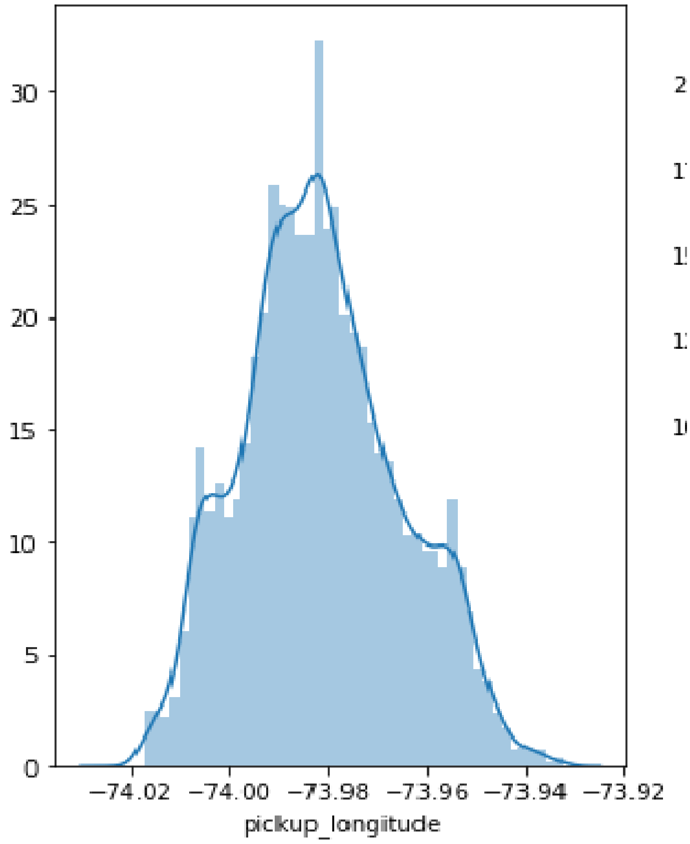

Distribution of dropoff_longitude
dropoff_longitude
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
0.0008
−80
−60
−40
−20
0
20
40

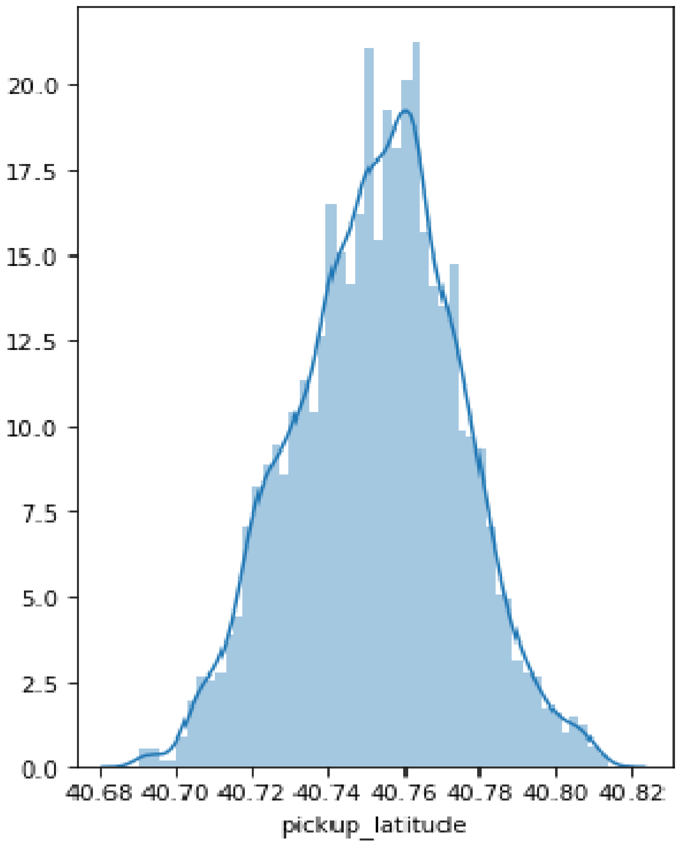

Distribution of dropoff_latitude
dropoff_latitude
0.0
0.2
0.4
0.6
0.8
1.0
−60
−40
−20
0
20
40

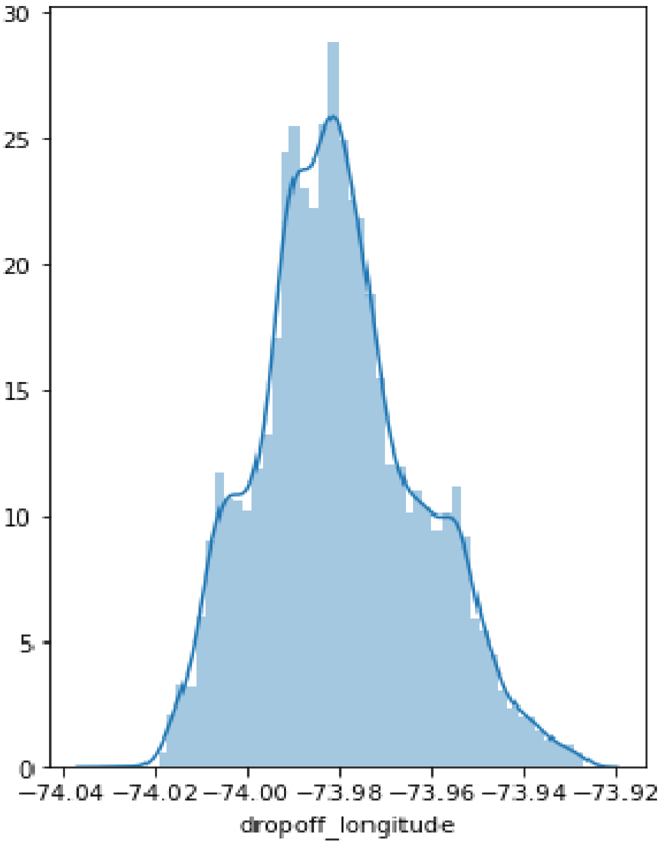

Distribution of passenger_count
passenger_count
0.000000
0.000002
0.000004
0.000006
0.000008
0.000010
0
1000
2000
3000
4000
5000

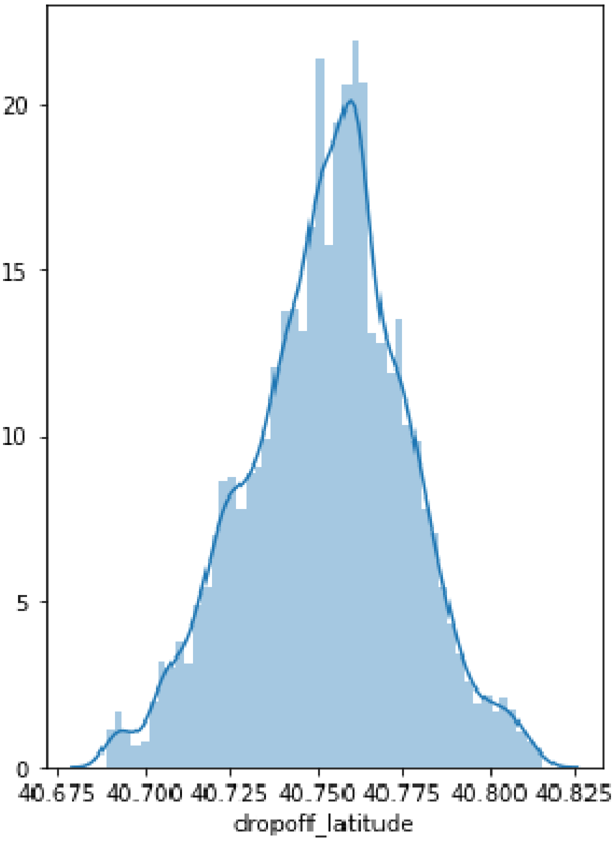
After Log Scale and some data pre-processing techniques:

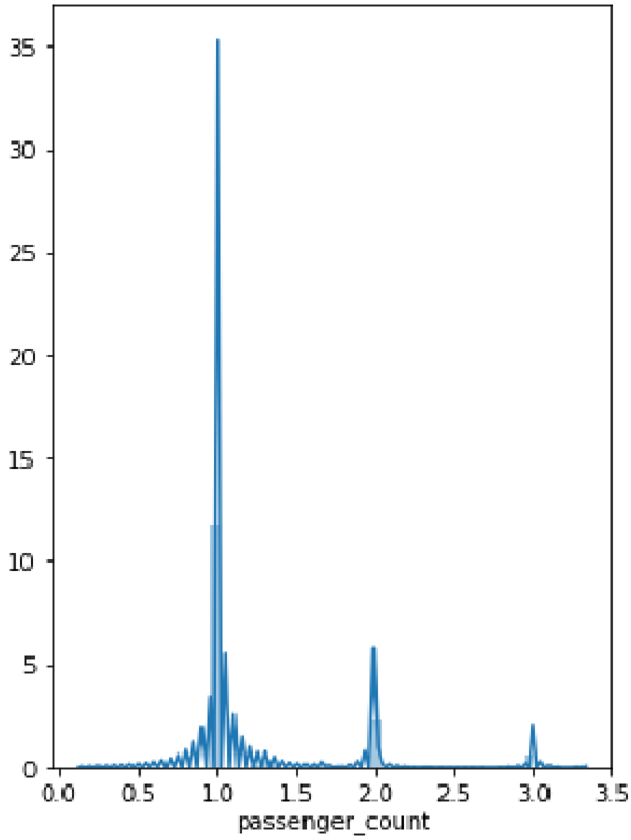










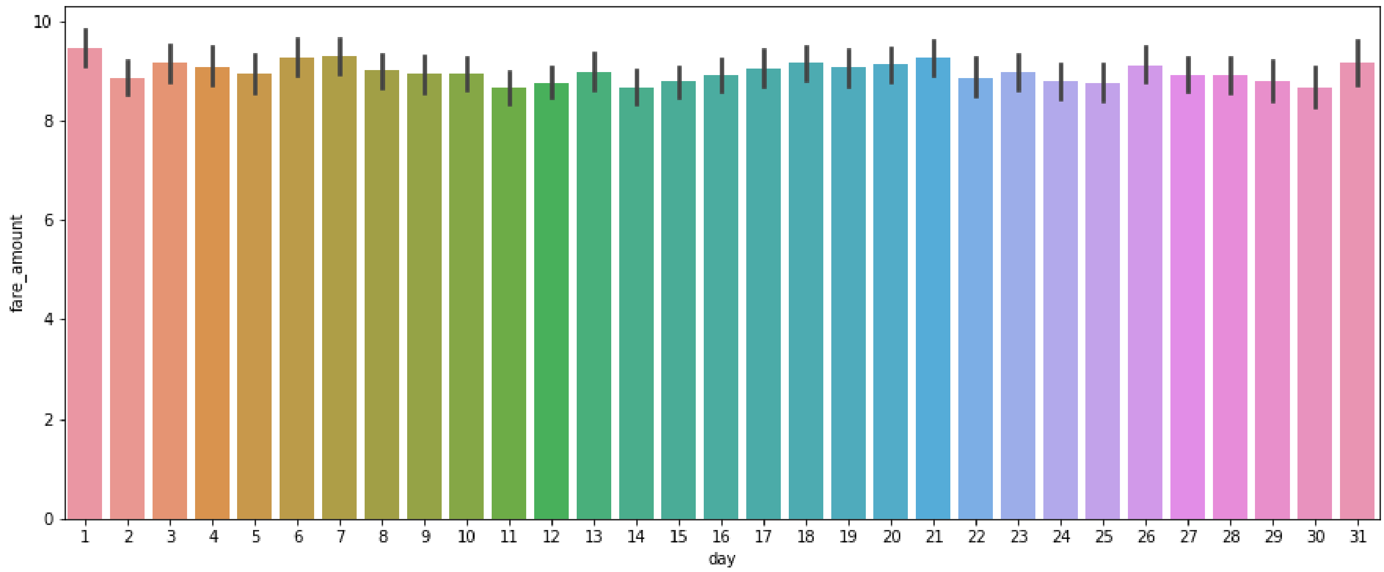


As we can see in the above graphs variables are normally distributed

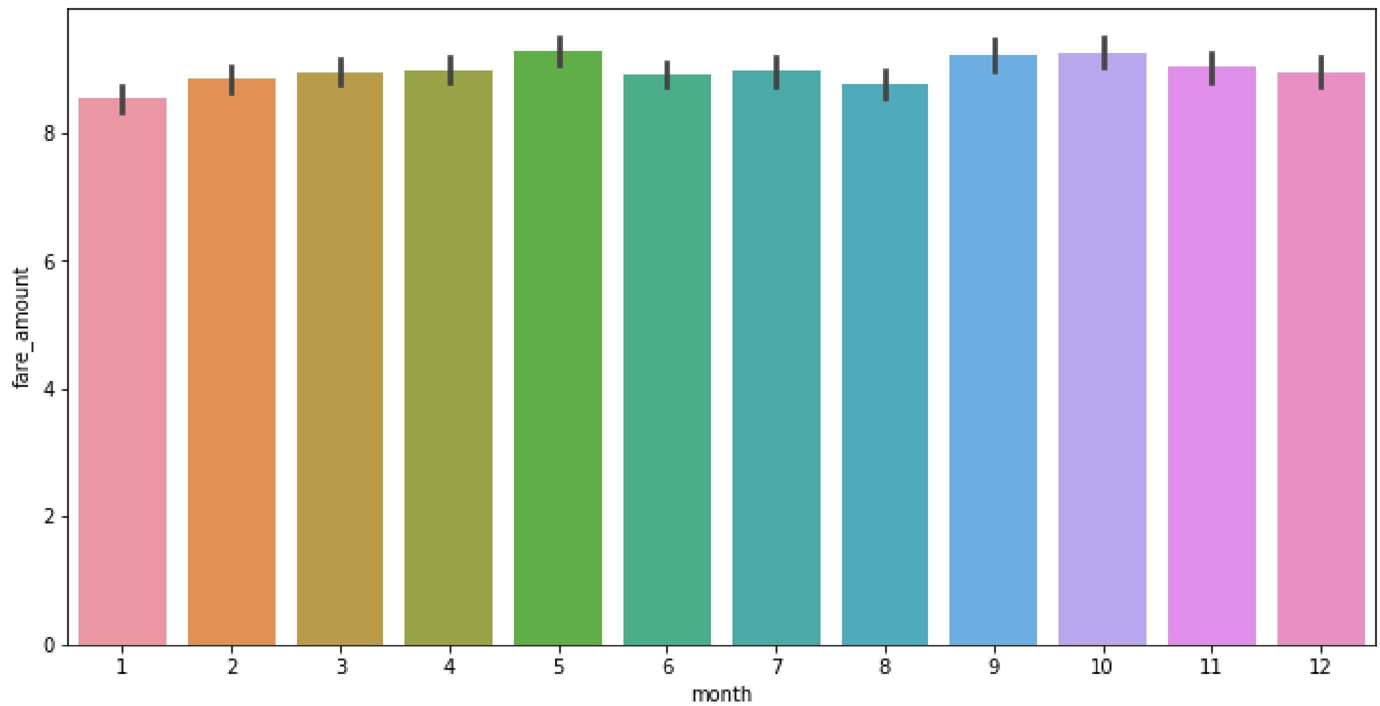
2.1.2 Data Preparation

There is 1 variable in the dataset – “pickup_datatime” which contains various information inside it, basically it is a combination of day, month, year, day of week, hour, minute, seconds.
We can divide this variable into “day, month, year, day of week, hour”.
Relation between these categorical variables and target variable:

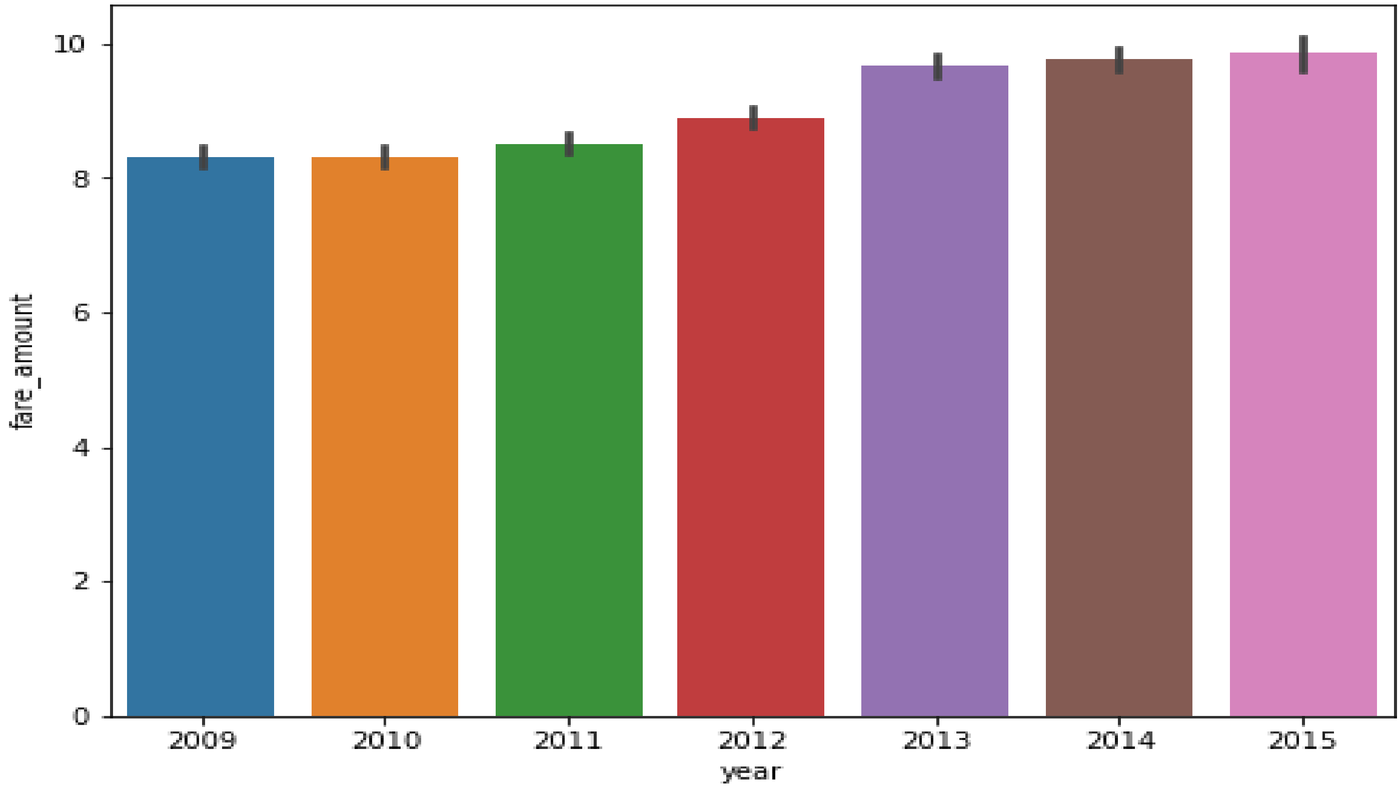
1. Day vs fare amount



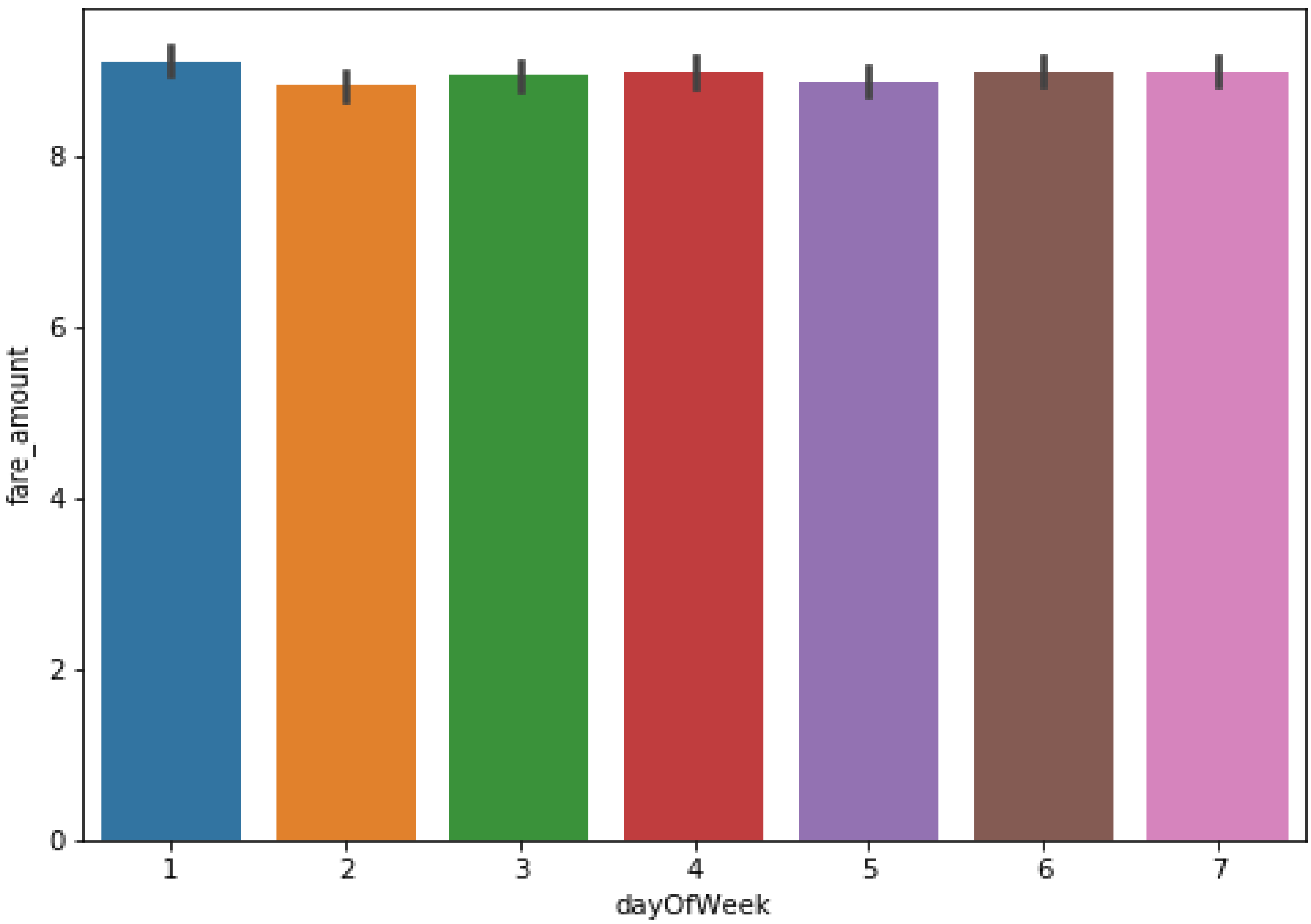
2. Month vs fare amount



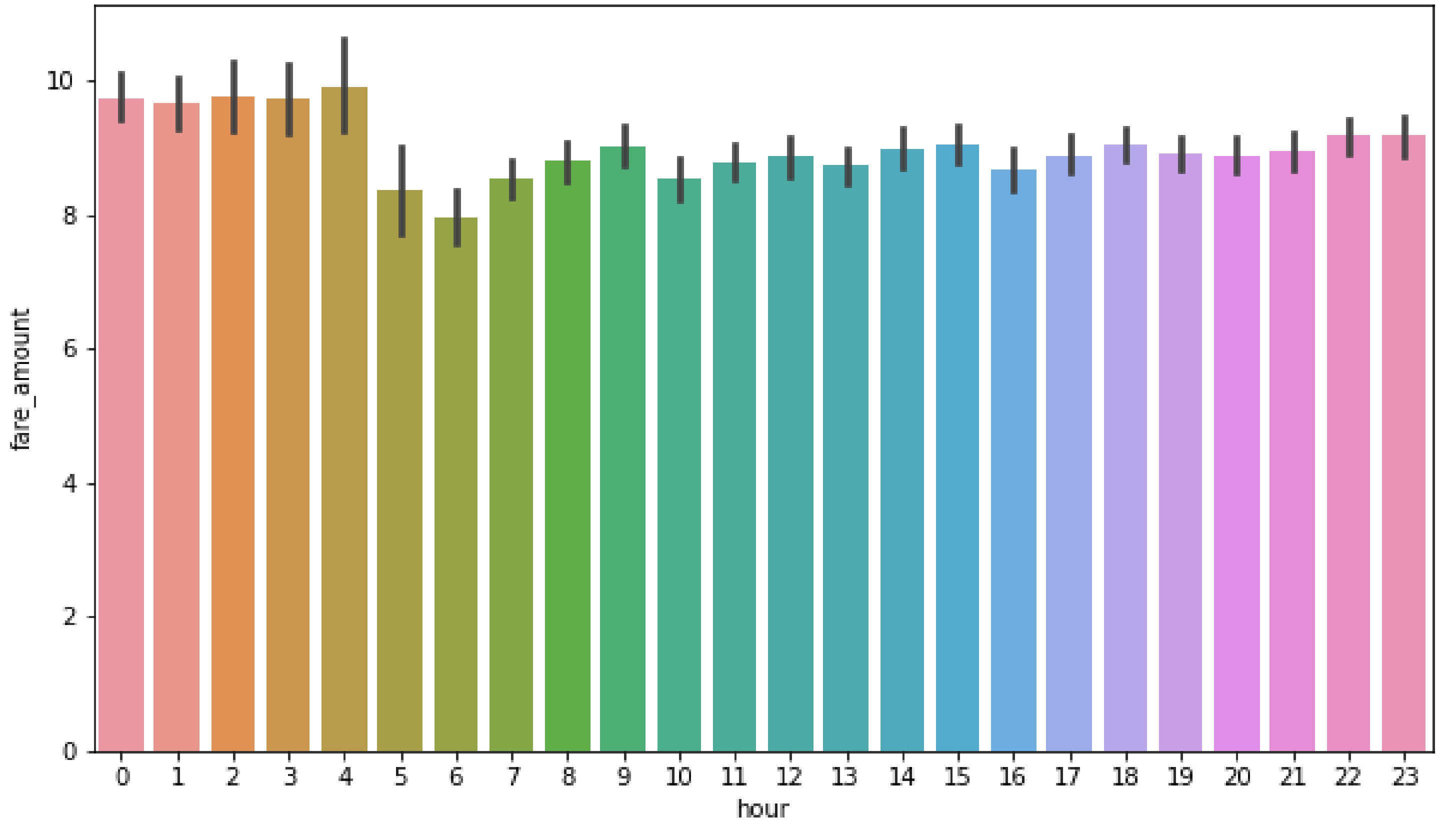
3. Year vs fare amount



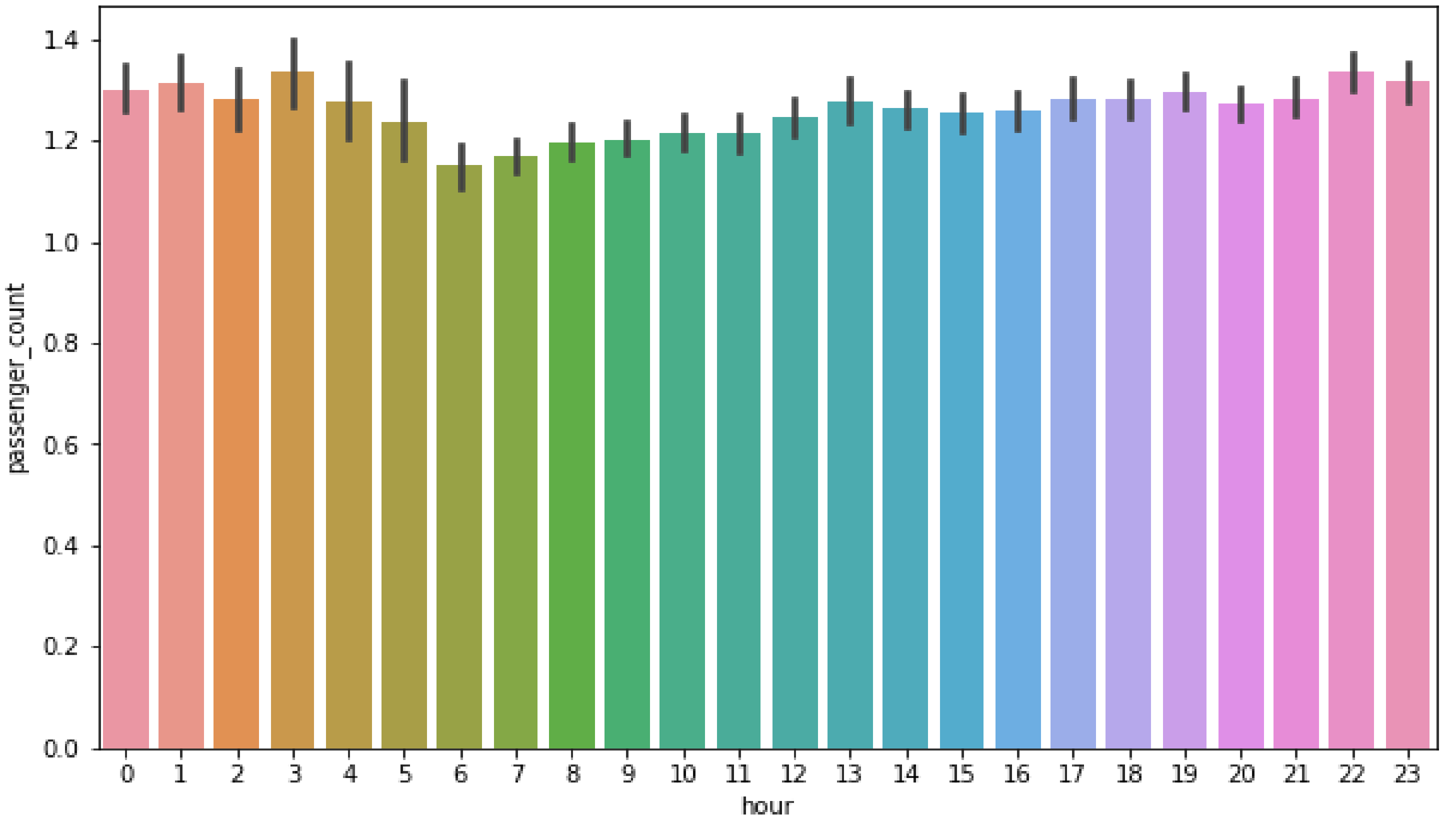
4. Day of Week vs Fare Amount



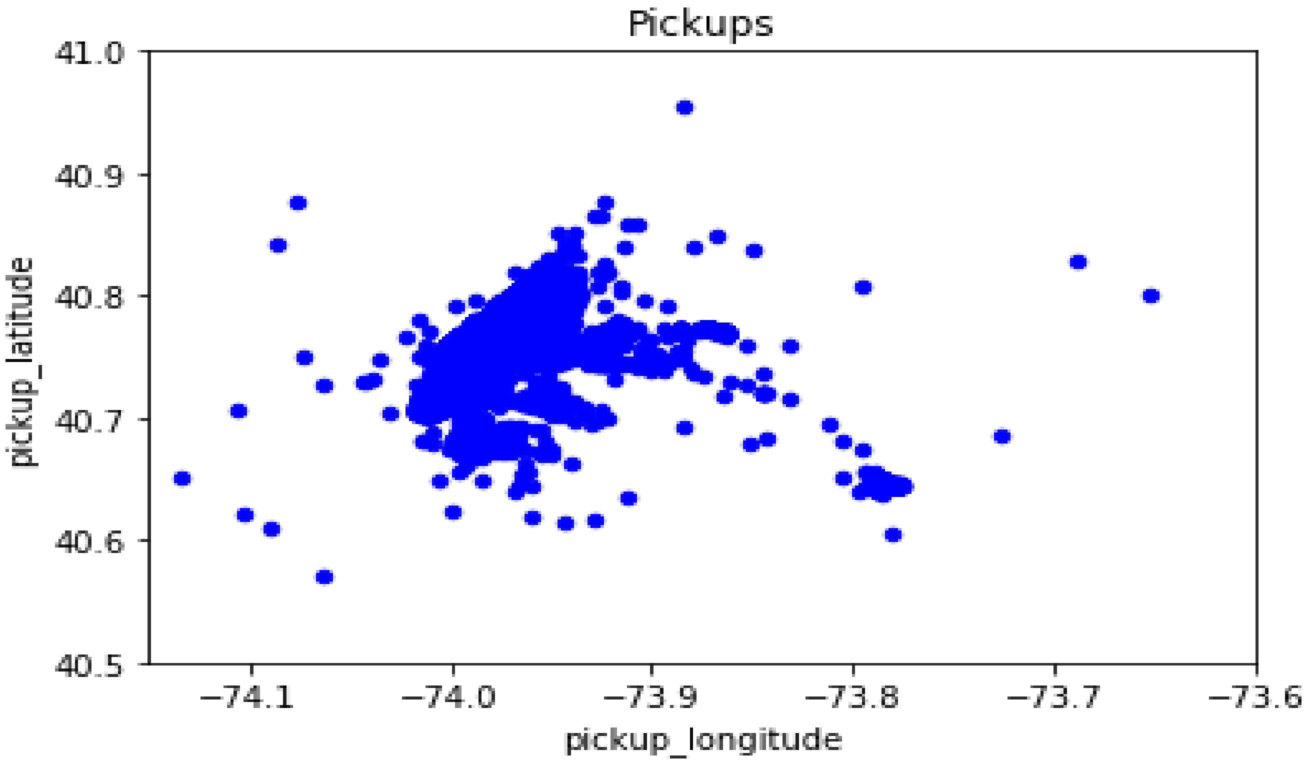
5. Hour vs Fare Amount

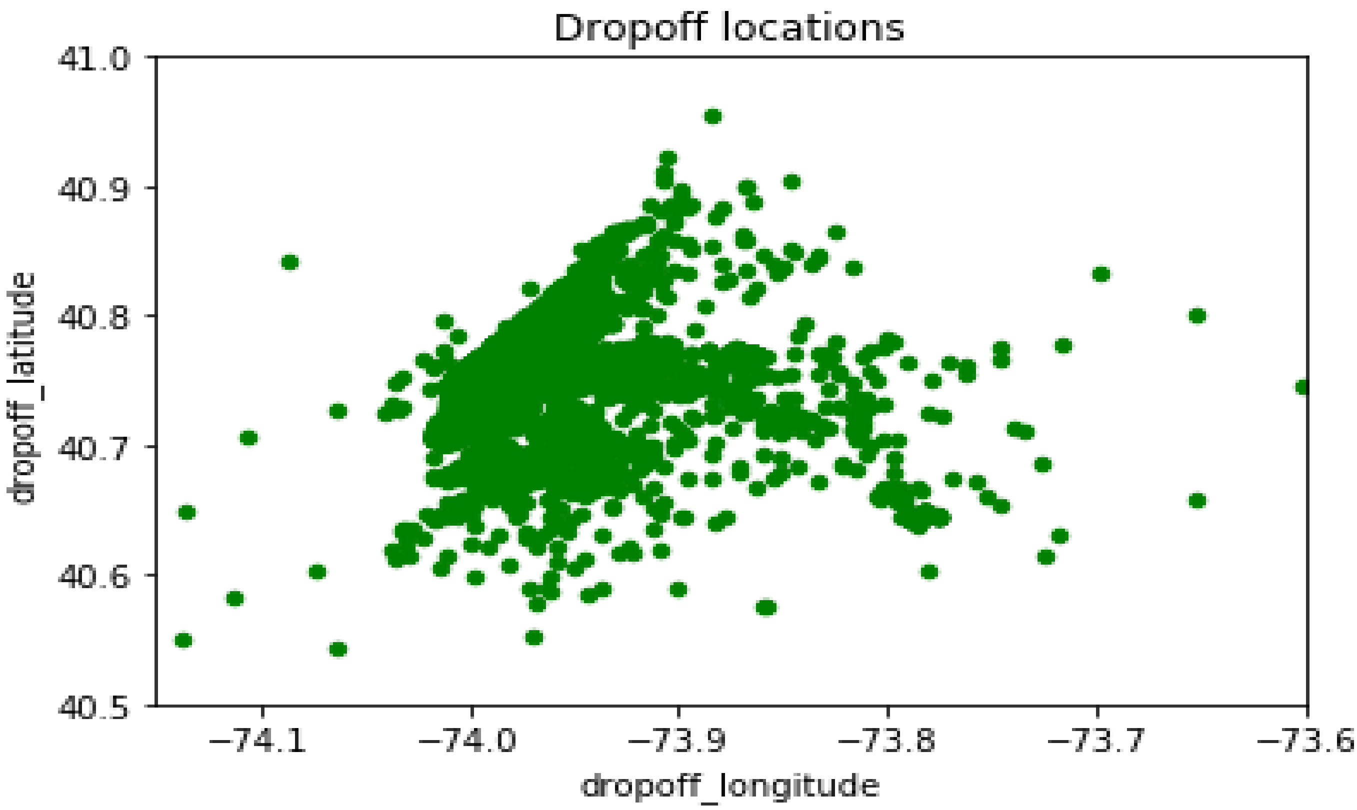


6. Hour vs Passenger Count



Scatter Plot for Pickup and Drop off latitude and longitude





2.1.3 Data Statistics

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16067.000000	16067.000000	16067.000000	16067.000000	16067.000000	16067.000000
mean	8.974418	-73.981496	40.753012	-73.980026	40.753183	1.262620
std	4.084775	0.016023	0.021403	0.017202	0.022772	0.518473
min	-3.000000	-74.023050	40.690497	-74.029461	40.688839	0.120000
25%	6.000000	-73.992457	40.738646	-73.991532	40.738758	1.000000
50%	8.000000	-73.982410	40.753935	-73.981474	40.754767	1.000000
75%	11.000000	-73.971146	40.767729	-73.969479	40.768073	1.000000
max	22.100000	-73.932250	40.813695	-73.927115	40.815770	3.333366

2.1.4 Missing Value Analysis:

In the dataset there are several observations present which contains missing values
So to deal with missing values we can fill the missing values using various techniques:

1. Mean
2. Median
3. KNN Imputation

To find the best fit technique, we can replace any 1 known observation and then we will apply all 3 techniques and then we will check which is giving us the best result:

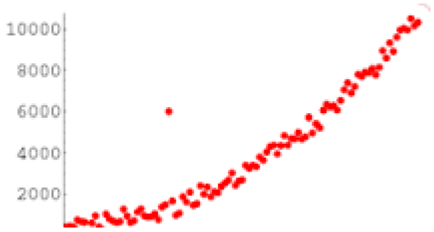
Eg:

```
Actual       Mean     Median      KNN
-74.00096 -72.46269 -73.98170 -73.99202
```

So here we can see that KNN is given us the exact value so we will use KNN imputation to fill the missing values.

2.1.5 Outlier Analysis:

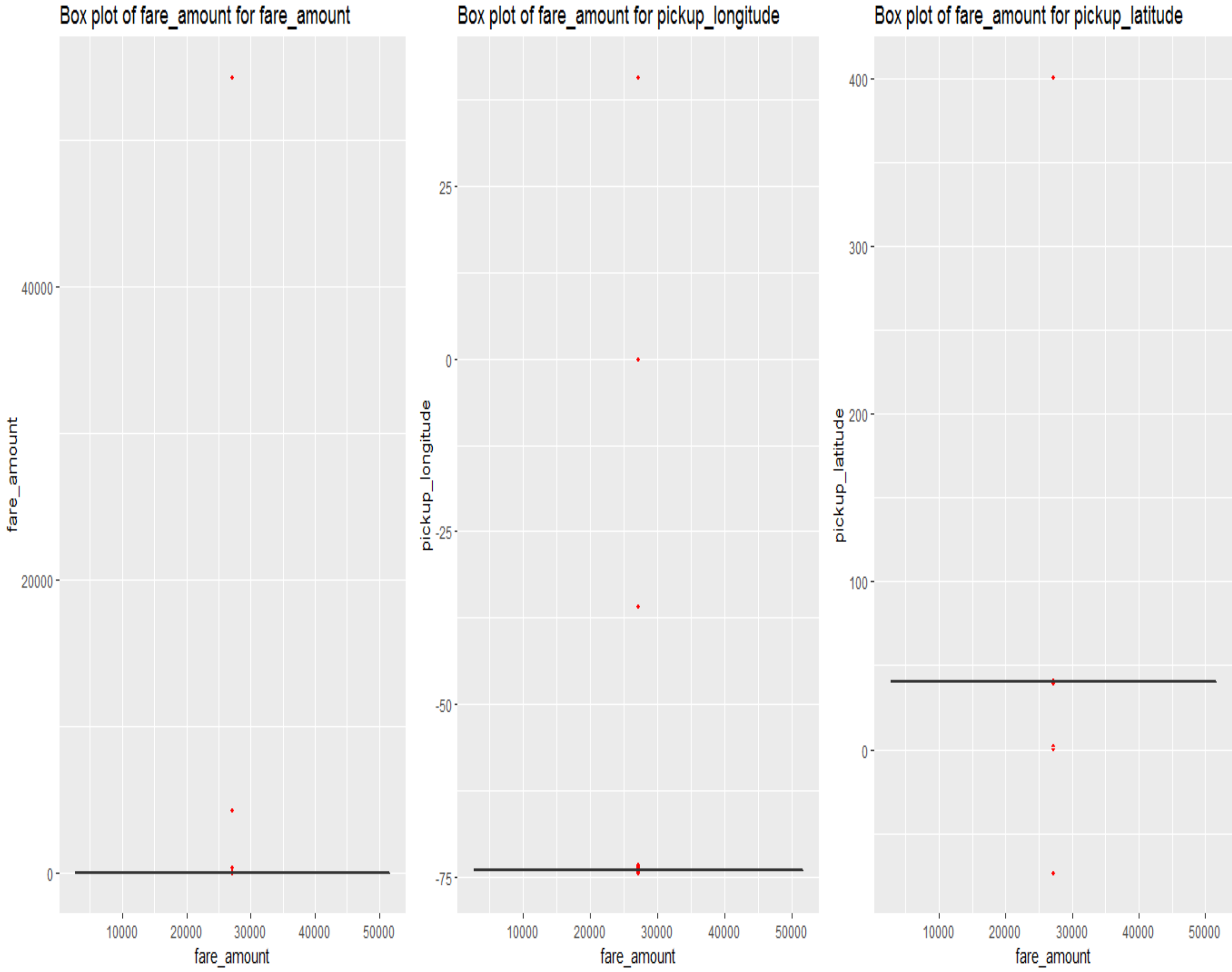
One of the important steps in data pre-processing is outlier analysis. In statistics, an Outlier is an observation point that is distant from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set. Outliers are only present in continuous variable not in categorical variable.

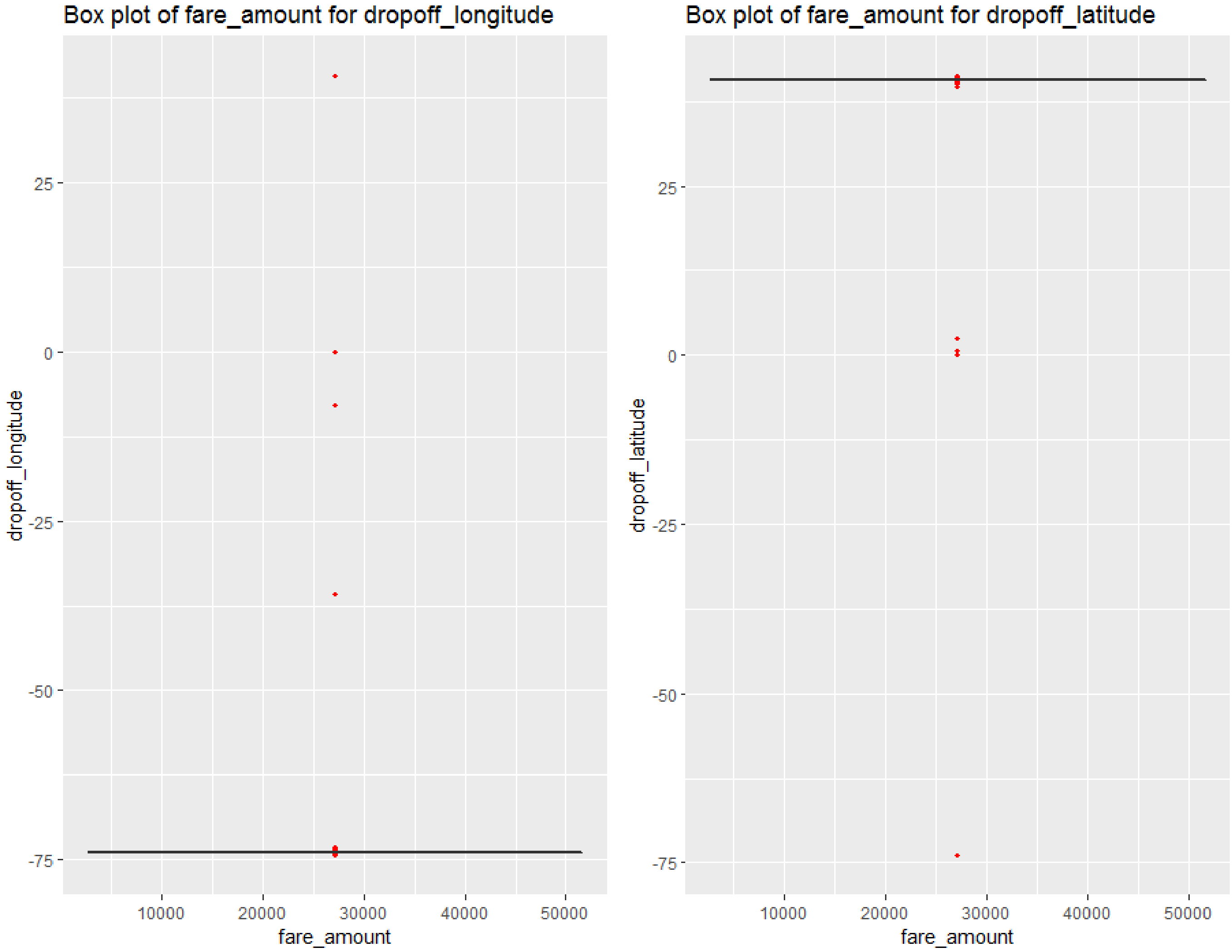


An **outlier** can cause serious problems in statistical analyses.
So to prevent these problems we need to take care of outliers that means we need to either remove that observation or replace the outlier using missing values and then refill them using missing values analysis.

First we need to identify outliers:

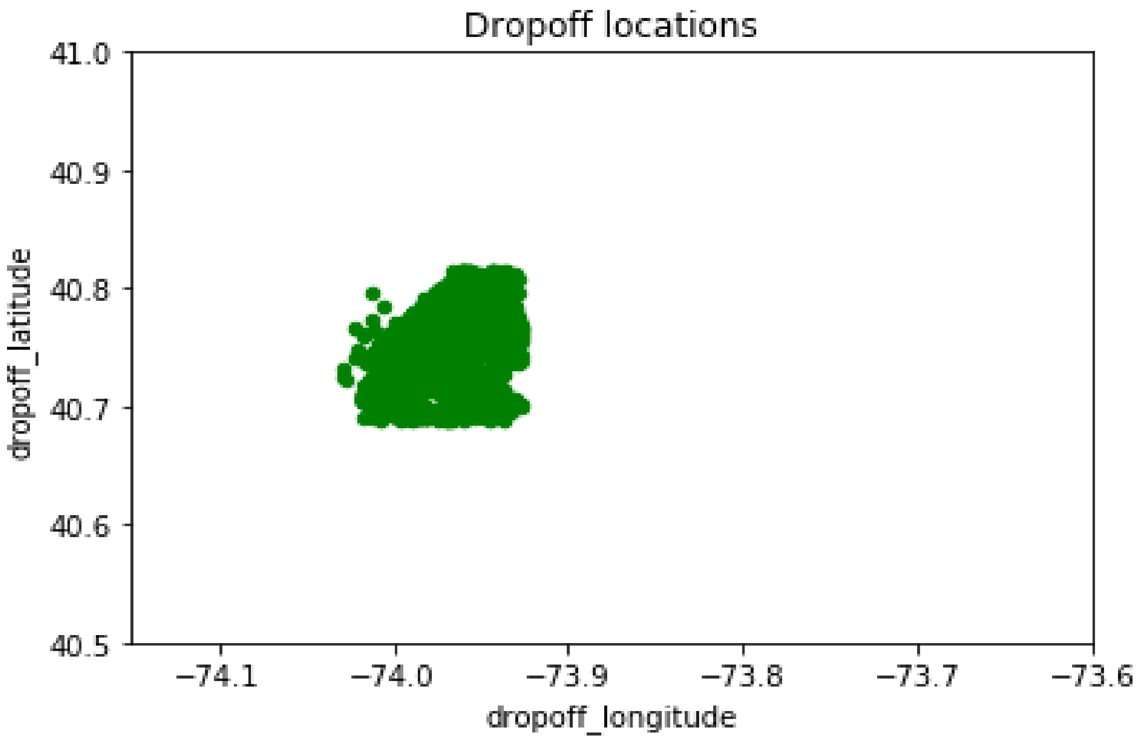
We can identify using **Boxplot.** In the Boxplot the points that are present above the header line or below the tailor line are the outliers

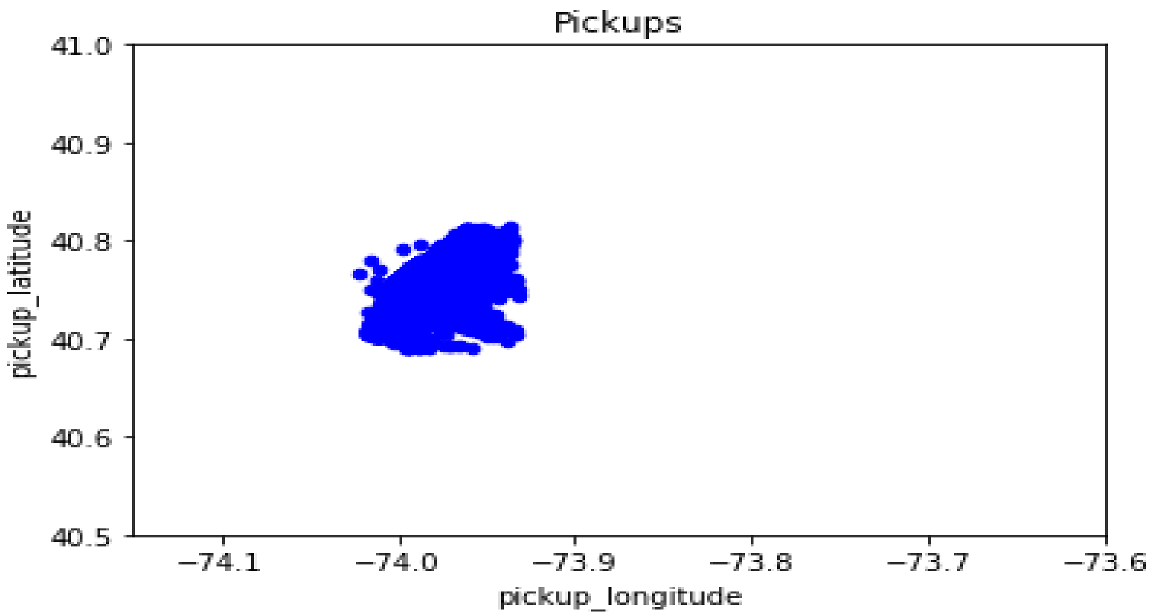




So, for treating the outliers we replace the outlier values with NA and then, fill the NA with KNN imputation

After treating outliers we can see the major difference in the scatter plot of pickup and dropoff latitude and longtitude.

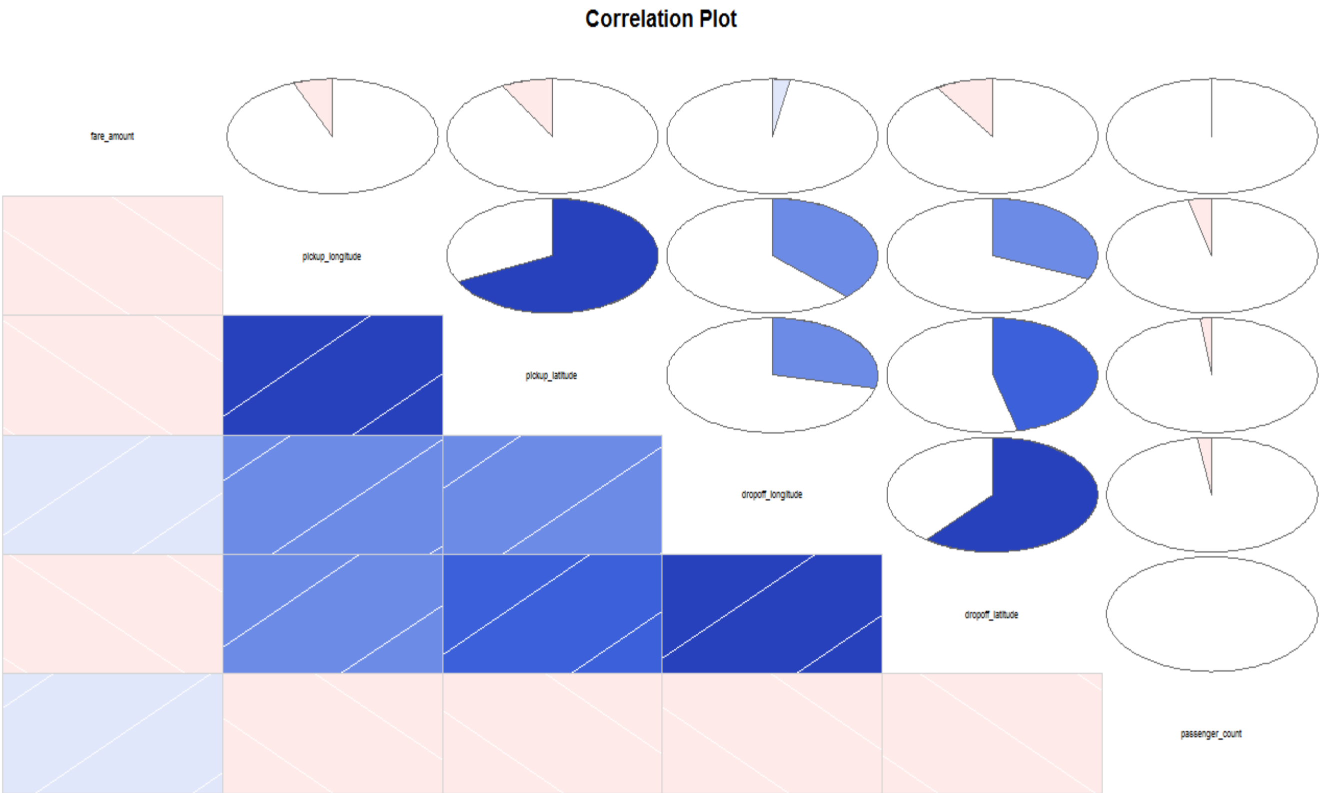




2.1.6 Feature Selection:

Before performing any type of modelling we need to make sure that all the variables which we will use in the model must have unique characteristics there should be no correlation between independent variables so here I used correlation plot to identify the correlation between continuous variable: (Python)

(In R)



There is one variable in the dataset “pickup_dataTime” – we already divide it into day, month, year, hour, day of week, so we can remove it.
And in the above graph we can see no variable is fully correlated to any other variable. So keep all the variables.

2.1.7 Feature Scaling:

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization.
Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without feature scaling
As we have seen in the above graphs that the continuous variables are normally distributed. So we apply standardisation technique to solve the problem of varying range scale.
Standardisation:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Where x is the original feature vector, x bar is the mean of that feature vector and sigma is the standard deviation.

2.2 Modelling

2.2.1 Model Selection:

Our dataset contains 2 types of variable:

1. Continuous
2. Categorical

So identify which machine learning gives us the best result we will apply all the algorithm and then will compare the results with an evaluation metric

First we will divide our dataset into 2 parts:

1. Training set
2. Validation Set

We train our model using training set and test the model performance using validation set.

Models:

1. **Mutiple Linear Regression**: Multiple linear regression is the most common form of linear regression analysis. As a predictive analysis, the multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables.

   Assumptions of Linear Regression:
   a. Data should be normally distributed
   b. No multi-collinearity.

   RMSE = 4.00

   Multiple linear regression is good for linear and simple data but sometimes it does not give good predictions when data is more complex. So we can use Decision tree and random forest to predict the value of target variable.

2. **Decision Tree:** Highly interpretable classification or regression model that splits data-feature values into branches at decision nodes until a final decision output is made. It allows input variables to be a mixture of continuous and categorical variables.

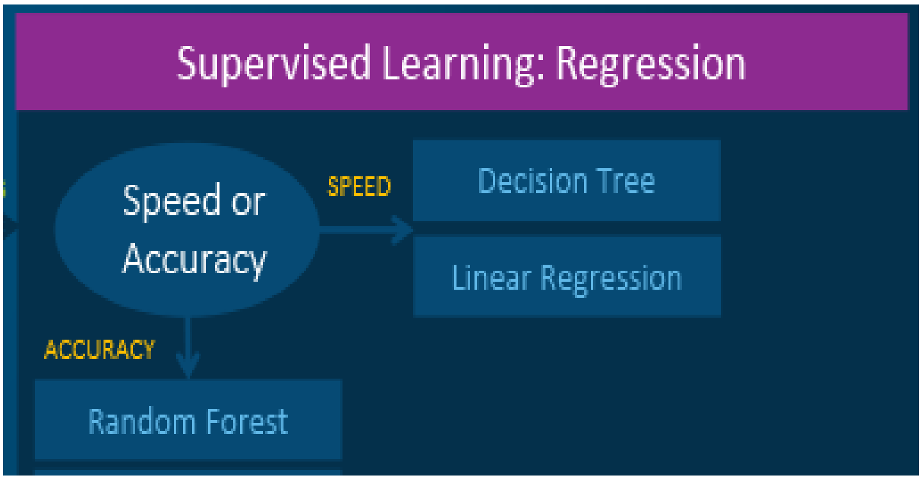
   RMSE = 3.56

3. **<u>Random Forest:</u>** Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

   RMSE = 2.60

Advantages of Decision Tree and Random Forest:

- Great at learning complex, highly non-linear relationships. They usually can achieve pretty high performance, better than polynomial regression and often on par with neural networks.
- Very easy to interpret and understand. Although the final trained model can learn complex relationships, the decision boundaries that are built during training are easy and practical to understand.



Chapter 3
Conclusion

3.1 Model Evaluation

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. Computational Efficiency

In our case of Dataset, the latter one, Computation Efficiency, does 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.

Mean Absolute Error (MAE) and **Root mean squared error (RMSE)** are two of the most common metrics used to measure accuracy for continuous variables.

1. **<u>Mean Absolute Error (MAE):</u>** MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$

2. **<u>Root Mean Squared Error (RMSE):</u>** is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Similarities: Both MAE and RMSE express average model prediction error in units of the variable of interest. Both metrics can range from 0 to ∞ and are indifferent to the direction of errors. They are negatively-oriented scores, which means lower values are better.

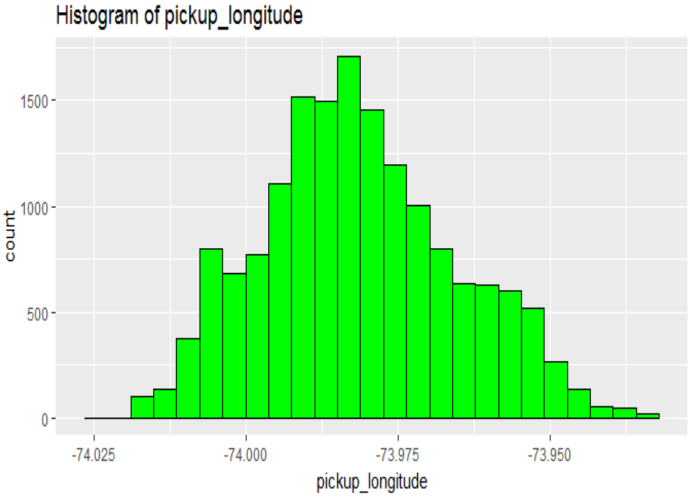
Differences: Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable.

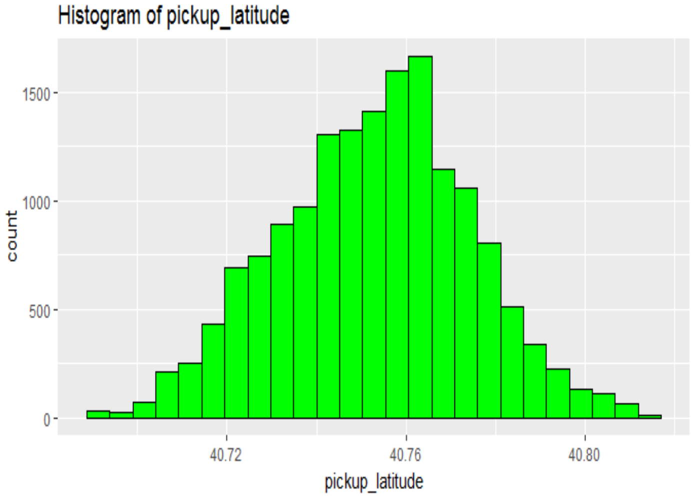
Which one to choose?
RMSE has the benefit of penalizing large errors more so can be more appropriate.
So, in our case, if we increase the cab fare by double then, we start losing the customer. That's why we use RMSE.

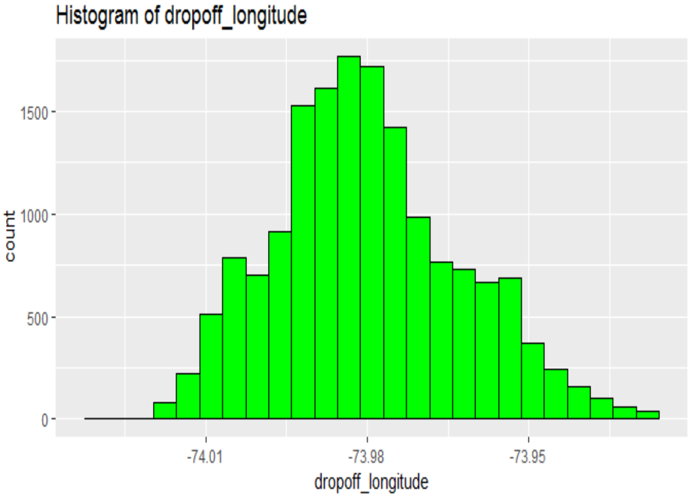
3.2 Model Selection

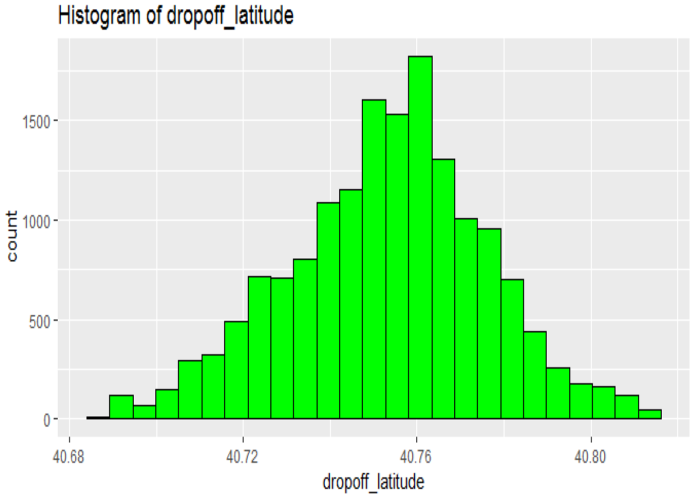
As we can see that in above results "**Random forest**" is giving us the best overall result, so we can use it.

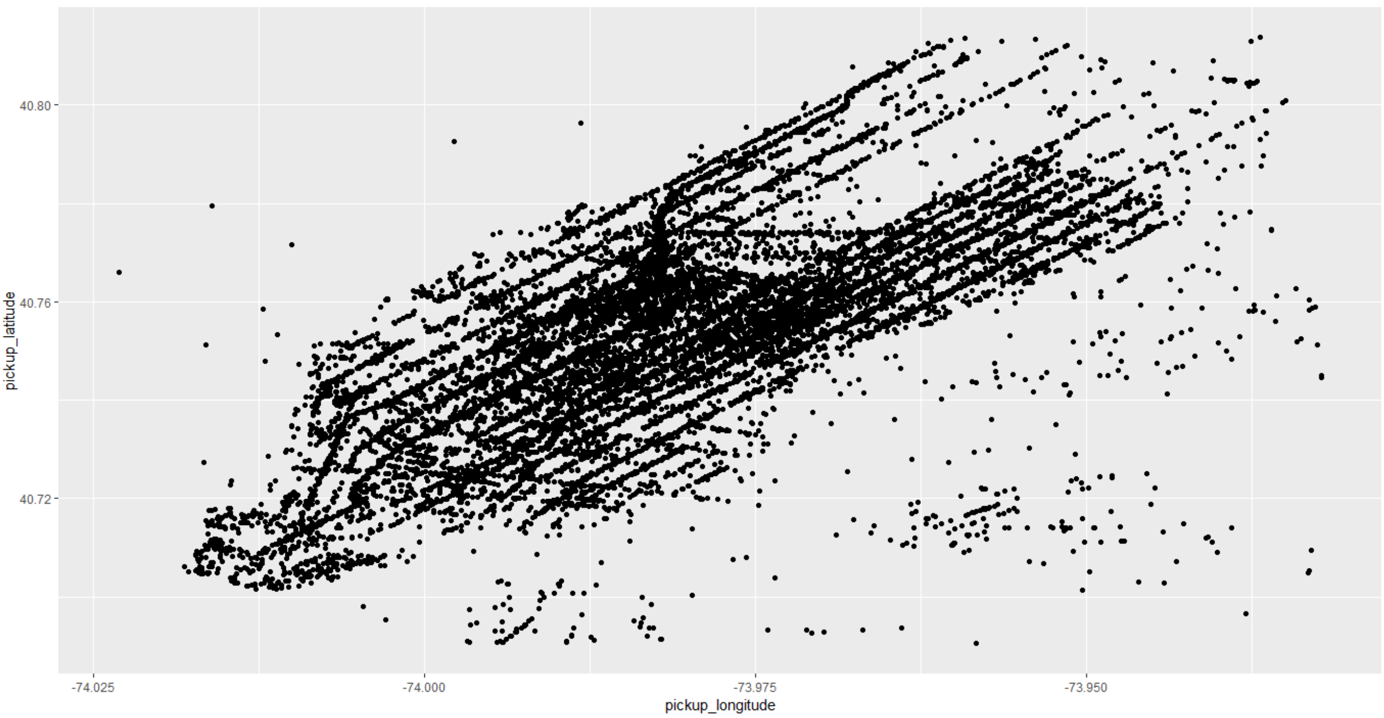
Appendix A - Extra Figures

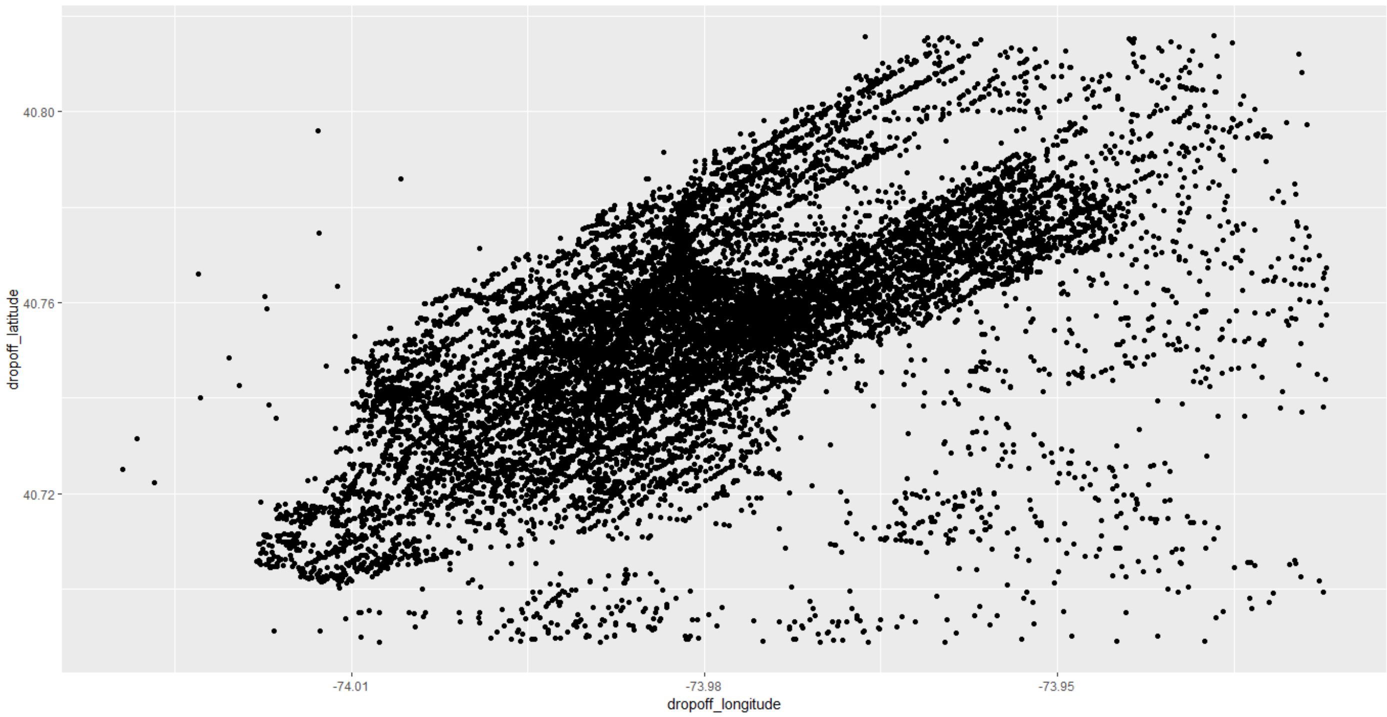


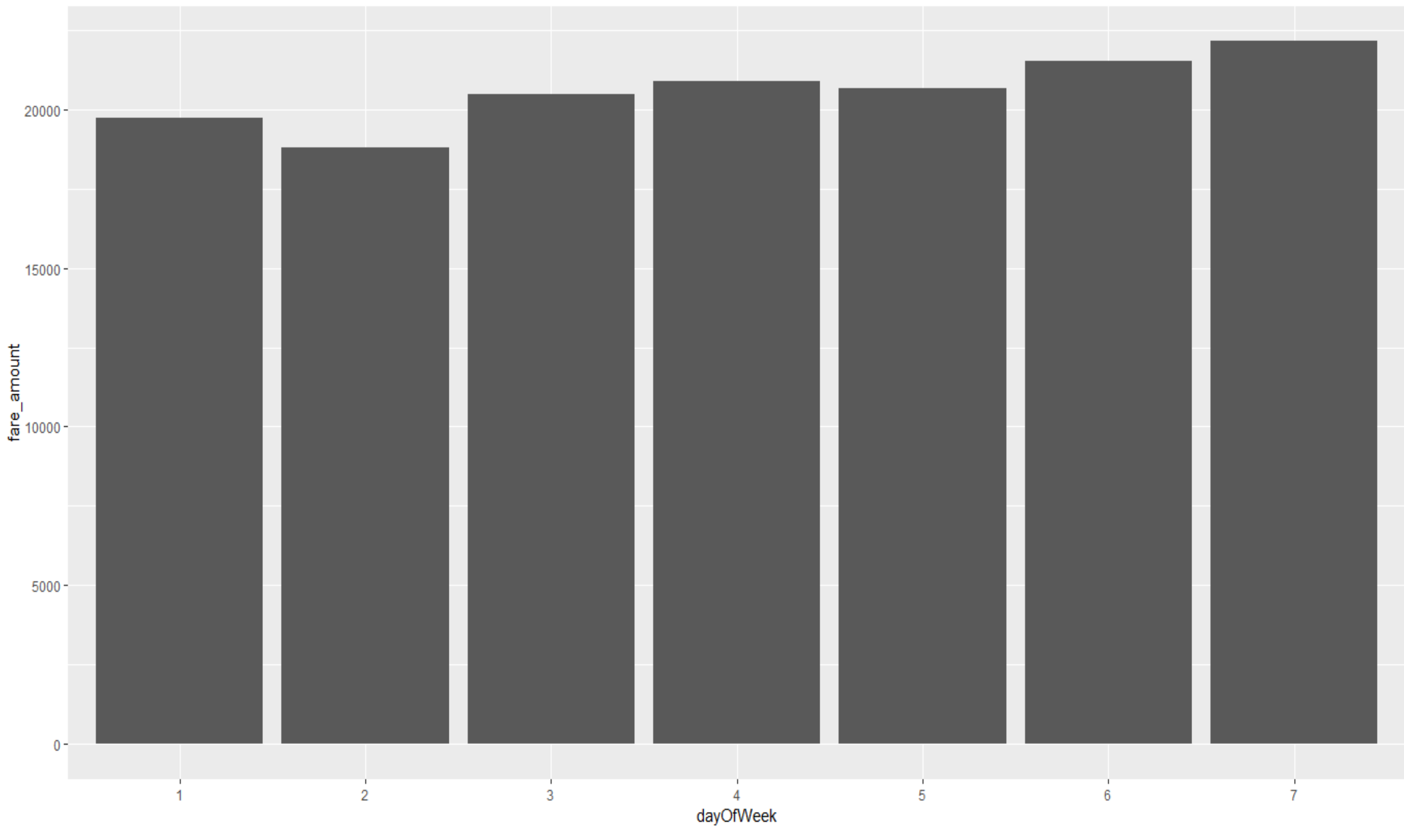


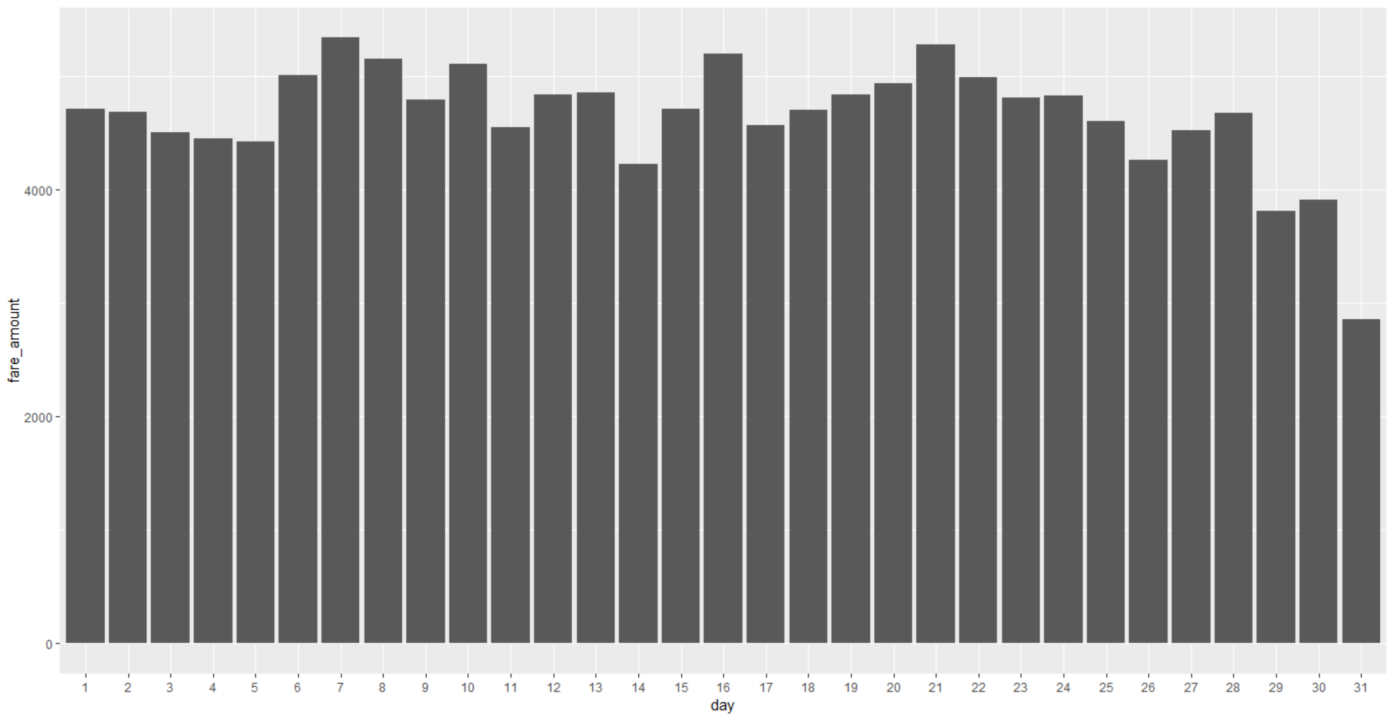


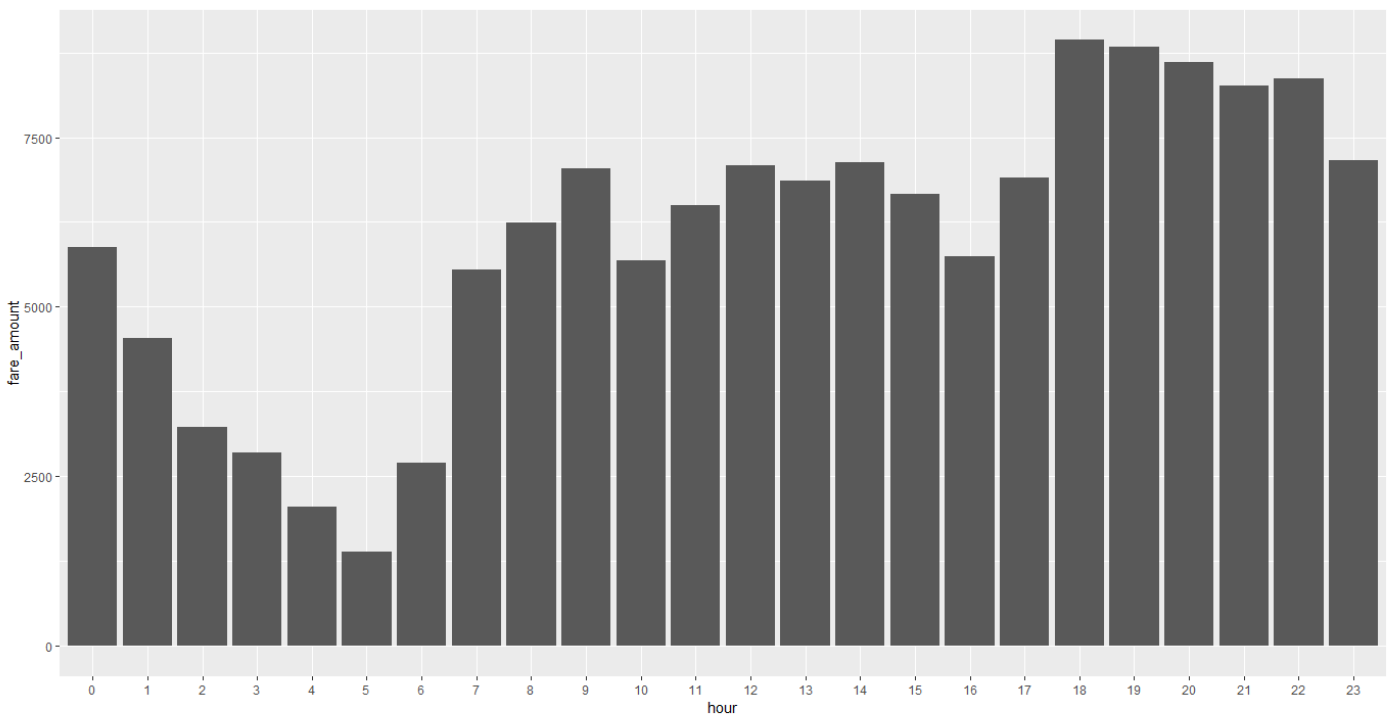


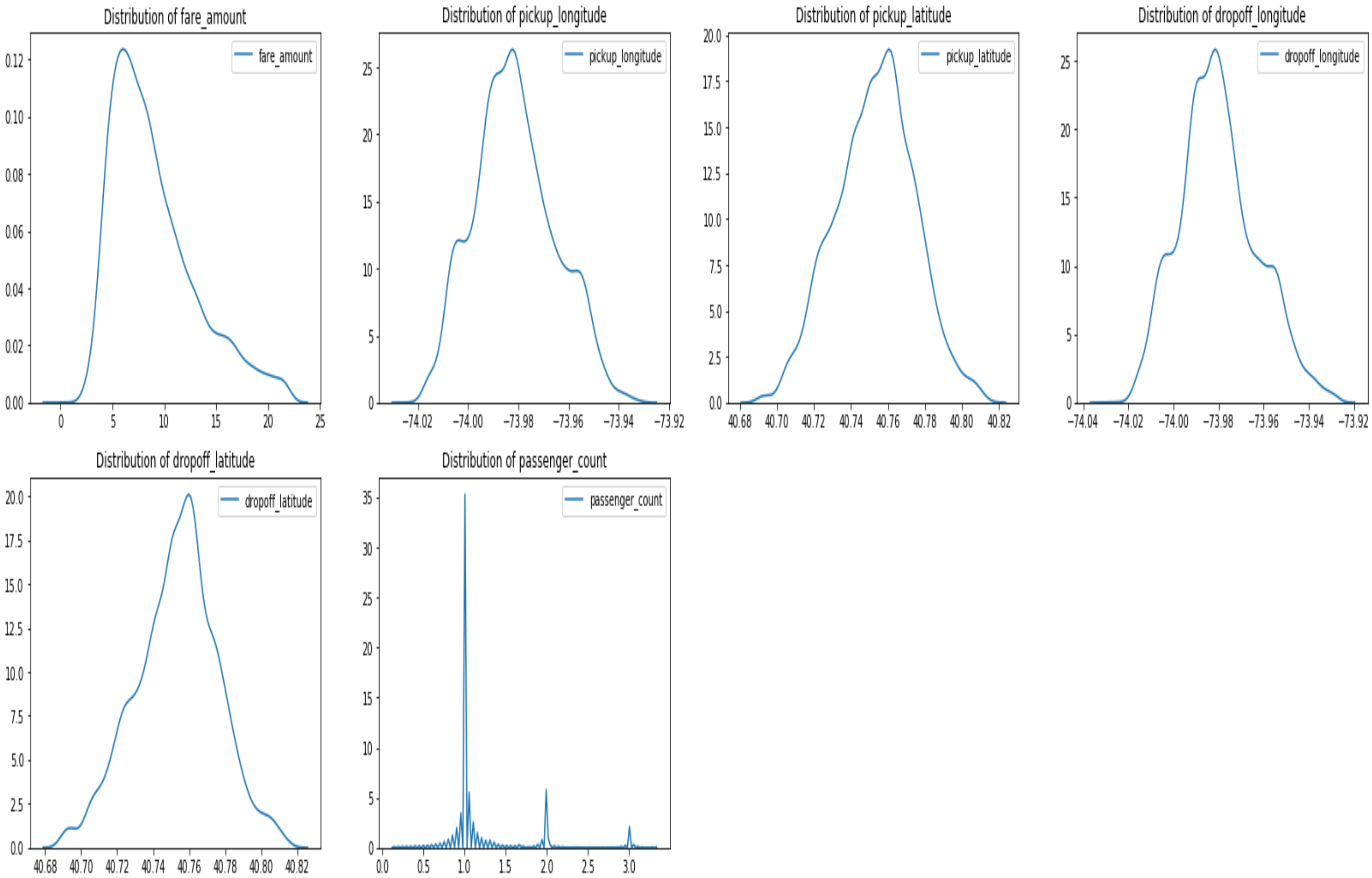



40.80
40.76
40.72
dropoff_latitude
-74.01
-73.98
-73.95
dropoff_longitude


20000
15000
10000
5000
0
fare_amount
1
2
3
4
5
6
7
dayOfWeek


fare_amount
4000
2000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
day


fare_amount
7500
5000
2500
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
hour


Distribution of fare_amount
fare_amount
Distribution of pickup_longitude
pickup_longitude
Distribution of pickup_latitude
pickup_latitude
Distribution of dropoff_longitude
dropoff_longitude
Distribution of dropoff_latitude
dropoff_latitude
Distribution of passenger_count
passenger_count

Appendix B- Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun May 19 23:45:55 2019

@author: vinayak
"""
#Load libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from fancyimpute import KNN
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from scipy.stats import chi2_contingency

################################### User defined Functions
##################################################
def divideDateTime(data):
    data['pickup_datetime'] = pd.to_datetime(data['pickup_datetime'], errors='coerce',
format='%Y-%m-%d %H:%M:%S UTC')
    #dividepickup_datetime into parts = date + month + year + day
    data['day']=data['pickup_datetime'].apply(lambda x:x.day)
    data['month']=data['pickup_datetime'].apply(lambda x:x.month)
    data['year']=data['pickup_datetime'].apply(lambda x:x.year)
    data['dayOfWeek']=data['pickup_datetime'].apply(lambda x:x.weekday())
    data['hour']=data['pickup_datetime'].apply(lambda x:x.hour)
    data = data.drop(["pickup_datetime"], axis=1)
    return data


def convertIntoProperDataTypes(data):
    if(data.shape[1]==11):
        cnumber_factor = [6,7,8,9,10]
    else:
        cnumber_factor = [5,6,7,8,9]
    for i in cnumber_factor:
        data.iloc[:,i] = data.iloc[:,i].astype('object')
```

```
    return data

def getCnamesFactor(data):
    lis = []
    for i in range(0, data.shape[1]):
        if(data.iloc[:,i].dtypes == 'object'):
            lis.append(data.columns[i])
    return lis

def getCnamesNumeric(data):
    lis = []
    for i in range(0, data.shape[1]):
        if(data.iloc[:,i].dtypes != 'object'):
            lis.append(data.columns[i])
    return lis

def getOptimalImputeMethod(data):
    bestFit = {'Actual':-73.99578100000001}
    data['pickup_longitude'].loc[70] = np.nan
    data_knn = pd.DataFrame.copy(data)
    data_mean= pd.DataFrame.copy(data)
    data_median= pd.DataFrame.copy(data)
    print(data_median['pickup_longitude'].loc[70])
    #Impute with mean
    data_mean['pickup_longitude'] = 
data_mean['pickup_longitude'].fillna(data_mean['pickup_longitude'].mean())
    print(data_median['pickup_longitude'].loc[70])
    #Impute with median
    data_median['pickup_longitude'] = 
data_median['pickup_longitude'].fillna(data_median['pickup_longitude'].median())
    #Impute with KNN
    data_knn = pd.DataFrame(KNN(k = 3) .fit_transform(data_knn), columns = 
data_knn.columns)

    bestFit['Using KNN'] = data_knn['pickup_longitude'].loc[70]
    bestFit['Using Mean'] = data_mean['pickup_longitude'].loc[70]
    bestFit['Using Median'] = data_median['pickup_longitude'].loc[70]
    return bestFit;

def imputeMissingValues(data, cnames_numeric, cnames_factor):
    for i in cnames_numeric:
        data[i] = data[i].replace(0, np.nan)

    #KNN imputation
```

```
    #Assigning levels to the categories
    lis = []
    for i in range(0, data.shape[1]):
        if(data.iloc[:,i].dtypes == 'object'):
            data.iloc[:,i] = pd.Categorical(data.iloc[:,i])
            data.iloc[:,i] = data.iloc[:,i].cat.codes
            data.iloc[:,i] = data.iloc[:,i].astype('object')
            lis.append(data.columns[i])
    #replace -1 with NA to impute
    for i in range(0, data.shape[1]):
        data.iloc[:,i] = data.iloc[:,i].replace(-1, np.nan)
    #Apply KNN imputation algorithm
    data = pd.DataFrame(KNN(k = 3) .fit_transform(data), columns = data.columns)
    #Convert into proper datatypes
    for i in lis:
        data.loc[:,i] = data.loc[:,i].round()
        data.loc[:,i] = data.loc[:,i].astype('object')
    data.passenger_count = data.passenger_count.round()
    #missing_val = pd.DataFrame(data.isnull().sum())
    return data

def checkOutlier(data, cnames_numeric):
    number_of_columns=len(cnames_numeric)
    number_of_rows = len(cnames_numeric)-1/number_of_columns
    plt.figure(figsize=(number_of_columns,5*number_of_rows))
    for i in range(0,len(cnames_numeric)):
        plt.subplot(number_of_rows + 1,number_of_columns,i+1)
        sns.set_style('whitegrid')
        sns.boxplot(data[cnames_numeric[i]],color='green',orient='v')
        plt.tight_layout()

def removeOutlier(data, cnames_numeric, cnames_factor):
    for i in cnames_numeric:
        q75, q25 = np.percentile(data.loc[:,i], [75 ,25])
        #Calculate IQR
        iqr = q75 - q25
        #Calculate inner and outer fence
        minimum = q25 - (iqr*1.5)
        maximum = q75 + (iqr*1.5)
        #Replace with NA
        data.loc[data.loc[:,i] < minimum,i] = np.nan
        data.loc[data.loc[:,i] > maximum,i] = np.nan
```

```
    #Apply KNN imputation algorithm
    data = pd.DataFrame(KNN(k = 3) .fit_transform(data), columns = data.columns)
    #Convert into proper datatypes
    for i in cnames_factor:
        data.loc[:,i] = data.loc[:,i].round()
        data.loc[:,i] = data.loc[:,i].astype('object')
    data.passenger_count = data.passenger_count.round()
    #missing_val = pd.DataFrame(data.isnull().sum())
    return data

def featureScaling(data, cnames_numeric):
    for i in cnames_numeric:
        print(i)
        data[i] = (data[i] - data[i].mean())/data[i].std()
    return data

#################################### User defined Functions End
############################################

#Set directory
os.chdir("C:/Users/vinayak\Desktop/Car Fare prediction")

#################################### Load dataset
###########################################################
train = pd.read_csv("train_cab.csv")
test = pd.read_csv("test.csv")

#################################### Data Cleaning and preparation
##########################################
train.info()
train['fare_amount'] = pd.to_numeric(train['fare_amount'], errors='coerce')
train.info()
test.info()

train = divideDateTime(train)
test = divideDateTime(test)

train = convertIntoProperDataTypes(train)
test = convertIntoProperDataTypes(test)

cnames_numeric = getCnamesNumeric(train)
cnames_factor = getCnamesFactor(train)

train.info()
```

```
test.info()

##################################### Missing Value Analysis
####################################################
bestImpute = getOptimalImputeMethod(train)

train = imputeMissingValues(train, cnames_numeric, cnames_numeric)

missing_val = pd.DataFrame(train.isnull().sum())
train = convertIntoProperDataTypes(train)
##################################### Analyse Data Distribution and Graphs
#####################################
train.describe()

#Analyze Distribution
number_of_columns=6
number_of_rows = len(cnames_numeric)-1/number_of_columns
plt.figure(figsize=(7*number_of_columns,5*number_of_rows))
for i in range(0,len(cnames_numeric)):
    plt.subplot(number_of_rows + 1,number_of_columns,i+1)
    sns.kdeplot(train[cnames_numeric[i]]).set_title("Distribution of "+cnames_numeric[i])

plt.figure(figsize=(7*number_of_columns,5*number_of_rows))
for i in range(0,len(cnames_numeric)):
    plt.subplot(number_of_rows + 1,number_of_columns,i+1)
    sns.kdeplot(np.log(train[cnames_numeric[i]].values)).set_title("Distribution of
"+cnames_numeric[i] + "Using log scale")

#Analyze Distribution of Latitude-Longitude points
city_lat_border = (40.50, 41.00)
city_long_border = (-74.15, -73.60)
train.plot(kind='scatter', x='dropoff_longitude', y='dropoff_latitude', color='green')
plt.title("Dropoff locations")
plt.ylim(city_lat_border)
plt.xlim(city_long_border)

train.plot(kind='scatter', x='pickup_longitude', y='pickup_latitude',color='blue')
plt.title("Pickups")
plt.ylim(city_lat_border)
plt.xlim(city_long_border)

#Relation Between Categorical variable and target
plt.figure(figsize=(8,6))
sns.barplot(data=train, x="year", y="fare_amount")
```

```
plt.figure(figsize=(12,6))
sns.barplot(data=train, x="month", y="fare_amount")

plt.figure(figsize=(15,6))
sns.barplot(data=train, x="day", y="fare_amount")

plt.figure(figsize=(8,6))
sns.barplot(data=train, x="dayOfWeek", y="fare_amount")

plt.figure(figsize=(8,6))
sns.barplot(data=train, x="hour", y="fare_amount")

################################## Outlier Analysis
######################################################
checkOutlier(train, cnames_numeric)

train = removeOutlier(train, cnames_numeric, cnames_factor)
train = convertIntoProperDataTypes(train)

################ Analyse Data Distribution and Graphs (After Outlier Analysis)
###########################
train.describe()

#Analyze Distribution
number_of_columns=6
number_of_rows = len(cnames_numeric)-1/number_of_columns
plt.figure(figsize=(7*number_of_columns,5*number_of_rows))
for i in range(0,len(cnames_numeric)):
    plt.subplot(number_of_rows + 1,number_of_columns,i+1)
    sns.kdeplot(train[cnames_numeric[i]]).set_title("Distribution of "+cnames_numeric[i])

#Analyze Distribution
plt.figure(figsize=(5*number_of_columns,8*number_of_rows))
for i in range(0,len(cnames_numeric)):
    plt.subplot(number_of_rows + 1,number_of_columns,i+1)
    sns.distplot(train[cnames_numeric[i]],kde=True)

#Analyze Distribution of Latitude-Longitude points
city_lat_border = (40.50, 41.00)
city_long_border = (-74.15, -73.60)
train.plot(kind='scatter', x='dropoff_longitude', y='dropoff_latitude', color='green')
plt.title("Dropoff locations")
plt.ylim(city_lat_border)
```

```
plt.xlim(city_long_border)

train.plot(kind='scatter', x='pickup_longitude', y='pickup_latitude',color='blue')
plt.title("Pickups")
plt.ylim(city_lat_border)
plt.xlim(city_long_border)

#Relation Between Categorical variable and target
plt.figure(figsize=(8,6))
sns.barplot(data=train, x="year", y="fare_amount")

plt.figure(figsize=(12,6))
sns.barplot(data=train, x="month", y="fare_amount")

plt.figure(figsize=(15,6))
sns.barplot(data=train, x="day", y="fare_amount")

plt.figure(figsize=(8,6))
sns.barplot(data=train, x="dayOfWeek", y="fare_amount")

plt.figure(figsize=(10,6))
sns.barplot(data=train, x="hour", y="fare_amount")

#Relation Between Categorical variable and number of passengers
plt.figure(figsize=(8,6))
sns.barplot(data=train, x="year", y="passenger_count")

plt.figure(figsize=(12,6))
sns.barplot(data=train, x="month", y="passenger_count")

plt.figure(figsize=(15,6))
sns.barplot(data=train, x="day", y="passenger_count")

plt.figure(figsize=(8,6))
sns.barplot(data=train, x="dayOfWeek", y="passenger_count")

plt.figure(figsize=(10,6))
sns.barplot(data=train, x="hour", y="passenger_count")
#################################### Feature Selection
################################################
df_corr = train.loc[:,cnames_numeric]
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(10, 10))
#Generate correlation matrix
```

```
corr = df_corr.corr()
#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap = 'viridis',
            square=True, ax=ax, annot=True)



for i in range(0,len(cnames_factor)):
    for j in range(i+1,len(cnames_factor)):
        print(cnames_factor[i], " VS ", cnames_factor[j])
        chi2, p, dof, ex = chi2_contingency(pd.crosstab(train[cnames_factor[i]],
train[cnames_factor[j]]))
        print(p)

#No variable is highly correlated to any other variable so don't remove any variable

################################### Feature Scaling
#################################################
cnames_numeric.remove("fare_amount")
train = featureScaling(train, cnames_numeric)
test = featureScaling(test, getCnamesNumeric(test))

################################### Model Development
#################################################
#Divide data into train and test
X = train.values[:, 1:]
Y = train.values[:,0]

X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.2)

#Linear Regression
regressor = LinearRegression()
regressor.fit(X_train, y_train)                    #Fit the model on training data
y_pred = regressor.predict(X_test)                 #predicting the test set results

lm_rmse=np.sqrt(mean_squared_error(y_pred, y_test))
print("Test RMSE for Linear Regression is ",lm_rmse)    #4.003082145405653

#Decision tree for regression
regressor = DecisionTreeRegressor(max_depth =6, random_state = 0)
regressor.fit(X_train,y_train)
y_pred = regressor.predict(X_test)
dt_rmse=np.sqrt(mean_squared_error(y_pred, y_test))
print("Test RMSE for Decision tree is ",dt_rmse)    #3.5597152258765834
```

```
#Random Forest
regressor = RandomForestRegressor(n_estimators = 100, random_state = 883,n_jobs=-1)
regressor.fit(X_train,y_train)
y_pred= regressor.predict(X_test)
rf_rmse=np.sqrt(mean_squared_error(y_pred, y_test))
print("RMSE for Random Forest is ",rf_rmse)      #2.5970888685703235

#Random Forest :::: RMSE = 2.5970888685703235 --> best among all
result = regressor.predict(test)
resultDataFrame =
pd.concat([pd.DataFrame({"fare_amount":result}).reset_index(drop=True), test], axis=1)
#Note: RMSE may vary as training and test data may vary.
```

Appendix C- R Code

```
#Clear the environment
rm(list=ls())

#set the working directory
setwd(dir = "C:/Users/vinayak/Desktop/Car Fare prediction")

#Load Libraries
x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50",
"dummies", "e1071", "Information",
      "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine',
'inTrees',"usdm","randomForest","e1071","plyr", "dplyr","caTools",
      "tidyverse", "geosphere", "lubridate", "fpc")

install.packages(x)
lapply(x, require, character.only = TRUE)
rm(x)

#################################### User defined Functions
#################################################
divideDateTime <- function(data) {
  data = data %>%
    mutate(pickup_datetime = ymd_hms(pickup_datetime)
           ,year = year(pickup_datetime)
           ,month = month(pickup_datetime)
           ,day = day(pickup_datetime)
           ,dayOfWeek = wday(pickup_datetime)
           ,hour = hour(pickup_datetime))
  data=data[, !(colnames(data) %in% c("pickup_datetime"))]
  return(data)
}

convertIntoProperDataTypes <- function(data) {
  if(ncol(data)==11) {
    cnumber_factor = c(7,8,9,10,11)
    data$fare_amount = as.numeric(as.character(data$fare_amount))

  }
  else {
    cnumber_factor = c(6,7,8,9,10)
  }
  data[,cnumber_factor] <- lapply(data[,cnumber_factor] , factor)
  return(data)
```

```
}

getCnamesFactor <- function(data) {
  cnumber_factor = sapply(data, is.factor)
  factor_data = data[, cnumber_factor]
  return(colnames(factor_data))
}

getCnamesNumeric <- function(data) {
  cnumber_numeric = sapply(data, is.numeric)
  numeric_data = data[,cnumber_numeric]
  return(colnames(numeric_data))
}

getOptimalImputeMethod <- function(data) {
  actual = data[6,2]
  data[6,2] = NA

  dataKnn = data
  dataMean = data
  dataMedian = data

  #Mean Method
  dataMean$pickup_longitude[is.na(dataMean$pickup_longitude)] =
mean(dataMean$pickup_longitude, na.rm = T)
  #Median Method
  dataMedian$pickup_longitude[is.na(dataMedian$pickup_longitude)] =
median(dataMedian$pickup_longitude, na.rm = T)
  # kNN Imputation
  dataKnn = knnImputation(dataKnn, k = 3)

  result = c(actual, dataMean[6,2], dataMedian[6,2], dataKnn[6,2])
  names(result) = c("Actual", "Mean", "Median", "KNN")
  print(result)
}

imputeMissingValues <- function(data, cnames_numeric) {
  for(i in cnames_numeric) {
    data[,i][data[,i] %in% 0] = NA
  }
  print(paste("Before", sum(is.na(data))))
  # kNN Imputation
  data = knnImputation(data, k = 3)
  print(paste("After",sum(is.na(data))))
```

```
  return(data)
}

removeOutlier <- function(data, cnames_numeric) {
  for(i in cnames_numeric) {
    val = data[,i][data[,i] %in% boxplot.stats(data[,i])$out]
    data[,i][data[,i] %in% val] = NA
  }
  sum(is.na(data))

  #Impute NA using KNN impute
  data = knnImputation(data, k = 3)
  return(data)
}

chiSquareTest <- function(data) {
  factor_data = train[, sapply(train, is.factor)]
  for (i in 1:length(cnames_factor)) {
    for(j in i+1:length(cnames_factor)) {
      if(j<=length(cnames_factor)) {
        print(paste(cnames_factor[i], " VS ", cnames_factor[j]))
        print(chisq.test(table(factor_data[,i],factor_data[,j])))
      }
    }
  }
}

featureScaling <- function(data, cnames_numeric) {
  for(i in cnames_numeric) {
    print(i)
    data[,i] = (data[,i] - mean(data[,i]))/sd(data[,i])
  }
  return(data)
}

################################### User defined Functions End
################################################

################################### Load dataset
################################################
train = read.csv("train_cab.csv")
test = read.csv("test.csv")
```

```
################################### Data Cleaning and preparation
##################################################
train <- divideDateTime(train)
test <- divideDateTime(test)

train <- convertIntoProperDataTypes(train)
test <- convertIntoProperDataTypes(test)

str(train)

cnames_factor = getCnamesFactor(train)
cnames_numeric = getCnamesNumeric(train)

################################### Missing Value Analysis
##################################################
getOptimalImputeMethod(train)

train = imputeMissingValues(train, cnames_numeric)

################################### Outlier Analysis
##################################################
for (i in 1:length(cnames_numeric)) {
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames_numeric[i]), x = "fare_amount"), data
= subset(train))+
         stat_boxplot(geom = "errorbar", width = 0.5) +
         geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                      outlier.size=1, notch=FALSE) +
         theme(legend.position="bottom")+
         labs(y=cnames_numeric[i],x="fare_amount")+
         ggtitle(paste("Box plot of fare_amount for",cnames_numeric[i])))
}

gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)
gridExtra::grid.arrange(gn4,gn5,ncol=3)

train = removeOutlier(train, cnames_numeric)

################################### Analyse Data Distribution and Graphs
############################################
ggplot(train, aes(x=pickup_longitude, y=pickup_latitude)) + geom_point()
ggplot(train, aes(x=dropoff_longitude, y=dropoff_latitude)) + geom_point()

cnames_numeric = cnames_numeric[-1]
#Analyze Distribution
```

```
for(i in 1:length(cnames_numeric)) {
  assign(paste0("gn",i), ggplot(data = train, aes_string(x = cnames_numeric[i])) +
geom_histogram(bins = 25, fill="green", col="black")+ ggtitle(paste("Histogram
of",cnames_numeric[i])))
}

gridExtra::grid.arrange(gn1,gn2,gn3,gn4,ncol=2)

ggplot(data = train, aes(x=year, y= fare_amount)) + geom_bar(stat = 'identity')
ggplot(data = train, aes(x=month, y= fare_amount)) + geom_bar(stat = 'identity')
ggplot(data = train, aes(x=day, y= fare_amount)) + geom_bar(stat = 'identity')
ggplot(data = train, aes(x=dayOfWeek, y= fare_amount)) + geom_bar(stat = 'identity')
ggplot(data = train, aes(x=hour, y= fare_amount)) + geom_bar(stat = 'identity')
ggplot(data = train, aes(x=hour, y= passenger_count)) + geom_bar(stat = 'identity')
##################################### Feature Selection
####################################################
corrgram(train[getCnamesNumeric(train)], order = F,
        upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

#Chi-square test for correlation between categorical variable
chiSquareTest(data)

##################################### Feature Scaling
####################################################
qqnorm(train$pickup_longitude)
hist(train$dropoff_latitude)

train = featureScaling(train, cnames_numeric)
test = featureScaling(test, getCnamesNumeric(test))
##################################### Model Development
####################################################
#Clean the environment
rmExcept(c("train","test"))

#Split training data into training and test set
set.seed(123)
split = sample.split(train$fare_amount, SplitRatio = 0.8)
training_set = subset(train, split == TRUE)
test_set = subset(train, split == FALSE)

#Apply linear regression
regressor = lm(formula = fare_amount ~ ., data = training_set)
y_pred = predict(regressor, newdata = test_set[,-1])
sqrt(mean((test_set$fare_amount - y_pred)^2)/nrow(test_set))  #RMSE = 0.07478457
```

```
#Apply Decision tree
regressor = rpart(fare_amount ~ ., data = training_set, method = "anova")
y_pred = predict(regressor, newdata = test_set[,-1])
sqrt(mean((test_set$fare_amount - y_pred)^2)/nrow(test_set))  #RMSE = 0.0678244

#Apply Random Forest
set.seed(1234)
regressor = randomForest(x = training_set[,-1], y = training_set$fare_amount, ntree = 500)
y_pred = predict(regressor, newdata = test_set[,-1])
sqrt(mean((test_set$fare_amount - y_pred)^2)/nrow(test_set))  #RMSE = 0.05815893

#Random Forest :::: RMSE = 0.05815893 --> best among all
#applying test data
#testResult = predict(regressor, newdata = test)

#Note: RMSE may vary as training and test data may vary.
```

Instructions to run code file:

1. Open Anaconda-Spyder (For Python) or R-Studio (For R)
2. Open my code
3. Execute the code

Reference:

1. https://www.wikipedia.org/
2. https://edwisor.com/home