**Prediction of Car Fare**

*Sahayaraj Daniel*

May 22 2019

**Contents**

[**Chapter 1** 3](#_Toc9417247)

[**Introduction** 3](#_Toc9417248)

[**1.1** **Problem Statement** 3](#_Toc9417249)

[**1.2** **Data** 3](#_Toc9417250)

[**1.2.1 Exploratory Data Analysis:** 5](#_Toc9417251)

[**Chapter 2** 11](#_Toc9417252)

[**Methodology** 11](#_Toc9417253)

[**2.1 Pre-Processing** 11](#_Toc9417254)

[**2.1.1 Outlier Analysis** 12](#_Toc9417255)

[**2.1.2 Missing Value Analysis** 15](#_Toc9417256)

[**2.1.3 Feature Selection** 16](#_Toc9417257)

[**2.2 Modelling** 17](#_Toc9417258)

[**2.2.1 Model Selection** 17](#_Toc9417259)

[**2.2.2 Decision Tree** 17](#_Toc9417260)

[**2.2.3 Random Forest** 18](#_Toc9417261)

[**2.2.4 Linear Regression** 19](#_Toc9417262)

[**Chapter 3** 21](#_Toc9417263)

[**Conclusion** 21](#_Toc9417264)

[**3.1 Model Evaluation** 21](#_Toc9417265)

[**3.2 Root Mean Squared Error:** 21](#_Toc9417266)

[**3.3** **Model Selection** 22](#_Toc9417267)

[**Appendix A – R Code** 23](#_Toc9417268)

[**Appendix B – Python Code** 30](#_Toc9417269)

# **Chapter 1**

# **Introduction**

## **Problem Statement**

The objective of this project is to predict the cab fare amount based on the history data details collected year-wise. In general, there are lot of factors that decide the cab fare such as the pick up location, number of kilometres travelled, number of passengers who take the ride, peak time of the day, whether the ride is taken on a weekday or weekend and so on. This fare prediction is more helpful to the organization as to check the favourable criteria required for a pickup and to accurately charge for the ride, which in turn earns the customer satisfaction as customers can also rely on the fare amount shown for each ride.

## **Data**

Our task is to build regression models which will predict the cab fare amount based on the history data collected. Given below is a sample of the data set that we are using to predict the cab fare:

The details of data attributes in the dataset are as follows -

* **pickup\_datetime** - timestamp value indicating when the cab ride started.
* **pickup\_longitude** - float for longitude coordinate of where the cab ride started.
* **pickup\_latitude** - float for latitude coordinate of where the cab ride started.
* **dropoff\_longitude** - float for longitude coordinate of where the cab ride ended.
* **dropoff\_latitude** - float for latitude coordinate of where the cab ride ended.
* **passenger\_count** - an integer indicating the number of passengers in the cab ride.
* **fare\_amount –** an integer value representing the cab fare amount collected for the ride.

Here the dependant variable is the fare\_amount, which is determined by other independent variables like pickup\_datetime, latitude and longitude values and passenger count.

**Some sample data:**

***Table1.1 Sample data (Raw Data)***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| fare\_amount | pickup\_datetime | pickup\_longitude | pickup\_latitude | dropoff\_longitude | dropoff\_latitude | passenger\_count |
| 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.84161 | 40.712278 | 1 |
| 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1 |
| 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.76127 | -73.991242 | 40.750562 | 2 |
| 7.7 | 2012-04-21 04:30:42 UTC | -73.98713 | 40.733143 | -73.991567 | 40.758092 | 1 |
| 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1 |
| 12.1 | 2011-01-06 09:50:45 UTC | -74.000964 | 40.73163 | -73.972892 | 40.758233 | 1 |
| 7.5 | 2012-11-20 20:35:00 UTC | -73.980002 | 40.751662 | -73.973802 | 40.764842 | 1 |
| 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.84161 | 40.712278 | 1 |

The original dataset contains features as pickup and dropoff locations, as longitude and latitude coordinates, ride fare, and passenger count; which is not enough to predict the cab fare. So, we need to extract the other required fields from the available data. For example, we can extract, day, month, year, hour details from datetime field and calculate the actual distance from pickup and drop location points.

cab\_data\_train$pickup\_datetime = strptime(x=as.character(cab\_data\_train$pickup\_datetime), format = "%Y-%m-%d %H:%M:%S", tz = "UTC")

**#extrapolating hours, months, years values from date time field**

cab\_data\_train$pickup\_year <- cab\_data\_train$pickup\_datetime$year+1900

cab\_data\_train$pickup\_month <- cab\_data\_train$pickup\_datetime$mon

cab\_data\_train$pickup\_weekday <- cab\_data\_train$pickup\_datetime$wday

cab\_data\_train$pickup\_hour <- cab\_data\_train$pickup\_datetime$hour

**#Calculating Distance from Latitudes and Longitudes**

cab\_data\_train$Distance\_Haversine = distHaversine(cbind(cab\_data\_train$pickup\_longitude, cab\_data\_train$pickup\_latitude),cbind(cab\_data\_train$dropoff\_longitude, cab\_data\_train$dropoff\_latitude))

The data was processed to extract separate features for year, month, day, weekday, hour, as well as trip distance as the difference between dropoff and pickup latitudes and longitudes and shown below.

Processed data looks like:

***Table 1.2 Processed Data (Column 1 to 6)***

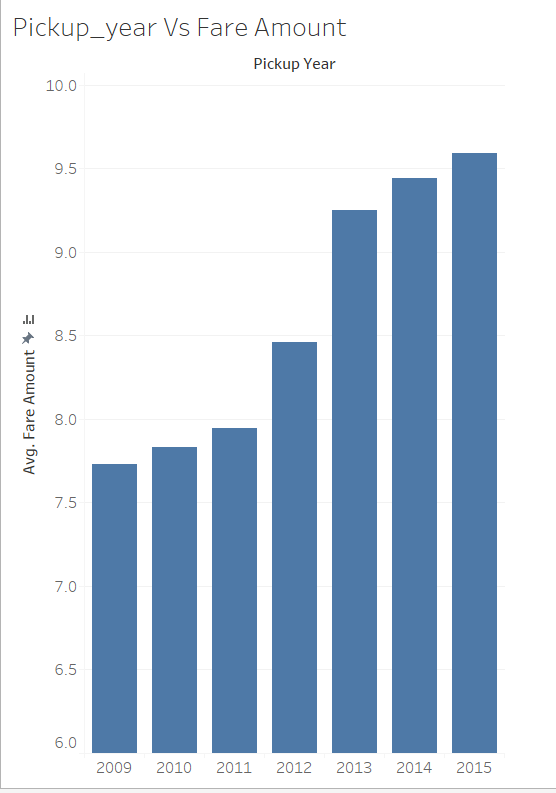
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| fare\_amount | pickup\_datetime | pickup\_longitude | pickup\_latitude | dropoff\_longitude | dropoff\_latitude |
| 4.5 | 15-06-2009 17:26 | -73.8443 | 40.72132 | -73.8416 | 40.71228 |
| 5.7 | 18-08-2011 00:35 | -73.9827 | 40.76127 | -73.9912 | 40.75056 |
| 7.7 | 21-04-2012 04:30 | -73.9871 | 40.73314 | -73.9916 | 40.75809 |
| 5.3 | 09-03-2010 07:51 | -73.9681 | 40.76801 | -73.9567 | 40.78376 |
| 12.1 | 06-01-2011 09:50 | -74.001 | 40.73163 | -73.9729 | 40.75823 |
| 7.5 | 20-11-2012 20:35 | -73.98 | 40.75166 | -73.9738 | 40.76484 |
| 16.5 | 04-01-2012 17:22 | -73.9513 | 40.77414 | -73.9901 | 40.75105 |
| 8.9 | 02-09-2009 01:11 | -73.9807 | 40.73387 | -73.9915 | 40.75814 |

***Table 1.3 Processed Data (Column 7 to 13)***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| passenger\_count | pickup\_year | pickup\_month | pickup\_weekday | pickup\_hour | Distance\_Haversine | Distance\_Meeus |
| 1 | 2009 | 5 | 1 | 17 | 1031.919 | 1029.61 |
| 2 | 2011 | 7 | 4 | 0 | 1391.082 | 1389.139 |
| 1 | 2012 | 3 | 6 | 4 | 2802.406 | 2795.815 |
| 1 | 2010 | 2 | 2 | 7 | 2001.396 | 1998.349 |
| 1 | 2011 | 0 | 4 | 9 | 3791.482 | 3787.994 |
| 1 | 2012 | 10 | 2 | 20 | 1557.55 | 1554.458 |
| 1 | 2012 | 0 | 3 | 17 | 4160.099 | 4159.936 |
| 2 | 2009 | 8 | 3 | 1 | 2852.819 | 2847.05 |

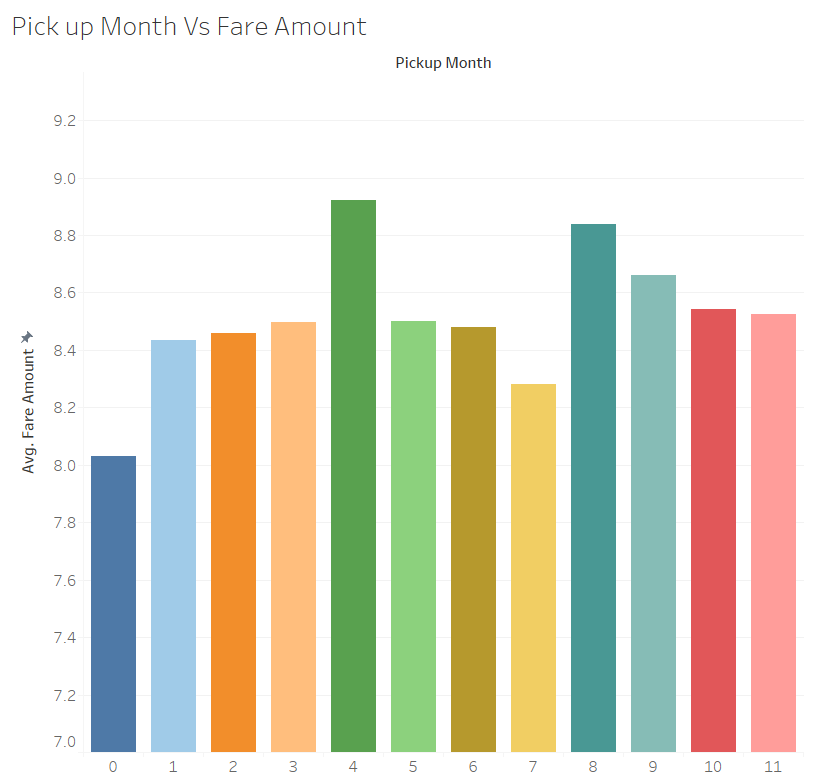
### **1.2.1 Exploratory Data Analysis:**

On Analysing the cleaned data, we can observe the following relation between various independent variables and fare\_amount. Different comparison and its inferences are given below.

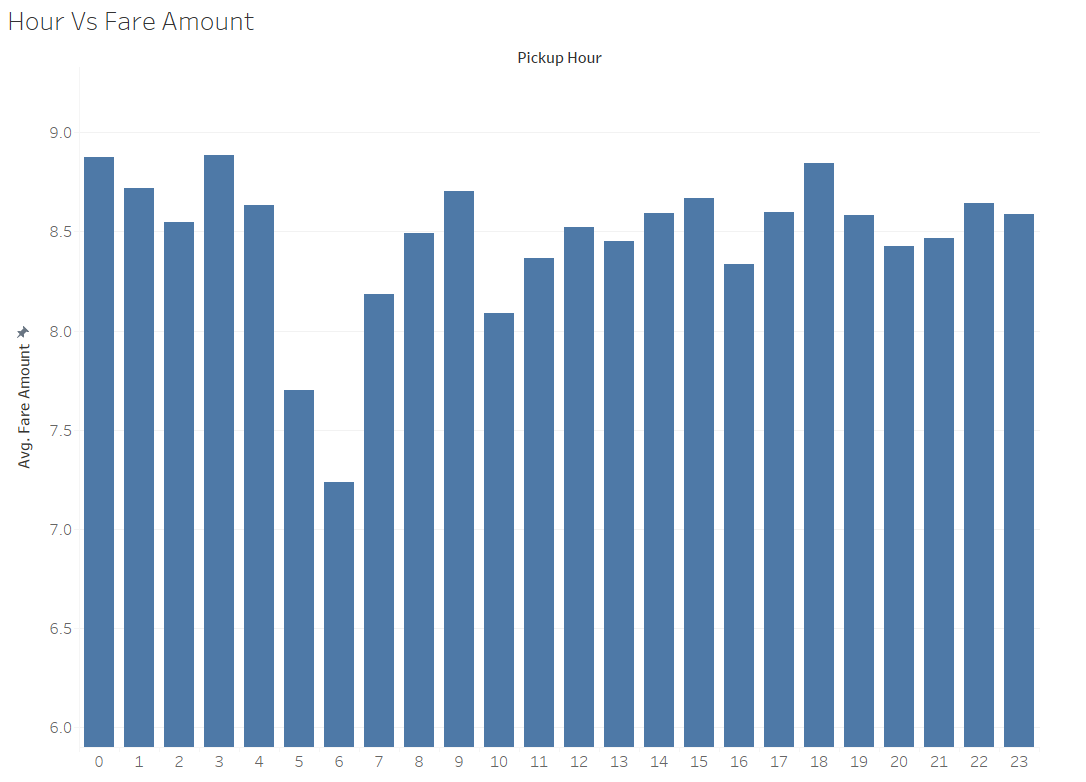


*Fig 1.1 Pick up Year Vs Fare amount*

It can be observed that the fare amount has been increasing steadily over the years and while comparing month wise data, fare is pretty less during January and July months due to holiday seasons.

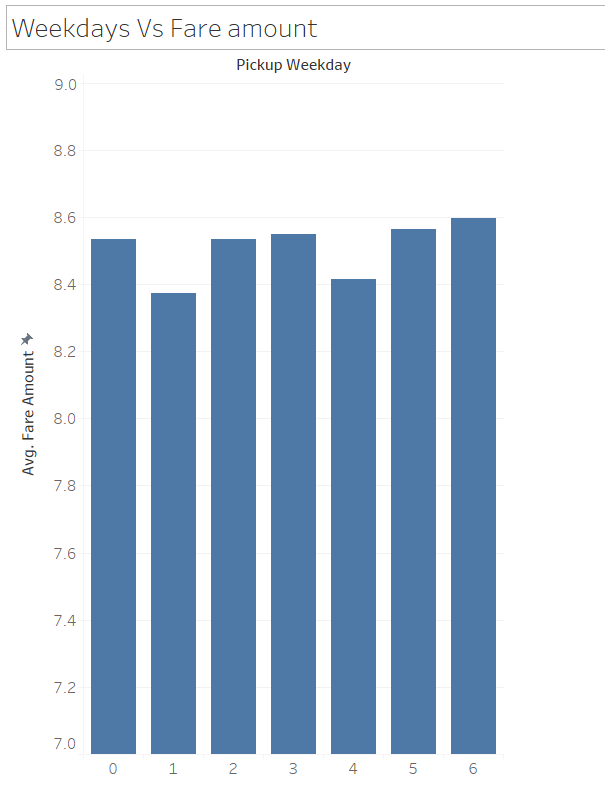


*Fig 1.2 Month Vs Fare amount*

**

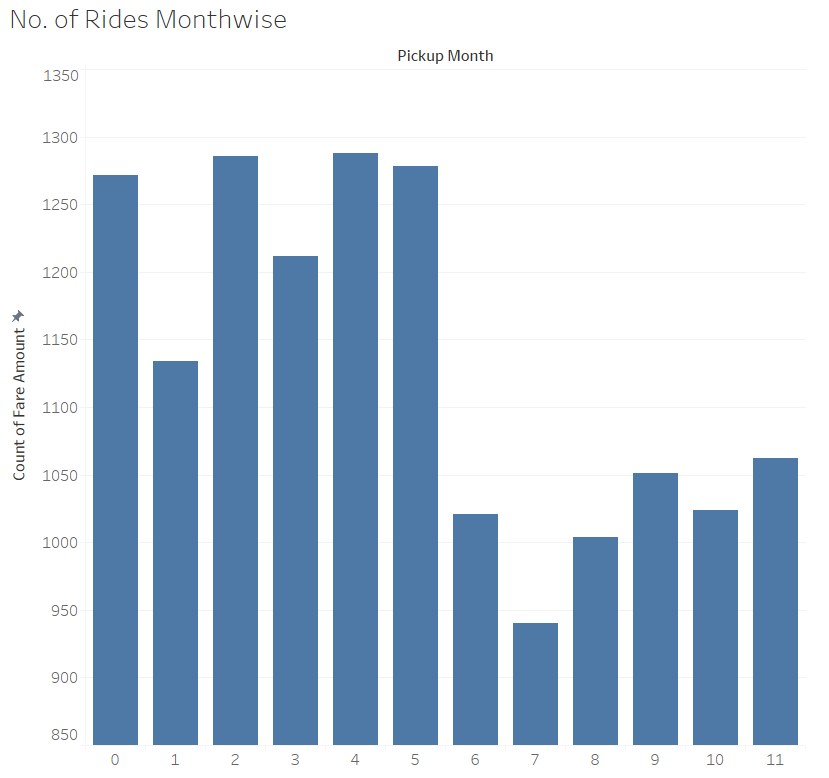
*Fig 1.3 Pick Up Hour Vs Fare amount*

This visualisation shows us that the fare amount is pretty less around early morning times and more around midnight and evening times which are the peak hours and pick ups to airport in general. That explains the reason for the higher fare amount in such times.

**

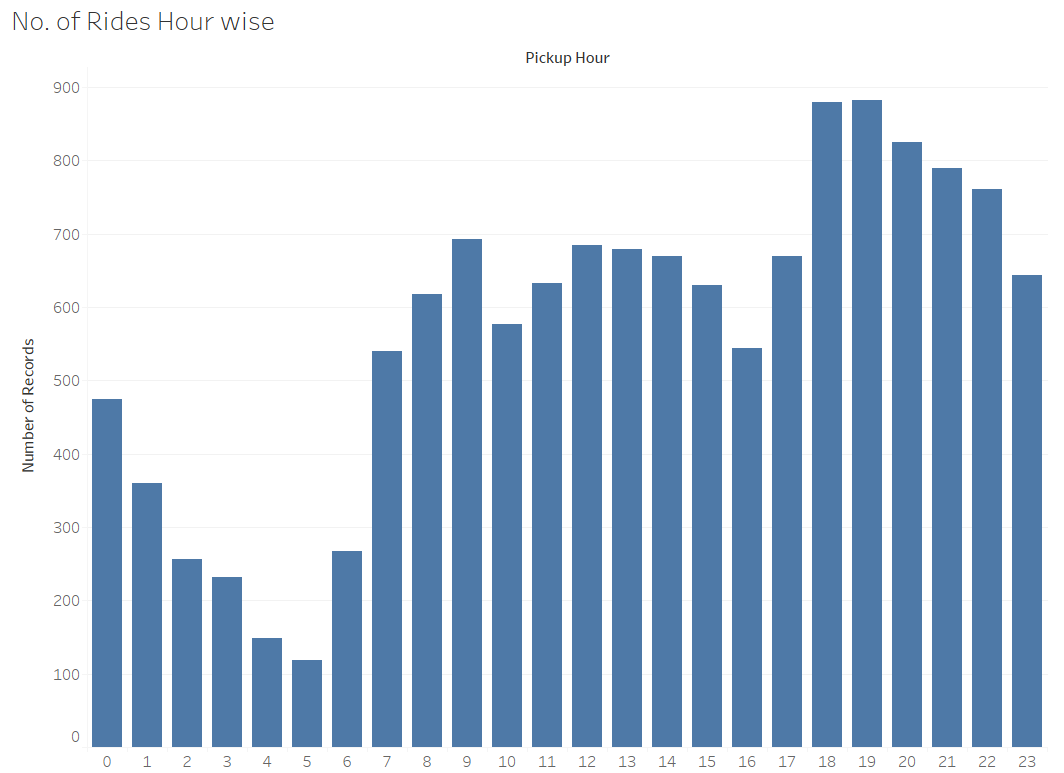
*Fig 1.4 Weekdays vs fare amount*

Cab fare is generally high during weekends, I.e., from Friday to Sunday and relatively lesser during the mid-weekdays, which are generally office hours.

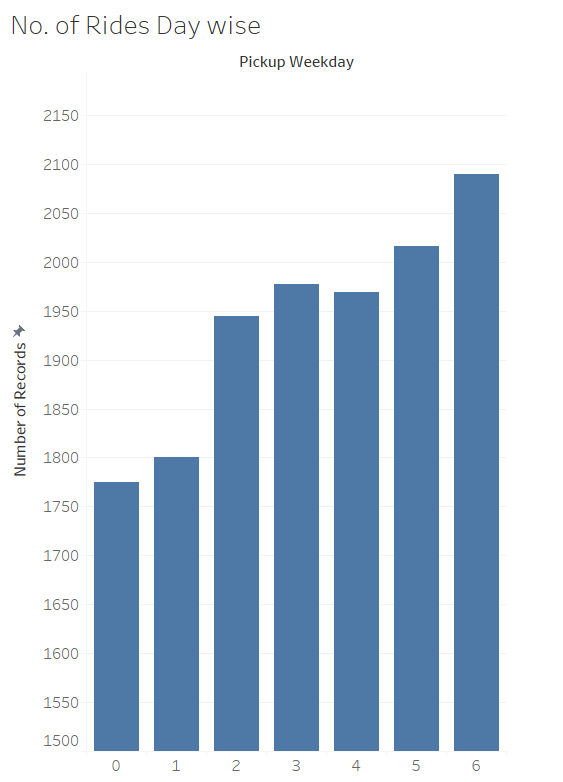
**

*Fig 1.5 Number of rides Month-wise*

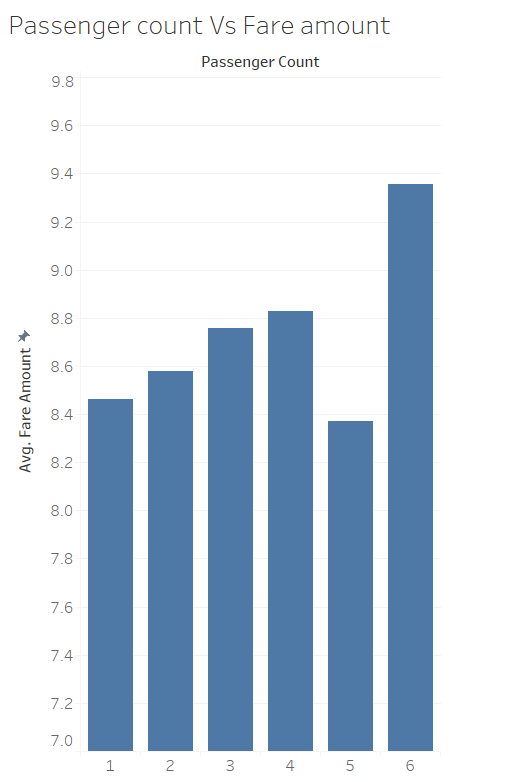
A larger number of rides are seen during the first half of the year whereas during the winter season of the year, the number of rides is quite less. On the other hand, the number of rides is very high during the evening times, i.e., after 17:00 hours and the number of rides is very less in the midnight. Also the number of rides is high during weekends, from Friday to Sunday.

**

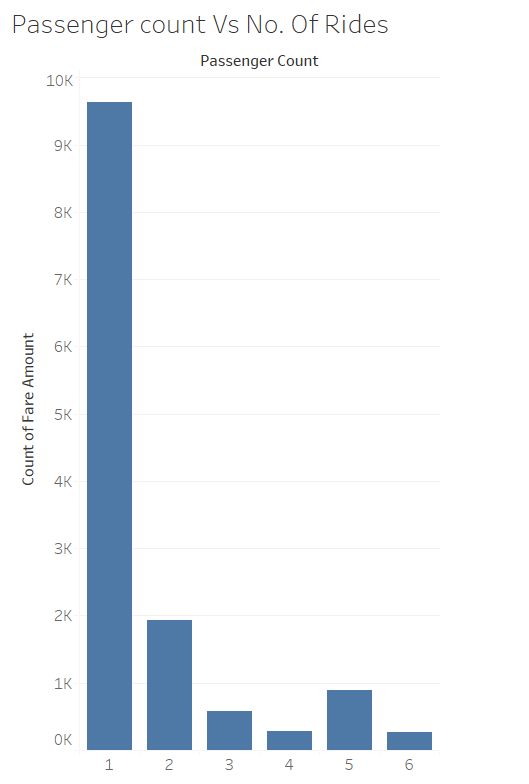
*Fig 1.6 Number of rides Hour-wise*

**

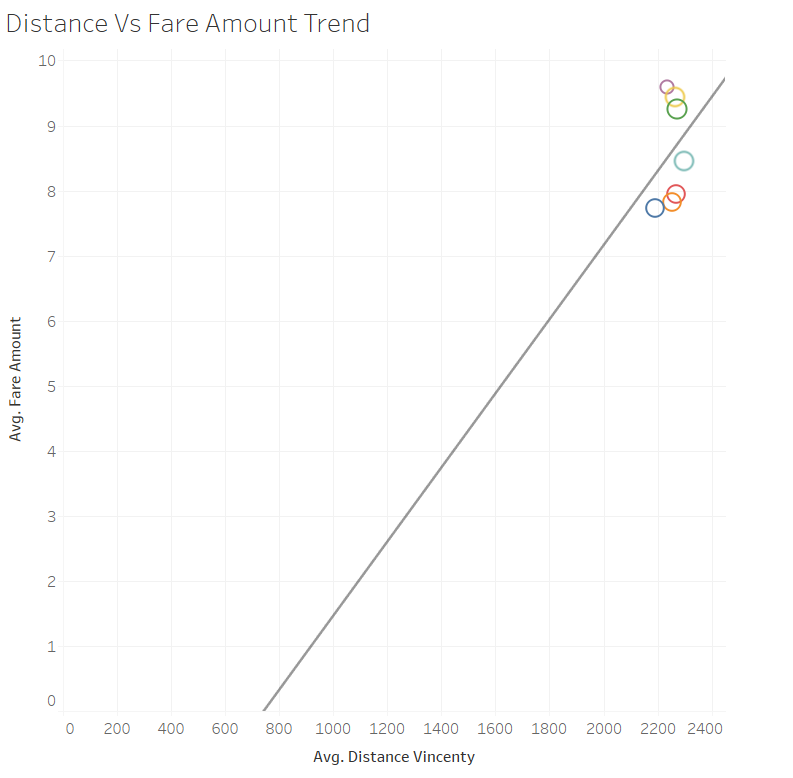
*Fig 1.7 Day-wise count of rides*

**

*Fig 1.8 Passenger count and Fare amount variation*

**

*Fig 1.9 Number of passengers and no. of rides trend*

**

*Fig 1.10 Distance Vs Fareamount*

This can easily be interpreted that as the distance increases, the fare amount increased, which is an obvious prediction. Hence this has to be the major predictor in Cab fare detection.

Therefore, the predictor variables we will be using for our project are:

***Table 1.4 Predictor variable list***

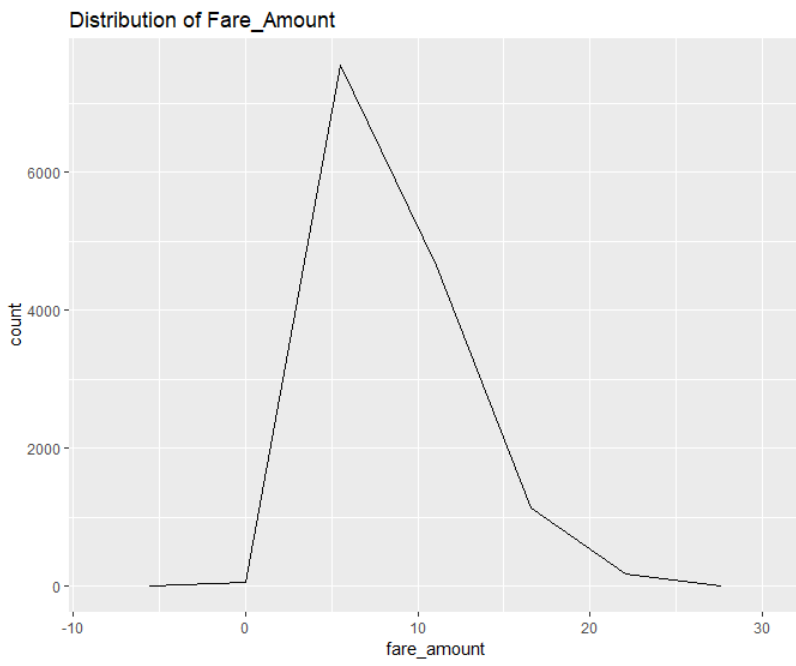
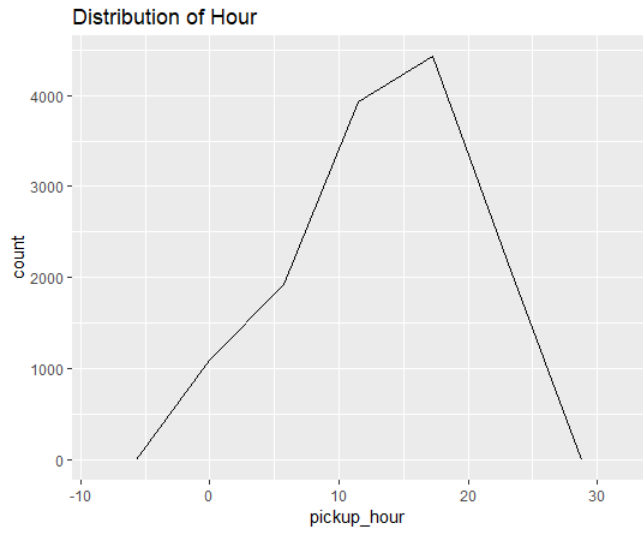
|  |  |
| --- | --- |
| S.No | Predictors |
| 1 | pickup\_latitude |
| 2 | pickup\_longitude |
| 3 | passenger\_count |
| 4 | Distance variables |
| 5 | pickup\_weekday |
| 6 | pickup\_hour |
| 7 | pickup\_month |

# **Chapter 2**

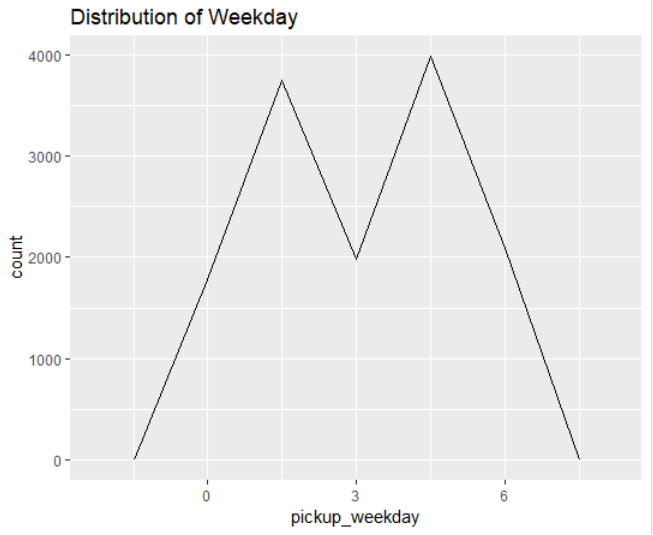
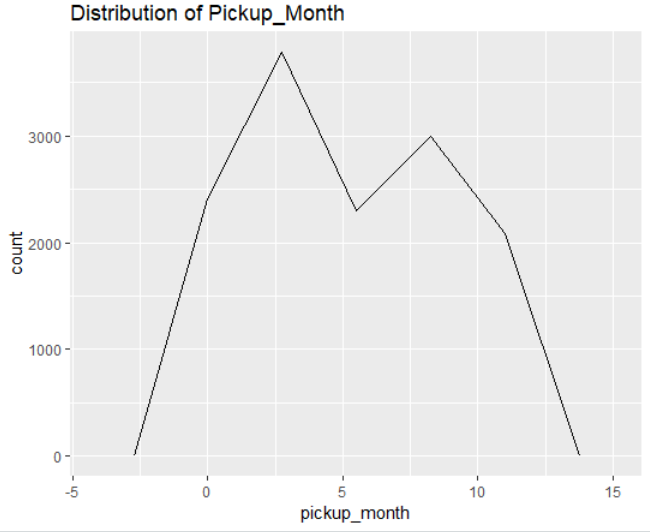
# **Methodology**

## **2.1 Pre-Processing**

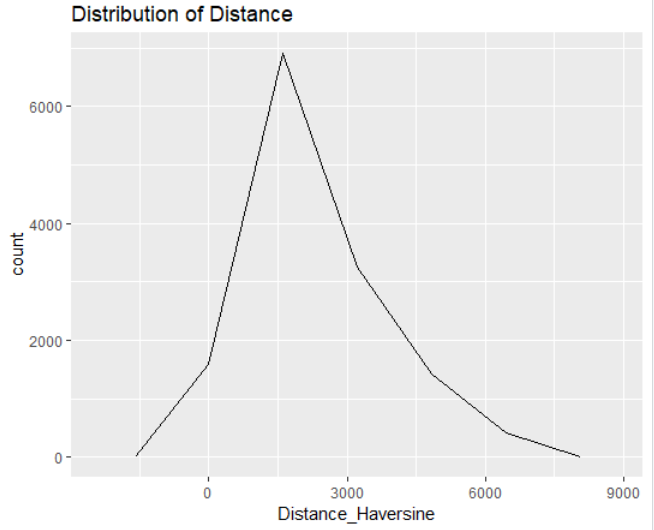
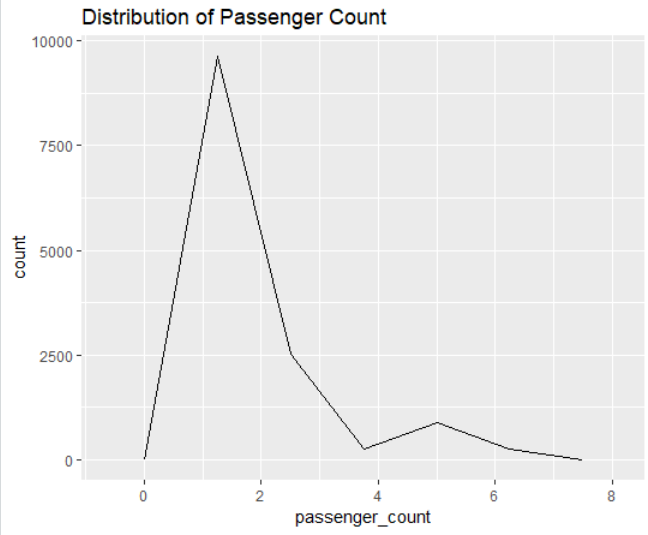
Any predictive modelling requires that we look at the data before we start modelling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. To start this process we will ﬁrst try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions of the predictor variables. Also, the distribution of factor variables are shown in box plot.

(a) (b)

(c) (d)

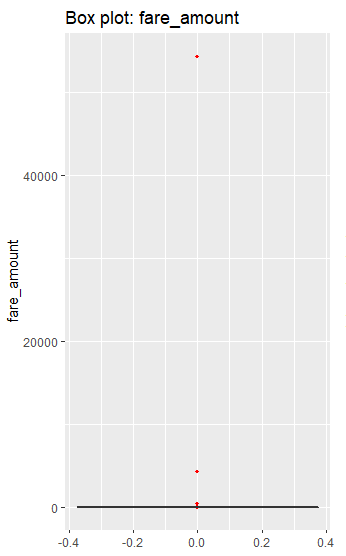
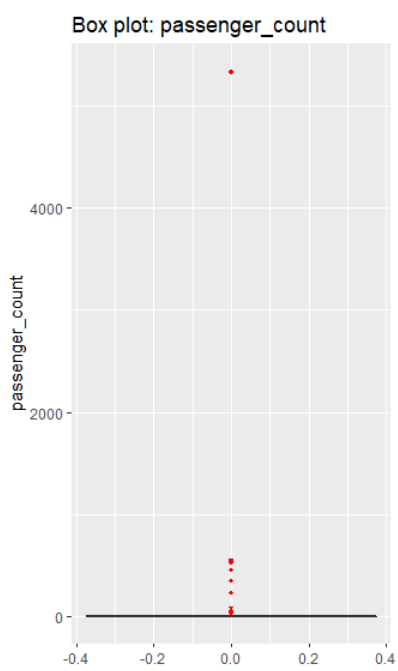
(e) (f)

*Fig 2.1 Density Function of continuous predictor variables*

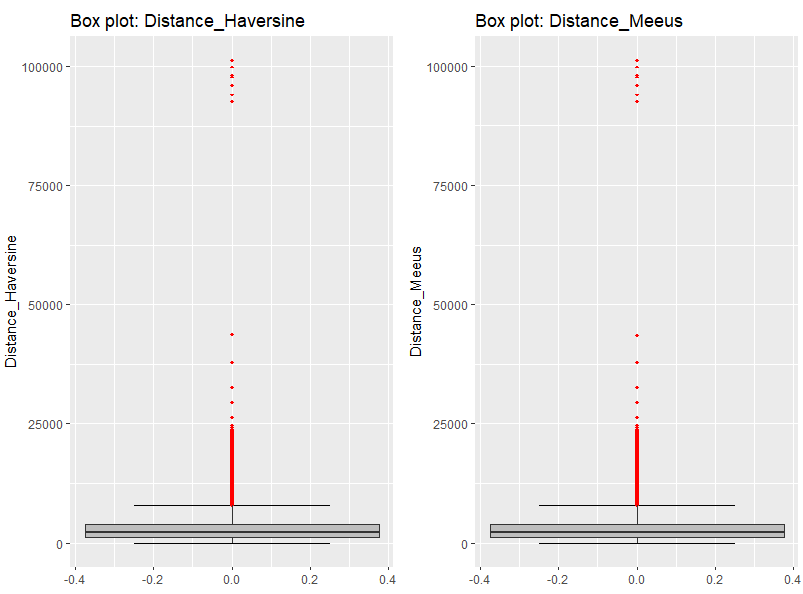
### **2.1.1 Outlier Analysis**

An outlier is an element of a data set that distinctly stands out from the rest of the data. In other words, outliers are those data points that lie outside the overall pattern of distribution as shown in figure below. The easiest way to detect outliers is to create a graph. Plots such as Box plots, Scatterplots and Histograms can help to detect outliers. In our case, we have used Box plots for various predictor variables. Since we have environmental parameters, we have plotted the box plot in such a way that how these parameters differ in given a season.

Here Fare\_amount, Passenger count and Distance have some serious outliers, which will shift our analysis to a considerable extend. We need to filter this values out before we proceed to our other exploratory analysis. For passenger count, from the test data given, we can conclude that the maximum number of passengers is 6 and the minimum count is 1. So, values that are greater than this limit can be removed as outliers . For Fare Amount and distance variables, we should calculate the outlier values and remove them through usual procedure.

1. (b)



(c)

*Fig 2.2 Outlier analysis – Box Plots showing outliers*

Outlier removal:

**#removing negative values of fare amount**

nrow(cab\_data\_train) ##15687

#row.names(cab\_data\_train) <- NULL

**#removing passengers greater than 6 and less than 1**

cab\_data\_train <- cab\_data\_train[which((cab\_data\_train$passenger\_count <= max\_passenger\_cnt) & (cab\_data\_train$passenger\_count >= min\_passenger\_cnt)), ]

nrow(cab\_data\_train) #15559 after removing outliers in passenger count

**#removing outliers in fare amount**

fare\_outlier = cab\_data\_train$fare\_amount[cab\_data\_train$fare\_amount %in% boxplot.stats(cab\_data\_train$fare\_amount)$out]

cab\_data\_train = cab\_data\_train[which(!cab\_data\_train$fare\_amount %in% fare\_outlier),]

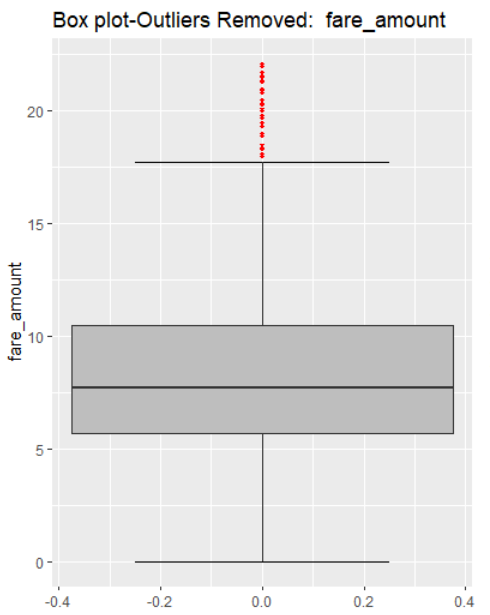
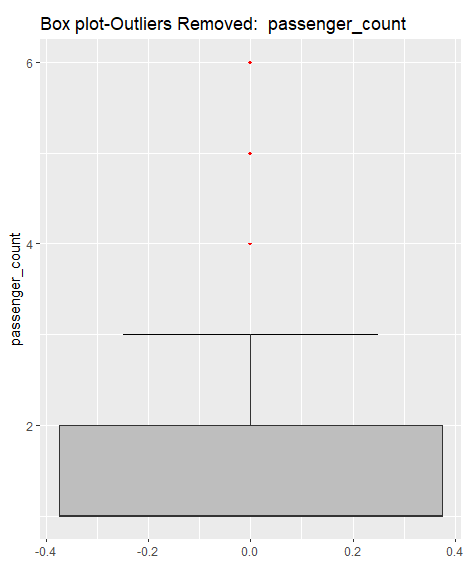
nrow(cab\_data\_train) #14210 after removing outliers

**#removing outliers in Distance**

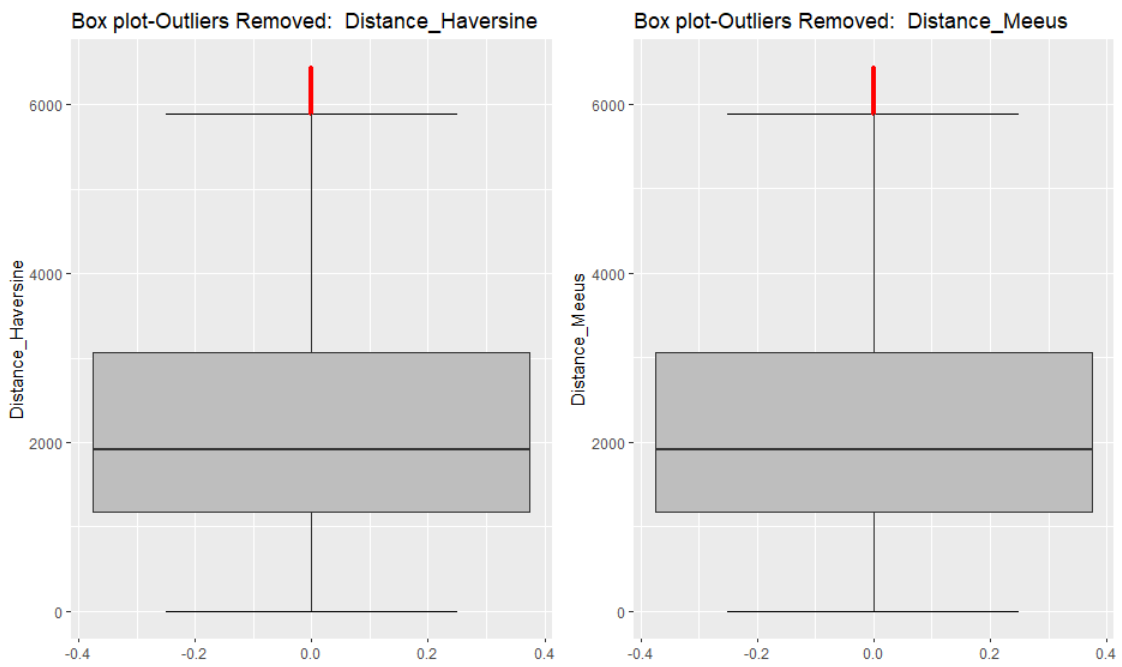
dist\_outlier = cab\_data\_train$Distance\_Haversine[cab\_data\_train$Distance\_Haversine %in% boxplot.stats(cab\_data\_train$Distance\_Haversine)$out]

cab\_data\_train = cab\_data\_train[which(!cab\_data\_train$Distance\_Haversine %in% dist\_outlier),]

nrow(cab\_data\_train) #13572 after removing outliers

1. (b)



(c)

*Fig 2.3 Outlier analysis – Box Plots after removing Outliers*

### **2.1.2 Missing Value Analysis**

Missing value analysis helps address several concerns caused by incomplete data. If cases with missing values are systematically different from cases without missing values, the results can be misleading. Also, missing data may reduce the precision of calculated statistics because there is less information than originally planned. We checked for missing values in all the variables of the data set and there is missing value found for fare amount and passenger count. However the percentage of missing values is very less (<.5%), hence they are removed.

missing\_val = data.frame(apply(cab\_data\_train,2,function(x){sum(is.na(x))}))

missing\_val$Columns = row.names(missing\_val)

names(missing\_val)[1] = "Missing Value Count"

missing\_val$Missing\_percentage = (missing\_val$`Missing Value Count`/nrow(cab\_data\_train)) \* 100

missing\_val = missing\_val[order(-missing\_val$Missing\_percentage),]

> missing\_val

Columns Missing Value Count Missing\_percentage

1 passenger\_count 55 0.342316549

2 fare\_amount 25 0.155598432

3 pickup\_datetime 1 0.006223937

4 pickup\_year 1 0.006223937

5 pickup\_month 1 0.006223937

6 pickup\_weekday 1 0.006223937

7 pickup\_hour 1 0.006223937

8 pickup\_longitude 0 0.000000000

9 pickup\_latitude 0 0.000000000

10 dropoff\_longitude 0 0.000000000

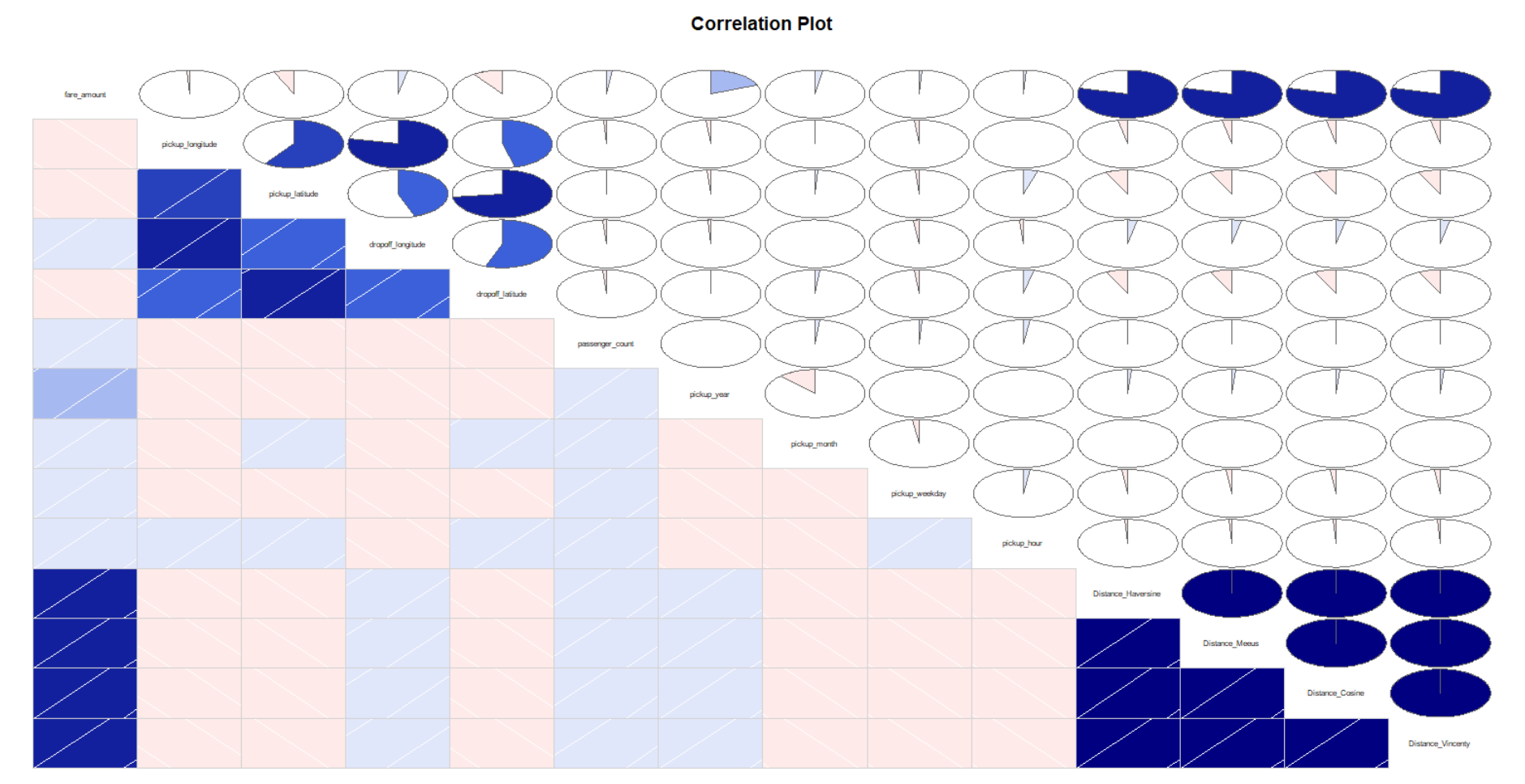
11 dropoff\_latitude 0 0.000000000

### **2.1.3 Feature Selection**

Before performing any type of modelling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. There are several methods of doing that and we have used a very simple method of finding the correlation between predictor variables through a correlation plot.

##Feature Selection - Correlation Plot

corrgram(cab\_data\_train[,numeric\_index], order = F, upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

**

*Fig 2.4 Correlation Plot of all the variables*

In this plot, we can see that Latitudes and Longitudes are correlated to each other as they are subjected to a single nearby location spots, which can be ignored. Also, the various distance we have calculated are also highly correlated to each other, so we can consider only one distance field for our calculations. We have chosen Haversine Distance Method for our distance calculation.

Also, we are selecting variables like Pick up Latitudes, Pick up Longitudes, Distance, Pick up Month, Pick up Hour and Passenger counts, as these variables play a vital role in determining the Fare amount. In addition, their p-values are also very much less than 0.05.

cab\_data\_train <- cab\_data\_train[, c("fare\_amount", "pickup\_longitude", "pickup\_latitude", "passenger\_count", "pickup\_hour", "pickup\_weekday", "Distance\_Haversine")]

**Coefficients:**

Estimate Std. Error t value Pr(>|t|)

(Intercept) 5.854e+02 1.157e+02 5.059 4.30e-07 \*\*\*

pickup\_longitude 5.051e+00 1.078e+00 4.683 2.86e-06 \*\*\*

pickup\_latitude -5.108e+00 1.142e+00 -4.471 7.86e-06 \*\*\*

passenger\_count 4.675e-02 1.803e-02 2.593 0.00953 \*\*

pickup\_hour 7.853e-03 3.624e-03 2.167 0.03028 \*

pickup\_weekday 3.545e-02 1.170e-02 3.030 0.00246 \*\*

Distance\_Haversine 2.044e-03 1.638e-05 124.810 < 2e-16 \*\*\*

## **2.2 Modelling**

### **2.2.1 Model Selection**

Selecting a model for data set purely depends on the nature of the dependant variable or the variable that must be predicted. The dependent variable can fall in either of the four categories:

1. Nominal
2. Ordinal
3. Interval
4. Ratio

If the dependent variable(in our case Fare amount) is Nominal the only predictive analysis that we can perform is Classiﬁcation, and if the dependent variable is Interval or Ratio the normal method is to do a Regression analysis, or classiﬁcation after binning; and if the dependent variable is Ordinal, then both classiﬁcation and regression can be done.

In our case, the variable to be predicted being continuous, we are going for Regression techniques. You always start your model building from the simplest to more complex. Here we are going to use three regression techniques, namely Decision tree Algorithm, Random Forest and Linear Regression techniques.

### **2.2.2 Decision Tree**

Decision tree is a type of supervised learning algorithm that can be used in both regression and classification problems. It works for both categorical and continuous input and output variables. Since our data is a continuous data, we have chosen this algorithm.

We are splitting the data into test and train data, where training data being 70% of the total data.

**#Divide the data into train and test**

train\_index = sample(1:nrow(cab\_data\_train), 0.70 \* nrow(cab\_data\_train))

train = cab\_data\_train[train\_index,]

test = cab\_data\_train[-train\_index,]

We are using anova method to predict the fare amount variable and the decision tree used for regression is shown below. A clearer Decision tree is attached as pdf for reference.

##Decision Tree Regression algorithm - anova

regression\_result = rpart(fare\_amount ~ ., data = train, method = "anova", control = rpart.control(cp = 0.00085, minsplit = 4, minbucket = 5, maxdepth = 10))

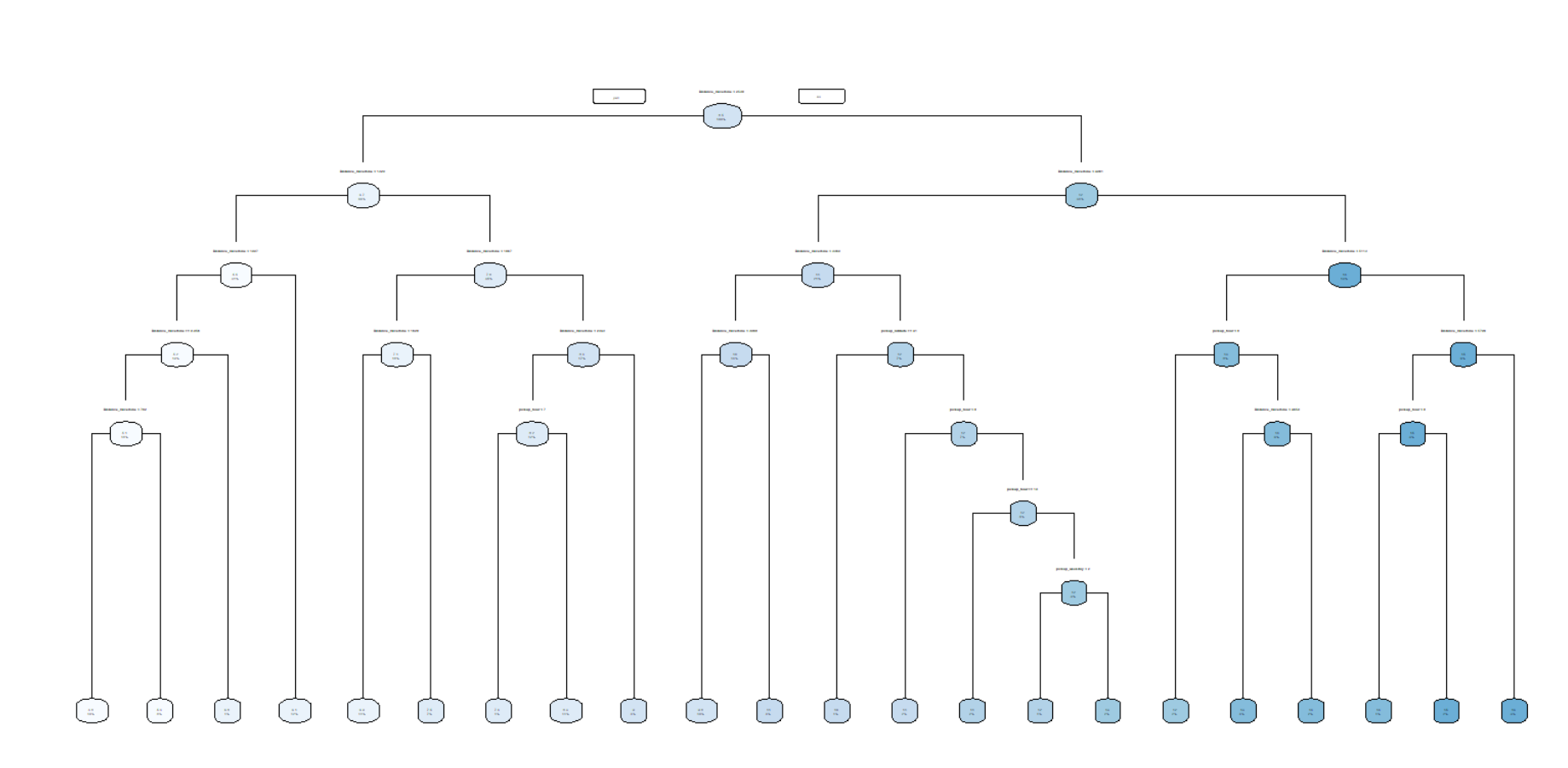
#prediction of test values

predict\_DT = predict(regression\_result, test[,-1])

rpart.plot(regression\_result, type = 1)

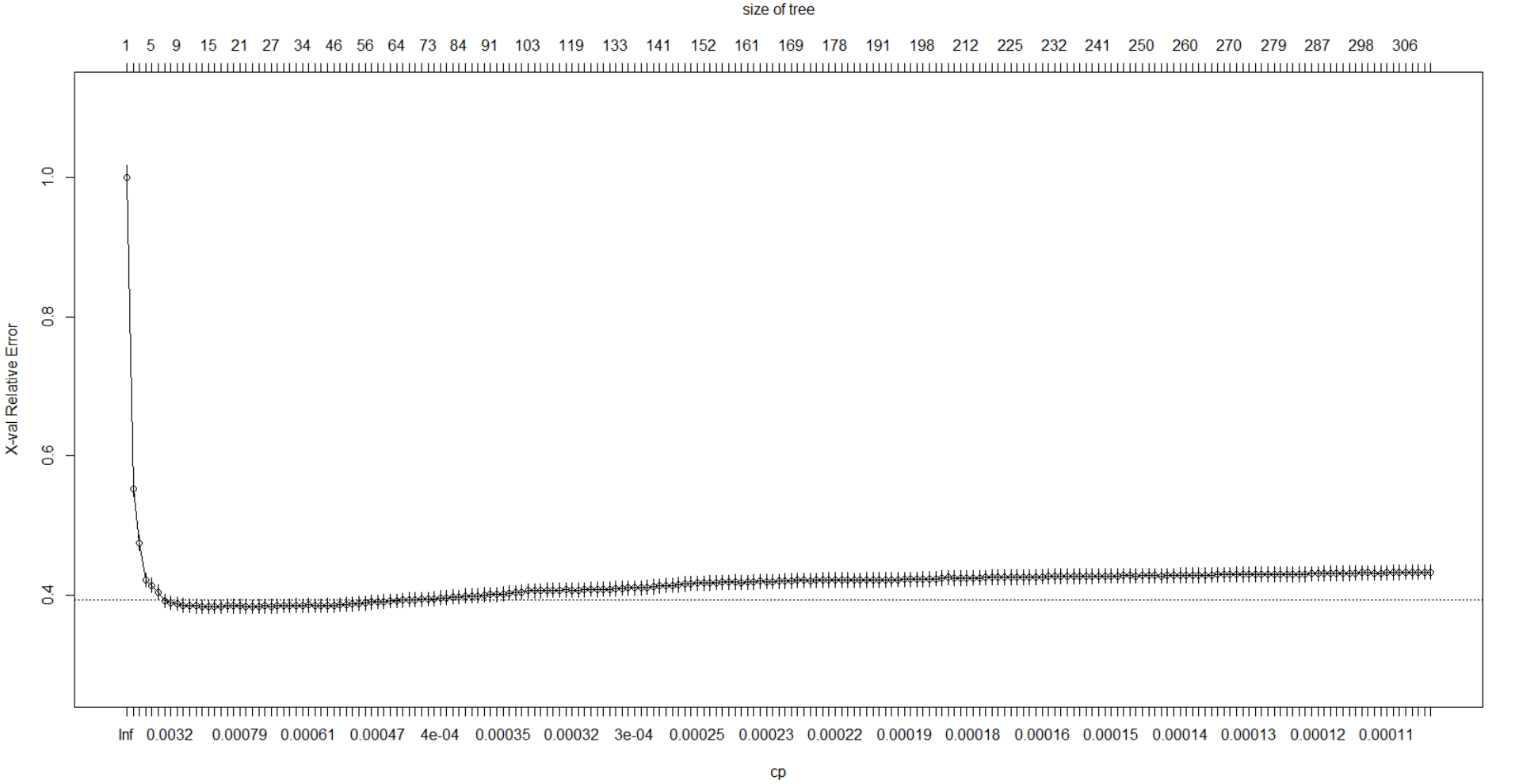
printcp(regression\_result)

plotcp(regression\_result)

**

*Fig 2.5 Decision tree built by Decision tree algorithm*

Cp value (Complexity parameter) is determined by growing the decision tree till over fitting and then determine the cp values for which the ‘x error’ is the least.



*Fig 2.6 plotcp(regression\_result) cp with min x error = 0.00085*

### **2.2.3 Random Forest**

Random forest is a tree-based algorithm which involves building several trees (decision trees), then combining their output to improve generalization ability of the model. The method of combining trees is known as an ensemble method. For our modelling we have used 50 decision trees in our random forest algorithm

##2. Random Forest algorithm

RF\_model = randomForest(fare\_amount ~ ., train, importance = TRUE, ntree = 50, mtry = 2)

#Predict test data using random forest model

RF\_Predictions = predict(RF\_model, test[,-1])

#accuracy\_RF = 100 - (mape(cab\_data\_test[,1], RF\_Predictions))

rmse\_RF = Rmse(test[,1], RF\_Predictions)

rmse\_RF

#regr.eval(test[,1], RF\_Predictions, stats =c('mae', 'rmse', 'mse'))

bestmtry <- tuneRF(cab\_data\_train, cab\_data\_train$fare\_amount, stepFactor = 2, improve = 0.01, trace = T, plot = T)

### **2.2.4 Linear Regression**

Linear regression is one of the most commonly used predictive modelling techniques. The aim of linear regression is to find a mathematical equation for a continuous response variable Y as a function of one or more X variable(s). So that you can use this regression model to predict the Y when only the X is known.

> **#run regression model**

> lm\_model = lm(fare\_amount ~., data = train)

> **#Summary of the model**

> summary(lm\_model)

Call:

lm(formula = fare\_amount ~ ., data = train)

Residuals:

Min 1Q Median 3Q Max

-12.6067 -1.3809 -0.4813 0.8455 18.2205

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 5.381e+02 1.154e+02 4.663 3.16e-06 \*\*\*

pickup\_longitude 4.818e+00 1.119e+00 4.305 1.68e-05 \*\*\*

pickup\_latitude -4.370e+00 1.116e+00 -3.915 9.11e-05 \*\*\*

passenger\_count 6.431e-02 1.868e-02 3.442 0.000579 \*\*\*

pickup\_hour 7.099e-03 3.652e-03 1.944 0.051974 .

pickup\_weekday 4.631e-02 1.175e-02 3.941 8.19e-05 \*\*\*

Distance\_Haversine 2.020e-03 1.646e-05 122.712 < 2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.283 on 9493 degrees of freedom

Multiple R-squared: 0.6156, Adjusted R-squared: 0.6154

F-statistic: 2534 on 6 and 9493 DF, p-value: < 2.2e-16

The summary statistics above tells us several things.

One of them is the model’s p-Value (in last line) and the p-Value of individual predictor variables (extreme right column under ‘Coefficients’).

The p-Values are very important and the variables whose p values are less than 0.05 are taken for linear prediction of fare amount.

Because, we can consider a linear model to be statistically significant only when both these p-Values are less than the pre-determined statistical significance level of 0.05.

This can visually interpreted by the significance stars at the end of the row against each X variable.

The more the stars beside the variable’s p-Value, the more significant the variable.

# **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. Interpretability
3. Computational Eﬃciency

In our case of Bike rental Data, the latter two, Interpretability and Computation Eﬃciency, do not hold much signiﬁcance. Therefore we will use Predictive performance as the criteria to compare and evaluate models. Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

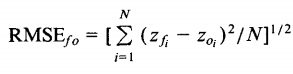
There are so many error metrics available for regression analysis such as *Mean Absolute error (MAE), Mean Squared Error (MSE), mean absolute percentage error (MAPE) and Root Mean Squared Error (RMSE).* MSE and RMSE are widely used for time series analysis and therefore we are going to use MAPE to evaluate our model.

## **3.2 Root Mean Squared Error:**

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

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

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

The bar above the squared differences is the mean (similar to x̄). The same formula can be written with the following, slightly different, notation (Barnston, 1992):  
 [](https://www.statisticshowto.datasciencecentral.com/wp-content/uploads/2016/10/root-mean-square-error.png)  
**Where**:

Σ = [summation](https://www.statisticshowto.datasciencecentral.com/summation/)(“add up”)

(zfi – Zoi)Sup>2 = differences, squared

N = [sample size](https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/find-sample-size/).

You can use whichever formula you feel most comfortable with, as they both do the same thing. **If you don’t like formulas, you can find the RMSE by:**

* Squaring the residuals.
* Finding the [average](https://www.statisticshowto.datasciencecentral.com/average/)of the residuals.
* Taking the square root of the result.

#**calculate RMSE**

Rmse = function(x,xhat){

sqrt(mean(x-xhat)^2)

}

|  |
| --- |
| [1] "RMSE of Decision Tree Method"  > rmse\_DT  [1] 0.02744117  [1] "RMSE of Random Forest Method"  > rmse\_RF  [1] 0.01481105  [1] "RMSE of Linear Regression Method"  > rmse\_LR  [1] 0.01877469 |
|  |
| |  | | --- | |  | |

## **Model Selection**

It can be seen that there is no much difference between the RMSE Value of the three models evaluated and any one of the models can be used to predict the cab fare amount. In case of python code, Random Forest provides more accuracy compared to other two models.

# **Appendix A – R Code**

rm(list = ls())

#install.packages(c("ggplot2", "corrgram", "rpart", "geosphere"))

library(ggplot2)

library(DMwR)

library(dplyr)

library(corrgram)#for correlation calculaltion

library(rpart)#decision tree alg

library(rpart.plot)

library(randomForest)#for random forest algorithm

library(geosphere)

#set working directory

setwd("E:/EDW/Projects/Cab Fare Prediction")

#Loading Train Data and Test data to R environment

cab\_data\_train = read.csv("train\_cab\_data.csv", header = T)

cab\_data\_test = read.csv("test\_cab\_data.csv", header = T)

str(cab\_data\_train) #'data.frame': 16067 obs. of 07 variables

#Converting to required data types

cab\_data\_train$fare\_amount <- as.numeric(as.character(cab\_data\_train$fare\_amount))

cab\_data\_train$pickup\_datetime = strptime(x=as.character(cab\_data\_train$pickup\_datetime), format = "%Y-%m-%d %H:%M:%S", tz = "UTC")

#Train and Test data Exploration

train\_sum <- summary(cab\_data\_train)

test\_sum <- summary(cab\_data\_test)

test\_summary <- as.data.frame(sapply(cab\_data\_test[ ,c("pickup\_longitude", "pickup\_latitude", "dropoff\_longitude", "dropoff\_latitude", "passenger\_count")], summary))

#Outlier Limit Values determined from Test data

min\_pickup\_long = test\_summary$pickup\_longitude[1]

min\_pickup\_lat = test\_summary$pickup\_latitude[1]

min\_drop\_long = test\_summary$dropoff\_longitude[1]

min\_drop\_lat = test\_summary$dropoff\_latitude[1]

max\_pickup\_long = test\_summary$pickup\_longitude[6]

max\_pickup\_lat = test\_summary$pickup\_latitude[6]

max\_drop\_long = test\_summary$dropoff\_longitude[6]

max\_drop\_lat = test\_summary$dropoff\_latitude[6]

min\_passenger\_cnt = test\_summary$passenger\_count[1]

max\_passenger\_cnt = test\_summary$passenger\_count[6]

#Finding Latitude and Longitude Limits

min\_latitude = min(min\_pickup\_lat, min\_drop\_lat)

max\_latitude = max(max\_pickup\_lat, max\_drop\_lat)

min\_longitude = min(min\_pickup\_long, min\_drop\_long)

max\_longitude = max(max\_pickup\_long, max\_drop\_long)

#extrapolating hours, months, years values from date time field

cab\_data\_train$pickup\_year <- cab\_data\_train$pickup\_datetime$year+1900

cab\_data\_train$pickup\_month <- cab\_data\_train$pickup\_datetime$mon

cab\_data\_train$pickup\_weekday <- cab\_data\_train$pickup\_datetime$wday

cab\_data\_train$pickup\_hour <- cab\_data\_train$pickup\_datetime$hour

str(cab\_data\_train) #'data.frame': 16067 obs. of 11 variable

cab\_data\_train$pickup\_datetime <- as.POSIXct(cab\_data\_train$pickup\_datetime)

#Missing Value analysis:

missing\_val = data.frame(apply(cab\_data\_train,2,function(x){sum(is.na(x))}))

missing\_val$Columns = row.names(missing\_val)

names(missing\_val)[1] = "Missing Value Count"

missing\_val$Missing\_percentage = (missing\_val$`Missing Value Count`/nrow(cab\_data\_train)) \* 100

missing\_val = missing\_val[order(-missing\_val$Missing\_percentage),]

row.names(missing\_val) = NULL

missing\_val = missing\_val[,c(2,1,3)]

#

# #Actual value

# # > cab\_data\_train$passenger\_count[14] # [1] 2

# # > cab\_data\_train$passenger\_count[36] # [1] 6

# # > cab\_data\_train$passenger\_count[52] # [1] 4

# cab\_data\_train$passenger\_count[14] = NA

# cab\_data\_train$passenger\_count[36] = NA

# cab\_data\_train$passenger\_count[52] = NA

# # #Mean:

# # > cab\_data\_train$passenger\_count[14]#[1] 2.629447 #Without Outliers - 1.649872

# # > cab\_data\_train$passenger\_count[36]#[1] 2.629447

# # > cab\_data\_train$passenger\_count[52]#[1] 2.629447

# #Median

# # > cab\_data\_train$passenger\_count[14]# [1] 1 #Without Outliers - 1

# # > cab\_data\_train$passenger\_count[36]# [1] 1

# # > cab\_data\_train$passenger\_count[52]# [1] 1

# #kNN Imputation

# # > cab\_data\_train$passenger\_count[14]# [1] 1

# # > cab\_data\_train$passenger\_count[36]# [1] 1.325

# # > cab\_data\_train$passenger\_count[52]# [1] 1.296

#

# #Mean Method

# #cab\_data\_train$passenger\_count[is.na(cab\_data\_train$passenger\_count)] = mean(cab\_data\_train$passenger\_count, na.rm = T)

#

# #Median Method

# #cab\_data\_train$passenger\_count[is.na(cab\_data\_train$passenger\_count)] = median(cab\_data\_train$passenger\_count, na.rm = T)

#

# # kNN Imputation

# #temp\_df <- cab\_data\_train[, -2 ]

# cab\_data\_train = knnImputation(cab\_data\_train[, -2], k = 3)

# #sum(is.na(cab\_data\_train))

# row.names(cab\_data\_train) <- NULL

#removing outliers in longitude and latitude values based on limit values from test data

cab\_data\_train <- cab\_data\_train[!((cab\_data\_train$pickup\_longitude == 0) | (cab\_data\_train$pickup\_latitude == 0) | (cab\_data\_train$dropoff\_longitude==0) | (cab\_data\_train$dropoff\_latitude == 0)), ]

nrow(cab\_data\_train) #15741

cab\_data\_train <- cab\_data\_train[!((cab\_data\_train$pickup\_longitude <= min\_longitude) | (cab\_data\_train$pickup\_longitude >= max\_longitude) | (cab\_data\_train$dropoff\_longitude >= max\_longitude) | (cab\_data\_train$dropoff\_longitude <= min\_longitude)), ]

nrow(cab\_data\_train) #15729

cab\_data\_train <- cab\_data\_train[ !((cab\_data\_train$pickup\_latitude <= min\_latitude) | (cab\_data\_train$pickup\_latitude >= max\_latitude) | (cab\_data\_train$dropoff\_latitude >= max\_latitude) | (cab\_data\_train$dropoff\_latitude <= min\_latitude)) , ]

nrow(cab\_data\_train) #15714

#Calculating Distance from Latitudes and Longitudes

cab\_data\_train$Distance\_Haversine = distHaversine(cbind(cab\_data\_train$pickup\_longitude, cab\_data\_train$pickup\_latitude),cbind(cab\_data\_train$dropoff\_longitude, cab\_data\_train$dropoff\_latitude))

cab\_data\_train$Distance\_Meeus = distMeeus(cbind(cab\_data\_train$pickup\_longitude, cab\_data\_train$pickup\_latitude),cbind(cab\_data\_train$dropoff\_longitude, cab\_data\_train$dropoff\_latitude))

cab\_data\_train$Distance\_Cosine = distCosine(cbind(cab\_data\_train$pickup\_longitude, cab\_data\_train$pickup\_latitude),cbind(cab\_data\_train$dropoff\_longitude, cab\_data\_train$dropoff\_latitude))

cab\_data\_train$Distance\_Vincenty = distVincentyEllipsoid(cbind(cab\_data\_train$pickup\_longitude, cab\_data\_train$pickup\_latitude),cbind(cab\_data\_train$dropoff\_longitude, cab\_data\_train$dropoff\_latitude))

#Outlier analysis

numeric\_index = sapply(cab\_data\_train,is.numeric) #selecting only numeric

numeric\_data = cab\_data\_train[,numeric\_index]

cnames = colnames(numeric\_data)

for (i in 1:length(cnames))

{

assign(paste0("boxplot",i), ggplot(aes\_string(y = (cnames[i])), data = subset(cab\_data\_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[i])+

ggtitle(paste("Box plot:",cnames[i])))

}

# Plotting box plots together for dependant variables

gridExtra::grid.arrange(boxplot1, boxplot2, ncol=2)

gridExtra::grid.arrange(boxplot3, boxplot4,ncol=2)

gridExtra::grid.arrange(boxplot5, boxplot6, ncol=2)

gridExtra::grid.arrange(boxplot7, boxplot8,ncol=2)

gridExtra::grid.arrange(boxplot9, boxplot10,ncol=2)

gridExtra::grid.arrange(boxplot11, boxplot12,ncol=2)

gridExtra::grid.arrange(boxplot13, boxplot14,ncol=2)

#removing outliers in passenger count as the max passenger count can only be 6 (based on test data)

n1 <- sapply(cab\_data\_train$pickup\_datetime, is.na) #identifying the NA value in datetime column and removing it

cab\_data\_train <- cab\_data\_train[-which(n1), ]

nrow(cab\_data\_train) ##15713

n2 <- sapply(cab\_data\_train$fare\_amount, is.na) #identifying the NA value in fare\_amount column and removing it

cab\_data\_train <- cab\_data\_train[-which(n2), ]

cab\_data\_train <- cab\_data\_train[-which(cab\_data\_train$fare\_amount<0), ] #removing negative values of fare amount

nrow(cab\_data\_train) ##15687

#row.names(cab\_data\_train) <- NULL

#removing passengers greater than 6 and less than 1

cab\_data\_train <- cab\_data\_train[which((cab\_data\_train$passenger\_count <= max\_passenger\_cnt) & (cab\_data\_train$passenger\_count >= min\_passenger\_cnt)), ]

nrow(cab\_data\_train) #15559 after removing outliers in passenger count

#removing outliers in fare amount

fare\_outlier = cab\_data\_train$fare\_amount[cab\_data\_train$fare\_amount %in% boxplot.stats(cab\_data\_train$fare\_amount)$out]

cab\_data\_train = cab\_data\_train[which(!cab\_data\_train$fare\_amount %in% fare\_outlier),]

nrow(cab\_data\_train) #14210 after removing outliers

#removing outliers in Distance

dist\_outlier = cab\_data\_train$Distance\_Haversine[cab\_data\_train$Distance\_Haversine %in% boxplot.stats(cab\_data\_train$Distance\_Haversine)$out]

cab\_data\_train = cab\_data\_train[which(!cab\_data\_train$Distance\_Haversine %in% dist\_outlier),]

nrow(cab\_data\_train) #13572 after removing outliers

#BoxPlots after removing outliers

numeric\_index = sapply(cab\_data\_train,is.numeric) #selecting only numeric

numeric\_data = cab\_data\_train[,numeric\_index]

cnames = colnames(numeric\_data)

for (i in 1:length(cnames))

{

assign(paste0("boxplot",i+14), ggplot(aes\_string(y = (cnames[i])), data = subset(cab\_data\_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[i])+

ggtitle(paste("Box plot-Outliers Removed: ",cnames[i])))

}

# Plotting box plots together for dependant variables

gridExtra::grid.arrange(boxplot15, boxplot16, ncol=2)

gridExtra::grid.arrange(boxplot17, boxplot18,ncol=2)

gridExtra::grid.arrange(boxplot19, boxplot20, ncol=2)

gridExtra::grid.arrange(boxplot21, boxplot22,ncol=2)

gridExtra::grid.arrange(boxplot23, boxplot24,ncol=2)

gridExtra::grid.arrange(boxplot25, boxplot26,ncol=2)

gridExtra::grid.arrange(boxplot27, boxplot28,ncol=2)

#Missing Value analysis after removing Outliers:

missing\_val = data.frame(apply(cab\_data\_train,2,function(x){sum(is.na(x))}))

missing\_val$Columns = row.names(missing\_val)

names(missing\_val)[1] = "Missing Value Count"

missing\_val$Missing\_percentage = (missing\_val$`Missing Value Count`/nrow(cab\_data\_train)) \* 100

missing\_val = missing\_val[order(-missing\_val$Missing\_percentage),]

row.names(missing\_val) = NULL

missing\_val = missing\_val[,c(2,1,3)]

#No missing Values found

#Visualization:

#distribution of variables

ggplot(data= cab\_data\_train, aes(x=fare\_amount)) + ggtitle(paste("Distribution of Fare\_Amount"))+ geom\_freqpoly(bins =5)

ggplot(data= cab\_data\_train, aes(x=pickup\_hour)) + ggtitle(paste("Distribution of Hour"))+ geom\_freqpoly(bins = 5)

ggplot(data= cab\_data\_train, aes(x=pickup\_weekday)) + ggtitle(paste("Distribution of Weekday"))+ geom\_freqpoly(bins = 5)

ggplot(data= cab\_data\_train, aes(x=pickup\_month)) + ggtitle(paste("Distribution of Pickup\_Month"))+ geom\_freqpoly(bins = 5)

ggplot(data= cab\_data\_train, aes(x=Distance\_Haversine)) +ggtitle(paste("Distribution of Distance"))+ geom\_freqpoly(bins = 5)

ggplot(data= cab\_data\_train, aes(x=passenger\_count)) +ggtitle(paste("Distribution of Passenger Count"))+ geom\_freqpoly(bins = 5)

cleaned\_data <- write.csv(cab\_data\_train, file = "cleaned\_cab\_train\_data.csv")

##Feature Selection - Correlation Plot

corrgram(cab\_data\_train[,numeric\_index], order = F, upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

cab\_data\_train <- cab\_data\_train[, c("fare\_amount","pickup\_longitude", "pickup\_latitude", "passenger\_count", "pickup\_hour", "pickup\_weekday", "Distance\_Haversine")]

#Matching the data frame structure of test and train data set

#Converting to required data types

cab\_data\_test$fare\_amount <- 0

cab\_data\_test$pickup\_datetime = strptime(x=as.character(cab\_data\_test$pickup\_datetime), format = "%Y-%m-%d %H:%M:%S", tz = "UTC")

#extrapolating hours, months, years values from date time field for test data

cab\_data\_test$pickup\_year <- cab\_data\_test$pickup\_datetime$year+1900

cab\_data\_test$pickup\_month <- cab\_data\_test$pickup\_datetime$mon

cab\_data\_test$pickup\_weekday <- cab\_data\_test$pickup\_datetime$wday

cab\_data\_test$pickup\_hour <- cab\_data\_test$pickup\_datetime$hour

cab\_data\_test$Distance\_Haversine = distHaversine(cbind(cab\_data\_test$pickup\_longitude, cab\_data\_test$pickup\_latitude),cbind(cab\_data\_test$dropoff\_longitude, cab\_data\_test$dropoff\_latitude))

str(cab\_data\_test) ##'data.frame': 9914 obs. of 11 variables

cleaned\_test\_data <- write.csv(cab\_data\_test, file = "cleaned\_cab\_test\_data.csv")

cab\_data\_test <- cab\_data\_test[, c("fare\_amount","pickup\_longitude", "pickup\_latitude", "passenger\_count", "pickup\_hour", "pickup\_weekday", "Distance\_Haversine")]

#Divide the data into train and test

train\_index = sample(1:nrow(cab\_data\_train), 0.70 \* nrow(cab\_data\_train))

train = cab\_data\_train[train\_index,]

test = cab\_data\_train[-train\_index,]

#1. Decision tree algorith for regression

# cab\_data\_test <- cab\_data\_test[ , -1]

# cab\_data\_train <- cab\_data\_train[, -2]

##Regression algorithm - anova

regression\_result = rpart(fare\_amount ~ ., data = train, method = "anova", control = rpart.control(cp = 0.00085, minsplit = 4, minbucket = 5, maxdepth = 10))

#prediction of test values

predict\_DT = predict(regression\_result, test[,-1])

rpart.plot(regression\_result, type = 1)

printcp(regression\_result)

plotcp(regression\_result)

#

# #MAPE

# #calculate MAPE

# mape = function(y, yhat){

# mean(abs((y - yhat)))\*100

# }

#RMSE

#calculate RMSE

Rmse = function(x,xhat){

sqrt(mean(x-xhat)^2)

}

#accuracy\_DT = 100 - (mape(cab\_data\_test[,6], predict\_DT))

rmse\_DT = Rmse(test[,1], predict\_DT)

rmse\_DT

#alternate method to find error metrics for regression.

#regr.eval(test[,1], predict\_DT, stats = c('mae', 'rmse', 'mse'))

##2. Random Forest algorithm

RF\_model = randomForest(fare\_amount ~ ., train, importance = TRUE, ntree = 50, mtry = 2)

#Predict test data using random forest model

RF\_Predictions = predict(RF\_model, test[,-1])

#accuracy\_RF = 100 - (mape(cab\_data\_test[,1], RF\_Predictions))

rmse\_RF = Rmse(test[,1], RF\_Predictions)

rmse\_RF

#regr.eval(test[,1], RF\_Predictions, stats =c('mae', 'rmse', 'mse'))

bestmtry <- tuneRF(cab\_data\_train, cab\_data\_train$fare\_amount, stepFactor = 1.5, improve = 0.01, trace = T, plot = T)

##3.Linear regression

#check multicollearity

library(usdm) #calculate Variation Inflation factor

#install.packages("rms")

require(rms)

vif(train[,-1]) #to find there is any multicollinearity in the data

vifcor(train[,-1], th = 0.9)

#run regression model

lm\_model = lm(fare\_amount ~., data = train)

# model1 <- ols(fare\_amount~., data=cab\_data\_train)

# model2 = fastbw(model1, rule="p", sls=0.4)

#Summary of the model

summary(lm\_model)

#Predict

predictions\_LR = predict(lm\_model, test[,-1])

#accuracy\_LR\_wo = 100 - (mape(cab\_data\_test[,1], predictions\_LR\_wo))

rmse\_LR = Rmse(test[,1], predictions\_LR)

rmse\_LR

#regr.eval(test[,1], predictions\_LR, stats = c('rmse'))

print("RMSE of Decision Tree Method")

rmse\_DT

print("RMSE of Random Forest Method")

rmse\_RF

print("RMSE of Linear Regression Method")

rmse\_LR

# **Appendix B – Python Code**

# In[2]:

#Load libraries\n",

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from datetime import datetime

import calendar

from geopy.distance import geodesic

from math import radians, sin, cos, acos

import math

import sklearn

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

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

#Import libraries for LR

import statsmodels.api as sm

import statistics

# In[3]:

#Set working directory

os.chdir("E:\EDW\Projects\Cab Fare Prediction")

#os.getcwd()

#'E:\\EDW\\Projects\\Cab Fare Prediction'

# In[4]:

#Load Data

cab\_data\_train = pd.read\_csv("train\_cab\_data.csv")

cab\_data\_test = pd.read\_csv("test\_cab\_data.csv")

# In[5]:

min\_longitude = min(min(cab\_data\_test['pickup\_longitude']), min(cab\_data\_test['dropoff\_longitude']))

max\_longitude = max(max(cab\_data\_test['pickup\_longitude']), max(cab\_data\_test['dropoff\_longitude']))

min\_latitude = min(min(cab\_data\_test['pickup\_latitude']), min(cab\_data\_test['dropoff\_latitude']))

max\_latitude = max(max(cab\_data\_test['pickup\_latitude']), max(cab\_data\_test['dropoff\_latitude']))

max\_passenger\_count = max(cab\_data\_test['passenger\_count'])

min\_passenger\_count = min(cab\_data\_test['passenger\_count'])

# In[6]:

#converting to datetime format

cab\_data\_train = cab\_data\_train.sort\_values(by = ['pickup\_datetime'], ascending = False)

cab\_data\_train = cab\_data\_train.iloc[1:]

cab\_data\_train['pickup\_datetime'] = pd.to\_datetime(cab\_data\_train.pickup\_datetime, format = '%Y-%m-%d %H:%M:%S UTC')

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

#Extrapolating time/day/month details

cab\_data\_train['pickup\_date']= cab\_data\_train['pickup\_datetime'].dt.date

cab\_data\_train['pickup\_day']=cab\_data\_train['pickup\_datetime'].apply(lambda x:x.day)

cab\_data\_train['pickup\_hour']=cab\_data\_train['pickup\_datetime'].apply(lambda x:x.hour)

cab\_data\_train['pickup\_day\_of\_week']=cab\_data\_train['pickup\_datetime'].apply(lambda x:calendar.day\_name[x.weekday()])

cab\_data\_train['pickup\_month']=cab\_data\_train['pickup\_datetime'].apply(lambda x:x.month)

cab\_data\_train['pickup\_year']=cab\_data\_train['pickup\_datetime'].apply(lambda x:x.year)

#calculate trip distance in Kms

def distance(lat1, lat2, lon1,lon2):

p = 0.017453292519943295 # Pi/180

a = 0.5 - np.cos((lat2 - lat1) \* p)/2 + np.cos(lat1 \* p) \* np.cos(lat2 \* p) \* (1 - np.cos((lon2 - lon1) \* p)) / 2

return 12742 \* np.arcsin(np.sqrt(a))

cab\_data\_train['trip\_distance']=cab\_data\_train.apply(lambda row:distance(row['pickup\_latitude'],row['dropoff\_latitude'],row['pickup\_longitude'],row['dropoff\_longitude']),axis=1)

# In[7]:

#Missing Values before Outlier removal

missing\_value\_b = pd.DataFrame(cab\_data\_train.isnull().sum()).reset\_index().rename(columns = {'index' : 'Variable', 0: 'Missing Values'})

missing\_value\_b['Missing\_Percentage'] = (missing\_value\_b['Missing Values'])/(len(cab\_data\_train))\*100

missing\_value\_b.sort\_values(by = ['Missing Values'], ascending = False)

# In[8]:

#removing negligible na values

cab\_data\_train = cab\_data\_train.loc[~((cab\_data\_train.fare\_amount.isnull() == True) | (cab\_data\_train.passenger\_count.isnull() == True))]

# In[9]:

#Plot boxplot to visualize Outliers\n",

get\_ipython().run\_line\_magic('matplotlib', 'inline')

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

# In[10]:

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

# In[11]:

plt.boxplot(cab\_data\_train['trip\_distance'])

# In[12]:

#removing Outlier values

cab\_data\_train = cab\_data\_train.loc[~((cab\_data\_train.pickup\_latitude==0) | (cab\_data\_train.pickup\_longitude)==0 | (cab\_data\_train.dropoff\_latitude==0)|(cab\_data\_train.dropoff\_longitude==0))]

cab\_data\_train = cab\_data\_train.loc[~((cab\_data\_train['pickup\_longitude'] > max\_longitude) | (cab\_data\_train['pickup\_longitude'] < min\_longitude) |

(cab\_data\_train['pickup\_latitude'] > max\_latitude) | (cab\_data\_train['pickup\_latitude'] < min\_latitude) |

(cab\_data\_train['dropoff\_longitude'] > max\_longitude) | (cab\_data\_train['dropoff\_longitude'] < min\_longitude) |

(cab\_data\_train['dropoff\_latitude'] > max\_latitude) | (cab\_data\_train['dropoff\_latitude'] < min\_latitude))]

cab\_data\_train = cab\_data\_train.loc[~((cab\_data\_train['passenger\_count'] > max\_passenger\_count) | (cab\_data\_train['passenger\_count'] < min\_passenger\_count))]

cab\_data\_train = cab\_data\_train.loc[~(cab\_data\_train['fare\_amount'] < 0)]

#removing negligible na values

cab\_data\_train = cab\_data\_train.loc[~((cab\_data\_train.fare\_amount.isnull() == True) | (cab\_data\_train.passenger\_count.isnull() == True))]

# In[13]:

#Identifying Outlier values in Fare\_amount and removing it

cnames = ["fare\_amount", "trip\_distance"]

for i in cnames:

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

iqr = q75 - q25

min = q25 - (iqr\*1.5)

max = q75 + (iqr\*1.5)

print(i)

print(min)

print(max)

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

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

# In[14]:

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

# In[15]:

plt.boxplot(cab\_data\_train['trip\_distance'])

# In[16]:

#missing\_value after removing outliers

missing\_value\_b = pd.DataFrame(cab\_data\_train.isnull().sum()).reset\_index().rename(columns = {'index' : 'Variable', 0: 'Missing Values'})

missing\_value\_b['Missing\_Percentage'] = (missing\_value\_b['Missing Values'])/(len(cab\_data\_train))\*100

missing\_value\_b.sort\_values(by = ['Missing Values'], ascending = False)

# In[17]:

# Convert Object field of Weekday to Numeric

# creating a dict file

weekday = {'Sunday': 0,'Monday': 1, 'Tuesday':2, 'Wednesday':3, 'Thursday':4, 'Friday':5, 'Saturday':6}

cab\_data\_train.pickup\_day\_of\_week = [weekday[item] for item in cab\_data\_train.pickup\_day\_of\_week]

# In[18]:

##Correlation analysis

#Correlation plot

df\_corr = cab\_data\_train.loc[:,:]

#Set the width and hieght of the plot

f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix

corr = df\_corr.corr()

#Plot using seaborn library\n",

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

# In[26]:

"""

#Chisquare test of independence

#Save categorical variables

from scipy.stats import chi2\_contingency

#loop for chi square values

for i in cab\_data\_train.columns:

print(i)

chi2, p, dof, ex = chi2\_contingency(pd.crosstab(cab\_data\_train['fare\_amount'], cab\_data\_train[i]))

print(p)

if p < 0.05:

print(i, "Variable used for predicting")

"""

# In[19]:

cab\_data\_train.columns

cab\_data\_train = cab\_data\_train[['fare\_amount', 'pickup\_longitude', 'pickup\_latitude', 'passenger\_count', 'pickup\_day', 'pickup\_hour', 'pickup\_day\_of\_week',

'pickup\_month', 'pickup\_year', 'trip\_distance']]

# In[20]:

cab\_data\_test['fare\_amount'] = 0

cab\_data\_test['pickup\_datetime'] = pd.to\_datetime(cab\_data\_test.pickup\_datetime, format = '%Y-%m-%d %H:%M:%S UTC')

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

#Extrapolating time/day/month details

cab\_data\_test['pickup\_date']= cab\_data\_test['pickup\_datetime'].dt.date

cab\_data\_test['pickup\_day']=cab\_data\_test['pickup\_datetime'].apply(lambda x:x.day)

cab\_data\_test['pickup\_hour']=cab\_data\_test['pickup\_datetime'].apply(lambda x:x.hour)

cab\_data\_test['pickup\_day\_of\_week']=cab\_data\_test['pickup\_datetime'].apply(lambda x:calendar.day\_name[x.weekday()])

cab\_data\_test['pickup\_month']=cab\_data\_test['pickup\_datetime'].apply(lambda x:x.month)

cab\_data\_test['pickup\_year']=cab\_data\_test['pickup\_datetime'].apply(lambda x:x.year)

#calculate trip distance in Kms

def distance(lat1, lat2, lon1,lon2):

p = 0.017453292519943295 # Pi/180

a = 0.5 - np.cos((lat2 - lat1) \* p)/2 + np.cos(lat1 \* p) \* np.cos(lat2 \* p) \* (1 - np.cos((lon2 - lon1) \* p)) / 2

return 12742 \* np.arcsin(np.sqrt(a))

cab\_data\_test['trip\_distance']=cab\_data\_test.apply(lambda row:distance(row['pickup\_latitude'],row['dropoff\_latitude'],row['pickup\_longitude'],row['dropoff\_longitude']),axis=1)

# In[21]:

cab\_data\_test = cab\_data\_test[['fare\_amount', 'pickup\_longitude', 'pickup\_latitude', 'passenger\_count', 'pickup\_day', 'pickup\_hour', 'pickup\_day\_of\_week',

'pickup\_month', 'pickup\_year', 'trip\_distance']]

# In[22]:

#Divide data into train and test

train, test = train\_test\_split(cab\_data\_train, test\_size=0.3)

# In[23]:

#######################################################################################################################

#Decision tree for regression

fit\_DT = DecisionTreeRegressor(max\_depth=2).fit(train.iloc[:,1:10], train.iloc[:,0])

#Apply model on test data

predictions\_DT = fit\_DT.predict(test.iloc[:,1:10])

#Calculate RMSE

def RMSE(y\_true, y\_pred):

rmse = math.sqrt(statistics.mean((y\_true - y\_pred)\*\*2))

return rmse

DT\_RMSE = RMSE(test.iloc[:,0], predictions\_DT)

print("RMSE - Decision tree", DT\_RMSE)

# In[31]:

#Random Forest for regression

fit\_RF = RandomForestRegressor(bootstrap=True, max\_depth=4, random\_state=0 ,n\_estimators=50).fit(train.iloc[:,1:10], train.iloc[:,0])

#Apply model on test data

predictions\_RF = fit\_RF.predict(test.iloc[:,1:10])

RF\_RMSE = RMSE(test.iloc[:,0], predictions\_RF)

print("RMSE - Random Forest", RF\_RMSE)

# In[26]:

# Train the model using the training sets

linear\_model = sm.OLS(train.iloc[:,0], train.iloc[:,1:10]).fit()

# Print out the statistics

linear\_model.summary()

# make the predictions by the model

predictions\_LR = linear\_model.predict(test.iloc[:,1:10])

#Calculate MAPE

LR\_RMSE = RMSE(test.iloc[:,0], predictions\_LR)

print("RMSE - Linear Regression ", LR\_RMSE)