CAB FARE PREDICTION

Raghav Kotwal

17 JUNE 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

Dataset Details:
2 separate data set for train and test data
Train Data:
Number of Attributes: 7
Number of Observations: 16067
Missing Values: Yes
Train Data:
Number of Attributes: 6
Number of Observations: 9914
Missing Values: No

	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
5	12.1	2011-01-06 09:50:45 UTC	-74.000964	40.731630	-73.972892	40.758233	1.0
6	7.5	2012-11-20 20:35:00 UTC	-73.980002	40.751662	-73.973802	40.764842	1.0
7	16.5	2012-01-04 17:22:00 UTC	-73.951300	40.774138	-73.990095	40.751048	1.0
8	NaN	2012-12-03 13:10:00 UTC	-74.006462	40.726713	-73.993078	40.731628	1.0
9	8.9	2009-09-02 01:11:00 UTC	-73.980658	40.733873	-73.991540	40.758138	2.0

Table 1.1 Structure of train Data

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1
5	2012-12-01 21:12:12 UTC	-73.960983	40.765547	-73.979177	40.740053	1
6	2011-10-06 12:10:20 UTC	-73.949013	40.773204	-73.959622	40.770893	1
7	2011-10-06 12:10:20 UTC	-73.777282	40.646636	-73.985083	40.759368	1
8	2011-10-06 12:10:20 UTC	-74.014099	40.709638	-73.995106	40.741365	1
9	2014-02-18 15:22:20 UTC	-73.969582	40.765519	-73.980686	40.770725	1

Table 1.2 Structure of test Data

The table below contains the variables which we will be using to predict the target variable.

S.No.	Predictor
1	fare_amount
2	pickup_datetime
3	pickup_longitude
4	pickup_latitude
5	dropoff_longitude
6	dropoff_latitude
7	passenger_count

Chapter 2

Methodology

2.1 Pre-Processing

Any predictive modeling requires that we look at the data before we start modeling. 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 do missing value analysis followed by visualization of data for distribution of all the variables.

2.1.1 Missing Value Analysis/Outlier Analysis/Variable Conversion

A. In the given data, we will begin with converting data time variable into some more useful variables such as below.

• Year • Month • Date • Day of Week • Hour • Minute

There are some missing values in these date and time which we will impute using missing value analysis.

B. Passenger count should not be more than 7 even if it is an SUV.

C. Fare amount is consisting of some outliers which are removed.

D. Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rows if any latitude and longitude lies beyond the ranges.

E. Passenger count should not be zero.

F. Haversine formula is used to convert longitude

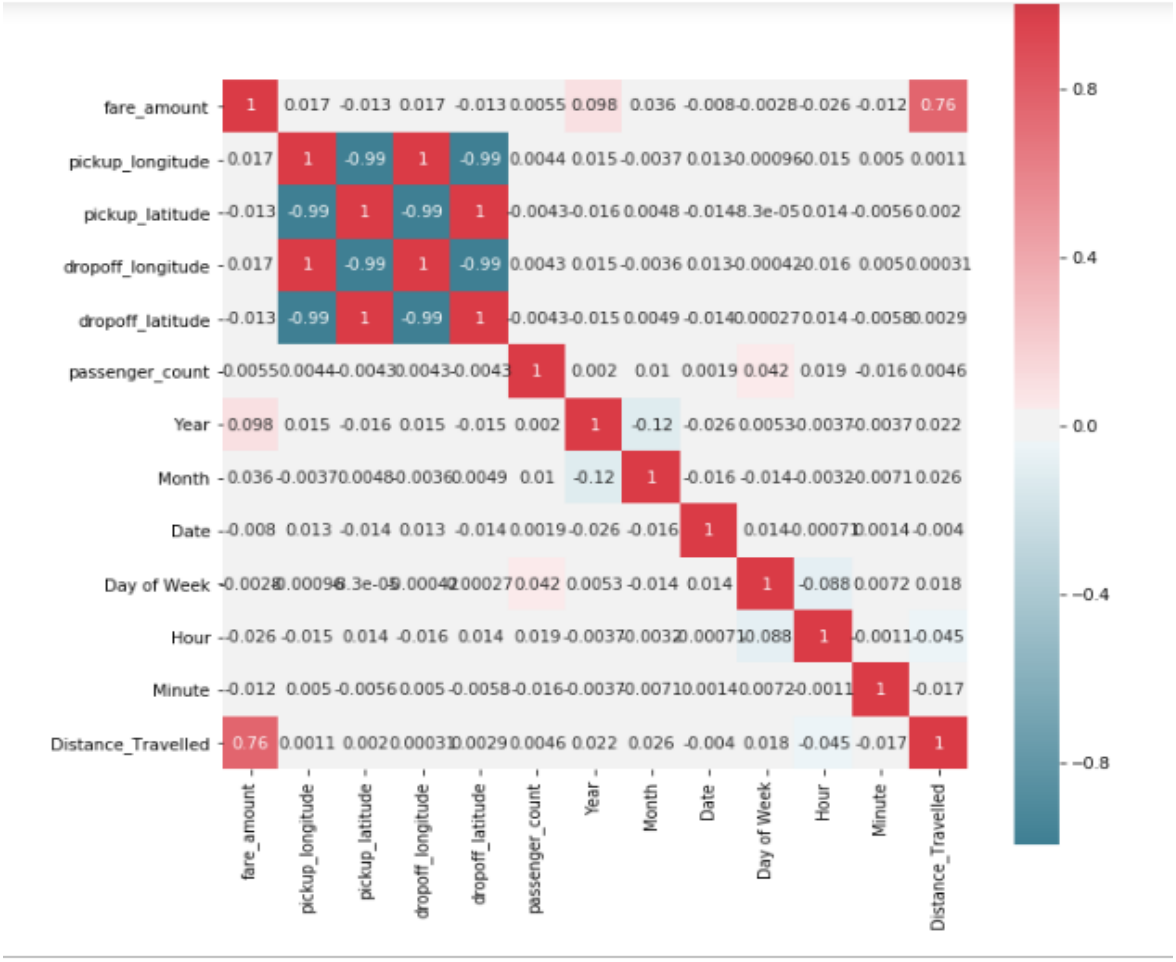
and latitude locations into distance.

2.1.2 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. We will be doing this with the use of correlation analysis.

A correlation analysis gives us the idea about the multicollinearity between different independent variables. If the correlation between two variables is high, it means they are actually saying the same thing. So, for better analysis we need to remove such variables.

The figure 4.1 shows the correlation between the numerical variables in the data.



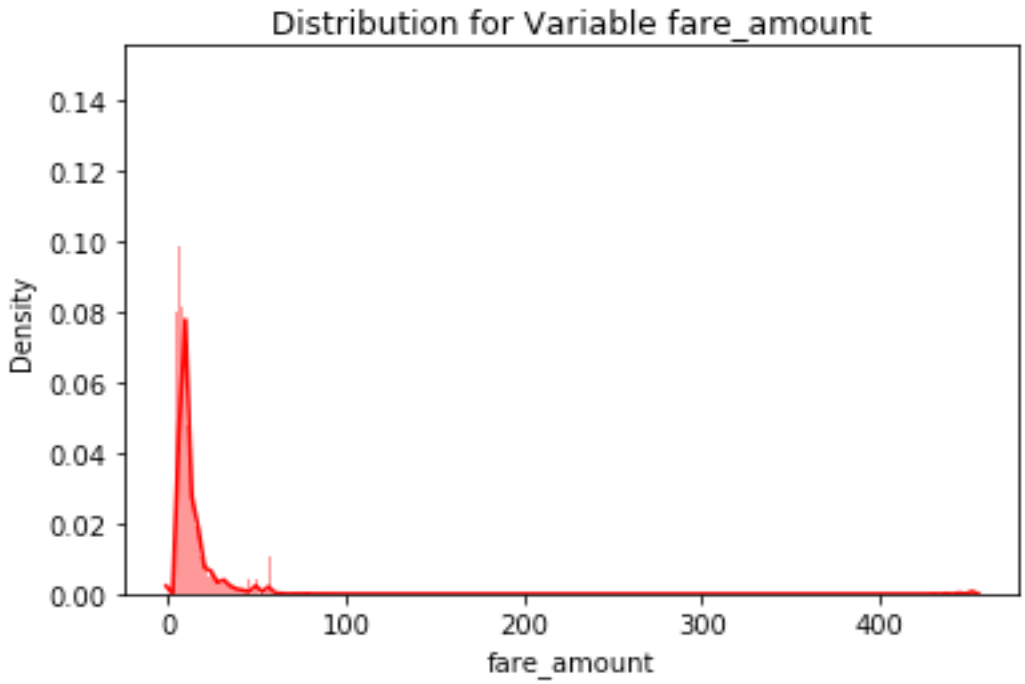
By the below table it is clear that there is no collinearity between variables.

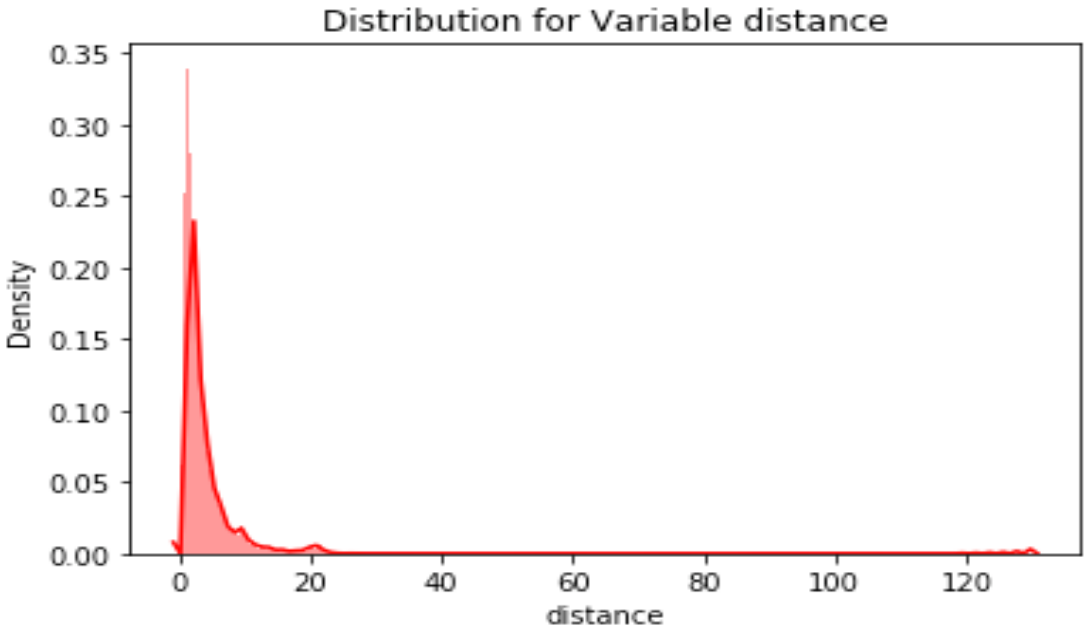
2.1.3 Feature Scaling

Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that our target variable absenteeism in hours having is one sided skewed so

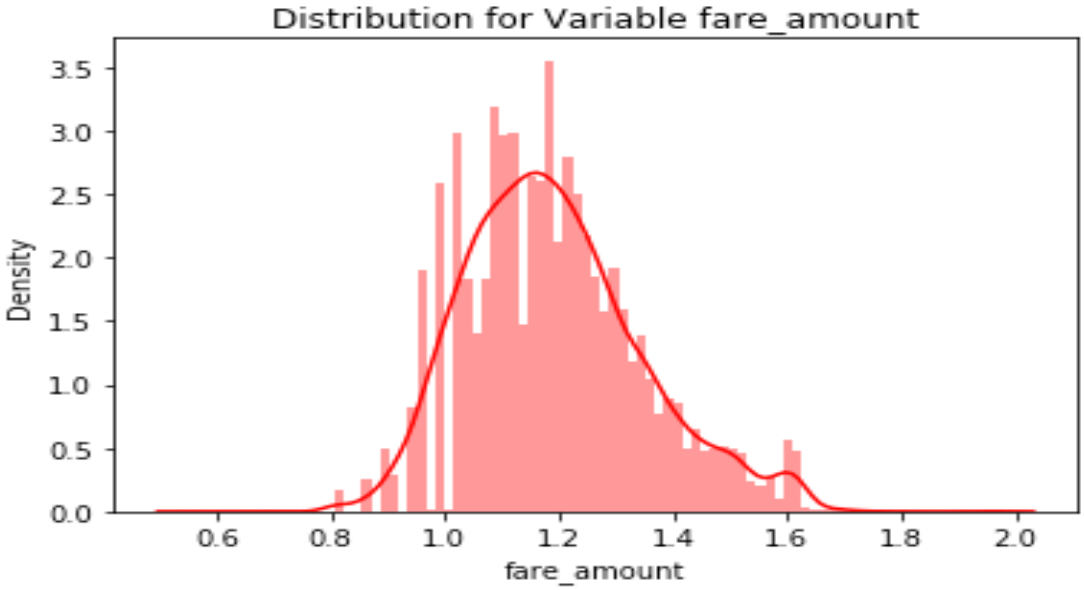
by using log transform technique we tried to reduce the skewness of the same.

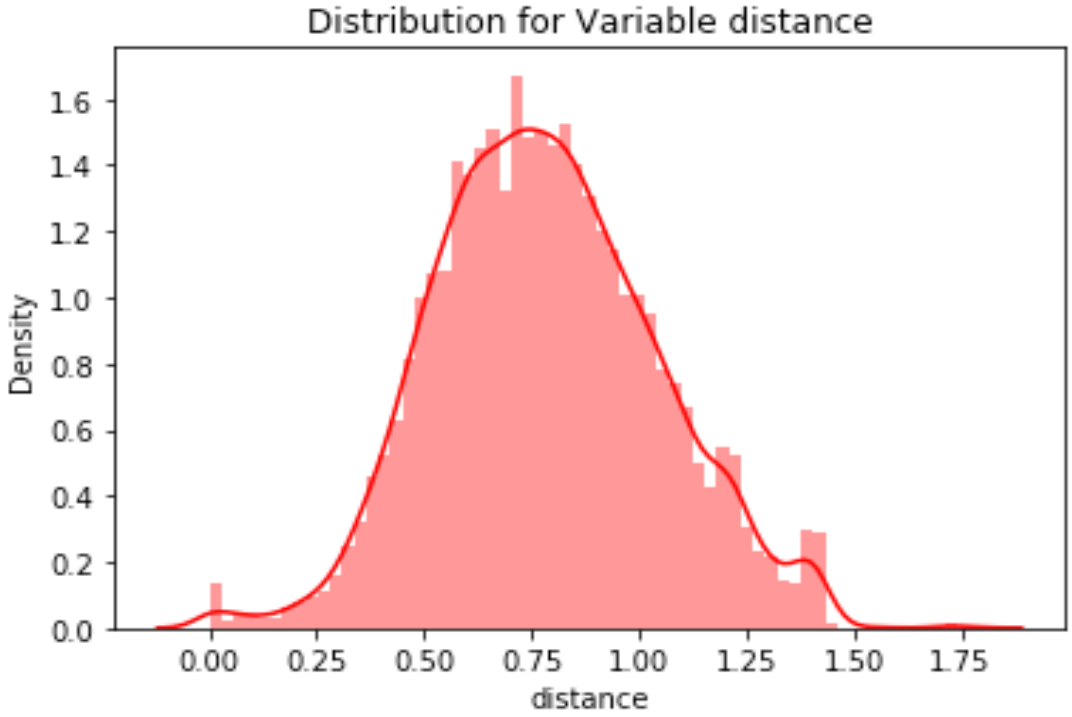
Below mentioned graphs shows the probability distribution plot to check distribution before log transformation:





Below mentioned graphs shows the probability distribution plot to check distribution after log transformation:

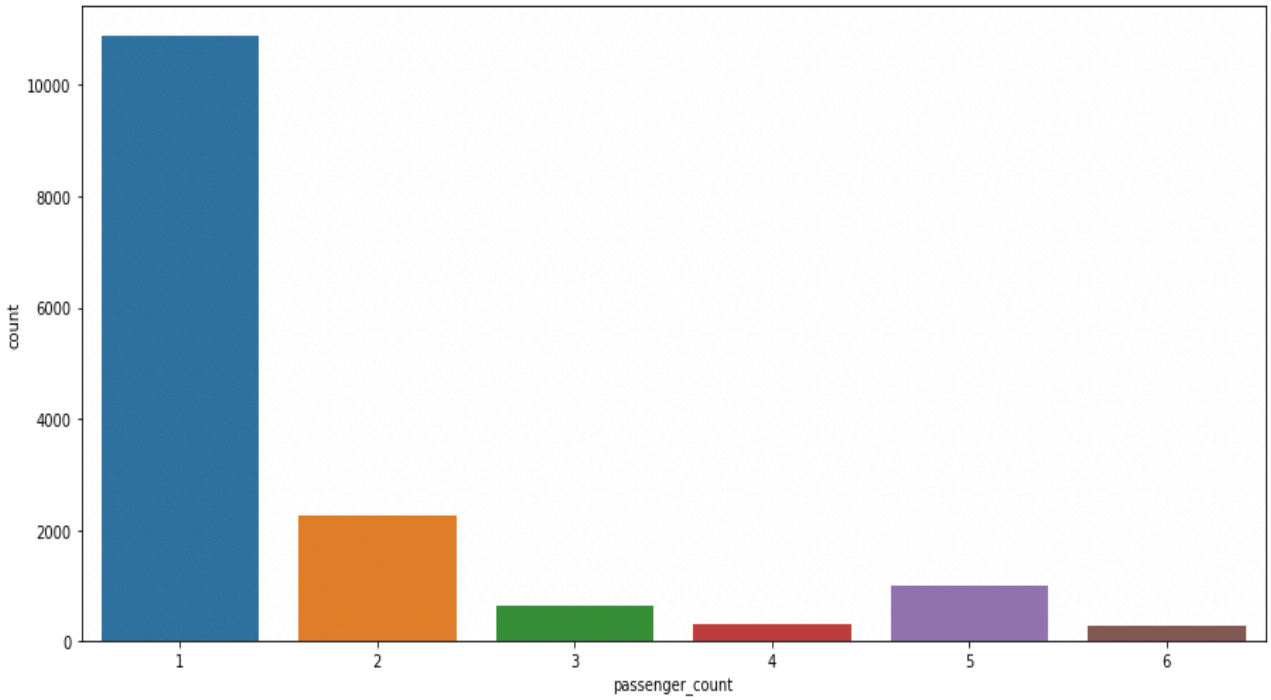




2.1.4 Visualization

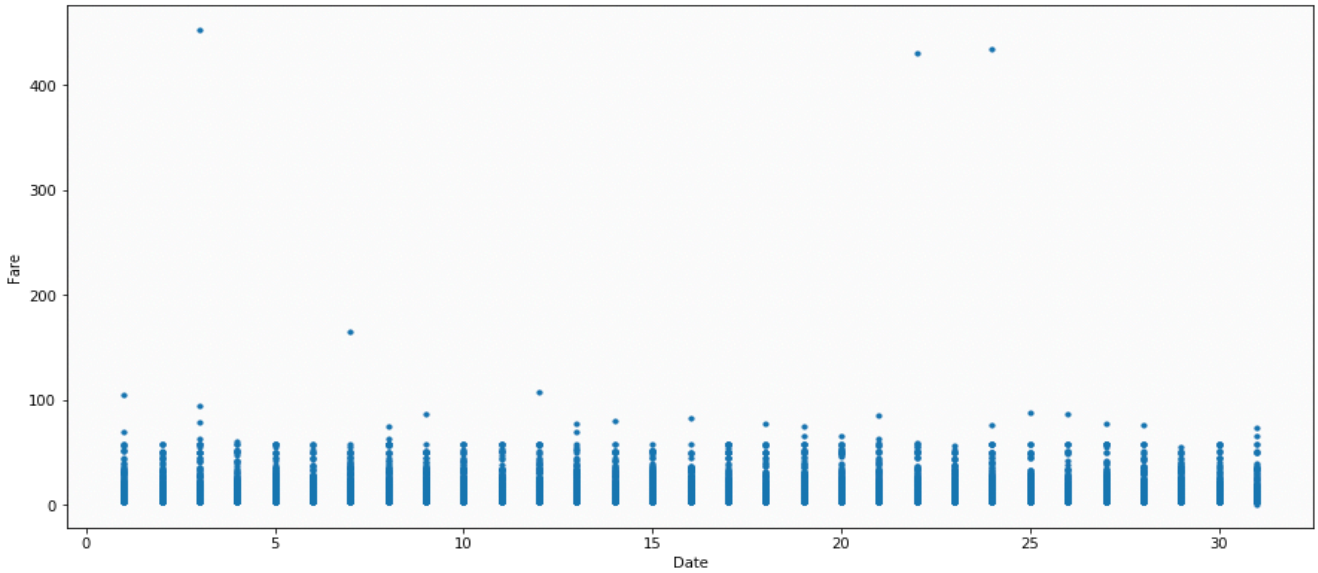
Number of passenger's vs fare

.



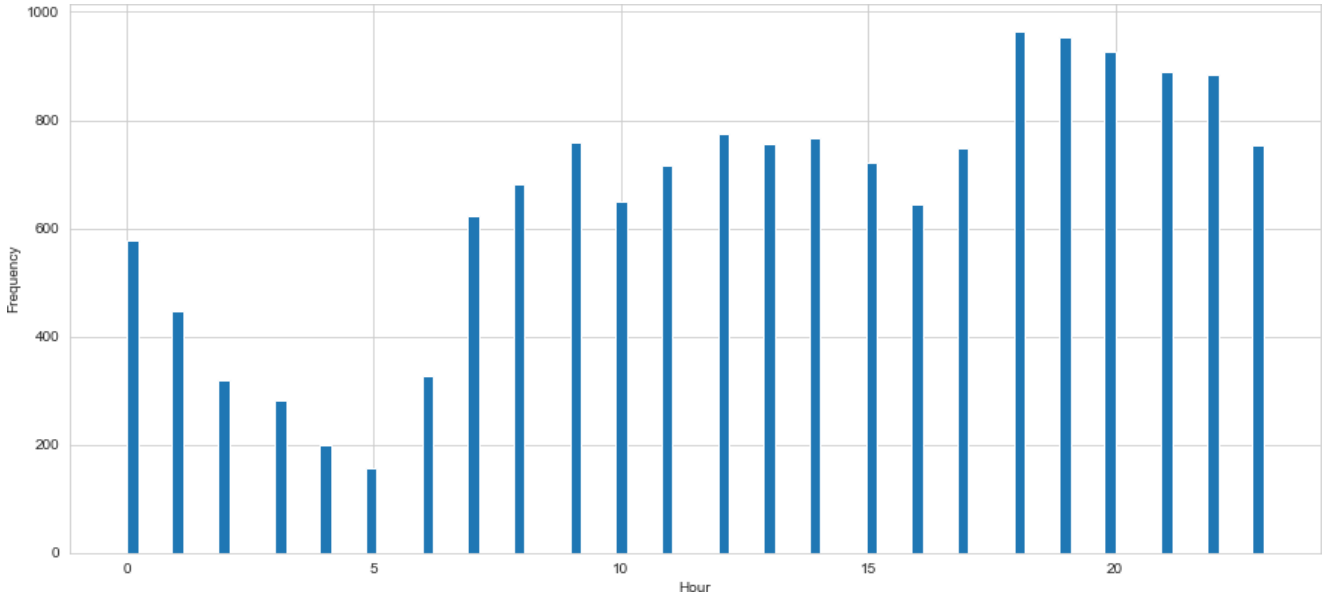
We can see in below graph that single passengers are the most frequent travellers, and the highest fare also seems to come from cabs which carry just 1 passenger.

Date of month VS fares



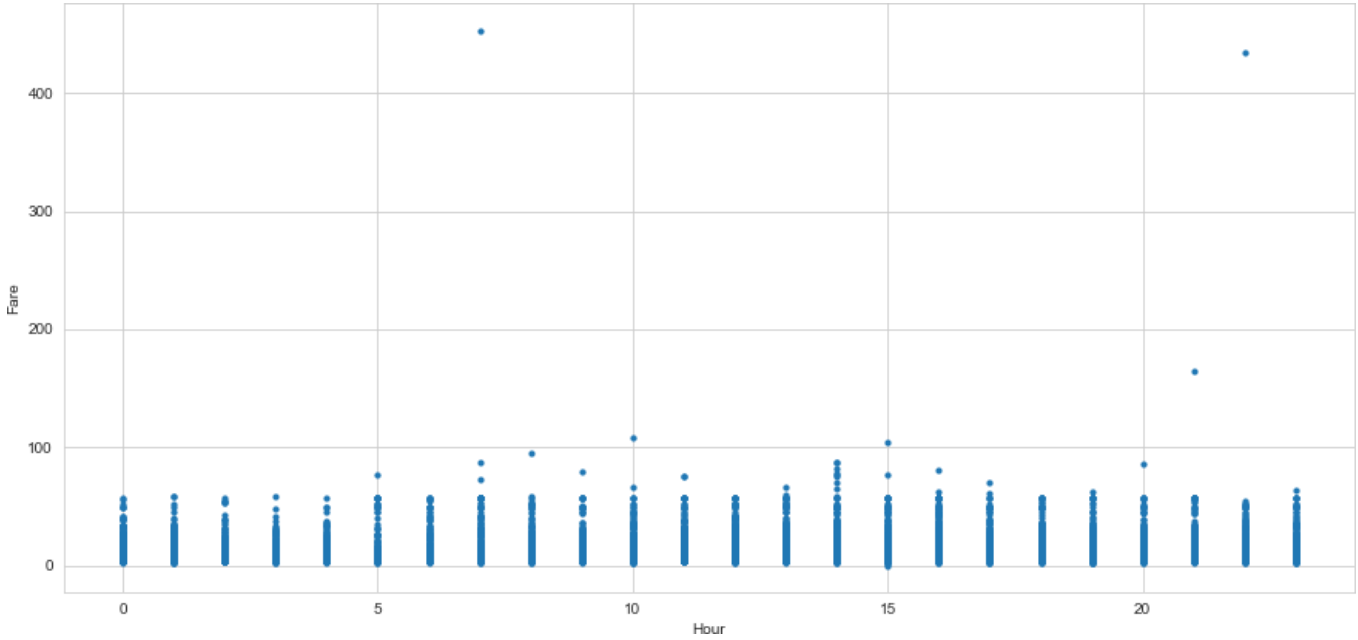
The fares throughout the month mostly seem uniform.

Hours vs Fares

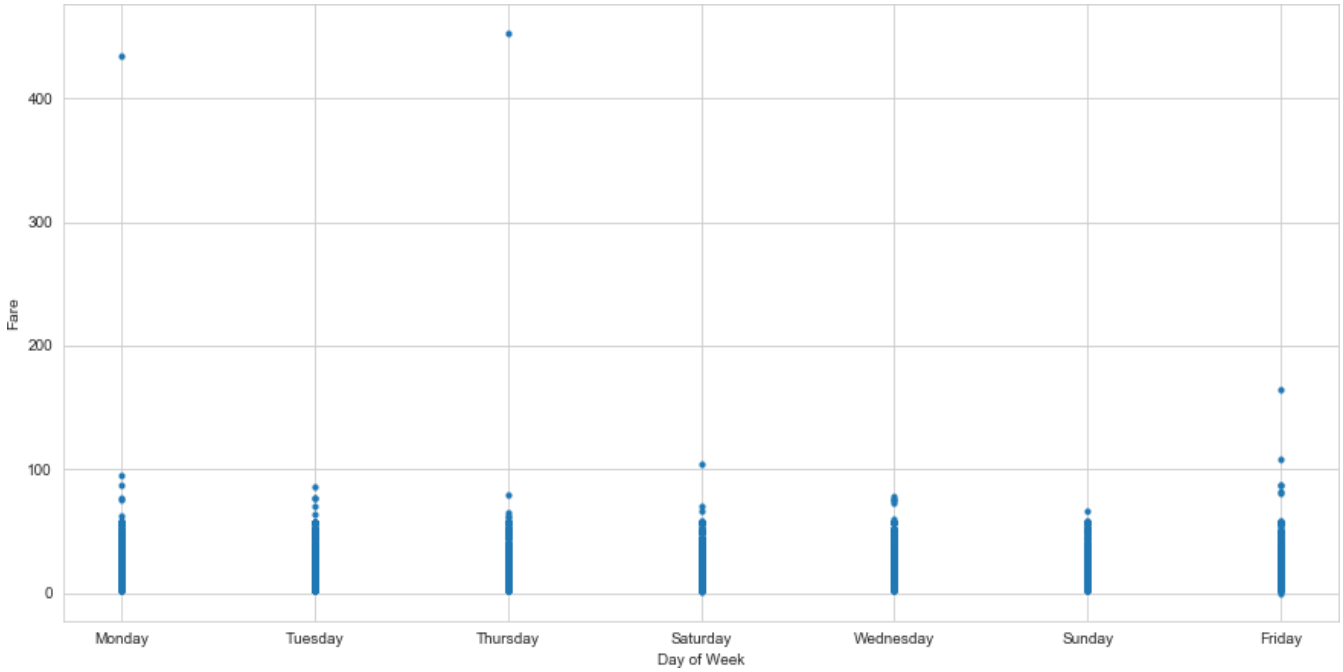


During hours 6 PM to 11PM the frequency of cab boarding is very due to peak hours

Fare prices during 2PM to 8PM is bit high compared to all other time might be due to high demands.



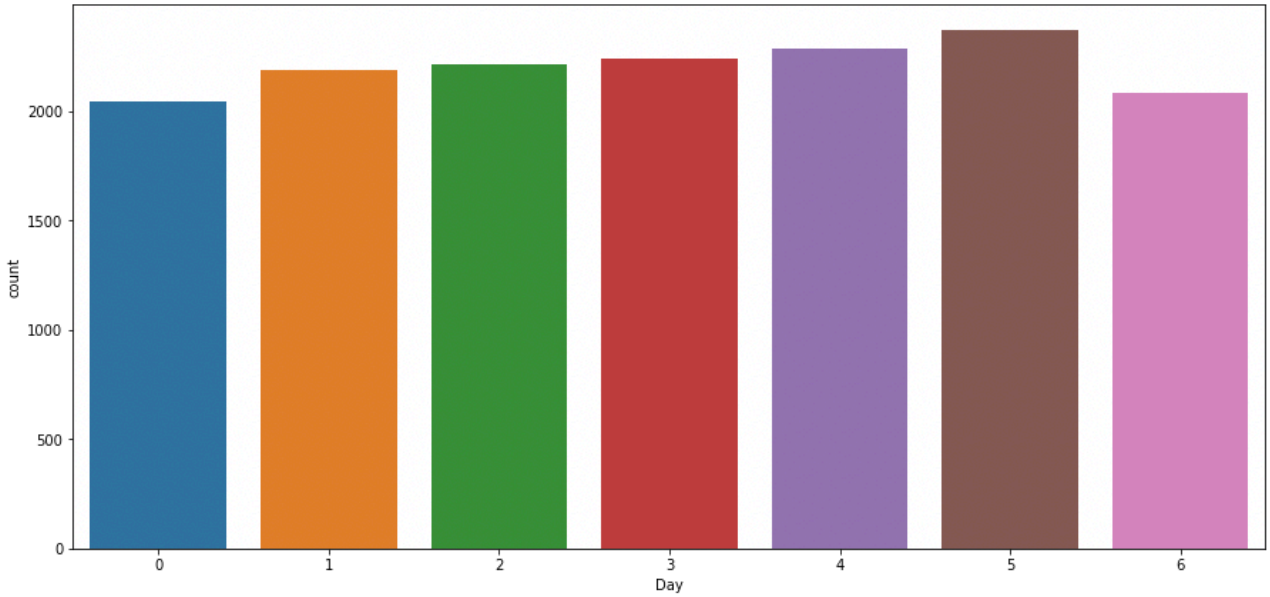
Week Day vs fare



Cab fare is high on Friday, Saturday and Monday,

may be during weekend and first day of the working day they charge high fares because of high demands of cabs.

Day vs Number of Cab rides:



Day of the week does not seem to have much influence on the number of cabs ride.

2.2 Modelling

2.2.1 Model Selection

In our early stages of analysis during pre-processing we have converted our data to non- skewed and in normal continuous form and since our target variable is also continuous, so it will be best to predict the target variable using Regression models.

We are starting the modelling with Linear regression, Decision Tree and then Random Forest.

2.2.1.1 Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an extension of simple linear regression. It is used as a predictive analysis, when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable).

```
# Building model on top of training dataset
fit_LR = LinearRegression().fit(X_train , y_train)
```

```
#prediction on train data
pred_train_LR = fit_LR.predict(X_train)
```

```
#prediction on test data
pred_test_LR = fit_LR.predict(X_test)
```

```
##calculating RMSE for test data
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

##calculating RMSE for train data
RMSE_train_LR= np.sqrt(mean_squared_error(y_train, pred_train_LR))
```

```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_LR))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_LR))
```

```
Root Mean Squared Error For Training data = 0.08193242069614293
Root Mean Squared Error For Test data = 0.07535529738213868
```

```
#calculate R^2 for train data
from sklearn.metrics import r2_score
r2_score(y_train, pred_train_LR)
```

```
0.7495502651880408
```

```
r2_score(y_test, pred_test_LR)
```

```
0.7827019104296645
```

2.2.1.2 Decision Tree

```
fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)
```

```
#prediction on train data
pred_train_DT = fit_DT.predict(X_train)

#prediction on test data
pred_test_DT = fit_DT.predict(X_test)
```

```
##calculating RMSE for train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

##calculating RMSE for test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))
```

```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))
```

```
Root Mean Squared Error For Training data = 0.08673116163909031
Root Mean Squared Error For Test data = 0.0836793140649239
```

```
## R^2 calculation for train data
r2_score(y_train, pred_train_DT)
```

```
0.6942867527622156
```

```
## R^2 calculation for test data
r2_score(y_test, pred_test_DT)
```

```
0.6969285508683054
```

2.2.1.3 Random Forest

```
fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)
```

```
#prediction on train data
pred_train_RF = fit_RF.predict(X_train)
#prediction on test data
pred_test_RF = fit_RF.predict(X_test)
```

```
##calculating RMSE for train data
RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))
##calculating RMSE for test data
RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))
```

```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))
```

```
Root Mean Squared Error For Training data = 0.09539351238936768
Root Mean Squared Error For Test data = 0.23489262331763652
```

```
## calculate R^2 for train data

r2_score(y_train, pred_train_RF)
```

```
0.9699315443839244
```

```
#calculate R^2 for test data
r2_score(y_test, pred_test_RF)
```

```
0.8009225247741288
```

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. Since our models are regression type so we will measure using the following methods.

3.1.1 RMSE :(Root Mean Square Error): Is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.

Linear Regression – 0.07

Decision Tree – 0.08

Random Forest – 0.07

3.1.2 R Squared(R^2): Is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. In other

words, we can say it explains as to how much of the variance of the target variable is explained.

Linear Regression – .78

Decision Tree – .69

Random Forest – .80

3.2 Model Conclusion

From the result of the three models used above it is very clear that random Forest has the best prediction. Thus, by using Random forest for prediction we can see that Reason for Absence has the most importance in predicting the model and also, we have seen that out of the Reason of Absence medical reasons are occurring the most (from data visualization).

Finally, I used this method to predict the target variable for the test data file shared in the problem statement. Results that I found are attached with my submissions.

Appendix (R-code)

```
# Clening the environment
rm(list = ls(all = T))
# Setting working directory
setwd("/Users/raghavkotwal/Documents/Data
Science/Cab Fare")
getwd()

#Load Libraries
x = c(
  "ggplot2",
  "corrgram",
  "DMwR",
  "caret",
  "randomForest",
  "unbalanced",
  "C50",
  "dummies",
  "e1071",
  "Information",
  "MASS",
  "rpart",
  "gbm",
  "ROSE",
  'sampling',
  'DataCombine',
  'inTrees'
)

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

```
## Reading the data
Train = read.csv('train_cab.csv')
Test = read.csv('test.csv')

#########################Exploratory          Data
Analysis#################################
# Structure of the data
str(Train)

#summary of the data set
summary(Train)

# Structure of the data
str(Test)

#summary of the data set
summary(Test)


####converting the fare_amount to to numerical ####
Train$fare_amount                                =
as.character(Train$fare_amount)
Train$fare_amount = as.numeric(Train$fare_amount)


#######################Missing              Values
Analysis#############################
sapply(Train, function(x) {
  sum(is.na(x))
})
missing_values      =      data.frame(sapply(Train,
function(x) {
  sum(is.na(x))
```

```
}))

#Calculate missing percentage and arrage in order
missing_values$Var = row.names(missing_values)
row.names(missing_values) = NULL
names(missing_values)[1] = "Percentage"
missing_values$Percentage                          =
((missing_values$Percentage / nrow(Train)) *
                                  100)
missing_values = missing_values[, c(2, 1)]
missing_values      =      missing_values[order(-
missing_values$Percentage), ]

#Create missing value and impute using mean, median
and knn
#Train[1,1]
#Train[1,1]=NA
#actual value = 4.5
##mean = 15.01
## median = 8.5


### Median Method for missing value Imputation ###
Train$fare_amount[is.na(Train$fare_amount)]      =
median(Train$fare_amount, na.rm = T)
Train$pickup_longitude[is.na(Train$pickup_longitud
e)] = median(Train$pickup_longitude, na.rm = T)
Train$pickup_latitude[is.na(Train$pickup_latitude)
] = median(Train$pickup_latitude, na.rm = T)
Train$dropoff_longitude[is.na(Train$dropoff_longit
ude)] = median(Train$dropoff_longitude, na.rm = T)
Train$dropoff_latitude[is.na(Train$dropoff_latitud
e)] = median(Train$dropoff_latitude, na.rm = T)
```

```
Train$passenger_count[is.na(Train$passenger_count)
] = median(Train$passenger_count, na.rm = T)

#Check if any missing values
sum(is.na(Train))


############### Outlier Analysis
######################

# Boxplot for continuous variables

boxplot(Train$fare_amount, xlab = "Fare", ylab =
"count")
boxplot(Train$pickup_longitude, xlab =
"Pickup_Longitude", ylab = "count")
boxplot(Train$pickup_latitude, xlab =
"Pickup_Latitude", ylab = "count")
boxplot(Train$dropoff_latitude, xlab =
"Dropoff_Latitude", ylab = "count")
boxplot(Train$dropoff_longitude, xlab =
"Dropoff_Longitude", ylab = "count")
boxplot(Train$passenger_count, xlab =
"Passenger_Count", ylab = "count")


##### Pickup time
#converting factor type pickup time to a character
first
Train$pickup_datetime =
as.character(Train$pickup_datetime)
#converting into datetime
Train$pickup_datetime <-
```

```
  as.POSIXct(Train$pickup_datetime, format = "%Y-
%m-%d %H:%M:%S", tz = "")

Train$Hours <-
  as.numeric(format(as.POSIXct(
    strptime(Train$pickup_datetime,       "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%H"))
Train$Minutes <-
  as.numeric(format(as.POSIXct(
    strptime(Train$pickup_datetime,       "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%M"))
Train$Date <-
  as.numeric(format(as.POSIXct(
    strptime(Train$pickup_datetime,       "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%d"))
Train$Month <-
  as.numeric(format(as.POSIXct(
    strptime(Train$pickup_datetime,       "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%m"))
Train$Year <-
  as.numeric(format(as.POSIXct(
    strptime(Train$pickup_datetime,       "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%Y"))

Train = na.omit(Train)

#converting Test data into datetime

Test$pickup_datetime <-
```

```
  as.POSIXct(Test$pickup_datetime, format = "%Y-%m-
%d %H:%M:%S", tz = "")

Test$Hours <-
  as.numeric(format(as.POSIXct(
    strptime(Test$pickup_datetime,        "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%H"))
Test$Minutes <-
  as.numeric(format(as.POSIXct(
    strptime(Test$pickup_datetime,        "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%M"))
Test$Date <-
  as.numeric(format(as.POSIXct(
    strptime(Test$pickup_datetime,        "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%d"))
Test$Month <-
  as.numeric(format(as.POSIXct(
    strptime(Test$pickup_datetime,        "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%m"))
Test$Year <-
  as.numeric(format(as.POSIXct(
    strptime(Test$pickup_datetime,        "%Y-%m-%d
%H:%M:%S", tz = "")
  ) , format = "%Y"))

Test = na.omit(Test)


#### Passenger Count should be less than 8 and
greater than 0. Also remove 0.12 in train data
```

```
Train = subset(Train, passenger_count < 8 &
passenger_count > 0)
Train = subset(Train, passenger_count != 0.12)

Test = subset(Test, passenger_count < 8 &
passenger_count > 0)

#### Fare Amount should be greater than 0.

Train = subset(Train, fare_amount > 0)

#### Pickup Longitude and latitude

install.packages('geosphere')
library(geosphere)

Train = subset(Train, (pickup_longitude < 180 &
                         pickup_longitude > -180))
Train = subset(Train, (pickup_latitude < 90 &
                         pickup_latitude > -90))
Train = subset(Train, (dropoff_latitude < 90 &
                         dropoff_latitude > -90))
Train = subset(Train, (dropoff_longitude < 180 &
                         dropoff_longitude > -
180))

Test = subset(Test, (pickup_longitude < 180 &
                       pickup_longitude > -180))
Test = subset(Test, (pickup_latitude < 90 &
pickup_latitude > -90))
Test = subset(Test, (dropoff_latitude < 90 &
                       dropoff_latitude > -90))
```

```
Test = subset(Test, (dropoff_longitude < 180 &
                       dropoff_longitude > -180))


#Using Haversine Function of Geosphere package
Train$Distance <-
  distHaversine(
    cbind(Train$pickup_longitude,
Train$pickup_latitude),
    cbind(Train$dropoff_longitude,
Train$dropoff_latitude)
  )
Test$Distance  <-
  distHaversine(
    cbind(Test$pickup_longitude,
Test$pickup_latitude),
    cbind(Test$dropoff_longitude,
Test$dropoff_latitude)
  )

#Hence removing these values (0.00000,0.00000)

Train = subset(Train,
               (pickup_longitude   !=   0.00000   &
pickup_longitude != 0.00000))
Train = subset(Train,
               (dropoff_longitude   !=   0.00000   &
dropoff_longitude != 0.00000))

Train[order(-Train$Distance), ]

Test = subset(Test, (pickup_longitude != 0.00000 |
                       pickup_longitude          !=
0.00000))
```

```
Test = subset(Test,
              (dropoff_longitude   !=   0.00000   |
dropoff_longitude != 0.00000))


Test[order(-Test$Distance), ]

Train = subset(Train, Distance < 4452067)



############### Visualizations ##############
df_columns = colnames(Train)

df_columns = c(
  'fare_amount',
  'passenger_count',
  'Hours',
  'Minutes',
  'Date',
  'Month',
  'Year',
  'Distance'
)

for (i in df_columns) {
  hist(
    Train[, i],
    col = "Blue",
    prob = TRUE,
    main = "Histogram",
    xlab = i,
    ylab = "Count"
```

```
  )
  lines(density(Train[, i]), lwd = 2)
}


############ Normalising data ##############


for (i in df_columns)
{
  print(i)
  Train[, i] = (Train[, i] - min(Train[, i])) /
(max(Train[, i]) - min(Train[, i]))
}


#####################       Model       Development
#########################


#Divide data into train and test using stratified
sampling method
set.seed(123)
train.index   =   sample(1:nrow(Train),   0.8   *
nrow(Train))
train = Train[train.index, ]
test  = Train[-train.index, ]

train = train[, -2]
test = test[, -2]

##Decision tree for classification
#Develop Model on training data
```

```
library(rpart)
df_dtree = rpart(fare_amount ~ ., data = train,
method = "anova")

dev.new()
plot(df_dtree)
text(df_dtree)
summary(df_dtree)
printcp(df_dtree)


#write rules into disk
write(capture.output(summary(df_dtree)),
"Rules.txt")

#Lets predict for test data
pred_test_DT = predict(df_dtree, test[, -1])

#Lets predict for train data
pred_train_DT = predict(df_dtree, train[, -1])


install.packages("caret")
library(caret)


# For training data
print(postResample(pred = pred_train_DT, obs =
train[, 1]))
#      RMSE  Rsquared       MAE
#6.779453e-04 1.627864e-01 8.867387e-05
```

```
# For testing data
print(postResample(pred = pred_test_DT, obs =
test[, 1]))
#      RMSE  Rsquared       MAE
#1.786968e-02 4.505198e-07 4.064647e-04

#############Linear Regression
install.packages("caret")
library(caret)

#check multicollearity
install.packages("usdm")
library(usdm)
vif(Train[, -1:-2])

vifcor(Train[, -1:-2], th = 0.9)

#run regression model
LR_model = lm(fare_amount ~ ., data = train)

#Summary of the model
summary(LR_model)


#Lets predict for test data
pred_test_LR = predict(LR_model, test[, -1])

#Lets predict for train data
pred_train_LR = predict(LR_model, train[, -1])

#Predict
predictions_LR = predict(LR_model, test[, 2:12])
```

```
# For training data
print(postResample(pred = pred_train_LR, obs = train[, 1]))
#RMSE     Rsquared          MAE
#7.305286e-04 2.787657e-02 7.222486e-05

# For testing data
print(postResample(pred = pred_test_LR, obs = test[, 1]))
#RMSE     Rsquared          MAE
#0.0178638895 0.0003458237 0.0003830841



#------------------RANDOM FOREST-----------------

install.packages("randomForest")
library(randomForest)
#RUN RANDOM FOREST
RF_model = randomForest(fare_amount ~ ., train, importance = TRUE, ntree = 300)

#Summary of the model
summary(RF_model)


#Lets predict for test data
pred_test_RF = predict(RF_model, test[, -1])

#Lets predict for train data
pred_train_RF = predict(RF_model, train[, -1])

#Predict
```

```
predictions_RF = predict(RF_model, test[, 2:12])

# For training data
print(postResample(pred  =  pred_train_RF,  obs  =
train[, 1]))
#         RMSE  Rsquared       MAE
#3.368938e-04 9.494762e-01 2.335838e-05

# For testing data
print(postResample(pred  =  pred_test_RF,  obs  =
test[, 1]))
#     RMSE  Rsquared       MAE
#0178634671 0.0002464168 0.0003647339




#--------------------------------------Prdicting
using Random forest--------------

predictions_RF = predict(RF_model, Test[, 2:12])
```