CAB FARE PREDICTION

Shridhar S

11 September 2019

INDEX

Chapter 1

Introduction

1.1 Problem Statement :

Predicting a Cab fare for any trip before the users book is a crucial step for any cab rental service. It gives much clarity and cost estimation to the users about the trip they are going to take. We are a Start-up Cab company. Having successfully run our Pilot project across the country, we have the same data at our disposal using which, we need to come up with a model to predict cab fare for each trip

1.2 Data

Our aim is to build a model which will predict the cab fare for each trip. We have a sample dataset given below to have an idea of how our dataset looks and what are the variables that help us in predicting the cab fare.

Table 1.1 : Cab fare prediction sample 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
16.5	2012-01-04 17:22:00 UTC	-73.9513	40.774138	-73.990095	40.751048	1

As we can see from the sample data, we have a total of 7 variables in the dataset.

Table 1.2 Unique Variables

1	fare_amount
2	pickup_datetime
3	pickup_longitude
4	pickup_latitude
5	dropoff_longitude
6	dropoff_latitude
7	passenger_count

Following is the explaination of the variables.

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.

Chapter 2

Methodology

2.1 Pre – Processing

The data we receive would not be clean or in-line with what we need for analysis. In order to make the data best for visualization, graphs and plots, we need to clean the data and make necessary changes. This is called Data Pre-Processing and it is one of the most important steps in Data Science. We have to check for missing values, outliers and treat them to keep the sanity of the data. Also we need to extract new features out of given data and see how they can help us in predicting the dependent variables.

2.2 Missing Value Analysis

We see that we have 25 missing values(NA's) in fare_amount and 55 missing values in passenger_count. Usually we deal with missing values by substituting them with mean, median or nearest value. Since in our analysis, fare_amount is the dependent variable and also the missing values are not much in number, we will delete the rows with missing values instead.

2.3 Feature Engineering

a) We can extract some features like year, month, day, hour, date out of the datestamp provided in the pickup_datetime variable given in the data. This gives us some valuable inputs as to how the fare_amount depends on different part of the day, different months of the year, on weekdays and weekends etc.

b) We can also extract distance between two geocodes using Haversine formula.

Haversine formula to find distance between two points on a sphere

The **Haversine** formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface. It is important for use in navigation. The haversine can be expressed in trignometric function as:

$$haversine(\theta) = sin^2\left(\frac{\theta}{2}\right)$$

The haversine of the central angle (which is d/r) is calculated by the following formula:

$$\left(\frac{d}{r}\right) = haversine(\Phi_2 - \Phi_1) + cos(\Phi_1)cos(\Phi_2)haversine(\lambda_2 - \lambda_1)$$

where r is the radius of earth(6371 km), d is the distance between two points, ϕ_1, ϕ_2 is latitude of the two points and λ_1, λ_2 is longitude of the two points respectively.

Solving d by applying the inverse haversine or by using the inverse sine function, we get:

$$d = rhav^{-1}(h) = 2rsin^{-1}(\sqrt{h})$$

or

$$d = 2rsin^{-1}\left(\sqrt{sin^2\left(\frac{\Phi_2-\Phi_1}{2}\right) + cos(\Phi_1)cos(\Phi_2)sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$$

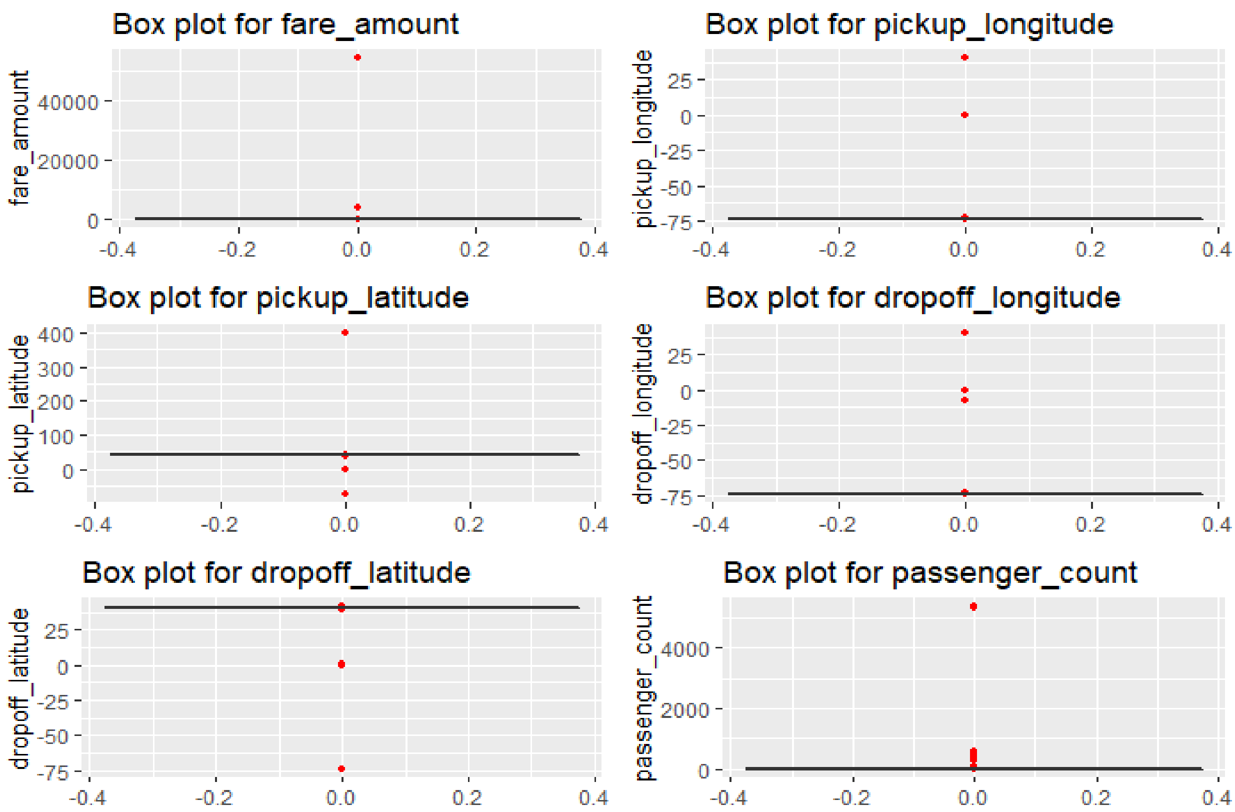
2.4 Outlier Analysis

Outliers are those values which lie on the outside of the given range. These extreme values deviate us from the actual observations and the actual pattern of the observations. It's important to treat the outliers as this affects our prediction adversely in the prediction stages. Outliers can be easily identified with the help of statistical methods and graphs(Boxplots). Boxplots is one such graphs which contains 3 quantiles(Q1, Median, Q3). Q1 is the 25th percentile of the variable and Q3 is the 75th percentile. The difference between Q3 and Q1 is called the Inter Quantile Range or IQR. So the value which is outside of Q3 by 1.5 times the IQR is called the Upper limit. Values which are outside of Q1 by 1.5 times the IQR is called Lower limit.

Outliers for continuous variables for using statistical mean, median, max and min.

```
> summary(numeric_data)
  fare_amount       pickup_longitude pickup_latitude  dropoff_longitude dropoff_latitude passenger_count
 Min.   :   -3.00   Min.   :-74.44   Min.   :-74.01   Min.   :-74.43    Min.   :-74.01   Min.   :   0.000
 1st Qu.:    6.00   1st Qu.:-73.99   1st Qu.: 40.73   1st Qu.:-73.99    1st Qu.: 40.73   1st Qu.:   1.000
 Median :    8.50   Median :-73.98   Median : 40.75   Median :-73.98    Median : 40.75   Median :   1.000
 Mean   :   15.03   Mean   :-72.46   Mean   : 39.92   Mean   :-72.46    Mean   : 39.90   Mean   :   2.623
 3rd Qu.:   12.50   3rd Qu.:-73.97   3rd Qu.: 40.77   3rd Qu.:-73.96    3rd Qu.: 40.77   3rd Qu.:   2.000
 Max.   :54343.00   Max.   : 40.77   Max.   :401.08   Max.   : 40.80    Max.   : 41.37   Max.   :5345.000
> 
```

Outlier for the continuous variables in the data using boxplot.



Once the values which are Once the outliers are identified, we can treat them by replacing them with suitable values such as upper limit or lower limit of the variables.

2.5 Distribution of Continuous Variables

From the below graphs we can see the distribution of continuous variables like fare_amount, pickup_longitude, pickup_latitude etc.

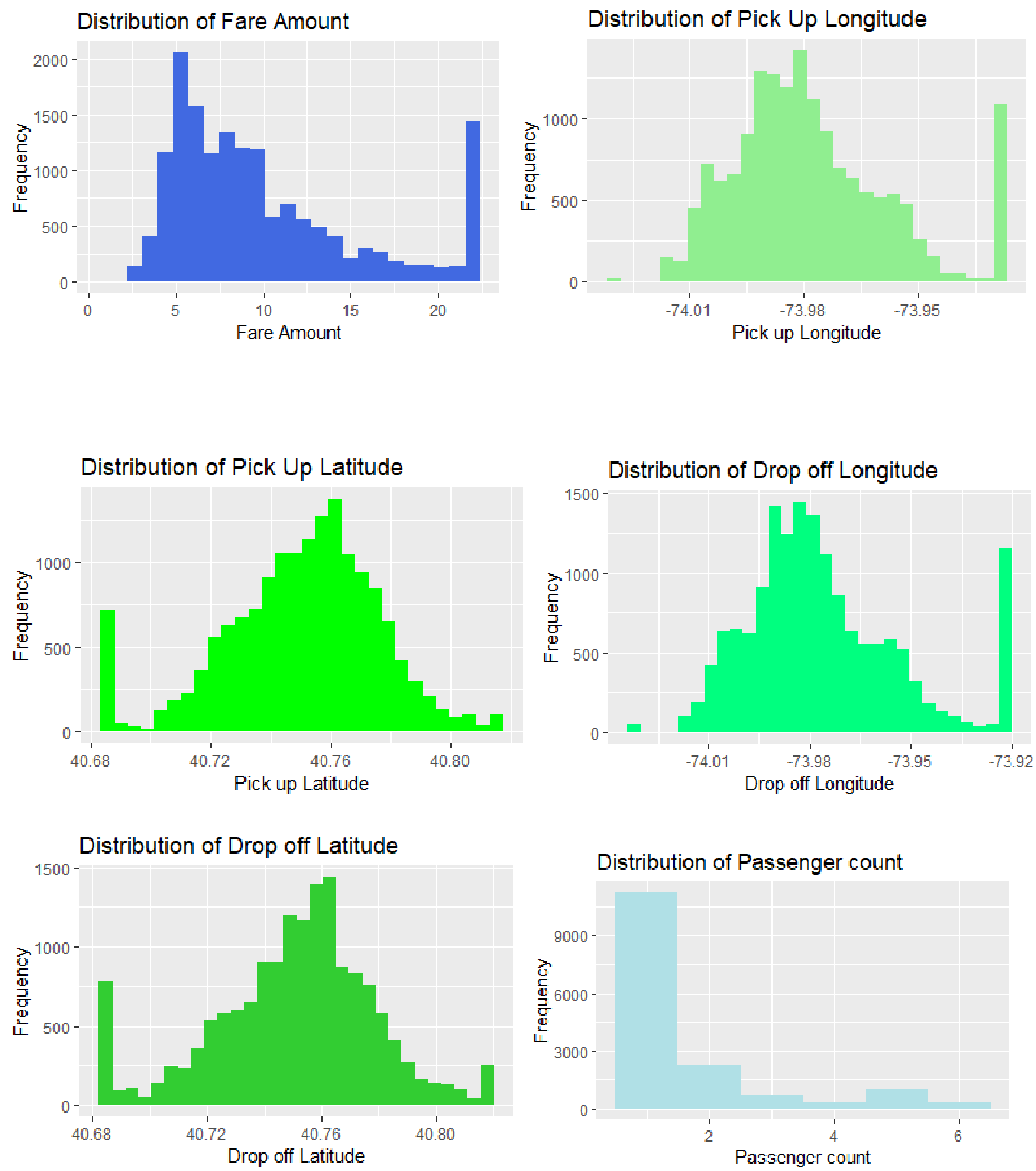
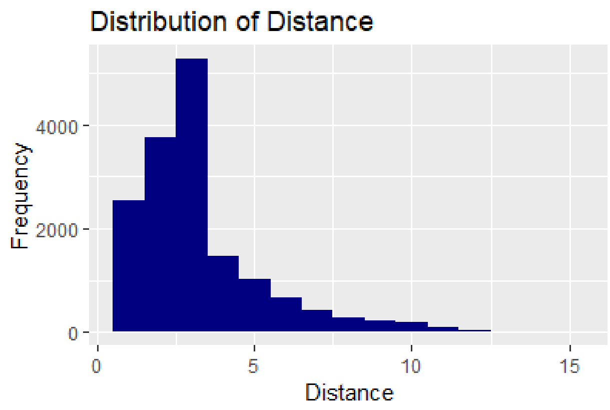
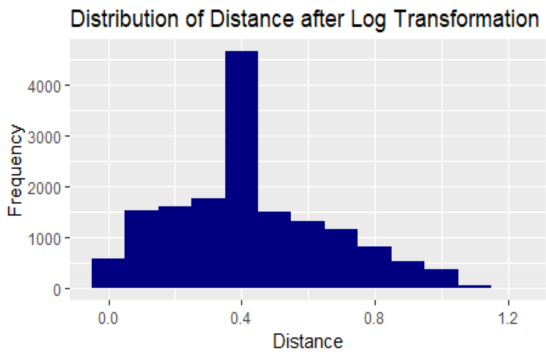


Fig. 2.1 : Categorical variables vs count

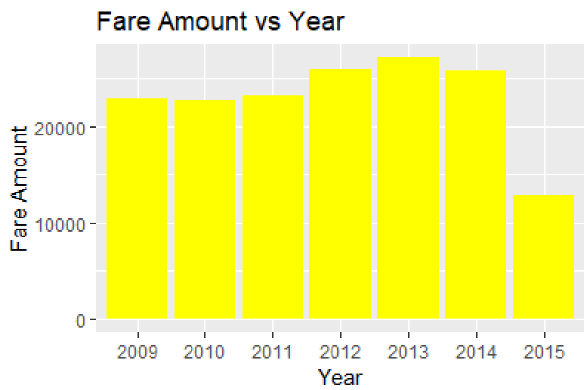


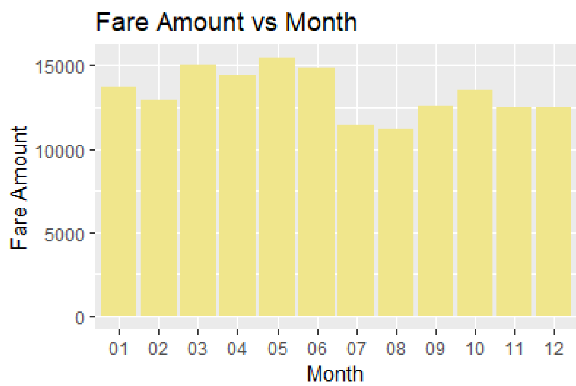


We can see the frequency distribution of all the continuous variables in the above graph. We observe that all the variables are normally distributed(more or less) and that is as expected except for the distance. We can see that Distance is right skewed and this skewness can be removed by log transformation. Also the Passenger count distribution need not be a normal distribution as there could be a possibility that a single passengers took most of the trips compared to the other numbers

2.6 Distribution of Categorical Variables

In the below graph we can see the distribution of continuous variables such as year, month, day, date, hour.





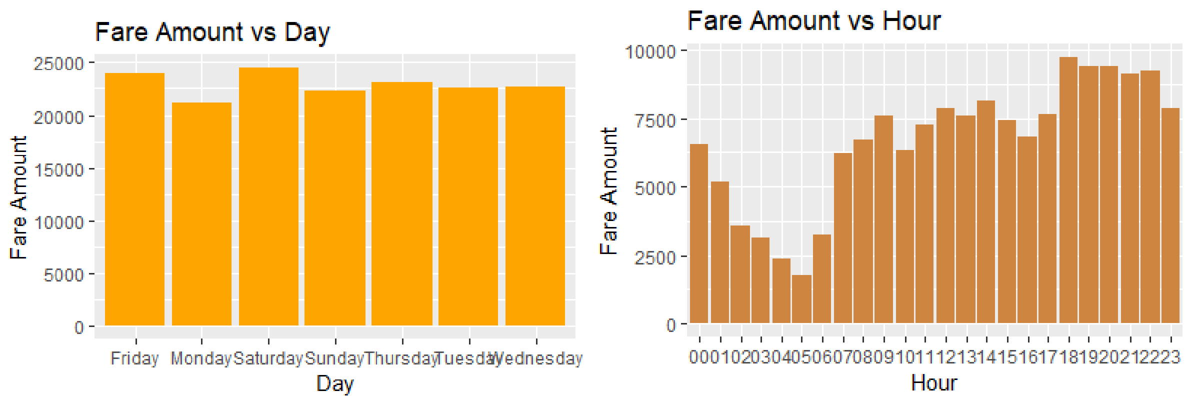


Fig. 2.2 : Categorical variables vs fare amount

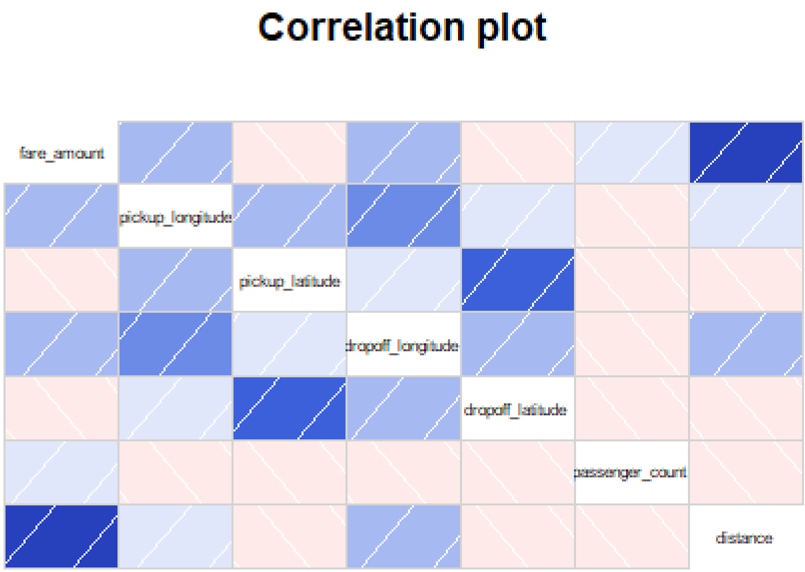
We see that the total fare amount increased over the years, however by 2014-15 it reduced. We can see less fare amount in July and August. Fare amount remained same throughout the week. When it comes to the hours, we can see lesser fare amount in the early morning and good number of booking after the evening.

2.7 Feature Selection

As we looked the dependency of dependent variable on other variable, we could see what are the variables which are affecting the number of counts. It is important for us to select the variables which are highly related to the dependent variable and reduce any error that we may have. This can be done using the Correlation plot or Correlograms.

```
[1] "date"
               Df Sum Sq Mean Sq F value Pr(>F)
train[, i]     30    797   26.56    0.89  0.639
Residuals   15892 474127   29.83
[1] "year"
               Df Sum Sq Mean Sq F value Pr(>F)
train[, i]      6  10776  1796.0   61.59 <2e-16 ***
Residuals   15916 464147    29.2
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
[1] "month"
               Df Sum Sq Mean Sq F value   Pr(>F)
train[, i]     11   1625  147.70   4.965 9.41e-08 ***
Residuals   15911 473299   29.75
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
[1] "day"
               Df Sum Sq Mean Sq F value Pr(>F)
train[, i]      6    299   49.80    1.67  0.124
Residuals   15916 474625   29.82
[1] "hour"
               Df Sum Sq Mean Sq F value   Pr(>F)
train[, i]     23   3176  138.10   4.654 9.38e-13 ***
Residuals   15899 471747   29.67
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

After running ANOVA(Analysis of Variance) for the categorical variables, we can see p values as less than 0.05 for day and the date variables. We can remove these variables.



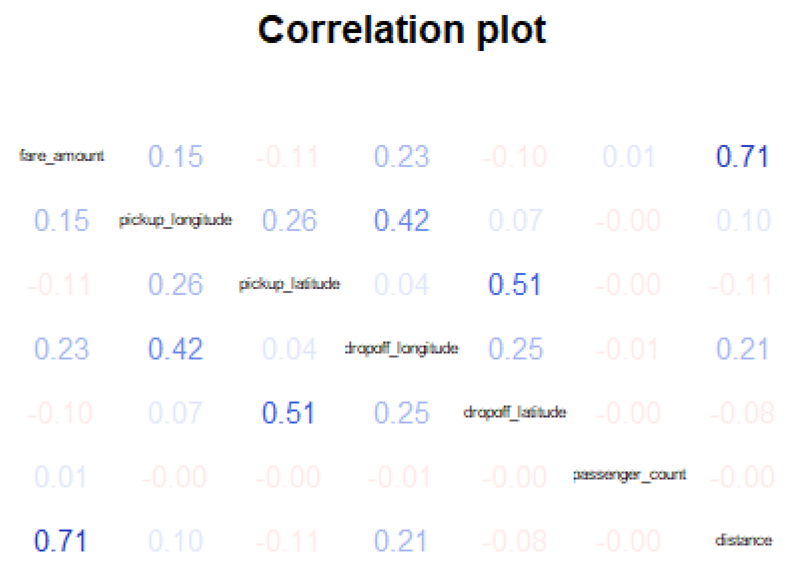


Fig. 2.3 : Correlation plot

Chapter 3

Modelling

3.1 Model Selection

Model Selection is where we select the suitable modelling technique based on the type of dependent variable. Since the cab fare is a continuous variable we are going with Linear Regression and Random Forest(Classification) technique.

3.2 Multiple Linear Regression

In linear regression, the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.

We can see that the Adjusted R squared is 54.6% which means we can explain 54.6% of the data using our model. F-statistic is 334 and p-value is 2.2e-16 which can reject the null hypothesis that target variable does not depend on any of the predictor variables.

The MAPE is 38.7% and F-statistic is 334. Hence the accuracy of the model is 61.3%. This means our model is not good.

```
Residual standard error: 3.686 on 12691 degrees of freedom
Multiple R-squared:  0.548,     Adjusted R-squared:  0.5463
F-statistic: 334.5 on 46 and 12691 DF,  p-value: < 2.2e-16
```

3.3 Random Forest

Apart from Regression, we shall use one Classification model for the predictions. Number of trees used in this case is 500. MAPE for the model is 20.55%. Hence the accuracy is 79.45%.

Chapter 4

Conclusion

4.1 Model Evaluation

We have 2 models for predicting the cab fare amount. Now, we need to decide on which one to choose.

We can compare the models using any of the following criteria :

1. Predictive Performance

2. Interpretability

3. Computational Efficiency

For our case, criteria 2 and 3 does not hold good. Hence we go with Predictive Performance.

Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

4.2 Error Metrics

1) Mean Absolute Percentage Error (MAPE):

MAPE measures the average magnitude of errors in a set of predictions without considering directions. It’s an percentage average of difference between predicted and actual values .

2) Root Mean Square Error (RMSE):

RMSE is more of a magnitude of error calculation metrics. It’s the square root of the average of squared difference between predicted and actual values.

Even though MAPE and RMSE both express average model prediction errors, RMSE is squared before it is averaged, it gives more weight to large errors compared to MAPE. Since these errors are negatively oriented(lower the better), it's good to consider MAPE for our error metrics.

4.3 Mean Absolute Percentage Error(MAPE)

MAPE can be calculated by the below formula and MAPE for Linear Regression is 38.7% and For Random Forest it is 20.55%.

MAPE = ((Actual_value – Predicted Value)/Actual Value)*100

4.3 Model Selection

As we have seen that from the below table, Random Forest model has better Accuracy and less MAPE. So we can choose Random Forest

Model	**MAPE**	**Accuracy**
Linear Regression - Model1	38.7%	61.3%
Random Forest - Model 2	20.55%	79.45%

Table 4.1 Model Accuracy

Chapter 5

R- Code

```
#################################### CAB FARE PREDICTION
################################################



#Clean the environment

rm(list = ls(all=T))


#Set Working Directory

setwd("C:/Users/shrid/Downloads/edWisor/Project 2")

getwd()


#Load Libraries

library("readr")

library("plyr")

library("dplyr")

library("DMwR")

library("ggplot2")

library("corrgram")

library("rlist")

library("purrr")

library("geosphere")

library("car")

library("randomForest")
```

###LOAD THE DATA###

```
#Load the training data and have a glimpse of the data
train = read.csv(file = "train_cab.csv", header = TRUE, sep = ",", na.strings = c("", " ", "NA"))
head(train)
class(train)
names(train)
str(train) # Checking for structure of the data
summary(train)

#Convert variables into appropriate datatypes
train$fare_amount <- as.numeric(as.character(train$fare_amount))
typeof(train$fare_amount)

summary(train)
```

###DATA PRE-PROCESSING###

```
#Missing value Ananlysis

#Let's check for missing values
missing_values = sapply(train, function(x){sum(is.na(x))})
missing_values
#We have 25 missing values in fare_amount and 55 in passenger_count
```

```
#Let's check it in terms of percentage

missing_values_percentage = (missing_values*100/nrow(train))

missing_values_percentage

#less than 30% so no need to delete any dependent variables. Let's just delete rows for simplicity


#Let's drop the rows with missing values for simplicity

train <- na.omit(train)

train

str(train)


missing_values = sapply(train, function(x){sum(is.na(x))})

missing_values

#We do not see any missing values after clean up



#Feature Engineering


#Let's draw features out of datetime stamp


train$pickup_datetime <- gsub(' UTC', '', train$pickup_datetime)

train$pickup_datetime


train$date <- as.Date(train$pickup_datetime)

train$date
```

```
train$year <- substr(train$pickup_datetime,1,4)
train$year

train$month <- substr(train$pickup_datetime,6,7)
train$month

train$day <- weekdays(as.POSIXct(train$date), abbreviate = F)
train$day

train$date <- substr(train$date, 9, 10)
train$date

train$hour <- substr(train$pickup_datetime, 12, 13)
train$hour

class(train$month)

#Let's check for missing values after Feature Engineering

missing_values = sapply(train, function(x){sum(is.na(x))})
missing_values

#There is one Missing row, let's remove that
train = na.omit(train)

missing_values = sapply(train, function(x){sum(is.na(x))})
```

missing_values

#Outlier Analysis

#Outlier Ananlysis for continous variables

numeric_data <- subset(train, select = c(fare_amount,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude,passenger_count))

cnames <- colnames(numeric_data)

cnames

summary(numeric_data)

#Boxplot of all numeric variables

```
for(i in 1:length(cnames))
  {
  assign(paste0("b",i), ggplot(aes_string(y = cnames[i]), data = train) +
    stat_boxplot(geom = "errorbar", width = 0.5) +
    geom_boxplot(outlier.color = "red", fill = "green", outlier.shape = 16, outlier.size = 1, notch = FALSE)+
      theme(legend.position = "bottom") + labs(cnames[i])+
      ggtitle(paste("Box plot for", cnames[i]))
    )
}
```

gridExtra::grid.arrange(b1,b2,b3,b4,b5,b6,ncol=2)

```
#Let's treat the Outliers

#Fare amount
summary(train$fare_amount)

Q1 <- quantile(train$fare_amount, 0.25)
Q1#Quantile 1
Q3 <- quantile(train$fare_amount, 0.75)
Q3#Quantile 3
UL <- Q3 + (1.5*IQR(train$fare_amount))
UL#Upper Limit
LL <- Q1 - (1.5*IQR(train$fare_amount))
LL#Lower limit

#Let's replace the values outside of range with Lower Limit and Upper Limit
train[train$fare_amount < LL, "fare_amount"] <- LL
train[train$fare_amount > UL, "fare_amount"] <- UL

summary(train$fare_amount)

#Since the fare can not be in negative and zero, let's replace those values with NA
train$fare_amount[train$fare_amount <= 0.01] <- NA

#Let's check the NA values and remove them
sum(is.na(train$fare_amount))
```

```
train <- na.omit(train)


#Pick up longitude
summary(train$pickup_longitude)


Q1 <- quantile(train$pickup_longitude, 0.25)
Q1
Q3 <- quantile(train$pickup_longitude, 0.75)
Q3
UL <- Q3 + (1.5*IQR(train$pickup_longitude))
UL
LL <- Q1 - (1.5*IQR(train$pickup_longitude))
LL


train[train$pickup_longitude < LL, "pickup_longitude"] <- LL
train[train$pickup_longitude > UL, "pickup_longitude"] <- UL


summary(train$pickup_longitude)


#Pick up lattitude
summary(train$pickup_latitude)


Q1 <- quantile(train$pickup_latitude, 0.25)
Q1
Q3 <- quantile(train$pickup_latitude, 0.75)
Q3
```

```
UL <- Q3 + (1.5*IQR(train$pickup_latitude))

UL

LL <- Q1 - (1.5*IQR(train$pickup_latitude))

LL


train[train$pickup_latitude < LL, "pickup_latitude"] <- LL

train[train$pickup_latitude > UL, "pickup_latitude"] <- UL


summary(train$pickup_latitude)


#Drop off Longitude


summary(train$dropoff_longitude)


Q1 <- quantile(train$dropoff_longitude, 0.25)

Q1

Q3 <- quantile(train$dropoff_longitude, 0.75)

Q3

UL <- Q3 + (1.5*IQR(train$dropoff_longitude))

UL

LL <- Q1 - (1.5*IQR(train$dropoff_longitude))

LL


train[train$dropoff_longitude < LL, "dropoff_longitude"] <- LL

train[train$dropoff_longitude > UL, "dropoff_longitude"] <- UL
```

summary(train$dropoff_longitude)

#Drop off latitude

summary(train$dropoff_latitude)

Q1 <- quantile(train$dropoff_latitude, 0.25)

Q1

Q3 <- quantile(train$dropoff_latitude, 0.75)

Q3

UL <- Q3 + (1.5*IQR(train$dropoff_latitude))

UL

LL <- Q1 - (1.5*IQR(train$dropoff_latitude))

LL

train[train$dropoff_latitude < LL, "dropoff_latitude"] <- LL

train[train$dropoff_latitude > UL, "dropoff_latitude"] <- UL

summary(train$dropoff_latitude)

#Passenger count

summary(train$passenger_count)

Q1 <- quantile(train$passenger_count, 0.25)

Q1

```
Q3 <- quantile(train$passenger_count, 0.75)

Q3

UL <- Q3 + (1.5*IQR(train$passenger_count))

UL

LL <- Q1 - (1.5*IQR(train$passenger_count))

LL


train[train$passenger_count < 0, "passenger_count"] <- 0

train[train$passenger_count > 6, "passenger_count"] <- 6 #As there can be a max of 6 passengers


#Since there should be at least 1 passenger for the trip to count, let's replace all the values less than 1 with NA

train$passenger_count[train$passenger_count < 1] <- NA


sum(is.na(train$passenger_count))


#Let's remove the missing values

train = na.omit(train)


sum(is.na(train$passenger_count))


#Let's visualize the boxplot after removing outliers


for(i in 1:length(cnames))

{
```

```
  assign(paste0("b",i), ggplot(aes_string(y = cnames[i]), data = train) +
           stat_boxplot(geom = "errorbar", width = 0.5) +
           geom_boxplot(outlier.color = "red", fill = "grey", outlier.shape = 16, outlier.size = 1,
notch = FALSE)+
           theme(legend.position = "bottom") + labs(y=cnames[i])+
           ggtitle(paste("Box plot for", cnames[i]))
  )
}


gridExtra::grid.arrange(b1,b2,b3,b4,b5,b6,ncol=3)



#Let's calculate distance between 2 geo codes


distance1 = function(long1, lat1, long2, lat2){
  loadNamespace("purrr")
  loadNamespace("geosphere")

  l1 = purrr::map2(long1, lat1, function(x,y) c(x,y))
  l2 = purrr::map2(long2, lat2, function(x,y) c(x,y))

  dist = purrr::map2(l1, l2, function(x,y) geosphere::distHaversine(x, y))
  distance_m = list.extract(dist, position = 1)

  distance = distance_m / 1000.0 ;#in km
```

```
  distance
}

for( i in (1:nrow(train)))
{
  train$distance[i] = distance1(train$pickup_longitude[i], train$pickup_latitude[i], train$dropoff_longitude[i], train$dropoff_latitude[i])
}

head(train)
str(train)


# Outliers for Distance

bp_dist = ggplot(aes_string( x = "distance", y = "fare_amount"), data = train) +
  stat_boxplot(geom = "errorbar", width = 0.25) +
  geom_boxplot(outlier.color = "red", fill = "green", outlier.shape = 16, outlier.size = 1, notch = FALSE) +
  ggtitle(paste("Boxplot for Distance with Outliers"))

bp_dist

summary(train$distance)

train$distance[train$distance < 1] <- mean(train$distance)
```

```
summary(train$distance)



# Exploratory data analysis

# Frequency distribution of Numeric variables

#Fare amount
a1 <- ggplot(data = train, aes( x = train$fare_amount)) + geom_histogram(fill = "royalblue", bins = 20) +
  labs(x = "Fare Amount", y = "Frequency") + ggtitle("Distribution of Fare Amount")
a1

#Pick up Longitude
a2 <- ggplot(data = train, aes( x = train$pickup_longitude)) + geom_histogram( fill = "lightgreen") +
  labs(x = "Pick up Longitude", y = "Frequency") + ggtitle("Distribution of Pick Up Longitude")
a2

#Pick up Latitude
a3 <- ggplot(data = train, aes( x = train$pickup_latitude)) + geom_histogram(fill = "green") +
  labs(x = "Pick up Latitude", y = "Frequency") + ggtitle("Distribution of Pick Up Latitude")
a3

#Drop off Longitude
```

```
a4 <- ggplot(data = train, aes( x = train$dropoff_longitude)) + geom_histogram( fill =
"springgreen") +
  labs(x = "Drop off Longitude", y = "Frequency") + ggtitle("Distribution of Drop off
Longitude")
a4

#Drop off Latitude
a5 <- ggplot(data = train, aes( x = train$dropoff_latitude)) + geom_histogram( fill =
"limegreen") +
  labs(x = "Drop off Latitude", y = "Frequency") + ggtitle("Distribution of Drop off Latitude")
a5

#Passenger Count
a6 <- ggplot(data = train, aes( x = train$passenger_count)) + geom_histogram(binwidth = 1,
fill = "powderblue") +
  labs(x = "Passenger count", y = "Frequency") + ggtitle("Distribution of Passenger count")
a6

#Distance
a7 <- ggplot(data = train, aes( x = train$distance)) + geom_histogram(binwidth = 1, fill =
"navy") +
  labs(x = "Distance", y = "Frequency") + ggtitle("Distribution of Distance")
a7#distance is right skewed. Let's remove skewness by applying log

logof10 = function(x) {
  ifelse(abs(x) <= 1, 0, sign(x)*log10(abs(x)))
}
```

```
train$distance = logof10(train$distance)

#Distance after Log transformation
a7 <- ggplot(data = train, aes( x = train$distance)) + geom_histogram(binwidth = 0.1, fill = "navy") +
  labs(x = "Distance", y = "Frequency") + ggtitle("Distribution of Distance after Log Transformation")
a7

#Categorical variables

#Fare amount vs Year
b1 <- ggplot(train, aes(train$year, train$fare_amount)) + geom_bar(stat = "identity", fill = "yellow") +
  labs( x = "Year", y = "Fare Amount") + ggtitle("Fare Amount vs Year")
b1

#Fare Amount vs Month
b2 <- ggplot(train, aes(train$month, train$fare_amount)) + geom_bar(stat = "identity", fill = "khaki") +
  labs( x = "Month", y = "Fare Amount") + ggtitle("Fare Amount vs Month")
b2

#Fare Amount vs Day
```

```
b3 <- ggplot(train, aes(train$day, train$fare_amount)) + geom_bar(stat = "identity", fill =
"orange") +
  labs( x = "Day", y = "Fare Amount") + ggtitle("Fare Amount vs Day")
b3

#Fare Amount vs Hour

b4 <- ggplot(train, aes(train$hour, train$fare_amount)) + geom_bar(stat = "identity", fill =
"peru") +
  labs( x = "Hour", y = "Fare Amount") + ggtitle("Fare Amount vs Hour")
b4


#Correlation plot

c1 <- corrgram(train, order = FALSE, main = "Correlation plot")
c1

c2 <- corrgram(train, order = FALSE, main = "Correlation plot", panel = panel.cor)
c2


#ANOVA for categorical variables

categorical <- c("date", "year", "month", "day", "hour")
```

```
for(i in categorical){

  print(i)

  anova = summary(aov(fare_amount~train[,i],train))

  print(anova)

}

#p > 0.05 for day and date. So let's remove those variables


names(train)


data1 <- subset(train, select = c(fare_amount,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude,passenger_count,year,month,hour,distance))

data1


#Let's create train and test dataset


index = sample(1:nrow(data1), as.integer(0.8*nrow(data1)) )


train_data = data1[index,]

test_data = data1[-index,]


#Linear regression

model1 = lm(fare_amount~., train_data)

summary(model1)


par(mfrow = c(2,2))
```

```
plot(model1)

vif(model1) #less than 5
dwt(model1)
prediction1 <- predict(model1, test_data[,-1])

df = data.frame("actual" = test_data[,1], "pred" = prediction1)
df

MAPE = function(y, yhat){
  mean(abs((y-yhat)/y))
}

MAPE(test_data[,1],prediction1)

#Random Forest

model2 <- randomForest(fare_amount~., train_data, tree = 500)

summary(model2)

prediction2 <- predict(model2, test_data[-1])

df = cbind(df, prediction2)
df
```

```
MAPE(test_data[,1],prediction2)

#Model Evaluation

test <- read.csv(file = "test.csv", header = TRUE, sep = ",", na.strings = c("", " ", "NA"))

head(test)
str(test)
summary(test)

#Feature engineering

test$pickup_datetime <- gsub(" UTC", "", test$pickup_datetime)

test$date <- as.Date(test$pickup_datetime)

test$year <- substr(as.character(test$pickup_datetime),1,4)

test$month <- substr(as.character(test$pickup_datetime),6,7)

test$day <- weekdays.POSIXt(test$date, abbreviate = FALSE)

test$date <- substr(as.character(test$date),9,10)
```

```
test$hour <- substr(as.factor(test$pickup_datetime), 12, 13)

str(test)

# Data preprocessing

#Missing value
sum(is.na(test))

#Outlier
summary(test)

#distance

for( i in (1:nrow(test)))
{
  test$distance[i] = distance1(test$pickup_longitude[i], test$pickup_latitude[i], test$dropoff_longitude[i], test$dropoff_latitude[i])
}

head(test)
summary(test$distance)

test$distance[test$distance < 1] = mean(test$distance)

#Distribution of distance
```

```
z <- ggplot(test, aes_string(x = test$distance)) + geom_histogram() + geom_density()

z

test$distance <- logof10(test$distance)

summary(test$distance)

#Modelling

Final <- subset(test, select =
c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude,passenger_count,year,
month,hour,distance   ))

Final_model <- randomForest(fare_amount~. , data1, ntree = 500, method = "anova")

predictions <- predict(Final_model, Final)

predictions

test$fare_amount <- predictions

summary(test)

summary(test$fare_amount)

head(test)
```

Chapter 6

Python Code

```
###############################################CAB FARE PREDICTION
###################################################


#Import the library
import pandas as pd
import os
import numpy as np
from ggplot import *
import seaborn as sns


#set the Working Directory
os.chdir("C:¥¥Users¥shrid¥Downloads¥edWisor¥Project 2")
os.getcwd()


#Load the data
train = pd.read_csv("train_cab.csv")
#Have a glimpse of the data
train.head()
train.shape
#Data type check
train.dtypes
#Summary of the data
train.describe()
```

```python
# Let's convert fare_amount to numeric

train['fare_amount'] = pd.to_numeric(train['fare_amount'], errors = 'coerce')

train['fare_amount'].dtypes

train.describe()

#train['pickup_datetime'] =  pd.to_datetime(train['pickup_datetime'], format='%Y-%m-%d %H:%M:%S UTC')

#Getting Error because of some value, let's make it NA and drop

train.loc[train['pickup_datetime'] == '43' ,'pickup_datetime'] = np.nan

train = train.drop(train[train['pickup_datetime'].isnull()].index, axis = 0)

train['pickup_datetime'] =  pd.to_datetime(train['pickup_datetime'], format='%Y-%m-%d %H:%M:%S UTC')


#Checking for data types and top 5 rows

train.dtypes

train.head()


#Let's check for Missing values

train.isnull().sum()


#Let's remove Missing values

train = train.drop(train[train['fare_amount'].isnull()].index,axis=0)

train = train.drop(train[train['passenger_count'].isnull()].index,axis=0)

#Let's check if there are any missing values after removal

train.isnull().sum()

train.shape
```

```
#Feature Engineering
#Let's extract Date, year, month, day, hour out of the timestamp
train['year'] = train['pickup_datetime'].dt.year
train['month'] = train['pickup_datetime'].dt.month
train['date'] = train['pickup_datetime'].dt.day
train['day'] = train['pickup_datetime'].dt.dayofweek
train['hour'] = train['pickup_datetime'].dt.hour

#Let's have a glimpse of the data after extracting new variables
train.shape
train.head()

#Let's drop the pickup_datetime variable
train = train.drop('pickup_datetime',axis=1)
train.head()

#Outlier analysis
cnames =['fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count']
import matplotlib.pyplot as plt
%matplotlib inline
for i in cnames:
    print(i)
    plt.boxplot(train[i])
    plt.xlabel(i)
```

```
    plt.ylabel('fare_amount')

    plt.title('outlier analysis')

    plt.show()


#Let's treat outliers


#Fare Amount

# Quartiles

Q1,Q3 = np.percentile(train['fare_amount'],[25,75])

#IQR

IQR = Q3-Q1

# Lower and upper limits

LL = Q1 - (1.5 * IQR)

UL = Q3 + (1.5 * IQR)


# Capping with ul for maxmimum values

train.loc[train['fare_amount'] < LL ,'fare_amount'] = LL

train.loc[train['fare_amount'] > UL ,'fare_amount'] = UL

#Less than 0 and 0.01 values

train.loc[train.fare_amount <= 0,'fare_amount'] = np.nan

train.loc[train.fare_amount == 0.01,'fare_amount'] = np.nan


train = train.drop(train[train['fare_amount'].isnull()].index, axis = 0)

train['fare_amount'].describe()
```

```
#Pick up Longitude

train['pickup_longitude'].describe()

# Quartiles

Q1,Q3 = np.percentile(train['pickup_longitude'],[25,75])

#IQR

IQR = Q3-Q1

# Lower and upper limits

LL = Q1 - (1.5 * IQR)

UL = Q3 + (1.5 * IQR)

# Max of this variable is 40.77 which we can consider as outlier and capping with UL

train.loc[train['pickup_longitude'] < LL ,'pickup_longitude'] = LL

train.loc[train['pickup_longitude'] > UL ,'pickup_longitude'] = UL


#Pick up Latitude

train['pickup_latitude'].describe()

# Quartiles

Q1,Q3 = np.percentile(train['pickup_latitude'],[25,75])

#IQR

IQR = Q3-Q1

# Lower and upper limits

LL = Q1 - (1.5 * IQR)

UL = Q3 + (1.5 * IQR)


# Capping with ul for maxmimu values

train.loc[train['pickup_latitude'] < LL ,'pickup_latitude'] = LL

train.loc[train['pickup_latitude'] > UL ,'pickup_latitude'] = UL
```

```
train['pickup_latitude'].describe()


#Drop off Longitude
train['dropoff_longitude'].describe()
# Quartiles
Q1,Q3 = np.percentile(train['dropoff_longitude'],[25,75])
#IQR
IQR = Q3-Q1
# Lower and upper limits
LL = Q1 - (1.5 * IQR)
UL = Q3 + (1.5 * IQR)


# Capping with ul for maxmimu values
train.loc[train['dropoff_longitude'] < LL,'dropoff_longitude'] = LL
train.loc[train['dropoff_longitude'] > UL,'dropoff_longitude'] = UL


#Drop off Latitude
train['dropoff_latitude'].describe()
# Quartiles
Q1,Q3 = np.percentile(train['dropoff_latitude'],[25,75])
#IQR
IQR = Q3-Q1
# Lower and upper limits
LL = Q1 - (1.5 * IQR)
UL = Q3 + (1.5 * IQR)
```

```
# Capping with ul for maxmimu values
train.loc[train['dropoff_latitude'] < LL ,'dropoff_latitude'] = LL
train.loc[train['dropoff_latitude'] > UL ,'dropoff_latitude'] = UL
train['dropoff_latitude'].describe()

#Passenger Count
train['passenger_count'].describe()
# Quartiles
Q1,Q3 = np.percentile(train['passenger_count'],[25,75])
#IQR
IQR = Q3-Q1
# Lower and upper limits
LL = round(Q1 - (1.5 * IQR))
UL = round(Q3 + (1.5 * IQR))

# Capping with UL for maxmimum values
train.loc[train['passenger_count'] < LL ,'passenger_count'] = LL
train.loc[train['passenger_count'] > 6 ,'passenger_count'] = UL

train.loc[train['passenger_count'] < 1 ,'passenger_count'] = np.nan

train = train.drop(train[train['passenger_count'].isnull()].index, axis = 0)
train['passenger_count'].describe()
```

```
#Boxplot after removal of Outliers
import matplotlib.pyplot as plt

%matplotlib inline

for i in cnames:
    print(i)
    plt.boxplot(train[i])
    plt.xlabel(i)
    plt.ylabel('fare_amount')
    plt.title('outlier analysis')
    plt.show()
train.head()
train.shape

#Let's calculate distance between two geo codes using haversine formula
from math import radians, cos, sin, asin, sqrt

def distance1(lat1, long1, lat2, long2):
    R_earth = 6371 # earth radius (km)
    #Convert degrees to radians
    lat1, long1, lat2, long2 = map(np.radians, [lat1, long1, lat2, long2])

    #Compute distances along lat, lon dimensions
    dlat = lat2 - lat1
    dlon = long2 - long1
```

```
    #Compute haversine distance
    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2

    return 2 * R_earth * np.arcsin(np.sqrt(a))

#Let's calculate distance using above function
train['distance'] =
distance1(train['pickup_latitude'],train['pickup_longitude'],train['dropoff_latitude']
,train['dropoff_longitude'])
train.head()
train.shape
train.dtypes

#Outlier for distance
%matplotlib inline
plt.boxplot(train['distance'])
plt.xlabel('distance')
plt.title('outlier analysis')
plt.show()
train['distance'].describe()

#Let's replace less than 1 values with mean
train.loc[train.distance < 1,'distance'] = train['distance'].mean()
```

```
#Boxplot after removing outliers

%matplotlib inline

plt.boxplot(train['distance'])

plt.xlabel('distance')

plt.title('outlier analysis')

plt.show()

train.head()

train.shape


#Exploratory Data Analysis


#Continous variables analysis


#Fare amount

a1 = ggplot(train, aes( x = 'fare_amount')) + geom_histogram(fill = "royalblue", bins = 25) +¥

  labs(x = "Fare Amount", y = "Frequency") + ggtitle("Distribution of Fare Amount")

a1


#Pickup Longitude

a2 = ggplot(train, aes( x = 'pickup_longitude')) + geom_histogram( fill = "lightgreen") +¥

  labs(x = "Pick up Longitude", y = "Frequency") + ggtitle("Distribution of Pick Up
Longitude")

a2

#Pickup Latitude

a3 = ggplot(train, aes( x = 'pickup_latitude')) + geom_histogram(fill = "green") +¥

  labs(x = "Pick up Latitude", y = "Frequency") + ggtitle("Distribution of Pick Up Latitude")
```

```
a3

#Drop off Longitude

a4 = ggplot(train, aes( x = 'dropoff_longitude')) + geom_histogram( fill = "springgreen") +¥

  labs(x = "Drop off Longitude", y = "Frequency") + ggtitle("Distribution of Drop off
Longitude")

a4

#Drop off Latitude

a5 = ggplot(train, aes( x = 'dropoff_latitude')) + geom_histogram( fill = "limegreen") +¥

  labs(x = "Drop off Latitude", y = "Frequency") + ggtitle("Distribution of Drop off Latitude")

a5

#Passenger Count

a6 = ggplot(train, aes( x = 'passenger_count')) + geom_histogram(binwidth = 1, fill =
"powderblue") + ¥

  labs(x = "Passenger count", y = "Frequency") + ggtitle("Distribution of Passenger count")

a6

#Distance

a7 = ggplot(train, aes( x = 'distance')) + geom_histogram(binwidth = 1, fill = "navy") +¥

  labs(x = "Distance", y = "Frequency") + ggtitle("Distribution of Distance")

a7

#Distance is right skewed, let's remove skewness using log transformation

train['distance'] = np.log(train['distance'])

#Let's check distribution after transformation

a7 = ggplot(train, aes( x = 'distance')) + geom_histogram( fill = "navy") +¥

  labs(x = "Distance", y = "Frequency") + ggtitle("Distribution of Distance")

a7
```

```
#Categorical Variables


#Fare amount vs Year
b1 = ggplot(train, aes('year', 'fare_amount')) + geom_bar(stat = "identity", fill = "yellow") + ¥
  labs( x = "Year", y = "Fare Amount") + ggtitle("Fare Amount vs Year")
b1
#Fare amount vs Month
b2 = ggplot(train, aes('month', 'fare_amount')) + geom_bar(stat = "identity", fill = "khaki") +¥
  labs( x = "Month", y = "Fare Amount") + ggtitle("Fare Amount vs Month")
b2
#Fare amount vs Day
b3 = ggplot(train, aes('day', 'fare_amount')) + geom_bar(stat = "identity", fill = "orange") +¥
  labs( x = "Day", y = "Fare Amount") + ggtitle("Fare Amount vs Day")
b3
#Fare amount vs Hour
b4 = ggplot(train, aes('hour', 'fare_amount')) + geom_bar(stat = "identity", fill = "peru") +¥
  labs( x = "Hour", y = "Fare Amount") + ggtitle("Fare Amount vs Hour")
b4


#Correlation plot
cnames = ['fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count','distance']


df_corr = train.loc[:,cnames]
c1 = df_corr.corr()
print(c1)
```

```
sns.heatmap(c1,square=True,annot=True)
#Anova test for categorical variables
cat = ['year','month', 'date', 'day', 'hour']

import statsmodels.api as sm

from statsmodels.formula.api import ols

for i in cat:
    mod = ols('fare_amount' + '~' + i, data = train).fit()
    aov_table = sm.stats.anova_lm(mod, typ = 2)
    print(aov_table)
#p > 0.05 for day and date. So let's remove those variables
DropVar = ['date', 'day']

train = train.drop(DropVar , axis = 1)

#Let's create Train and Test dataset
from sklearn.model_selection import train_test_split

train_data,test_data = train_test_split(train, test_size = 0.2, random_state = 123)
train_data.head()

#Linear regression
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
```

```
from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score


#Let's train the model

model1 = sm.OLS(train_data.iloc[:,0].astype(float), train_data.iloc[:,1:10].astype(float)).fit()

model1.summary()


#Let's predict

prediction1 = model1.predict(test_data.iloc[:,1:10])

data_model1 = pd.DataFrame({'actual': test_data.iloc[:,0], 'pred': prediction1})

data_model1.head()


#Function to calculate MAPE

def MAPE(y_actual,y_pred):

    mape = np.mean(np.abs((y_actual - y_pred)/y_actual)*100)

    return mape

MAPE(test_data.iloc[:,0],prediction1)


#Random Forest

from sklearn.ensemble import RandomForestRegressor


#Let's train the model

model2 =
RandomForestRegressor(n_estimators=500,random_state=123).fit(train_data.iloc[:,1:10],
train_data.iloc[:,0])
```

```
#Let's predict

prediction2 = model2.predict(test_data.iloc[:,1:10])

data_model2 = pd.DataFrame({"actual" : test_data.iloc[0:,0],"pred" : prediction2})

data_model2.head()

MAPE(test_data.iloc[:,0], prediction2)


#Model Evaluation


#Load the test data

test = pd.read_csv('test.csv')

#Glimpse of the data

test.head()

test.dtypes

test.describe()

#Format conversion

test['pickup_datetime'] =  pd.to_datetime(test['pickup_datetime'], format='%Y-%m-%d
%H:%M:%S UTC')


#Feature Engineering


test['year'] =test['pickup_datetime'].dt.year

test['month'] = test['pickup_datetime'].dt.month

test['date'] = test['pickup_datetime'].dt.day

test['day'] = test['pickup_datetime'].dt.dayofweek

test['hour'] = test['pickup_datetime'].dt.hour

#Let's delete Pick up datetime
```

```
test = test.drop(['pickup_datetime'], axis = 1)

test.head()


#Let's check for Missing values

test.isnull().sum()


#Outlier Analysis

test.describe()


#Let's calculate Distance

test['distance'] =
distance1(test['pickup_latitude'],test['pickup_longitude'],test['dropoff_latitude'],test['dropoff_l
ongitude'])

test.head()


#Outliers for Distance

test['distance'].describe()


#Let's replace 0 with mean distance values

test.loc[test.distance < 1,'distance'] = test['distance'].mean()

test['distance'].describe()

test.head()

test = test.drop(['date', 'day'], axis = 1)


#Columns in Test data
```

```
test.columns

#Outliers in Distance
sns.distplot(test['distance'],color='black')
plt.title("Distribution of variable distance")
plt.ylabel("Density")
plt.show()
#Right skewed, Let's remove skewness using log function

test['distance'] = np.log(test['distance'])

#Data after log transformation
sns.distplot(test['distance'],color='black')
plt.title("Distribution of distance")
plt.ylabel("Density")
plt.show()

#Let's apply Random Forest model
Final_model = model2.predict(test)

#Dependent variable
test['Predicted_Fare_Amount'] = Final_model
test.head()
```