

Predication of fare amount for a cab ride

Satyam Sharma 23-07-2019



Contents

[Chapter 1 2](#_Toc14741091)

[Introduction 2](#_Toc14741092)

[1.1 Problem Statement 2](#_Toc14741093)

[1.2 Data 2](#_Toc14741094)

[Chapter 2 4](#_Toc14741095)

[Methodology 4](#_Toc14741096)

[2.1 Pre-processing 4](#_Toc14741097)

[2.2 Modelling 12](#_Toc14741098)

[Chapter 3 14](#_Toc14741099)

[Conclusion 14](#_Toc14741100)

[3.1 Model Evaluation 14](#_Toc14741101)

[3.2 Model Tuning 14](#_Toc14741102)

[3.2 Model Selection 14](#_Toc14741103)

[Graphs 15](#_Toc14741104)

[Prediction File 15](#_Toc14741105)

[Appendix A - R code 15](#_Toc14741106)

[Appendix B - Python code 29](#_Toc14741107)

[References 39](#_Toc14741108)

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 39](#_Toc14741109)

# 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

Our aim is to predict Fare amount for a particular ride in the city.

**Data Details:**

Number of attributes: 6 in Test data, 7 in training data

7th column is the fare amount column which we will need to predict for test data.

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.

**Below is the sample data for the same:**

Table 1.1: Column 1 - 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| pickup\_datetime | pickup\_longitude | pickup\_latitude | dropoff\_longitude | dropoff\_latitude | passenger\_count |
| 2015-01-27 13:08:24 UTC | -73.97332001 | 40.76380539 | -73.98143005 | 40.74383545 | 1 |
| 2015-01-27 13:08:24 UTC | -73.98686218 | 40.71938324 | -73.99888611 | 40.73920059 | 1 |
| 2011-10-08 11:53:44 UTC | -73.982524 | 40.75126 | -73.979654 | 40.746139 | 1 |
| 2012-12-01 21:12:12 UTC | -73.98116 | 40.767807 | -73.990448 | 40.751635 | 1 |
| 2012-12-01 21:12:12 UTC | -73.966046 | 40.789775 | -73.988565 | 40.744427 | 1 |
| 2012-12-01 21:12:12 UTC | -73.960983 | 40.765547 | -73.979177 | 40.740053 | 1 |

Table 1.2: Column 6 – 10

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| fare\_amount | pickup\_datetime | pickup\_longitude | pickup\_latitude | dropoff\_longitude | dropoff\_latitude | passenger\_count |
| 4.5 | 2009-06-15 17:26:21 UTC | -73.8443 | 40.72132 | -73.8416 | 40.71228 | 1 |
| 16.9 | 2010-01-05 16:52:16 UTC | -74.016 | 40.7113 | -73.9793 | 40.782 | 1 |
| 5.7 | 2011-08-18 00:35:00 UTC | -73.9827 | 40.76127 | -73.9912 | 40.75056 | 2 |
| 7.7 | 2012-04-21 04:30:42 UTC | -73.9871 | 40.73314 | -73.9916 | 40.75809 | 1 |
| 5.3 | 2010-03-09 07:51:00 UTC | -73.9681 | 40.76801 | -73.9567 | 40.78376 | 1 |

This is the column we need to correctly predict

Table 1.3: Column 11

|  |
| --- |
| fare\_amount |
| 4.5 |
| 16.9 |
| 5.7 |
| 7.7 |
| 5.3 |

# Chapter 2

## Methodology

### 2.1 Pre-processing

It is a data mining technique that transforms raw data into an understandable format. Raw data(real world data) is always incomplete and that data cannot be sent through a model. That would cause certain errors. That is why we need to pre-process data before sending through a model.

Steps in Data Preprocessing

Here are the steps:

* Import libraries
* Read data
* Type conversion
* Missing values
* Outlier Analysis
* Feature Selection

#### 2.1.1 Type Conversion

Take a glance at structure of data:

Training data:

'data.frame': 16067 obs. of 7 variables:

$ fare\_amount : Factor w/ 469 levels "","-2.5","-2.9",..: 302 59 374 433 371 27 432 57 1 454 ...

$ pickup\_datetime : Factor w/ 16021 levels "2009-01-01 01:31:49 UTC",..: 1115 2509 6550 8252 2919 4986 9743 7479 9838 1606 ...

$ pickup\_longitude : num -73.8 -74 -74 -74 -74 ...

$ pickup\_latitude : num 40.7 40.7 40.8 40.7 40.8 ...

$ dropoff\_longitude: num -73.8 -74 -74 -74 -74 ...

$ dropoff\_latitude : num 40.7 40.8 40.8 40.8 40.8 ...

$ passenger\_count : num 1 1 2 1 1 1 1 1 1 2 ...

Test data:

'data.frame': 9914 obs. of 6 variables:

$ pickup\_datetime : Factor w/ 1753 levels "2009-01-01 11:04:24 UTC",..: 1648 1648 747 1041 1041 1041 744 744 744 1384 ...

$ pickup\_longitude : num -74 -74 -74 -74 -74 ...

$ pickup\_latitude : num 40.8 40.7 40.8 40.8 40.8 ...

$ dropoff\_longitude: num -74 -74 -74 -74 -74 ...

$ dropoff\_latitude : num 40.7 40.7 40.7 40.8 40.7 ...

$ passenger\_count : int 1 1 1 1 1 1 1 1 1 1 ...

Observations:

* Fare\_amount should be numeric
* Pickup\_datetime should be splitted into proper date and time format

Type conversion and column split:

* Split Pickup\_datetime into day,month,year and time columns

After converting data types:

Train Data:

'data.frame': 16067 obs. of 10 variables:

$ fare\_amount : num 302 59 374 433 371 27 432 57 1 454 ...

$ pickup\_datetime : Factor w/ 16021 levels "2009-01-01 01:31:49 UTC",..: 1115 2509 6550 8252 2919 4986 9743 7479 9838 1606 ...

$ pickup\_longitude : num -73.8 -74 -74 -74 -74 ...

$ pickup\_latitude : num 40.7 40.7 40.8 40.7 40.8 ...

$ dropoff\_longitude: num -73.8 -74 -74 -74 -74 ...

$ dropoff\_latitude : num 40.7 40.8 40.8 40.8 40.8 ...

$ passenger\_count : num 1 1 2 1 1 1 1 1 1 2 ...

$ month : num 6 1 8 4 3 1 11 1 12 9 ...

$ year : num 2009 2010 2011 2012 2010 ...

$ time : 'ITime' int 17:26:21 16:52:16 00:35:00 04:30:42 07:51:00 09:50:45 20:35:00 17:22:00 13:10:00 01:11:00 ...

Test Data:

'data.frame': 9914 obs. of 9 variables:

$ pickup\_datetime : Factor w/ 1753 levels "2009-01-01 11:04:24 UTC",..: 1648 1648 747 1041 1041 1041 744 744 744 1384 ...

$ pickup\_longitude : num -74 -74 -74 -74 -74 ...

$ pickup\_latitude : num 40.8 40.7 40.8 40.8 40.8 ...

$ dropoff\_longitude: num -74 -74 -74 -74 -74 ...

$ dropoff\_latitude : num 40.7 40.7 40.7 40.8 40.7 ...

$ passenger\_count : int 1 1 1 1 1 1 1 1 1 1 ...

$ month : num 1 1 10 12 12 12 10 10 10 2 ...

$ year : num 2015 2015 2011 2012 2012 ...

$ time : 'ITime' int 13:08:24 13:08:24 11:53:44 21:12:12 21:12:12 21:12:12 12:10:20 12:10:20 12:10:20 15:22:20 ...

We will drop old datetime column before feeding it to model as it has now been split.

#### 2.1.2 Missing value analysis

Below are the sheets having missing value details in both Train and test data:



Observation:

* Test data has no missing values.
* Train data is having missing values below are the column wise details:

fare\_amount pickup\_datetime pickup\_longitude pickup\_latitude

0 0 0 0

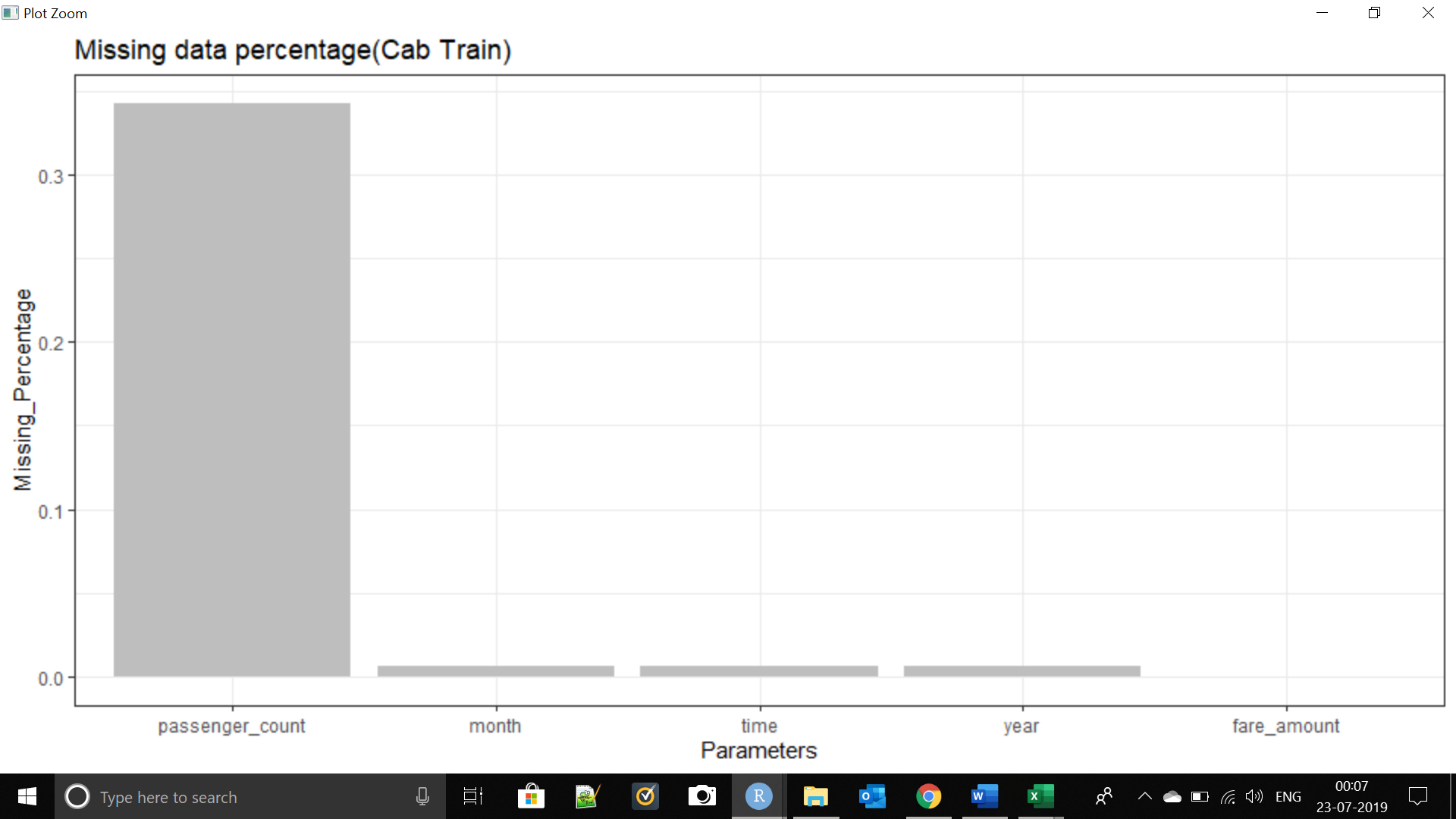
dropoff\_longitude dropoff\_latitude passenger\_count month

0 0 55 1

year time

1 1

Missing Value graph:



Missing Value Treatment:

We have used KNN imputation for missing value treatment

Description

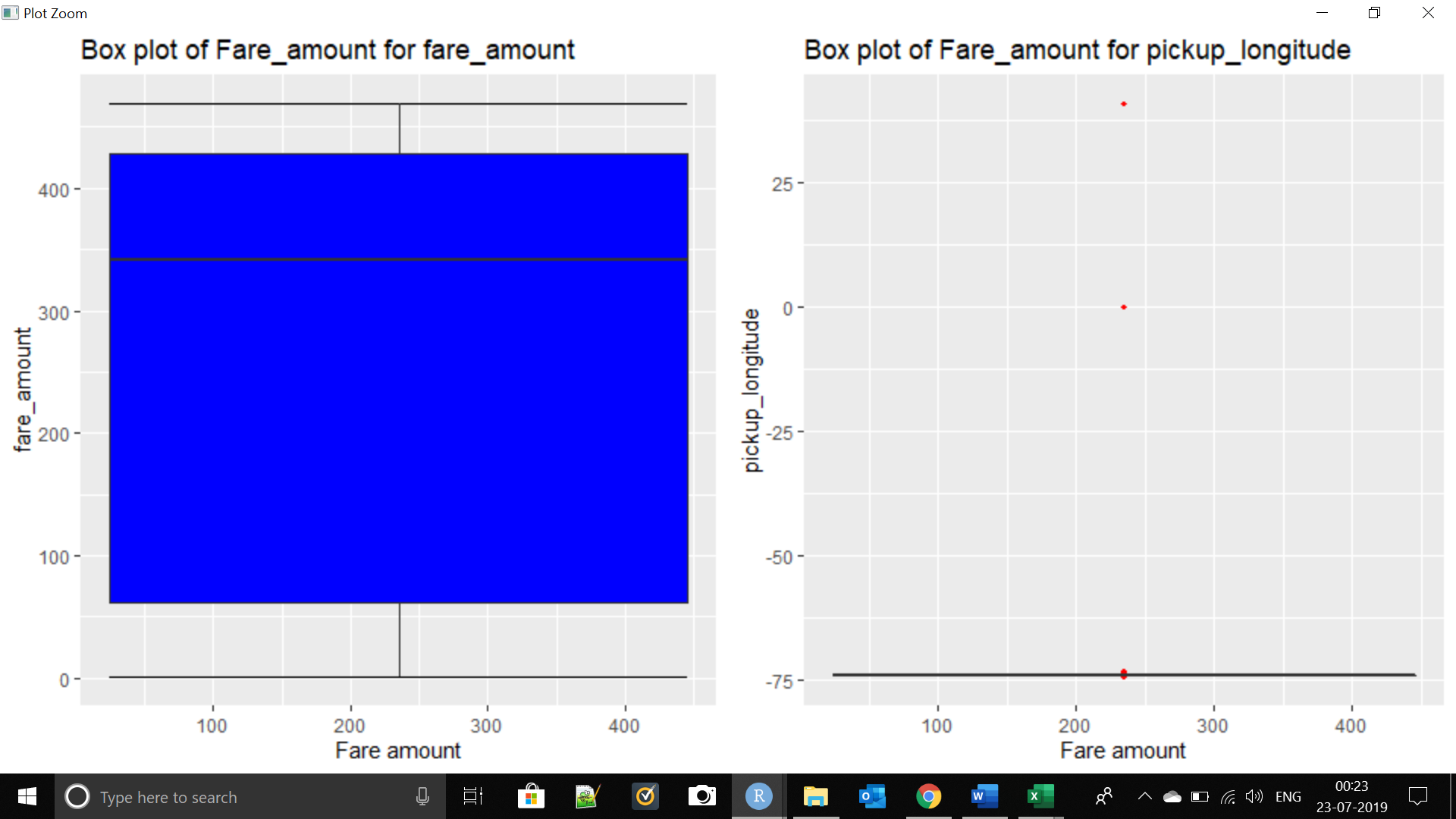
Function that fills in all NA values using the k Nearest Neighbours of each case with NA values. By default it uses the values of the neighbours and obtains an weighted (by the distance to the case) average of their values to fill in the unknows. If meth='median' it uses the median/most frequent value, instead.

#### 2.1.3 Outlier analysis

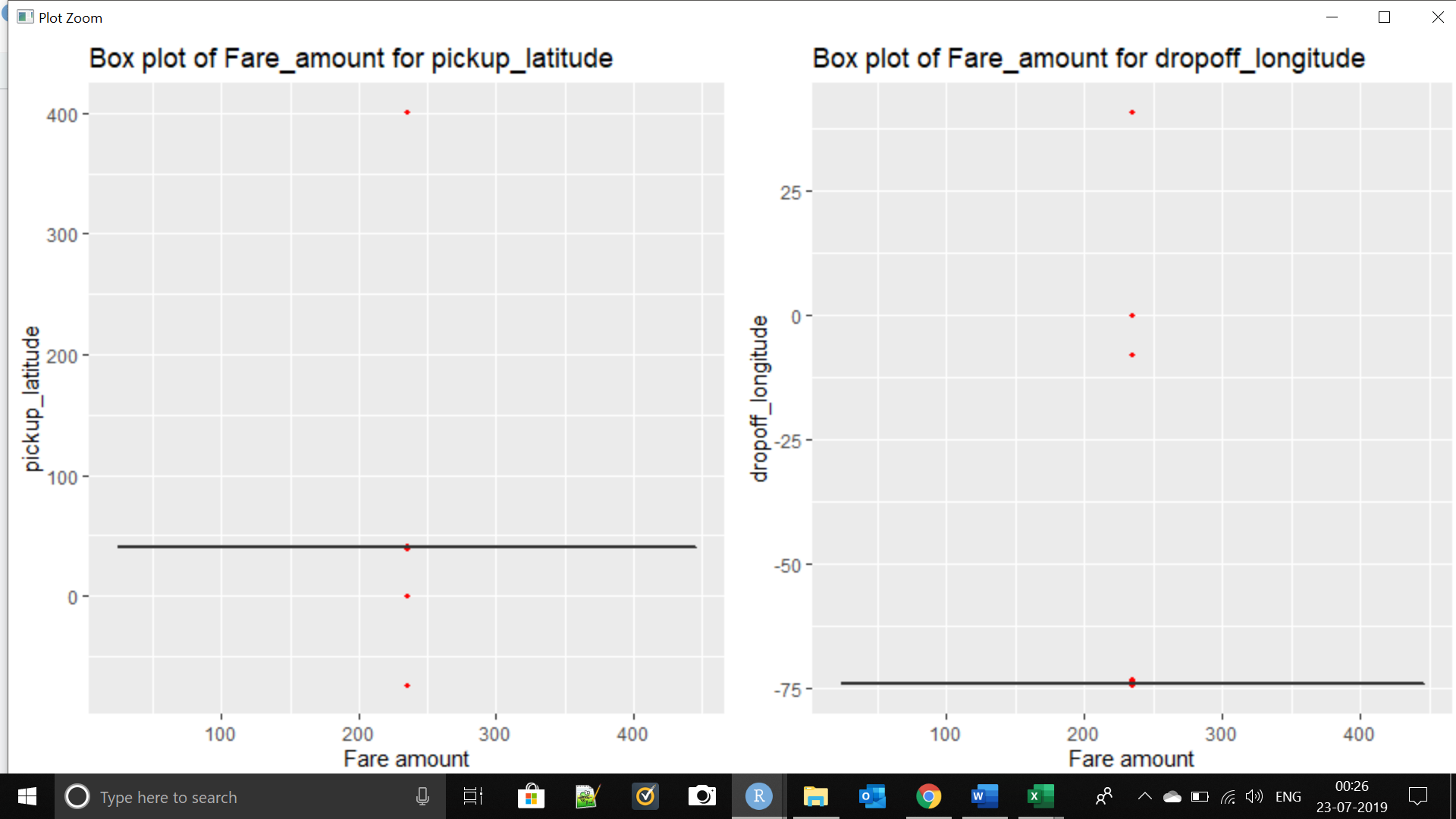
we have to select only numeric data for outlier analysis. In our case we have all numeric data.

Boxplot:

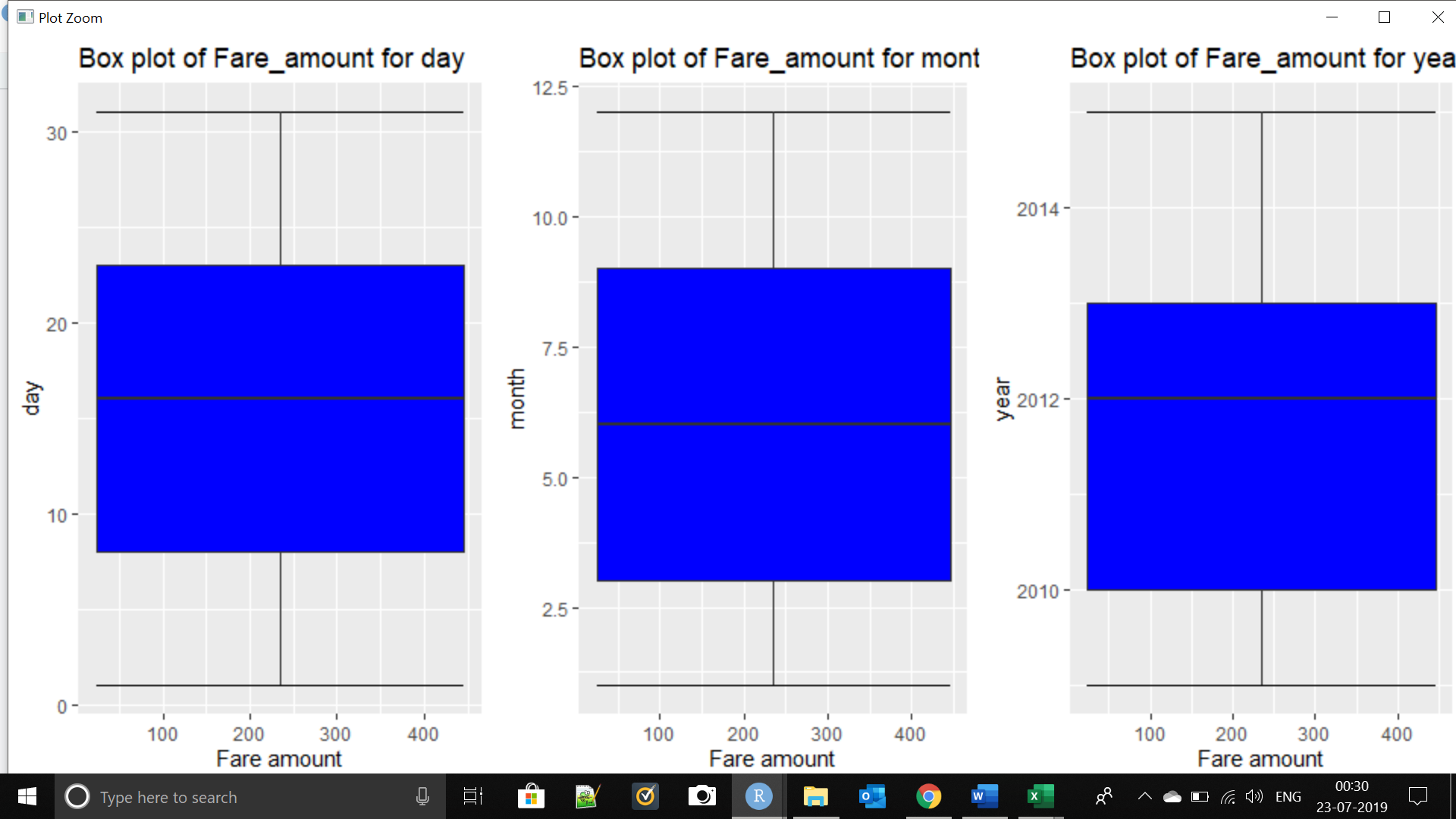
Fare amount vs fare amount & fare amount vs pickup longitude



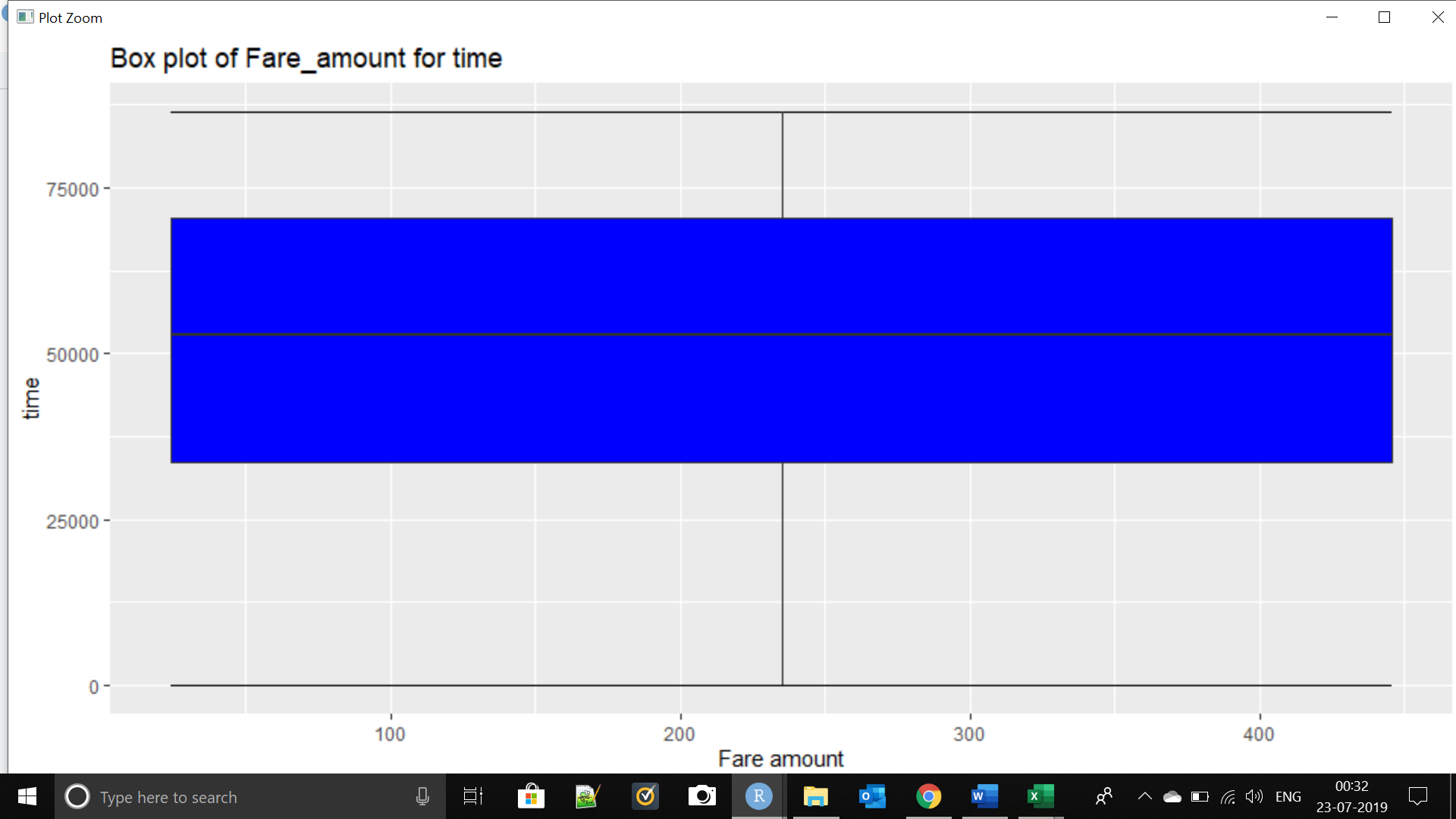
Fare amount vs dropoff longitude & fare amount vs pickup latitude



Fare amount vs day & fare amount vs month & fare amount vs year



Fare amount vs time



Observation:

* Graphs are against fare\_amount at x axis
* Graphs are representing outliers in red colour
* Except day time month year we have outliers in each column

Outlier Treatment:

We have removed outliers from our data and below is the report of outliers treated:

[1] "fare\_amount"

[1] 0

[1] "pickup\_longitude"

[1] 1114

[1] "pickup\_latitude"

[1] 267

[1] "dropoff\_longitude"

[1] 678

[1] "dropoff\_latitude"

[1] 408

[1] "passenger\_count"

[1] 1445

[1] "day"

[1] 0

[1] "month"

[1] 0

[1] "year"

[1] 0

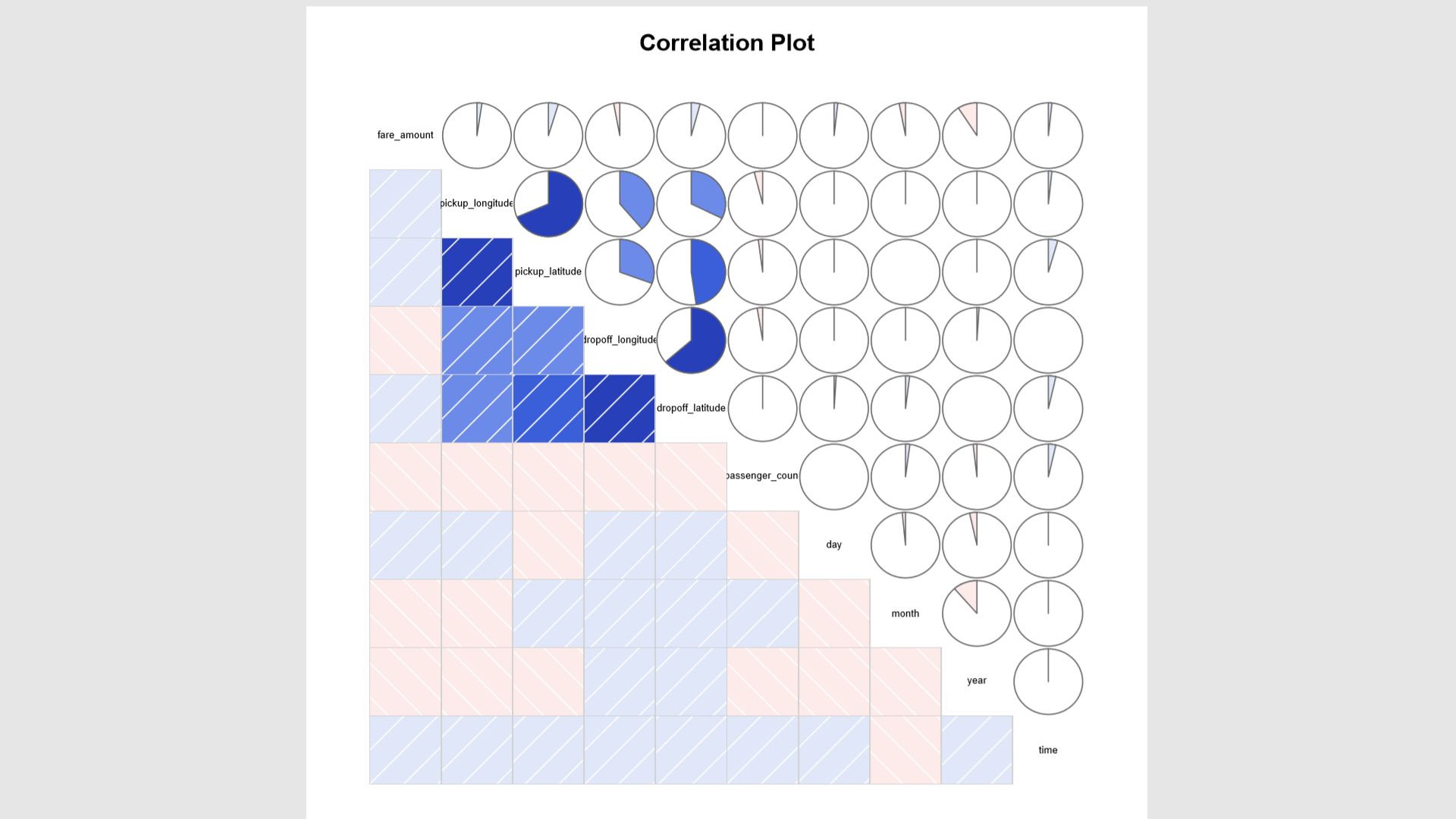
[1] "time"

[1] 0

#### 2.1.3 Feature Selection

Correlation Analysis:

Correlation Plot



Observations:

* Blue colour represents that variables are positively correlated.
* pickup\_longitude and pickup\_latitude are highly correlated.
* dropoff\_longitude and dropoff\_latitude are highly correlated.
* We can drop one of above two observations but we are keeping them for distance because distance is very important factor.

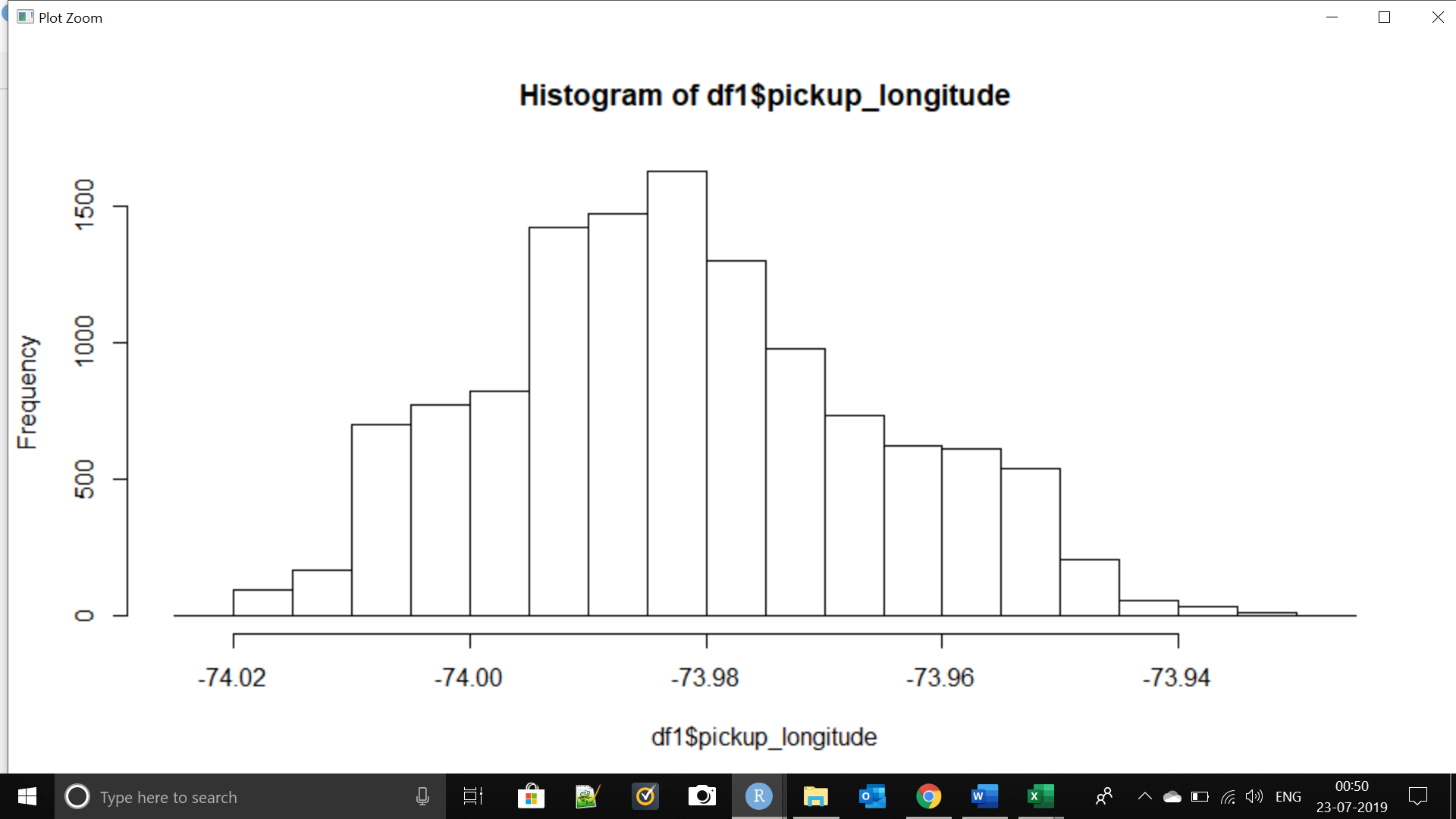
**Knowledge**: Chi square test will be applied for categorical variables We are not having that's why leaving it.

#### 2.1.3 Feature Scaling

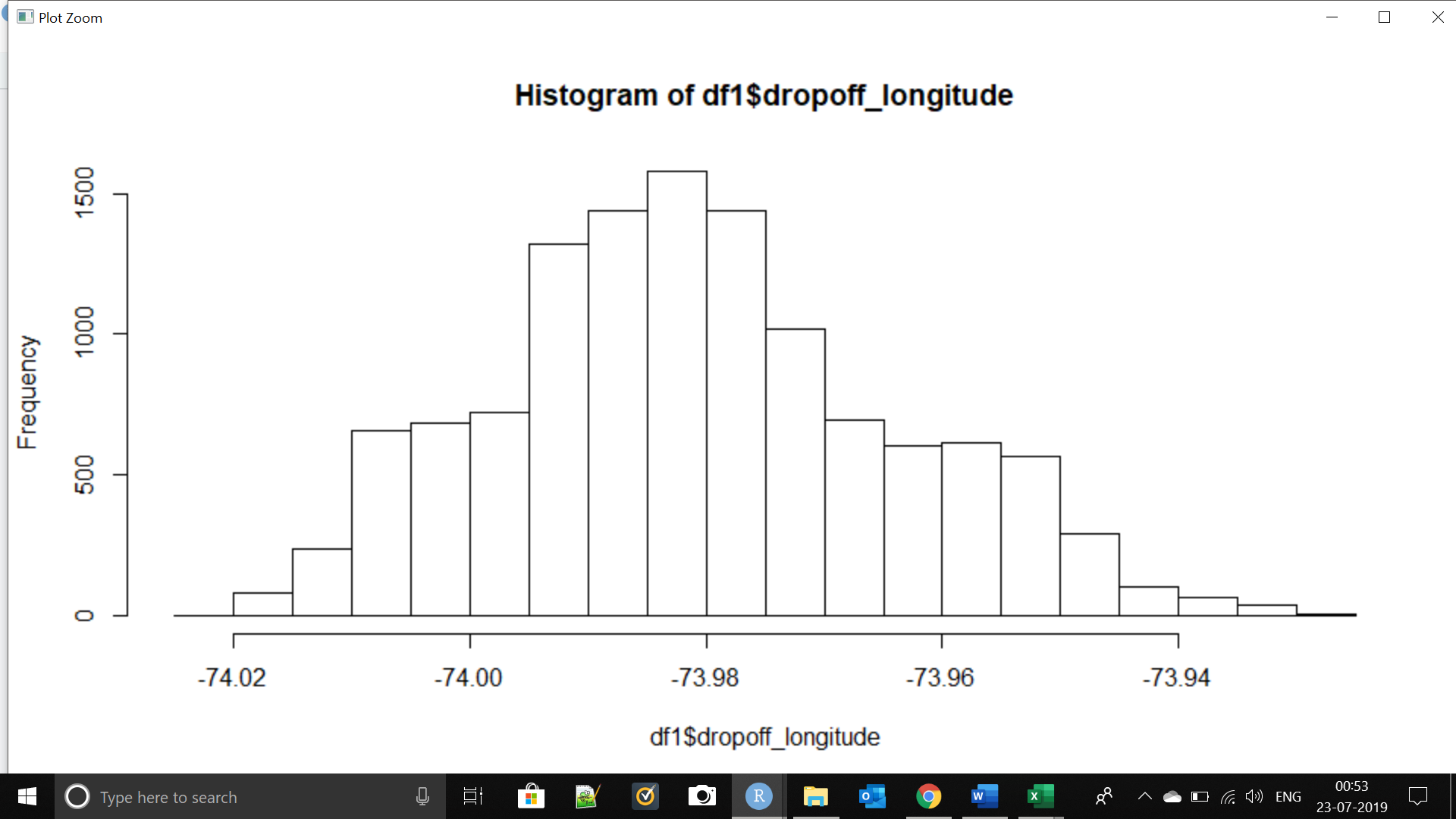
Normalization Check

Histograms:

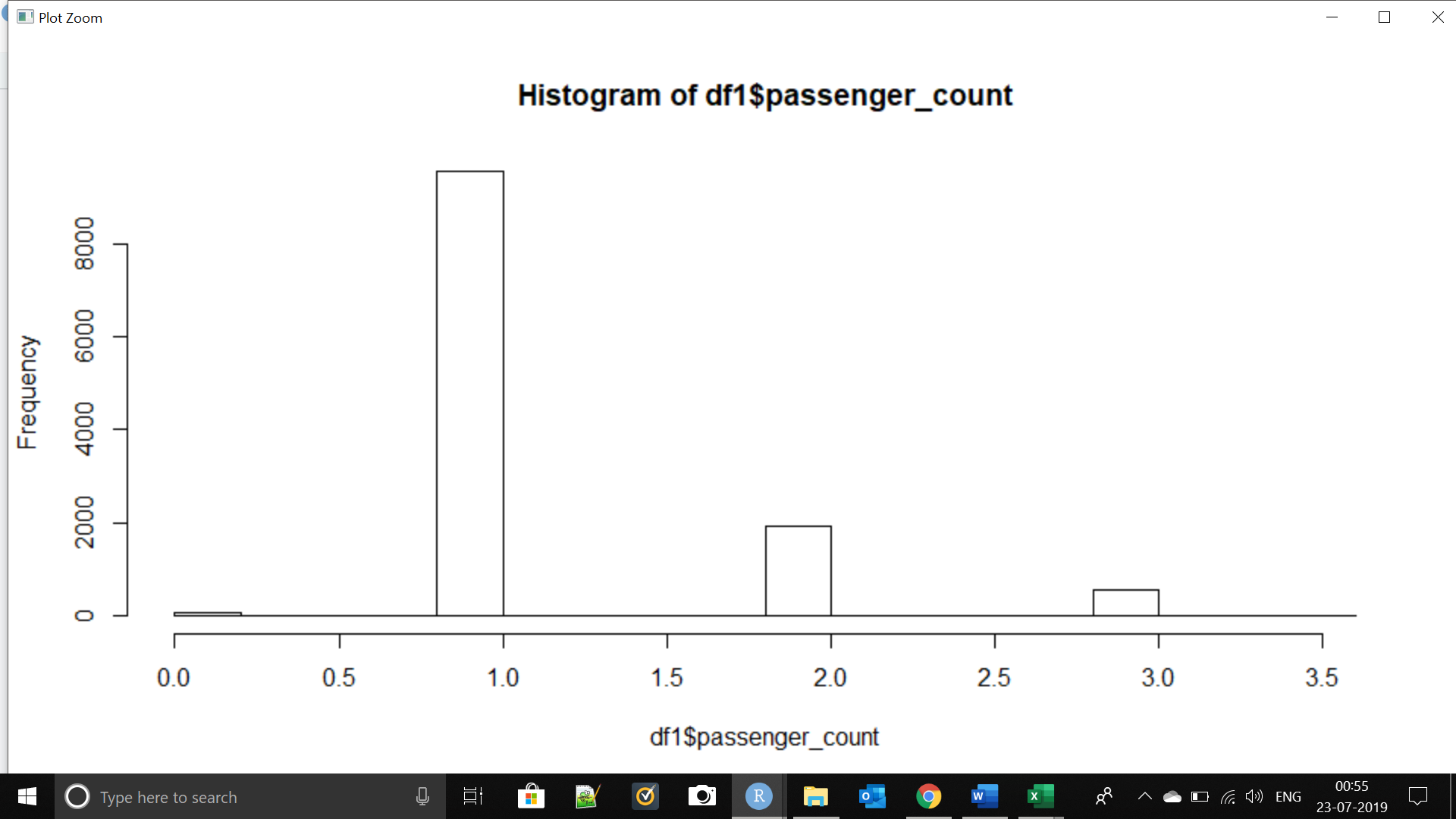
Pickup\_longitude



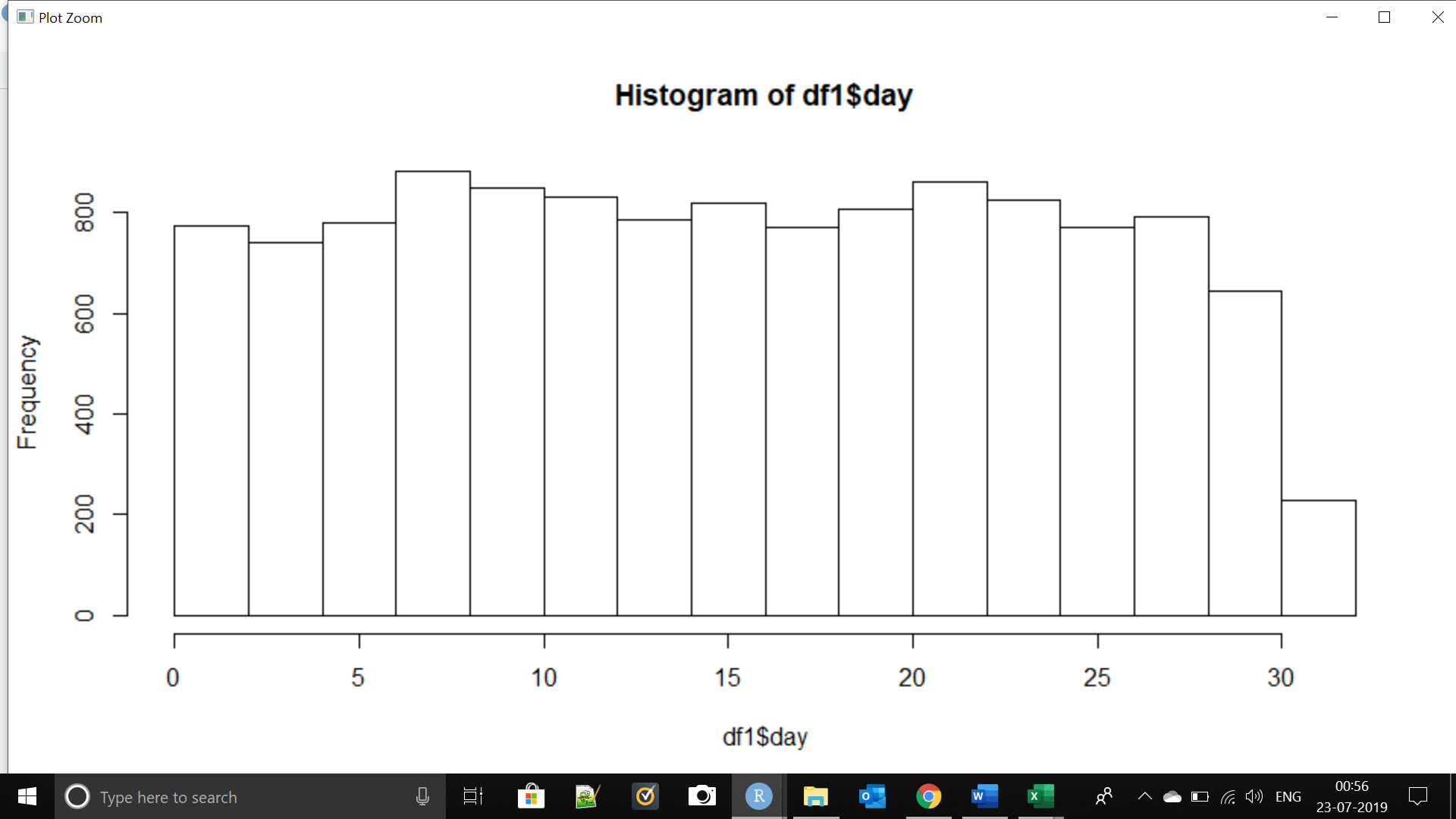
dropoff\_longitude



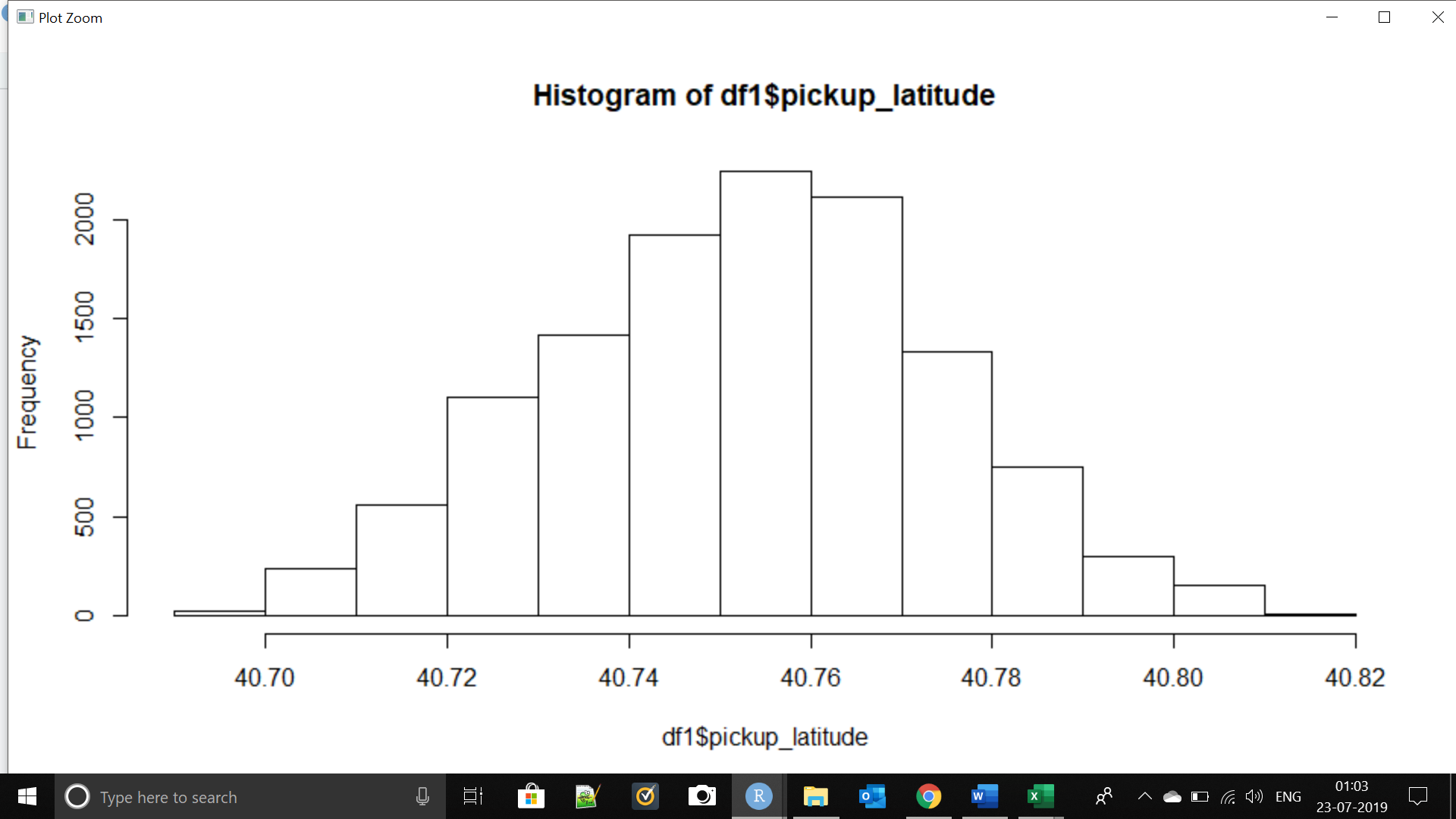
passenger\_count



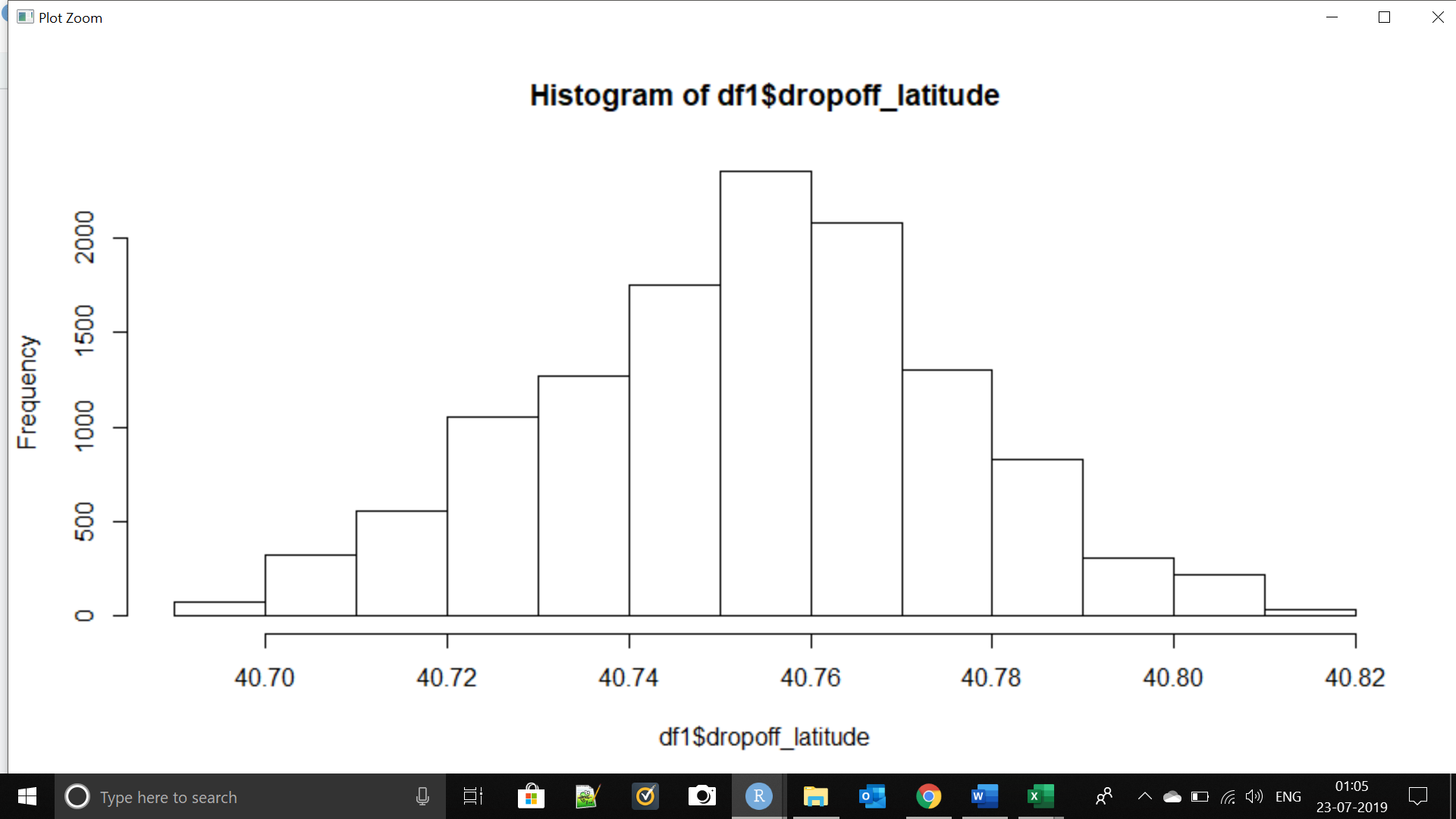
day



pickup\_latitude



dropoff\_latitude



Observation:

Ignoring date values as they should not actually be treated as numeric we have all the columns almost symmetric.

### 2.2 Modelling

Problem statement tells that this is a regression problem. So we will try some regression algorithms and evaluate the results.

In regression problems our aim is to predict a value by training our model with dependent variables.

Below are some of the important and basic regression algorithms:

* **Decision Tree**

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data

* **Random Forest**

Random forest is a kind of universal machine learning technique. It can be used for both regression (target is a continuous variable) or classification (target is a categorical variable) problems

* **Linear Regression**

Linear Regression is a machine learning algorithm based on supervised learning. It performs regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

We are splitting our training data into train and test for evaluation purpose. We will keep 80% data for train and 20% for test.

We will try three methods consecutively and evaluate the results and whichever has least MAPE will be used for our model.

Decision Tree

We are using “Anova” method as this is regression problem.

Random Forest

We are taking 300 trees initially and will increase or decrease during model tuning.

#### 2.2 Modelling

#### 2.2.1 Model Selection

Problem statement tells that this is a regression problem. So we will try some regression algorithms and evaluate the results.

Decision tree

Since the process of constructing these decision trees assume no distributional patterns in the data (non-parametric), characteristics of the input data are usually not given much attention. We consider some characteristics of input data and their effect on the learning performance of decision trees. Preliminary results indicate that the performance of decision trees can be improved with minor modifications of input data.

Random forest

Random Forest is an ensemble machine learning technique capable of performing both regression and classification tasks using multiple decision trees and a statistical technique called **bagging.**

Linear Regression

Learning a linear regression model means estimating the values of the coefficients used in the representation with the data that we have available.

# 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.

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

#### 3.1.1 Mean Absolute Percentage Error (MAE)/RMSE/MSE/MAPE

MAE is one of the error measures used to calculate the predictive performance of the model. We will apply this measure to our models that we have generated in the previous section.

|  |
| --- |
|  |
|  |
|  |

### 3.2 Model Tuning

Using random forest as it is giving least error we will try some combination with number of trees and will evaluate which is the best and will use that in model.

> predict\_RF4 = predict(model\_RF4, test[,-1])

>

> regr.eval(test[,1],predict\_RF4,stats = c('mae','rmse','mape','mse'))

mae rmse mape mse

111.681136 138.548239 3.323446 19195.614639

We tried some combination but the best exist with 300 so we will use that one only for prediction.

### 3.2 Model Selection

Least MAPE is second one using Random forest with 300 trees so we will use that for prediction.

> final\_predict\_RF = predict(model\_RF, testDF)

> testDF$Predicted\_Fare = final\_predict\_RF

> #write file

> write.csv(testDF,"Prediction.csv",row.names = FALSE)

# Graphs

Attachment has PDF with all graphs.



# Prediction File



# R code file



# Python File



# Appendix A - R code

####Clear objects####

rm(list=ls(all=T))

####set working directory#####

setwd("C:/Users/ASUS/Desktop/Edwisor Training/Project-3\_Cab\_fare/R")

###importing packages####

library(ggplot2)

#library(tidyverse)

####Read file####

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

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

####Glance at data ########

head(cab\_data)

dim(cab\_data)

#We have 16067 observation and 7 features in our train data##

head(cab\_test\_data)

dim(cab\_test\_data)

#We have 9914 observation and 6 features in test data##

####Checking structure of both data files#####

str(cab\_data)

#we have first two features(fare\_amount and pickup\_datetime) as factor

str(cab\_test\_data)

# here also pickup\_datetime is factor

######converting data types#####

#copy data ##

df1 = cab\_data

testDF = cab\_test\_data

df1$fare\_amount = as.numeric(df1$fare\_amount)

#testDF$fare\_amount = as.numeric(testDF$fare\_amount)

#We will separate date time values in pickup\_datetime feature for ease in analysis

#and model

#Separating date from pickup\_datetime to dteday

df1$date=as.Date(df1$pickup\_datetime,format="%Y-%m-%d")

testDF$date=as.Date(testDF$pickup\_datetime,format="%Y-%m-%d")

str(testDF)

#check type of dteday

str(df1$date)

str(testDF$date)

df1$day=format(as.Date(df1$date,format="%Y-%m-%d"), "%d")

df1$month=format(as.Date(df1$date,format="%Y-%m-%d"), "%m")

df1$year=format(as.Date(df1$date,format="%Y-%m-%d"), "%Y")

testDF$day=format(as.Date(testDF$date,format="%Y-%m-%d"), "%d")

testDF$month=format(as.Date(testDF$date,format="%Y-%m-%d"), "%m")

testDF$year=format(as.Date(testDF$date,format="%Y-%m-%d"), "%Y")

#Drop date variable as it is of no use

#str(df1)

df1 = df1[,-8]

testDF = testDF[,-7]

#Convert all to numeric for ease of feeding

df1$day=as.numeric(df1$day)

df1$month=as.numeric(df1$month)

df1$year=as.numeric(df1$year)

testDF$day=as.numeric(testDF$day)

testDF$month=as.numeric(testDF$month)

testDF$year=as.numeric(testDF$year)

#str(df1)

#str(testDF)

#same for time

df1$time = strptime(df1$pickup\_datetime,"%Y-%m-%d %H:%M:%S")

testDF$time = strptime(testDF$pickup\_datetime,"%Y-%m-%d %H:%M:%S")

library(data.table)

df1$time = as.ITime(df1$time)

testDF$time = as.ITime(testDF$time)

#df1=df1[,-12]

#str(df1)

#str(testDF)

#check type

#str(df1$daytime)

#str(testDF$daytime)

#Type is POSIXlt now

#Hence now we can feed this to as.ITime function which is part of "data.table" Library to fetch only time from pickup\_datetime feature

#df2=df1

#Check type now

str(df1)

str(testDF)

#It is integer time format now

head(df1)

#check the structure

str(df1)

###########Missing value analysis######

#Checking total missing values

cat("Total missing value=",sum(is.na(df1)))

#Checking columns having missing values

colSums(sapply(df1, is.na))

#######Below is the explanation of function used below####

##data.frame = used to create a dataframe##

##apply = used for loop (as loop are slow we used apply fro faster processing##

##df1 = our dataset##

##function(x)=is the function we are creating in the line itself ##

##followed by {} which will count null values in each column##

##X = the count which we are getting from sunction##

missing\_val = data.frame(apply(df1,2,function(x){sum(is.na(x))}))

missing\_val2 = data.frame(apply(testDF,2,function(x){sum(is.na(x))}))

missing\_val

missing\_val2

##Test data does not have missing values

####converting row names into columns###

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

#missing\_val2$Columns=row.names(missing\_val2)

##remove index = remove frist variable##

row.names(missing\_val) = NULL

#row.names(missing\_val2) = NULL

##row names has been removed

##Rename the variable name of missing values##

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

names(missing\_val2)[1] = "Missing\_Percentage"

missing\_val

missing\_val2

##column name has been changed##

##calculating percentage of missing values###

missing\_val$Missing\_Percentage = (missing\_val$Missing\_Percentage/nrow(df1))\*100

missing\_val2$Missing\_Percentage = (missing\_val2$Missing\_Percentage/nrow(testDF))\*100

###Arrange in descending order to show highest percentage on top##

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

missing\_val2 = missing\_val2[order(-missing\_val2$Missing\_Percentage),]

missing\_val

missing\_val2

##Missing Value observation##

#Test datadoe not have missing values

#Passenger\_count has 0.3423 % of missing values

#dteday and daytime has minor 0.00622 % of missing value

#Passenger count has 55 missing values and dteday & daytime has 1 missing value##

##Rearranging columns of missing value dataframe##

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

#missing\_val2 = missing\_val2[,c(2,1)]

missing\_val

#missing\_val2

#2nd column has now became first column###

####Writing the results into disk###

write.csv(missing\_val,"missing\_perce.csv",row.names = FALSE)

write.csv(missing\_val2,"missing\_perc\_test\_data.csv",row.names = FALSE)

###Plot graph for missing values###

ggplot(data=missing\_val[1:5,],aes(x=reorder(Columns,-Missing\_Percentage),y= Missing\_Percentage))+

geom\_bar(stat = "identity",fill="grey")+ xlab("Parameters")+

ggtitle("Missing data percentage(Cab Train)")+theme\_bw()

#str(missing\_val2)

#ggplot(data=missing\_val2[1:3,],aes(x=reorder(Columns,-Missing\_Percentage),y= Missing\_Percentage))+

# geom\_bar(stat = "identity",fill="grey")+ xlab("Parameters")+

# ggtitle("Missing data percentage(Cab Test)")+theme\_bw()

####As per the rule if missing values are greater then 30% then we are not

####going to deal with that instead we will ignore that###

#####Missing value treatment ######

#Mean Method###

#df2=df1

#str(df2)

#df1$dteday=as.numeric(df1$dteday)

#df1$daytime=as.numeric(df1$daytime)

#testDF$dteday=as.numeric(testDF1$dteday)

#testDF$daytime=as.numeric(testDF$daytime)

#head(df1$dteday)

#we will apply this on copy of our data just to test##

#df2$passenger\_count[is.na(df2$passenger\_count)] = mean(df2$passenger\_count,na.rm = T)

#sum(is.na(df2$daytime))

#have replaced nul values with mean of the column

#KNN imputation##

library(DMwR)

str(df1)

#df2=df2[,-1]

#colnames(df2)

#df2=df2[,-6]

#df2=df2[,-6]

#colnames(df2)

df1=knnImputation(df1,k=5)

#testDF=knnImputation(testDF,k=5)

##there are no missing values now###

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

##We can apply outlier analysis only on numerical varibales##

###Passenger count could have been converted to factor but we are

##keeping it as numeric for ease of analysis and model feeding##

##Excluding pickupdatetime as we have splitted it previously#

df1 = df1[,-2]

cnames = colnames(df1)

cnames

#Loop to plot boxplot for each column#

#Explanation of below code:

#as outlier can be performed on numeric data only we are

#storing numeric features only

#storing required paramters to be passed in for graph

numeric\_index = sapply(df1,is.numeric)

numeric\_data = df1[,numeric\_index]

cnames = colnames(numeric\_data)

cnames

#Assign - will assign name to a parameter passed to it

#Ex : assign("fg",data.frame(missing\_val))

#will assign fg to dataframe

#inside assign we are giving random name

#so paste0 will join "gn" and i iteratively

#hence paste0 argument will become the name of reast of the object

#after ,

#In the below code we are storing each graph in name = gn"i"

for (i in 1:length(cnames))

{

assign(paste0("Graph",i), ggplot(aes\_string(y = (cnames[i]), x = "fare\_amount"), data = subset(df1))+

stat\_boxplot(geom = "errorbar", width = 0.5) +

geom\_boxplot(outlier.colour="red", fill = "blue" ,outlier.shape=18,

outlier.size=1, notch=FALSE) +

theme(legend.position="bottom")+

labs(y=cnames[i],x="Fare amount")+

ggtitle(paste("Box plot of Fare\_amount for",cnames[i])))

}

#Now we will plot all graphs together

#gridextra is library name for grid.arrange function

#grid.arrange function is arranging plots side by side

#Didn't include Graph1 because it is for fare amount

gridExtra::grid.arrange(Graph1,Graph2,ncol=2)

gridExtra::grid.arrange(Graph3,Graph4,ncol=2)

gridExtra::grid.arrange(Graph5,Graph6,ncol=2)

gridExtra::grid.arrange(Graph7,Graph8,Graph9,ncol=3)

gridExtra::grid.arrange(Graph10,ncol=1)

#Graphs are against fare\_amount at x axis

#Graphs are representing ooutliers in red color

#Except day time month year we have outliers in each column

###Outlier treatment##

#Remove outliers##

#Explanation of below code##

#Example:

# val = df1$pickup\_longitude[df1$pickup\_longitude %in% boxplot.stats(df1$pickup\_longitude)$out]

# In above code we are extracting outliers from graph

#%in% is used to search

#out function is to detect outlier

# After detecting indexes of outliers we have removed them

for (i in cnames) {

print(i)

val = df1[,i][df1[,i] %in% boxplot.stats(df1[,i])$out]

print(length(val))

df1 = df1[which(!df1[,i] %in% val),]

}

##Second method to remove outliers is replace with NA

#and use knnimputation to replace them

#Below is the code for same

#for (i in cnames) {

# print(i)

# val = df1[,i][df1[,i] %in% boxplot.stats(df1[,i])$out]

# print(length(val))

# df1[,i][df1[,i] %in% val] = NA

#

#}

#df1 = knnImputation(df1,k=3)

#we can use mean method as well if knn gives error because

#of neighbours

##############Feature Selection########

####Correlation Analysis########

library(corrgram)

corrgram(df1[,numeric\_index],order = F,

upper.panel = panel.pie,text.panel = panel.txt,

main="Correlation Plot")

str(df1)

#Blue color represents that variables are positively correlated

##Understanding

#pickup\_longitude and pickup\_latitude are highly correlated

#dropoff\_longitude and dropoff\_latitude are highly correlated

#We can drop one of above two observationss but we are keeping them for

#distance because distance is very important factor

##Chi square test will be applied for categorical variables

##We are not having that's why leaving it

####Dimension Reduction##

#df1 = subset(df1,select = -c(pickup\_latitude,dropoff\_latitude))

#str(df1)

######Feature Scaling#####

##Normalisation check##

##pickup\_longitude

library(qqplotr)

qqnorm(df1$pickup\_longitude)

hist(df1$pickup\_longitude)

##pickup\_latitude

qqnorm(df1$pickup\_latitude)

hist(df1$pickup\_latitude)

#dropoff\_longitude

qqnorm(df1$dropoff\_longitude)

hist(df1$dropoff\_longitude)

#dropoff\_latitude

qqnorm(df1$dropoff\_latitude)

hist(df1$dropoff\_latitude)

#passenger\_count

qqnorm(df1$passenger\_count)

hist(df1$passenger\_count)

#dteday

qqnorm(df1$day)

hist(df1$day)

#month

qqnorm(df1$month)

hist(df1$month)

#year

qqnorm(df1$year)

hist(df1$year)

############Modelling####################

##Sampling###

##Drop datetime from df1

str(df1)

train\_index = sample(1:nrow(df1), 0.8 \* nrow(df1))

train = df1[train\_index,]

test = df1[-train\_index,]

#######Dicision Tree #######

#library(C50)

##Explanation of code##

# We will use rpart for regression

library(rpart)

library(rpart.plot)

fit = rpart(fare\_amount ~ ., data = train, method = "anova")

str(test)

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

rpart.plot(fit,extra = 101)

##########Random forest#######

library(randomForest)

#Importance is to tell algorithm that show me the important variables

#and their calculation

#str(test)

model\_RF = randomForest(fare\_amount ~., train, importance = TRUE, ntree = 300)

predict\_RF = predict(model\_RF, test[,-1])

plot(model\_RF)

#########Linear regression###########

library(e1071)

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

summary(lm\_model)

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

##Explanation of summary of model

#Residuals means Errors

###Accuracy ##

#Dicision tree

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

#Random forest

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

#Linear regression

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

#Random forest has least MAPE so use that to predict

#########Model Tuning########

#check by increasing number of trees

#Tree = 500

#model\_RF2 = randomForest(fare\_amount ~., train, importance = TRUE, ntree = 500)

#predict\_RF2 = predict(model\_RF2, test[,-1])

#regr.eval(test[,1],predict\_RF2,stats = c('mae','rmse','mape','mse'))

#3.13 error %

#Another attempt

#Tree = 700

#model\_RF3 = randomForest(fare\_amount ~., train, importance = TRUE, ntree = 700)

#predict\_RF3 = predict(model\_RF3, test[,-1])

#regr.eval(test[,1],predict\_RF3,stats = c('mae','rmse','mape','mse'))

#Result is same

#Another attempt

#Number of trees = 1000

model\_RF4 = randomForest(fare\_amount ~., train, importance = TRUE, ntree = 1000)

predict\_RF4 = predict(model\_RF4, test[,-1])

regr.eval(test[,1],predict\_RF4,stats = c('mae','rmse','mape','mse'))

#Result = Minor change in error

#We will go with last attempt i.e. tree = 1000 commenting rest all

str(testDF)

testDF = testDF[,-1]

final\_predict\_RF = predict(model\_RF, testDF)

testDF$Predicted\_Fare = final\_predict\_RF

str(testDF)

#write file

write.csv(testDF,"Prediction.csv",row.names = FALSE)

rm(list=ls(all=T))

# Appendix B - Python code

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

################################################# Cab fare Prediction ############################################

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

############importing Libraries##########

import knnimpute

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import chi2\_contingency

import seaborn as sns

from random import randrange, uniform

import datetime as dt

#from sklearn.cross\_validation import train\_test\_split

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

from sklearn.tree import DecisionTreeRegressor

import statsmodels.api as sm

from sklearn.ensemble import RandomForestRegressor

from matplotlib import pyplot

############working directory#################

os.chdir("C:/Users/ASUS/Desktop/Edwisor Training/Project-3\_Cab\_fare/Python")

###########loading file####################

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

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

############exploratory data analysis#######################

####Type Conversion#####

cab\_data.columns

cab\_data['fare\_amount'].describe()

cab\_data.dtypes

cab\_data.shape

cab\_data.head(5)

########Splitting daytime to day month year########

##Training data##

#Day

d1=cab\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%d')

cab\_data['day']=d1

#Month

d1=cab\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%m')

cab\_data['month']=d1

#Year

d1=cab\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%Y')

cab\_data['year']=d1

#hour

d1=cab\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%H')

cab\_data['hour']=d1

#Minutes

d1=cab\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%M')

cab\_data['minutes']=d1

#Seconds

d1=cab\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%S')

cab\_data['Seconds']=d1

cab\_data.dtypes

#Date columns data type conversion

cab\_data['day'] = cab\_data['day'].astype('int')

cab\_data['month'] = cab\_data['month'].astype('int')

cab\_data['year'] = cab\_data['year'].astype('int')

#Time columns data type conversion

cab\_data['hour'] = cab\_data['hour'].astype('int')

cab\_data['minutes'] = cab\_data['minutes'].astype('int')

cab\_data['Seconds'] = cab\_data['Seconds'].astype('int')

cab\_data.dtypes

#Drop datetime as it is of no use now

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

##Test data##

#Day

d1=test\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%d')

test\_data['day']=d1

#Month

d1=test\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%m')

test\_data['month']=d1

#Year

d1=test\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%Y')

test\_data['year']=d1

#hour

d1=test\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%H')

test\_data['hour']=d1

#Minutes

d1=test\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%M')

test\_data['minutes']=d1

#Seconds

d1=test\_data['pickup\_datetime'].copy()

for i in range (0,d1.shape[0]):

d1[i]=dt.datetime.strptime(d1[i], "%Y-%m-%d %H:%M:%S UTC").strftime('%S')

test\_data['Seconds']=d1

test\_data.dtypes

#Date columns data type conversion

test\_data['day'] = test\_data['day'].astype('int')

test\_data['month'] = test\_data['month'].astype('int')

test\_data['year'] = test\_data['year'].astype('int')

#Time columns data type conversion

test\_data['hour'] = test\_data['hour'].astype('int')

test\_data['minutes'] = test\_data['minutes'].astype('int')

test\_data['Seconds'] = test\_data['Seconds'].astype('int')

test\_data.dtypes

#Drop datetime as it is of no use now

test\_data = test\_data.drop(['pickup\_datetime'], axis=1)

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

############Missing value analysis#################

##Checking Null values#

resp = cab\_data.isnull().values.any()

print("Missing values in training data :",resp)

#Yes there are missing values in train data

resp = test\_data.isnull().values.any()

print("Missing values in test data :",resp)

#There are no missing values in test data

#Let's go ahead with training dta

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

missing\_val

#reset index

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

missing\_val

#Rename varibale

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

missing\_val

#calculate percentage

missing\_val['Missing\_percentage']=(missing\_val['Missing\_percentage']/len(cab\_data))\*100

missing\_val

#save output

missing\_val.to\_csv("missingInTrain.csv",index=False)

#Missing value treatment

#Removing all missing values

cab\_data = cab\_data.drop(cab\_data[cab\_data.isnull().any(1)].index, axis = 0)

resp = test\_data.isnull().values.any()

print("Missing values in test data :",resp)

#No missing values now

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

cab\_data.dtypes

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

plt.show()

#we can see outliers here

plt.boxplot(cab\_data['pickup\_longitude'])

plt.show()

#we have already plotted in R now let us handle it here with code

#storing numerical columns

cname = cab\_data.columns

#cname = cname.drop('pickup\_datetime')

#cname = cname.drop('time')

for i in cname:

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

iqr = q75 - q25

min = q25 - (iqr\*1.5)

max = q75 + (iqr\*1.5)

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

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

#Have removed all outliers

########Feature Selection########

##Correlation Analysis##

cname = cab\_data.columns

#correlation plot

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

#set width and height of plot

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

#Generate correlation matrix

corr = df\_corr.corr()

#PLot using seaborn

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

square=True,ax=ax)

plt.show()

#########Modelling##########

#divide data into train and test

cname

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

#Dicision tree for regression

#max depth = 2 means we are limiting depth of model

#maximum node to a leaf is 2

fit = DecisionTreeRegressor().fit(train.iloc[:,1:12], train.iloc[:,0])

#Apply model on test data

#fit

predict\_DT = fit.predict(test.iloc[:,1:12])

#predict\_DT

#Calculate MAPE

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

mape = np.mean(np.abs((y\_true - y\_pred) / y\_true))

return mape

MAPE(test.iloc[:,0], predict\_DT)

#0.33011

#Random forest regressor

RFmodel = RandomForestRegressor(n\_estimators = 1000).fit(train.iloc[:,1:12], train.iloc[:,0])

RF\_Predictions = RFmodel.predict(test.iloc[:,1:12])

MAPE(test.iloc[:,0], RF\_Predictions)

#MAPE = 0.24

#Linear Regression

model = sm.OLS(train.iloc[:,0], train.iloc[:,1:12]).fit()

predictions\_LR = model.predict(test.iloc[:,1:12])

MAPE(test.iloc[:,0], predictions\_LR)

#MAPE = 0.53

#Least is 0.24 for RF

#We will make final prediction by RF model

#test\_data.dtypes

result=pd.DataFrame(test\_data.iloc[:,0:12])

RF\_Predictions = RFmodel.predict(test\_data.iloc[:,0:12])

result['Predicted\_Fare'] = (RF\_Predictions)

result.to\_csv("Predicted\_fare\_RF.csv",index=False)

print("\*\*\*\*\*File has been created\*\*\*\*")

# References

https://www.datasciencecentral.com/profiles/blogs/implemetation-of-17-classification-algorithms-in-r

https://data-flair.training/blogs/classification-in-r/

http://r-statistics.co/Linear-Regression.html

https://hackernoon.com/choosing-the-right-machine-learning-algorithm-68126944ce1f - Very Effective for algorithms summary

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*