PREDICATION OF FARE AMOUNT FOR A CAB RIDE

Satyam Sharma 23-07-2019

Contents

Chapter 1 2
Introduction 2
1.1 Problem Statement 2
1.2 Data 2
Chapter 2 4
Methodology 4
2.1 Pre-processing 4
2.2 Modelling 12
Chapter 3 14
Conclusion 14
3.1 Model Evaluation 14
3.2 Model Tuning 14
3.2 Model Selection 14
Graphs 15
Prediction File 15
Appendix A - R code 15
Appendix B - Python code 29
References 39
********** 39

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:

missing_perc_test_data.csv missing_perce_Traindata.csv

OBSERVATION:

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

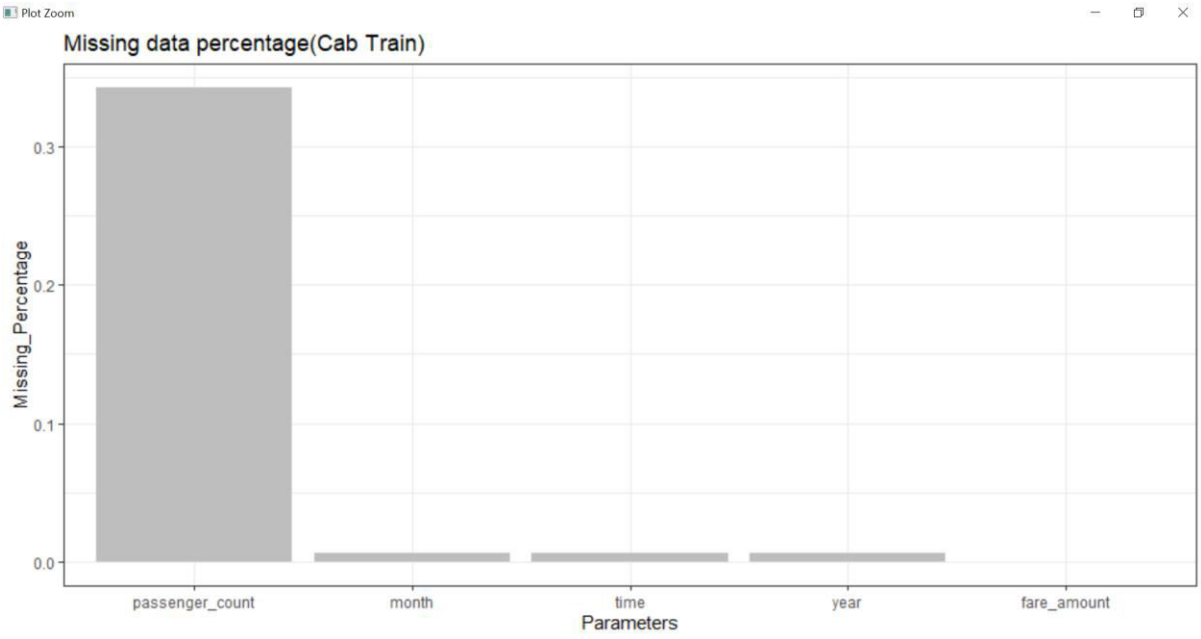
```
fare_amount    pickup_datetime  pickup_longitude    pickup_latitude
                  0                 0                   0                 0


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

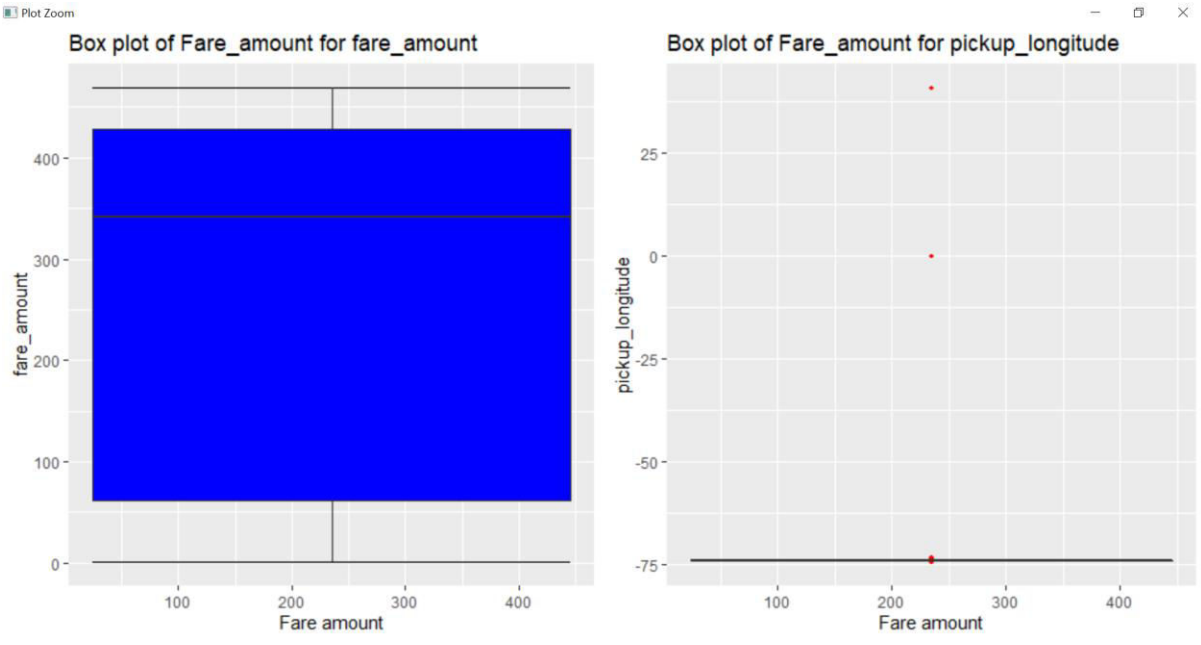
Function that fills in all NA values using the k Nearest Neighbours of each case with NA values. By default it uses the values of the neighbours and obtains an weighted (by the distance to the case) average of their values to fill in the unknows. If meth='median' it uses the median/most frequent value,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

2.1.3 Outlier analysis

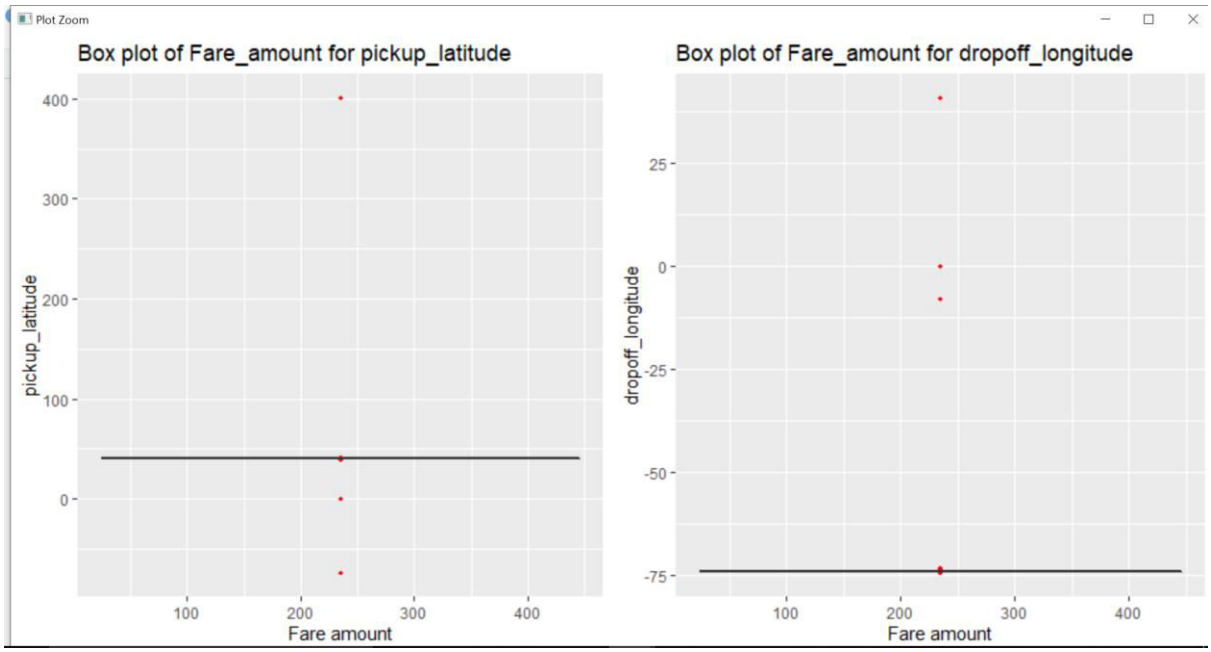
we have to select only numeric data for outlier analysis. In our case we have all numeric data.

BOXPLOT:

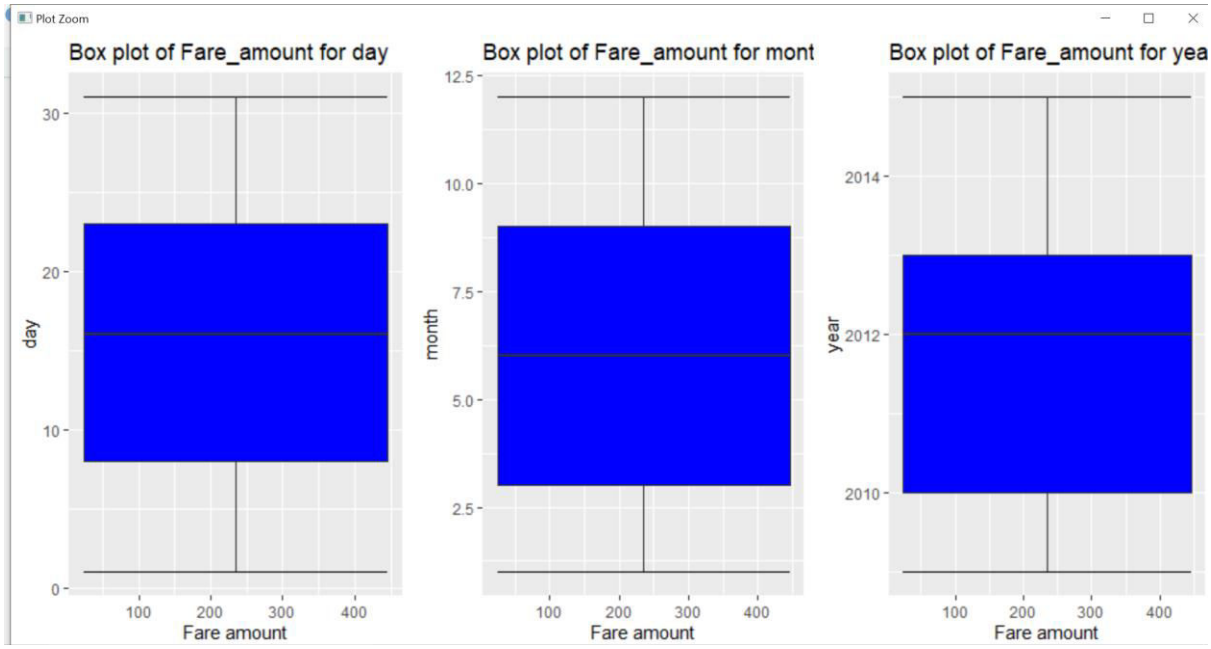
Fare amount vs fare amount & fare amount vs pickup longitude



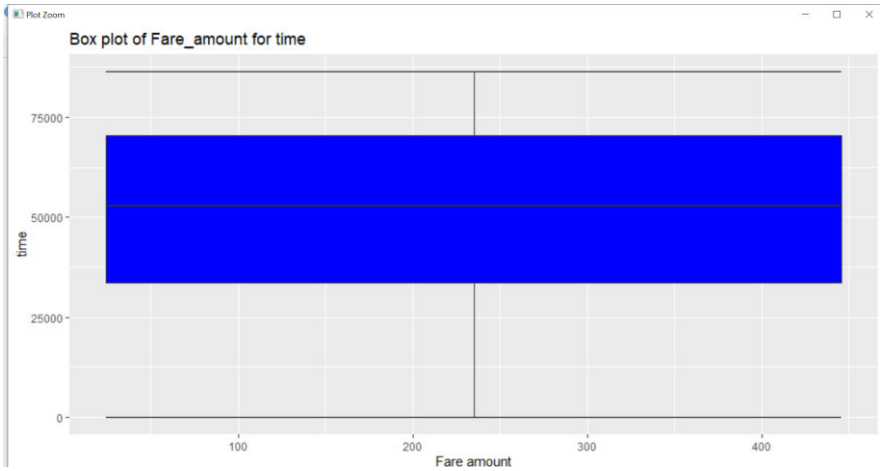
Fare amount vs dropoff longitude & fare amount vs pickup latitude



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:

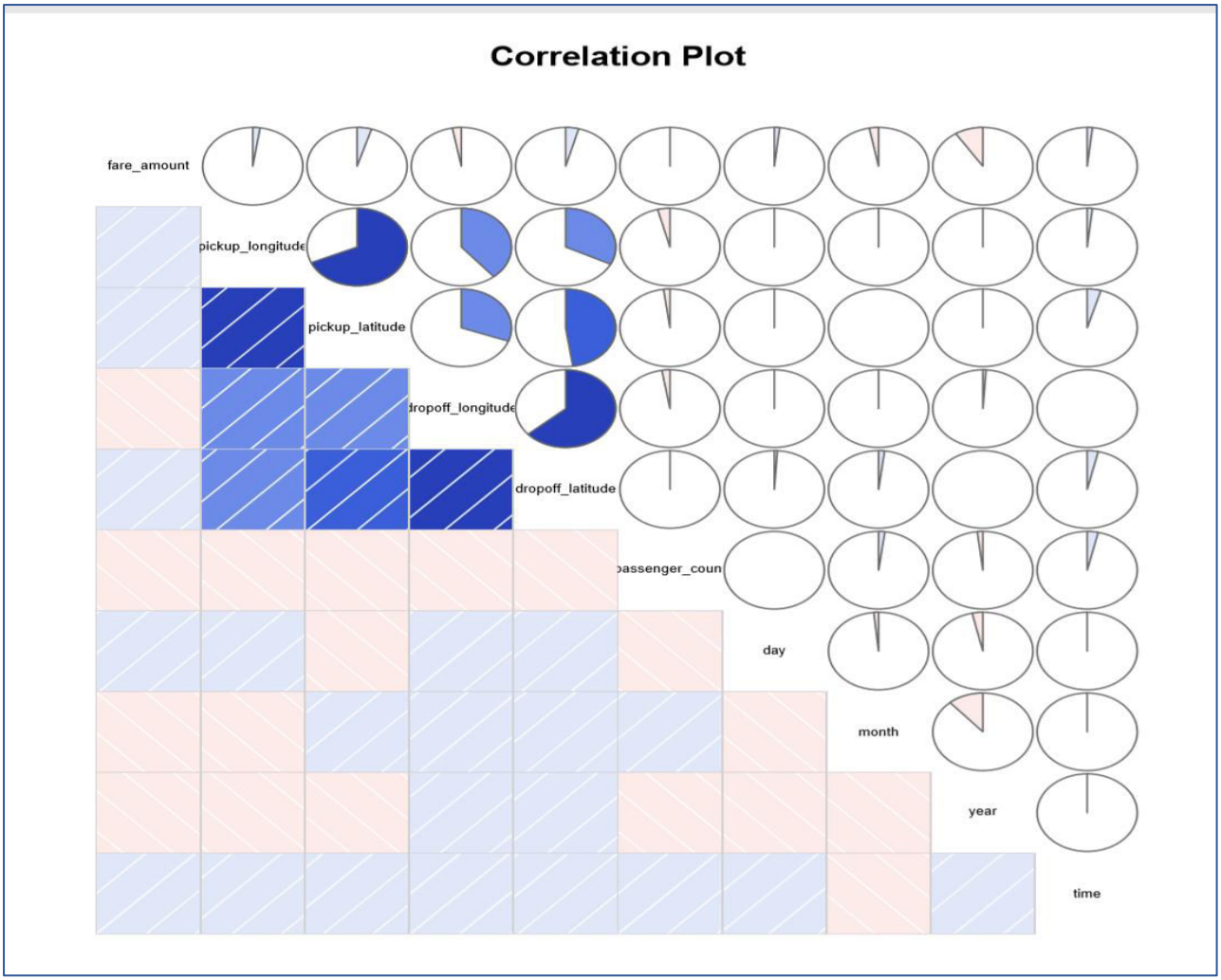
We have removed outliers from our data and below is the report of outliers treated:

[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.

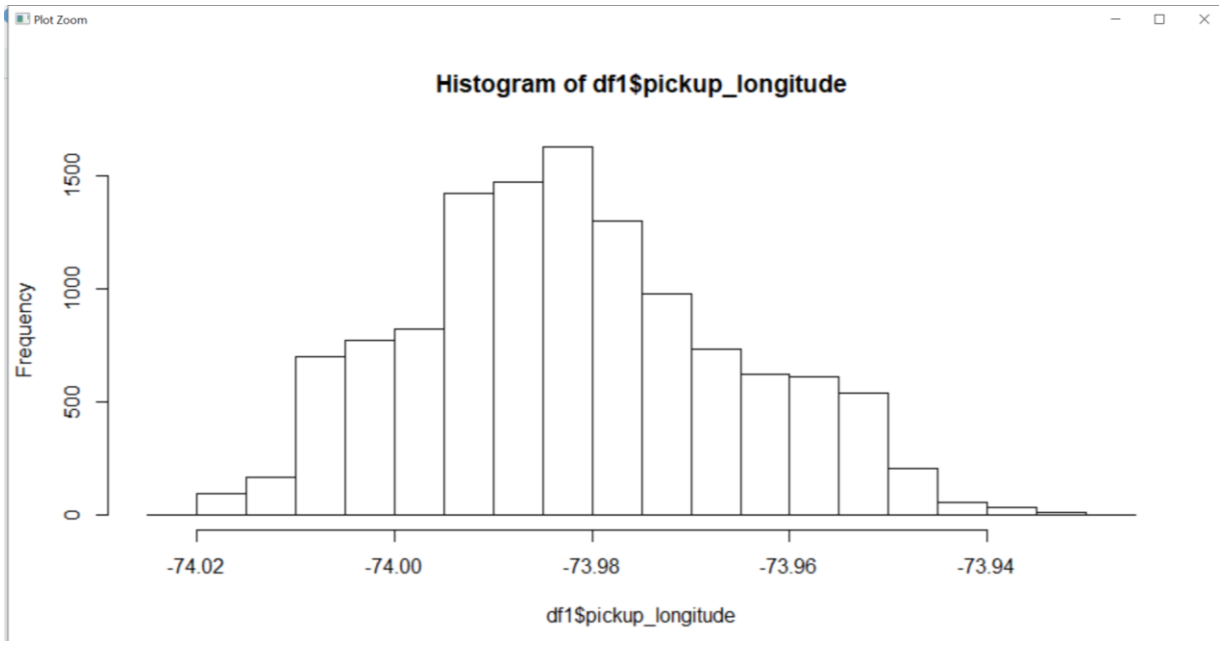
Knowledge: Chi square test will be applied for categorical variables We are not having that's why leaving it.

2.1.3 Feature Scaling

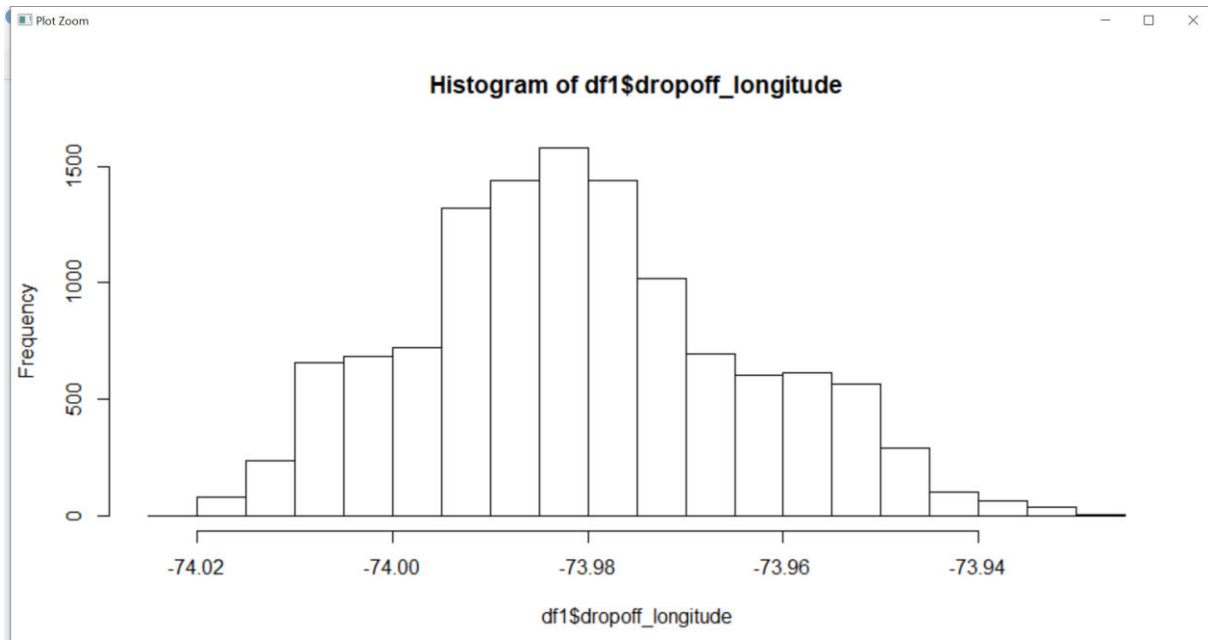
NORMALIZATION CHECK

Histograms:

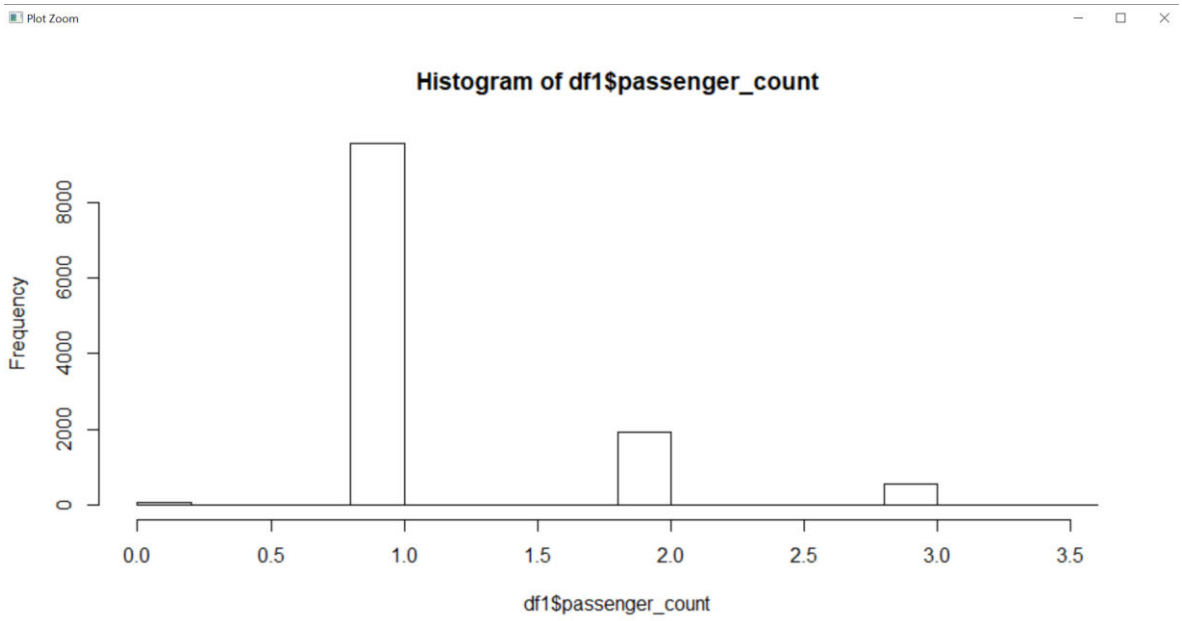
Pickup_longitude



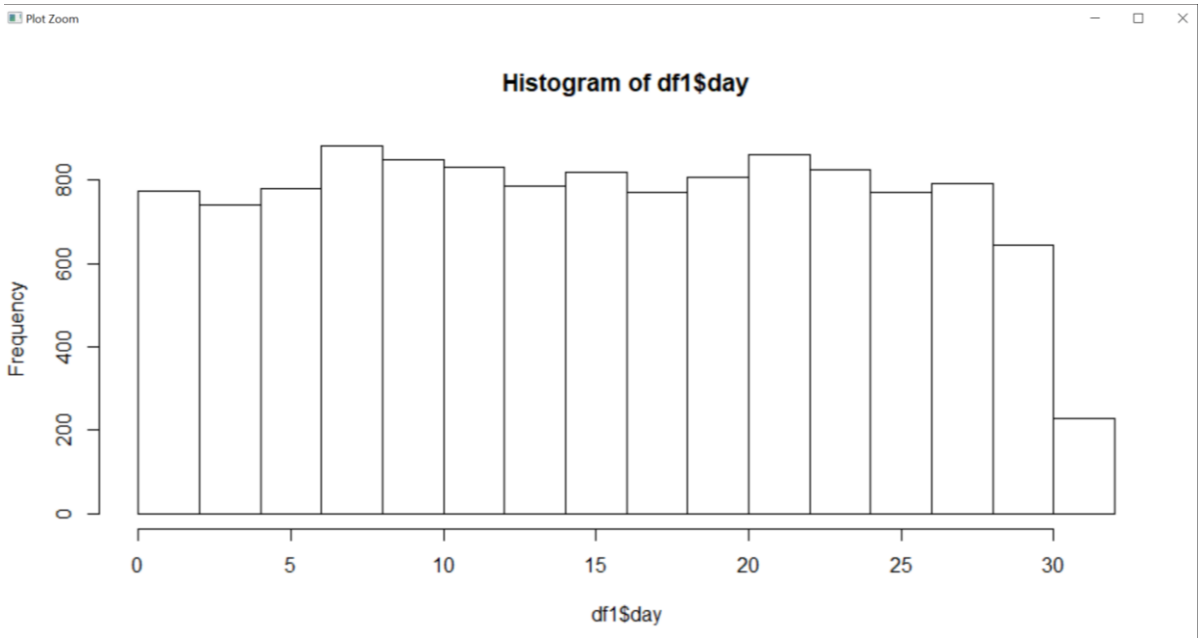
DROPOFF_LONGITUDE



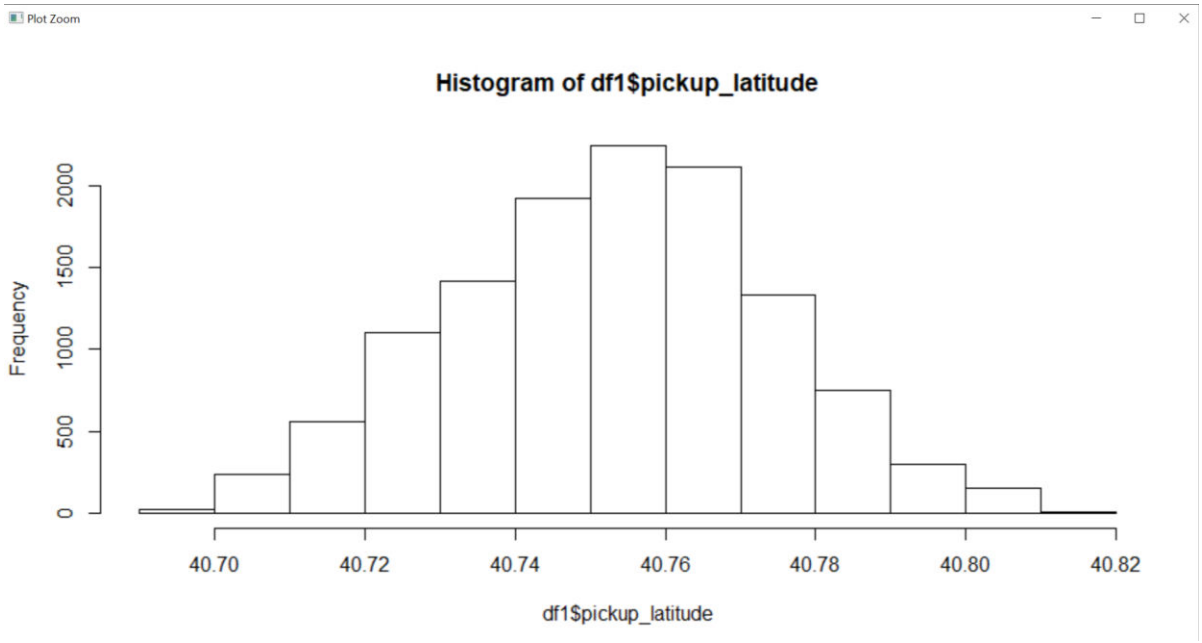
passenger_count



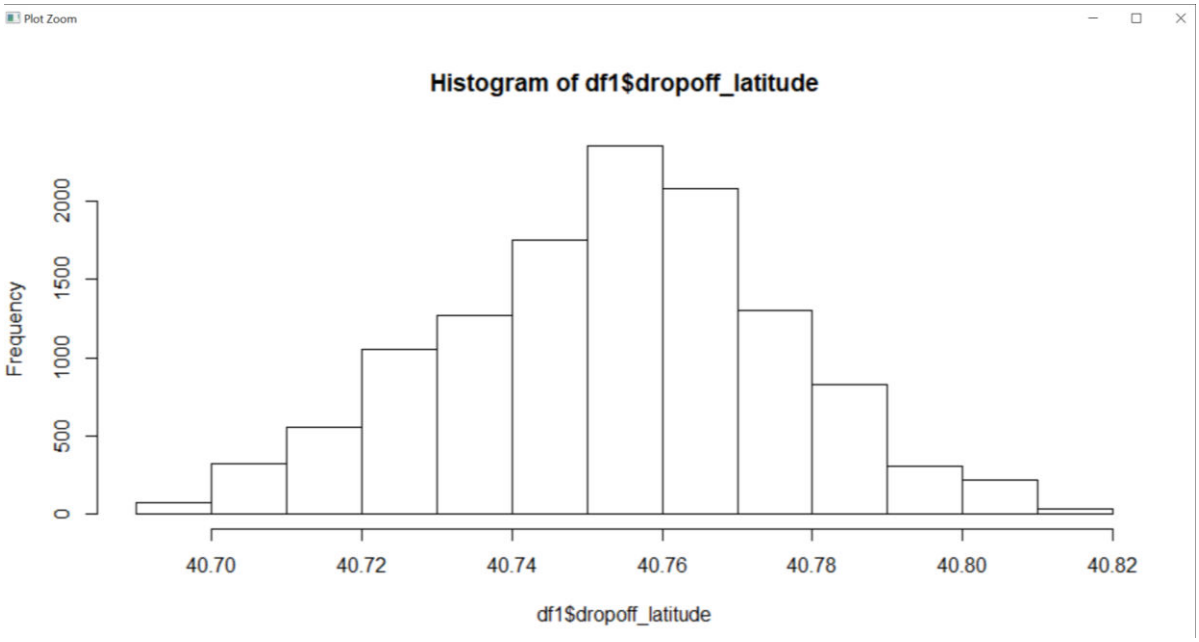
DAY



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.

```
> ###Accuracy ##
> #Dicision tree
> regr.eval(test[,1],predict_DT,stats = c('mae','rmse','mape','mse'))
         mae         rmse         mape          mse
  132.447624   161.250843     3.937815 26001.834319
>
> #Random forest
> regr.eval(test[,1],predict_RF,stats = c('mae','rmse','mape','mse'))
         mae         rmse         mape          mse
  111.459669   138.273893     3.308601 19119.669542
>
> #Linear regression
> regr.eval(test[,1],predict_LM,stats = c('mae','rmse','mape','mse'))
        mae        rmse        mape         mse
  155.80312   172.87739     4.13754 29886.59235
>
```

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.

Rplots.pdf

Prediction File

Prediction.csv

R code file

cab_fare.R

Python File

Cab_fare.py

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

****