rm(list=ls())

setwd("C:/Users/17519")

getwd()

### INSTALLING THE REQUIRED LIBRARIES##

install.packages (c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information","MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees'))

### loading the required data ##

## train data

train=read.csv("train\_cab.csv",header=TRUE)

str(train)

summary(train)

### load test data

test=read.csv("test.csv",header=TRUE)

str(test)

summary(test)

## lets check few obesrvations

head(train,10)

head(test,10)

### in train data it is observed that

##data types need to be converted into proper format

train$passenger\_count=round(train$passenger\_count)

train$fare\_amount = as.numeric(as.character(train$fare\_amount))

####### data exploring and cleaning #######

#1. passenger\_count

# this can not be less than 1 , and cannot be more than 6.

train[which(train$passenger\_count < 1 ),] #this will show all values less than 1

nrow(train[which(train$passenger\_count < 1 ),]) ## actual count of values less than 1

train=train[-which(train$passenger\_count < 1 ),] # removed values less than 1

train[which(train$passenger\_count > 6),] # this will show values more than 6

nrow(train[which(train$passenger\_count > 6),])

train=train[-which(train$passenger\_count > 6),] ## removed values more than 6

#2. lattitude and longitude values

## lattitude must be in range (-90 to 90), and longitude in (-180 to 180)

print(paste('pickup\_latitude above 90=',nrow(train[which(train$pickup\_latitude > 90 ),])))

print(paste('pickup\_latitude above -90=',nrow(train[which(train$pickup\_latitude < -90 ),])))

## here is a value in pickup\_lattitude > 90, lets remove it

train = train[-which(train$pickup\_latitude > 90),]

print(paste('pickup\_longitude above 180=',nrow(train[which(train$pickup\_longitude >180 ),])))

print(paste('pickup\_longitude above -180=',nrow(train[which(train$pickup\_longitude < -180 ),])))

## dropoff co-ordinates

print(paste('dropoff\_latitude above -90=',nrow(train[which(train$dropoff\_latitude < -90 ),])))

print(paste('dropoff\_latitude above 90=',nrow(train[which(train$dropoff\_latitude > 90 ),])))

print(paste('dropoff\_longitude above 180=',nrow(train[which(train$dropoff\_longitude > 180 ),])))

print(paste('dropoff\_longitude above -180=',nrow(train[which(train$dropoff\_longitude < -180 ),])))

### no errors found here

train=train[-which(train$pickup\_longitude == 0 ),]## removing error values

train=train[-which(train$dropoff\_longitude == 0),]

#3. fare amount cannot be 0, negative or less than 1

## also it is observed that above 453 some values are very high so removing them

train[which(train$fare\_amount < 1 ),] ## number of values less than 1

nrow(train[which(train$fare\_amount < 1 ),]) ## actual count

train = train[-which(train$fare\_amount < 1 ),] ## removing 5 error values

## fare amount not more than 453

train[which(train$fare\_amount>453),]

nrow(train[which(train$fare\_amount >453 ),]) # actual count

train = train[-which(train$fare\_amount >453 ),] ## removing the 2 values

## checking the null values in datasets

sum(is.na(train))## 77 na values

sum(is.na(test))## here no error values present

train=na.omit(train) ## removed na values in train

sum(is.na(train))

#4. the pickup\_datetime column will be properly formatted using "strptime"

train$pickup\_datetime=as.Date(train$pickup\_datetime)## converted into date from factor

pickuptime = strptime(train$pickup\_datetime,format='%Y-%m-%d %H:%M:%S UTC')## use of "strptime"

train$year = as.integer(format(train$pickup\_date,"%Y"))## creating year value

train$month = as.integer(format(train$pickup\_date,"%m"))# creating month value

train$date = as.integer(format(train$pickup\_date,"%d")) # creating date value

## for test data similar operations wil be performed

test$pickup\_datetime=as.Date(test$pickup\_datetime)

pickuptime = strptime(test$pickup\_datetime,format='%Y-%m-%d %H:%M:%S UTC')

test$year = as.integer(format(test$pickup\_date,"%Y"))

test$month = as.integer(format(test$pickup\_date,"%m"))

test$date = as.integer(format(test$pickup\_date,"%d"))

## plotting a boxplot for outliers detection

library(ggplot2)

plot = ggplot(train,aes(x = factor(passenger\_count),y = fare\_amount))

plot + geom\_boxplot(outlier.colour="blue", fill = "yellow" ,outlier.shape=18,outlier.size=1, notch=FALSE)+ylim(0,100)

## we use haversine formula for for distance calculation from the co ordinates

deg\_to\_rad = function(deg){

(deg \* pi) / 180

}

haversine = function(long1,lat1,long2,lat2){

#pickup\_long = deg\_to\_rad(long1)

phi1 = deg\_to\_rad(lat1)

#dropoff\_long = deg\_to\_rad(long2)

phi2 = deg\_to\_rad(lat2)

delphi = deg\_to\_rad(lat2 - lat1)

dellamda = deg\_to\_rad(long2 - long1)

a = sin(delphi/2) \* sin(delphi/2) + cos(phi1) \* cos(phi2) \*

sin(dellamda/2) \* sin(dellamda/2)

c = 2 \* atan2(sqrt(a),sqrt(1-a))

R = 6371e3 ##km earth's radius

R \* c / 1000

}

## creating new value distance using haversine formula in test and train data

## from given co-ordinates

train$distance = haversine(train$pickup\_longitude,train$pickup\_latitude,train$dropoff\_longitude,train$dropoff\_latitude)

## distance in train data

## similar procedure in test data

test$distance = haversine(test$pickup\_longitude,test$pickup\_latitude,test$dropoff\_longitude,test$dropoff\_latitude)

## now we remove the co ordinate values used to create distance variable

train = subset(train,select = -c(pickup\_longitude,pickup\_latitude,dropoff\_longitude,dropoff\_latitude,pickup\_datetime))

test = subset(test,select = -c(pickup\_longitude,pickup\_latitude,dropoff\_longitude,dropoff\_latitude,pickup\_datetime))

str(train)

head(train,10)

summary(train)## shows maximum and minimum distances have errors

nrow(train[which(train$distance ==0 ),])## minimum distance can't be 0. 155 values

nrow(test[which(test$distance==0 ),])## in test 85 values

train=train[-which(train$distance ==0 ),]## removing 155 values

test=test[-which(test$distance ==0 ),] ## removing 85 values

## the distance after a point 128km became too high so considered it outlier

## 129 will be the max distance limit set for analysis

nrow(train[which(train$distance >129 ),])## 129 is the limit 4 values are above it

nrow(test[which(test$distance >129 ),])## no errors in test$distance

train=train[-which(train$distance >129 ),]## removing 4 values

##feature selection

numeric = sapply(train,is.numeric) ## selects only numeric

data\_1 = train[,numeric]

cnames = colnames(data\_1)

#plotting a correlation plot for numeric data

library(corrgram)

corrgram(train[,numeric],upper.panel=panel.pie, main = "Correlation Plot")

#removing date it has p value greater than 0.05

train = subset(train,select=-date)## removing from train data

test = subset(test,select=-date)#removing from test data

## feature scaling ##

library(MASS)

truehist(train$fare\_amount) # truehist() scales the counts to give an estimate of the probability density.

lines(density(train$fare\_amount)) # lines() and density() functions to overlay a density plot on histogram

A=density(train$fare\_amount)

plot(A,main="distribution")

polygon(A,col="yellow",border="red")

B=density(train$distance)

plot(B,main="distribution")

polygon(B,col="blue",border="yellow")

C=density(test$distance)

plot(C,main="distribution")

polygon(C,col="green",border="red")

#Normalisation

# log transformation.

train$fare\_amount=log1p(train$fare\_amount)

test$distance=log1p(test$distance)

train$distance=log1p(train$distance)

# checking back features after transformation.

A=density(train$fare\_amount)

plot(A,main="distribution")

polygon(A,col="red",border="green")

B=density(train$distance)

plot(B,main="distribution")

polygon(B,col="yellow",border="blue")

C=density(test$distance)

plot(C,main="distribution")

polygon(C,col="red",border="blue")

###check multicollearity

install.packages("usdm")

library(usdm)

vif(train[,-1])

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

sum(is.na(train))

train=na.omit(train)

str(train)

summary(train)

# model building

# preparing the data

set.seed(1400)

Train.index = sample(1:nrow(train), 0.8 \* nrow(train))

Train = train[ Train.index,]

Test = train[-Train.index,]

TestData=test

# linear regression

linear\_regr=lm(fare\_amount~.,data=Train)

summary(linear\_regr)

predict\_lm=predict(linear\_regr,Test[,2:5])

predict\_test=predict(linear\_regr,TestData)

library(DMwR)

regr.eval(Test[,1],predict\_lm)

# decision tree regressor

library(rpart)

D\_Tree=rpart(fare\_amount~.,data=Train,method="anova")

predictions\_dt=predict(D\_Tree,Test[,2:5])

predictions\_test=predict(D\_Tree,TestData)

summary(D\_Tree)

regr.eval(Test[,1],predictions\_dt)

# random forest regressor

library(randomForest)

r\_forest = randomForest(fare\_amount~ ., Train, importance = TRUE, ntree = 500)

#Extract rules fromn random forest

#transform rf /object to an inTrees' format

library(inTrees)

tree\_size = RF2List(r\_forest)

#Extract rules

rules= extractRules(tree\_size, Train[,2:5])

#Visualize some rules

rules[1:2,]

#Make rules more readable:

read\_rule\_1 = presentRules(rules, colnames(Train))

read\_rule\_1[1:2,]

#Predict test data using random forest model

Prediction\_Rf = predict(r\_forest, Test[,2:5])

regr.eval(Test[,1],Prediction\_Rf)

### hence we choose the random forest model as it has the best criterion for our data

test\_rf=predict(r\_forest, TestData)

# saving the results in hard disk

write(capture.output(summary(r\_forest)),"R\_Forest1.txt")