Cab Fare Prediction# R_code

```
# Cab Fare Prediction

rm(list = ls())
setwd("C:/Users/Poo/Documents/edWisor/online project no 01")
# #loading Libraries
install.packages(c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
            "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','xgboost','stats'))
#load Packages
library(e1071, warn.conflicts = FALSE)
library(seriation)
library(lattice)
library(grid)
library(sp)
library(raster)
library(DataCombine, warn.conflicts = FALSE)
library(foreach)
library(iterators)
library(snow)
library(randomForest, warn.conflicts = FALSE)
library(inTrees, warn.conflicts = FALSE)
library(inTrees, warn.conflicts = FALSE)
lapply(require, character.only = TRUE)
# The details of data attributes in the dataset are as follows:
# 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.


# loading datasets
train = read.csv("train_cab_01.csv", header = T, na.strings = c(" ", "", "NA"))
test = read.csv("test_01.csv")
#test_pickup_datetime = test["pickup_datetime"]

# Structure of data
str(train)
str(test)
summary(train)
summary(test)
head(train,5)
head(test,5)

#############                    Exploratory Data Analysis                ########################
# Changing the data types of variables
#first for train dataset
train$pickup_datetime <- gsub('\\ UTC','',train$pickup_datetime)
#Splitting Date and time
```

```
train$Date <- as.Date(train$pickup_datetime)
train$Year <- substr(as.character(train$Date),1,4)
train$Month <- substr(as.character(train$Date),6,7)
train$Weekday <- weekdays(as.POSIXct(train$Date), abbreviate = F)
train$Date <- substr(as.character(train$Date),9,10)
train$Time <- substr(as.factor(train$pickup_datetime),12,13)
train= subset(train, select = -c(pickup_datetime))

# for test data set
test$pickup_datetime <- gsub('\\ UTC','',test$pickup_datetime)
#Splitting Date and time
test$Date <- as.Date(test$pickup_datetime)
test$Year <- substr(as.character(test$Date),1,4)
test$Month <- substr(as.character(test$Date),6,7)
test$Weekday <- weekdays(as.POSIXct(test$Date), abbreviate = F)
test$Date <- substr(as.character(test$Date),9,10)
test$Time <- substr(as.factor(test$pickup_datetime),12,13)
test = subset(test, select = -c(pickup_datetime))


train$fare_amount = as.numeric(as.character(train$fare_amount))

train$passenger_count=round(train$passenger_count)

### Removing values which are not within desired range(outlier) depending upon basic
understanding of dataset.

# 1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve and
also cannot be 0. So we will remove these fields.
train[which(train$fare_amount < 1 ),]
nrow(train[which(train$fare_amount < 1 ),])
train = train[-which(train$fare_amount < 1 ),]

#2.Passenger_count variable
train[which(train$passenger_count > 6 ),]
# Also we need to see if there are any passenger_count==0
train[which(train$passenger_count <1 ),]
nrow(train[which(train$passenger_count <1 ),])
# We will remove these 58 observations and 20 observation which are above 6 value because a cab
cannot hold these number of passengers.
train = train[-which(train$passenger_count < 1 ),]
train = train[-which(train$passenger_count > 6),]

# 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.Removing which does not
satisfy these ranges
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 ),])))
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 ),])))
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 ),])))
```

```
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 ),])))
# There's only one outlier which is in variable pickup_latitude.So we will remove it with nan.
# Also we will see if there are any values equal to 0.
nrow(train[which(train$pickup_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
nrow(train[which(train$dropoff_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
# there are values which are equal to 0. we will remove them.
train = train[-which(train$pickup_latitude > 90),]
train = train[-which(train$pickup_longitude == 0),]
train = train[-which(train$dropoff_longitude == 0),]

##### Function to calculate distance ######
lat1 = train['pickup_latitude']
lat2 = train['dropoff_latitude']
long1 = train['pickup_longitude']
long2 = train['dropoff_longitude']

gcd_hf <- function(long1, lat1, long2, lat2) {
  R <- 6371.145 # Earth mean radius [km]
  delta.long <- (long2 - long1)
  delta.lat <- (lat2 - lat1)
  a <- sin(delta.lat/2)^2 + cos(lat1) * cos(lat2) * sin(delta.long/2)^2
  c <- 2 * atan2(sqrt(a),sqrt(1-a))
  d = R * c
  return(d) # Distance in km
}

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

#train = subset(train, select = -c(pickup_datetime))

#same we will do for test dataset
lat11 = test['pickup_latitude']
lat12= test['dropoff_latitude']
long11 = test['pickup_longitude']
long12 = test['dropoff_longitude']

gcd_hf1 <- function(long11, lat11, long12, lat12) {
  R <- 6371.145 # Earth mean radius [km]
  delta.long1 <- (long12 - long11)
  delta.lat1 <- (lat12 - lat11)
  A <- sin(delta.lat1/2)^2 + cos(lat11) * cos(lat12) * sin(delta.long1/2)^2
  C <- 2 * atan2(sqrt(A),sqrt(1-A))
  D = R * C
```

```
  return(D) # Distance in km
}
for (i in 1:nrow(test))
{
  test$distance[i]= gcd_hf1(test$pickup_longitude[i],
test$pickup_latitude[i],test$dropoff_longitude[i],
                        test$dropoff_latitude[i])
}


train = subset(train, select = -c(Weekday))
test = subset(test, select = -c(Weekday))
# Make a copy
new=train
train=new

#############                    Missing Value Analysis               #############
missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(train)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
missing_val


unique(train$passenger_count)
unique(test$passenger_count)
train[,'passenger_count'] = factor(train[,'passenger_count'], labels=(1:6))
test[,'passenger_count'] = factor(test[,'passenger_count'], labels=(1:6))

train$Date<-as.numeric(as.character(train$Date))
train$Year<-as.numeric(as.character(train$Year))
train$Month<-as.numeric(as.character(train$Month))
train$Time<-as.numeric(as.character(train$Time))
train$passenger_count<-as.numeric(as.factor(train$passenger_count))
train$fare_amount<-as.numeric(as.factor(train$fare_amount))
str(train)

# 1.For Passenger_count:
# Actual value = 1
# Mode = 1
# KNN = 1
train$passenger_count[1000]
train$passenger_count[1000] = NA
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}
```

```
# Mode Method
getmode(train$passenger_count)
# We can't use mode method because data will be more biased towards passenger_count=1
train$passenger_count[is.na(train$passenger_count)] =median(train$passenger_count,
na.rm=TRUE)
# 2.For fare_amount:
# Actual value = 108,
# Mean = 64.34,
# Median = 8.5,
# KNN = 18.28
sapply(train, sd, na.rm = FALSE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.968236      4635.700531         2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
# 2.613305         2.710835         2.632400
# passenger_count
# 1.266104
train$fare_amount[1000]
train$fare_amount[1000]= NA

# Mean Method
mean(train$fare_amount, na.rm = T)

#Median Method
median(train$fare_amount, na.rm = T)

# kNN Imputation
train = knnImputation(train, k = 181)
train$fare_amount[1000]
train$passenger_count[1000]
sapply(train, sd, na.rm = TRUE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.661952      4635.700531         2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
# 2.613305         2.710835         2.632400
# passenger_count
# 1.263859
sum(is.na(train))
str(train)
summary(train)

new1=train
train=new1
#####################                    Outlier Analysis              #################

# We Will do Outlier Analysis only on Fare_amount just for now and we will do outlier analysis after
feature engineering laitudes and longitudes.
# Boxplot for fare_amount
pl1 = ggplot(train,aes(x = factor(passenger_count),y = fare_amount))
pl1 + geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1,
notch=FALSE)+ylim(0,100)
```

```
# Replace all outliers with NA and impute
vals = train[,"fare_amount"] %in% boxplot.stats(train[,"fare_amount"])$out

train[which(vals),"fare_amount"] = NA

#lets check the NA's
sum(is.na(train$fare_amount))

#Imputing with KNN
train = knnImputation(train,k=3)
#train = train[-which(train$fare_amount >93.30 ),]
#train = train[-which(train$distance >93.73 ),]

# lets check the missing values
sum(is.na(train$fare_amount))
sum(is.na(train))

#train <- DropNA(train)
str(train)
#new2=train
train = subset(train, select = -
c(pickup_latitude,dropoff_latitude,pickup_longitude,dropoff_longitude))
test= subset(test, select = -c(pickup_latitude,dropoff_latitude,pickup_longitude,dropoff_longitude))
new2=train
#train=new2


####################          Feature Scaling       ######################################
#Normality check
#view Data before Normalisation
new3=train
head(new3)
signedlog10 = function(x) {
  ifelse(abs(x) <= 1, 0, sign(x)*log10(abs(x)))
}
new3$fare_amount = signedlog10(new3$fare_amount)
head(new3$fare_amount)
new3$distance = signedlog10(new3$distance)

test$Date<-as.numeric(as.character(test$Date))
test$Year<-as.numeric(as.character(test$Year))
test$Month<-as.numeric(as.character(test$Month))
test$Time<-as.numeric(as.character(test$Time))
test$passenger_count<-as.numeric(as.factor(test$passenger_count))
test = subset(test, select = -c(Weekday))

#test$distance = signedlog10(test$distance)

##checking distribution
hist(new3$fare_amount,col = "blue")
```

```
hist(new3$distance,col ="green")

#Normalization
cont=c("fare_amount","passenger_count","distance")
for(i in cont)
{
  print(i)
  new3[,i] = (new3[,i] - min(new3[,i]))/(max(new3[,i])-min(new3[,i]))
}

hist(new3$distance,col="green")
hist(new3$fare_amount,col="blue")
#Viewing data after Normalization
head(train)

################### Splitting train into train and validation subsets ###################
#install.packages('caret')
library(caret)
#train1=train = subset(train, select = -c(fare_amount))
set.seed(1000)
tr.idx = createDataPartition(train$fare_amount,p=0.75,list = FALSE) # 75% in trainin and 25% in
Validation Datasets
train_data = train[tr.idx,]
test_data = train[-tr.idx,]


##################Model Selection###############
#Error metric used to select model is RMSE

#############          Linear regression            #################
lm_model = lm(fare_amount ~.,data=train_data)

summary(lm_model)
str(train_data)

lm_predictions = predict(lm_model,test_data)

regr.eval(test_data[,1],lm_predictions)
# mae        mse       rmse       mape
# 3.5303114 19.3079726  4.3940838  0.4510407


#############                        Decision Tree          ####################

Dt_model = rpart(fare_amount ~ ., data = train_data, method = "anova")

summary(Dt_model)
#Predict for new test cases
predictions_DT = predict(Dt_model, test_data)

regr.eval(test_data[,1],predictions_DT)
```

```
# mae       mse      rmse      mape
# 1.8981592 6.7034713 2.5891063 0.2241461


#############                   Random forest          ####################
rf_model = randomForest(fare_amount ~.,data=train_data)

summary(rf_model)

rf_predictions = predict(rf_model,test_data)

regr.eval(test_data[,1],rf_predictions)
# mae       mse      rmse      mape
# 1.9053850 6.3682283 2.5235349 0.2335395


#############          Finalizing and Saving Model for later use                ####################

rf_model2 = randomForest(fare_amount ~.,data=train_data)

# Saving the trained model
saveRDS(rf_model2, "./final_GRF_model_using_R.rds")

# loading the saved model
super_model <- readRDS("./final_GRF_model_using_R.rds")
print(super_model)

# Lets now predict on test dataset
grf = predict(super_model,test_data)

#grf_pred = data.frame(test_distance,"predictions" = grf)

# Now lets write(save) the predicted fare_amount in disk as .csv format
write.csv(grf,"grf_predictions_R.csv",row.names = FALSE)
```