Cab Fare Prediction

AKSHAY PRABHU

14th July 2019

Introduction

Problem Statement

We are a cab rental start-up company. We have successfully run the pilot project and now want to launch our cab service across the country. We have collected the historical data from our pilot project and now have a requirement to apply analytics for fare prediction. We need to design a system that predicts the fare amount for a cab ride in the city.

Data Set

1) train_cab.zip
2) test.zip

Attributes

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
Fare_amount - float indicating the fare amount for the corresponding ride.

Methodology

The language used in report is R.

Pre-processing

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.
First we'll import all the necessary libraries required to perform our analysis.
The labrires which we have used are as follows:

library(lubridate): Functions to work with date-times and time-spans: fast and user friendly parsing of date-time data, extraction and updating of components of a date-time (years, months, days, hours, minutes, and seconds), algebraic manipulation on date-time and time-span objects. The 'lubridate' package has a consistent and memorable syntax that makes working with dates easy and fun

library(dplyr): A fast, consistent tool for working with data frame like objects, both in memory and out of memory.

library(DMwR): Functions and data for Data Mining with R.

library(tidyr): . It's designed specifically for data tidying (not general reshaping or aggregating) and works well with 'dplyr' data pipelines.

library(caTools): Contains several basic utility functions including: moving (rolling, running) window statistic functions, read/write for GIF and ENVI binary files, fast calculation of AUC, LogitBoost classifier, base64 encoder/decoder, round-off-error-free sum and cumsum, etc

library(Hmisc): n Contains many functions useful for data analysis, high-level graphics, utility operations, functions for computing sample size and power, importing and annotating datasets, imputing missing values, advanced table making, variable clustering, character string manipulation, conversion of R objects to LaTeX and html code, and recoding variables.

library(Metrics): An implementation of evaluation metrics in R that are commonly used in supervised machine learning. It implements metrics for regression, time series, binary classification, classification, and information retrieval problems. It has zero dependencies and a consistent, simple interface for all functions.

library(OutlierDetection): To detect outliers using different methods namely model based outlier detection
library(randomForest): Regression based on a forest of trees using random inputs
library(rpart): Recursive partitioning for classification, regression and survival trees.
library(ggplot2): Library used for plotting graphs.

Importing data sets:

After importing libraries, we need to import datasets and set the working directory to perform further analysis.

```
setwd("C:/Users/Akshay Prabhu/Desktop/edwisor project")
train_fare = read.csv("train_cab.csv", header=T, stringsAsFactors = FALSE)
test_fare = read.csv("test.csv", header = T, stringsAsFactors = FALSE)
```

Imputing missing values:

With the help of summary() function, we can see that there are missing values in the variable passenger_count and fare_amount. As passenger count cannot be in decimals, we have to go for median imputation.
Before that we also need to convert fare_amount into numeric data type.

```
train_fare$fare_amount=as.numeric(as.character(train_fare$fare_amount))
train_fare$passenger_count[is.na(train_fare$passenger_count)]=median(train_fare$passeng
er_count, na.rm=T)
train_fare$fare_amount[is.na(train_fare$fare_amount)]=median(train_fare$fare_amount,
na.rm=T)
```

Data cleaning:

As passenger count in a cab cannot exceed beyond 7, and fare amount cannot be below 0, we have to drop such observations where we find such data.

```
#we need to remove the passenger count greater than 7
train_fare=filter(train_fare, passenger_count<8)

#we also need to remove fare_amount <=0
train_fare=filter(train_fare, fare_amount>0)
```

The range of latitudes is from -90 to 90 and range of longitudes is from -180 to 180. Hence we have to set this range of latitudes and longitudes in train data.

```
train_fare=filter(train_fare, pickup_latitude<=90, pickup_latitude>=-90,
                  pickup_longitude<=180, pickup_longitude>=-180,
                  dropoff_latitude<=90, pickup_longitude>=-90,
                  dropoff_longitude<=180, pickup_longitude>=-90)
```

Creating new features:

From our date_time feature, we can add new features like year, month, day, time, days_of_week to our data set to analyse how the fares have changed over time.
We have to manipulate dat_time in both train data as well as test data.

```
datetxt=as.Date(train_fare$pickup_datetime)

train_fare$Year = as.numeric(format(datetxt, format = "%Y"))
train_fare$Month = as.numeric(format(datetxt, format = "%m"))
train_fare$Day = as.numeric(format(datetxt, format = "%d"))
train_fare$Time <- format(as.POSIXct(train_fare$`pickup_datetime`, "%Y-%m-%d
%H:%M:%S", tz = ""), format = "%H:%M:%S")
train_fare$Days_of_week=wday(datetxt, label = TRUE)

datetxt=as.Date(test_fare$pickup_datetime)

test_fare$Year = as.numeric(format(datetxt, format = "%Y"))
test_fare$Month = as.numeric(format(datetxt, format = "%m"))
test_fare$Day = as.numeric(format(datetxt, format = "%d"))
test_fare$Time <- format(as.POSIXct(test_fare$`pickup_datetime`, "%Y-%m-%d
%H:%M:%S", tz = ""), format = "%H:%M:%S")
test_fare$Days_of_week=wday(datetxt, label = TRUE)
```

There is a date_time value of ‘43’ which we have to drop because it is not of required datatype.

```
train_fare=train_fare[-c(1315),]
```

Now with the help of latitudes and longitudes we can calculate distance between pickup point and drop off point.
We need to calculate distance for both train and test data.

```
fun_dist <-function(lat1, lat2, lon1,lon2){
  p = 0.017453292519943295 # Pi/180
  a = 0.5 - cos((lat2 - lat1) * p)/2 + cos(lat1 * p) * cos(lat2 * p) * (1 - cos((lon2 - lon1) * p)) / 2
  return (0.6213712 * 12742 * asin(sqrt(a)))
}
i = 1
dis = ''
while (i<=16041) {
  lat1 = train_fare$pickup_latitude[i]
  lat2 = train_fare$dropoff_latitude[i]
  lon1 = train_fare$pickup_longitude[i]
  lon2 = train_fare$dropoff_longitude[i]
  x = fun_dist(lat1,lat2, lon1,lon2)
  dis[[i]]<-x
  i=i+1
}
train_fare$Distance <- dis
train_fare$Distance=as.numeric(as.character(train_fare$Distance))

i = 1
dis_test = ''
while (i<=9914) {
  lat1 = test_fare$pickup_latitude[i]
  lat2 = test_fare$dropoff_latitude[i]
  lon1 = test_fare$pickup_longitude[i]
  lon2 = test_fare$dropoff_longitude[i]
  y = fun_dist(lat1,lat2, lon1,lon2)
  dis_test[[i]]<-y
  i=i+1
}

test_fare$Distance <- dis_test
test_fare$Distance=as.numeric(as.character(test_fare$Distance))
```

Now that we have our new feature Distance, we also have to clean it up. As distance cannot be 0 we have remove such observations where distance value is 0.

```
dis_zero = which(train_fare$Distance == 0)
train_fare = train_fare [-dis_zero, ]
```

Managing outliers

An Outlier is a rare chance of occurrence within a given data set. In Data Science, an Outlier is an observation point that is distant from other observations. An Outlier may be due to variability in the measurement or it may indicate experimental error. We have to remove outliers as it can cause deviation in our analysis. We can detect outliers with the help of method OutlierDetection.

```
variable1=subset(train_fare, select=-c(2,3,4,5,6,7,8,9,10,11,12))
OutlierDetection(variable1, k = 0.05 * nrow(variable1), cutoff = 0.95,
        Method = "euclidean", rnames = FALSE, depth = FALSE,
        dense = FALSE, distance = FALSE, dispersion = FALSE)
```

From the above code we can infer that distance greater than 2000 and fare_amountgreater than 4000 have to be removed.

```
#removing distance>2000
dis_2k=which(train_fare$Distance>2000)
train_fare=train_fare[-dis_2k,]


#removing fare_amount greater than 4k
fa_4k=which(train_fare$fare_amount>4000)
train_fare=train_fare[-fa_4k,]
```

Feature selection
Correlation and P-Value

For a better view of analysis, we have to exclude certain features which are not important for building our model.
Correlation is a statistical term which in common usage refers to how close two variables are to having a linear relationship with each other. Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features.

P-value
Before we try to understand about about p-value, we need to know about the null hypothesis.

Null hypothesis is a general statement that there is no relationship between two measured phenomena.

P-value or probability value or asymptotic significance is a probability value for a given statistical model that, if the null hypothesis is true, a set of statistical observations more commonly known as the statistical summary is greater than or equal in magnitude to the observed results.
In other words, P-value gives us the probability of finding an observation under an assumption that a particular hypothesis is true. This probability is used to accept or reject that hypothesis.

P value less than 0.05 is considered to be of high significance.

```
#lets find correlation with p-values between variables
data_cor=subset(train_fare, select=-c(2,3,4,5,6,11))
data_cor=sapply(data_cor, as.numeric)
cor_cof=rcorr(as.matrix(data_cor))
cor_cof
```

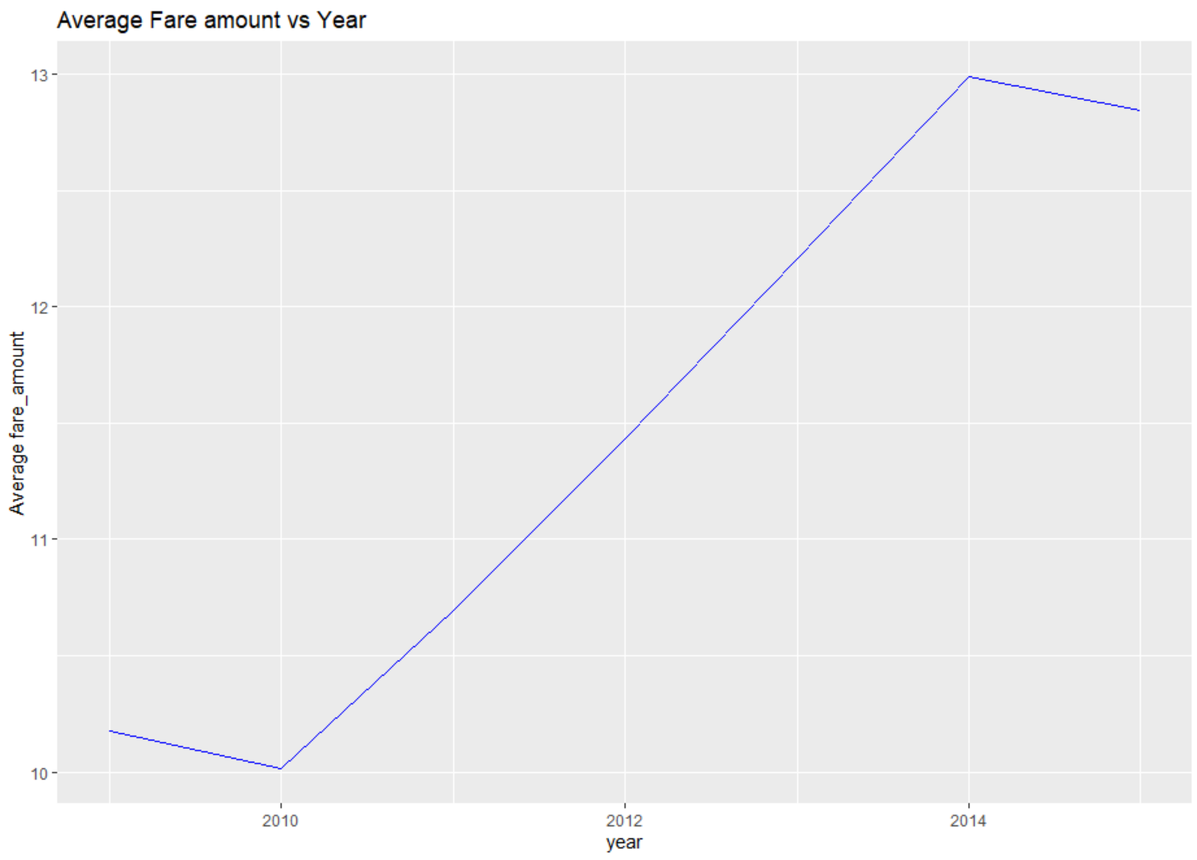
We can see that Distance,Month and Year are highly correlated with Fare_amount. We can neglect other variables.

```
train_datamodel=subset(train_fare, select=-c(2,3,4,5,6,7,10,11,12))
train_datamodel=train_datamodel[,c(3,2,4,1)]
```

Relationship between significant features

Now that we have our significant variables, we can observe the relationship between them graphically.

1. Lets see how the fare amount has changed yearly.



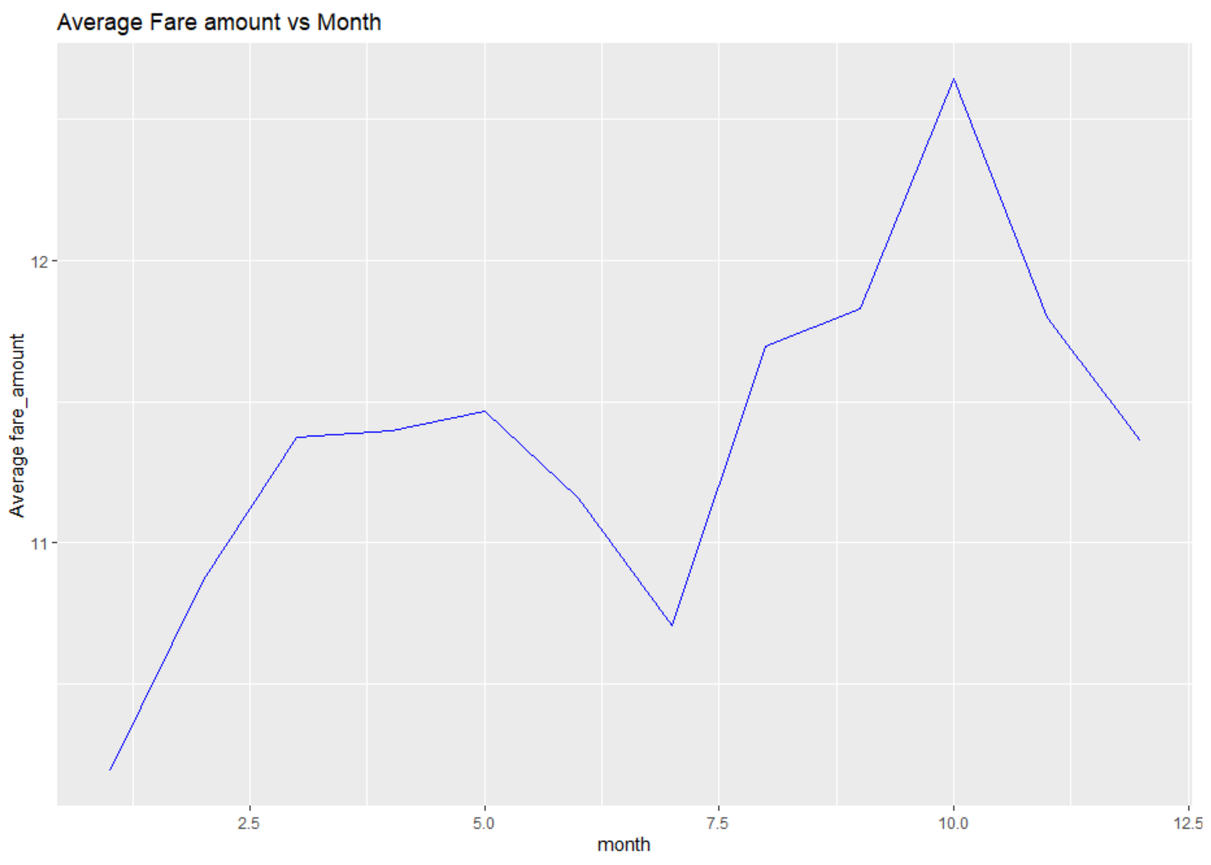
From the above line graph we can infer that prices of fares have been rising since 2010. Inflation could be a cause for that rise. But after 2014 we can see that there is a downfall in prices of cab rides.

```
var_year=group_by(train_fare, Year)

var_summary=summarise(var_year, minFare = min(fare_amount), maxFare =
max(fare_amount), meanFare = mean(fare_amount))

ggplot()+
  geom_line(aes(x=var_summary$Year, y=var_summary$meanFare),color='blue')+
  ggtitle('Average Fare amount vs Year')+
  xlab('year')+
  ylab('Average fare_amount')
```

2. Lets see how the fare amounts have behaved according to the months.



From the above graph we can infer that there are lower fares in the months of January, February and July.

var_month=group_by(train_fare, Month)

var_summary_month=summarise(var_month, minFare = min(fare_amount), maxFare = max(fare_amount), meanFare = mean(fare_amount))

```
ggplot()+
  geom_line(aes(x=var_summary_month$Month,
y=var_summary_month$meanFare),color='blue')+
  ggtitle('Average Fare amount vs Month')+
  xlab('month')+
  ylab('Average fare_amount')
```

Model selection

As we have numerical data to predict, we have to use regression models.
We are going to use three approaches for regression

1. Simple linear regression
2. Decision tree
3. Random Forest

Simple linear regression

Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables:
One variable, denoted x, is regarded as the predictor, explanatory, or independent variable.
The other variable, denoted y, is regarded as the response, outcome, or dependent variable.

Lets run the simple linear regression and observe the predicted values.
Before running the model, we have to split the train data into train and test.

```
set.seed(123)
train_index=sample.split(train_datamodel, SplitRatio = 0.8)
train_trainFair=subset(train_datamodel, train_index==TRUE)
train_testFair=subset(train_datamodel, train_index==FALSE)

#lets run the linear regression model
lm_model=lm(fare_amount~., data=train_trainFair)
summary(lm_model)

#lets run the linear regression model
lm_model=lm(fare_amount~., data=train_trainFair)
summary(lm_model)
```

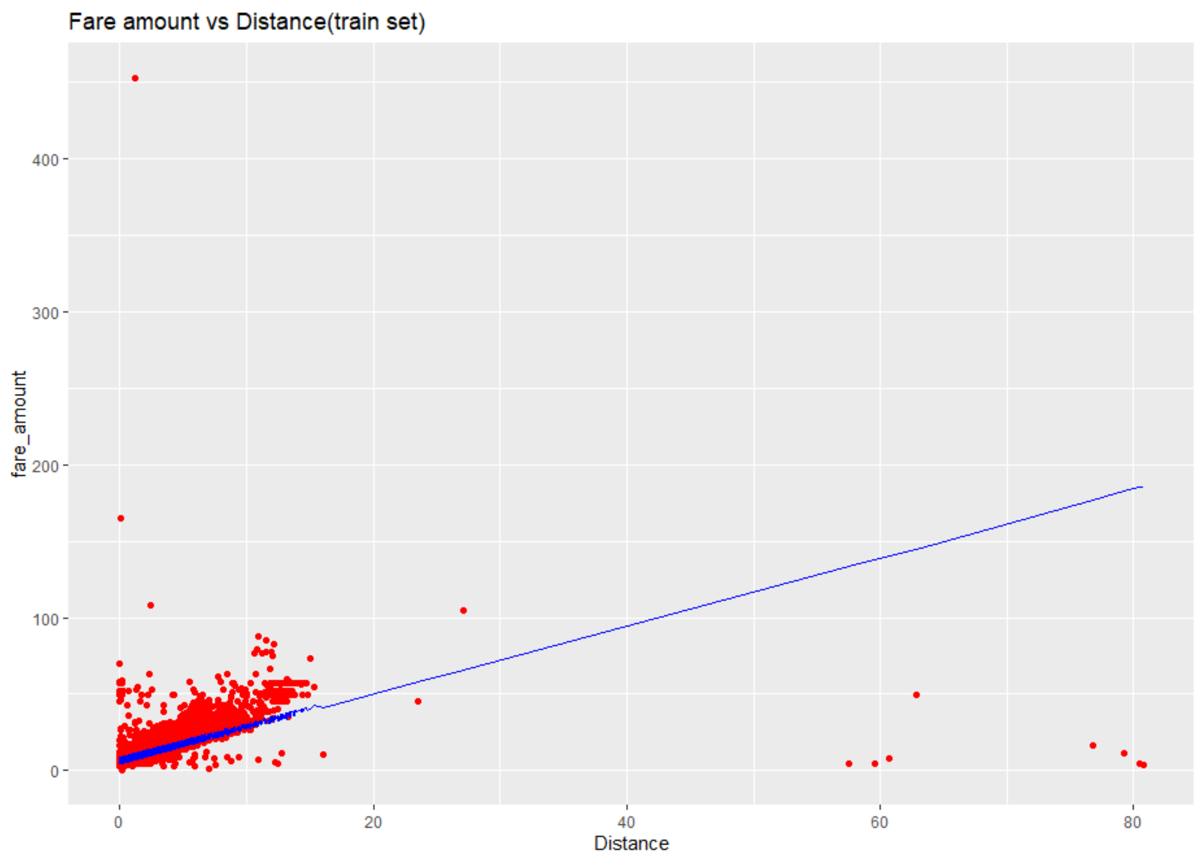
The p-value is less than 0.05 for regression model and hence we can neglect the null hypothesis.

Y_pred gives us the predicted fare amount on test data.

```
y_pred=predict(lm_model, newdata = train_testFair)
```

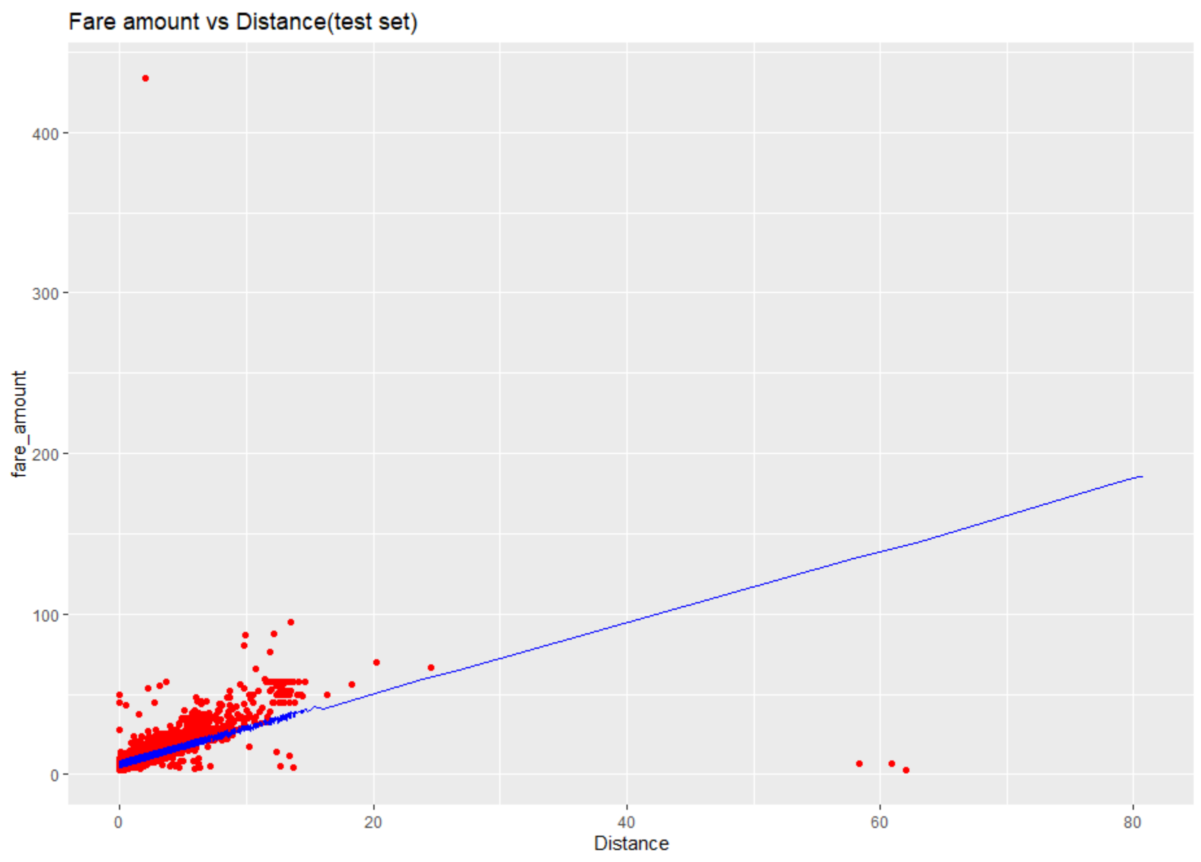
Lets plot the results of linear regression model on training set

```
ggplot()+
  geom_point(aes(x=train_trainFair$Distance, y=train_trainFair$fare_amount),color='red')+
  geom_line(aes(x=train_trainFair$Distance, y=predict(lm_model, newdata =
train_trainFair)),color='blue')+
  ggtitle('Fare amount vs Distance(train set)')+
  xlab('Distance')+
  ylab('fare_amount')
```



Now lets plot the results of linear regression model on test set.

```
ggplot()+
  geom_point(aes(x=train_testFair$Distance, y=train_testFair$fare_amount),color='red')+
  geom_line(aes(x=train_trainFair$Distance, y=predict(lm_model, newdata =
train_trainFair)),color='blue')+
  ggtitle('Fare amount vs Distance(test set)')+
  xlab('Distance')+
  ylab('fare_amount')
```



Decision tree
A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, its also widely used in machine learning.

Before applying the decision tree model, we have to split the train data into train and test again.
set.seed(123)

train_index=sample.split(train_datamodel, SplitRatio = 0.8)
train_trainFair=subset(train_datamodel, train_index==TRUE)
train_testFair=subset(train_datamodel, train_index==FALSE)

Lets apply the decision tree model.
fit=rpart(fare_amount~., data = train_trainFair, method = "anova")
summary(fit)

We can see that from summary, distance is the most important variable.

Lets predict the decision tree on test data.
dt_predict=predict(fit,train_testFair[-4])
Dt_predict will give us the predicted fare amount on test data.

Random forest.
A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging.

Before applying the Random forest, we have to split the train data into train and test again.
set.seed(123)
train_index=sample.split(train_datamodel, SplitRatio = 0.8)
train_trainFair=subset(train_datamodel, train_index==TRUE)
train_testFair=subset(train_datamodel, train_index==FALSE)

Lets apply Random Forest model
RF_model=randomForest(fare_amount~.,train_trainFair,importance=TRUE, ntree=500)
RF_prediction=predict(RF_model, train_testFair[-4])

Model evaluation

For model evaluation we are going to choose RMSE(Root Mean Squared Error) technique to find the errors in predicted values with actual values. We are opting for RMSE because it is used for evaluating time series based data and our data is also a time series data. Hence we are going to select RMSE.

RMSE for linear regression
rmse(train_testFair$fare_amount,y_pred)
The RMSE for linear regression is 9.3

RMSE for decision tree
rmse(train_testFair$fare_amount,dt_predict)
The RMSE for decision tree is 8.3

RMSE for Random forest
rmse(train_testFair$fare_amount,RF_prediction)

The RMSE for Random forest is 8.1

Model selection

We select Random forest model to predict the fare amount in test data as it has the lowest RMSE value.

```
#lets impute predicted values from random forest model into test data.
test_dataModel=subset(test_fare, select=-c(1,2,3,4,5,6,9,10,11))
pred_value=predict(RF_model, test_dataModel)
pred_value<-data.frame(pred_value)
test_fare$predicted_fairamount<-pred_value[1]
```

Python code for cab fare prediction project:

```
import calendar
import pandas as pd
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
from math import sqrt

#read data sets
df = pd.read_csv("train_cab.csv")
df['pickup_datetime']=pd.to_datetime(df['pickup_datetime'],format='%Y-%m-%d %H:%M:%S
UTC',errors='coerce' )
df["fare_amount"] = pd.to_numeric(df["fare_amount"], errors='coerce')

#clean date_time feature
df = df[df["pickup_datetime"] < df["pickup_datetime"].max()]
df = df[df["pickup_datetime"] > df["pickup_datetime"].min()]
df.reset_index(inplace=True)
df = df.drop(['index'], axis= 1)

#creation of hour,day,day_of_week,month,year
```

```
df['pickup_day'] = df['pickup_datetime'].apply(lambda x: x.day)
df['pickup_hour'] = df['pickup_datetime'].apply(lambda x: x.hour)
df['pickup_day_of_week'] = df['pickup_datetime'].apply(lambda x:
calendar.day_name[x.weekday()])
df['pickup_month'] = df['pickup_datetime'].apply(lambda x: x.month)
df['pickup_year'] = df['pickup_datetime'].apply(lambda x: x.year)

#calculating distance
def distance(lat1, lat2, lon1,lon2):
    p = 0.017453292519943295 # Pi/180
    a = 0.5 - np.cos((lat2 - lat1) * p)/2 + np.cos(lat1 * p) * np.cos(lat2 * p) * (1 - np.cos((lon2 -
lon1) * p)) / 2
    return 0.6213712 * 12742 * np.arcsin(np.sqrt(a))
X = []
for i in range(len(df)):
    lat1 = df["pickup_latitude"][i]
    lat2 =df["dropoff_latitude"][i]
    lon1 =df["pickup_longitude"][i]
    lon2 =df["dropoff_longitude"][i]
    d = distance(lat1,lat2,lon1,lon2)
    X.append(d)
print(X[0:10])

df["Distance"] = X

#removing outliers
len(df.loc[df["passenger_count"] > 8])
df = df[df["Distance"] < 2000]
df = df[df["Distance"] > 0 ]
df = df[df["passenger_count"] >= 1]
df = df[df["passenger_count"] <= 8]
df = df.loc[df["fare_amount"] >= 1]
df = df.loc[df["fare_amount"] <4000 ]

print(df.describe())

#convery pick up day of week into numeric
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(df['pickup_day_of_week'].drop_duplicates())
df['pickup_day_of_week'] = encoder.transform(df['pickup_day_of_week'])

#correlation for feature selection
df_corr = df.drop(["pickup_datetime"], axis= 1)
corr = df_corr.corr()
#p-value calculation
import statsmodels.formula.api as sm
x = (df_corr.iloc[:,5:])
```

```
Y = (df_corr.iloc[:, 0])
def backwardElimination(x, sl):
    numVars = len(x[0])
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(Y, x).fit()
        maxVar = max(regressor_OLS.pvalues)

        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    x = np.delete(x, j, 1)
    print(regressor_OLS.summary())

    return x
SL = 0.05
data_modeled = backwardElimination(x.values, SL )

#split train data into train and test
from sklearn.model_selection import train_test_split
X = df_corr.iloc[:,-3:]
y = df_corr[["fare_amount"]]
df_train, df_test,y_train,y_test = train_test_split(X,y, test_size=0.2, random_state=42)

#linear regression
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(df_train, y_train)
#X_test = df_test[[""]]
y_pred = reg.predict(df_test)

#visualising train set
plt.scatter(df_train[['Distance']],y_train,color='red')
plt.plot(df_train[['Distance']],reg.predict(df_train),color='blue')
plt.title('Distance vs fare_amount(train_set)')
plt.xlabel('distance')
plt.ylabel('fare_amount')

#visualising test set
plt.scatter(df_test[['Distance']],y_test,color='red')
plt.plot(df_train[['Distance']],reg.predict(df_train),color='blue')
plt.title('Distance vs fare_amount(test_set)')
plt.xlabel('distance')
plt.ylabel('fare_amount')

#error rate
from sklearn.metrics import mean_squared_error
score = mean_squared_error(y_test, y_pred)
print(score)
```

```
root_mean= sqrt(score)
print("RMSE of linear regression:", root_mean)

#decision tree
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor(max_depth=500)
dt.fit(df_train,y_train)
dt_pred=dt.predict(df_test)

#error rate for dt
from sklearn.metrics import mean_squared_error
score = mean_squared_error(y_test, dt_pred)
print(score)
root_mean= sqrt(score)
print("RMSE of decision tree:", root_mean)

#random forest
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(max_depth=10, random_state=0, n_estimators=200)
rf.fit(df_train, y_train)
y_dec_pred = rf.predict(df_test)

#error rate for rf
from sklearn.metrics import mean_squared_error
score = mean_squared_error(y_test, y_dec_pred)
root_mean= sqrt(score)
print("RMSE for Random forest:", root_mean)

#read test data
test = pd.read_csv("test.csv")

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

#creation of variables from date time feature
test['pickup_day'] = test['pickup_datetime'].apply(lambda x: x.day)
test['pickup_hour'] = test['pickup_datetime'].apply(lambda x: x.hour)
test['pickup_day_of_week'] = test['pickup_datetime'].apply(lambda x:
calendar.day_name[x.weekday()])
test['pickup_month'] = test['pickup_datetime'].apply(lambda x: x.month)
test['pickup_year'] = test['pickup_datetime'].apply(lambda x: x.year)

#distance for test
X_testfare = []
for i in range(len(test)):
    lat1 = test["pickup_latitude"][i]
    lat2 =test["dropoff_latitude"][i]
    lon1 =test["pickup_longitude"][i]
    lon2 =test["dropoff_longitude"][i]
    d = distance(lat1,lat2,lon1,lon2)
    X_testfare.append(d)
print(X_testfare[0:10])

test["Distance"] = X_testfare

#imputing random forest model into test data as it has lowest RMSE value
test_to=test[['pickup_month','pickup_year','Distance']]
test_model=rf.predict(test_to)
test["predicted_fareamount"]=test_model
```