# Cab Fare Prediction

Date : 27-11-2019

Author : Pritam Sonawane

# Introduction

## 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.

## Data-set :

The details of data attributes in the dataset are as follows -

```
In [57]: ############################################Explore the data#########################################
## Read the data
train_df <- read.csv("./train_cab.csv", stringsAsFactors=FALSE)
dim(train_df)
test_df <- read.csv("./test.csv", stringsAsFactors=FALSE)
# column names
names(train_df)
# datatypes
str(train_df)
```

```
16067   7

'fare_amount'   'pickup_datetime'   'pickup_longitude'   'pickup_latitude'   'dropoff_longitude'   'dropoff_latitude'   'passenger_count'

'data.frame':   16067 obs. of  7 variables:
 $ fare_amount      : chr  "4.5" "16.9" "5.7" "7.7" ...
 $ pickup_datetime  : chr  "2009-06-15 17:26:21 UTC" "2010-01-05 16:52:16 UTC" "2011-08-18 00:35:00 UTC" "2012-0
21 04:30:42 UTC" ...
 $ 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 ...
```

**From above data we can see fare amount having char datatype also from datetime we can extract year,date and time for model development**

As we can see from dataset the target variable contains continuous values so our task here is to build a regression model which will predict fare amount for car.

Here we have independent variables like pickup_datetime, pickup_longitude, pickup_latitude, dropof_longitude, dropof_latitude, passenger_count and fare_amount is a target variable

From datetime we can get year , month , time features which can be useful for model building. Also from extract feature like distance from pickup and dropoff latitude ,longitude respectively.

# Methodology

## Pre Processing

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.
Steps Involved in Data Preprocessing:

### 1. Data Cleaning:

This step is important because in most situations data provided by the customer has a bad quality or just cannot be directly fed to some kind of ML model. It includes data type conversion, data validation, handling dates, handling nominal and categorical variables. But in this case dataset variable data type is as follows:

- We need to convert fare amount variable to numeric value and pickup_datetime to datetime
- From pickup_datetime i have extracted date, year, and time variable

The great circle distance or orthodromic distance is the shortest distance between two points on a sphere (or the surface of Earth). In order to use this method, we need to have the coordinates of point A and point B.

Find the value of longitude in radians:

Value of Longitude in Radians, long = Longitude / (180/pi) OR

Value of Longitude in Radians, long = Longitude / 57.29577951

to get the distance between point A and point B use the following formula:

Distance, d = 3963.0 * arccos[(sin(lat1) * sin(lat2)) + cos(lat1) * cos(lat2) * cos(long2 – long1)]

## 2. Basic-data-exploration:

```
In [188]: train_df.describe()
```

Out[188]:

| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count |
|---|---|---|---|---|---|---|
| **count** | 16027.000000 | 16027.000000 | 16027.000000 | 16027.000000 | 16027.000000 | 15972.000000 |
| **mean** | 11.269636 | -72.472867 | 39.920276 | -72.472420 | 39.903395 | 2.624444 |
| **std** | 9.375223 | 10.544818 | 6.813129 | 10.541479 | 6.170564 | 60.918805 |
| **min** | 2.500000 | -74.438233 | -74.006893 | -74.227047 | -74.006377 | 0.000000 |
| **25%** | 6.000000 | -73.992153 | 40.734950 | -73.991182 | 40.734732 | 1.000000 |
| **50%** | 8.500000 | -73.981704 | 40.752615 | -73.980182 | 40.753577 | 1.000000 |
| **75%** | 12.500000 | -73.966848 | 40.767366 | -73.963666 | 40.768008 | 2.000000 |
| **max** | 96.000000 | 40.766125 | 401.083332 | 40.802437 | 41.366138 | 5345.000000 |

The results show 8 numbers for each column in your original dataset. The first number, the **count**, shows how many rows have non-missing values.

The second value is the **mean**, which is the average. Under that, **std** is the standard deviation, which measures how numerically spread out the values are.

## 3. Missing value analysis:

The concept of missing values is important to understand in order to successfully manage data. If the missing values are not handled properly then we may end up drawing an inaccurate inference about the data.

We can impute missing values by mean ,median of variable or for categorical variable we use mode.

Here there are some missing values found in fare amount, passenger count ,date, time and year :

```
In [60]: ##################################Missing Values Analysis##############################################
missing_val = data.frame(apply(train_df,2,function(x){sum(is.na(x))}))
missing_val
```

A data.frame: 12 × 1

| | apply.train_df..2..function.x... |
|---|---|
| | <int> |
| fare_amount | 24 |
| pickup_datetime | 0 |
| pickup_longitude | 0 |
| pickup_latitude | 0 |
| dropoff_longitude | 0 |
| dropoff_latitude | 0 |
| passenger_count | 55 |
| pickup_date | 1 |
| pickup_mnth | 1 |
| pickup_yr | 1 |
| pickup_hour | 1 |

To impute missing values in fare_amount I have used mean value imputation technique

For passenger count i used median value and i dropped missing values rows for date,time and year variable

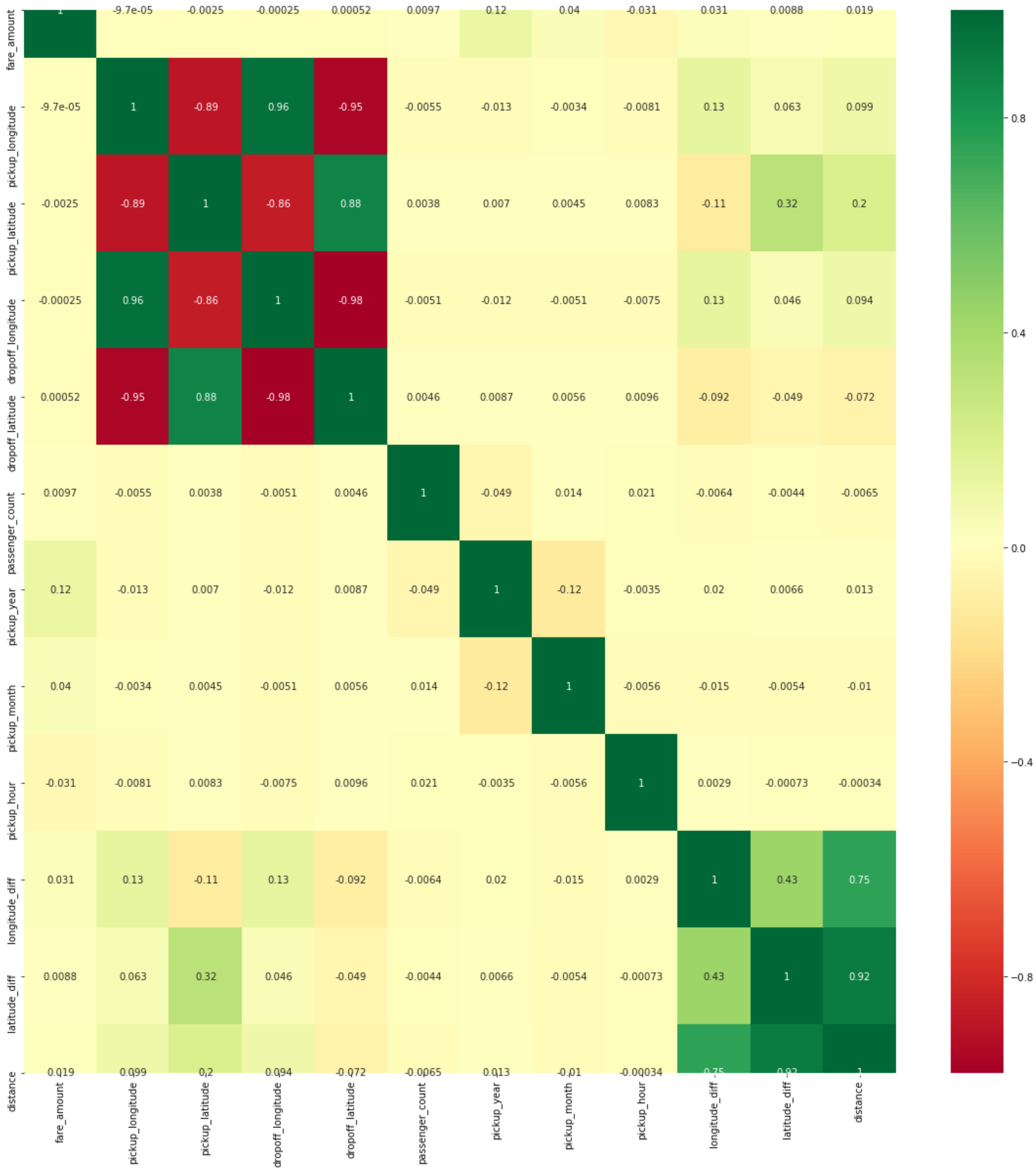
## 3. Outlier analysis:

An outlier is an element of a data set that distinctly stands out from the rest of the data. It can affect the overall observation made from the data series.

From date description shown above we observed that there is 1 outlier in **pickup_latitude,**

Also fare_amount have -ve values ,zero values which are removed

## 4. Feature engineering :



**From above correlation graph I understand that there are few multicollinearity in the dataset.**

# Decision tree for regression

- Here our target variable having continuous values hence we have to use regression model in which by variance we decide best splits
- lower values of variance clearly leading to more pure node and high value of variance lead to impure node
- We will use variance reduction method for node splitting:
  In the anova method
  the splitting criteria is
  SST − (SSL + SSR), where SST = P(yi − y¯)2 is the sum of squares for the node, and SSR, SSL are the sums of squares for the right and left son, respectively.
  This is equivalent to choosing the split to maximize the between-groups sum-of-squares in a simple analysis of variance. This rule is identical to the regression option for tree
- rpart for regression
  rpart(formula, data=, method=,control=) where

  formula : is in the format

  outcome ~ predictor1+predictor2+predictor3+ect.

  data : training dataset

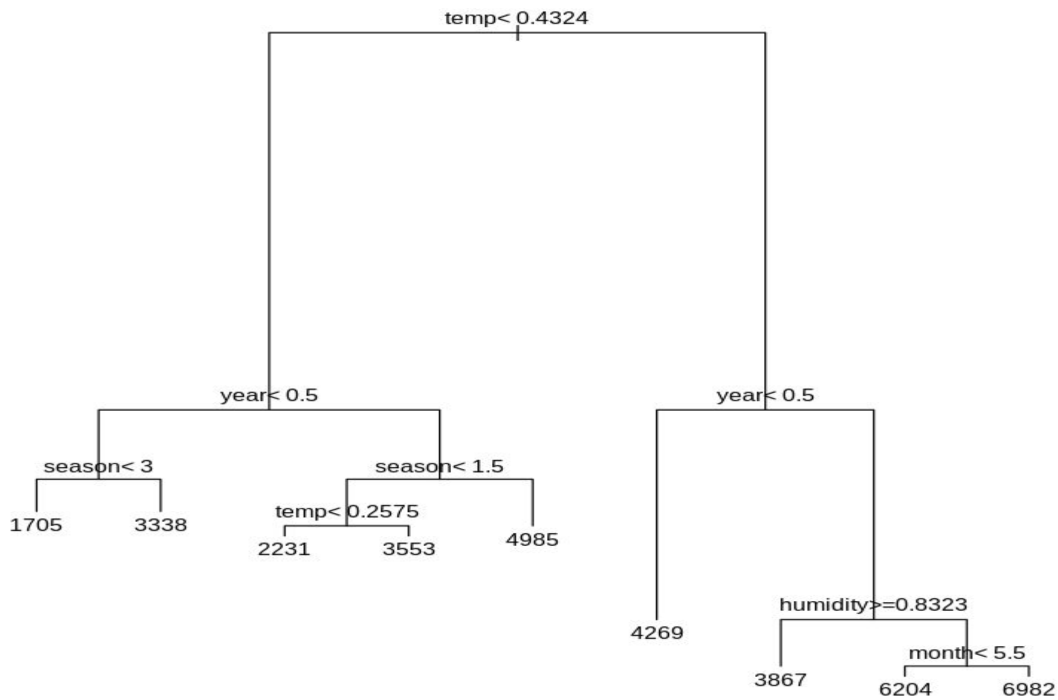
  method :

  "class" for a classification tree

  "anova" for a regression tree

  control : optional parameters for controlling tree growth.

  **Decision tree model visualisation**




## Random Forest for regression

A random forest allows us to determine the most important predictors across the explanatory variables by generating many decision trees and then ranking the variables by importance.

In [32]:
```
# Number of variables randomly sampled as candidates at each split. Note that the default values are different for
# where p is number of variables in x) and regression (p/3)
##############################Random Forest model###################################

rf_2=randomForest(total_count ~ . , data = train,mtry =4,ntree=100 ,nodesize =10 ,importance =TRUE)
```

In [33]:
```
predictions_RF2 = predict(rf_2, test[,-11])
```

Number of variables randomly sampled as candidates at each split.

From above

**mtry :**

Number of variables randomly sampled as candidates at each split. Note that the default values are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)

**ntree :** Number of trees to grow.

**nodesize :** Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time). Note that the default values are different for classification (1) and regression (5).

## Linear Regression

Regression is a parametric technique used to predict continuous (dependent) variable given a set of independent variables. Mathematically, regression uses a linear function to approximate (predict) the dependent variable given as: $Y = \beta o + \beta 1X + \in$ where, Y - Dependent variable X - Independent variable $\beta o$ - Intercept $\beta 1$ - Slope $\in$ - Error

- $\beta o$ and $\beta 1$ are known as coefficients. This is the equation of simple linear regression.
- Error is an inevitable part of the prediction-making process. No matter how powerful the algorithm we choose, there will always remain an ($\in$) irreducible error The formula to calculate coefficients goes like this: $\beta 1 = \Sigma(xi - xmean)(yi-ymean)/ \Sigma (xi - xmean)^2$ where i= 1 to n (no. of obs.)

$\beta o$ = ymean - $\beta 1$(xmean)

```
In [86]: ##################################################Regression Analysis############################################
         #the base function lm is used for regression.
         regmodel <- lm(total_count ~ ., data = train)
         summary(regmodel)
```

```
Call:
lm(formula = total_count ~ ., data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-4148.0  -454.5    41.6   542.4  3104.0 

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
(Intercept)        1593.50     253.71   6.281 6.65e-10 ***
season              547.82      59.75   9.169  < 2e-16 ***
year               2066.45      72.30  28.581  < 2e-16 ***
month               -36.49      18.79  -1.942  0.05258 .  
holiday            -655.13     233.84  -2.802  0.00526 ** 
weekday              70.51      17.84   3.952 8.73e-05 ***
workingday          132.48      78.94   1.678  0.09383 .  
weather_condition  -637.24      88.03  -7.239 1.46e-12 ***
temp               5068.96     212.95  23.803  < 2e-16 ***
humidity          -1143.12     353.56  -3.233  0.00129 ** 
windspeed         -2322.85     536.55  -4.329 1.76e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 864.8 on 576 degrees of freedom
Multiple R-squared:  0.8074,	Adjusted R-squared:  0.804 
F-statistic: 241.4 on 10 and 576 DF,  p-value: < 2.2e-16
```

- Intercept - This is the $\beta o$ value. It's the prediction made by model when all the independent variables are set to zero.
- Estimate - This represents regression coefficients for respective variables.
- Std. Error - This determines the level of variability associated with the estimates.
- t value - t statistic is generally used to determine variable significance, i.e.
  if a variable is significantly adding information to the model.
- t value > 2 suggests the variable is significant.
- p value - It's the probability value of respective variables determining their significance in the model.
  p value < 0.05 is always desirable.

The adjusted $R^2$ implies that our model explains ~80.4% total variance in the data.

## Model Evaluation :

Mean Absolute Error (MAE) and Root mean squared error (RMSE) are two of the most common metrics used to measure accuracy for continuous variables.

**Mean Absolute Error (MAE):** MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

MAE = 1/n $\sum_{i-0}^{n} yi - yj$

**Root mean squared error (RMSE)**: RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

RMSE = $\sqrt{1/n \sum_{i=0}^{n} (yi - yj)}$

## Multiple Linear Regression model

- mean absolute percentage error = 0.186550951376743
- accuracy = 81.3449048623257 %
- Root mean square error = 950.226605469929

## Random Forest model

- mean absolute percentage error = 0.116119353398914
- accuracy = 88.3880646601086 %
- Root mean square error = 661.487858223226

## Decision tree model

- mean absolute percentage error = 0.169504694884855
- accuracy = 83.0495305115145 %
- Root mean square error = 910.098855669325

## By comparing Decision tree, Random Forest and Multiple Linear Regression models we can say that Random Forest model performing very well on this dataset

## R-Code :

jupyter bike_rental_project (1) (1).r 6 hours ago

File Edit View Language

```r

# Load all the packages required for the analysis
library(tidyverse)
library(ggplot2) # Visualisation
install.packages("corrgram")
install.packages("caret")
library(caret)
library(corrgram)
install.packages("Metrics")
library(Metrics)
library(rpart)

##############################################Explore the data#########################################
# print dimention of the datast
dim(bike_pr_day)
# column names
names(bike_pr_day)
# datatypes of variable values
str(bike_pr_day)

## Read the data
bike_pr_day <- read.csv("./day.csv", stringsAsFactors=FALSE)

head(bike_pr_day)

###################################Missing Values Analysis###############################################
#calculate missing values present in dataset
missing_val = data.frame(apply(bike_pr_day,2,function(x){sum(is.na(x))}))
missing_val
```

```r

# rename columns of the dataset
names(bike_pr_day)<-
c('id','datetime','season','year','month','holiday','weekday','workingday','weather_condition','temp','atemp','humidity','windspeed','casual','registered','total_count')
head(bike_pr_day)

#add act_season , act_holiday, act_weathersit columns for better visualisation
bike_pr_day$act_season = factor(x = bike_pr_day$season, levels = c(1,2,3,4), labels = c("Spring","Summer","Fall","Winter"))
bike_pr_day$act_holiday = factor(x = bike_pr_day$holiday, levels = c(0,1), labels = c("Working day","Holiday"))

# 1: Clear, Few clouds, Partly cloudy, Partly cloudy
#2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
#3: Light Snow, Light Rain + Thunderstorm + Scattered clouds,
#Light Rain + Scattered clouds
#4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
# we will take 1=Clear, 2=Cloudy/Mist ,3=Light Rain/snow/Scattered clouds, 4=Heavy Rain/Snow/Fog
bike_pr_day$act_weathersit = factor(x = bike_pr_day$weather_condition, levels = c(1,2,3,4),
                                    labels = c("Clear","Cloudy/Mist","Light Rain/snow/Scattered clouds","Heavy Rain/Snow/Fog"))


bike_pr_day$act_weakday = factor(x = bike_pr_day$weekday, levels = c(0,1,2,3,4,5,6),
                                 labels = c("Monday","Tuesday", "Wednesday","Thursday","Friday","Saturday","Sunday"))



# Bar graph of Season wise monthly distribution of counts
ggplot(bike_pr_day,aes(x=month,y=total_count,fill=act_season))+theme_bw()+geom_col()+
labs(x='Month',y='Total_Count',title='Season wise monthly distribution of counts')

```

```r

# Bar graph of Season wise monthly distribution of counts
ggplot(bike_pr_day,aes(x=month,y=total_count,fill=act_season))+theme_bw()+geom_col()+
labs(x='Month',y='Total_Count',title='Season wise monthly distribution of counts')

#column plot for weekday wise monthly distribution of counts
ggplot(bike_pr_day,aes(x=month,y=total_count,fill=act_weakday))+theme_bw()+geom_col()+
labs(x='Month',y='Total_Count',title='Weekday wise monthly distribution of counts')

# Bar graph of Holiday wise monthly distribution of counts
ggplot(bike_pr_day,aes(x=month,y=total_count,fill=act_holiday))+theme_bw()+geom_col()+
labs(x='Month',y='Total_Count',title='Holiday wise monthly distribution of counts')

# Scatterplot of Distribution of Temperature
ggplot(data = bike_pr_day, aes(x =temp, y = total_count)) + ggtitle("Distribution of Temperature") + geom_point() +
xlab("Temperature") + ylab("Bike COunt")

#Scatterplot of Distribution of Humidity
ggplot(data = bike_pr_day, aes(x =humidity, y = total_count)) + ggtitle("Distribution of Humidity") + geom_point(color="red") +
xlab("Humidity") + ylab("Bike COunt")

Scatterplot of Distribution of Windspeed
ggplot(data = bike_pr_day, aes(x =windspeed, y = total_count)) + ggtitle("Distribution of Windspeed") + geom_point(color="red")
+ xlab("Windspeed") + ylab("Bike COunt")
###############################################Outlier analysis##########################################
#boxplot for total_count_outliers
par(mfrow=c(1, 1))#divide graph area in 1 columns and 1 rows
boxplot(bike_pr_day$windspeed,main='Total_count',sub=paste(boxplot.stats(bike_pr_day$windspeed)$out))

boxplot(bike_pr_day$temp,main='Total_count',sub=paste(boxplot.stats(bike_pr_day$temp)$out))

boxplot(bike_pr_day$humidity,main='Total_count',sub=paste(boxplot.stats(bike_pr_day$humidity)$out))

```

```r
boxplot(bike_pr_day$total_count,main='Total_count',sub=paste(boxplot.stats(bike_pr_day$total_count)$out))

###############################################Outlier value imputation#####################################################

#create subset for windspeed and humidity variable
wind_hum<-subset(bike_pr_day,select=c('windspeed','humidity'))

cnames<-colnames(wind_hum)
for(i in cnames){
    val=wind_hum[,i][wind_hum[,i] %in% boxplot.stats(wind_hum[,i])$out] #outlier values
    wind_hum[,i][wind_hum[,i] %in% val]= NA  # Replace outliers with NA
}
#Imputating the outlier values using mean imputation method
wind_hum$windspeed[is.na(wind_hum$windspeed)]<-mean(wind_hum$windspeed,na.rm=T)
wind_hum$humidity[is.na(wind_hum$humidity)]<-mean(wind_hum$humidity,na.rm=T)
new_df<-subset(bike_pr_day,select=-c(windspeed,humidity))

bike_rent_df<-cbind(new_df,wind_hum)
head(bike_rent_df)

###################################Feature Selection################################################################
numeric_index= sapply(bike_rent_df,is.numeric) #selecting only numeric
## Correlation Plot
corrgram(bike_rent_df[,numeric_index], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")


#Create a new subset for training model
final_bkr_data<-subset(bike_rent_df,select=c('season','year','month','holiday',
'weekday','workingday','weather_condition','temp','humidity','windspeed','total_count'))


```

```
####################################Model Development######################################


#Divide data into train and test
set.seed(1234)
train.index = createDataPartition(final_bkr_data$total_count, p = .80, list = FALSE)
train = final_bkr_data[ train.index,]
test  = final_bkr_data[-train.index,]
str(train)

####################################Decision tree for regression##########################################
#
# lets develope decision rules for predicting a continuous (regression tree) outcome.

# ##rpart for regression


fit = rpart(total_count ~ ., data = train, method = "anova")


# decision tree model visualisation
par(cex= 0.8)
plot(fit)
text(fit)

#Predict for new test cases

predictions_DT = predict(fit, test[,-11])


```

```
####################Mean Absolute Percentage Error###########################################

error <- mape(predictions_DT,test$total_count)
error
accuracy <- (1-error)*100
accuracy


####################Root Mean square Error###########################################
rmse(actual = test$total_count,predicted = predictions_DT)



# Accuracy of decision tree is 83.0495305115145 %
##################################################################################################
# save the model to disk
saveRDS(fit, "./DT_model.rds")

# load the model
model <- readRDS("./DT_model.rds")

# make a predictions on new data using saved model
final_predictions <- predict(model, test[,-11])




# Actual value vs predicted value plot tells about variance between actual target value and predicted target value
plot(test$total_count,final_predictions,xlab='Actual value',ylab='predicted value',main='Actual value vs predicted value plot')
abline(0,0)
```

```
##############################################Random Forest##########################################################

library(randomForest)

rf <- randomForest(total_count ~ ., data = train, ntree=20)
predictions_RF = predict(rf, test[,-11])

####################Mean Absolute Percentage Error############################################

error <- mape(predictions_RF,test$total_count)
error
accuracy <- (1-error)*100
accuracy


####################Root Mean square Error############################################
rmse(actual = test$total_count,predicted = predictions_RF)



# Number of variables randomly sampled as candidates at each split. Note that the default values are different for
classification (sqrt(p)
# where p is number of variables in x) and regression (p/3)

rf_2=randomForest(total_count ~ . , data = train,mtry =4,ntree=100 ,nodesize =10 ,importance =TRUE)

predictions_RF2 = predict(rf_2, test[,-11])

####################Mean Absolute Percentage Error############################################

error <- mape(predictions_RF2,test$total_count)
error
```

```
error
accuracy <- (1-error)*100
accuracy


####################Root Mean square Error############################################
rmse(actual = test$total_count,predicted = predictions_RF2)



####################################################Regression Analysis#############################################
#the base function lm is used for regression.
regmodel <- lm(total_count ~ ., data = train)
summary(regmodel)


#check the residual plots, understand the pattern and derive actionable insights (if any):

par(mfrow=c(2,2))
#create residual plots
plot (regmodel)

# lets test model
regpred <- predict(regmodel, test[-11])

error <- mape(regpred,test$total_count)
error
accuracy <- (1-error)*100
accuracy
rmse(actual = test$total_count,predicted = regpred)
```

Original R code and python code is attached with this document