***CAB FARE PREDICTION***

***MOULEESHWARAN.S***

***16.10.2019***

**TABLE OF CONTENTS**

**1.INTRODUCTION .............................................................................3**

1.1 Problem Statement ................................................................3

1.2 Data overview ........................................................................3

**2.METHODOLOGY .............................................................................4**

2.1 Data Pre -Processing ...............................................................4

2.1.1 Data Exploration ..................................................................4

2.1.2 Missing Value Analysis .........................................................4

2.1.3 Outlier Analysis ....................................................................5

2.1.3a Creating New Variables………………………………………….5

2.1.4 Data visualization .................................................................5

2.1.4a Distribution of numerical variables………….……….…...5

2.1.4b Distribution of variables with respect to target variable…………………………………………………………………………….6

2.1.5 Feature Selection .................................................................8

2.1.5a Correlation matrix ………….…………………………………….8

2.1.5b Heat map……………………………………………………………….8

2.1.5c Analysis of Variance…………………………………………………9

2.1.5c Dimension Reduction……………………………………………….9

2.1.6 Feature Scaling ......................................................................10

2.2 model development .................................................................10

2.2.1 Linear Regression.........................................................10

2.2.2 Decision Tree................................................................11

2.2.3 Random Forest .............................................................11

2.3 Hyper parameters tuning……………………………………………………....11

**3 MODEL EVALUATION ........................................................................12**

3.1 Evaluation Metrics…………………………..........................................12

3.2 Model Selection ........................................................................12

3.3 Model Evaluation using Test Data………………………………………….13

**4. APPENDIX A – R-CODE .....................................................................14**

**5. APPENDIX B – PYTHON-CODE……………………….…………………………………36**

**6.REFERENCES......................................................................................61**

**1.INTRODUCTION**

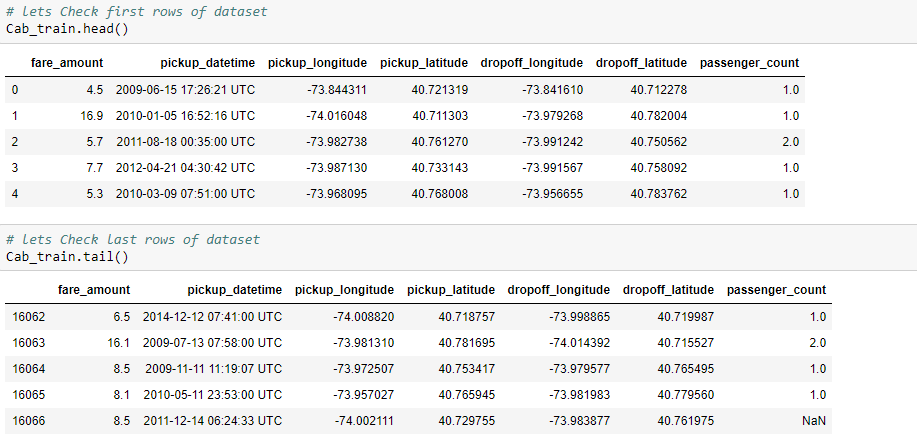
**1.1 PROBLEM STATEMENT:**

In our project we need to predict fare amount using the pilot project which is already run by our startup company.

**1.2 DATA OVERVIEW:**

We have 7 variables and 16067 observations. In that 6 variables are independent and 1 dependent variables.

Let’s have a look at the data:



Here fare amount is our dependent variables

Remaining all are independent variables.

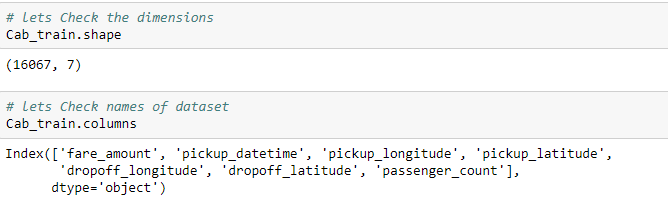
**2. METHODOLOGY**

**2.1 DATA PRE-PROCESSING:**

Data preprocessing is a data mining technique which transforms raw data into an understandable format. Data goes through series of steps during preprocessing. They are data cleaning, data visualization, data transformation, data reduction.

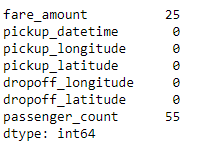
**2.1.1 DATA EXPLORATION:**

We need to check dimensions of the data, data types of the data, summary of the data .so that we can get good understandings about the data and also identify the target variable.



**2.1.2 MISSING VALUE ANALYSIS:**

Missing values are the data which is not present in the particular variable or observations. It may happen due to human error, or it may mark as an optional during the survey. If the data set contains missing values which is above 35%, either we need to drop the column or that particular observation. There are many methods to impute the missing values they are central tendency method and knn imputation.



In my dataset, fare amount is my target variable and passenger count play important role in prediction. Hence, I dropped the missing values instead of imputing because the missing value count is low.

**2.1.3 OUTLIER ANALYSIS:**

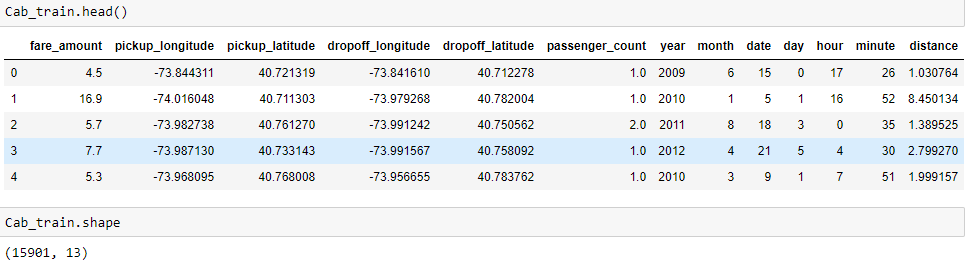
Basically, outliers are the values which are lying far away from the remaining variables which may lead biased towards the higher value which results in the performance of our model. So that we need to treat the outliers.

1. As we know we have some negative values in fare amount so we have to remove those values, since fare can’t be negative. There are some outlier figures in the fare (like 54,343, 4343) so we need to remove those.
2. Passenger count would be max 7 if it is a SUV vehicle not more than that. We have to remove the rows having passenger count more than 7 and less than 1.
3. Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rows if any latitude and longitude lies beyond the ranges.

**2.1.3.a) CREATING NEW VARIABLES:**

* Here in our data set our variable name pickup\_datetime which contains date and time for pickup. So, we tried to extract some important variables from pickup\_datetime.
* Let’s extract distance from the provided latitude and longitude using the haversine formula.

Let’s us have look of our data after the creation of new variables.



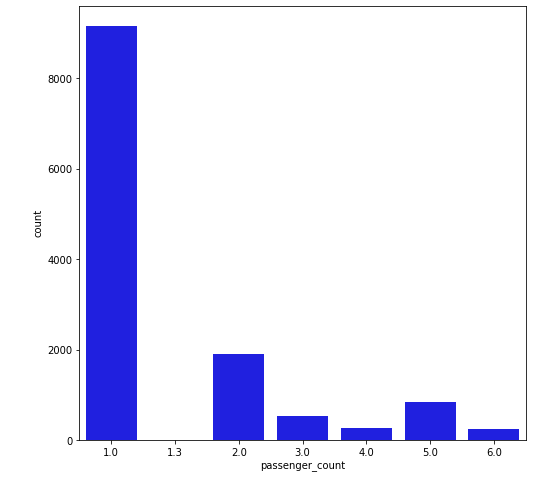
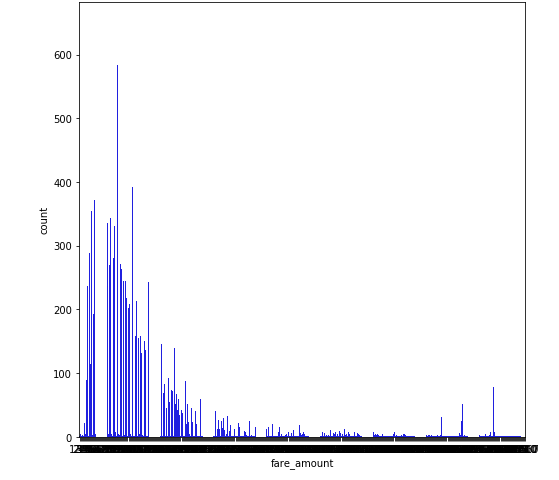
After variable creations we have 13 variables in which fare amount is our target variable.

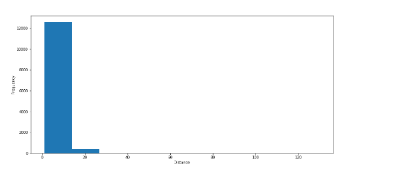
**2.1.4 DATA VISUALIZATION:**

**2.1.4a: DISTRIBUTION OF THE NUMERIC VARIABLE**:

Data visualization is the easy method to understand the data. It will give clear idea of our data and also impact of dependent variables.

Distribution plot helps us to know the distribution of the data and makes us easily understandable. So that it used in the both and Python languages.

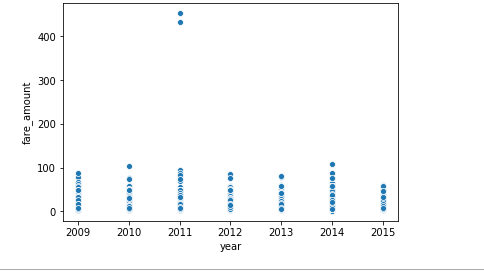


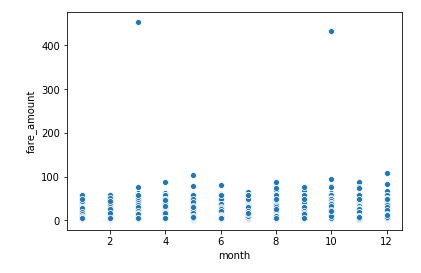


These are the distribution of numerical variables. here I plotted visualizations for distance, fare amount, passenger count.

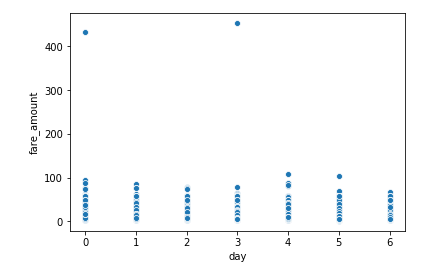
**2.1.4b: DISTRIBUTION OF VARIABLES WITH RESPECT TO TARGET VARIABLE:**

Here we used scatterplot our visualization. Here I plotted visualizations for both categorical as well numerical variables. Plots are shown below.

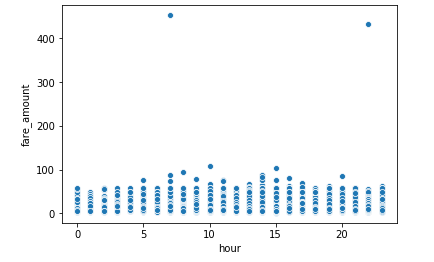


 **B) FARE\_AMOUNT VS MONTH**

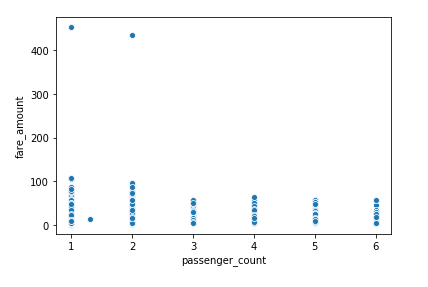
**A) FARE\_AMOUNT VS YEAR**



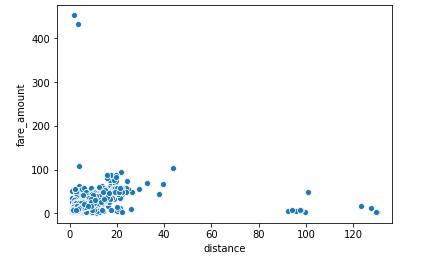
**C) FARE\_AMOUNT VS DAY**



**D) FARE\_AMOUNT VS HOUR**



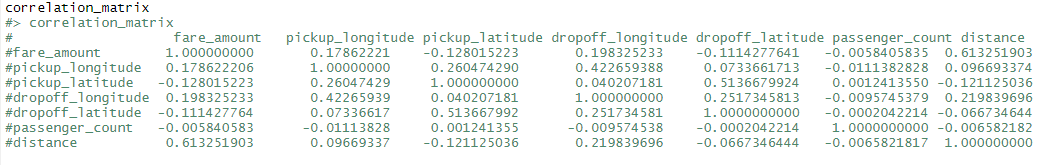
**E) FARE\_AMOUNT VS PASSENGER COUNT**



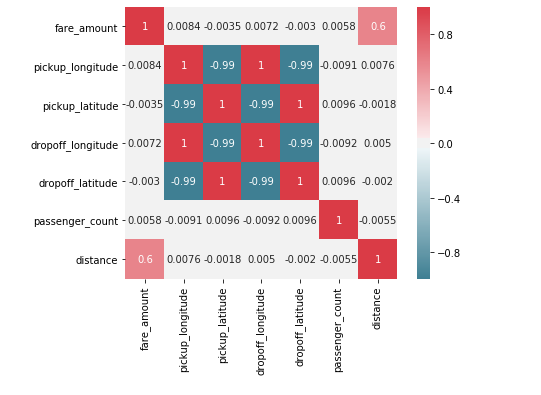
**D) FARE\_AMOUNT VS DISTANCE**

**2.1.5 FEATURE SELECTION:**

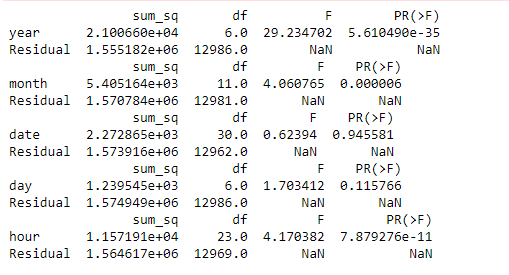
We can use correlation analysis for numerical variables and Analysis of Variance for categorical variables. It shows correlation between the two variables. So that if two variables carrying same information can be removed.

** 2.1.5a: CORRELATION MATRIX:**

**2.1.5b: HEAT MAP:**

****

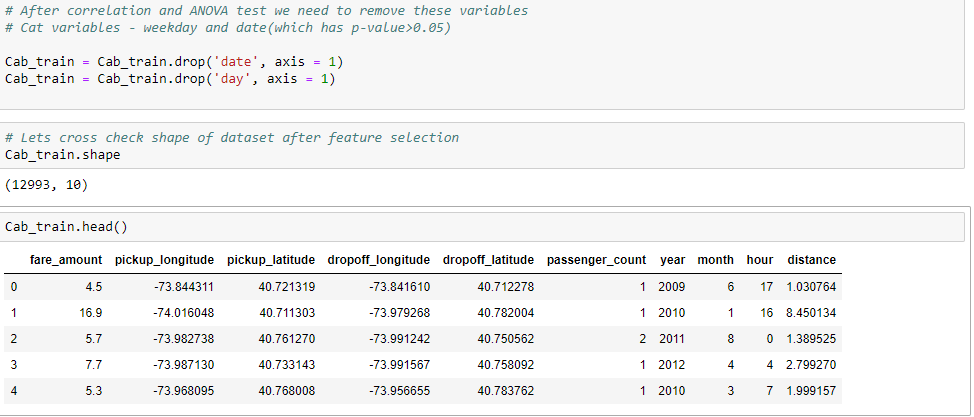
**2.1.5c.ANALYSIS OF VARIANCE:**

****

From the above diagram, date and day has p-value which is higher than 0.05.so that we need to drop these variables.

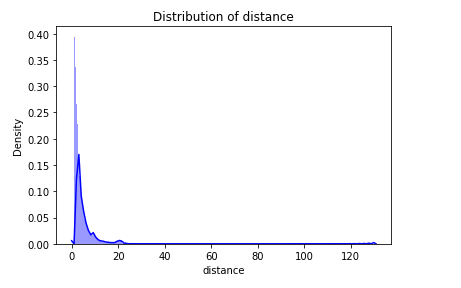
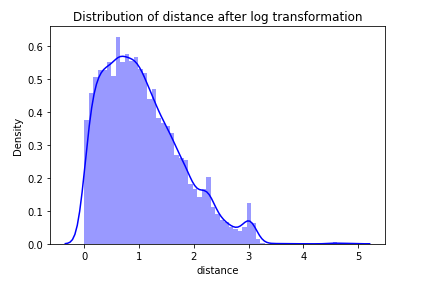
**2.1.5c DIMENSION REDUCTION:**

After the feature selection, we have only these 10 variables. They are mentioned in the below diagram.

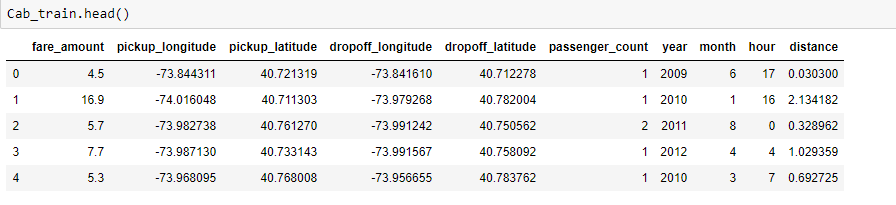


**2.1.6 FEATURE SCALING:**

In our dataset, fare\_ amount and distance variable distribution are left skewed. So, we need to take log for normalize the variable.



Dataset after log transformation.



**2.2 MODEL DEVELOPMENT:**

Next, we need to split the data into train and test data and build a model using train data to predict the output using test data. Different models to be built and the model which gives more accurate values must be selected.

**2.2.1 LINEAR REGRESSION:**

Linear regression is a basic and commonly used type of predictive analysis.  The overall idea of regression is to examine two things:

1. Does a set of predictor variables do a good job in predicting an outcome (dependent) variable?
2. Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable?

These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. We trained our model in both R and Python and predicted in these languages using test data.

**2.2.2 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 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.

**2.2.3. RANDOM FOREST:**

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees, which involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. The higher no of trees in the random forest will give higher no of accuracy, so in random forest we can go for multiple trees. It can handle large no of independent variables without variable deletion and it will give the estimates that what variables are important.

**2.3 HYPERPARAMETER TUNING**

In statistics, hyperparameter is a parameter from a prior distribution; it captures the prior belief before data is observed. In any machine learning algorithm, these parameters need to be initialized before training a model. Choosing appropriate hyperparameters plays a crucial role in the success of good model. Since it makes a huge impact on the learned model.

* + 1. **TUNING PARAMETERS:**

We will explore two different methods for optimizing hyperparameters:

* Grid Search
* Random Search

**2.3.1A). RANDOM SEARCH:**

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. In this search pattern, random combinations of parameters are considered in every iteration. The chances of finding the optimal parameter are comparatively higher in random search because of the random search pattern where the model might end up being trained on the optimised parameters without any aliasing.

**2.3.1.B) Grid Search**

Grid search is a technique which tends to find the right set of hyperparameters for the particular model. Hyperparameters are not the model parameters and it is not possible to find the best set from the training data. In this tuning technique, we simply build a model for every combination of various hyperparameters and evaluate each model. The model which gives the highest accuracy will be selected.

**3. MODEL EVALUATION**

**3.1 EVALUATION METRICS:**

In regression problems, we have three important metrics. They are

* MAPE (Mean Absolute Percentage Error)
* R-SQUARED
* RMSE (Root Mean Square Error)

**3.1.1 MAPE (Mean Absolute Percentage Error**)

MAPE is a measure of prediction accuracy of a forecasting method. It measures accuracy in terms of percentage. Lower value of MAPE indicates better fit.

**3.1.2 R-SQUARED**

R-squared is basically explains the degree to which input variable explain the variation of the output. In simple words R-squared tells how much variance of dependent variable explained by the independent variable. It is a measure if goodness of fit in regression line. Higher values of R-square indicate better fit.

**3.1.3 RMSE (Root Mean Square Error)**

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are, RMSE is a measure of how spread out these residuals are. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit.

**3.2 MODEL SELECTION:**

From the predicted output in R and Python, the random forest model can have explained almost 90% of the predictor matches with the target variable. The values of the random forest model are mentioned below.

**METRICS VALUES USING PYTHON:**

* MAPE = 0.82
* R-SQUARED =0.75
* RMSE = 4.83

**METRICS VALUES USING R:**

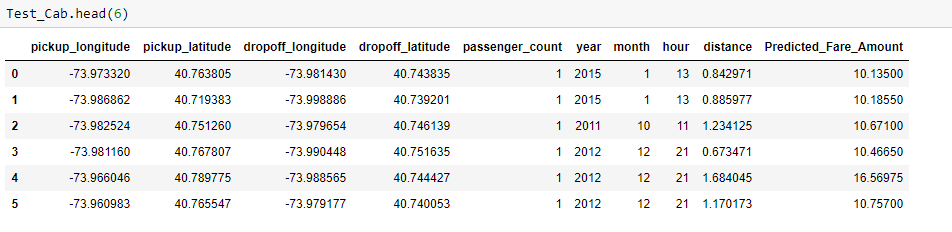
* MAPE = 0.80
* R-SQUARED =0.71
* RMSE = 5.07

**3.2.1 MODEL EVALUATION USING GIVEN TEST DATA:**

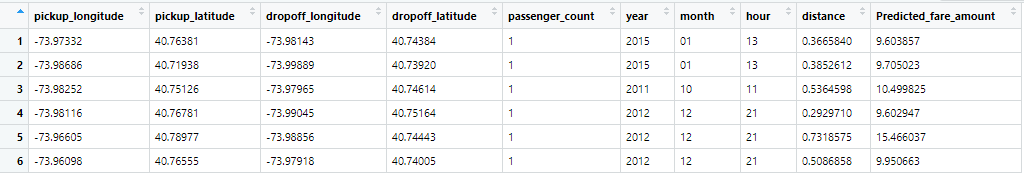
we have predicted random forest model performs well and we will apply it in our given test data and predict the fare amount.

let’s have a final data with predicted fare amount.

**IN PYTHON:**



IN R:



**4. R-CODE**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Cab fare Prediction

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Clean the environment

rm(list=ls())

# Set working directory

setwd("D:/Data Science/Cab Fare")

# Load required Libraries for analysis ----------------------------------

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced","C50",

"dummies", "e1071", "Information", "MASS", "rpart", "gbm", "ROSE",

'sampling', 'DataCombine', 'inTrees',"scales","gplots")

#install.packages(x)

lapply(x, require, character.only = TRUE)

rm(x)

# Load the data -----------------------------------------------------------

Train\_Cab = read.csv("train\_cab.csv")

# Explore the data -------------------------------------------------------

# Check class of the data

class(Train\_Cab)

#Check the dimensions(no of rows and no of columns)

dim(Train\_Cab)

#Check names of dataset(no need of renaming variables)

names(Train\_Cab)

#Check top(first) rows of dataset

head(Train\_Cab)

#Check bottom(last) rows of dataset

tail(Train\_Cab)

#Check structure of dataset(data structure of each variable)

str(Train\_Cab)

#Check summary of dataset

summary(Train\_Cab)

# Variable Identification

# Target variable - fare\_amount

str(Train\_Cab$fare\_amount) # fare\_amount is a continous variabe

# Data type conversion

# As observed, we have to change fare\_amount from factor to numeric

Train\_Cab$fare\_amount = as.numeric(as.character(Train\_Cab$fare\_amount))

summary((Train\_Cab$fare\_amount))

# we need to convert pickupdatetime as well we did it in feature engineering

# Missing Value Analysis --------------------------------------------------

# Total number of missing values present in whole datset

Missing\_val = sum(is.na(Train\_Cab))

Missing\_val

#lets drop those missing values

Train\_Cab = na.omit(Train\_Cab)

#lets verify after dropping missing values.

sum(is.na(Train\_Cab))

# We need to change pickup\_datetime from factor to datetime

# But first, let's replace UTC in pickup\_datetime variable with ''(space)

Train\_Cab$pickup\_datetime = gsub('// UTC','',Train\_Cab$pickup\_datetime)

# Now convert variable pickup\_dattime to date time format by creating

# new variable with name Date

Train\_Cab$date = as.Date(Train\_Cab$pickup\_datetime)

# Lets split this new variable Date into year,month,weekday

# Extract the year

Train\_Cab$year = substr(as.character(Train\_Cab$date),1,4)

# Extract the month

Train\_Cab$month =substr(as.character(Train\_Cab$date),6,7)

# Extract the weekday

Train\_Cab$day = weekdays(as.POSIXct(Train\_Cab$date),abbreviate = F)

# Extract the date

Train\_Cab$date = substr(as.character(Train\_Cab$date),9,10)

# Extract the time / hour

Train\_Cab$hour = substr(as.factor(Train\_Cab$pickup\_datetime),12,13)

#Lets delete picupdate time as we converted this variable into day,month,year,hour

Train\_Cab$pickup\_datetime = NULL

dim(Train\_Cab)

head(Train\_Cab)

# Lets check summary again after new feature creation

summary(Train\_Cab)

# Lets check for NA after conversion.if present removing those variables.

sum(is.na(Train\_Cab))

Train\_Cab = na.omit(Train\_Cab)

#lets check the dimension after removing missing values

dim(Train\_Cab)

# Outlier analysis --------------------------------------------------------

# Boxplots-Distribution and outlier check

numeric\_index = sapply(Train\_Cab,is.numeric)# Selecting only numeric

numeric\_index

numeric\_data =Train\_Cab[,numeric\_index]

cnames = colnames(numeric\_data)

cnames

# Boxplot for all continous varaibles

for (i in 1:length(cnames))

{

assign(paste0("gn",i), ggplot(aes\_string(y = cnames[i]), data = Train\_Cab)+

stat\_boxplot(geom = "errorbar", width = 0.5) +

geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,

outlier.size=1, notch=FALSE) +

theme(legend.position="bottom")+

labs(y=cnames[i])+

ggtitle(paste("Box plot for",cnames[i])))

}

gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,gn6,ncol=2)

# We have different methods to remove outliers.

#here i am going to remove all outliers one by one and

#capping with minimum and maximum values for some variables

###### fare\_amount ######

# Let use summary function to check min max values and identify outliers

summary(Train\_Cab$fare\_amount)

#by seeing the summary we say that amount cant be in negative value as well as fare amount cannot exceed 500

Train\_Cab$fare\_amount[Train\_Cab$fare\_amount < 1] = NA

Train\_Cab$fare\_amount[Train\_Cab$fare\_amount >500] = NA

sum(is.na(Train\_Cab))

Train\_Cab = na.omit(Train\_Cab)

#lets very after performing outlier analysis

summary(Train\_Cab$fare\_amount)

------------------------------------------------------------------------------------

# pickup\_longitude

------------------------------------------------------------------------------------

#Originally, Latitudes range from -90 to 90.

#Originally, Longitudes range from -180 to 180.

summary(Train\_Cab$pickup\_longitude)

Q1 <- quantile(Train\_Cab$pickup\_longitude,0.25)#-73.99126

Q3 <- quantile(Train\_Cab$pickup\_longitude,0.75)#-73.96684

UL <- Q3 + (1.5\*IQR(Train\_Cab$pickup\_longitude))# -73.92884

LL <- Q1-(1.5\*IQR(Train\_Cab$pickup\_longitude)) # -74.03013

Train\_Cab[Train\_Cab$pickup\_longitude < LL ,"pickup\_longitude"] <- LL

Train\_Cab[Train\_Cab$pickup\_longitude > UL ,"pickup\_longitude"] <- UL

---------------------------------------------------------------------------------------

# pickup\_latitude

---------------------------------------------------------------------------------------

summary(Train\_Cab$pickup\_latitude)

Q1 = quantile(Train\_Cab$pickup\_latitude,0.25)

Q3 = quantile(Train\_Cab$pickup\_latitude,0.75)

UL = Q3 + (1.5\*IQR(Train\_Cab$pickup\_latitude))

LL = Q1 - (1.5\*IQR(Train\_Cab$pickup\_latitude))

Train\_Cab[Train\_Cab$pickup\_latitude < LL,"pickup\_latitude"] = LL

Train\_Cab[Train\_Cab$pickup\_latitude > UL,"pickup\_latitude"] = UL

#### Dropoff longitude #####

summary(Train\_Cab$dropoff\_longitude)

Q1 = quantile(Train\_Cab$dropoff\_longitude,0.25)

Q3 = quantile(Train\_Cab$dropoff\_longitude,0.75)

UL = Q3 + (1.5\*IQR(Train\_Cab$dropoff\_longitude))

LL = Q1 - (1.5\*IQR(Train\_Cab$dropoff\_longitude))

Train\_Cab[Train\_Cab$dropoff\_longitude < LL,"dropoff\_longitude"] = LL

Train\_Cab[Train\_Cab$dropoff\_longitude > UL,"dropoff\_longitude"] = UL

-----------------------------------------------------------------------------------------

# dropoff\_lattitude

-----------------------------------------------------------------------------------------

summary(Train\_Cab$dropoff\_latitude)

Q1 = quantile(Train\_Cab$dropoff\_latitude,0.25)

Q3 = quantile(Train\_Cab$dropoff\_latitude,0.75)

UL = Q3 + (1.5\*IQR(Train\_Cab$dropoff\_latitude))

LL = Q1 - (1.5\*IQR(Train\_Cab$dropoff\_latitude))

Train\_Cab[Train\_Cab$dropoff\_latitude < LL,"dropoff\_latitude"] = LL

Train\_Cab[Train\_Cab$dropoff\_latitude > UL,"dropoff\_latitude"] = UL

# passenger\_count

summary(Train\_Cab$passenger\_count)

# practically maximum 6 passenger can travel in a cab

Train\_Cab[Train\_Cab$passenger\_count < 1,"passenger\_count"] = NA

Train\_Cab[Train\_Cab$passenger\_count > 6,"passenger\_count"] = NA

sum(is.na(Train\_Cab))

Train\_Cab = na.omit(Train\_Cab)

#lets very after performing outlier analysis

summary(Train\_Cab$passenger\_count)

# Lets visualize boxplots again after outlier removal

for (i in 1:length(cnames))

{

assign(paste0("gn",i), ggplot(aes\_string(y = cnames[i]), data = Train\_Cab)+

stat\_boxplot(geom = "errorbar", width = 0.5) +

geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,

outlier.size=1, notch=FALSE) +

theme(legend.position="bottom")+

labs(y=cnames[i])+

ggtitle(paste("Box plot for",cnames[i])))

}

gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,gn6,ncol=2)

# Lets check dimensions of data after outlier removal

dim(Train\_Cab)

# Now, let's create distance using Haversine Formula

# Calculates the geodesic distance between two points specified by

# radian latitude/longitude using the Haversine formula

library(purrr)

library(geosphere)

library(rlist)

get\_geo\_distance = function(long1, lat1, long2, lat2) {

loadNamespace("purrr")

loadNamespace("geosphere")

longlat1 = purrr::map2(long1, lat1, function(x,y) c(x,y))

longlat2 = purrr::map2(long2, lat2, function(x,y) c(x,y))

distance\_list = purrr::map2(longlat1, longlat2, function(x,y) geosphere::distHaversine(x, y))

distance\_m = list.extract(distance\_list, position = 1)

#if (units == "km") {

distance = distance\_m / 1000.0;

distance

}

# Applying distance formula for train data

for(i in (1:nrow(Train\_Cab)))

{

Train\_Cab$distance[i]= get\_geo\_distance(Train\_Cab$pickup\_longitude[i],Train\_Cab$pickup\_latitude[i],Train\_Cab$dropoff\_longitude[i],Train\_Cab$dropoff\_latitude[i])

}

# Lets check data after distance variable creation

head(Train\_Cab)

# Lets check whether distance variables has any outlier using boxplot

ggplot(aes\_string(y = 'fare\_amount', x = "distance"), data = subset(Train\_Cab))+

stat\_boxplot(geom = "errorbar", width = 0.25) +

geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,

outlier.size=1, notch=FALSE) +theme(legend.position="bottom")+

ggtitle(paste("Box plot of distnce variable with outliers"))

# Lets check summary of distance variable no outliers

summary(Train\_Cab$distance)

# we can notice this variable has values which are less than 1 around 2978 in No

# we will replace these values with average distance as the numner of 0's are more

length(Train\_Cab$distance[Train\_Cab$distance < 1])#2978

Train\_Cab$distance[Train\_Cab$distance < 1] = mean(Train\_Cab$distance)

summary(Train\_Cab$distance)

# The data left after all preprocessing

df = Train\_Cab

dim(df)

head(Train\_Cab)

# Exploratory Analysis with visualizations after data cleaning --------------------------------

# Lets check distribution of each numeric and categorical variables

names(Train\_Cab)

# Univariate Analysis of continous variables

# Lets save numeric varaibles

num\_var = c("fare\_amount","pickup\_longitude" ,"pickup\_latitude", "dropoff\_longitude",

"dropoff\_latitude","passenger\_count","distance")

# Histogram for continuous variables to check distribution of each variable

# fare\_amount

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$fare\_amount)) +

geom\_histogram(fill="skyblue", colour = "black") + geom\_density() +

theme\_bw() + xlab("Fare amount") + ylab("Frequency")+ggtitle("distribution of fare\_amount")

#right skewed

# pickup\_longitude

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$pickup\_longitude)) +

geom\_histogram(fill="skyblue", colour = "black",bins = 100,boundry =1) + geom\_density() +

theme\_bw() + xlab("pickup\_longitude") + ylab("Frequency")+ggtitle("distribution of pickup\_longitude")

# almost normally distributed

# pickup\_latitude

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$pickup\_latitude)) +

geom\_histogram(fill="skyblue", colour = "black",bins = 100) + geom\_density() +

theme\_bw() + xlab("pickup\_latitude") + ylab("Frequency")+ggtitle("Frequency of pickup\_latitude")

# almost normally distributed

# dropoff\_longitude

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$dropoff\_longitude)) +

geom\_histogram(fill="skyblue", colour = "black",bins = 100) + geom\_density() +

theme\_bw() + xlab("dropoff\_longitude") + ylab("Frequency")+ggtitle("Frequency of dropoff\_longitude")

# almost normally distributed

# dropoff\_latitude

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$dropoff\_latitude)) +

geom\_histogram(fill="skyblue", colour = "black",bins = 100) + geom\_density() +

theme\_bw() + xlab("dropoff\_latitude") + ylab("Frequency")+ggtitle("Frequency of dropoff\_latitude")

# almost normally distributed

# passenger\_count

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$passenger\_count)) +

geom\_histogram(fill="skyblue", colour = "black") + geom\_density() +

theme\_bw() + xlab("passenger\_count") + ylab("Frequency")+ggtitle("distribution of passenger\_count")

# distance

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$distance)) +

geom\_histogram(fill="skyblue", colour = "black") + geom\_density() +

theme\_bw() + xlab("distance") + ylab("Frequency")+ggtitle(" distribution of distance ")

#right skewed

# Bivariate Analysis ------------------------------------------------------

# Bar plot for categorical and target variables

# Visualization between fare\_amount and years.

ggplot(data = Train\_Cab, aes(x = year,y = fare\_amount))+

geom\_bar(stat = "identity",color ="DarkSlateBlue")+

labs(title = "Fare Amount Vs. year", x = "years", y = "Fare")+

theme(plot.title = element\_text(hjust = 0.5, face = "bold"))+

theme(axis.text.x = element\_text( color="black", size=6, angle=45))

# We can see, in year 2013 there were rides which got high fare\_amount

# Visualization between fare\_amount and months.

ggplot(Train\_Cab, aes(x = month,y = fare\_amount))+

geom\_bar(stat = "identity",color = "DarkSlateBlue")+

labs(title = "Fare Amount Vs. Month", x = "Month", y = "Fare")+

theme(axis.text.x = element\_text( color="navy blue", size=8))

# We can see month March collects the highest fare\_amount

# Visualization between fare\_amount and weekday.

ggplot(data = Train\_Cab, aes(x = day,y = fare\_amount))+

geom\_bar(stat = "identity",color = "DarkSlateBlue")+

labs(title = "Fare Amount Vs. weekday", x = "Days of the week", y = "Fare")+

theme(plot.title = element\_text(hjust = 0.5, face = "bold"))+

theme(axis.text.x = element\_text( color="black", size=6, angle=45))

# We can see that Thursday to Saturday rides has the highest fare\_amount

# Visualization between fare\_amount and time.

ggplot(data = Train\_Cab, aes(x = hour, y = fare\_amount))+

geom\_bar(stat = "identity",color = "DarkSlateBlue")+

labs(title = "Fare Amount Vs.hour", x = "hour", y = "Fare")+

theme(plot.title = element\_text(hjust = 0.5, face = "bold"))+

theme(axis.text.x = element\_text( color="black", size=6, angle=45))

# Rides taken during 6 pm to 10 pm gives highest fare\_amount

# Lets plot scatter plot for target and continous variables

# Visualization between fare\_amount and pickup\_longitude.

ggplot(Train\_Cab,aes(pickup\_longitude,fare\_amount)) +

geom\_point(alpha=0.5,color="DarkSlateBlue") +

labs(title = "Scatter Plot b/w pickup\_longitude and fare", x = "pickup\_longitude", y = "Fare Amount")+

scale\_color\_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +

theme\_bw()

ggplot(Train\_Cab,aes(pickup\_latitude,fare\_amount)) +

geom\_point(alpha=0.5,color="DarkSlateBlue") +

labs(title = "Scatter Plot b/w pickup\_latitude and fare", x = "pickup\_latitude", y = "Fare Amount")+

scale\_color\_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +

theme\_bw()

ggplot(Train\_Cab,aes(dropoff\_longitude,fare\_amount)) +

geom\_point(alpha=0.5,color="DarkSlateBlue") +

labs(title = "Scatter Plot b/w dropoff\_longitude and fare", x = "dropoff\_longitude", y = "Fare Amount")+

scale\_color\_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +

theme\_bw()

ggplot(Train\_Cab,aes(dropoff\_latitude,fare\_amount)) +

geom\_point(alpha=0.5,color="DarkSlateBlue") +

labs(title = "Scatter Plot b/w dropoff\_latitude and fare", x = "dropoff\_latitude", y = "Fare Amount")+

scale\_color\_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +

theme\_bw()

ggplot(Train\_Cab,aes(passenger\_count,fare\_amount)) +

geom\_point(alpha=0.5,color="DarkSlateBlue") +

labs(title = "Scatter Plot b/w passengercount and fare", x = "passenger\_count", y = "Fare Amount")+

scale\_color\_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +

theme\_bw()

# single passenger are frequent travellers

ggplot(Train\_Cab,aes(distance,fare\_amount)) +

geom\_point(alpha=0.5,color="DarkSlateBlue") +

labs(title = "Scatter Plot b/w distance and fare", x = "Distance", y = "Fare Amount")+

scale\_color\_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark orange','black')) +

theme\_bw()

# We can say as distance increases fare amount also increases

# Feature selection ------------------------------------------------------

numeric\_index = sapply(Train\_Cab,is.numeric)# Selecting only numeric

numeric\_index

numeric\_data =Train\_Cab[,numeric\_index]

cnames = colnames(numeric\_data)

cnames

# Correlation Plot for to select significant continous variables

#correlation matrix

correlation\_matrix = cor(Train\_Cab[,cnames])

correlation\_matrix

#> correlation\_matrix

# fare\_amount pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude passenger\_count distance

#fare\_amount 1.000000000 0.17862221 -0.128015223 0.198325233 -0.1114277641 -0.0058405835 0.613251903

#pickup\_longitude 0.178622206 1.00000000 0.260474290 0.422659388 0.0733661713 -0.0111382828 0.096693374

#pickup\_latitude -0.128015223 0.26047429 1.000000000 0.040207181 0.5136679924 0.0012413550 -0.121125036

#dropoff\_longitude 0.198325233 0.42265939 0.040207181 1.000000000 0.2517345813 -0.0095745379 0.219839696

#dropoff\_latitude -0.111427764 0.07336617 0.513667992 0.251734581 1.0000000000 -0.0002042214 -0.066734644

#passenger\_count -0.005840583 -0.01113828 0.001241355 -0.009574538 -0.0002042214 1.0000000000 -0.006582182

#distance 0.613251903 0.09669337 -0.121125036 0.219839696 -0.0667346444 -0.0065821817 1.000000000

#correlation plot

corrgram(Train\_Cab[,numeric\_index],order = F,upper.panel = panel.pie,

text.panel = panel.txt,main = 'Correlation plot')

# We can say distance variable is moderately correlated with fare amount

# rest of the variables also correlated positively and negative but we can

# say them as weakly correlated we can use them in model

# Anova Test is performed between cat\_var (categorical independent variables) & fare\_amount (continuous target variable)

str(Train\_Cab)

names(Train\_Cab)

cat\_var =c("date","year","month","day","hour")

# aov(Train\_Cab$fare\_amount~Train\_Cab$year)

# for all categorical variables

for(i in cat\_var){

print(i)

Anova\_test\_result = summary(aov(formula = fare\_amount~Train\_Cab[,i],Train\_Cab))

print(Anova\_test\_result)

}

names(Train\_Cab)

# From the anova result, we can observe Date and day

# has p value > 0.05, so delete this variable not consider in model.

# lets delete date and day variable

Train\_Cab$day = NULL

Train\_Cab$date = NULL

head(Train\_Cab)

# Feature Scaling ---------------------------------------------------------

# In our dataset fare amount and distance are the two continous

# variables whose disribution is slightly skewed

# Checking distance variable distribution using histogram

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$distance)) +

geom\_histogram(fill="skyblue", colour = "black") + geom\_density() +

theme\_bw() + xlab("distance") + ylab("Frequency")+ggtitle(" distribution of distance ")

# this variable is right skewed

# Lets take log transformation to remove skewness

# Lets define function for log transformation of variables

signedlog10 = function(x) {

ifelse(abs(x) <= 1, 0, sign(x)\*log10(abs(x)))

}

# Applying log function to distance variable

Train\_Cab$distance = signedlog10(Train\_Cab$distance)

# Checking distance distribution after applying function

ggplot(Train\_Cab, aes\_string(x = Train\_Cab$distance)) +

geom\_histogram(fill="skyblue", colour = "black") + geom\_density() +

theme\_bw() + xlab("distance") + ylab("Frequency")+ggtitle(" distribution of distance after log transformation")

head(Train\_Cab)

# Model development -------------------------------------------------------

# Let's clean R Environment, as it uses RAM which is limited

library(DataCombine)

rmExcept("Train\_Cab")

# Split the data set into train and test

set.seed(1234)

train.index = createDataPartition(Train\_Cab$fare\_amount, p = .80, list = FALSE)

train\_data = Train\_Cab[train.index,]

test\_data = Train\_Cab[-train.index,]

#check multicollinearity

library(usdm)

vif(Train\_Cab[,cnames])

vifcor(Train\_Cab[,cnames], th = 0.9)

#No variable from the 7 input variables has collinearity problem.

#The linear correlation coefficients ranges between:

# min correlation ( distance ~ passenger\_count ): 0.0003133683

#max correlation ( dropoff\_latitude ~ pickup\_latitude ): 0.5237966

#---------- VIFs of the remained variables --------

# Variables VIF

#1 fare\_amount 1.410975

#2 pickup\_longitude 1.385007

#3 pickup\_latitude 1.599655

#4 dropoff\_longitude 1.426348

#5 dropoff\_latitude 1.565798

#6 passenger\_count 1.000346

#7 distance 1.396560

# Linear Regression model -------------------------------------------------

# fit linear regression model

# we will use the lm() function in the stats package

lm\_model = lm(fare\_amount ~.,data = train\_data)

summary(lm\_model)

#Residual standard error: 9.153 on 12691 degrees of freedom

#Multiple R-squared: 0.3237, Adjusted R-squared: 0.3212

#F-statistic: 132 on 46 and 12691 DF, p-value: < 2.2e-16

# Function for Error metrics to calculate the performance of model

#lets build function for MAPE

#calculate MAPE

MAPE = function(y, y1){

mean(abs((y - y1)/y))

}

# Function for r2 to calculate the goodness of fit of model

rsquare=function(y,y1){

cor(y,y1)^2

}

# Function for RMSE value

RMSE = function(y,y1){

difference = y - y1

root\_mean\_square = sqrt(mean(difference^2))

}

#lets predict for train and test data

Predictions\_LR\_train = predict(lm\_model,train\_data)

Predictions\_LR\_test = predict(lm\_model,test\_data)

#let us check performance of our model

#let us check performance of our model

#mape calculation

LR\_train\_mape = MAPE(Predictions\_LR\_train,train\_data[,1])

LR\_test\_mape = MAPE(test\_data[,1],Predictions\_LR\_test)

#Rsquare calculation

LR\_train\_r2 = rsquare(train\_data[,1],Predictions\_LR\_train)

LR\_test\_r2 = rsquare(test\_data[,1],Predictions\_LR\_test)

#rmse calculation

LR\_train\_rmse = RMSE(train\_data[,1],Predictions\_LR\_train)

LR\_test\_rmse = RMSE(test\_data[,1],Predictions\_LR\_test)

print(LR\_train\_mape)#1.26

print(LR\_test\_mape)#0.527

print(LR\_train\_r2)#0.32

print(LR\_test\_r2)#0.44

print(LR\_train\_rmse)#9.13

print(LR\_test\_rmse)#7.06

#Decision tree regression

library(rpart)

DT\_model = rpart(fare\_amount ~ ., data = train\_data, method = "anova")

DT\_model

# Lets predict for train and test data

predictions\_DT\_train= predict(DT\_model,train\_data)

predictions\_DT\_test= predict(DT\_model,test\_data)

# MAPE calculation

DT\_train\_mape = MAPE(train\_data[,1],predictions\_DT\_train)

DT\_test\_mape = MAPE(test\_data[,1],predictions\_DT\_test)

# Rsquare calculation

DT\_train\_r2= rsquare(train\_data[,1],predictions\_DT\_train)

DT\_test\_r2 = rsquare(test\_data[,1],predictions\_DT\_test)

# RMSE calculation

DT\_train\_rmse = RMSE(train\_data[,1],predictions\_DT\_train)

DT\_test\_rmse = RMSE(test\_data[,1],predictions\_DT\_test)

print(DT\_train\_mape)#0.338

print(DT\_test\_mape)#0.324

print(DT\_train\_r2)#0.456

print(DT\_test\_r2)#0.599

print(DT\_train\_rmse)#8.19

print(DT\_test\_rmse)#6.02

#Random search CV in decision tree

#setting parameters for training using caret library

control = trainControl(method="repeatedcv", number=5, repeats=1,search='random')

maxdepth = c(1:30)

tunegrid = expand.grid(.maxdepth=maxdepth)

# Lets build a model using above parameters on train data

RDT\_model = caret::train(fare\_amount~., data=train\_data, method="rpart2",trControl=control,tuneGrid= tunegrid)

print(RDT\_model)

#lets look best parameter

best\_parameter = RDT\_model$bestTune

print(best\_parameter)

#maximum depth =11

#again build a decsion tree using best parameters

RDT\_bestmodel = rpart(fare\_amount~.,train\_data,method = 'anova',maxdepth=11)

print(RDT\_bestmodel)

#lets predict for train and test data

predictions\_RDT\_train = predict(RDT\_bestmodel,train\_data)

predictions\_RDT\_test = predict(RDT\_bestmodel,test\_data)

#model performance

# MAPE calculation

RDT\_train\_mape = MAPE(train\_data[,1],predictions\_RDT\_train)

RDT\_test\_mape = MAPE(test\_data[,1],predictions\_RDT\_test)

# Rsquare calculation

RDT\_train\_r2= rsquare(train\_data[,1],predictions\_RDT\_train)

RDT\_test\_r2 = rsquare(test\_data[,1],predictions\_RDT\_test)

# RMSE calculation

RDT\_train\_rmse = RMSE(train\_data[,1],predictions\_RDT\_train)

RDT\_test\_rmse = RMSE(test\_data[,1],predictions\_RDT\_test)

print(RDT\_train\_mape)#0.338

print(RDT\_test\_mape)#0.324

print(RDT\_train\_r2)#0.456

print(RDT\_test\_r2)#0.599

print(RDT\_train\_rmse)#8.19

print(RDT\_test\_rmse)#6.02

#Grid search CV decision tree

#setting parameters for training using caret library

control = trainControl(method="repeatedcv", number=5, repeats=2,search='grid')

maxdepth = c(6:30)

tunegrid = expand.grid(maxdepth=maxdepth)

# Lets build a model using above parameters on train data

GDT\_model = caret::train(fare\_amount~., data=train\_data, method="rpart2",trControl=control,tuneGrid= tunegrid)

print(GDT\_model)

#lets look best parameter

best\_parameter = GDT\_model$bestTune

print(best\_parameter)

#maximum depth = 12

#again build a decsion tree using best parameters

GDT\_bestmodel = rpart(fare\_amount~.,train\_data,method = 'anova',maxdepth=12)

print(GDT\_bestmodel)

#lets predict for train and test data

predictions\_GDT\_train = predict(GDT\_bestmodel,train\_data)

predictions\_GDT\_test = predict(GDT\_bestmodel,test\_data)

#model performance

# MAPE calculation

GDT\_train\_mape = MAPE(train\_data[,1],predictions\_GDT\_train)

GDT\_test\_mape = MAPE(test\_data[,1],predictions\_GDT\_test)

# Rsquare calculation

GDT\_train\_r2= rsquare(train\_data[,1],predictions\_GDT\_train)

GDT\_test\_r2 = rsquare(test\_data[,1],predictions\_GDT\_test)

# RMSE calculation

GDT\_train\_rmse = RMSE(train\_data[,1],predictions\_GDT\_train)

GDT\_test\_rmse = RMSE(test\_data[,1],predictions\_GDT\_test)

print(GDT\_train\_mape)#0.338

print(GDT\_test\_mape)#0.324

print(GDT\_train\_r2)#0.456

print(GDT\_test\_r2)#0.599

print(GDT\_train\_rmse)#8.19

print(GDT\_test\_rmse)#6.02

#RANDOM FOREST

#lets build the random forest model

RF\_model = randomForest(fare\_amount~.,data = train\_data,n.trees = 500)

print(RF\_model)

#Call:

# randomForest(formula = fare\_amount ~ ., data = train\_data, n.trees = 500)

#Type of random forest: regression

#Number of trees: 500

#No. of variables tried at each split: 3

#Mean of squared residuals: 58.7358 % Var explained: 52.41

#lets predict for both train and test data

predictions\_RF\_train = predict(RF\_model,train\_data)

predictions\_RF\_test = predict(RF\_model,test\_data)

#MAPE calculation

RF\_train\_mape = MAPE(predictions\_RF\_train,train\_data[,1])

RF\_test\_mape = MAPE(predictions\_RF\_test,test\_data[,1])

#Rsquare calculation

RF\_train\_r2 = rsquare(predictions\_RF\_train,train\_data[,1])

RF\_test\_r2 = rsquare(predictions\_RF\_test,test\_data[,1])

#RMSE calculation

RF\_train\_rmse = RMSE(train\_data[,1],predictions\_RF\_train)

RF\_test\_rmse = RMSE(test\_data[,1],predictions\_RF\_test)

print(RF\_train\_mape)#0.09

print(RF\_test\_mape)#0.20

print(RF\_train\_r2)#0.910

print(RF\_test\_r2)#0.715

print(RF\_train\_rmse)#3.67

print(RF\_test\_rmse)#5.077

#Random search CV random forest= 0.01,trace = T,plot = T)

control = trainControl(method="repeatedcv", number=5, repeats=3,search='random')

#maxdepth = c(1:30)

#tunegrid = expand.grid(maxdepth=maxdepth)

#lets build Random forest model using the above parameters

RRF\_model = caret::train(fare\_amount~.,data=train\_data,method ='rf',trcontrol=control)

print(RRF\_model)

best\_parameter = RRF\_model$bestTune

print(best\_parameter)

#mtry = 13

#lets again build the random forest by above paremeters

RRF\_bestmodel = randomForest(count~.,data = train\_data,method = 'rf',mtry = 13,importance = TRUE)

print(RRF\_bestmodel)

#lets predict for both train and test data

prediction\_RRF\_train = predict(RRF\_bestmodel,train\_data)

prediction\_RRF\_test = predict(RRF\_bestmodel,test\_data)

#MAPE calculation

RRF\_train\_mape = MAPE(train\_data[,1],prediction\_RRF\_train)

RRF\_test\_mape = MAPE(test\_data[,1],prediction\_RRF\_test)

#Rsquare calculation

RRF\_train\_r2 = rsquare(train\_data[,1],prediction\_RRF\_train)

RRF\_test\_r2 = rsquare(test\_data[,1],prediction\_RRF\_test)

#RMSE calculation

RRF\_train\_rmse = RMSE(train\_data[,1],prediction\_RRF\_train)

RRF\_test\_rmse = RMSE(test\_data[,1],prediction\_RRF\_test)

print(RRF\_train\_mape)

print(RRF\_test\_mape)

print(RRF\_train\_r2)

print(RRF\_test\_r2)

print(RRF\_train\_rmse)

print(RRF\_test\_rmse)

#GRID SEARCH CV RANDOM FOREST

#lets set require parameters using caret library

control = trainControl(method="repeatedcv", number=5, repeats=4,search='grid')

maxdepth = c(6:30)

tunegrid = expand.grid(maxdepth=maxdepth)

#lets build Random forest model using the above parameters

GRF\_model = caret::train(fare\_amount~.,data=train\_data,method ='repeatedcv',trcontrol=control,tunegrid=tunegrid)

print(GRF\_model)

best\_parameter = GRF\_model$bestTune

print(best\_parameter)

#mtry = 13

#lets again build the same model using bestparameter

GRF\_bestmodel = randomForest(count~.,data = train\_data,mtry =13,importance = TRUE,method='rf')

print(GRF\_bestmodel)

#lets predict on train and test data,

predictions\_GRF\_train = predict(GRF\_bestmodel,train\_data)

predictions\_GRF\_test = predict(GRF\_bestmodel,test\_data)

#MAPE calculation

GRF\_train\_mape = MAPE(predictions\_GRF\_train,train\_data[,1])

GRF\_test\_mape = MAPE(predictions\_GRF\_test,test\_data[,1])

#Rsquare calculation

GRF\_train\_r2 = rsquare(predictions\_GRF\_train,train\_data[,1])

GRF\_test\_r2 = rsquare(predictions\_GRF\_test,test\_data[,1])

#RMSE calculation

GRF\_train\_rmse = RMSE(predictions\_GRF\_train,train\_data[,1])

GRF\_test\_rmse = RMSE(predictions\_GRF\_test,test\_data[,1])

print(GRF\_train\_mape)

print(GRF\_test\_mape)

print(GRF\_train\_r2)

print(GRF\_test\_r2)

print(GRF\_train\_rmse)

print(GRF\_test\_rmse)

#MODEL SELECTION

Model\_name = c('Linear regression',

'Decision tree','Random search CV decision tree','Grid search CV decision tree',

'Random forest','Random search CV random forest','Grid search CV random forest')

MAPE\_train = c(LR\_train\_mape,DT\_train\_mape,RDT\_train\_mape,GDT\_train\_mape,

RF\_train\_mape,GRF\_train\_mape,GRF\_train\_mape)

MAPE\_test = c(LR\_test\_mape,DT\_test\_mape,RDT\_test\_mape,GDT\_test\_mape,

RF\_test\_mape,GRF\_test\_mape,GRF\_test\_mape)

Rsquare\_train = c(LR\_train\_r2,DT\_train\_r2,RDT\_train\_r2,GDT\_train\_r2,

RF\_train\_r2,GRF\_train\_r2,GRF\_train\_r2)

Rsquare\_test = c(LR\_test\_r2,DT\_test\_r2,RDT\_test\_r2,GDT\_test\_r2,

RF\_test\_r2,GRF\_test\_r2,GRF\_test\_r2)

RMSE\_train = c(LR\_train\_rmse,DT\_train\_rmse,RDT\_train\_rmse,GDT\_train\_rmse,

RF\_train\_rmse,GRF\_train\_rmse,GRF\_train\_rmse)

RMSE\_test = c(LR\_test\_rmse,DT\_test\_rmse,RDT\_test\_rmse,GDT\_test\_rmse,

RF\_test\_rmse,RRF\_test\_rmse,GRF\_test\_rmse)

FINAL\_RESULTS = data.frame(Model\_name,MAPE\_train,MAPE\_test,Rsquare\_train,Rsquare\_test,

RMSE\_train,RMSE\_test)

print(FINAL\_RESULTS)

# Model evaluation -------------------------------------------------------

# Lets do all we have done for Train\_Cab data

Test\_Cab = read.csv("test.csv")

#Check summary ofs dataset

summary(Test\_Cab)

# we are going to change pickup\_datetime from factor to datetime

# But first, let's replace UTC in pickup\_datetime variable with ''(space)

Test\_Cab$pickup\_datetime = gsub('// UTC','',Test\_Cab$pickup\_datetime)

# Now convert variable pickup\_dattime to date time format by creating

# new variable with name Date

Test\_Cab$date = as.Date(Test\_Cab$pickup\_datetime)

# Lets split this new variable Date into year,month,weekday

# Extract the year

Test\_Cab$year = substr(as.character(Test\_Cab$date),1,4)

# Extract the month

Test\_Cab$month =substr(as.character(Test\_Cab$date),6,7)

# Extract the weekday

Test\_Cab$day = weekdays(as.POSIXct(Test\_Cab$date),abbreviate = F)

# Extract the date

Test\_Cab$date = substr(as.character(Test\_Cab$date),9,10)

# Extract the time

Test\_Cab$hour = substr(as.factor(Test\_Cab$pickup\_datetime),12,13)

# Let us delete pickup\_dataetime as we extracted required substitutes of date

Test\_Cab$pickup\_datetime = NULL

# Data after datatype conversion

Data1 = Test\_Cab # will keep a copy of original data

head(Test\_Cab)

# Missing Value Analysis --------------------------------------------------

sum(is.na(Test\_Cab)) # no missing values

# Outlier analysis --------------------------------------------------------

summary(Test\_Cab) # no outliers

# Lets calculate distance for test data using function which we already created for train data

# # #Applying distance formula for train data

get\_geo\_distance = function(long1, lat1, long2, lat2) {

loadNamespace("purrr")

loadNamespace("geosphere")

longlat1 = purrr::map2(long1, lat1, function(x,y) c(x,y))

longlat2 = purrr::map2(long2, lat2, function(x,y) c(x,y))

distance\_list = purrr::map2(longlat1, longlat2, function(x,y) geosphere::distHaversine(x, y))

distance\_m = list.extract(distance\_list, position = 1)

#if (units == "km") {

distance = distance\_m / 1000.0;

distance

}

for(i in (1:nrow(Test\_Cab)))

{

Test\_Cab$distance[i]= get\_geo\_distance(Test\_Cab$pickup\_longitude[i],Test\_Cab$pickup\_latitude[i],Test\_Cab$dropoff\_longitude[i],Test\_Cab$dropoff\_latitude[i])

}

head( Test\_Cab)

#write.csv(Test\_Cab,"distance\_test.csv")

# Lets check whether distance variables has any outlie

summary(Test\_Cab$distance)

# distance can not be less than 1 so we replace with average distance

Test\_Cab$distance[Test\_Cab$distance < 1] = mean(Test\_Cab$distance)

# During model development, we deleted few varaibles based on anova test and correlation analysis

# The variables in the test case should exactly match with the variables in the trained model

Test\_Cab$date = NULL

Test\_Cab$day = NULL

# Feature scaling for test data -------------------------------------------

# Checking distance variable distribution using histogram

ggplot(Test\_Cab, aes\_string(x = Test\_Cab$distance)) +

geom\_histogram(fill="skyblue", colour = "black") + geom\_density() +

theme\_bw() + xlab("distance") + ylab("Frequency")+ggtitle(" distribution of distance ")

# We are going to define function using log

signedlog10 = function(x) {

ifelse(abs(x) <= 1, 0, sign(x)\*log10(abs(x)))

}

# Applying log function to distance variable

Test\_Cab$distance = signedlog10(Test\_Cab$distance)

# Checking distance distribution after applying function

ggplot(Test\_Cab, aes\_string(x = Test\_Cab$distance)) +

geom\_histogram(fill="skyblue", colour = "black") + geom\_density() +

theme\_bw() + xlab("distance") + ylab("Frequency")+ggtitle(" distribution of distance ")

# Let's look at summary again

summary(Test\_Cab$distance)

# Model evaluation using this test data -----------------------------------

# Code for development of model

#RF\_model = randomForest(fare\_amount~.,data = train\_data,n.trees = 500)

#print(RF\_model)

# Predicting model on Test\_Cab data

RFTest\_Cab = predict(RF\_model, Test\_Cab)

# Adding our obtained predictions as Predicted Fare Amount variable to test\_cab dataset

Test\_Cab$Predicted\_fare\_amount = RFTest\_Cab

# lets have a look of our predicted fare amount data

head(Test\_Cab)

summary(Test\_Cab$Predicted\_fare\_amount)

summary(Train\_Cab$fare\_amount)

# Finally, we designed a model, which predicts the cab fare.

**5. PYTHON CODE**

# Cab fare prediction

*#load required libraries*

**import** pandas **as** pd

**import** os

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**from** collections **import** Counter

**from** datetime **import** datetime

**from** math **import** sin, cos, sqrt, atan2, radians

**from** scipy.stats **import** chi2\_contingency

**from** random **import** randrange, uniform

*#import the dataset*

os.chdir("D:/Data Science/Cab Fare")

​

*#lets check the working directory*

os.getcwd()

Cab\_train **=** pd.read\_csv("train\_cab.csv")

# Explore the data

#*lets Check class of the data*

type(Cab\_train)

*# lets Check the dimensions*

Cab\_train.shape

*# lets Check names of dataset*

Cab\_train.columns

*# lets Check first rows of dataset*

Cab\_train.head()

*# lets Check last rows of dataset*

Cab\_train.tail()

*#lets Check structure of dataset*

Cab\_train.dtypes

*#lets Check summary of dataset*

Cab\_train.describe()

*# Variable Identification*

Cab\_train['fare\_amount'].dtypes

*# lets convert our fare\_amount variable from object to numeric data type*

​

Cab\_train['fare\_amount'] **=** pd.to\_numeric(Cab\_train['fare\_amount'], errors **=** "coerce")

Cab\_train['fare\_amount'].dtypes

*# when we tried convert pickup\_datetime variable to date format it was throwing error coz of a starnge value in the variable*

*# So first treat it as NA and drop*

​

Cab\_train.loc[Cab\_train['pickup\_datetime'] **==** '43' ,'pickup\_datetime'] **=** np.nan

​

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['pickup\_datetime'].isnull()].index, axis **=** 0)

​

​

*# Now lets convert pickup\_datetime*

​

Cab\_train['pickup\_datetime'] **=** pd.to\_datetime(Cab\_train['pickup\_datetime'], format**=**'%Y-%m-%d %H:%M:%S UTC')

*# lets see the datatypes after conversion*

Cab\_train.dtypes

# Missing Value Analysis

*#lets see missing values in our dataset*

missing\_val **=** Cab\_train.isnull().sum()

missing\_val

*# Lets drop the observations with missing values because our missing values are not exceed 35%*

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['passenger\_count'].isnull()].index,axis **=** 0)

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['fare\_amount'].isnull()].index,axis **=** 0)

*# Lets check dimensions after missing value analysis*

​

Cab\_train.shape

*# Lets split the pickup\_date time variable into year, month, date, day, hour, minute.*

​

Cab\_train['year'] **=**Cab\_train['pickup\_datetime'].dt.year

​

Cab\_train['month'] **=** Cab\_train['pickup\_datetime'].dt.month

​

Cab\_train['date'] **=** Cab\_train['pickup\_datetime'].dt.day

​

Cab\_train['day'] **=** Cab\_train['pickup\_datetime'].dt.dayofweek

​

Cab\_train['hour'] **=** Cab\_train['pickup\_datetime'].dt.hour

​

Cab\_train['minute'] **=** Cab\_train['pickup\_datetime'].dt.minute

​

*#lets drop the pickup datetime variable because we splitted into many variables*

Cab\_train **=** Cab\_train.drop('pickup\_datetime',axis**=**1)

*#lets check dimemsion after conversion of pickup datetime*

Cab\_train.shape

*# Lets check summary after conversion*

Cab\_train.describe()

# Outlier analysis

*#lets remove outlier one by one*

*#passenger count*

*#here in cab maximum passenger count will be 7. above 7 will be considered as outliers*

print(Counter(Cab\_train['passenger\_count']**<**1))

print(Counter(Cab\_train['passenger\_count']**>**7))

*#here passenger count has totally 77 outliers where 58 variables are 0 and 19 variables are above 7.*

*#there is no use of having these data.hence we drop it*

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['passenger\_count'] **>** 7].index, axis**=**0)

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['passenger\_count'] **<** 1].index, axis**=**0)

​

*# verify passenger count*

print(Counter(Cab\_train['passenger\_count'] **<** 1))

print(Counter(Cab\_train['passenger\_count'] **>** 7))

*#here in latitude ranges from -90 to +90 and longitude ranges from -180 to +180.Beyond these variables considered as an outlier*

*#by seeing the summary of the dataset, we can see if there is any outliers present in our dataset*

Cab\_train.describe()

*#from the above summary, we can see that outlier present in only pickup latitude.lets dop that variable*

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['pickup\_latitude'] **>** 90].index, axis**=**0)

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['pickup\_latitude'] **<** **-**90].index, axis**=**0)

​

*#let verify the dataset*

Cab\_train.describe()

*#fareamount*

*# Let use describe function to check min max values and identify outliers*

Cab\_train['fare\_amount'].describe()

​

*# We can observe the max value as 54343 and min as -3. Practically its not possible.It clearly defines outliers.*

*#from the above inferences,we can say that the fare amount for a cab cant be in negative as well as it cannot exceed 500*

*#above and below the values will be considered as outliers and drop it.*

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['fare\_amount'] **>** 500].index, axis**=**0)

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['fare\_amount'] **<** 1].index, axis**=**0)

​

*#lets verify*

Cab\_train['fare\_amount'].describe()

*# Save numeric names*

cnames **=**['fare\_amount', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count']

*# Lets visualize boxplots again after outlier removal*

​

​

**for** i **in** cnames:

print(i)

plt.boxplot(Cab\_train[i])

plt.xlabel(i)

plt.ylabel('fare\_amount')

plt.title('outlier analysis')

plt.show()

*# Lets check shape dataset after outlier removal*

Cab\_train.shape

Cab\_train.head()

*# Now, let's create distance using Haversine Formula*

​

*# Calculates the geodesic distance between two points specified by*

​

*# radian latitude/longitude using the Haversine formula*

**from** math **import** radians, cos, sin, asin, sqrt

​

**def** distance(pickup\_lat, pickup\_lon, dropoff\_lat, dropoff\_lon):

*#Define earth radius (km)*

R\_earth **=** 6371

*#Convert degrees to radians*

pickup\_lat, pickup\_lon, dropoff\_lat, dropoff\_lon **=** map(np.radians,

[pickup\_lat, pickup\_lon,

dropoff\_lat, dropoff\_lon])

*#Compute distances along lat, lon dimensions*

dlat **=** dropoff\_lat **-** pickup\_lat

dlon **=** dropoff\_lon **-** pickup\_lon

*#Compute haversine distance*

a **=** np.sin(dlat**/**2.0)**\*\***2 **+** np.cos(pickup\_lat) **\*** np.cos(dropoff\_lat) **\*** np.sin(dlon**/**2.0)**\*\***2

**return** 2 **\*** R\_earth **\*** np.arcsin(np.sqrt(a))

Cab\_train['distance'] **=** distance(Cab\_train['pickup\_latitude'],

Cab\_train['pickup\_longitude'],

Cab\_train['dropoff\_latitude'] ,

Cab\_train['dropoff\_longitude'])

Cab\_train.head()

Cab\_train.shape

*# Lets check is there any outliers in this distance variable using describe function*

*# Lets plot boxplot for distance variable*

​

plt.boxplot(Cab\_train['distance'])

​

plt.xlabel('distance')

​

plt.title('outlier analysis')

​

plt.show()

Cab\_train['distance'].describe()

*#from above inferences, we can see that distance can't be above 130 in our dataset and less than 1 is also not practically possible.*

*#lets drop those variables.*

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['distance'] **>** 130].index, axis**=**0)

Cab\_train **=** Cab\_train.drop(Cab\_train[Cab\_train['distance'] **<** 1].index, axis**=**0)

​

*#lets verify the dataset*

Cab\_train['distance'].describe()

*# Lets check the boxplot after removing outliers*

​

plt.boxplot(Cab\_train['distance'])

​

plt.xlabel('distance')

​

plt.title('outlier analysis')

​

plt.show()

*#lets save our preprocessed data*

df **=** Cab\_train

Cab\_train **=** df

# Data visualization

Cab\_train.columns

*# Univariate Analysis*

​

​

*# fare\_amount*

​

plt.figure(figsize**=**(8,8))

sns.countplot(x**=**'fare\_amount',data **=** Cab\_train,color**=**'blue')

*# pickup\_longitude*

​

plt.figure(figsize**=**(8,8))

sns.countplot(x**=**'pickup\_longitude',data **=** Cab\_train,color**=**'blue')

​

*# pickup\_latitude*

plt.figure(figsize**=**(8,8))

sns.countplot(x**=**'pickup\_latitude',data **=** Cab\_train,color**=**'blue')

​

*# dropoff\_longitude*

plt.figure(figsize**=**(8,8))

sns.countplot(x**=**'dropoff\_longitude',data **=** Cab\_train,color**=**'blue')

​

*#dropoff\_latitude*

plt.figure(figsize**=**(8,8))

sns.countplot(x**=**'dropoff\_latitude',data **=** Cab\_train,color**=**'blue')

​

*# passenger\_count*

​

plt.figure(figsize**=**(8,8))

sns.countplot(x**=**'passenger\_count',data **=** Cab\_train,color**=**'blue')

​

*#based on the passengers, we can see that single passengers travelled higher*

*# distance*

*#distance*

*# plot Distance*

plt.figure(figsize**=**(15,7))

plt.hist(Cab\_train['distance'], bins**=**10)

plt.xlabel('Distance')

plt.ylabel('Frequency')

*#Maximum number of passengers are travelled less than 20km*

*# Bivariate Analysis*

​

*# Visualization between fare\_amount and years.*

sns.scatterplot(x**=**'year',y**=**'fare\_amount',data **=** Cab\_train)

*# Based on the scatterplot, in year 2012 and 2013 there were rides which got high fare\_amount and very low on year 2015*

*# Visualization between fare\_amount and months.*

sns.scatterplot(x**=**'month',y**=**'fare\_amount',data **=** Cab\_train)

*# Based on the scatterplot,We can see March month fare amount is very high and low in July month.*

*# Visualization between fare\_amount and weekday.*

sns.scatterplot(x**=**'day',y**=**'fare\_amount',data **=** Cab\_train)

​

*# Based on the scatterplot,We can see that Friday ride has the highest fare\_amount*

*# Visualization between fare\_amount and time.*

​

sns.scatterplot(x**=**'hour',y**=**'fare\_amount',data **=** Cab\_train)

*# Rides taken during 6 pm to 8 pm gives highest fare\_amount*

*# Visualization between fare\_amount and passenger\_count*

sns.scatterplot(x**=**"passenger\_count",y**=**"fare\_amount",data**=**Cab\_train)

*# Visualization between fare\_amount and distance*

sns.scatterplot(x**=**"distance",y**=**"fare\_amount",data**=**Cab\_train)

*# we can see as the distance increases fare amount also increases*

# Feature selection

print(Cab\_train.columns)

*#lets drop minute variable*

Cab\_train **=** Cab\_train.drop('minute',axis **=** 1)

Cab\_train.shape

Cab\_train.head()

Cab\_train.dtypes

*# Lets convert passenger\_count into int64 data type*

Cab\_train['passenger\_count'] **=** Cab\_train['passenger\_count'].astype('int64')

​

​

*# Similarly convert year,month,date,day,hour as these comes under categorical variables*

Cab\_train['year'].unique()

​

Cab\_train['month'].unique()

​

Cab\_train['date'].unique()

​

Cab\_train['day'].unique()

​

Cab\_train['hour'].unique()

​

Cab\_train['year'] **=** Cab\_train['year'].astype('object')

​

Cab\_train['month'] **=** Cab\_train['month'].astype('object')

​

Cab\_train['date'] **=** Cab\_train['date'].astype('object')

​

Cab\_train['day'] **=** Cab\_train['day'].astype('object')

​

Cab\_train['hour'] **=** Cab\_train['hour'].astype('object')

​

Cab\_train.dtypes

Cab\_train.columns

*# So lets define numeric and categorical variables to find significant variables to build a model*

​

*# Storing continuous variables into an object called cnames*

​

cnames **=** ['fare\_amount', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count','distance']

​

*# Storing categorical variables into catnames object*

​

catnames **=** ['year','month', 'date', 'day', 'hour']

*##Correlation analysis*

*#Correlation plot*

​

df\_corr **=** Cab\_train.loc[:,cnames]

​

*#Set the width and hieght of the plot*

f, ax **=** plt.subplots(figsize**=**(7, 5))

​

*#Generate correlation matrix*

corr **=** df\_corr.corr()

​

print(corr)*# From correlation analysis we can say distance is significant variable to pass in to the model*

​

*#Plot using seaborn library*

sns.heatmap(corr, mask**=**np.zeros\_like(corr, dtype**=**np.bool), cmap**=**sns.diverging\_palette(220, 10, as\_cmap**=True**),

square**=True**, ax**=**ax,annot**=True**)

*# Anova Test is performed between catnames (categorical independent variables) & fare\_amount(continuous target variable)*

​

**import** statsmodels.api **as** sm

​

**from** statsmodels.formula.api **import** ols

​

**for** i **in** catnames:

mod **=** ols('fare\_amount' **+** '~' **+** i, data **=** Cab\_train).fit()

aov\_table **=** sm.stats.anova\_lm(mod, typ **=** 2)

print(aov\_table)

*# From the anova result, we can observe Date ,weekday*

​

*# has p value > 0.05, so delete these variables not consider in model.*

*# After correlation and ANOVA test we need to remove these variables*

*# Cat variables - weekday and date(which has p-value>0.05)*

​

Cab\_train **=** Cab\_train.drop('date', axis **=** 1)

Cab\_train **=** Cab\_train.drop('day', axis **=** 1)

​

*# Lets cross check shape of dataset after feature selection*

Cab\_train.shape

Cab\_train.head()

*# Storing final categorical variables into cat\_var object for model development*

Catnmaes **=** ['year', 'month', 'hour']

# Feature Scaling

*# Lets check the distribution of our data*

*# passenger\_count*

sns.distplot(Cab\_train['passenger\_count'],bins**=**'auto',color**=**'blue')

plt.title("Distribution of passenger\_count")

plt.ylabel("Density")

plt.show()

*# distance variable*

sns.distplot(Cab\_train['distance'],bins**=**'auto',color**=**'blue')

plt.title("Distribution of distance")

plt.ylabel("Density")

plt.show()

*# The distance variable is right skewed we can reduce this skewness using log transformation*

Cab\_train['distance'] **=** np.log(Cab\_train['distance'])

*# Cab\_train['fare\_amount'] = np.log(Cab\_train['fare\_amount'])*

Cab\_train.head()

*# We shall check for normality again after log transformation of distance variable*

​

sns.distplot(Cab\_train['distance'],bins**=**'auto',color**=**'blue')

plt.title("Distribution of distance after log transformation")

plt.ylabel("Density")

plt.show()

​

*# We can observe now, after applying log function, the data is normalised*

# Model development

*# Load Required libraries for model development*

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.metrics **import** mean\_squared\_error

**from** sklearn.metrics **import** r2\_score

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.tree **import** DecisionTreeRegressor

**from** sklearn.ensemble **import** RandomForestRegressor

**from** sklearn **import** metrics

*# Lets Divide the data into train and test set*

​

X**=** Cab\_train.drop(['fare\_amount'],axis**=**1)

y**=** Cab\_train['fare\_amount']

*# Now Split the data into train and test using train\_test\_split function*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split( X, y, test\_size**=**0.2, random\_state**=**101)

*# Function for Error metrics to calculate the performance of model*

**def** MAPE(y\_true,y\_prediction):

mape**=** np.mean(np.abs(y\_true**-**y\_prediction)**/**y\_true)**\***100

**return** mape

*# Linear Regression model*

np.asarray(Cab\_train)

*#lets build Linear Regression model on train data*

LinearRegression\_model **=** LinearRegression().fit(X\_train, y\_train)

*# Model prediction on train data*

LinearRegression\_train**=** LinearRegression\_model.predict(X\_train)

​

*# Model prediction on test data*

LinearRegression\_test**=** LinearRegression\_model.predict(X\_test)

​

*# MAPE for train data*

MAPE\_train**=** MAPE(y\_train,LinearRegression\_train)

​

*# MAPE for on test data*

MAPE\_test**=** MAPE(y\_test,LinearRegression\_test)

​

*# r2 value for train data*

r2\_train**=** r2\_score(y\_train,LinearRegression\_train)

​

*# r2 value for test data-*

r2\_test**=**r2\_score(y\_test,LinearRegression\_test)

​

*# RMSE value for train data*

RMSE\_train **=** np.sqrt(metrics.mean\_squared\_error(y\_train,LinearRegression\_train))

​

*# RMSE value for test data*

RMSE\_test **=** np.sqrt(metrics.mean\_squared\_error(y\_test,LinearRegression\_test))

​

print("Mean Absolute Precentage Error for train data="**+**str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="**+**str(MAPE\_test))

print("R^2\_score for train data="**+**str(r2\_train))

print("R^2\_score for test data="**+**str(r2\_test))

print("RMSE for train data="**+**str (RMSE\_train))

print("RMSE for test data="**+**str(RMSE\_test))

Error\_MetricsLT **=** {'Model Name': ['Linear Regression'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

​

LinearRegression\_Results **=** pd.DataFrame(Error\_MetricsLT)

LinearRegression\_Results

*#Decision tree*

*# Decision tree for regression*

DecisionTree\_model**=** DecisionTreeRegressor(max\_depth**=**2).fit(X\_train,y\_train)

​

*# Model prediction on train data*

DecisionTree\_train**=** DecisionTree\_model.predict(X\_train)

​

*# Model prediction on test data*

DecisionTree\_test**=** DecisionTree\_model.predict(X\_test)

​

*# Model performance on train data*

MAPE\_train**=** MAPE(y\_train,DecisionTree\_train)

​

*# Model performance on test data*

MAPE\_test**=** MAPE(y\_test,DecisionTree\_test)

​

*# r2 value for train data*

r2\_train**=** r2\_score(y\_train,DecisionTree\_train)

​

*# r2 value for test data*

r2\_test**=**r2\_score(y\_test,DecisionTree\_test)

​

*# RMSE value for train data*

RMSE\_train **=** np.sqrt(metrics.mean\_squared\_error(y\_train,DecisionTree\_train))

​

*# RMSE value for test data*

RMSE\_test **=** np.sqrt(metrics.mean\_squared\_error(y\_test,DecisionTree\_test))

​

print("Mean Absolute Precentage Error for train data="**+**str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="**+**str(MAPE\_test))

print("R^2\_score for train data="**+**str(r2\_train))

print("R^2\_score for test data="**+**str(r2\_test))

print("RMSE for train data="**+**str(RMSE\_train))

print("RMSE for test data="**+**str(RMSE\_test))

Error\_MetricsDT **=** {'Model Name': ['Decision Tree'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

DecisionTree\_Results **=** pd.DataFrame(Error\_MetricsDT)

DecisionTree\_Results

*#Random search cv in decision tree*

*# Import libraries*

**from** sklearn.model\_selection **import** RandomizedSearchCV

​

RandomDecisionTree **=** DecisionTreeRegressor(random\_state **=** 0)

depth **=** list(range(1,20,2))

random\_search **=** {'max\_depth': depth}

​

*# Lets build a model using above parameters on train data*

RandomDecisionTree\_model**=** RandomizedSearchCV(RandomDecisionTree,param\_distributions**=** random\_search,n\_iter**=**5,cv**=**5)

RandomDecisionTree\_model**=** RandomDecisionTree\_model.fit(X\_train,y\_train)

​

*# Lets look into best fit parameters*

best\_parameters **=** RandomDecisionTree\_model.best\_params\_

print(best\_parameters)

*# Again rebuild decision tree model using randomsearch best fit parameter ie*

*# with maximum depth = 1*

RDT\_best\_model **=** RandomDecisionTree\_model.best\_estimator\_

print(RDT\_best\_model)

*# Prediction on train data*

RDT\_train **=** RDT\_best\_model.predict(X\_train)

​

*# Prediction on test data*

RDT\_test **=** RDT\_best\_model.predict(X\_test)

​

*# Lets check Model performance on both test and train using error metrics of regression like mape,rsquare value*

*# MAPE for train data*

MAPE\_train**=** MAPE(y\_train,RDT\_train)

​

*# MAPE for test data*

MAPE\_test**=** MAPE(y\_test,RDT\_test)

​

*# Rsquare for train data*

r2\_train**=** r2\_score(y\_train,RDT\_train)

​

*# Rsquare for test data*

r2\_test**=**r2\_score(y\_test,RDT\_test)

​

*# RMSE value for train data*

RMSE\_train **=** np.sqrt(metrics.mean\_squared\_error(y\_train,RDT\_train))

​

*# RMSE value for test data*

RMSE\_test **=** np.sqrt(metrics.mean\_squared\_error(y\_test,RDT\_test))

​

​

*# Lets print the results*

print("Best Parameter="**+**str(best\_parameters))

print("Best Model="**+**str(RDT\_best\_model))

print("Mean Absolute Precentage Error for train data="**+**str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="**+**str(MAPE\_test))

print("R^2\_score for train data="**+**str(r2\_train))

print("R^2\_score for test data="**+**str(r2\_test))

print("RMSE for train data="**+**str (RMSE\_train))

print("RMSE for test data="**+**str(RMSE\_test))

Error\_MetricsRDT **=** {'Model Name': ['Random Search CV Decision Tree'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

RandomDecisionTree\_Results **=** pd.DataFrame(Error\_MetricsRDT)

RandomDecisionTree\_Results

*# Grid Search CV in Decision Tree*

*# Import libraries*

**from** sklearn.model\_selection **import** GridSearchCV

​

GridDecisionTree**=** DecisionTreeRegressor(random\_state**=**0)

depth**=** list(range(1,20,2))

grid\_search**=** {'max\_depth':depth}

​

*# Lets build a model using above parameters on train data*

GridDecisionTree\_model**=** GridSearchCV(GridDecisionTree,param\_grid**=**grid\_search,cv**=**5)

GridDecisionTree\_model**=** GridDecisionTree\_model.fit(X\_train,y\_train)

*# Lets look into best fit parameters from gridsearch cv DT*

best\_parameters **=** GridDecisionTree\_model.best\_params\_

print(best\_parameters)

*# Again rebuild decision tree model using gridsearch best fit parameter ie*

*# with maximum depth = 1*

GDT\_best\_model **=** GridDecisionTree\_model.best\_estimator\_

*# Prediction on train data*

GDT\_train **=** GDT\_best\_model.predict(X\_train)

​

*# Prediction on train data test data-*

GDT\_test **=** GDT\_best\_model.predict(X\_test)

​

*# Lets check Model performance on both test and train using error metrics of regression like mape,rsquare value*

*# MAPE for train data*

MAPE\_train**=** MAPE(y\_train,GDT\_train)

​

*# MAPE for test data*

MAPE\_test**=** MAPE(y\_test,GDT\_test)

​

*# Rsquare for train data*

r2\_train**=** r2\_score(y\_train,GDT\_train)

​

*# Rsquare for train data*

r2\_test**=**r2\_score(y\_test,GDT\_test)

​

*# RMSE value for train data*

RMSE\_train **=** np.sqrt(metrics.mean\_squared\_error(y\_train,GDT\_train))

​

*# RMSE value for test data*

RMSE\_test **=** np.sqrt(metrics.mean\_squared\_error(y\_test,GDT\_test))

​

​

print("Best Parameter="**+**str(best\_parameters))

print("Best Model="**+**str(GDT\_best\_model))

print("Mean Absolute Precentage Error for train data="**+**str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="**+**str(MAPE\_test))

print("R^2\_score for train data="**+**str(r2\_train))

print("R^2\_score for test data="**+**str(r2\_test))

print("RMSE for train data="**+**str (RMSE\_train))

print("RMSE for test data="**+**str(RMSE\_test))

​

Error\_MetricsGDT **=** {'Model Name': ['Grid Search CV Decision Tree'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

GridDecisionTree\_Results **=** pd.DataFrame(Error\_MetricsGDT)

GridDecisionTree\_Results

*#Random forest*

*# Random Forest for regression*

RF\_model**=** RandomForestRegressor(n\_estimators**=**100).fit(X\_train,y\_train)

​

*# Prediction on train data*

RF\_train**=** RF\_model.predict(X\_train)

​

*# Prediction on test data*

RF\_test**=** RF\_model.predict(X\_test)

​

*# MAPE For train data*

MAPE\_train**=** MAPE(y\_train,RF\_train)

​

*# MAPE For test data*

MAPE\_test**=** MAPE(y\_test,RF\_test)

​

*# Rsquare For train data*

r2\_train**=** r2\_score(y\_train,RF\_train)

​

*# Rsquare For test data*

r2\_test**=**r2\_score(y\_test,RF\_test)

​

*# RMSE value for train data*

RMSE\_train **=** np.sqrt(metrics.mean\_squared\_error(y\_train,RF\_train))

​

*# RMSE value for test data*

RMSE\_test **=** np.sqrt(metrics.mean\_squared\_error(y\_test,RF\_test))

​

print("Mean Absolute Precentage Error for train data="**+**str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="**+**str(MAPE\_test))

print("R^2\_score for train data="**+**str(r2\_train))

print("R^2\_score for test data="**+**str(r2\_test))

print("RMSE for train data="**+**str (RMSE\_train))

print("RMSE for test data="**+**str(RMSE\_test))

​

Error\_MetricsRF **=** {'Model Name': ['Random Forest'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

RandomForest\_Results **=** pd.DataFrame(Error\_MetricsRF)

RandomForest\_Results

*# Random Search CV in Random Forest*

*# Import libraries*

**from** sklearn.model\_selection **import** RandomizedSearchCV

​

RandomRandomForest **=** RandomForestRegressor(random\_state **=** 0)

n\_estimator **=** list(range(1,100,2))

depth **=** list(range(1,20,2))

random\_search **=** {'n\_estimators':n\_estimator, 'max\_depth': depth}

​

*# Lets build a model using above parameters on train data*

RandomRandomForest\_model**=** RandomizedSearchCV(RandomRandomForest,param\_distributions**=** random\_search,n\_iter**=**5,cv**=**5)

RandomRandomForest\_model**=** RandomRandomForest\_model.fit(X\_train,y\_train)

*# Best parameters for model*

best\_parameters **=** RandomRandomForest\_model.best\_params\_

print(best\_parameters)

*# Again rebuild random forest model using gridsearch best fit parameter*

*#n\_estimators: 59, max\_depth: 3*

RRF\_best\_model **=** RandomRandomForest\_model.best\_estimator\_

*# Prediction on train data*

RRF\_train **=** RRF\_best\_model.predict(X\_train)

​

*# Prediction on test data*

RRF\_test **=** RRF\_best\_model.predict(X\_test)

​

*# Lets check Model performance on both test and train using error metrics of regression like mape,rsquare value*

*# MAPE for train data*

MAPE\_train**=** MAPE(y\_train,RRF\_train)

​

*# MAPE for test data*

MAPE\_test**=** MAPE(y\_test,RRF\_test)

​

*# Rsquare for train data*

r2\_train**=** r2\_score(y\_train,RRF\_train)

​

*# Rsquare for test data*

r2\_test**=**r2\_score(y\_test,RRF\_test)

​

*# RMSE value for train data*

RMSE\_train **=** np.sqrt(metrics.mean\_squared\_error(y\_train,RRF\_train))

​

*# RMSE value for test data*

RMSE\_test **=** np.sqrt(metrics.mean\_squared\_error(y\_test,RRF\_test))

​

​

print("Best Parameter="**+**str(best\_parameters))

print("Best Model="**+**str(RRF\_best\_model))

print("Mean Absolute Precentage Error for train data="**+**str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="**+**str(MAPE\_test))

print("R^2\_score for train data="**+**str(r2\_train))

print("R^2\_score for test data="**+**str(r2\_test))

print("RMSE for train data="**+**str (RMSE\_train))

print("RMSE for test data="**+**str(RMSE\_test))

Error\_MetricsRSRF **=** {'Model Name': ['Random Search CV Random Forest'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

RandomSearchRandomForest\_Results **=** pd.DataFrame(Error\_MetricsRSRF)

RandomSearchRandomForest\_Results

*# Grid search CV in Random Forest*

*# Import libraries*

**from** sklearn.model\_selection **import** GridSearchCV

​

GridRandomForest**=** RandomForestRegressor(random\_state**=**0)

n\_estimator **=** list(range(1,20,2))

depth**=** list(range(1,20,2))

grid\_search**=** {'n\_estimators':n\_estimator, 'max\_depth': depth}

*# Lets build a model using above parameters on train data using random forest grid search cv*

GridRandomForest\_model**=** GridSearchCV(GridRandomForest,param\_grid**=**grid\_search,cv**=**5)

GridRandomForest\_model**=** GridRandomForest\_model.fit(X\_train,y\_train)

*# Best fit parameters for model*

best\_parameters\_GRF **=** GridRandomForest\_model.best\_params\_

print(best\_parameters\_GRF)

*# Again rebuild random forest model using gridsearch best fit parameter*

*#n\_estimators: 19, max\_depth: 3*

GRF\_best\_model **=** GridRandomForest\_model.best\_estimator\_

*# Prediction on train data*

GRF\_train **=** GRF\_best\_model.predict(X\_train)

​

*# Prediction on test data*

GRF\_test **=** GRF\_best\_model.predict(X\_test)

​

*# Lets check Model performance on both test and train using error metrics of regression like mape,rsquare value*

*# MAPE for train data*

MAPE\_train**=** MAPE(y\_train,GRF\_train)

​

*# MAPE for test data*

MAPE\_test**=** MAPE(y\_test,GRF\_test)

​

*# Rsquare for train data*

r2\_train**=** r2\_score(y\_train,GRF\_train)

​

*# Rsquare for test data*

r2\_test**=**r2\_score(y\_test,GRF\_test)

​

*# RMSE value for train data*

RMSE\_train **=** np.sqrt(metrics.mean\_squared\_error(y\_train,GRF\_train))

​

*# RMSE value for test data*

RMSE\_test **=** np.sqrt(metrics.mean\_squared\_error(y\_test,GRF\_test))

​

print("Best Parameter="**+**str(best\_parameters))

print("Best Model="**+**str(GRF\_best\_model))

print("Mean Absolute Precentage Error for train data="**+**str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="**+**str(MAPE\_test))

print("R^2\_score for train data="**+**str(r2\_train))

print("R^2\_score for test data="**+**str(r2\_test))

print("RMSE for train data="**+**str (RMSE\_train))

print("RMSE for test data="**+**str(RMSE\_test))

Error\_MetricsGSRF **=** {'Model Name': ['Grid search CV Random Forest'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

GridSearchRandomForest\_Results **=** pd.DataFrame(Error\_MetricsGSRF)

GridSearchRandomForest\_Results

Final\_Results **=** pd.concat([LinearRegression\_Results,

DecisionTree\_Results,

RandomDecisionTree\_Results,

GridDecisionTree\_Results,

RandomForest\_Results,

RandomSearchRandomForest\_Results,

GridSearchRandomForest\_Results,], ignore\_index**=True**, sort **=False**)

Final\_Results

*# From above results Random Forest model have optimum values and this*

*# Random forest algorithm is good for our data*

# Model evaluation for test data

*#lets import test data and apply random forest model for prediction*

*# Import the test dataset*

Cab\_test **=** pd.read\_csv("test.csv")

*# Exploring the test data*

​

*# Check class of the data*

type(Cab\_test)

*#Check the dimensions(no of rows and no of columns)*

Cab\_test.shape

*#Check names of dataset(no need of renaming variables)*

Cab\_test.columns

​

*# we can see here there is no fare\_amount varible we need to predict using RF\_model*

*#Check top rows of dataset*

Cab\_test.head()

*#Check bottom rows of dataset*

Cab\_test.tail()

*#Check structure of dataset(data structure of each variable)*

Cab\_test.dtypes

*# we have to change pickup\_datetime from object to datetime*

​

Cab\_test['pickup\_datetime'] **=** pd.to\_datetime(Cab\_test['pickup\_datetime'], format**=**'%Y-%m-%d %H:%M:%S UTC')

​

print(Cab\_test.dtypes)

*# we can see out pickupdate time is converted to date format*

​

*# Let us see our data after data type conversion of variables*

​

print(Cab\_test.head())

*# Lets split the pickup\_date time variable into year, month, date, day, hour, minute.*

​

Cab\_test['year'] **=**Cab\_test['pickup\_datetime'].dt.year

​

Cab\_test['month'] **=** Cab\_test['pickup\_datetime'].dt.month

​

Cab\_test['date'] **=** Cab\_test['pickup\_datetime'].dt.day

​

Cab\_test['day'] **=** Cab\_test['pickup\_datetime'].dt.dayofweek

​

Cab\_test['hour'] **=** Cab\_test['pickup\_datetime'].dt.hour

*# Lets cross check our test data*

Cab\_test.head()

*# Missing Value Analysis for testdata*

*# Total number of missing values present in whole datset*

Cab\_test.isnull().sum()

​

*# no missing values*

*# Outlier analysis ------------------------------------*

*# lets check description of variables there we can figure out outliers easily*

Cab\_test.describe()

​

*#no outliers*

*# Now, let's create distance using Haversine Formula*

​

*# Calculates the geodesic distance between two points specified by*

​

*# radian latitude/longitude using the Haversine formula*

​

*# we already defined these function in Cab\_train dataset*

​

*# Lets apply the function on logitudes and lattitudes to derive distance*

​

​

Cab\_test['distance'] **=** distance(Cab\_test['pickup\_latitude'],

Cab\_test['pickup\_longitude'],

Cab\_test['dropoff\_latitude'] ,

Cab\_test['dropoff\_longitude'])

*# cross check few observation after creating new variable distance*

Cab\_test.head()

*# Lets check is there any outliers in this distance variable using describe function*

Cab\_test['distance'].describe() *# we can see 0 so delete such observations*

Counter(Cab\_test['distance'] **<** 1)

*#insted of dropping those variables,let substitute with mean*

Cab\_test.loc[Cab\_test.distance **<** 1,'distance'] **=** Cab\_test['distance'].mean()

print(Cab\_test['distance'].describe())

*# Lets delete pickup\_datetime as we derived year,month,date,day,hour,minute*

Cab\_test **=** Cab\_test.drop('pickup\_datetime', axis **=** 1)

Cab\_test.head()

*# During model development, we deleted few varaibles based on anova test and correlation analysis*

​

*# The variables in the test case should exactly match with the variables in the trained model*

​

Cab\_test.columns

​

​

Cab\_test **=** Cab\_test.drop('date', axis **=** 1)

Cab\_test **=** Cab\_test.drop('day', axis **=** 1)

print(Cab\_test.columns)

*# Normalizing distance variable*

​

*# We shall check for normality for distance. Let's first check about the skewness of the distance variable*

​

sns.distplot(Cab\_test['distance'],bins**=**'auto',color**=**'blue')

plt.title("Distribution of variable distance")

plt.ylabel("Density")

plt.show()

​

*# We can observe, the variable is right skewed.*

*# Using log function to reduce the skewness in distance*

​

Cab\_test['distance'] **=** np.log(Cab\_test['distance'])

*# We shall check for normality for distance after taking log function*

sns.distplot(Cab\_test['distance'],bins**=**'auto',color**=**'blue')

plt.title("Distribution of variable distance")

plt.ylabel("Density")

plt.show()

​

*# We can observe now, after applying log function, the data is normalised*

Cab\_test.head()

*# Now lets apply our RF model on this Test\_Cab data after cleaning this data*

​

Predictions\_Cab\_test **=** RF\_model.predict(Cab\_test)

Predictions\_Cab\_test

*# Fianlly creating new variable - Predicted Fare Amount*

​

Cab\_test['Predicted\_Fare\_Amount'] **=** Predictions\_Cab\_test

*# Lets see our data*

Cab\_test.head()

Cab\_test['Predicted\_Fare\_Amount'].describe()

Test\_Cab.head(6)

*#save output results*

Test\_Cab.to\_csv("Cabfarepredicted.csv", index **=** **False**)

​

**6. REFERENCES**

​

1) Edwisor Learning

2) www.analyticsvidhya.com

3) www.geeksforgeeks.org

4) towardsdatascience.com

5) rbloggers.com

6) Kaggle.com

7) github.com