Cab Fare Prediction Project

Name: Ganesh Ramachandran S

Date: April 2020

Contents

Chapter 1

Introduction

1.1 Project 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 requirement to apply have analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

1.2 Data

The data has been given as train and test for analysis. Both train and test have 6 attributes in common. The target variable is 'fare amount' it is continuous in nature. Since target variable is continuous in nature, we will go for regression-based models.

Attribute information: (dependent variables)

1. pickup_datetime - timestamp value indicating when the cab ride started.
2. pickup_longitude - float for longitude coordinate of where the cab ride started.
3. pickup_latitude - float for latitude coordinate of where the cab ride started.
4. dropoff_longitude - float for longitude coordinate of where the cab ride ended.
5. dropoff_latitude - float for latitude coordinate of where the cab ride ended.
6. passenger_count - an integer indicating the number of passengers in the cab ride.

1.3 Sample data

The train data have 16067 observations and 7 variables which include 6 dependent variables listed above and target variable 'fare amount'.

```
# First 5 Observation
cab_train.head(5)
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1.0
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.0
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2.0
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1.0
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1.0

The test data have 9914 observations and 6 dependent variables which are listed above.

```
cab_test.head(5)
```

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

In both test data and train data ‘pickup_datatime’ variables are object in class, remaining variables are numerical in class. Some of the observation has missing values

Chapter 2

Pre-processing

2.1 Exploratory data analysis

The train data has missing values and few variables has '0' (zero) values for numerical class with character vale level, as per given data attribute information 'fare_amount' variable has negative values and has a maximum value of 54343, these values are checked and processed.

The 'pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude', variables have some mismatch, looks like longitude and latitude values are misplaced in different pickup and drop-off related longitude and latitude variables. These values are checked related to degree values and processed.

The 'passenger_count' variable has '0' (zero) and maximum value as 5345. Some of the values are more than 6, which come around 20 observations are processed.

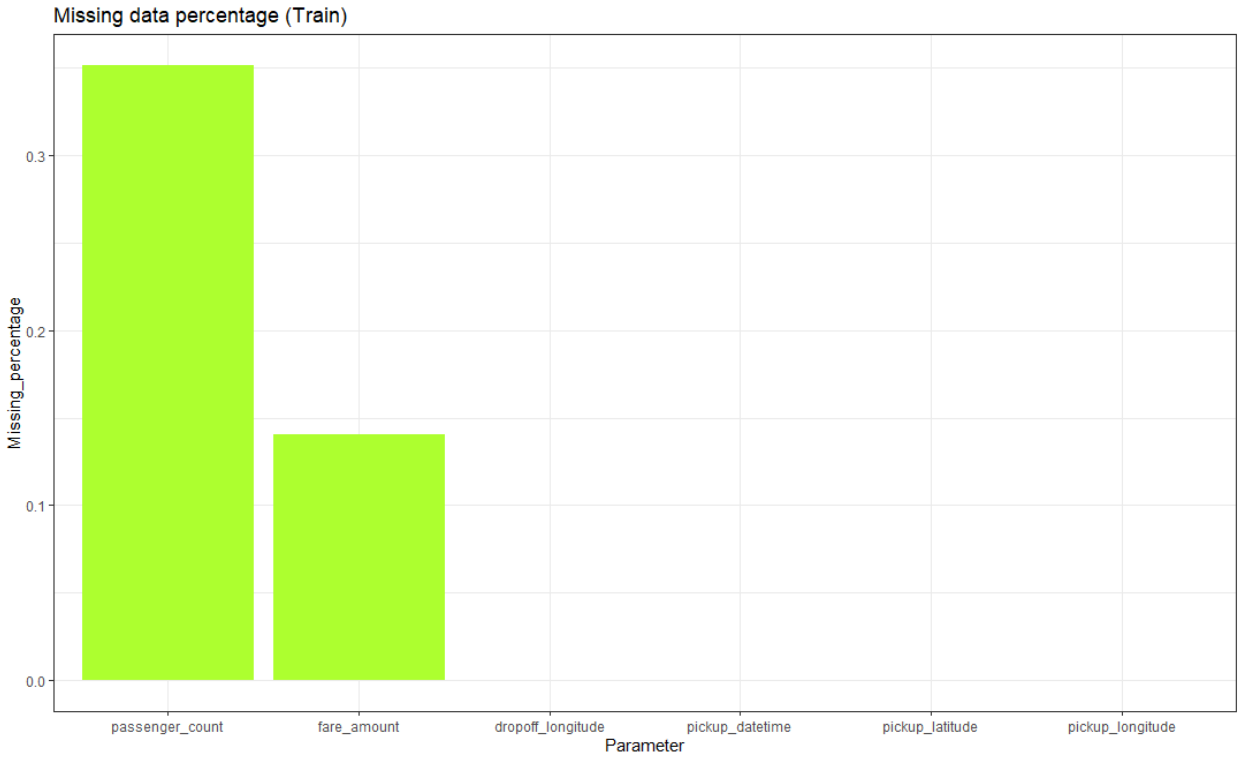
Data transformation has done based on the summary statistics information.

2.2 Missing value analysis

The training data has 7 variables and 16067 observations, some of the data are missing, it may be due to human error, refuse to answer while surveying and optional box in the questionnaire.

These missing values are understood by using graph, if each variable missing percentage is more than 30%, we drop variable or observation depends on the data. In this train data missing value percentage is less than 30%, we will try to impute by using mean, median and KNN method. Which imputation method gives nearest possible result in actual value would be chosen for the model. In this case KNN gives impute values are nearest to actual value compared to mean and median.

The train data has missing values percentage from 0% to 0.35%. it is less than 30% missing value; hence we will go for imputation methods to replace NA in observation. In this missing value imputation mean, median and KNN methods are used. From all three methods KNN imputation gives better imputed value. The K = 3 is used for missing value imputation.

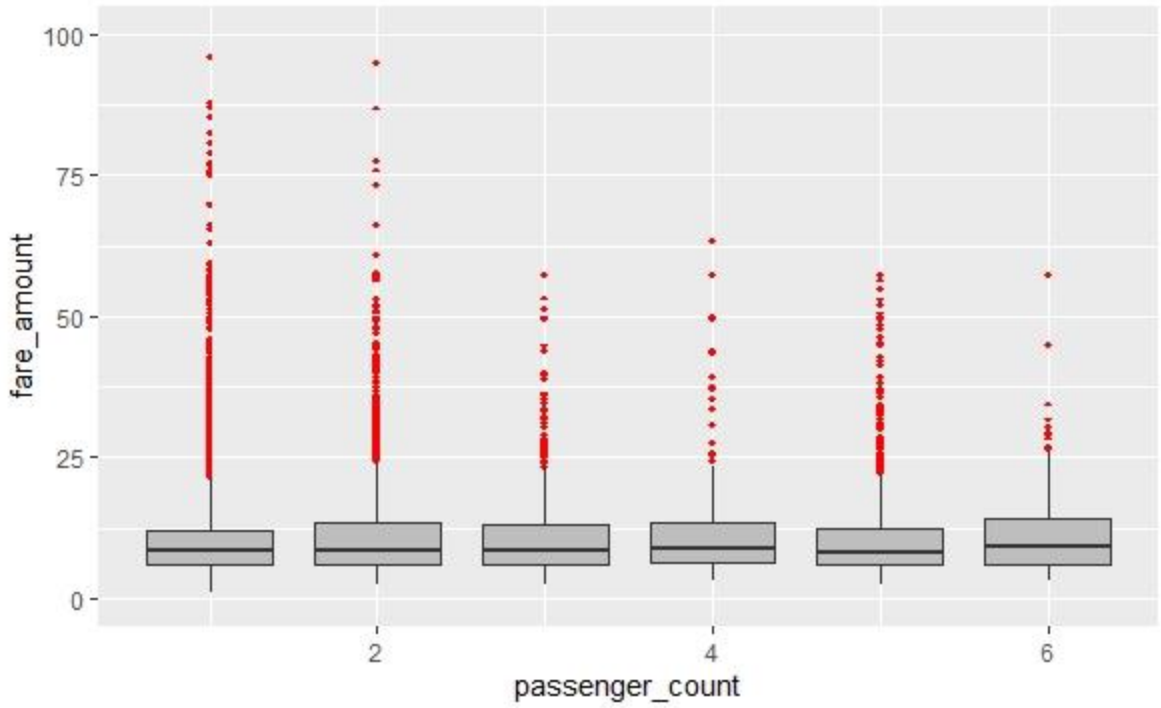


2.3 Outlier analysis

Outlier are observations inconsistent with rest of the dataset, it is due to poor data quality, low quality measurement, malfunctioning equipment and manual error. Sometime data may be correct, but exceptional data. In the statistical central tendency measure, the mean is more sensitive to outliers compared to median measure.

Outliers are detected by box plot (Graphical tool), The Grubb's test for outliers, outliers are detected with the help of R and Python tools, based on the data, Outlier will be removed or replaced as a missing value. These missing values are imputed with better method.

In this train data Box plot has been used to identify the Outliers in continuous variable 'fare_amount'. These outliers are replaced with NA, the 'fare_amount" variable has missing percentage of around 8.7%. The KNN imputation method is used to impute outlier in this analysis. The K = 3 is used for missing value imputation.



2.4 Feature engineering

- Feature engineering is deriving new variables from existing variables, the 'pickup_datetime' variable have information of date, weekday, month, year and hour information in factor data form, it is converted into date and time format. Then all the information is stored in new variables such as 'pickup_date', 'pickup_weekday', 'pickup_month', 'pickup_year', 'pickup_hour'. Same process is followed in test data.

- Distance travelled is calculated based on 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', variables by using Harversine formula. As we know that earth is spherical or elliptical in nature so the 'Harversine formula help has to calculate distance between two points that is shorter distance over the earth's surface distance. The formula to calculate the distance is shown below.

$$a = \sin^2(\Delta\phi/2) + \cos\phi1 \cdot \cos\phi2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

signs: ϕ is latitude in radian, λ is longitude, R is earth's radius (6,371km)

- Distance is calculated and stored in variable ‘distance’, some the values are ranges over thousand, those values are due to human error or data collection error, those values are considered as outlier and deleted.

- Variables used for feature engineering are deleted such as ‘pickup_longitude’, ’pickup_latitude’, ’dropoff_longitude’, ’dropoff_latitude’, ‘pickup_datetime’, ‘pickup_date’.

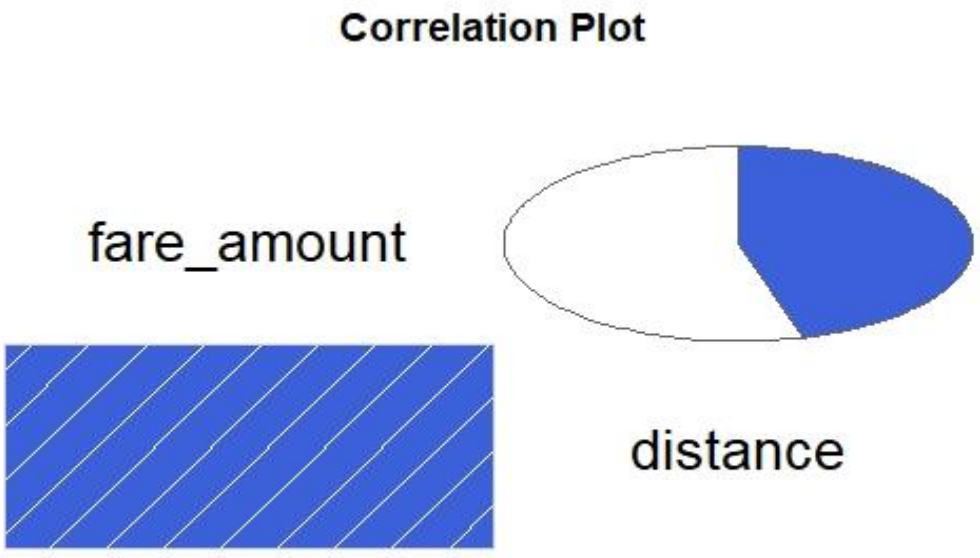
2.5 Feature selection

- Feature selection/Variable importance is selecting a subset of relevant feature (variable/predictor) for use in model construction.

- Data subset of a learning algorithm’s input variable upon which it should focus attention, while ignoring the rest.

It is useful for dimensionality reduction, few variables are highly correlated to each other, it will reduce the model accuracy, for dimensionality reduction correlation analysis used for continuous variables.

In this analysis Correlation plot is used for the continuous variables to check highly correlated variables, we have found some correlation between ‘fare_amount and ‘distance variable around 0.41. it is in acceptable range.



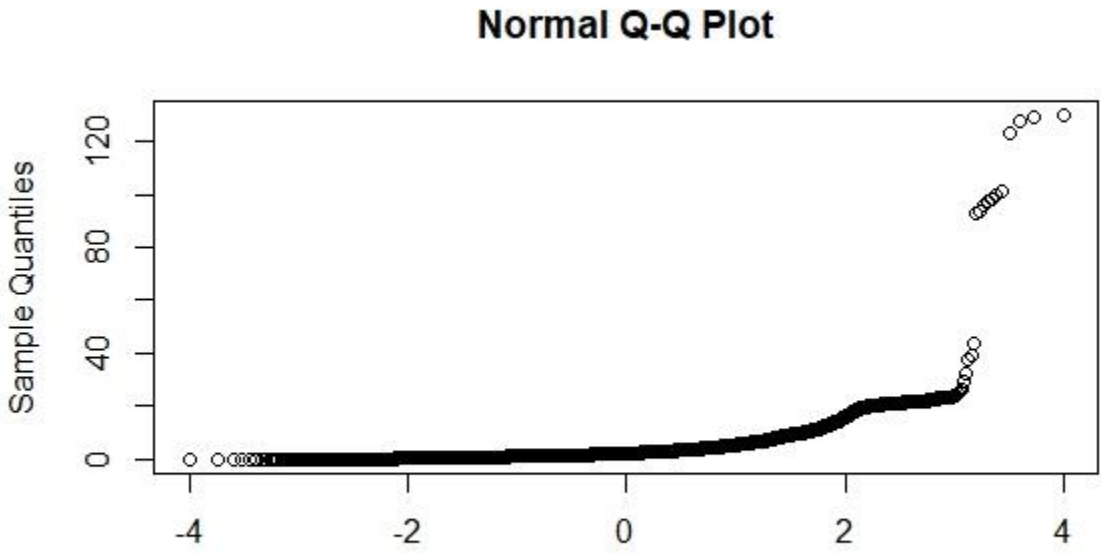
2.6 Feature scaling

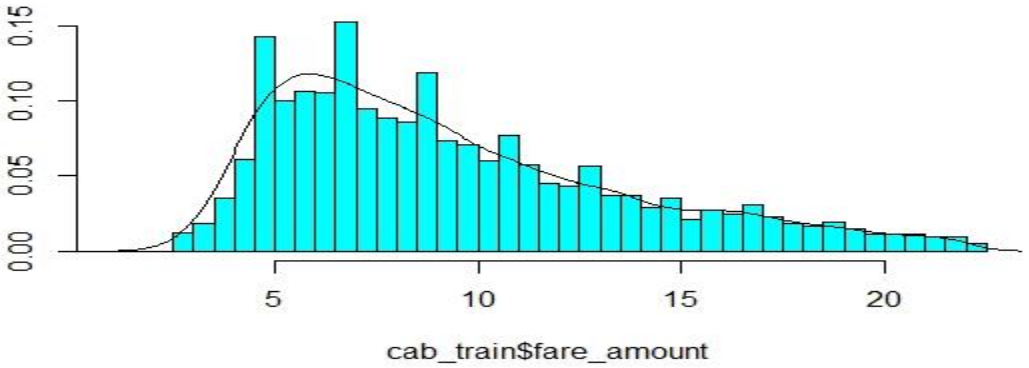
Feature scaling involves normalization and standardization processing of the data.

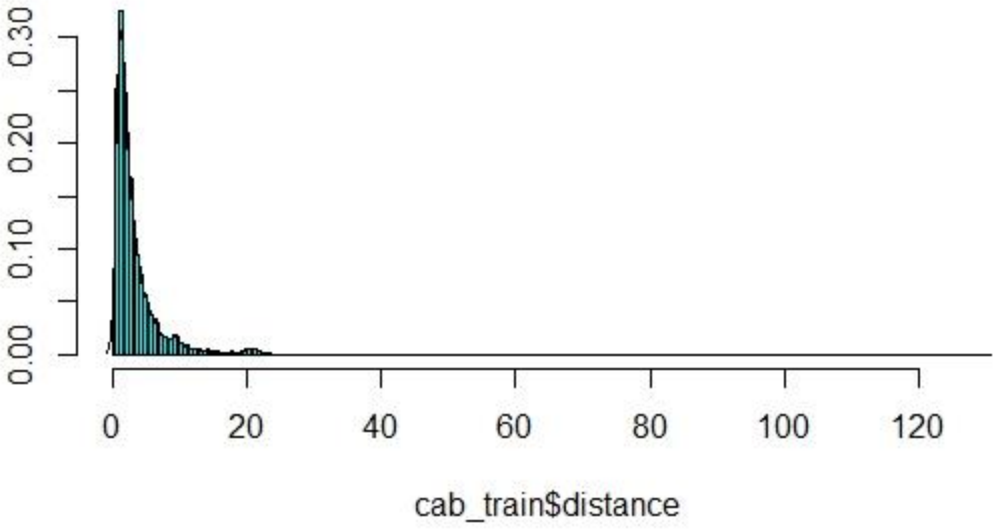
Normalization – it is a database design technique which reduces redundancy and dependency of data. It is used to bring data variables to a common scale into proportion with one another in a range between 0 to 1. If data are not uniformly distributed, will for normalization technique.

Standardization – it works well if the data is uniformly distributed, data points are expressed in positive and negative standard deviation. It shows how many units away from the mean.

The Normal Q-Q plot and histogram on continuous variables shows that the data is non-uniformly distributed. For this analysis normalization technique is used to bring all the data in a common scale







2.7 Dummy variables

The train data set has 5 categorical variables are in numeric form, our target variable 'fare_amount 'are continuous, we will develop model related to regression, Hence, all the categorical variables transformed into dummy variables based on number of categories present in each variable. This method also followed in test data.

Chapter 3

Model development

3.1 Model selection

The data underwent exploratory analysis, pre-processing analysis for model development, our target variable is the continuous variable. Hence, we have to go for regression related models, to check the performance of the model, we used error metrics, based error metrics we will decide the best predictive model for fare_amount prediction

3.2 Linear regression

The linear regression is one of the prediction models, it classified into simple linear regression and multiple linear regression model, it describes the relationship among variables such as one dependent variable may relate to one independent variable or multiple independent variables.

Linear regression (tools)	Root Mean Squared Error (RMSE)	Coefficient of determination (R2)
R software	3.6149488	0.3088124
Python software	3.9543800581630935	0.22577649071824646

The error rate almost similar to R and Python linear models but coefficient of determination value is too low. Hence, we will go for other models

3.3 Decision tree-regression model

The Decision tree model is predictive model, it belongs to supervised learning model, this model used for both classification and regression problems. Here we are using this model for regression purpose; regression trees are used to predict the target variable. It breaks down the dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.

Decision tree model (tools)	Root Mean Squared Error (RMSE)	Coefficient of determination (R2)
R software	2.6349247	0.6226021
Python software	2.690814781759607	0.6415099143079993

The error rate of Decision tree model and Coefficient of determination values are better than linear model, but coefficient of determination is lower, so we will go for other models

3.4 Random forest model

This model is one of the supervised learning models. The random forest is an ensemble that consists of many decision trees, this method combines Breiman's "bagging idea" and the random selection of features. This method can be used for both classification and regression.

In this analysis we have used the random forest model for regression purpose, the number of trees used for analysis is 500 in both R and Python

Random forest (tools)	Root Mean Squared Error (RMSE)	Coefficient of determination (R2)
R software	2.3220966	0.7070344
Python software	2.415041845926718	0.7112254868070074

The error rate of Random forest model is better than Linear regression and Decision tree models, coefficient of determination value also better than two models, but it takes more time to run the model. We will look for another model.

3.5 XGboost model

Gradient boosting is currently one of the most popular machine learning techniques for efficient modeling of tabular datasets of all sizes with quick pace.

It is a technique which applicable for regression and classification type of problems. In this method, multiple weak learners are ensemble to create strong learners. Gradient boosting uses all data to train each learner. But instances that were misclassified by the previous learners are given more weight, so that subsequent learners can give more focus to them during training. Here we select XGBoost for model development.

XGboost model (tools)	Root Mean Squared Error (RMSE)	Coefficient of determination (R2)
R software	2.3023984	0.7120007
Python software	2.35315518047069	0.7258358433836534

The error rate of XGboost model is acceptable and coefficient of determination value also better than other models, it is faster than random forest model and other models used in the model development. So, here we go for XGboost as final model.

Chapter 4

Conclusion

4.1 Model evaluation

Three models are used for 'fare_amount' prediction such as Linear regression mode, Decision tree, Random forest model, XGboost model. These models are evaluated by using error metrics such as Root mean squared error (RMSE) and Coefficient of determination (R2).

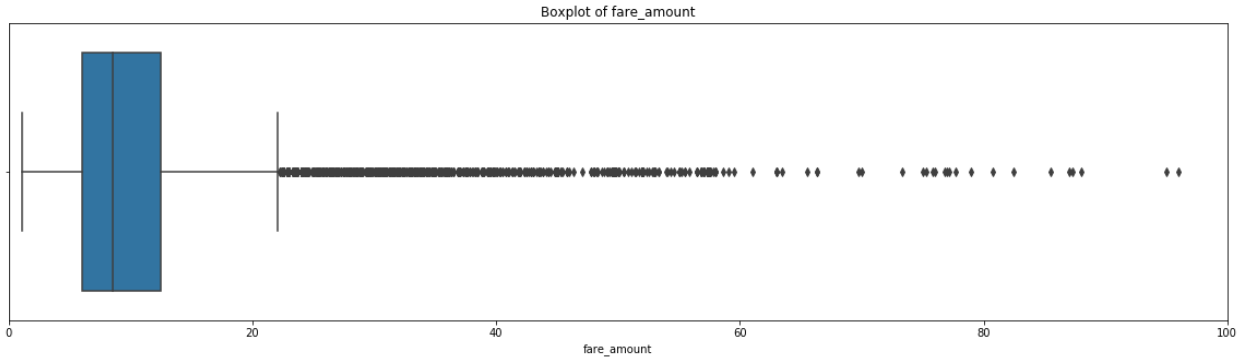
RMSE – It is based the assumption that data error follows normal distribution. This is the measure of the average deviation of model predictions from the actual values in the dataset. It ranges from 0 to 1, model should have lower value for better performance.

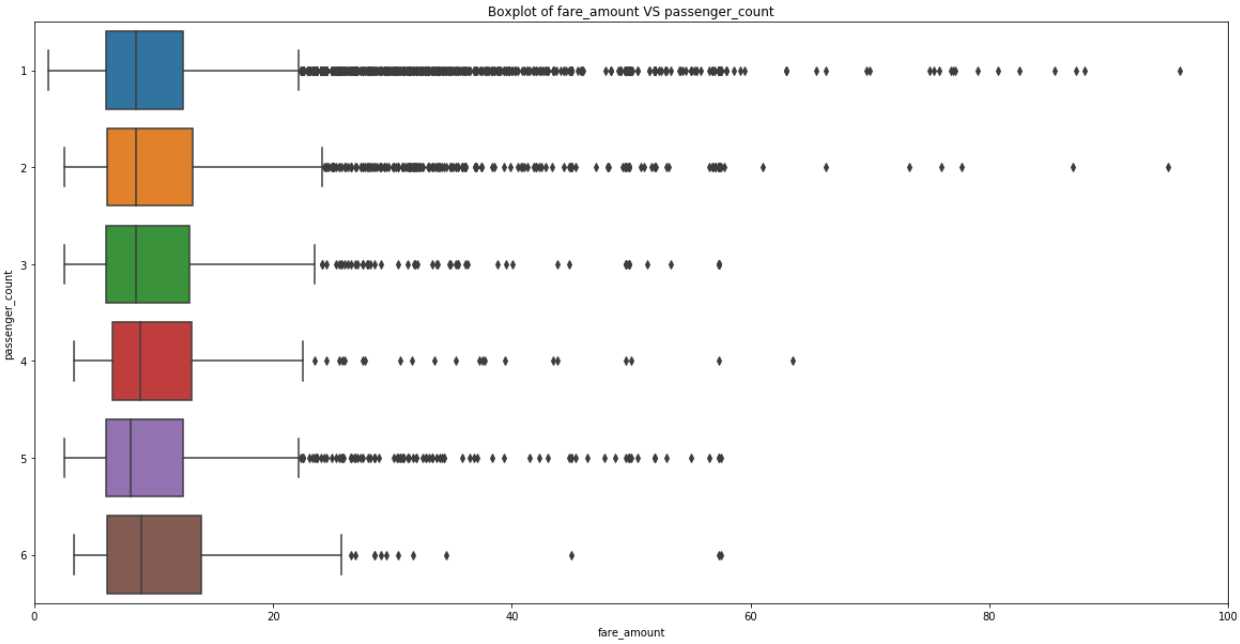
R2 - It is the proportion of the variance of a dependent variable that's explained by an independent variable or variables in the regression model. Whereas co-relation explains the strength of the relationship between an independent variables and dependent variable. It ranges from 0 to 1, model should have a higher value for goodness of fit.

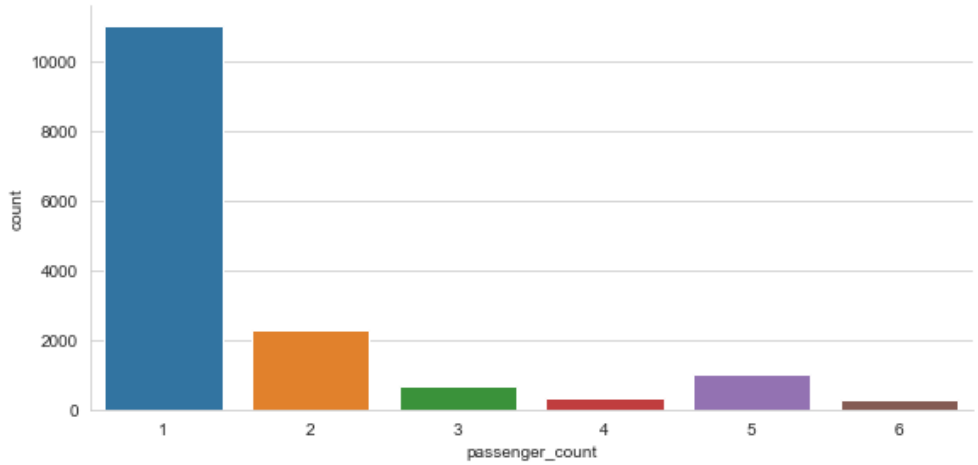
4.2 Model selection

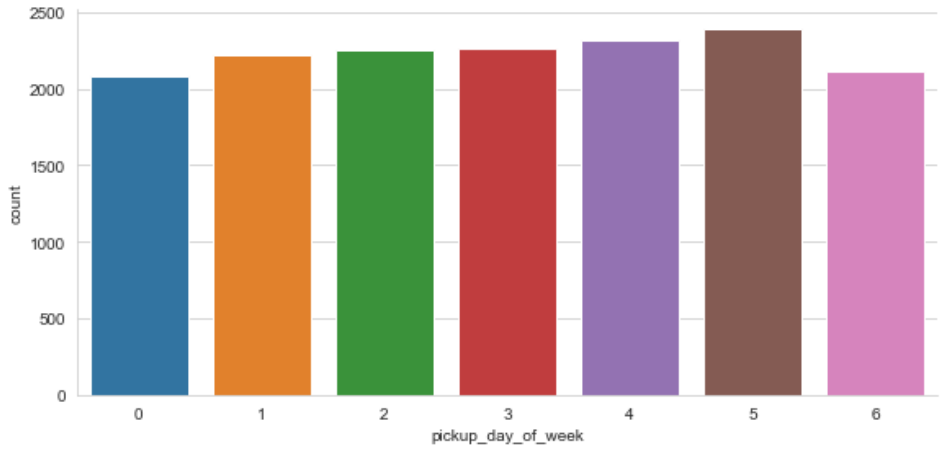
Based on the error scores ***XGboost model*** is better than linear regression, decision tree and random forest model. It has lower RMSE and higher R2 value. It is used for test data prediction.

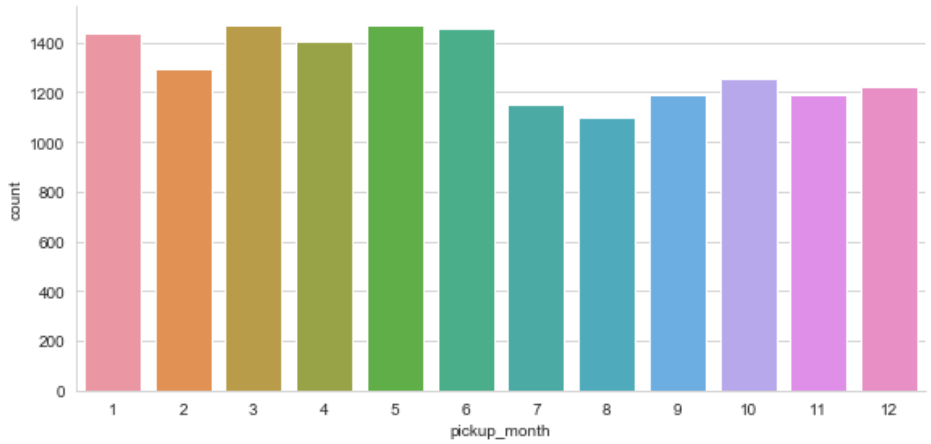
Appendix A - Python plots

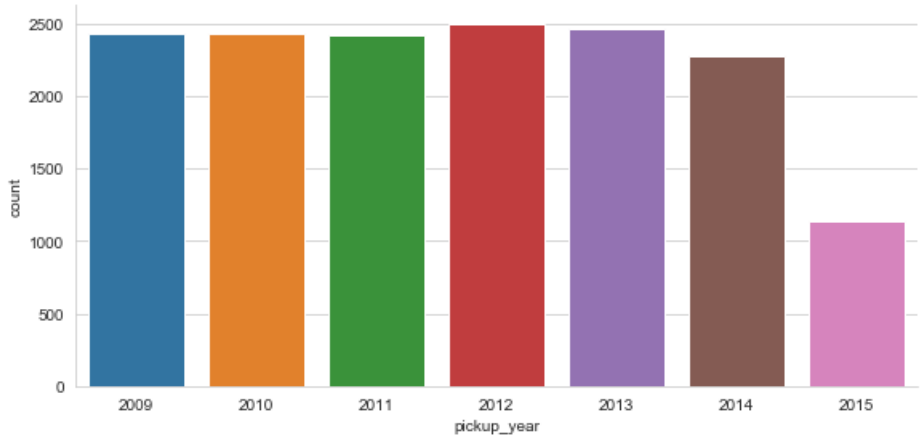


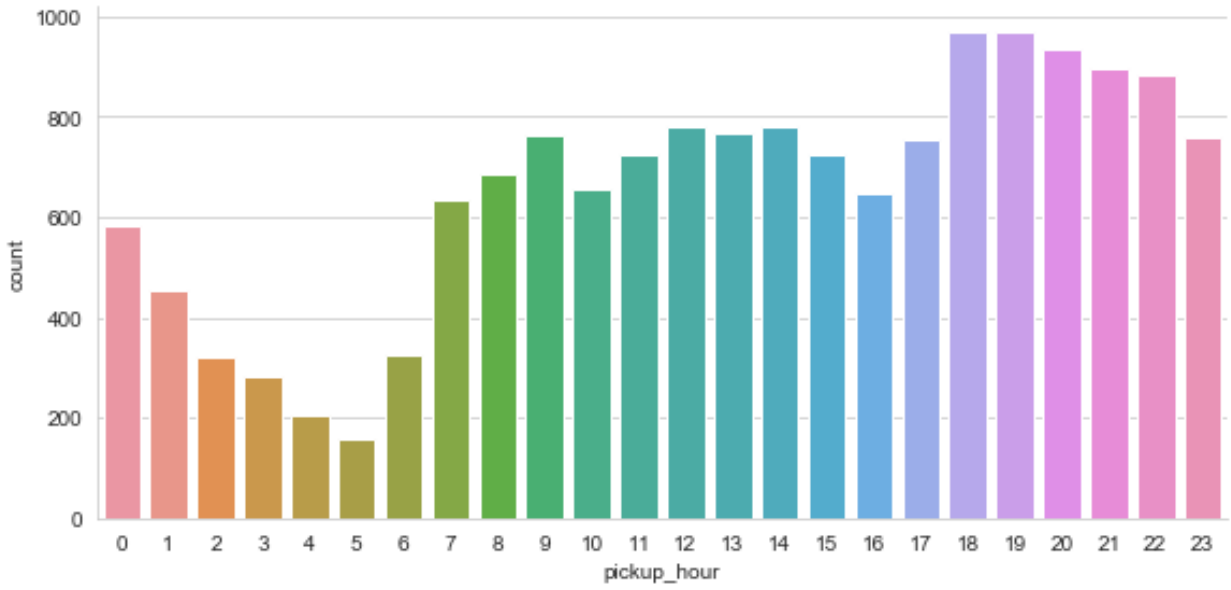


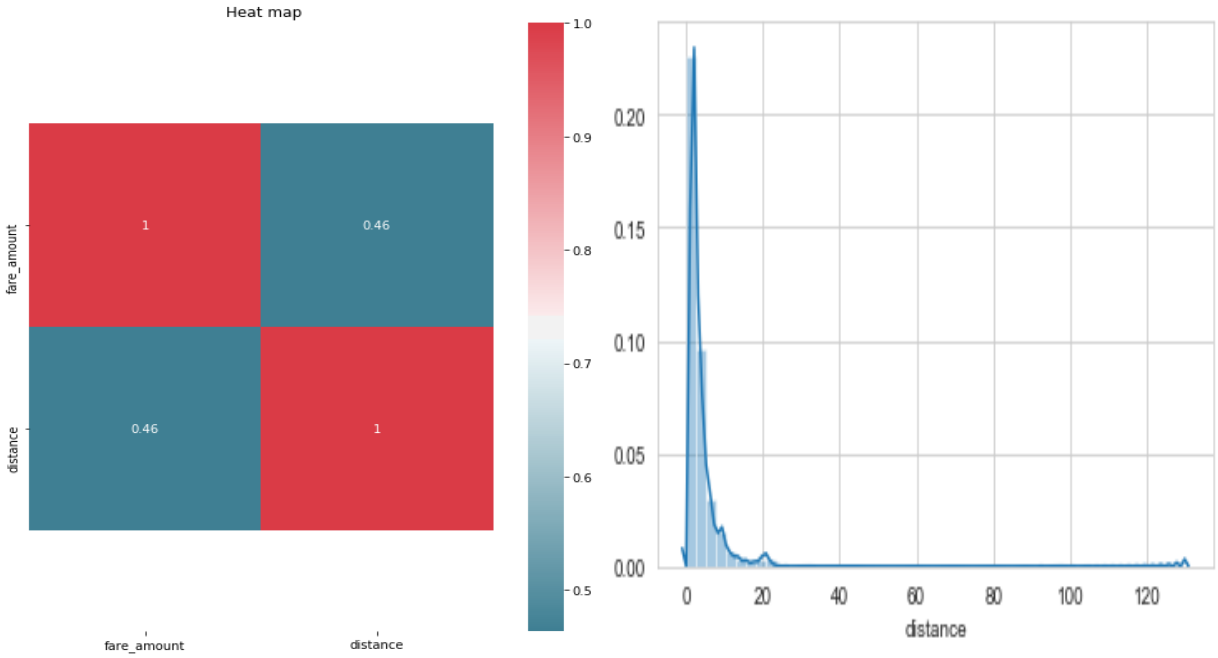



2500
2000
1500
1000
500
0
count
0
1
2
3
4
5
6
pickup_day_of_week

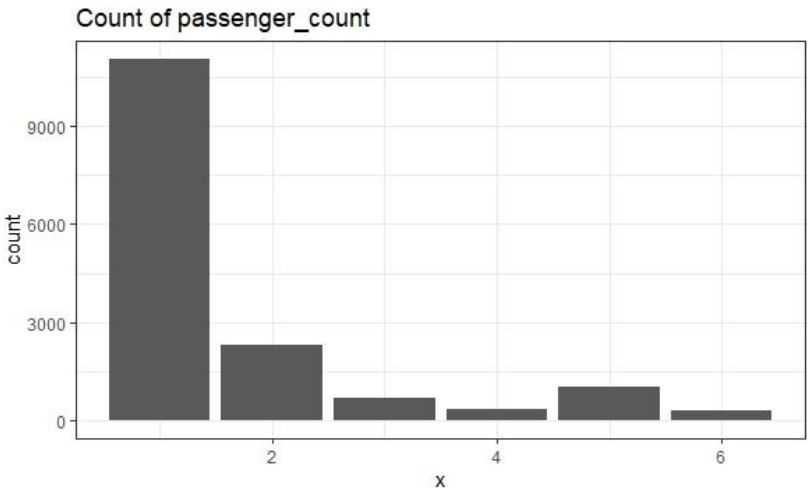

1400
1200
1000
800
600
400
200
0
count
1
2
3
4
5
6
7
8
9
10
11
12
pickup_month

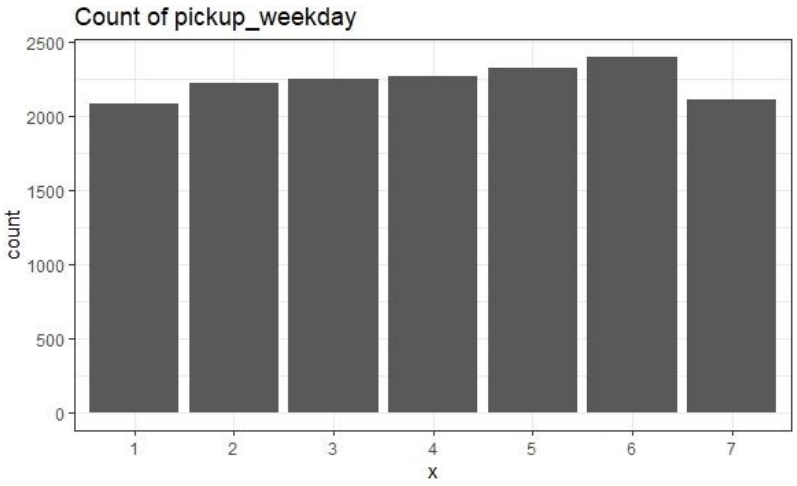

2500
2000
1500
1000
500
0
count
2009
2010
2011
2012
2013
2014
2015
pickup_year

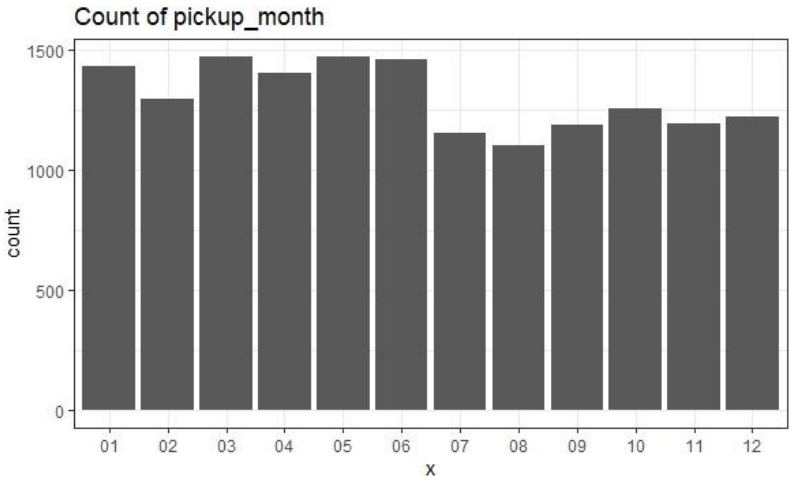

1000
800
600
400
200
0
count
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
pickup_hour

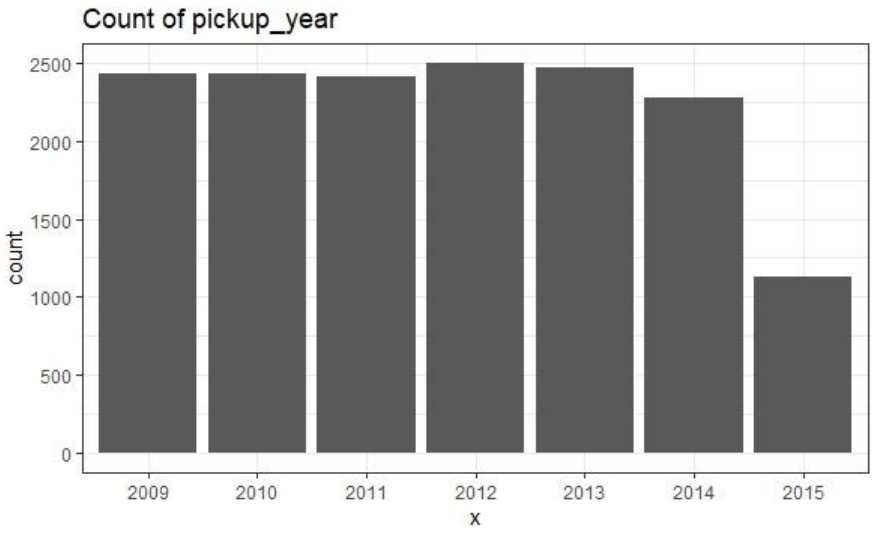

Heat map
fare_amount
distance
1
0.46
0.46
1
fare_amount
distance
1.0
0.9
0.8
0.7
0.6
0.5
0.20
0.15
0.10
0.05
0.00
0 20 40 60 80 100 120
distance

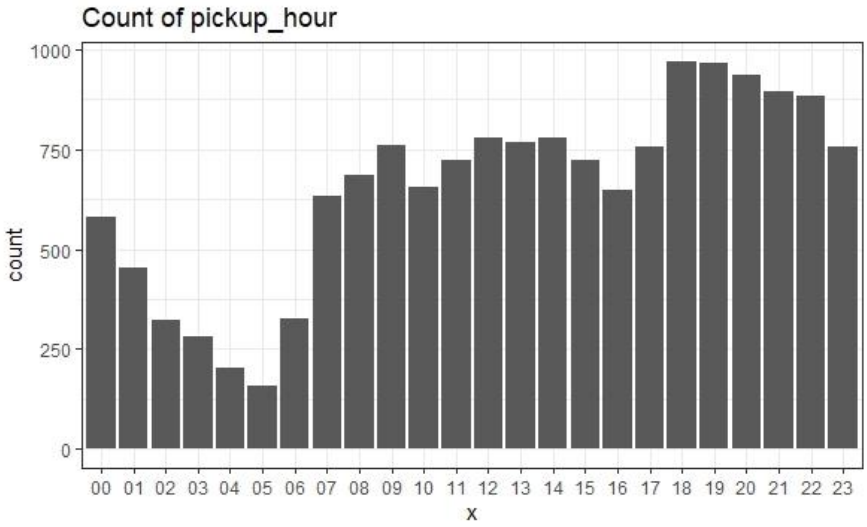
Appendix B – R plots

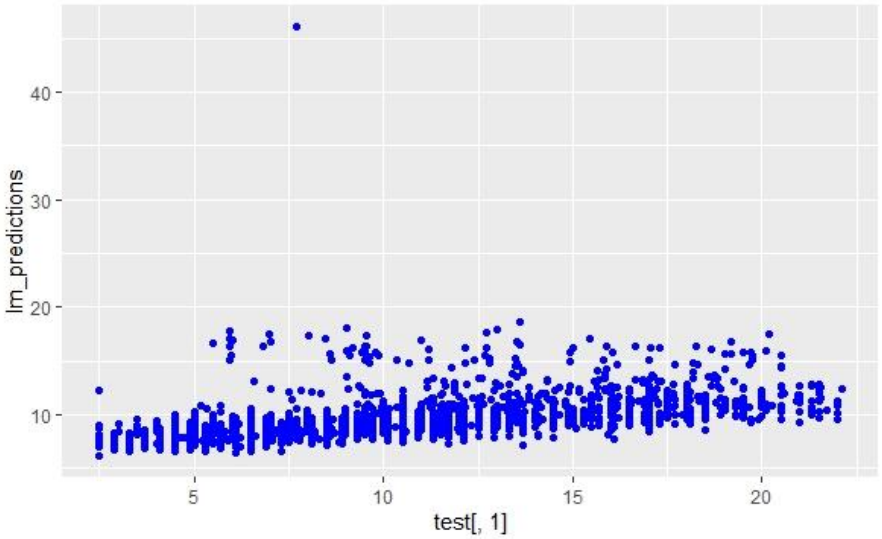


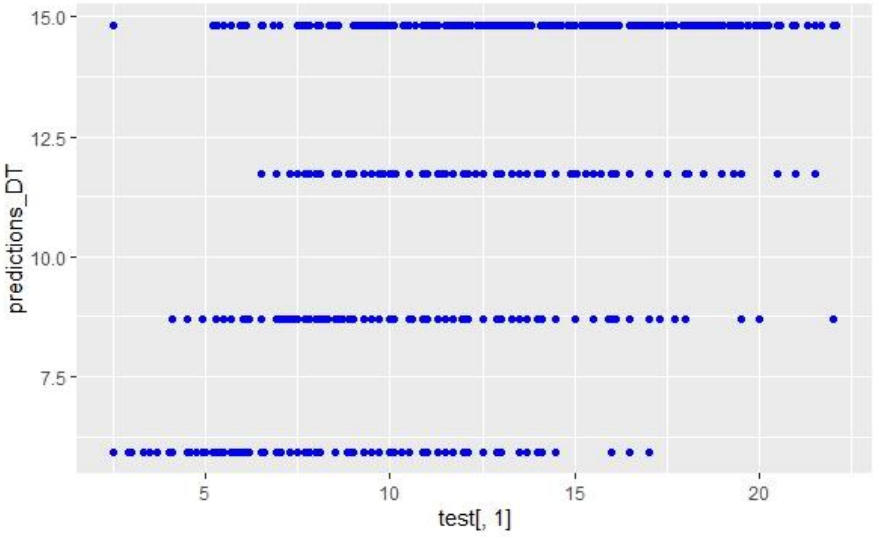


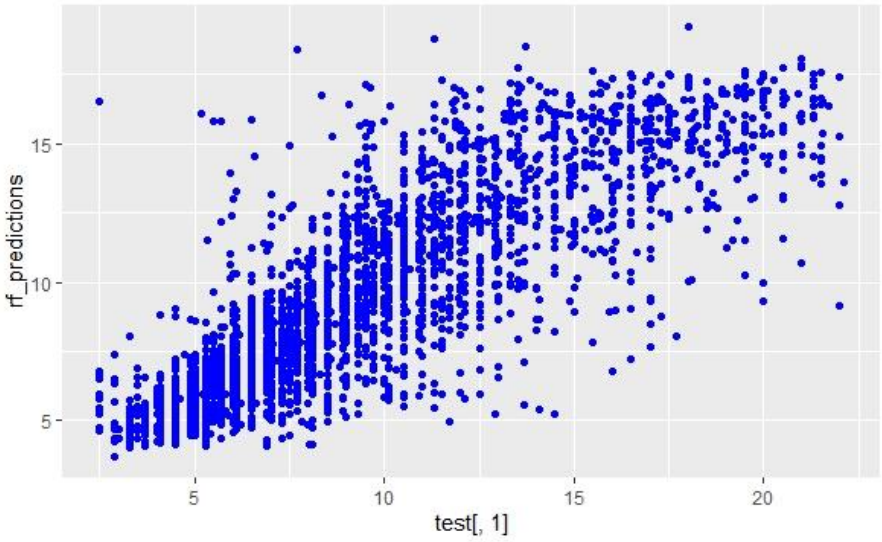


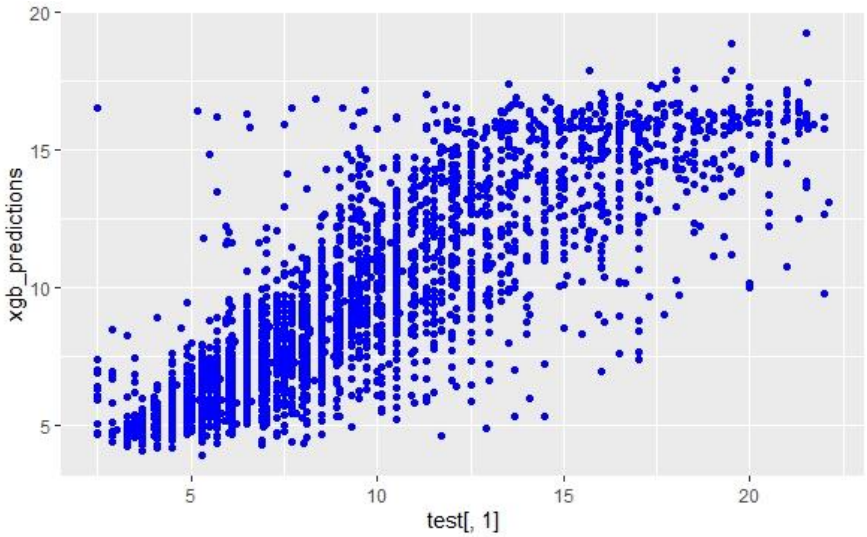

Count of pickup_year
2500
2000
1500
1000
500
0
count
2009
2010
2011
2012
2013
2014
2015
x


Count of pickup_hour
1000
750
500
250
0
count
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
x


40
30
20
10
lm_predictions
5
10
15
20
test[, 1]


predictions_DT
15.0
12.5
10.0
7.5
5
10
15
20
test[, 1]


rf_predictions
15
10
5
5
10
15
20
test[, 1]


xgb_predictions
20
15
10
5
5
10
15
20
test[, 1]

R – codes

```
# Setting working directory

setwd("E:/edWisor/Project")

#Loading libraries

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced","dummies", "e1071",
"Information", "MASS", "rpart","rpart.plot","gbm", "ROSE","sampling", "DataCombine",
"xgboost","inTrees","usdm","doSNOW","stats")


#Install packages

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


rm(x)

## Read the data

cab_train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))

cab_test =read.csv("test.csv",header = T)


##########################################EXPLORE THE
DATA#########################################

# Check number of rows and columns

dim(cab_train)

dim(cab_test)


# Column names

names(cab_train)

names(cab_test)


# Observe top 5 rows

head(cab_train,5)

head(cab_test,5)
```

```
# Structure of variables
str(cab_train)
str(cab_test)

#Transform data types
cab_train$fare_amount = as.numeric(as.character(cab_train$fare_amount))
cab_train$passenger_count = round(cab_train$passenger_count)

# Summary of the data
summary(cab_train)
summary(cab_test)

test_pickup_datetime = cab_test["pickup_datetime"]

# 1) Unique values ofpassenger_count
unique(cab_train$passenger_count)
table(cab_train$passenger_count)
table(cab_train$passenger_count>6) # 20 observations of passenger_count > 6
sum(is.na(cab_train$passenger_count)) # 55 observations of passenger_count == NA

# removing passenger counts '0' and above '6'
nrow(cab_train[which(cab_train$passenger_count > 6 ),])
nrow(cab_train[which(cab_train$passenger_count < 1 ),])
cab_train = cab_train[-which(cab_train$passenger_count<1),]
cab_train = cab_train[-which(cab_train$passenger_count>6),]

# 2) fare amount - Values below 1 and - ve values replaced with NA
cab_train[which(cab_train$fare_amount < 1),]
```

```
nrow(cab_train[which(cab_train$fare_amount < 1),])#checking number of values below 1


cab_train = cab_train[-which(cab_train$fare_amount < 1),]
nrow(cab_train[which(cab_train$fare_amount < 1),])#checking number of values below 1
sum(is.na(cab_train$fare_amount))


# 3) A latitude-longitude map displays data for any region in the world
# Singed degrees format data is used for analysis
# Latitudes range from -90 to 90, Longitudes range from -180 to 180
# In this format South latitudes and West longitudes preceded by a minus sign
print(paste('pickup_longitude above 180', nrow(cab_train[which(cab_train$pickup_longitude > 180),])))
print(paste('pickup_longitude below -180', nrow(cab_train[which(cab_train$pickup_longitude < -
180),])))
print(paste('pickup_latitude above 90', nrow(cab_train[which(cab_train$pickup_latitude > 90),])))
print(paste('pickup_latitude below -90', nrow(cab_train[which(cab_train$pickup_latitude < -90),])))
print(paste('dropoff_longitude above 180', nrow(cab_train[which(cab_train$dropoff_longitude >
180),])))
print(paste('dropoff_longitude below -180', nrow(cab_train[which(cab_train$dropoff_longitude < -
180),])))
print(paste('dropoff_latitude above 180', nrow(cab_train[which(cab_train$dropoff_latitude > 90),])))
print(paste('dropoff_latitude below -180', nrow(cab_train[which(cab_train$dropoff_latitude < -90),])))


# One oulier in variable 'pickup_latitute" would be removed
cab_train = cab_train[-which(cab_train$pickup_latitude >90),]


# Checking any values equal to '0'
nrow(cab_train[which(cab_train$pickup_longitude == 0),])
nrow(cab_train[which(cab_train$pickup_latitude == 0),])
nrow(cab_train[which(cab_train$dropoff_longitude == 0),])
nrow(cab_train[which(cab_train$dropoff_latitude == 0),])
```

```
# Removing values with '0'
cab_train = cab_train[-which(cab_train$pickup_longitude == 0),]
cab_train = cab_train[-which(cab_train$dropoff_longitude == 0),]

# deleting incorrect data
unique(cab_train$pickup_longitude[cab_train$pickup_longitude>=0])
unique(cab_train$pickup_latitude[cab_train$pickup_latitude<=0])
unique(cab_train$dropoff_longitude[cab_train$dropoff_longitude>=0])
unique(cab_train$dropoff_latitude[cab_train$dropoff_latitude<=0])

cab_train = cab_train[-which(cab_train$pickup_longitude >= 0),]
cab_train = cab_train[-which(cab_train$dropoff_longitude >= 0),]

# Make a copy
df= cab_train
# cab_train = df
############################## Missing value analysis ############################
missing_val = data.frame(apply(cab_train,2,function(x){sum(is.na(x))}))

#Converting row names into column names
missing_val$Columns = row.names(missing_val)
row.names(missing_val) = NULL
# Rename the variable
names(missing_val)[1]= "Missing_percentage"
# Calculating percentage
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(cab_train))*100
# Arrange in descending order
missing_val = missing_val[order(-missing_val$Missing_percentage),]
```

```
row.names(missing_val) = NULL

# Rearrange the column

missing_val= missing_val[,c(2,1)]


# Writing output result back to disk

write.csv(missing_val,"cab_train_mp.csv",row.names = F)


# Visualization of missing value

ggplot(data = missing_val[1:6,], aes(x=reorder(Columns, -Missing_percentage),y =
Missing_percentage))+

 geom_bar(stat = "identity",fill = "greenyellow")+xlab("Parameter")+

 ggtitle("Missing data percentage (Train)") + theme_bw()


# Identifying imputation method

cab_train[["fare_amount"]][15]

cab_train[["fare_amount"]][15]= NA


# Actual value = 12.5(fare amount),1(passenger_count)

# Mean imputed value = 15.11(fare amount),1.65(passenger_count)

# Median imputed value = 8.5(fare amount),1(passenger_count)

# KNN imputed value = 10(fare amount),1.33(passenger_count)


# Mean method

#cab_train$fare_amount[is.na(cab_train$fare_amount)] = median(cab_train$fare_amount, na.rm = T)

#cab_train$passenger_count[is.na(cab_train$passenger_count)] = median(cab_train$passenger_count,
na.rm = T)


# Median method

#cab_train$fare_amount[is.na(cab_train$fare_amount)] = median(cab_train$fare_amount, na.rm = T)
```

```
#cab_train$passenger_count[is.na(cab_train$passenger_count)] = median(cab_train$passenger_count,
na.rm = T)


# KNN imputation
cab_train = knnImputation(cab_train, k=3)
sum(is.na(cab_train))


#write.csv(cab_train, 'cab_train_missing.csv', row.names = F)
#################################### Outlier analysis ##################################
cab_train$passenger_count = round(cab_train$passenger_count)
# Boxplot for outliers
pl1 = ggplot(cab_train,aes(passenger_count,fare_amount))
pl1 +geom_boxplot(aes(group= cut_width(passenger_count,0.25)),outlier.colour="red",
                  fill = "grey" ,outlier.shape=18,outlier.size=1, notch=FALSE)+ylim(0,100)


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


# check the NA
fare_amount_missing_val = sum(is.na(cab_train$fare_amount))
fare_amount_missing_percentage = (fare_amount_missing_val/nrow(cab_train))*100
fare_amount_missing_percentage # 8.7%


#median imputation
#cab_train$fare_amount[is.na(cab_train$fare_amount)] = median(cab_train$fare_amount, na.rm = T)
# KNN imputation
cab_train = knnImputation(cab_train, k=3)
sum(is.na(cab_train))
```

```
df1 = cab_train
# cab_train = df1

############################## Feature Engineering ###################################
# 1. Feature engineering for 'pickup_datetime' variable

# Convering 'pickup_datetime' factor into date and time for cab_train
cab_train$pickup_date = as.Date(as.character(cab_train$pickup_datetime))
cab_train$pickup_weekday = as.factor(format(cab_train$pickup_date,"%u"))
cab_train$pickup_month = as.factor(format(cab_train$pickup_date,"%m"))
cab_train$pickup_year = as.factor(format(cab_train$pickup_date,"%Y"))
pickup_time = strptime(cab_train$pickup_datetime,"%Y-%m-%d %H:%M:%S")
cab_train$pickup_hour = as.factor(format(pickup_time,"%H"))

# Convering 'pickup_datetime' factor into date and time for cab_test
cab_test$pickup_date = as.Date(as.character(cab_test$pickup_datetime))
cab_test$pickup_weekday = as.factor(format(cab_test$pickup_date,"%u"))
cab_test$pickup_month = as.factor(format(cab_test$pickup_date,"%m"))
cab_test$pickup_year = as.factor(format(cab_test$pickup_date,"%Y"))
pickup_time = strptime(cab_test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
cab_test$pickup_hour = as.factor(format(pickup_time,"%H"))

sum(is.na(cab_train))
cab_train = na.omit(cab_train)

cab_train = subset(cab_train,select=-c(pickup_datetime,pickup_date))
cab_test = subset(cab_test,select=-c(pickup_datetime,pickup_date))
```

```
# 2.Calculating distance travelled using latitude and longitude
deg_to_rad = function(deg){
 (deg*pi)/180
}
harversine = function(long1,lat1,long2,lat2){
 phi1 = deg_to_rad(lat1)
 phi2 = deg_to_rad(lat2)
 delphi = deg_to_rad(lat2 - lat1)
 dellamda = deg_to_rad(long2 - long1)

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

 c = 2 * atan2(sqrt(a),sqrt(1-a))
 R = 6371e3
 R * c/1000 #1000 is used to convert to KM
}
# Harverstine formula is used to calculate distance for both cab_train and cab_test
cab_train$distance =
harversine(cab_train$pickup_longitude,cab_train$pickup_latitude,cab_train$dropoff_longitude,cab_train$dropoff_latitude)
cab_test$distance =
harversine(cab_test$pickup_longitude,cab_test$pickup_latitude,cab_test$dropoff_longitude,cab_test$dropoff_latitude)

# Removing variables used for feature enginerring the new variables
cab_train = subset(cab_train,select=-
c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
cab_test = subset(cab_test,select=-
c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
```

```
str(cab_train)

summary(cab_train)


nrow(cab_train[which(cab_train$distance >130 ),])

# considering the distance 130 as max and considering rest as outlier.

cab_train=cab_train[-which(cab_train$distance >130 ),] # removing 2 data

nrow(cab_test[which(cab_test$distance >130 ),])

#nrow(cab_train[which(cab_train$distance == 0),])

#nrow(cab_test[which(cab_test$distance == 0),])

############################################ Graph
############################################################


b1<-ggplot(data=cab_train, aes_string(x=cab_train$passenger_count),fill =cab_train$passenger_count)
+ geom_bar(stat ='count')+ ggtitle("Count of passenger_count") + theme_bw()

b1

b2<-ggplot(data=cab_train, aes_string(x=cab_train$pickup_weekday),fill=cab_train$pickup_weekday) +
geom_bar(stat = "count")+ ggtitle("Count of pickup_weekday")+ theme_bw()

b2

b3<-ggplot(data=cab_train, aes_string(x=cab_train$pickup_month),fill=cab_train$pickup_month) +
geom_bar(stat = "count")+ ggtitle("Count of pickup_month") + theme_bw()

b3

b4<-ggplot(data=cab_train, aes_string(x=cab_train$pickup_year),fill=cab_train$pickup_year) +
geom_bar(stat = "count")+ ggtitle("Count of pickup_year") + theme_bw()

b4

b5<-ggplot(data=cab_train, aes_string(x=cab_train$pickup_hour),fill=cab_train$pickup_hour) +
geom_bar(stat = "count")+ ggtitle("Count of pickup_hour") + theme_bw()

b5

#################                Feature selection            ###################

cab_train$passenger_count = as.factor(as.numeric(cab_train$passenger_count))

numeric_index = sapply(cab_train,is.numeric) #selecting only numeric
```

```
numeric_data = cab_train[,numeric_index]


# Selecting categorical data

categorical_data = cab_train[,!numeric_index]


cnames = colnames(numeric_data)

catnames = colnames(categorical_data)

#Correlation analysis for numeric variables

corrgram(cab_train[,numeric_index],upper.panel=panel.pie, main = "Correlation Plot")


#Check for multicollinearity using VIF

vifcor(numeric_data)




#################################### Feature Scaling
#################################################

#Normality check

qqnorm(cab_train$distance)

Ht1<-truehist(cab_train$fare_amount) # truehist() scales the counts to give an estimate of the
probability density.

lines(density(cab_train$fare_amount))


Ht2<-truehist(cab_train$distance)

lines(density(cab_train$distance))

#Normalisation


print('distance')

cab_train[,'distance'] = (cab_train[,'distance'] - min(cab_train[,'distance']))/
```

```
 (max(cab_train[,'distance'] - min(cab_train[,'distance'])))


sum(is.na(cab_train))

sum(is.na(cab_test))

#################### Splitting train into train and validation subsets ####################

# Creating dummy variables for categorical variables

categorical_names = names(categorical_data)

cab_train = dummy.data.frame(cab_train,categorical_names)

cab_test = dummy.data.frame(cab_test,categorical_names)


#Splitting data into train and test data

set.seed(1000)

train_index = sample(1:nrow(cab_train), 0.8*nrow(cab_train))

train = cab_train[train_index,]

test = cab_train[-train_index,]


##############          Linear regression          ##################

lm_model = lm(fare_amount ~.,data=train)


summary(lm_model)

str(train)

plot(lm_model$fitted.values,rstandard(lm_model),main = "Residual plot",

    xlab = "Predicted values of fare_amount",

    ylab = "standardized residuals")



lm_predictions = predict(lm_model,test[,2:58])


qplot(x = test[,1], y = lm_predictions, data = test, color = I("blue"), geom = "point")
```

```
regr.eval(test[,1],lm_predictions)
# mae       mse       rmse      mape
# 0.13432102 0.02976247 0.17251803 0.45597995
#Calculate MAE, RMSE, R-squared for testing data
print(postResample(pred = lm_predictions, obs = test$fare_amount))

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

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

summary(Dt_model)
#Predict for new test cases
predictions_DT = predict(Dt_model, test[,2:58])

qplot(x = test[,1], y = predictions_DT, data = test, color = I("blue"), geom = "point")

regr.eval(test[,1],predictions_DT)
# mae       mse       rmse      mape
# 0.09313101 0.01580352 0.12571206 0.28612238
print(postResample(pred = predictions_DT, obs = test$fare_amount))

##############                    Random forest          ######################
rf_model = randomForest(fare_amount ~.,data=train,importance = TRUE, ntree = 500)

summary(rf_model)

rf_predictions = predict(rf_model,test[,2:58])
```

```
qplot(x = test[,1], y = rf_predictions, data = test, color = I("blue"), geom = "point")


regr.eval(test[,1],rf_predictions)

# mae       mse       rmse      mape

# 0.07947964 0.01226177 0.11073290 0.23835453

print(postResample(pred = rf_predictions, obs = test$fare_amount))



############        Improving Accuracy by using Ensemble technique ---- XGBOOST
############################

train_data_matrix = as.matrix(sapply(train[-1],as.numeric))

test_data_data_matrix = as.matrix(sapply(test[-1],as.numeric))


xgboost_model = xgboost(data = train_data_matrix,label = train$fare_amount,nrounds = 15,verbose =
FALSE)


summary(xgboost_model)

xgb_predictions = predict(xgboost_model,test_data_data_matrix)


qplot(x = test[,1], y = xgb_predictions, data = test, color = I("blue"), geom = "point")


regr.eval(test[,1],xgb_predictions)

# regr.eval(test[,1],xgb_predictions)

# mae       mse       rmse      mape

# 0.07954457 0.01219059 0.11041099 0.23978363

print(postResample(pred = xgb_predictions, obs = test$fare_amount))


plot(test$fare_amount,type="l",lty=1.8,col="Green")

lines(xgb_predictions,type="l",col="Blue")
```

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

# In this step we will train our model on whole training Dataset and save that model for later use

train_data_matrix2 = as.matrix(sapply(cab_train[-1],as.numeric))

test_data_matrix2 = as.matrix(sapply(cab_test,as.numeric))


xgboost_model2 = xgboost(data = train_data_matrix2,label = cab_train$fare_amount,nrounds =
15,verbose = FALSE)


# Saving the trained model

saveRDS(xgboost_model2, "./final_Xgboost_model_using_R.rds")


# loading the saved model

super_model <- readRDS("./final_Xgboost_model_using_R.rds")

print(super_model)


# Lets now predict on test dataset

xgb = predict(super_model,test_data_matrix2)

test_final = data.frame(xgb)



# Now lets write(save) the predicted fare_amount in disk as .csv format

write.csv(test_final,"xgb_predictions_R.csv",row.names = FALSE)
```

Python – codes

```
# loading the required libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from fancyimpute import KNN
from random import randrange, uniform
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm
from statsmodels.formula.api import ols
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import metrics
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score

# Set working directory
os.chdir("E:\edWisor\Project")
```

```
os.getcwd()

# Load the data

cab_train = pd.read_csv("train_cab.csv",sep =
',',dtype={'fare_amount':np.float},na_values={'fare_amount':'430-'})

cab_test = pd.read_csv("test.csv",sep = ',')


data =[cab_train,cab_test]

for i in data:

   i['pickup_datetime']=pd.to_datetime(i['pickup_datetime'],errors='coerce')

# Shape

cab_train.shape,cab_test.shape

# Column names

cab_train.columns,cab_test.columns

# First 5 Observation

cab_train.head(5)

cab_test.head(5)

cab_train.dtypes

cab_test.dtypes

cab_train.describe()

cab_test.describe()

# Exploratory data analysis

cab_train.loc[:,'passenger_count']=cab_train.loc[:,'passenger_count'].round()


# 1)Passenger count( unique values)

cab_train['passenger_count'].unique()

sum(cab_train['passenger_count']>6) # sum of unique value in cab_train

cab_train[cab_train['passenger_count']>6]

len(cab_train[cab_train['passenger_count']<1])
```

```
# unique values in cab_train
cab_train['passenger_count'].unique()
# deleting out off range passenger count
cab_train = cab_train.drop(cab_train[cab_train['passenger_count']>6].index, axis=0)
cab_train = cab_train.drop(cab_train[cab_train['passenger_count']<1].index, axis=0)
sum(cab_train['passenger_count']>6),sum(cab_train['passenger_count']<1)
# 2) Fare amount
sum(cab_train['fare_amount']<1)
cab_train[cab_train['fare_amount']<1]
# fare amount - Values below 1 and - ve values are deleted
cab_train = cab_train.drop(cab_train[cab_train['fare_amount']<1].index, axis=0)
cab_train['fare_amount'].isnull().sum()
# 3) Latitudes range from -90 to 90, Longitudes range from -180 to 180
# In this format South latitudes and West longitudes preceded by a minus sign
print('pickup_longitude above 180={}'.format(sum(cab_train['pickup_longitude']>180)))
print('pickup_longitude below -180={}'.format(sum(cab_train['pickup_longitude']<-180)))
print('pickup_latitude above 90={}'.format(sum(cab_train['pickup_latitude']>90)))
print('pickup_latitude below -90={}'.format(sum(cab_train['pickup_latitude']<-90)))
print('dropoff_longitude above 180={}'.format(sum(cab_train['dropoff_longitude']>180)))
print('dropoff_longitude below -180={}'.format(sum(cab_train['dropoff_longitude']<-180)))
print('dropoff_latitude below -90={}'.format(sum(cab_train['dropoff_latitude']<-90)))
print('dropoff_latitude above 90={}'.format(sum(cab_train['dropoff_latitude']>90)))

for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    print(i,'equal to 0={}'.format(sum(cab_train[i]==0)))

cab_train = cab_train.drop(cab_train[cab_train['pickup_latitude']>90].index, axis=0)
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    cab_train = cab_train.drop(cab_train[cab_train[i]==0].index, axis=0)
```

```
cab_train = cab_train.drop(cab_train[cab_train['pickup_longitude']>=0].index, axis=0)
cab_train = cab_train.drop(cab_train[cab_train['pickup_latitude']<=0].index, axis=0)
cab_train = cab_train.drop(cab_train[cab_train['dropoff_longitude']>=0].index, axis=0)
cab_train = cab_train.drop(cab_train[cab_train['dropoff_latitude']<=0].index, axis=0)

cab_train.shape
df=cab_train.copy()
# cab_train=df.copy()

Missing value analysis
# Creating missing value analysis
missing_val = pd.DataFrame(cab_train.isnull().sum())

# Resetting index
missing_val = missing_val.reset_index()

# Rename variables
missing_val = missing_val.rename(columns={'index':"Variables",0:"Missing_perc"})

#Calculate percentage
missing_val['Missing_perc'] = (missing_val['Missing_perc']/len(cab_train))*100

# Descending order
missing_val = missing_val.sort_values('Missing_perc',ascending = False).reset_index(drop = True)

# Save the missing percentage document
missing_val.to_csv("Missing_perc_cab_train_python.csv",index = False)

missing_val
cab_train = cab_train.reset_index(drop=True)
```

```
#pickup_datetime variable is separated and formed into dataframe, it is mergerd to cab_train for feature engineering

pickup_datetime=pd.DataFrame(cab_train['pickup_datetime'])

pickup_datetime.shape

pickup_datetime.head(5)


Imputation method

cab_train['fare_amount'].loc[14]

# Actual value = 12.5

# Mean imputed value = 15.11

# Median imputed value = 8.5

# KNN imputed value = 8.8

cab_train['fare_amount'].loc[14]=np.nan

cab_train['fare_amount'].loc[14]

cab_train = cab_train.drop(['pickup_datetime'],axis=1)

cab_train.columns

# KNN imputation

imp1=pd.DataFrame(KNN(k=3).fit_transform(cab_train.iloc[0:10000,:]),columns = cab_train.columns)

imp2=pd.DataFrame(KNN(k=3).fit_transform(cab_train.iloc[10000:15652,:]),columns = cab_train.columns)

cab_train = imp1.append(imp2)

cab_train = cab_train.reset_index(drop=True)

cab_train['fare_amount'].loc[14]

df1=cab_train.copy()

# cab_train=df1.copy()

cab_train.loc[:,'passenger_count']=cab_train.loc[:,'passenger_count'].round()

cab_train['passenger_count']=cab_train['passenger_count'].astype('int')

cab_train['passenger_count'].describe()
```

```
Outlier analysis
plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=cab_train['fare_amount'],data=cab_train,orient='h')
plt.title('Boxplot of fare_amount')
# plt.savefig('bp of fare_amount.png')
plt.show()

plt.figure(figsize=(20,10))
plt.xlim(0,100)
_ = sns.boxplot(x=cab_train['fare_amount'],y=cab_train['passenger_count'],data=cab_train,orient='h')
plt.title('Boxplot of fare_amount VS passenger_count')
# plt.savefig('Boxplot of fare_amount w.r.t passenger_count.png')
plt.show()

def outlier_treatment(col):
    ''' calculating outlier indices and replacing them with NA  '''
    #Extract quartiles
    q75, q25 = np.percentile(cab_train[col], [75 ,25])
    print(q75,q25)
    #Calculate IQR
    iqr = q75 - q25
    #Calculate inner and outer fence
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)
    print(minimum,maximum)
    #Replace with NA
    cab_train.loc[cab_train[col] < minimum,col] = np.nan
    cab_train.loc[cab_train[col] > maximum,col] = np.nan

outlier_treatment('fare_amount')
```

```
cab_train.isnull().sum()


# KNN imputation

imp3=pd.DataFrame(KNN(k=3).fit_transform(cab_train.iloc[0:10000,:]),columns =
cab_train.columns)

imp4=pd.DataFrame(KNN(k=3).fit_transform(cab_train.iloc[10000:15652,:]),columns =
cab_train.columns)

# cab_train=pd.DataFrame(KNN(k=3).fit_transform(cab_train),columns = cab_train.columns)


cab_train = imp3.append(imp4)

cab_train = cab_train.reset_index(drop=True)

cab_train.isnull().sum()

df2=cab_train.copy()

# cab_train=df2.copy()


Feature engineering


cab_train['passenger_count']=cab_train['passenger_count'].astype('int').round().astype('object').astyp
e('category')

# we will join 2 Dataframes pickup_datetime and cab_train

cab_train = pd.merge(pickup_datetime,cab_train,right_index=True,left_index=True)

cab_train.head()

cab_train.shape

cab_train = cab_train.reset_index(drop=True)

cab_train.isnull().sum()

# We will drop 1 NA of pickup datetime

cab_train=cab_train.dropna()
```

```
# 1. Feature engineering for 'pickup_datetime' variable
data_1 = [cab_train,cab_test]
for i in data_1:
    i["pickup_year"] = i["pickup_datetime"].apply(lambda row: row.year)
    i["pickup_month"] = i["pickup_datetime"].apply(lambda row: row.month)
    i["pickup_day_of_week"] = i["pickup_datetime"].apply(lambda row: row.dayofweek)
    i["pickup_hour"] = i["pickup_datetime"].apply(lambda row: row.hour)

cab_train.columns
cab_test.columns

# 2.Calutlate the distance using longitude and latitude

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

#Calculate the great circle distance between two points on the earth (specifie d in decimal degrees)
def haversine_np(lon1, lat1, lon2, lat2):
    # Convert latitude and longitude to radians
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
   # Find the differences
    dlon = lon2 - lon1
    dlat = lat2 - lat1
   # Apply the formula
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    # Calculate the angle (in radians)
    c = 2 * np.arcsin(np.sqrt(a))
    # Convert to kilometers
    km = 6367 * c
    return km
```

```
cab_train['distance'] =
haversine_np(cab_train['pickup_longitude'],cab_train['pickup_latitude'],cab_train['dropoff_longitude'],
cab_train['dropoff_latitude'])

cab_test['distance'] =
haversine_np(cab_test['pickup_longitude'],cab_test['pickup_latitude'],cab_test['dropoff_longitude'],
cab_test['dropoff_latitude'])

sum(cab_train['distance']>130)

cab_train = cab_train.drop(cab_train[cab_train['distance']>130].index, axis=0)

cab_train = cab_train.drop(cab_train[cab_train['distance']>130].index, axis=0)

cab_train.head(5)

cab_test.head(5)


# dropping variables used for feature engineering

cab_train=cab_train.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude','dropoff_longitude',
'dropoff_latitude'],axis=1)

cab_test=cab_test.drop(['pickup_datetime','pickup_longitude', 'pickup_latitude','dropoff_longitude',
'dropoff_latitude'],axis=1)


cab_train.shape,cab_test.shape


#Visualization

sns.set_style("whitegrid")

sns.catplot(data=cab_train, x='passenger_count', kind= 'count',height=4,aspect=2)

sns.catplot(data=cab_train, x='pickup_day_of_week', kind= 'count',height=4,aspect=2)

sns.catplot(data=cab_train, x='pickup_month', kind= 'count',height=4,aspect=2)

sns.catplot(data=cab_train, x='pickup_year', kind= 'count',height=4,aspect=2)

sns.catplot(data=cab_train, x='pickup_hour', kind= 'count',height=4,aspect=2)


Feature Selection


# Categorizing data based on continuous and categorical variables


continuous_names =['fare_amount','distance']
```

```
categorical_names =['passenger_count','pickup_year', 'pickup_month', 'pickup_day_of_week', 'pickup_hour']

# Correlation analysis

df_corr = cab_train.loc[:,continuous_names]


#Set the width and height of the plot

f, ax = plt.subplots(figsize=(10, 10))


#Generate the correlation matrix

corr = df_corr.corr()


#Plot using seaborn library(bottom, top = ax1.get_ylim(),ax1.set_ylim(bottom + 0.5, top - 0.5)) is added to remove heat map cut-off

ax1 = 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)

bottom, top = ax1.get_ylim()

ax1.set_ylim(bottom + 0.5, top - 0.5)

plt.title(" Heat map")

plt.savefig('plot9.png', dpi=300, bbox_inches='tight')


#Multicollinearity Test

outcome, predictors = dmatrices('fare_amount ~ distance+passenger_count+pickup_year+pickup_month+pickup_day_of_week+pickup_hour',cab_train, return_type='dataframe')

# calculating VIF for each individual Predictors

vif = pd.DataFrame()

vif["VIF"] = [variance_inflation_factor(predictors.values, i) for i in range(predictors.shape[1])]

vif["features"] = predictors.columns

vif
```

Feature Scaling

```
sns.distplot(cab_train['distance'],bins=50)
# As our variables are left sckew here we select Normalisation method
for i in continuous_names:
    print(i)
    if i == 'fare_amount':
        continue
    cab_train[i] = (cab_train[i] - cab_train[i].min())/(cab_train[i].max() - cab_train[i].min())

#Create dummy variables of factor variables
dm1 = pd.get_dummies(data = cab_train, columns = categorical_names)
dm2 = pd.get_dummies(data = cab_test, columns = categorical_names)

dm3 = dm1.copy()
# dm1=dm3.copy()
dm4=dm2.copy()
# dm2=dm4.copy()
dm1.shape,dm2.shape
dm1.columns
dm2.columns
df3=cab_train.copy()
# cab_train=df3.copy()

# Divide the data into test and train (simple random sampling)
X = dm1.drop('fare_amount',axis=1).values
y = dm1['fare_amount'].values
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state = 1)
print(dm1.shape, X_train.shape, X_test.shape,y_train.shape,y_test.shape)
```

Model development

```
# Train the model using the training sets
model = sm.OLS(y_train, X_train).fit()
# predictions for train model
predictions_LR = model.predict(X_train)
# predictions with test sample
predictions_LR = model.predict(X_test)
#Define function to calculate RMSE
def RMSE(y_true,y_pred):
    rmse = np.sqrt(mean_squared_error(y_true,y_pred))
    return rmse
#Calculate RMSE and R-squared value
print("RMSE: "+str(RMSE(y_test,predictions_LR)))
print("R^2: "+str(r2_score(y_test,predictions_LR)))

model.summary()
#Decision tree

# Decision tree regression
fit_DT = DecisionTreeRegressor(max_depth=2).fit(X_train,y_train)
#Apply model for test data
predictions_DT = fit_DT.predict(X_test)
#Calculate RMSE and R-squared value
print("RMSE: "+str(RMSE(y_test,predictions_DT)))
print("R^2: "+str(r2_score(y_test,predictions_DT)))

#Random forest

rf_model = RandomForestRegressor(n_estimators = 500, random_state = 1).fit(X_train,y_train)
#Perdict for test cases
predictions_RF = rf_model.predict(X_test)
```

```
#Calculate RMSE and R-squared value
print("RMSE: "+str(RMSE(y_test,predictions_RF)))
print("R^2:"+str(r2_score(y_test,predictions_RF)))

#XGboost
# from sklearn.ensemble import GradientBoostingRegressor

# Building model on top of training dataset
fit_GB = GradientBoostingRegressor().fit(X_train, y_train)

predictions_xgb = fit_GB.predict(X_test)
#Calculate RMSE and R-squared value
print("RMSE: "+str(RMSE(y_test,predictions_xgb)))
print("R^2:"+str(r2_score(y_test,predictions_xgb)))

# finilizing model( XGboost)
test_1 = dm2
test_1.shape
train = dm3
train.shape

y_train= dm3['fare_amount']
dm3.drop(['fare_amount'], inplace = True, axis=1)
X_train = dm3

fit_GB = GradientBoostingRegressor().fit(X_train, y_train)
predictions_xgb = fit_GB.predict(test_1)
Test_prediction = pd.DataFrame(predictions_xgb)
Test_prediction.describe()
Test_prediction = Test_prediction.rename(columns={0:"Prediction_fare_amount"})
Test_prediction.to_csv("test_predict_py.csv",index=False)
```

References

1. Edwisor study materials

2. Analytics vidya blogs

3. Github - issues