ADVANCED ANALYSIS ON
USED BIKE PRICES
IN INDIA

GROUP 04

s14853 – Pramudi Rajamanthri
s15030 – Vidura Chathuranga
s15091 – Chalani Wijamunige

ST 3082
Final Project

Abstract

This report aims to outline the findings of the exploratory data analysis which was carried out based on the Used bike prices in India data set obtained from the open-source data site, Kaggle. We have done several comparisons and univariate/ bivariate analysis by comparing variables such as; brand, age, ownership, engine power, number of kilometers driven and state where we were able to understand how each of these play a role on the valuation of used bikes. Furthermore, an advanced analysis was carried out on the used bike prices dataset sourced from Kaggle with an objective of analyzing how the price which is our target variable is affected by many different variables on this dataset and a model with high prediction accuracy was selected in predicting the prices of used bikes.

Contents

List of Figures

List of Tables

1. Introduction

India is one of the largest markets for two-wheeler vehicles in the world, with over 21 million units sold in 2020 alone (source: Society of Indian Automobile Manufacturers). The motorbike segment accounts for a significant portion of this market, with over 15 million motorbikes sold in 2020. Two-wheelers account for over 80% of the total vehicles sold in India and are a popular mode of transportation for both urban and rural populations.

The Indian motorbike market is highly competitive, with several domestic and international players vying for market share. Also, it is characterized by a wide range of models with varying features, price points, and performance levels, catering to different segments of the market. From entry-level commuter bikes to premium sports bikes, there is a two-wheeler for every budget and need.

The market for used bikes in India has been growing steadily in recent years, as more and more consumers are looking for affordable and reliable transportation options. Used bikes provide a cost-effective alternative to buying a new bike, with prices typically ranging from a few thousand rupees to several lakhs depending on the make, model, and condition of the bike. So, for a vast market like Indian motorbike market, it's essential to have accurate and reliable methods for estimating the value of used motorbikes.

The goal of this project is to develop a predictive model for estimating the resale value of used motorbikes in India. Using data from buying and selling and various attributes related to these motorbikes, we aim to predict their prices accurately. By analyzing this dataset and building a predictive model, we can help individuals and businesses to make informed decisions when buying or selling used motorbikes in India which will be useful to a wide range of stakeholders, including buyers, sellers, insurers, and financiers of used motorbikes.

In the following report, we will describe the process of data cleaning, exploratory data analysis, and model development, along with the results and conclusions of our analysis.

2. Description of the Question

The market for used motorbikes in India is growing, and accurate valuation of these vehicles is crucial for buyers, sellers, and other stakeholders. However, estimating the resale value of a used motorbike can be challenging due to factors such as the bike's age, condition, make, and model. This project aims to address this problem by developing a predictive model that can accurately estimate the resale value of used motorbikes in India, based on various attributes related to these vehicles.

Objectives of the analysis: -

1. Identify the most significant factors that impact the resale value of used motorbikes in India.
2. Develop a predictive model for estimating the resale value of used motorbikes in India.

3. Description of the data set

This dataset contains information about 32648 used motor bikes sold on https://droom.in/, an Indian online platform used in buying and selling vehicles. The dataset includes 8 variables which are given below.

Variable	Description	Type
price	Resale price of the bike in Indian Rupees. This is the target variable of the dataset.	Quantitative
bike_name	Full name of the bike. This contains 471 unique categories.	Qualitative
city	City where the bike was listed. This contains 443 unique categories	Qualitative
kms_driven	Total kilometers driven by the bike	Quantitative
owner	Number of owners of the bike 1 – First owner, 2 – Second owner, 3 – Third owner, 4 – Fourth owner	Qualitative
age	Age of the bike in years	Quantitative
power	Power of the bike in CC	Quantitative
brand	Brand of the bike. This contains 23 unique categories.	Qualitative

Table 3.1

4. Data Pre-processing

Used Bikes Prices in India dataset contains data under 8 variables where 4 are categorical and 4 are numerical. In the dataset, out of 32648 records 25324 duplicated entries were discovered and were removed. There were no missing data records in the data set. Finally, cleaned dataset consists of 7324 records.

4.1 Feature Engineering

Initially there were 23 distinct categories in the *brand* variable. After identifying the categories which had the lowest number of cases (Ideal, Indian, LML, Rajdoot, Yezdi, MV and Jawa) they were combined into one category (Others), and a new variable was created named ***"brand_New"*** with 17 distinct categories.

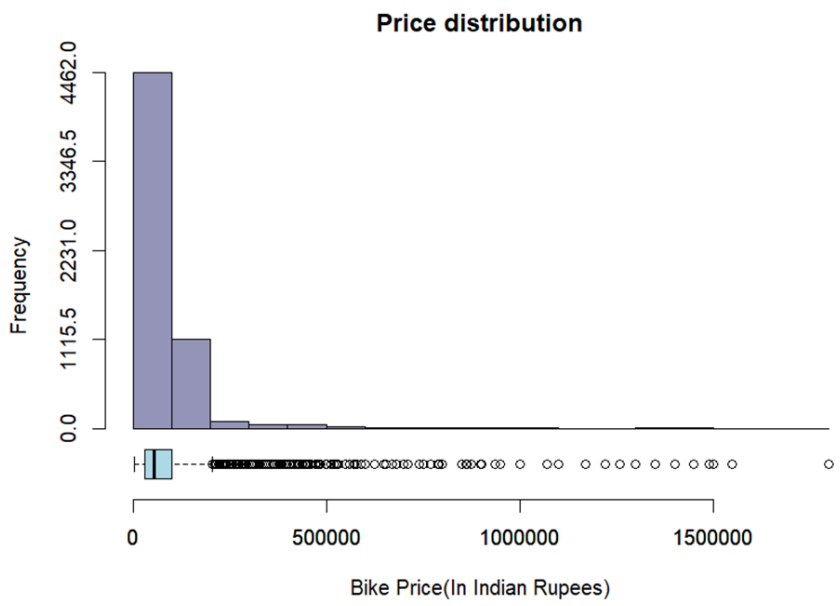
A variable called ***"state"*** was created by categorizing the *city* variable which has 443 unique categories, according to the state which they belong out of the 28 states in India.

Additionally, to simplify the analysis and avoid creating too many categories, three states namely, "Arunachal Pradesh", "Meghalaya", "Sikkim" with the lowest number of counts were grouped together under the category "Others". Then, the newly created ***"state"*** variable contained 25 distinct categories.

Finally, the dataset was randomly split into training and test datasets which contains 80%and 20% of the entries from original dataset respectively.

5. Important Results of the Descriptive Analysis

In the detailed descriptive analysis carried out previously, the behavior of the price was analyzed with several associative factors. Below are some important results obtained.



Boxplot Stats

Minimum - 4400

1st quartile - 30,000

Median - 55,000

3rd quartile - 99,900

Maximum - 2,04600

Figure 5.1

- When the price distribution was analyzed, a positively skewed distribution was observed along with 274 outliers. Hence, considering a transformation on the response would help to overcome the issues arising from the above.
- According to the bar plot in Figure 5.2, the order of preference of the used bikes were identified where brands namely, Bajaj, Royal Enfield, Hero and Honda occupy the front positions respectively. Moreover, they are identified as the brands which are well-established all-over India according to the reports of the "globalnewswire" (https://www.globenewswire.com/en/news-release/2022/06/21/2465834/28124/en/India-Used-Two-Wheeler-Market-Report-2022-2026-A-Highly-Fragmented-Market-with-the-Presence-of-Large-Number-of-Local-Dealers-Small-Mechanics-and-Individual-Sellers.html)considering their approachable prices regarding the mostly identified requirements for used two-wheeler market among Indians.

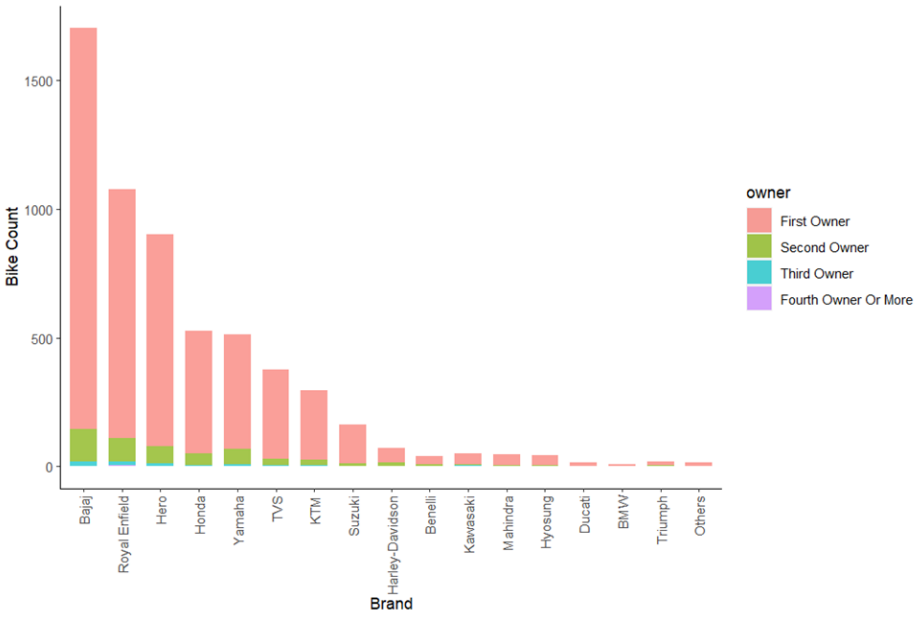


Figure 5.2

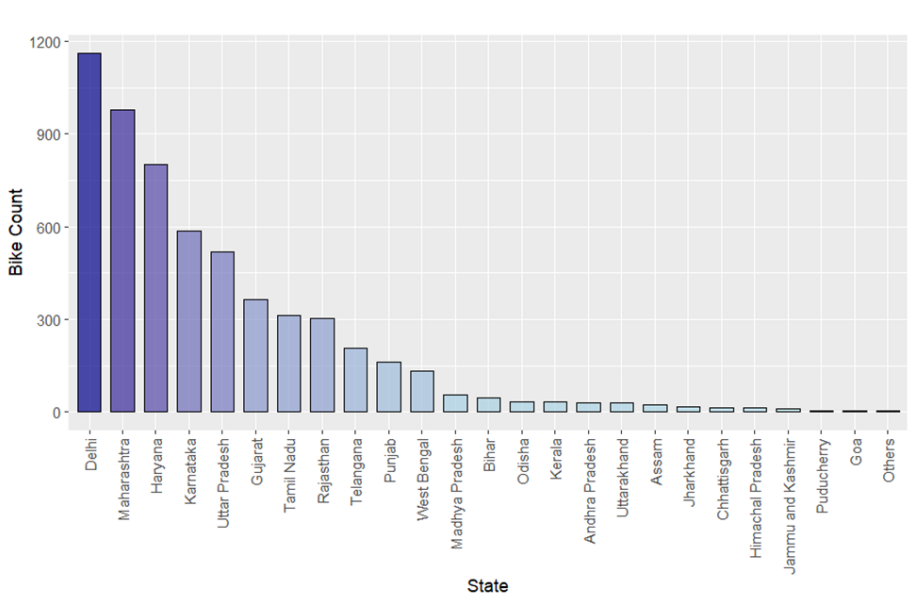


Figure 5.3

- Highly populated states namely, Delhi, Maharashtra seem to have a well-established used bikes market according to the bar plot in Figure 5.3, where "timesofindia" reports (https://timesofindia.indiatimes.com/auto/bikes/india-is-now-worlds-biggest-2-wheeler-market/articleshow/58555735.cms) that India has overtaken China in the two-wheeler

market while New Delhi, the Capital of India is identified among the prominent cities where the market is well established.

- According to scatterplot in Figure 5.4, ***power*** of the engine (in CC) depicts an overall positive linear relationship with the bike prices. Even though the fact that the bikes with the less number of kilometers driven with the small years of ownership(age) have higher prices, ***kms_driven*** and ***age*** variables do not depict highlighted linear relationships with bike price according to scatterplots in Figure 5.5 and Figure 5.6.

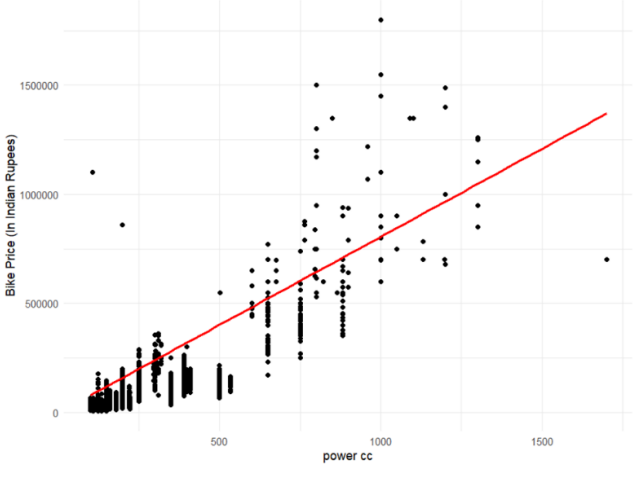


Figure 5.4

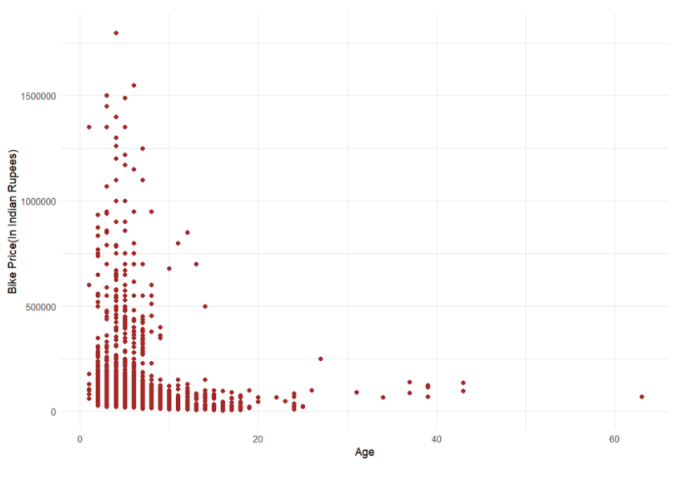


Figure 5.5

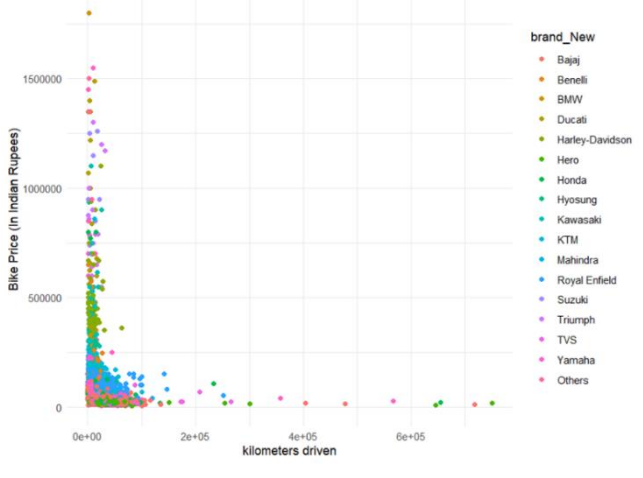


Figure 5.6

- Partial least squares regression (PLSR), resulted dominant VIP scores for ***power*** , ***kms_driven*** and ***age***. Moreover, 39 outliers were identified via PLSR.

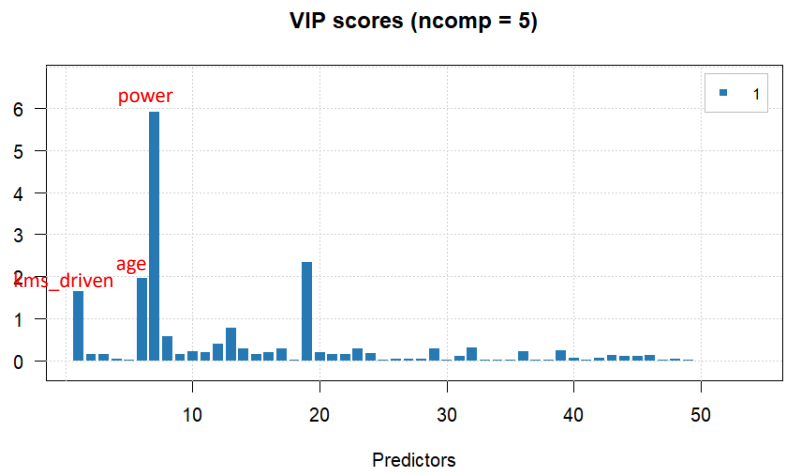


Figure 5.7

- According to Figure 5.8, cluster analysis confirmed the non-existence of clearly distinct clusters where it resulted average silhouette distances less than 0.5 for the considered k values in the range of 2:8 where the highest value was 0.2205 under k=4. Furthermore, Figure 5.9 depicts the unsystematic scattering of the data points with the evidence for the non-existence of clearly distinct clusters.

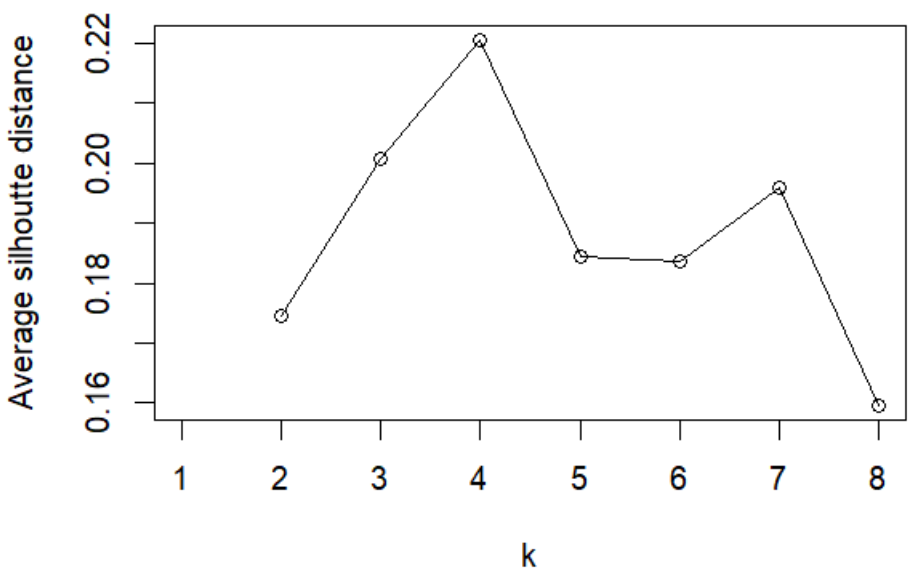


Figure 5.8

Figure 5.9

- From the correlation matrix in Figure 5.10, a fairly positive correlation was identified among the variables *kms_driven* and *age*, but other correlation values don't show any significance which is more clarified from the scatterplot matrix in Figure 5.11. Hence, we decided not to deny the fact of existence of multicollinearity. Moreover, the krukal wallis tests resulted significantly small p – values when considering the effect on the median of the numerical variables by the categories of the categorical variables.

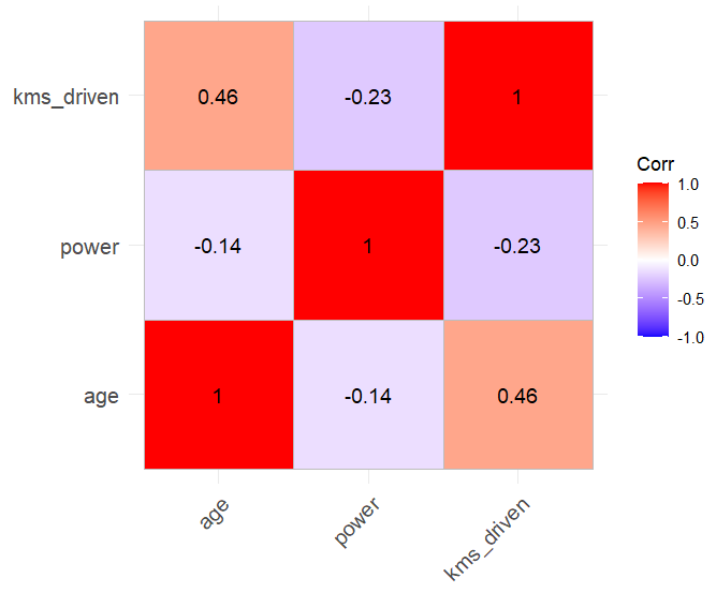


Figure 5.10

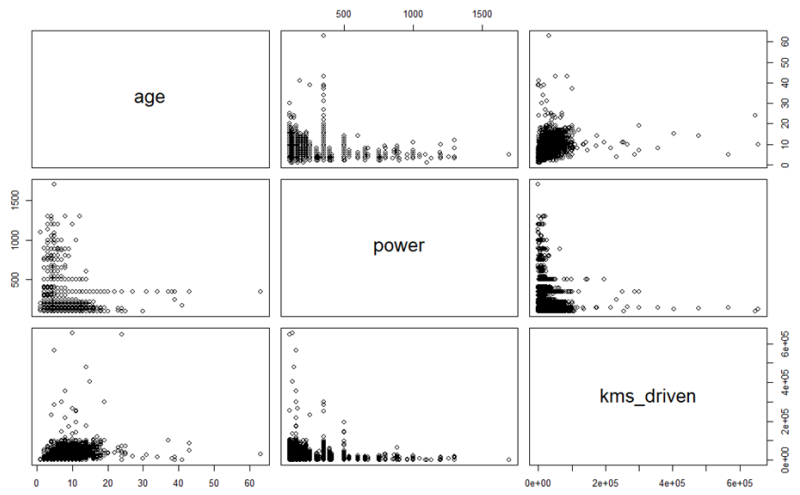


Figure 5.11

6. Important Results of the Advanced Analysis

The predictive models analyzed throughout this analysis are derived with the assumption that the *Price* variable is of continuous data. Accordingly, with regard to the final results of the descriptive analysis mentioned above, fitting a multiple linear regression model is not appropriate due to the presence of associations identified among the predictor variables. Hence, regularization techniques were used to reduce the variance of the coefficient estimates. Table 6.1 below is a model performance evaluation done on the predicted prices and that of actual counts using the Root Mean Squared Error (RMSE) metric with 10- fold cross validation.

Model	Training RMSE	Test RMSE	Training MAPE	Test MAPE
Ridge	29960.91	34176.13	23.31%	25.19%
Lasso	29962.57	34178.73	23.57%	25.43%
Elastic Net	29964.94	38746.6	24.16%	27.81%

Table 6.1

All of the above regularized models indicate approximately similar RMSEs for training data, also it is the same case when it comes to the test RMSEs of ridge and lasso models, while elastic net test RMSE deviates a bit. However, the difference between training and test RMSES of each model suggest overfitting. Obtaining test MAPEs of 25.19%,25.43% and 27.81% for the three models respectively, indicates that the predictions made by the three models are not fair enough. When considering the training MAPE values, the lower prediction accuracy is much clarified as the model performances on the training data are also unsatisfactory. Hence, selecting a model by considering the above three models would not be effective.

In order to improve the model accuracy further, more sophisticated but that can handle multiple issues simultaneously were applied on the data. Table 6.2 below are the results obtained using random forest and XGBoosting using hyper parameter tuning.

	With all the features					With the important features			
Model	**Best Parameters**	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**
Random Forest	mtry=30 splitrule= variance min.node. size=5	9753.169	24785.97	8.53%	15.66%	9236.477	25242.09	8.5%	16.13%
XGBoost	Nrounds =500 max_depth =4 eta=0.1 gamma=0.1 colsample_ bytree=0.6 min_child_ weight=1 subsample= 0.6	11506.04	25009.58	9.06%	15.97%	15463.6	26667.75	9.56%	16.68%

Table 6.2

Note : The number of important features were selected in random forest and xgboost models was 20 in both cases.

As depicted on Table 6.2, the results obtained when all the features were considered, both the techniques suggest that prediction accuracy is low due to overfitting in terms of both RMSEs and MAPE values. Therefore, after extracting the most important features, the techniques were applied again. But, it didn't remedy the overfitting issue .
Figure 6.1 and Figure 6.2 depict the variable importance plots in this scenario regarding the random forest and xgboost models.

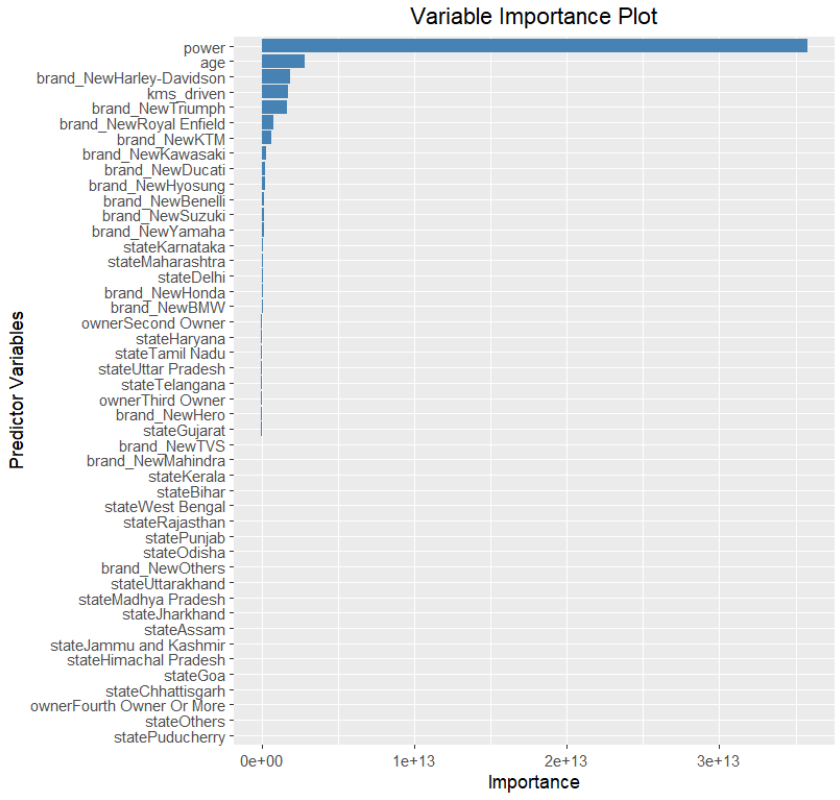


Figure 6.1

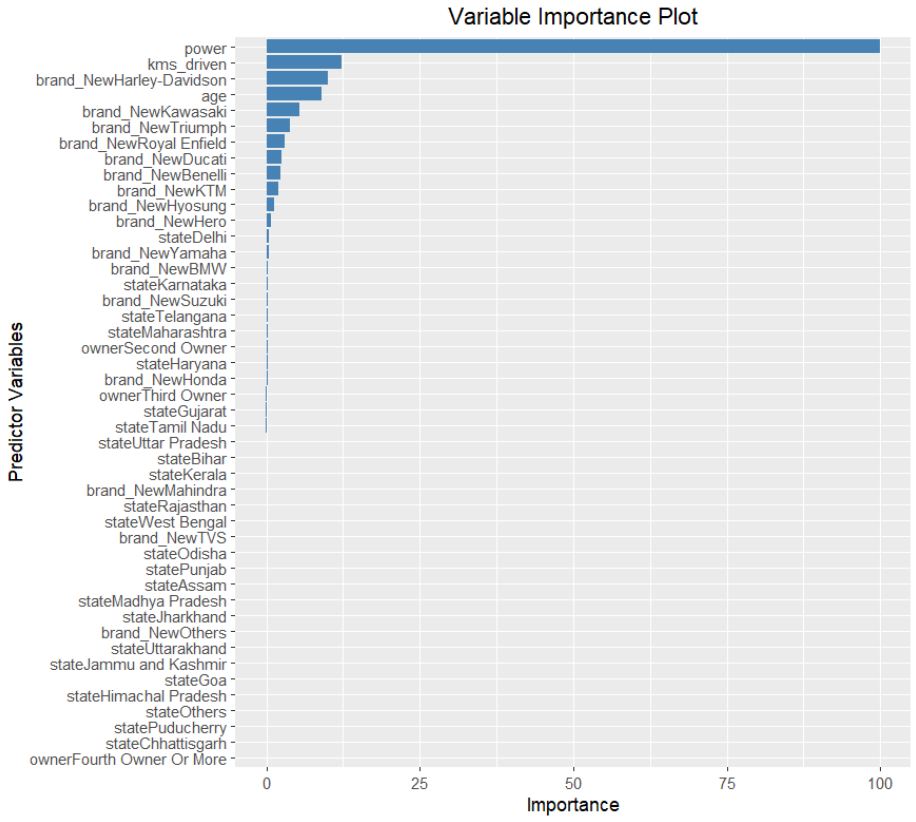


Figure 6.2

The response variable's discreteness and the possession of a positively skewed distribution indicates that using a transformation on the response might produce more accurate predictions. Therefore, **log transformation** is imposed on the response, ***Price*** .

Figure 6.3

	Response – Price	Response – ln(Price)
Skewness	4.769269	0.3629236
Kurtosis	35.56085	3.152006

Table 6.3 below is a model performance evaluation done on the predicted delays after transforming them back to the original scale and that of actual counts using the Root Mean Squared Error (RMSE) metric with 10- fold cross validation.

Model	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**
Ridge	68326.92	69562.18	32.24%	38.39%
Lasso	65687.94	69903.1	32.16%	38.43%
Elastic Net	51363.52	52736.99	31.91%	33.25%

Table 6.3

Above results show that the log transformation has not been successful in fitting linear relationships between the response and predictors with higher RMSEs suggesting the lower prediction accuracy.

Also, we were attentive on how the transformation affects on the relationships between response variable and predictors, where we found that *power* which shared a fairly positive linear relationship has become a non-linear association with the log price. On that note, we decided to proceed with, Random forest regression and XGBoost. Table 6.4 below are the results obtained using random forest and XGBoosting after transforming them back to the original scale.

	With all the features					**With the important features**			
Model	**Best Parameters**	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**
Random Forest	mtry=20 splitrule =variance min.node.size =15	16989.51	26014.15	11.43%	16.44%	15731.24	24042.76	9.63%	15.43%
XGBoost	nrounds=500 max_depth=4 eta=0.1 gamma=0.1 colsample_ bytree=0.5 min_child_ weight=1 subsample=0.6	16928.32	29126.05	10.28%	17.37%	18316.61	32476.92	12.11%	22.2%

Table 6.4

Note : The number of important features were selected in random forest and xgboost models were 11 and 10 respectively.

As depicted on Table 6.4, the results obtained when all the features were considered, both the techniques suggest that prediction accuracy is low due to overfitting. Moreover, extracting the important features also has not been successful in overcoming the overfitting issue. Figure 6.4 and Figure 6.5 depict the variable importance plots in this scenario regarding the random forest and xgboost models.

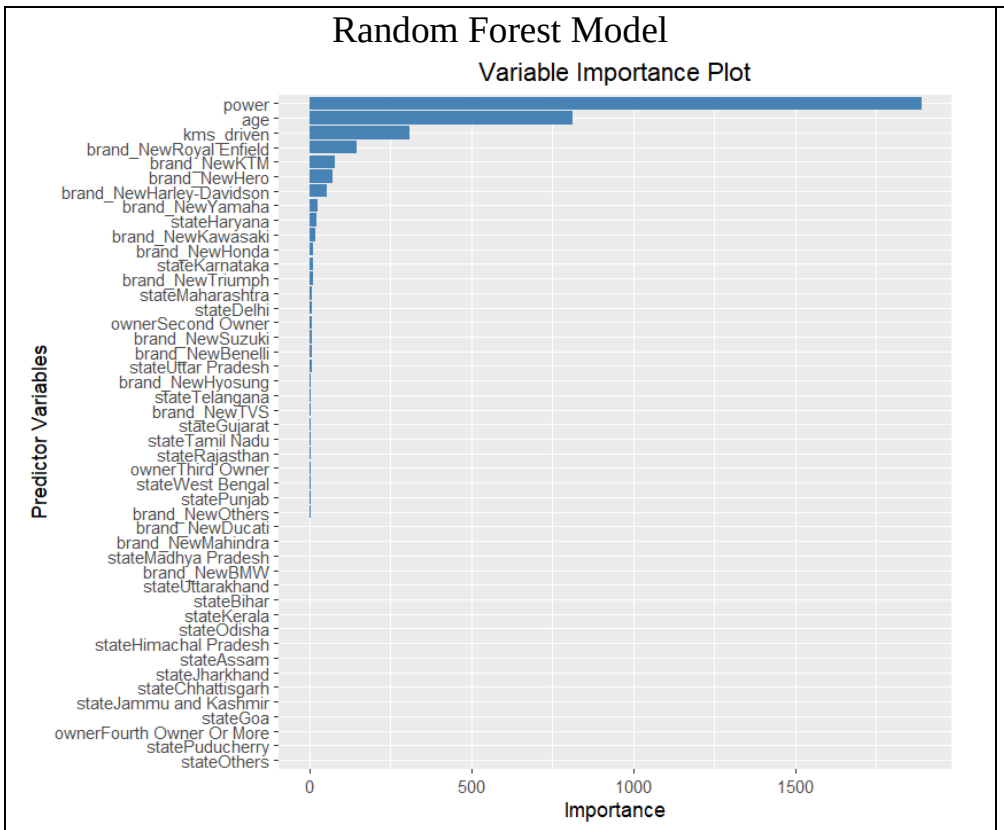


Figure 6.4

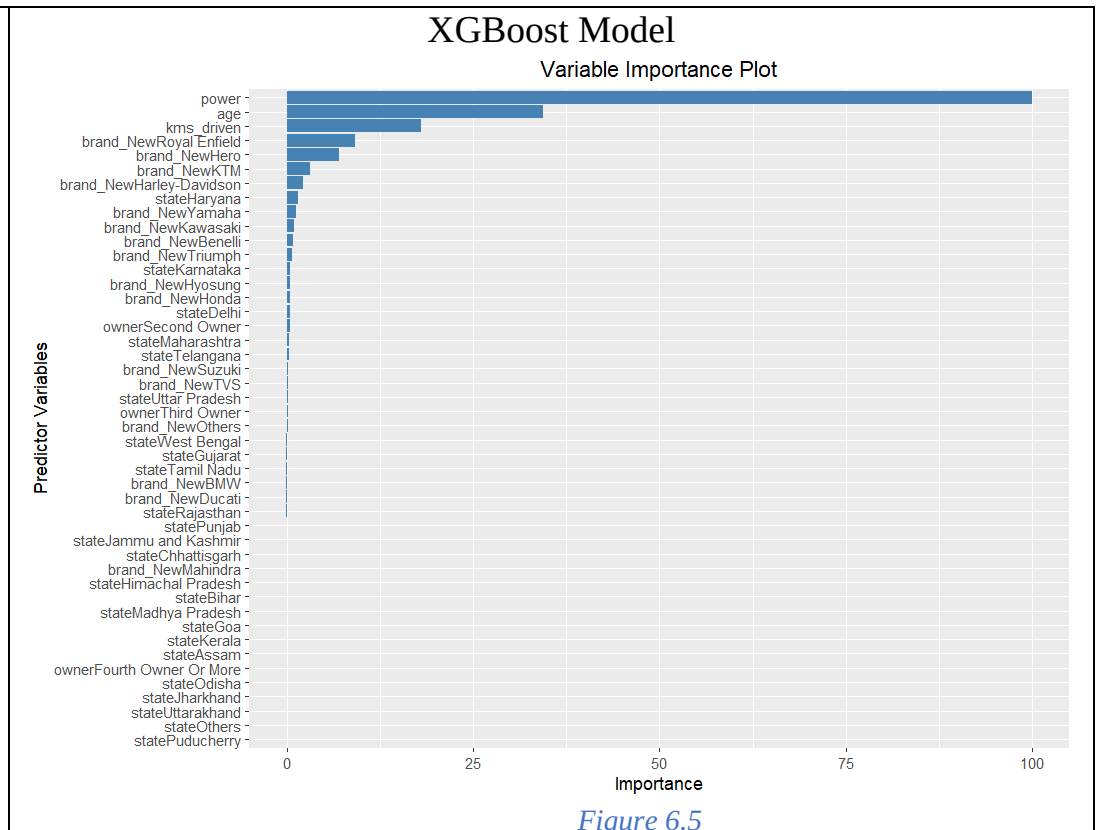


Figure 6.5

With the unsatisfactory results after the log transformation, we decided to implement **square root transformation** on the response variable as the techniques we use in model fitting is less affected by the distribution of the response. Here we found that the associations which the numerical predictors shared with the original response is less affected by the transformation, particularly the linear relationships has remained almost the same.

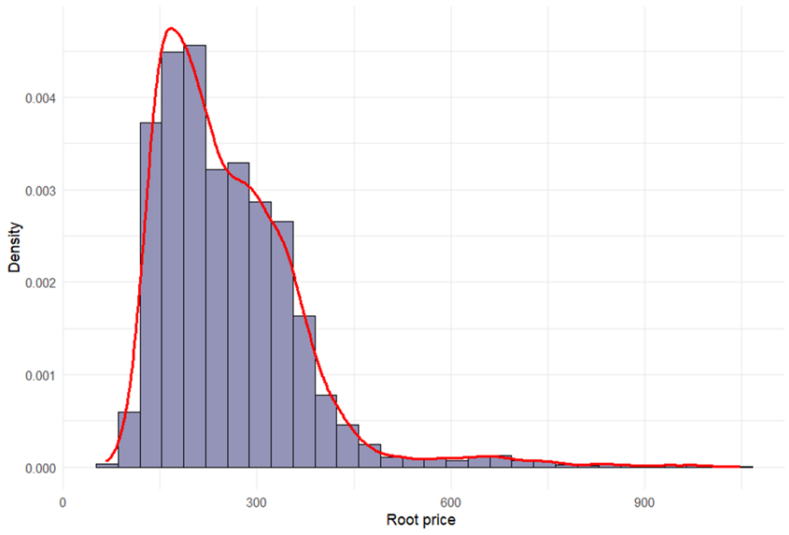


Figure 6.6

	Response – Price	Response – $\sqrt{\text{Price}}$
Skewness	4.769269	2.014945
Kurtosis	35.56085	10.08438

Table 6.5 below are the results obtained using regularization techniques after transforming them back to the original scale.

Model	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**
Ridge	26659.66	27567.9	16.37%	17.103%
Lasso	26646.46	27512.26	16.32%	16.97%
Elastic Net	26684.16	34654.12	16.93%	25.48%

Table 6.5

With reference to the RMSE and MAPE values, the ridge and lasso regression models produce a higher accuracy with less overfitting since they don't show a significant difference between the training RMSE and test RMSE so as the training and test MAPEs. Expecting more accuracy, random forest and xgboost models were fitted on the data with the transformed response. Table 6.6 below are the results obtained using random forest and XGBoosting after transforming them back to the original scale.

Model	**Best Parameters**	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**
Random Forest	mtry=25 splitrule =variance min.node.size =10	12937.58	23464.42	9.25%	14.76%	12340.9	23941.41	9.2%	15.2%
XGBoost	nrounds=500 max_depth=4 eta=0.1 gamma=0.1 colsample_ bytree=0.5 min_child_ weight=1 subsample =0.6	12014.08	25501.58	9.11%	16.39%	16195.55	28084.67	9.82%	17.21%

Table 6.6

Note : The number of important features were selected in random forest and xgboost models were 20 and 11 respectively.

As depicted on Table 6.6, the results obtained when all the features were considered, both the techniques suggest that prediction accuracy is low due to overfitting in terms of both RMSEs and MAPEs. Therefore, after extracting the most important features, the techniques were applied again. But the results show that satisfactory accuracy has not obtained due to overfitting. Figure 6.7 and Figure 6.8 depict the variable importance plots in this scenario regarding the random forest and xgboost models.

Random Forest Model

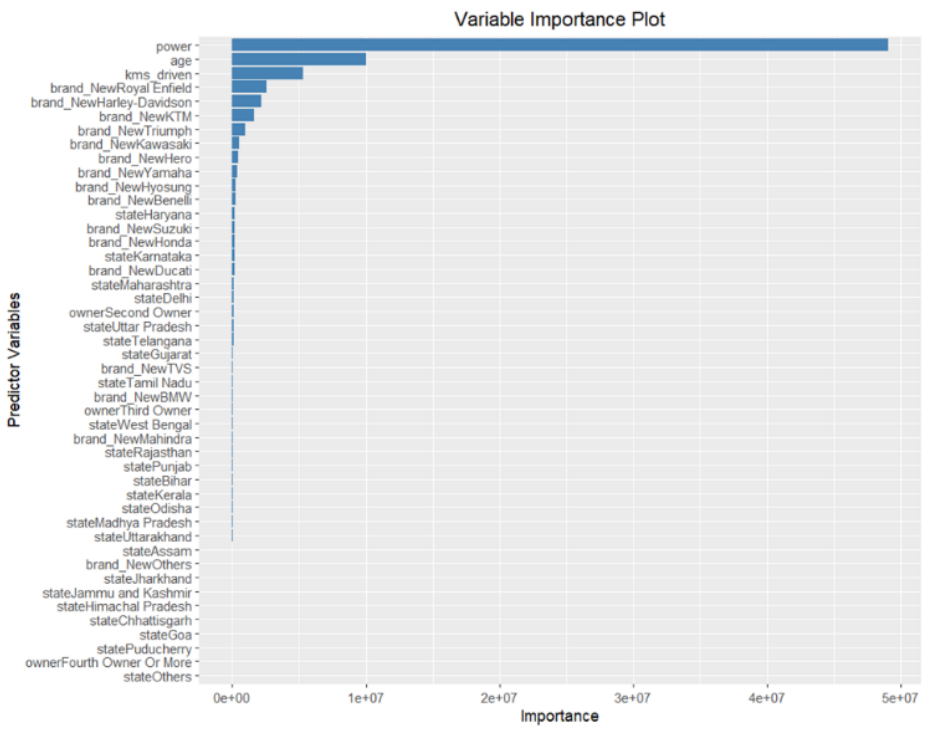


Figure 6.7

XGBoost Model

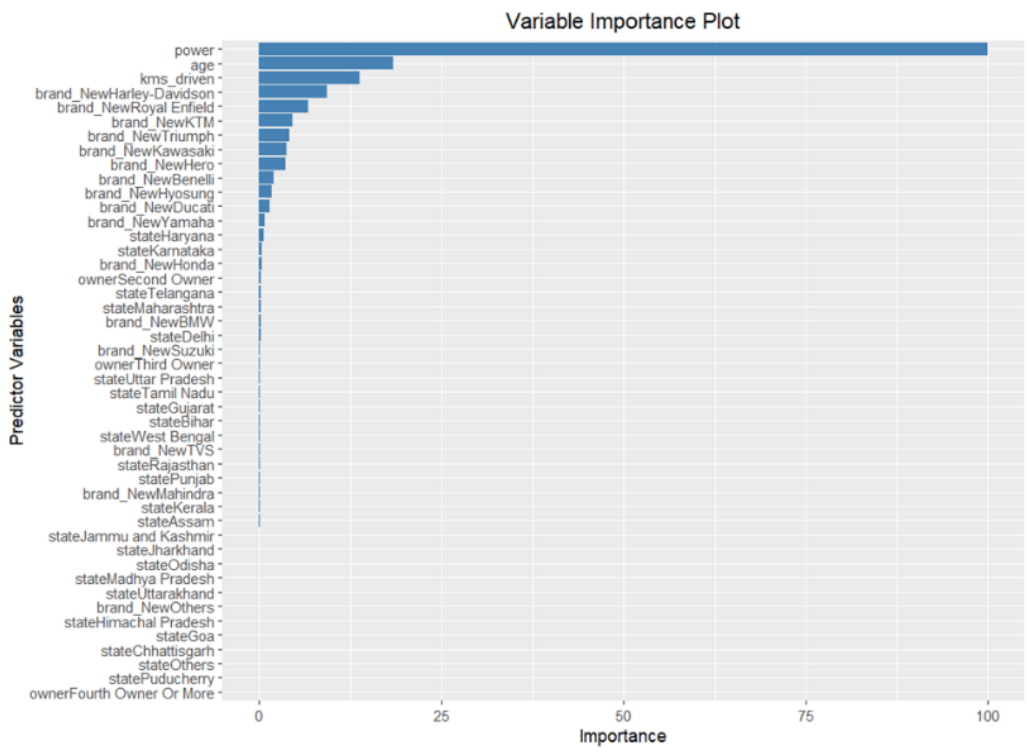


Figure 6.8

The ridge and lasso models with the square root transformed response which were identified to have high prediction accuracy with minimum overfitting were refitted with the 20 most important variables which were identified considering both random forest and xgboost in search of more accuracy. Table 6.7 shows the model evaluation results of the refitted models after transforming them back to original scale.

Model	**Training RMSE**	**Test RMSE**	**Training MAPE**	**Test MAPE**
Ridge	27823.66	29556.44	17.37%	19.62%
Lasso	27342.46	29573.56	17.32%	19.98%

Table 6.7

Even though the results obtained makes no vast difference with the models which were fitted with all the features, the accuracy has reduced in both the models in this case, in terms of both RMSEs and MAPEs.

7. Issues Encountered and Proposed Solutions

- The models fitted on original data implied overfitting and higher value of Test MAPEs. As a solution, transformations were applied on the response but couldn't achieve significant accuracies.
- Presence of a large number of factor levels in a categorical variable would make the model very complicated therefore before fitting the models some factor levels with very low representation were combined together with the intention of reducing the number of factor levels in a categorical variable.
- Random forest and XGBoosting models didn't perform well even after selecting the most important features because it showed significant overfitting. A proper factor analysis could improve these two models.
- The study was only carried out considering the brands of bikes leaving the bike name variable due to the complexity of the models affected by large number of categories Moreover, according to the reports of "kenresearch" website priority of the Indians in buying used bikes is the type of the two wheeler(motor cycle or scooter) more than the brand of the bike which suits their requirements. Otherwise, more accurate values would have been obtained, as bike prices vary with in a brand among various models in wider ranges.
- The dataset which was used for the analysis comprised of categorical variables with many factor levels. In order to fit a model, these variables had to be replaced with dummy variables. As a result, the different factors were separated into many dummy variables, it caused difficulties in interpreting the model since each factor level for the same categorical variable had different levels of importance where some factors had extremely high feature importance while the factors had very low feature importance.

8. Discussion and Conclusions

Since our initial distribution of the response variable was highly skewed to the right, the models which were fitted regarding the regularization techniques Ridge, lasso and Elastic Net resulted considerably higher RMSEs and Test MAPE values. Even though random forest and xgboost models resulted lower RMSE values in both scenarios of considering all the features and considering only the important features, the overfitting issue was clearly existing, which suggested low prediction accuracy which was further proved by the significant differences between their respective training and test MAPE values.
After log transformation of the response, the results obtained for the regularization techniques were still in unacceptable range suggesting a considerable underfitting with larger RMSE values. The random forest and xgboost techniques resulted lower RMSE and test MAPE values with a comparatively less overfitting than when considering the models relevant to the original response.
Thereafter the advanced analysis was carried out on the square root transformed response variable using regularization techniques ridge, lasso and elastic-net. Both ridge and lasso models resulted RMSEs and MAPE values in a justifiable range with comparatively less overfitting among all the regression models applied.
In order to look for further improvement, the attempts of fitting the random forest and xgboost models went in vain due to overfitting issue similar to the previous cases. Despite of the transformation on the original response, all the random forest and xgboost models resulted a higher variable importance for *power, kms_driven* and *age*.
When observing all the variable importance plots, it is evident that there are few dominant variables in predicting the prices . Hence they resulted not much difference in accuracy even after considering the important features comparatively with the model results with all the features considered, as the effect of other variables on accuracy is very low.

When all the models compared, the Lasso Regression model applied on the square root transformed response was identified as the best model which produces the most accurate predictions with training and test RMSEs of 26646.46 and 27512.26 respectively, where the training and test MAPEs were 16.32%, 16.97% respectively which suggests a higher prediction accuracy with less overfitting in this scenario. Even though the results of the ridge model doesn't make much difference with the lasso model in this case, we selected the lasso model as the best model based on interpretability since extracting the most important features in predicting the prices of these used bikes is among our main objectives.

Figure 6.7 and Figure 6.8 depict the coefficients plot related to the selected lasso model and the predicted values Vs observed values of the particular model on the test data respectively.

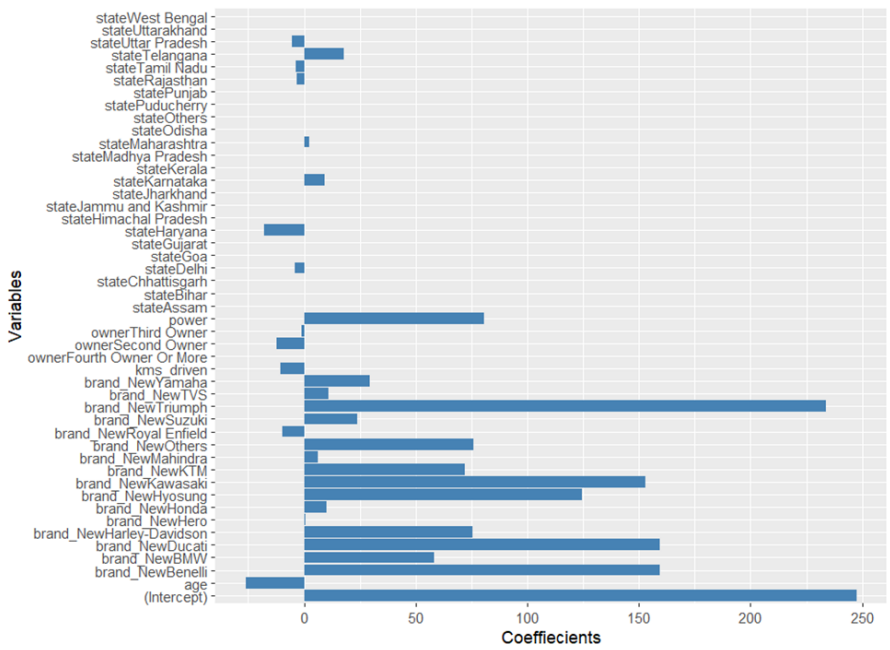


Figure 6.7

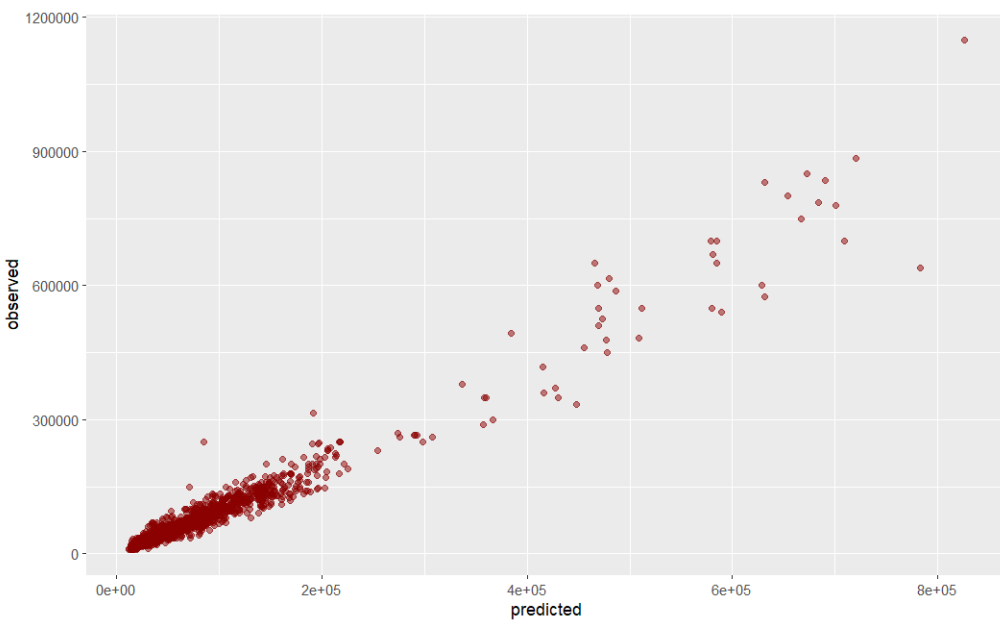


Figure 6.8

9. Appendix of the code

```r
data=read.csv("D:/3RD YEAR/SEMESTER II/ST3082/Project III Materials//Used_Bikes.csv")
View(data)
head(data)
names(data)
summary(data)
str(data)
library(plyr)
library(dplyr)
library(stringr)
library(mgsub)
library(ggplot2)
library(corrplot)
library(psych)
library(packHV)
library(moments)
library(tidyverse)
library(Metrics)


##############################   Pre- processing and Feature Engineering  ###################################

#Removing duplicates
sum(duplicated(data))
data=distinct(data)
nrow(data)
#Replacing "" with NA
data = replace(data, data=='', NA)
colSums(is.na(data))
#Removing records with price=0
data=data[data$price != 0, ]
data=data[data$kms_driven != 0, ]
data=data[data$age != 0, ]
data=data[data$power != 0, ]
nrow(data)

#########################################################################################
#Create brand_New column
table(data$brand)
brand_New=c()
for(i in 1:length(data$brand)){
  if((data$brand[i]=="Ideal")|(data$brand[i]=="Indian")|
     (data$brand[i]=="UM")|(data$brand[i]=="Rajdoot")|
     (data$brand[i]=="Yezdi")|(data$brand[i]=="MV")|(data$brand[i]=="Jawa")){
    brand_New[i]="Others"

  }else{
    brand_New[i]=data$brand[i]
  }

}
table(brand_New)
data=cbind(data,brand_New)



#Create the state column
# read in the second file with unique cities and their corresponding state
df= read.csv("D:/3RD YEAR/SEMESTER II/ST3082/Project III Materials//cities.csv")
sum(duplicated(df))
df=distinct(df)
table(df$State)
nrow(df)


state=c()
for(i in 1:nrow(df)){
  for(j in 1:nrow(data)){
    if(data$City[j]==df$City[i]){
      state[j]=df$State[i]
    }else{
      next
    }
  }
}

for(i in 1:length(state)){
  if((state[i]=="Arunachal Pradesh")|(state[i]=="Meghalaya")|
     (state[i]=="Sikkim")){
    state[i]="Others"
  }else{
    state[i]=state[i]
  }
}
table(state)
data=cbind(data,state)
table(data$owner)

#Factoring
data$owner= factor(data$owner,level=c("First Owner","Second Owner","Third Owner",
                                      "Fourth Owner Or More"))
data$brand_New = factor(data$brand_New,level=c("Bajaj","Benelli","BMW","Ducati","Harley-Davidson","Hero","Honda",
                                               "Hyosung","Kawasaki","KTM","Mahindra","Royal Enfield"
                                               ,"Suzuki","Triumph","TVS","Yamaha","Others"))
data$state = factor(data$state,level=c("Andhra Pradesh","Assam","Bihar","Chhattisgarh",
                                       "Delhi","Goa","Gujarat","Haryana",
                                       "Himachal Pradesh" ,"Jammu and Kashmir","Jharkhand", "Karnataka",
                                       "Kerala",    "Madhya Pradesh",       "Maharashtra",          "Odisha",
                                       "Puducherry" ,          "Punjab" ,          "Rajasthan",
                                       "Tamil Nadu",        "Telangana",     "Uttar Pradesh",       "Uttarakhand",
                                       "West Bengal" , "Others"))


str(data)
##########################################################################################
#Splitting the data in to training and testing sets
set.seed(100)
indexes=sample(1:nrow(data),0.2*nrow(data))
testset=data[indexes,]
trainset=data[-indexes,]
View(trainset)
View(testset)
head(trainset)
head(testset)
nrow(trainset)
nrow(testset)
table(trainset$brand_New)
table(trainset$owner)
table(trainset$state)
table(testset$brand_New)
table(testset$owner)
table(testset$state)
options(repr.plot.width=12,repr.plot.height=7)
hist_boxplot(trainset$price,main="Price distribution",col="#9494b8",xlab="Bike Price(In Indian Rupees)")
boxplot.stats(trainset$price)$stats
x=which(trainset$price %in% boxplot.stats(trainset$price)$out)
length(x)


ggplot(data=trainset,aes(x=age,y=price))+
  geom_point(color = "brown")+

  theme_minimal()+

  labs(x="Age",y="Bike Price(In Indian Rupees)")


ggplot(data=trainset, aes(x=kms_driven, y=price, fill=brand_New, color=brand_New)) +
  geom_point() +
  theme_minimal() +
  labs(x="kilometers driven", y="Bike Price (In Indian Rupees)")




ggplot(data=trainset,aes(x=kms_driven,y=price,fill=owner,color=owner))+
  geom_point()+

  theme_minimal()+

  labs(x="kilometers driven",y="Bike Price(In Indian Rupees)")


tg4= trainset %>%
  group_by(power) %>%
  summarize(mean_price = mean(price))

ggplot(data = trainset, aes(x = power, y = price)) +
  geom_point() +
  geom_smooth(data = tg4, aes(x = power, y = mean_price), method = "lm", color = "red", se = FALSE) +
  theme_minimal() +
  labs(x = "power cc", y = "Bike Price (In Indian Rupees)")


tg5 = trainset %>%
  group_by(state) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

ggplot(tg5, aes(x = reorder(state, -count), y = count, fill = count)) +
  geom_bar(stat = "identity", width = 0.7, alpha = 0.7, color = "black") +
  scale_fill_continuous(low = "lightblue", high = "darkblue",
                        guide = "none", aesthetics = "fill",
                        breaks = seq(0, max(tg5$count), by = 5)) +
  scale_x_discrete(guide = guide_axis(angle = 90)) +
  labs(x = "State", y = "Bike Count") +
  theme(plot.title = element_text(hjust = 0.5))


gm1=as.matrix(gd1)

df1[which(gm1==min(gm1[gm1!=min(gm1)]),arr.ind = TRUE)[1,],]
df1[which(gm1==max(gm1[gm1!=max(gm1)]),arr.ind = TRUE)[1,],]

#Graph to identify the number of clusters
sil_dist=c(NA)
for(i in 2:50){
  pam_fit1=pam(gd1,diss=TRUE,k=i)
  sil_dist[i]=pam_fit1$silinfo$avg.width
}
dev.off()
plot(1:8,sil_dist,
     xlab = "k",
     ylab="Average silhoutte distance",
     lines(1:8,sil_dist))

# Partitioning around medoids with 3 clusters
pam_fit = pam(gd1, diss = TRUE, k = 4)

# Visualizing the clusters
library(Rtsne)
tsne_obj= Rtsne::Rtsne(gd1, is_distance = TRUE)

tsne_data <- tsne_obj$Y %>%
  data.frame() %>%
  setNames(c("X", "Y")) %>%
  mutate(cluster = factor(pam_fit$clustering))

ggplot(aes(x = X, y = Y), data = tsne_data) +
  geom_point(aes(color = cluster))
##########################################################################
#Suggestions for advanced analysis
#Correlation matrix
quantitative= trainset_new %>%
  select(c(age,power,kms_driven))%>%
  as.data.frame()
corr=round(cor(quantitative[,]),2)
library(ggcorrplot)
ggcorrplot(corr,hc.order=F,lab=TRUE)
data5=trainset_new[-c(100:4000),]

quantitative= data5 %>%
```

```r
length(x)

options(repr.plot.width=12,repr.plot.height=7)
hist_boxplot(trainset$price,main="Price distribution",col="#9494b8",xlab="Bike Price(In Indian Rupees)")
boxplot.stats(trainset$price)$stats
x=which(trainset$price %in% boxplot.stats(trainset$price)$out)
length(x)
df1=data %>%  filter(row_number() %in% x)
View(df1)
table(df1$brand_New)
table(df1$owner)
table(df1$state)
trainset=trainset %>%  filter(!row_number() %in% x)
table(trainset$brand_New)
table(trainset$owner)
table(trainset$state)
options(repr.plot.width=12,repr.plot.height=7)
hist_boxplot(log(trainset$price),main="log(Price) distribution",col="#9494b8",xlab="log Bike Price(In Indian Rupees)")
boxplot.stats(log(trainset$price))$stats
x=which(log(trainset$price) %in% boxplot.stats(log(trainset$price))$out)
length(x)

#Mean of Delay time
mean(trainset$price)

#Skewness and kurtosis
skewness(trainset$price)#>1 positively skewed
kurtosis(trainset$price)
skewness(log(trainset$price))#>1 positively skewed
kurtosis(log(trainset$price))


tg2 = ddply(trainset, "brand_New", summarise, price = median(price))

ggplot(tg2, aes(x = brand_New, y = price, group=1)) +
  geom_point(color = "red") +
  geom_line(color = "black") +
  scale_x_discrete(guide = guide_axis(angle = 90)) +
  labs(x = "Brand", y = "Median Bike Price (In Indian Rupees)") +
  theme_minimal()


tg3 = trainset %>%
  group_by(brand_New, owner) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

ggplot(tg3, aes(x = reorder(brand_New, -count), y = count, fill = owner, label = count)) +
  geom_bar(stat = "identity", width = 0.7, alpha = 0.7) +
  scale_x_discrete(guide = guide_axis(angle = 90)) +
  labs( x = "Brand", y = "Bike Count") +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),


######################################## Cluster Check ###########################################################

library(cluster)
library(Rtsne)

#Computing gower distance

df=trainset_new
dummy_coding= dummyVars(" ~ . ", data = df)
df_encoded=predict(dummy_coding, newdata =df)
df_encoded=as.data.frame(df_encoded)
View(df_encoded)
df_encoded[,c(1,2,7,8)]=scale(df_encoded[,c(1,2,7,8)], center = TRUE, scale = TRUE)
df_encoded[,-c(1,2,7,8)] = lapply(df_encoded[,-c(1,2,7,8)], factor)
set.seed(100)
gower_sam1=sample(1:nrow(df),size=1000,replace=FALSE)
df1=df_encoded[gower_sam1,]
gd1=daisy(df1,metric="gower")
gm1=as.matrix(gd1)

df1[which(gm1==min(gm1[gm1!=min(gm1)]),arr.ind = TRUE)[1,],]
df1[which(gm1==max(gm1[gm1!=max(gm1)]),arr.ind = TRUE)[1,],]

#Graph to identify the number of clusters
sil_dist=c(NA)
for(i in 2:50){
  pam_fit1=pam(gd1,diss=TRUE,k=i)
  sil_dist[i]=pam_fit1$silinfo$avg.width
}
dev.off()
plot(1:8,sil_dist,
     xlab = "k",
     ylab="Average silhoutte distance",
     lines(1:8,sil_dist))

# Partitioning around medoids with 3 clusters
pam_fit = pam(gd1, diss = TRUE, k = 4)

# Visualizing the clusters
library(Rtsne)
tsne_obj= Rtsne::Rtsne(gd1, is_distance = TRUE)

#Suggestions for advanced analysis
#Correlation matrix
quantitative= trainset_new %>%
  select(c(age,power,kms_driven))%>%
  as.data.frame()
corr=round(cor(quantitative[,]),2)
library(ggcorrplot)
ggcorrplot(corr,hc.order=F,lab=TRUE)
data5=trainset_new[-c(100:4000),]

quantitative= data5 %>%
  select(c(age,power,kms_driven))%>%
  as.data.frame()
corr=round(cor(quantitative[,]),2)
library(ggcorrplot)
ggcorrplot(corr,hc.order=F,lab=TRUE)
cor_test_result1 = cor.test(trainset_new$kms_driven, trainset_new$age)
cor_test_result2 = cor.test(trainset_new$kms_driven, trainset_new$power)
cor_test_result3 = cor.test(trainset_new$power, trainset_new$age)
# create a scatterplot matrix
pairs(quantitative)
library(ggpubr)
library(rstatix)
ggqqplot(trainset, "age", facet.by = "brand_New")
ggqqplot(trainset, "age", facet.by = "owner")
ggqqplot(trainset, "age", facet.by = "state")
ggqqplot(trainset, "power", facet.by = "brand_New")
ggqqplot(trainset, "power", facet.by = "owner")
ggqqplot(trainset, "power", facet.by = "state")
ggqqplot(trainset, "kms_driven", facet.by = "brand_New")
ggqqplot(trainset, "kms_driven", facet.by = "owner")
ggqqplot(trainset, "kms_driven", facet.by = "state")
ggqqplot(trainset, "price", facet.by = "state")
#Since ANOVA normality assumptions are violated we are using Kruskal wallis test
# Performing Kruskal-Wallis test
result = kruskal.test(price ~ state,
                      data = trainset)
print(result)
```

```r
##################################PLS#########################################
trainset_new=subset(trainset,select=-c(bike_name,city,brand))
testset_new=subset(testset,select=-c(bike_name,city,brand))

library(mdatools)
library(caret)
dummy_coding= dummyVars(" ~ . ", data = trainset_new)
trainset_encoded=predict(dummy_coding, newdata =trainset_new)

View(trainset_encoded)

x=trainset_encoded[,-1]
x[,c(1,6,7)]=scale(x[,c(1,6,7)], center = TRUE, scale = TRUE)
y=as.matrix(trainset_encoded[,1])
set.seed(100)
ModelPLS = pls(x,y,cv=5, info = "Bike Price prediction")
#view summary of model fitting
summary(ModelPLS)

#visualize CV plots
dev.off()
plot(ModelPLS)
plotXScores(ModelPLS,show.label = FALSE)
plotXYLoadings(ModelPLS,show.label = FALSE)
plotVIPScores(ModelPLS,ncomp=19, type = "h",show.label = TRUE)
summary(ModelPLS$coeffs)
plot(ModelPLS$coeffs,show.label = TRUE)
summary(ModelPLS$resCal)
#Checking Outliers
Model0=setDistanceLimits(ModelPLS,lim.type = ModelPLS$lim.type,alpha=0.05)
plotXYResiduals(Model0,show.labels=FALSE,labels="indices")
# Identify outlier indices
# get row indices for outliers in calibration set
outliers = which(categorize(Model0, ModelPLS$resCal) == "outlier")
length(outliers)
df1=trainset[c(outliers),]
View(df1)
table(df1$brand_New)
table(df1$owner)
table(df1$state)
trainset_new=trainset_new %>%  filter(!row_number() %in% outliers)
```

```r
#model_cv = glmnet(x = model_matrix, y= y1, alpha = 0, lambda = best_lambda)

model_cv2 = glmnet(x = model_matrix, y=y1, alpha = 0, lambda = best_lambda2,standardize = FALSE)

y_hat_ridge2 = predict(model_cv2, model_matrix)

library(Metrics)
actual=y1

pred2=c(y_hat_ridge2)

mape_m = mape(y1, y_hat_ridge2)
rmse_m = rmse(y1, y_hat_ridge2)
mape_m
rmse_m



c2=coef(model_cv2 ,s=best_lambda2)
#For the test set
data2=testset_new
data2[,c(2,4,5)]=scale(data2[,c(2,4,5)], center = TRUE, scale = TRUE)
data2[,2]=as.numeric(data2[,2])
data2[,4]=as.numeric(data2[,4])
data2[,5]=as.numeric(data2[,5])

View(data2)
nrow(data2)
model_matrix1 = model.matrix(price~., data = data2)[,-1]

# Separate the response variable from the predictors
#y1 = data1$Min.Delay
y2 = data2$price
model_cv2 = glmnet(x = model_matrix1, y=y2, alpha = 0, lambda = best_lambda2,standardize = FALSE)

y_hat_ridge2 = predict(model_cv2, model_matrix1)


actual=y2
pred2=c(y_hat_ridge2)
mape_m = mape(y2, pred2)
```

```r
plot(lasso_cv)

# Best cross-validated lambda

lambda_cv2 = lasso_cv$lambda.1se

# Fit final model, get its sum of squared residuals and multiple R-squared
#model_cv = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv)
model_cv2 = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv2,standardize = FALSE)

y_hat_lasso2 = predict(model_cv2, model_matrix)

actual=y1
pred2=c(y_hat_lasso2)

mape_m = mape(y1, pred2)
rmse_m = rmse(y1, pred2)
mape_m
rmse_m


#rsq_lasso_cv = cor(y, y_hat_cv)^2
coef(model_cv1 ,s=lambda_cv1)
coef(model_cv2 ,s=lambda_cv2)

table(trainset_new$brand_New)
# See how increasing lambda shrinks the coefficients --------------------------

res = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq,standardize = FALSE)
plot(res, xvar = "lambda")
#legend("bottomright", lwd = 1, col = 1:6, legend = colnames(model_matrix), cex = .7)


#For the test set


model_cv2 = glmnet(x = model_matrix1, y=y2, alpha = 1, lambda = lambda_cv2,standardize = FALSE)


y_hat_lasso2 = predict(model_cv2, model_matrix1)

actual=y2
```

```r
# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data2[,-1])

actual=y2
pred=c(y_hat_enet )
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2, pred)
mape_m



#Random Forest

# Define the tuning grid
tune_grid = expand.grid(
  mtry =c(15,20,25,30),
  splitrule = "variance",
  min.node.size = c(1, 5, 10,15)

)
# Tune the hyperparameters using the tune() function
library(ranger)


# Perform 5-fold cross-validation
set.seed(100)
rf_model = train(
  price~ .,
  data = data1,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
  tuneGrid=tune_grid,
  importance="impurity"
)
# Get best hyperparameters
best_params = rf_model$bestTune
plot(rf_model)
#For the training set
y_hat_enet = predict(rf_model, data1[,-1])
actual=y1
pred=c(y_hat_enet)
```

trainControl(method = "boot", number = ifelse(grepl("cv", method), 10, 25), repeats = ifelse(grepl("[d_]cv$", method), 1, NA), p = 0.75, search = "grid", initialWindow = NULL, horizon = 1, fixedWindow = TRUE, <...truncated...>)

```r

df=var_imp[order(-var_imp$Importance),]
View(df)


nrow(df)

# Sort variable importances
var_imp <- var_imp[order(var_imp$importance), ]

# Calculate number of variables to be removed

vars_to_remove = df$Variables[33:46]

# Remove least important variables from data1

dummy_data1= dummyVars(" ~ . ", data = data1,sep = "")
data1_encoded=predict(dummy_data1, newdata =data1)

View(data1_encoded)
data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
View(data1_imp)
colnames(data1_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data1_imp = data1_imp[, !(colnames(data1_imp) %in% base_var)]
View(data1_imp)
str(data1_imp)
data1_imp=as.data.frame(data1_imp)
# Convert dummy variables to factors
num_cols = c("price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data1_imp), num_cols)
data1_imp[factor_cols] = apply(data1_imp[factor_cols], 2, function(x) as.factor(x))

# Check the data types
str(data1_imp)

# Define the tuning grid for random forest hyperparameters
tune_grid = expand.grid(
  mtry =32,
  splitrule = "variance",
  min.node.size = c(1, 5, 10,15)
```

```r
# Replace 'c1' with the actual coefficients obtained from your model
c1 = as.matrix(coef(model_cv1, s = best_lambda1))

# Get the column names of the original dataset used in the model
variable_names <- colnames(model_cv1$df)

# Convert the coefficients into a data frame for plotting, with variable names assigned
c1_df <- data.frame(variable = variable_names, coefficient = c1)

# Create the plot using ggplot
ggplot(data = c1_df, aes(x = variable, y = coefficient)) +
  geom_bar(stat = "identity", fill = "steelblue", width = 0.5) +
  labs(title = "Coefficients of Model", x = "Variable", y = "Coefficient") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))



#Lasso Regression
# Perform 10-fold cross-validation to select lambda ----------------------------
# Setting alpha = 1 implements lasso regression
#lasso_cv = cv.glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq, nfolds = 10)
set.seed(100)
lasso_cv = cv.glmnet(x = model_matrix, y=y1, alpha = 1, lambda = lambda_seq, nfolds = 10,standardize=FALSE)
# Plot cross-validation results
dev.off()
plot(lasso_cv)

# Best cross-validated lambda

lambda_cv2 = lasso_cv$lambda.1se

# Fit final model, get its sum of squared residuals and multiple R-squared
```

```r
actual=y2
pred2=c(y_hat_lasso2)

mape_m = mape(y2, pred2)
rmse_m = rmse(y2, pred2)
mape_m
rmse_m

#Elastic Net Regression
# Set training control
set.seed(100)

train_control = trainControl(method = "repeatedcv",
                             number = 10,
                             repeats = 5,
                             search = "random",
                             verboseIter = TRUE)

elastic_net_model = train(price ~ .,
                          data = data1,
                          method = "glmnet",
                          trControl = train_control)


# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data1[,-1])
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m
mape_m = mape(y1, pred)
mape_m
#For the test set


# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data2[,-1])

actual=y2
pred=c(y_hat_enet )
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2, pred)
mape_m
```

```r
#For the training set
y_hat_enet = predict(rf_model, data1[,-1])
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m
mape_m = mape(y1, pred)
mape_m


#For the test set

y_hat_enet = predict(rf_model, data2[,-1])

actual=y2
pred=c(y_hat_enet )
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2, pred)
mape_m


# get variable importance
# Print the variable importance measures
var_imp=rf_model$finalModel$variable.importance
# print the variable importance
print(var_imp)

# load ggplot2 library
library(ggplot2)


var_imp = data.frame(Variables = names(rf_model$finalModel$variable.importance),
                     Importance = rf_model$finalModel$variable.importance,
                     row.names = NULL)

ggplot(data = var_imp, aes(x = reorder(Variables, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
  theme(plot.title = element_text(hjust = 0.5))+
  coord_flip()
```

```r
  min.node.size = c(1, 5, 10,15)

)
set.seed(100)
rf_model = train(
  price~ .,
  data = data1_imp,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
  tuneGrid=tune_grid,
  importance="impurity"
)

# Make predictions

#For the training set
y_hat_enet = predict(rf_model, data1_imp[,-1])
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m
mape_m=mape(y1,pred)
mape_m

#For the test set
# Calculate number of variables to be removed

vars_to_remove = df$Variables[33:46]

# Remove least important variables from data1

dummy_data2= dummyVars(" ~ . ", data = data2,sep = "")
data2_encoded=predict(dummy_data2, newdata =data2)

View(data2_encoded)
data2_imp = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
View(data2_imp)
colnames(data2_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
View(data2_imp)
```

```
data2_imp=as.data.frame(data2_imp)
# Convert dummy variables to factors
num_cols = c("price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data2_imp), num_cols)
data2_imp[factor_cols] = apply(data2_imp[factor_cols], 2, function(x) as.factor(x))

y_hat_enet = predict(rf_model, data2_imp[,-1])

actual=y2
pred=c(y_hat_enet )
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2, pred)
mape_m


################################################################################
library(xgboost)
set.seed(100)
Xtrain = xgb.DMatrix(data = as.matrix(data1[,-1]),label=y1)
y_train = data1$price
Xtest = xgb.DMatrix(data = as.matrix(data2[,-1]),label=y2)
y_test = data2$price
xgb_model = train(Xtrain.,
                  y_train,
                  method = "xgbTree",
                  objective = "reg:squarederror",
                  trControl = trainControl(method = "cv",
                                           number = 5,

                                           verboseIter = TRUE),


                  tuneGrid = expand.grid(nrounds = c(500,1000),
                                         eta=0.1,
                                         max_depth = c(2,4,6),
                                         colsample_bytree = c(0.5,0.6),
                                         subsample = c(0.5,0.6),
                                         gamma=0.1,
                                         min_child_weight = c(1,3,5)
                  ))

plot(xgb_model)
y_hat_enet = predict(xgb_model, Xtrain)
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m


#For the test set

y_hat_enet = predict(xgb_model, Xtest)

actual=y2
pred=c(y_hat_enet )
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2,pred)
mape_m
```

```
# get variable importance
var_imp =varImp(xgb_model)

# print the variable importance
print(var_imp)

# load ggplot2 library
library(ggplot2)


# plot the variable importance
ggplot(data = var_imp, aes(x = reorder(rownames(var_imp), Overall), y = Overall)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
  theme(plot.title = element_text(hjust = 0.5))




df=as.data.frame(var_imp$importance)
nrow(df)
Variables=rownames(df)
df=cbind(df,Variables)

# Calculate number of variables to be removed

vars_to_remove = df$Variables[21:46]

# Remove least important variables from data1

dummy_data1= dummyVars(" ~ . ", data = data1,sep = "")
data1_encoded=predict(dummy_data1, newdata =data1)

View(data1_encoded)
data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
View(data1_imp)
colnames(data1_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data1_imp = data1_imp[, !(colnames(data1_imp) %in% base_var)]
View(data1_imp)
```

```
str(data1_imp)
Xtrain = xgb.DMatrix(data = as.matrix(data1_imp[,-1]),label=y1)
y_train = data1$price

set.seed(100)
xgb_model = train(Xtrain,
                  y_train,
                  method = "xgbTree",
                  objective = "reg:squarederror",
                  trControl = trainControl(method = "cv",
                                           number = 5,

                                           verboseIter = TRUE),


                  tuneGrid = expand.grid(nrounds = c(500,1000),
                                         eta=0.1,
                                         max_depth = c(2,4,6),
                                         colsample_bytree = c(0.5,0.6),
                                         subsample = c(0.5,0.6),
                                         gamma=0.1,
                                         min_child_weight = c(1,3,5)
                  ))

plot(xgb_model)
y_hat_enet = predict(xgb_model, Xtrain)
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m
```

```
#For the test set
# Remove least important variables from data2

dummy_data2= dummyVars(" ~ . ", data = data2,sep = "")
data2_encoded=predict(dummy_data2, newdata =data2)

View(data2_encoded)
data2_imp = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
View(data2_imp)
colnames(data2_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
View(data2_imp)
str(data2_imp)
data2_imp=as.data.frame(data2_imp)
# Convert dummy variables to factors
num_cols = c("price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data2_imp), num_cols)
data2_imp[factor_cols] = apply(data2_imp[factor_cols], 2, function(x) as.factor(x))

str(data2_imp)
Xtest = xgb.DMatrix(data = as.matrix(data2_imp[,-1]),label=y2)
y_test = data2$price
y_hat_enet = predict(xgb_model, Xtest)

actual=y2
pred=c(y_hat_enet )
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2,pred)
mape_m


################################################################################
nrow(trainset_new)
options(repr.plot.width=12,repr.plot.height=7)
hist_boxplot(trainset_new$price,main="Price distribution",col="#9494b8",xlab="Price")
boxplot.stats(trainset_new$price)$stats
x=which(trainset_new$price %in% boxplot.stats(trainset_new$price)$out)
length(x)
```

```
options(repr.plot.width=12,repr.plot.height=7)
hist_boxplot(log(trainset_new$price),main="Log Price distribution",col="#9494b8",xlab="Log price")
boxplot.stats(log(trainset_new$price))$stats
x=which(log(trainset_new$price) %in% boxplot.stats(log(trainset_new$price))$out)
length(x)

#Skewness and kurtosis
skewness(trainset_new$price)#>1 positively skewed
kurtosis(trainset_new$price)
skewness(log(trainset_new$price))#>1 positively skewed
kurtosis(log(trainset_new$price))


log_price=c(log(trainset_new$price))
trainset_new=cbind(log_price,trainset_new)
#trainset=trainset[,-2]
log_price=c(log(testset_new$price))
testset_new=cbind(log_price,testset_new)




################################################################################
#Creating the model matrix
View(trainset_new)
View(testset_new)
data1=trainset_new[,-2]
data2=testset_new[,-2]
```

```
data1[,c(2,4,5)]=scale(data1[,c(2,4,5)], center = TRUE, scale = TRUE)
data1[,2]=as.numeric(data1[,2])
data1[,4]=as.numeric(data1[,4])
data1[,5]=as.numeric(data1[,5])

View(data1)


# Create model matrix with categorical variables as dummies and numerical variable as is
#model_matrix = model.matrix(formula, data = data1)
model_matrix = model.matrix(log_price~., data = data1)[,-1]
#model_matrix2 = model.matrix(log_delay~., data = data1)[,-1]

#model_matrix = model.matrix(Min.Delay~., data = df)

# Separate the response variable from the predictors
y1 = data1$log_price
#y1 = df$Min.Delay
is.matrix(model_matrix)
dim(model_matrix)
# Create a sequence of lambda values to test
lambda_seq= 10^seq(-3, 10, length.out = 100)
library(glmnet)
# Use cv.glmnet to perform cross-validation and select the optimal lambda value
#cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq)
set.seed(100)
cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq,nfolds=10,standardize=FALSE)

# View the optimal lambda value

best_lambda2=cv_fit$lambda.1se
# Plot the cross-validation results
dev.off()
plot(cv_fit)

# Fit final model, get its sum of squared residuals and multiple R-squared
#Predictors are already scaled ----> No standardizing here
model_cv2 = glmnet(x = model_matrix, y=y1, alpha = 0, lambda = best_lambda2,standardize = FALSE)

y_hat_ridge2 = predict(model_cv2, model_matrix)
```

```
actual=y1
pred1=c(y_hat_ridge1)
pred2=c(y_hat_ridge2)
d_actual=trainset_new$price
d_pred1=as.integer(exp(pred1))
d_pred2=as.integer(exp(pred2))

rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m

coef(model_cv1 ,s=best_lambda1)
coef(model_cv2 ,s=best_lambda2)
max(d_actual-d_pred1)
max(d_actual)
max(d_pred1)

#For the test set


data2[,c(2,4,5)]=scale(data2[,c(2,4,5)], center = TRUE, scale = TRUE)
data2[,2]=as.numeric(data2[,2])
data2[,4]=as.numeric(data2[,4])
data2[,5]=as.numeric(data2[,5])

View(data2)
nrow(data2)
```

```
data2[,5]=as.numeric(data2[,5])

View(data2)
nrow(data2)
#model_matrix1 = model.matrix(log_delay~., data = data2)
model_matrix1 = model.matrix(log_price~., data = data2)[,-1]

# Separate the response variable from the predictors
#y1 = data1$Min.Delay
y2 = data2$log_price
model_cv1= glmnet(x = model_matrix1, y=y2, alpha = 0, lambda = best_lambda1,standardize = FALSE)
model_cv2 = glmnet(x = model_matrix1, y=y2, alpha = 0, lambda = best_lambda2,standardize = FALSE)

y_hat_ridge1 = predict(model_cv1, model_matrix1)
y_hat_ridge2 = predict(model_cv2, model_matrix1)

actual=y2
pred1=c(y_hat_ridge1)
pred2=c(y_hat_ridge2)

d_actual=testset_new$price
d_pred1=as.integer(exp(pred1))
d_pred2=as.integer(exp(pred2))
d_actual=testset_new$price
d_pred1=as.integer(exp(pred1))
d_pred2=as.integer(exp(pred2))

rmse_m = rmse(d_actual, d_pred1)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred2)
mape_m


#Lasso Regression
# Perform 10-fold cross-validation to select lambda ----------------------------
# Setting alpha = 1 implements lasso regression

#lasso_cv = cv.glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq, nfolds = 10)
```

```
#lasso_cv = cv.glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq, nfolds = 10)
set.seed(100)
lasso_cv = cv.glmnet(x = model_matrix, y=y1, alpha = 1, lambda = lambda_seq, nfolds = 10,standardize=FALSE)
# Plot cross-validation results
dev.off()
plot(lasso_cv)

# Best cross-validated lambda
lambda_cv2 = lasso_cv$lambda.1se

# Fit final model, get its sum of squared residuals and multiple R-squared
#model_cv = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv)
model_cv2 = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv2,standardize = FALSE)

y_hat_lasso2 = predict(model_cv2, model_matrix)
actual=y1
pred1=c(y_hat_lasso1)
pred2=c(y_hat_lasso2)

d_actual=trainset_new$price
d_pred1=as.integer(exp(pred1))
d_pred2=as.integer(exp(pred2))
rmse_m = rmse(d_actual, d_pred1)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred2)
mape_m

#rsq_lasso_cv = cor(y, y_hat_cv)^2
coef(model_cv1 ,s=lambda_cv1)
coef(model_cv2 ,s=lambda_cv2)
# See how increasing lambda shrinks the coefficients ----------------------------
# Each line shows coefficients for one variables, for different lambdas.
# The higher the lambda, the more the coefficients are shrinked towards zero.
res = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq,standardize = FALSE)
plot(res, xvar = "lambda",label=TRUE,lw=2)
#legend("bottomright", lwd = 1, col = 1:6, legend = colnames(model_matrix), cex = .7)


#For the test set


#For the test set

model_cv2 = glmnet(x = model_matrix1, y=y2, alpha = 1, lambda = lambda_cv2,standardize = FALSE)

y_hat_lasso2 = predict(model_cv2, model_matrix1)

actual=y2
```

```
y_hat_lasso2 = predict(model_cv2, model_matrix1)

actual=y2
pred1=c(y_hat_lasso1)
pred2=c(y_hat_lasso2)

d_actual=testset_new$price
d_pred1=as.integer(exp(pred1))
d_pred2=as.integer(exp(pred2))
rmse_m = rmse(d_actual, d_pred1)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred2)
mape_m


#Elastic Net Regression
# Set training control
set.seed(100)

train_control = trainControl(method = "repeatedcv",
                             number = 10,
                             repeats = 5,
                             search = "random",
                             verboseIter = TRUE)
elastic_net_model = train(log_price ~ .,
                          data = data1,
                          method = "glmnet",
                          trControl = train_control)

# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data1[,-1])
actual=y1
pred=c(y_hat_enet)
d_actual=trainset_new$price
d_pred=as.integer(exp(pred))
rmse_m = rmse(d_actual, d_pred)
rmse_m
mape_m = mape(d_actual, d_pred)
```

```
mape_m = mape(d_actual, d_pred)
mape_m
#For the test set
#elastic_net_model1 = train(log_delay ~ .,data = data2,method = "glmnet",trControl = train_control)

# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data2[,-1])

actual=y2
pred=c(y_hat_enet )
d_actual=testset_new$price
d_pred=as.integer(exp(pred))
rmse_m = rmse(d_actual, d_pred)
rmse_m
mape_m = mape(d_actual, d_pred)
mape_m

#Random Forest

# Define the tuning grid
tune_grid = expand.grid(
  mtry =c(15,20,25,30),
  splitrule = "variance",
  min.node.size = c(1, 5, 10,15)

)
# Tune the hyperparameters using the tune() function
library(ranger)


# Perform 5-fold cross-validation
set.seed(100)
rf_model = train(
  log_price ~ .,
  data = data1,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
  tuneGrid=tune_grid,
  importance="impurity"
)
# Get best hyperparameters
best_params = rf_model$bestTune
```

```
plot(rf_model)
#For the training set
y_hat_enet = predict(rf_model, data1[,-1])
actual=y1
pred=c(y_hat_enet)
d_actual=trainset_new$price
d_pred=as.integer(exp(pred))
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m

#For the test set

y_hat_enet = predict(rf_model, data2[,-1])

actual=y2
pred=c(y_hat_enet )
d_actual=testset_new$price
d_pred=as.integer(exp(pred))
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m


# get variable importance
# Print the variable importance measures
var_imp=rf_model$finalModel$variable.importance
# print the variable importance
print(var_imp)

# load ggplot2 library
library(ggplot2)
```

```

var_imp = data.frame(Variables = names(rf_model$finalModel$variable.importance),
                     Importance = rf_model$finalModel$variable.importance,
                     row.names = NULL)

ggplot(data = var_imp, aes(x = reorder(Variables, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
  theme(plot.title = element_text(hjust = 0.5))+
  coord_flip()


df=var_imp[order(-var_imp$Importance),]
View(df)
df[c(1:11),]

nrow(df)


# Calculate number of variables to be removed

vars_to_remove = df$Variables[12:46]

# Remove least important variables from data1

dummy_data1= dummyVars(" ~ . ", data = data1,sep = "")
data1_encoded=predict(dummy_data1, newdata =data1)

View(data1_encoded)
data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
View(data1_imp)
colnames(data1_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data1_imp = data1_imp[, !(colnames(data1_imp) %in% base_var)]
View(data1_imp)
str(data1_imp)
data1_imp=as.data.frame(data1_imp)
# Convert dummy variables to factors
num_cols = c("log_price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data1_imp), num_cols)
data1_imp[factor_cols] = apply(data1_imp[factor_cols], 2, function(x) as.factor(x))

# Check the data types
str(data1_imp)

# Define the tuning grid for random forest hyperparameters
tune_grid = expand.grid(
```

```
str(data1_imp)

# Define the tuning grid for random forest hyperparameters
tune_grid = expand.grid(
  mtry =11,
  splitrule = "variance",
  min.node.size = c(1, 5, 10,15)

)
set.seed(100)
rf_model = train(
  log_price~ .,
  data = data1_imp,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
  tuneGrid=tune_grid,
  importance="impurity"
)

# Make predictions

#For the training set
y_hat_enet = predict(rf_model, data1_imp[,-1])
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m
d_actual=trainset_new$price
d_pred=as.integer(exp(pred))
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m

#For the test set
# Calculate number of variables to be removed


# Remove least important variables from data1

dummy_data2= dummyVars(" ~ . ", data = data2,sep = "")
data2_encoded=predict(dummy_data2, newdata =data2)
```

```
data2_encoded=predict(dummy_data2, newdata =data2)

View(data2_encoded)
data2_imp = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
View(data2_imp)
colnames(data2_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
View(data2_imp)

data2_imp=as.data.frame(data2_imp)
# Convert dummy variables to factors
num_cols = c("log_price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data2_imp), num_cols)
data2_imp[factor_cols] = apply(data2_imp[factor_cols], 2, function(x) as.factor(x))

y_hat_enet = predict(rf_model, data2_imp[,-1])

actual=y2
pred=c(y_hat_enet )
d_actual=testset_new$price
d_pred=as.integer(exp(pred))
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2, pred)
mape_m
```

```
set.seed(100)
Xtrain = xgb.DMatrix(data = as.matrix(data1[,-1]),label=y1)
y_train = data1$log_price
Xtest = xgb.DMatrix(data = as.matrix(data2[,-1]),label=y2)
y_test = data2$log_price
xgb_model = train(Xtrain,
                  y_train,
                  method = "xgbTree",
                  objective = "reg:squarederror",
                  trControl = trainControl(method = "cv",
                                           number = 5,

                                           verboseIter = TRUE),


                  tuneGrid = expand.grid(nrounds = c(500,1000),
                                         eta=0.1,
                                         max_depth = c(2,4,6),
                                         colsample_bytree = c(0.5,0.6),
                                         subsample = c(0.5,0.6),
                                         gamma=0.1,
                                         min_child_weight = c(1,3,5)
                  ))

plot(xgb_model)
y_hat_enet = predict(xgb_model, Xtrain)
actual=y1
pred=c(y_hat_enet)
d_actual=trainset_new$price
d_pred=as.integer(exp(pred))
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
#for the test set

y_hat_enet = predict(xgb_model, Xtest)
```

```
rmse_m

# get variable importance
var_imp =varImp(xgb_model)

# print the variable importance
print(var_imp)

# load ggplot2 library
library(ggplot2)


# plot the variable importance
ggplot(data = var_imp, aes(x = reorder(rownames(var_imp), Overall), y = Overall)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
  theme(plot.title = element_text(hjust = 0.5))



df=as.data.frame(var_imp$importance)
nrow(df)
Variables=rownames(df)
df=cbind(df,Variables)

# Calculate number of variables to be removed

vars_to_remove = df$Variables[11:46]

# Remove least important variables from data1

dummy_data1= dummyVars(" ~ . ", data = data1,sep = "")
data1_encoded=predict(dummy_data1, newdata =data1)

View(data1_encoded)
data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
View(data1_imp)
colnames(data1_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data1_imp = data1_imp[, !(colnames(data1_imp) %in% base_var)]
View(data1_imp)
```

```
str(data1_imp)
Xtrain = xgb.DMatrix(data = as.matrix(data1_imp[,-1]),label=y1)
y_train = data1_imp$log_price

set.seed(100)
xgb_model = train(Xtrain,
                  y_train,
                  method = "xgbTree",
                  objective = "reg:squarederror",
                  trControl = trainControl(method = "cv",
                                           number = 5,

                                           verboseIter = TRUE),


                  tuneGrid = expand.grid(nrounds = c(500,1000),
                                         eta=0.1,
                                         max_depth = c(2,4,6),
                                         colsample_bytree = c(0.5,0.6),
                                         subsample = c(0.5,0.6),
                                         gamma=0.1,
                                         min_child_weight = c(1,3,5)
                  ))

plot(xgb_model)
y_hat_enet = predict(xgb_model, Xtrain)
actual=y1
pred=c(y_hat_enet)
d_actual=trainset_new$price
d_pred=as.integer(exp(pred))
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
```

```
str(data2_imp)
Xtest = xgb.DMatrix(data = as.matrix(data2_imp[,-1]),label=y2)
y_test = data2_imp$log_price
y_hat_enet = predict(xgb_model,Xtest)

actual=y2
pred=c(y_hat_enet )
d_actual=testset_new$price
d_pred=as.integer(exp(pred))
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2,pred)
mape_m
```

```

##############################################################################################
#Skewness and kurtosis
skewness(trainset_new$price)#>1 positively skewed
kurtosis(trainset_new$price)
skewness(sqrt(trainset_new$price))#>1 positively skewed
kurtosis(sqrt(trainset_new$price))

# Create a ggplot object
ggplot(trainset_new, aes(x = sqrt(price))) +

  # Add histogram layer
  geom_histogram( aes(y = ..density..), fill = "#9494b8", color = "black") +

  # Add density curve layer
  geom_density(color = "red", linetype = "solid", size = 1) +

  # Add x-axis label
  xlab("Root price ") +

  # Add y-axis label
  ylab("Density") +

  # Set theme
  theme_minimal()

trainset_new=trainset_new[,-1]
testset_new=testset_new[,-1]

sq_price=c(sqrt(trainset_new$price))
trainset_new=cbind(sq_price,trainset_new)
#trainset=trainset[,-2]
sq_price=c(sqrt(testset_new$price))
testset_new=cbind(sq_price,testset_new)


################################Square root transformation#####################################################
#Creating the model matrix
View(trainset_new)
```

```
##############################Square root transformation#########################################################
#Creating the model matrix
View(trainset_new)
View(testset_new)
data1=trainset_new[,-2]
data1[,c(2,4,5)]=scale(data1[,c(2,4,5)], center = TRUE, scale = TRUE)
data1[,2]=as.numeric(data1[,2])
data1[,4]=as.numeric(data1[,4])
data1[,5]=as.numeric(data1[,5])

View(data1)


# Create model matrix with categorical variables as dummies and numerical variable as is
#model_matrix = model.matrix(formula, data = data1)
model_matrix = model.matrix(sq_price~., data = data1)[,-1]
#model_matrix2 = model.matrix(log_delay~., data = data1)[,-1]

#model_matrix = model.matrix(Min.Delay~., data = df)

# Separate the response variable from the predictors
y1 = data1$sq_price
#y1 = df$Min.Delay
is.matrix(model_matrix)
dim(model_matrix)
# Create a sequence of lambda values to test
lambda_seq= 10^seq(-3, 10, length.out = 100)
library(glmnet)
# Use cv.glmnet to perform cross-validation and select the optimal lambda value
#cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq)
set.seed(100)
cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq,nfolds=10,standardize=FALSE)

# View the optimal lambda value
best_lambda2=cv_fit$lambda.1se
# Plot the cross-validation results
dev.off()
plot(cv_fit)

# Fit final model, get its sum of squared residuals and multiple R-squared
#model_cv = glmnet(x = model_matrix, y= y1, alpha = 0, lambda = best_lambda)
```

```
dim(model_matrix)
# Create a sequence of lambda values to test
lambda_seq= 10^seq(-3, 10, length.out = 100)
library(glmnet)
# Use cv.glmnet to perform cross-validation and select the optimal lambda value
#cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq)
set.seed(100)
cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq,nfolds=10,standardize=FALSE)

# View the optimal lambda value
best_lambda2=cv_fit$lambda.1se
# Plot the cross-validation results
dev.off()
plot(cv_fit)

# Fit final model, get its sum of squared residuals and multiple R-squared
#model_cv = glmnet(x = model_matrix, y= y1, alpha = 0, lambda = best_lambda)
model_cv2 = glmnet(x = model_matrix, y=y1, alpha = 0, lambda = best_lambda2,standardize = FALSE)

y_hat_ridge2 = predict(model_cv2, model_matrix)


actual=y1
pred1=c(y_hat_ridge1)
pred2=c(y_hat_ridge2)
d_actual=trainset_new$price
d_pred1=as.integer(pred1^2)
d_pred2=as.integer(pred2^2)
library(Metrics)
rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m

coef(model_cv2 ,s=best_lambda2)
```

```
#For the test set

data2=testset_new[,-2]
data2[,c(2,4,5)]=scale(data2[,c(2,4,5)], center = TRUE, scale = TRUE)
data2[,2]=as.numeric(data2[,2])
data2[,4]=as.numeric(data2[,4])
data2[,5]=as.numeric(data2[,5])

View(data2)
nrow(data2)
#model_matrix1 = model.matrix(log_delay~., data = data2)
model_matrix1 = model.matrix(sq_price~., data = data2)[,-1]

# Separate the response variable from the predictors
#y1 = data1$Min.Delay
y2 = data2$sq_price
model_cv2 = glmnet(x = model_matrix1, y=y2, alpha = 0, lambda = best_lambda2,standardize = FALSE)

y_hat_ridge2 = predict(model_cv2, model_matrix1)

actual=y2
pred1=c(y_hat_ridge1)
pred2=c(y_hat_ridge2)

d_actual=testset_new$price
d_pred1=as.integer(exp(pred1))
d_pred2=as.integer(exp(pred2))
d_actual=testset_new$price
d_pred1=as.integer(pred1^2)
d_pred2=as.integer(pred2^2)

rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m
```

```
#Lasso Regression
# Perform 10-fold cross-validation to select lambda ----------------------------
# Setting alpha = 1 implements lasso regression

#lasso_cv = cv.glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq, nfolds = 10)
set.seed(100)
lasso_cv = cv.glmnet(x = model_matrix, y=y1, alpha = 1, lambda = lambda_seq, nfolds = 10,standardize=FALSE)
# Plot cross-validation results
dev.off()
plot(lasso_cv)

# Best cross-validated lambda
lambda_cv2 = lasso_cv$lambda.1se

# Fit final model, get its sum of squared residuals and multiple R-squared
#model_cv = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv)
model_cv2 = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv2,standardize = FALSE)

y_hat_lasso2 = predict(model_cv2, model_matrix)
actual=y1
pred1=c(y_hat_lasso1)
pred2=c(y_hat_lasso2)

d_actual=trainset_new$price
d_pred1=as.integer(pred1^2)
```

```
d_pred1=as.integer(pred1^2)
d_pred2=as.integer(pred2^2)
rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m


#rsq_lasso_cv = cor(y, y_hat_cv)^2
coef2=coef(model_cv2 ,s=lambda_cv2)

# extract coefficients as a vector
coef_vec = as.vector(coef2)


# Convert coefficients to a data frame with variable names
df_coef = data.frame(var = row.names(coef(model_cv2, s=lambda_cv2)),
                     coef = coef(model_cv2, s=lambda_cv2)[,1])

ggplot(data = df_coef, aes(x =var,  y = coef)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "", x = "Variables", y = "Coefficients") +
  theme(plot.title = element_text(hjust = 0.5))+
  coord_flip()

# See how increasing lambda shrinks the coefficients --------------------------
# Each line shows coefficients for one variables, for different lambdas.
# The higher the lambda, the more the coefficients are shrinked towards zero.
res = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq,standardize = FALSE)
plot(res, xvar = "lambda",label=TRUE,lw=2)
#legend("bottomright", lwd = 1, col = 1:6, legend = colnames(model_matrix), cex = .7)


#For the test set

model_cv2 = glmnet(x = model_matrix1, y=y2, alpha = 1, lambda = lambda_cv2,standardize = FALSE)

y_hat_lasso2 = predict(model_cv2, model_matrix1)
```

```
actual=y2
pred1=c(y_hat_lasso1)
pred2=c(y_hat_lasso2)

d_actual=testset_new$price
d_pred1=as.integer(pred1^2)
d_pred2=as.integer(pred2^2)

rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m
d_rmse1=sqrt(sum((d_actual -d_pred1)^2)/nrow(data2))
d_rmse2=sqrt(sum((d_actual -d_pred2)^2)/nrow(data2))

#Elastic Net Regression
# Set training control
set.seed(100)

train_control = trainControl(method = "repeatedcv",
                             number = 10,
                             repeats = 5,
                             search = "random",
                             verboseIter = TRUE)

View(data1)
#str(data1)
# Train the model
#elastic_net_model = train(log_delay ~ .,data = data1,method = "glmnet",preProcess = c("center", "scale"),tuneLength = 25,trControl = train_control)
elastic_net_model = train(sq_price ~ .,
                          data = data1,
                          method = "glmnet",
                          trControl = train_control)

# Check multiple R-squared
```

```
# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data1[,-1])
actual=y1
pred=c(y_hat_enet)
rmse=sqrt(sum((actual -pred)^2)/nrow(data1))
d_actual=trainset_new$price
d_pred=as.integer(pred^2)

rmse_m = rmse(d_actual, d_pred)
rmse_m
mape_m = mape(d_actual, d_pred)
mape_m

d_rmse1=sqrt(sum((d_actual -d_pred)^2)/nrow(data1))
#For the test set
#elastic_net_model1 = train(log_delay ~ .,data = data2,method = "glmnet",trControl = train_control)

# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data2[,-1])

actual=y2
pred=c(y_hat_enet )
rmse=sqrt(sum((actual -pred)^2)/nrow(data2))
d_actual=testset_new$price
d_pred=as.integer(pred^2)

rmse_m = rmse(d_actual, d_pred)
rmse_m
mape_m = mape(d_actual, d_pred)
mape_m
#Random Forest
```

```
# Define the tuning grid
tune_grid = expand.grid(
  mtry =c(15,20,25,30),
  splitrule = "variance",
  min.node.size = c(1, 5, 10,15)
)
# Tune the hyperparameters using the tune() function
library(ranger)


# Perform 5-fold cross-validation
set.seed(100)
rf_model = train(
  sq_price ~ .,
  data = data1,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
  tuneGrid=tune_grid,
  importance="impurity"
)
# Get best hyperparameters
best_params = rf_model$bestTune
plot(rf_model)
#For the training set
y_hat_enet = predict(rf_model, data1[,-1])
actual=y1
pred=c(y_hat_enet)
d_actual=trainset_new$price
d_pred=as.integer(pred^2)
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
rmse=sqrt(sum((actual -pred)^2)/nrow(data1))
d_actual=trainset_new$price
d_pred=as.integer(exp(pred))
d_rmse1=sqrt(sum((d_actual -d_pred)^2)/nrow(data1))
```

```
#For the test set

y_hat_enet = predict(rf_model, data2[,-1])

actual=y2
pred=c(y_hat_enet )
d_actual=testset_new$price
d_pred=as.integer(pred^2)
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
rmse=sqrt(sum((actual -pred)^2)/nrow(data2))
d_actual=testset$Min.Delay
d_pred=as.integer(exp(pred))

d_rmse1=sqrt(sum((d_actual -d_pred)^2)/nrow(data2))

# get variable importance
# Print the variable importance measures
var_imp=rf_model1$finalModel$variable.importance
# print the variable importance
print(var_imp)

# load ggplot2 library
library(ggplot2)


var_imp = data.frame(Variables = names(rf_model$finalModel$variable.importance),
                     Importance = rf_model$finalModel$variable.importance,
                     row.names = NULL)

ggplot(data = var_imp, aes(x = reorder(Variables, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
  theme(plot.title = element_text(hjust = 0.5))+
  coord_flip()

df=var_imp[order(-var_imp$Importance),]
```

```
df=var_imp[order(-var_imp$Importance),]
View(df)
df[c(1:20),]

nrow(df)


# Calculate number of variables to be removed

vars_to_remove = df$Variables[21:46]

# Remove least important variables from data1

dummy_data1= dummyVars(" ~ . ", data = data1,sep = "")
data1_encoded=predict(dummy_data1, newdata =data1)

View(data1_encoded)
data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
View(data1_imp)
colnames(data1_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data1_imp = data1_imp[, !(colnames(data1_imp) %in% base_var)]
View(data1_imp)
str(data1_imp)
data1_imp=as.data.frame(data1_imp)
# Convert dummy variables to factors
num_cols = c("sq_price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data1_imp), num_cols)
data1_imp[factor_cols] = apply(data1_imp[factor_cols], 2, function(x) as.factor(x))

# Check the data types
str(data1_imp)

# Define the tuning grid for random forest hyperparameters
tune_grid = expand.grid(
  mtry =20,
  splitrule = "variance",
  min.node.size = c(1, 5, 10,15)
```

```
set.seed(100)
rf_model = train(
  sq_price~ .,
  data = data1_imp,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
  tuneGrid=tune_grid,
  importance="impurity"
)

# Make predictions

#For the training set
y_hat_enet = predict(rf_model, data1_imp[,-1])
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m

d_actual=trainset_new$price
d_pred=as.integer(pred^2)
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
```

```
#For the test set
# Calculate number of variables to be removed



# Remove least important variables from data1

dummy_data2= dummyVars(" ~ . ", data = data2,sep = "")
data2_encoded=predict(dummy_data2, newdata =data2)

View(data2_encoded)
data2_imp = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
View(data2_imp)
colnames(data2_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
View(data2_imp)

data2_imp=as.data.frame(data2_imp)
# Convert dummy variables to factors
num_cols = c("sq_price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data2_imp), num_cols)
data2_imp[factor_cols] = apply(data2_imp[factor_cols], 2, function(x) as.factor(x))
```

```r
data2_imp[factor_cols] = apply(data2_imp[factor_cols], 2, function(x) as.factor(x))

y_hat_enet = predict(rf_model, data2_imp[,-1])

actual=y2
pred=c(y_hat_enet )
d_actual=testset_new$price
d_pred=as.integer(pred^2)
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2, pred)
mape_m
```

```r
#Xgboost
set.seed(100)
Xtrain = xgb.DMatrix(data = as.matrix(data1[,-1]),label=y1)
y_train = data1$sq_price
xgb_model = train(Xtrain,
                  y_train,
                  method = "xgbTree",
                  objective = "reg:squarederror",
                  trControl = trainControl(method = "cv",
                                           number = 5,

                                           verboseIter = TRUE),


                  tuneGrid = expand.grid(nrounds = c(500,1000),
                                         eta=0.1,
                                         max_depth = c(2,4,6),
                                         colsample_bytree = c(0.5,0.6),
                                         subsample = c(0.5,0.6),
                                         gamma=0.1,
                                         min_child_weight = c(1,3,5)
                  ))

plot(xgb_model)
y_hat_enet = predict(xgb_model, Xtrain)
actual=y1
pred=c(y_hat_enet)
rmse=sqrt(sum((actual -pred)^2)/nrow(data1))
d_actual=trainset_new$price
d_pred=as.integer(pred^2)
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
```

```r
y_hat_enet = predict(xgb_model, data2[,-1])

actual=y2
pred=c(y_hat_enet )
rmse=sqrt(sum((actual -pred)^2)/nrow(data2))
d_actual=testset_new$price
d_pred=as.integer(pred^2)
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
d_rmse1=sqrt(sum((d_actual -d_pred)^2)/nrow(data2))

# get variable importance
var_imp =varImp(xgb_model)

# print the variable importance
print(var_imp)

# load ggplot2 library
library(ggplot2)


# plot the variable importance
ggplot(data = var_imp, aes(x = reorder(rownames(var_imp), Overall), y = Overall)) +
  geom_bar(stat = "identity", fill = "steelblue") +
```

```r
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
  theme(plot.title = element_text(hjust = 0.5))




df=as.data.frame(var_imp$importance)
nrow(df)
Variables=rownames(df)
df=cbind(df,Variables)

# Calculate number of variables to be removed

vars_to_remove = df$Variables[13:46]

# Remove least important variables from data1

dummy_data1= dummyVars(" ~ . ", data = data1,sep = "")
data1_encoded=predict(dummy_data1, newdata =data1)

View(data1_encoded)
data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
View(data1_imp)
colnames(data1_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data1_imp = data1_imp[, !(colnames(data1_imp) %in% base_var)]
View(data1_imp)
str(data1_imp)
data1_imp=as.data.frame(data1_imp)
# Convert dummy variables to factors
num_cols = c("sq_price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data1_imp), num_cols)
data1_imp[factor_cols] = apply(data1_imp[factor_cols], 2, function(x) as.factor(x))

# Check the data types
str(data1_imp)


set.seed(100)
```

```r
#For the test set
# Remove least important variables from data2

dummy_data2= dummyVars(" ~ . ", data = data2,sep = "")
data2_encoded=predict(dummy_data2, newdata =data2)

View(data2_encoded)
data2_imp = data2_encoded[, !(colnames(data2_encoded) %in% vars_to_remove)]
View(data2_imp)
colnames(data2_imp)
base_var=c("brand_NewBajaj","ownerFirst Owner","stateAndhra Pradesh" )
data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
View(data2_imp)
str(data2_imp)
data2_imp=as.data.frame(data2_imp)
# Convert dummy variables to factors
num_cols = c("log_price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data2_imp), num_cols)
data2_imp[factor_cols] = apply(data2_imp[factor_cols], 2, function(x) as.factor(x))
data2_imp$log_price=as.numeric(data2_imp$log_price)
# Check the data types
str(data2_imp)
y_hat_enet = predict(xgb_model, data2_imp[,-1])

actual=y2
pred=c(y_hat_enet )
d_actual=testset_new$price
d_pred=as.integer(exp(pred))
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
rmse_m = rmse(y2,pred)
rmse_m
mape_m = mape(y2,pred)
mape_m


################################################################################
#Skewness and kurtosis
skewness(trainset_new$price)#positively skewed
```

```r
data1_imp[factor_cols] = apply(data1_imp[factor_cols], 2, function(x) as.factor(x))

str(data1_imp)
Xtrain = xgb.DMatrix(data = as.matrix(data1_imp[,-1]),label=y1)
y_train = data1_imp$sq_price

set.seed(100)
xgb_model = train(Xtrain,
                  y_train,
                  method = "xgbTree",
                  objective = "reg:squarederror",
                  trControl = trainControl(method = "cv",
                                           number = 5,

                                           verboseIter = TRUE),


                  tuneGrid = expand.grid(nrounds = c(500,1000),
                                         eta=0.1,
                                         max_depth = c(2,4,6),
                                         colsample_bytree = c(0.5,0.6),
                                         subsample = c(0.5,0.6),
                                         gamma=0.1,
                                         min_child_weight = c(1,3,5)
                  ))

plot(xgb_model)
y_hat_enet = predict(xgb_model, Xtrain)
actual=y1
pred=c(y_hat_enet)
d_actual=trainset_new$price
d_pred=as.integer(pred^2)
mape_m = mape(d_actual, d_pred)
rmse_m = rmse(d_actual, d_pred)
mape_m
rmse_m
rmse_m = rmse(y1,pred)
rmse_m
```

```r
cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq,nfolds=10,standardize=FALSE)

# View the optimal lambda value
best_lambda1=cv_fit$lambda.min
best_lambda2=cv_fit$lambda.1se
# Plot the cross-validation results
dev.off()
plot(cv_fit)

# Fit final model, get its sum of squared residuals and multiple R-squared
#model_cv = glmnet(x = model_matrix, y= y1, alpha = 0, lambda = best_lambda)
model_cv1 = glmnet(x = model_matrix, y=y1, alpha = 0, lambda = best_lambda1,standardize = FALSE)
model_cv2 = glmnet(x = model_matrix, y=y1, alpha = 0, lambda = best_lambda2,standardize = FALSE)

y_hat_ridge1 = predict(model_cv1, model_matrix)
y_hat_ridge2 = predict(model_cv2, model_matrix)
library(Metrics)
actual=y1
pred1=c(y_hat_ridge1)
pred2=c(y_hat_ridge2)
d_actual=trainset_new$price
d_pred1=as.integer(pred1^2)
d_pred2=as.integer(pred2^2)
rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m

model_cv1 = glmnet(x = model_matrix1, y=y2, alpha = 0, lambda = best_lambda1,standardize = FALSE)
model_cv2 = glmnet(x = model_matrix1, y=y2, alpha = 0, lambda = best_lambda2,standardize = FALSE)

y_hat_ridge1 = predict(model_cv1, model_matrix1)
y_hat_ridge2 = predict(model_cv2, model_matrix1)
library(Metrics)
actual=y1
pred1=c(y_hat_ridge1)
pred2=c(y_hat_ridge2)
d_actual=testset_new$price
d_pred1=as.integer(pred1^2)
```

```r
mape_m = mape(d_actual, d_pred2)
mape_m

model_cv1 = glmnet(x = model_matrix1, y= y2, alpha = 1, lambda = lambda_cv1,standardize = FALSE)
model_cv2 = glmnet(x = model_matrix1, y= y2, alpha = 1, lambda = lambda_cv2,standardize = FALSE)

y_hat_lasso1 = predict(model_cv1, model_matrix1)
y_hat_lasso2 = predict(model_cv2, model_matrix1)
actual=y1
pred1=c(y_hat_lasso1)
pred2=c(y_hat_lasso2)

d_actual=testset_new$price
d_pred1=as.integer(pred1^2)
d_pred2=as.integer(pred2^2)
rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m
```

```r
################################################################################
data1=trainset_new[,-2]


dummy_1= dummyVars(" ~ . ", data = data1,sep = "")
data1_encoded=predict(dummy_1, newdata =data1)
vars_imp=c("sq_price","power","age","brand_NewHarley-Davidson","kms_driven","brand_NewTriumph","brand_NewRoyal Enfield","brand_NewKTM","brand_NewKawasaki"
           ,"brand_NewDucati","brand_NewHyosung","brand_NewBenelli","brand_NewSuzuki","brand_NewYamaha","stateKarnataka","stateMaharashtra","stateDelhi","brand_NewHonda"
           ,"ownerSecond Owner","brand_NewHero","stateHaryana","stateUttar Pradesh","stateTelangana","stateGujarat","brand_NewTVS","stateTamil Nadu","brand_NewBMW"
           ,"ownerThird Owner","statewest Bengal","brand_NewMahindra","stateRajasthan")
View(data1_encoded)
data1_imp = subset(data1_encoded,select=vars_imp)
View(data1_imp)
colnames(data1_imp)
data1_imp=as.data.frame(data1_imp)
# Convert dummy variables to factors
num_cols = c("sq_price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data1_imp), num_cols)
data1_imp[factor_cols] = apply(data1_imp[factor_cols], 2, function(x) as.factor(x))
data1_imp=as.data.frame(data1_imp)
str(data1_imp)
data1_imp[,c(2,3,5)]=scale(data1_imp[,c(2,3,5)], center = TRUE, scale = TRUE)
data1_imp[,2]=as.numeric(data1_imp[,2])
data1_imp[,3]=as.numeric(data1_imp[,3])
data1_imp[,5]=as.numeric(data1_imp[,5])
model_matrix = model.matrix(sq_price~., data = data1_imp)[,-1]

data2=testset_new[,-2]


dummy_2= dummyVars(" ~ . ", data = data2,sep = "")
data2_encoded=predict(dummy_2, newdata =data2)
vars_imp=c("sq_price","power","age","brand_NewHarley-Davidson","kms_driven","brand_NewTriumph","brand_NewRoyal Enfield","brand_NewKTM","brand_NewKawasaki"
           ,"brand_NewDucati","brand_NewHyosung","brand_NewBenelli","brand_NewSuzuki","brand_NewYamaha","stateKarnataka","stateMaharashtra","stateDelhi","br
           ,"ownerSecond Owner","brand_NewHero","stateHaryana","stateUttar Pradesh","stateTelangana","stateGujarat","brand_NewTVS","stateTamil Nadu","brand_
           ,"ownerThird Owner","statewest Bengal","brand_NewMahindra","stateRajasthan")
View(data2_encoded)
data2_imp = subset(data2_encoded,select=vars_imp)
View(data2_imp)
colnames(data2_imp)
data2_imp=as.data.frame(data2_imp)
# Convert dummy variables to factors
num_cols = c("sq_price", "kms_driven", "age","power")
# Convert non-numeric columns to factors
factor_cols = setdiff(names(data2_imp), num_cols)
data2_imp[factor_cols] = apply(data2_imp[factor_cols], 2, function(x) as.factor(x))
data2_imp=as.data.frame(data2_imp)
str(data2_imp)
data2_imp[,c(2,3,5)]=scale(data2_imp[,c(2,3,5)], center = TRUE, scale = TRUE)
data2_imp[,2]=as.numeric(data2_imp[,2])
data2_imp[,3]=as.numeric(data2_imp[,3])
data2_imp[,5]=as.numeric(data2_imp[,5])
model_matrix1 = model.matrix(sq_price~., data = data2_imp)[,-1]

# Separate the response variable from the predictors
y1 = data1$sq_price
y2 = data2$sq_price

#y1 = df$Min.Delay
is.matrix(model_matrix1)
dim(model_matrix1)
# Create a sequence of lambda values to test
lambda_seq= 10^seq(-3, 10, length.out = 100)
library(glmnet)
# Use cv.glmnet to perform cross-validation and select the optimal lambda value
#cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq)
set.seed(100)
cv_fit = cv.glmnet(x = model_matrix, y= y1 , alpha = 0, lambda = lambda_seq,nfolds=10,standardize=FALSE)
```

```r
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m
#lasso_cv = cv.glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq, nfolds = 10)
set.seed(100)
lasso_cv = cv.glmnet(x = model_matrix, y=y1, alpha = 1, lambda = lambda_seq, nfolds = 10,standardize=FALSE)
# Plot cross-validation results
dev.off()
plot(lasso_cv)

# Best cross-validated lambda
lambda_cv1 = lasso_cv$lambda.min
lambda_cv2 = lasso_cv$lambda.1se

# Fit final model, get its sum of squared residuals and multiple R-squared
#model_cv = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv)
model_cv1 = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv1,standardize = FALSE)
model_cv2 = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_cv2,standardize = FALSE)

y_hat_lasso1 = predict(model_cv1, model_matrix)
y_hat_lasso2 = predict(model_cv2, model_matrix)
actual=y1
pred1=c(y_hat_lasso1)
pred2=c(y_hat_lasso2)

d_actual=trainset_new$price
d_pred1=as.integer(pred1^2)
d_pred2=as.integer(pred2^2)
rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m

model_cv1 = glmnet(x = model_matrix1, y= y2, alpha = 1, lambda = lambda_cv1,standardize = FALSE)
model_cv2 = glmnet(x = model_matrix1, y= y2, alpha = 1, lambda = lambda_cv2,standardize = FALSE)

y_hat_lasso1 = predict(model_cv1, model_matrix1)
y_hat_lasso2 = predict(model_cv2, model_matrix1)
actual=y1
pred1=c(y_hat_lasso1)
pred2=c(y_hat_lasso2)

d_actual=testset_new$price
d_pred1=as.integer(pred1^2)
d_pred2=as.integer(pred2^2)
rmse_m = rmse(d_actual, d_pred1)
rmse_m
rmse_m = rmse(d_actual, d_pred2)
rmse_m
mape_m = mape(d_actual, d_pred1)
mape_m
mape_m = mape(d_actual, d_pred2)
mape_m
```