TORONTO BUS DELAYS

ST3082 Project II

Advanced Data Analysis

Group 04

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

Abstract

This report aims to outline the findings of the exploratory data analysis which was carried out based on the Toronto bus delays 2022 data set obtained from the open-source data site, Kaggle. Furthermore, an advanced analysis was carried out on identifying the frontline reasons for Toronto bus delays along with their inter relationships and fitting a suitable model to predict the delay time of buses with high accuracy.

Contents

List of Figures

List of Tables

1. Introduction

In Toronto, buses play a major role in public transportation. As per records in 2021, TTC bus system has 192 bus routes in operation including 28 night bus routes carrying over 264 million riders over 6686 kilometers of routes with buses travelling 143 million kilometers in the year. The system had a ridership of about 8,74,300 per weekday by the end of first 6 months of 2022. Bus routes extend throughout the city and are integrated with the subway system and the steer car system with free transfers among the 3 systems. In the recent past years, TTC is constantly being complained on the delays of the bus system despite of having a wide network. Numerous research studies have been done on the factors leading to these delays in order to fix this issue with the guidance of the professionals, since the primary objective of transportation efficiency is directly affected from these delays in the bus system. With the dataset in hand, an analysis can be performed to find the factors that have the most influence on bus delays in Toronto and a model can be built in order to predict the delay time in minutes accurately for the ease and the awareness of the riders.

2. Description of the Question

It is important and timely to address the issue in drop of efficiency in the Toronto bus system due to delays, in order to fix the problematic spots in the bus network, as it is directly connected with the day today life activities of millions of people. Hence, identifying and understanding the main factors resulting this issue is essential in setting the path to predict the delay time in minutes out of the visible causes such as route types, incidents causing delays, week of the day etc, to make the passenger lives easy by being aware of the possible amount of delay times.

3. Description of the data set

The Toronto bus delays 2022 dataset contains 27351 records and 10 variables, including Delay time as the response variable.

Variable Name	Description	Variable Type
Date	Date of the delay incident	Qualitative
Route	Route number which the incident happened	Qualitative
Time	Time of the day which the incident happened	Quantitative
Day	Day of the week which the incident happened	Qualitative
Location	Location in Toronto which the delay incident happened	Qualitative
Incident	The incident which caused the delay	Qualitative
Min.Delay	Amount of time which the bus got delayed	Quantitative
Min.Gap	Time gap in minutes with the next bus scheduled	Quantitative
Direction	Direction of the route which the particular bus travels	Qualitative
Vehicle	Vehicle number of the bus	Qualitative

Table 3.1

4. Pre-processing

Toronto bus delay dataset contains data under 10 variables where 7 are categorical and 3 are numerical. In the dataset, 154 duplicated entries were discovered and those were removed. Then it was found that there are 164 missing data records in the *Route* variable and 5529 missing records in the "Direction" variable. Afterwards, all the records having 0 as the *Min.Delay* and *Min.Gap* were removed from the dataset. By doing so, the count of missing values of *Route* and *Direction* variables got reduced to 26 and 4987 respectively with a total of 25898 records.

4.1 Feature Engineering

A separate "Month" variable was created by extracting the months in the "Date" variable. Distinct values in the "Direction" column were identified and entries with error values discovered in that variable were filtered out. Considering the "Route" variable, routes labelled in text form were replaced by a random number for the ease of cleaning process.

1.Regular and Limited Service Routes	7 – 189 series	Weekdays – 6am-1am Weekends - 8am-1am
2.Blue Night Routes	300 -399 series	Weekdays – 1am-6am Weekends - 1am-8am
3.Community Routes	400 – 499 series	Only in rush hours of weekdays Rush hours (According to TTC statistics) :- Morning – 6am- 10am Evening - 3pm- 7pm
4.Express Routes	900 – 999 series	Any time with in the 24 hours (according to requirements)

Table 4.1

Based on the above details we got from the TTC official website, the new variable *Route_New* was created by categorizing all the distinct routes in the *Route* variable into 5 categories (Blue Night Routes, Community Routes, Express Routes, Regular and limited service routes, Others) based on their route numbers. The *Hour* variable was introduced to the dataset by extracting the hours in the *Time* variable. The new variable *is_Weekday* was formed by categorizing the days in the *Day* column into 2 categories (Weekday, Weekend). The *Incident_New* variable was created by taking the distinct categories in the *Incident* column and combining 3 categories with lowest number of cases (Cleaning – Disinfection, Held By, Late Entering Service)

into one category (Others), and renaming the category *"*Road Blocked – NON–TTC Collision*"* into *"*Road Blocked*"*. The *Vehicle* variable was removed from the dataset as it acts as an identifying variable and it causes no effect for the delay analysis.

Furthermore, by referring to the website of TTC bus system, the operating days and times of the bus routes were identified and the records which do not follow this day and time system under its corresponding bus route were filtered out as they are contradictory with the system data which left us with a total of 23853 records. Finally, the dataset was split into training and test data sets as 80% and 20% from the original data respectively for further analysis. Afterwards the variables with missing values of the training set were taken into consideration and we identified missing values in *Route*,*Route_New* and *Direction* variables as 24, 24, 3528 respectively. As *Route* variable is not considered for further analysis while *Direction* variable leads to misinterpretations with imputing using the mode as all its categories have approximately equal number of data currently, we only imputed the *Route_New* variable using the *Hour* and the *is_Weekday* observations relevant to those records with data in Table 4.1.

5. Important Results of the Descriptive Analysis

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

● When the delay time distribution was analyzed, a positively skewed distribution was observed along with 994 outliers. Hence, considering a transformation on the response would help to overcome the issues arising from the above.

● Most delay cases were reported in the regular and limited service routes(7-189) as it functions from 6am-1am and 8am-1am on weekdays and weekends respectively .Furthermore the buses which travel towards North Toronto tend to report more delays.

● Weekdays record the most number of delay cases with in 6am-7pm particularly as those are the identified rush hours in weekdays according to the statistics of TTC. Mechanical problems and delays in operations due to the operators are the major reasons which were identified as the causes of the delays with in these identified time period.

Figure 5.1

● The time gap between the scheduled time of two buses shares a positive linear relationship with the delay time, along with a considerably higher correlation of 0.96.

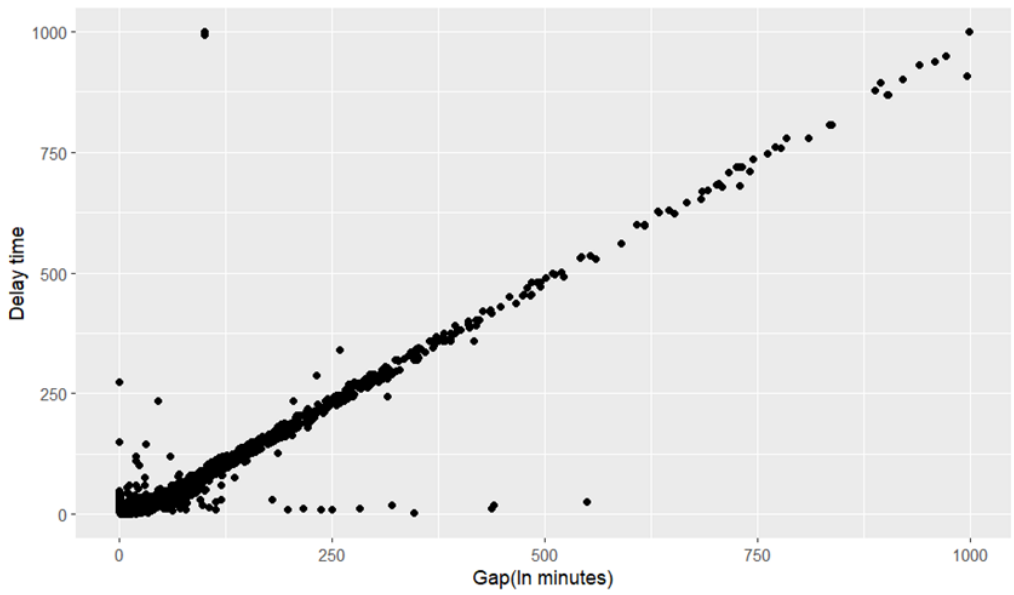

Figure 5.2

● Partial least squares regression(PLSR),also resulted a dominant VIP score for Min.Gap which confirmed the identified linear relationship between Min.Gap and Min.Delays . Moreover, 35 outliers were identified via PLSR.

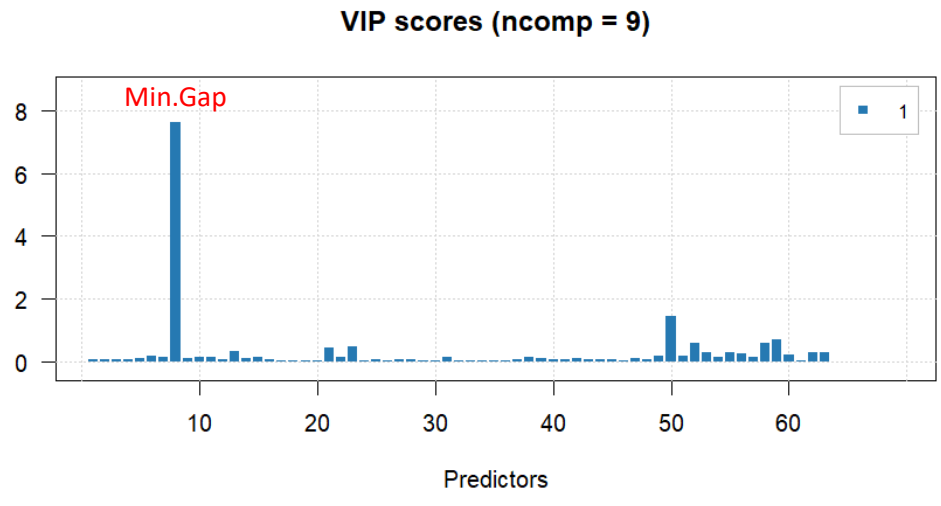

Figure 5.3

● Cluster analysis confirmed the non-existence of clearly distinct clusters where it resulted average silhouette distances less than 0.1 for the considered k values in the range of 2:8 where the highest value was 0.0894 under k=3.

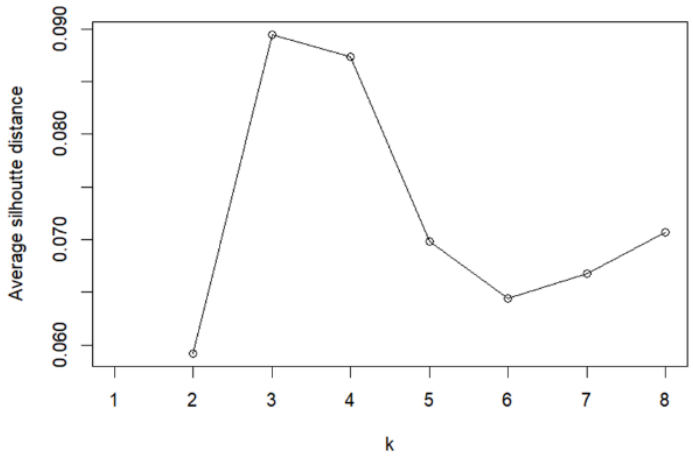

Figure 5.4

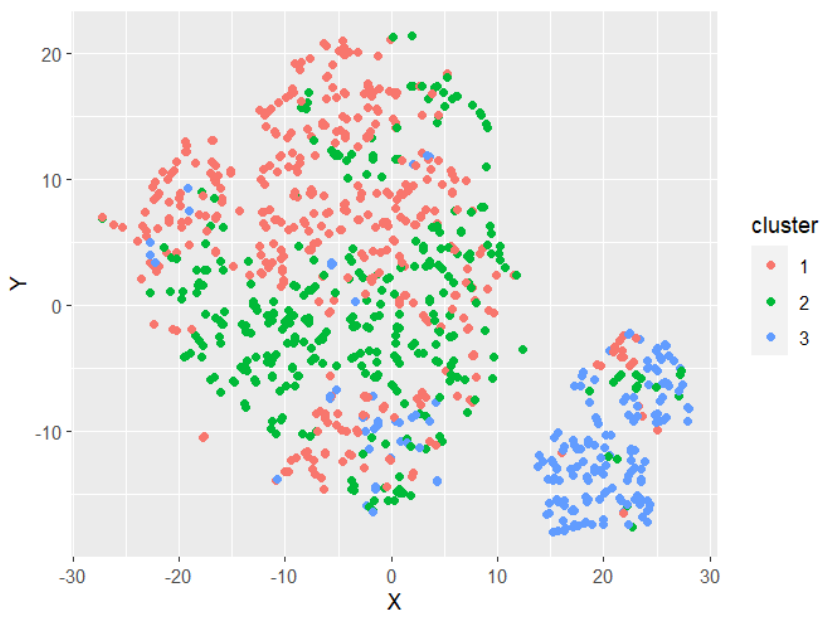

Figure 5.5

● Apart from the strong linear correlation of 0.96 identified between *Min.Gap* and *Min.Delay*, strong associations were identified among the *Min.Gap* and categorical predictors such as *Route_New* , *Incident_New* and *Hour* etc through the Kruskal Wallis results where there were strong associations among the categorical predictors identified through chi squared test results also.

6. Important Results of the Advanced Analysis

The predictive models analyzed throughout this analysis are derived with the assumption that the *Min.Delay* variable is of continuous data. Accordingly, with regard to the final result 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 delays and that of actual counts using the Root Mean Squared Error (RMSE) metric with 10- fold cross validation.

Model	Training RMSE	Test RMSE	Test MAPE
Ridge	4.919888	7.255654	15.89%
Lasso	4.93747	7.271554	16.16%
Elastic Net	4.797923	7.509318	22.75%

Table 6.1

Comparing the table values of the Test RMSE for the three models, clearly the ridge model reports the lowest, furthermore it has the lowest difference between the training and test RMSE values depicting to have the maximum minimization of the issue of overfitting. But, obtaining 15.89% MAPE value suggests a considerably less prediction accuracy relevant to the ridge model.

Hence, selecting a comprehensive model by considering the above three models might not be fair enough.

Hence, to improve the model accuracy regarding the MAPE, more sophisticated but that can handle multiple issues simultaneously were applied.

Table 6.2 below are the results obtained using random forest and XGBoosting after hyper parameter tuning.

	With all the features				With the important features		
Model	**Best Parameters**	**Training RMSE**	**Test RMSE**	**Test MAPE**	**Training RMSE**	**Test RMSE**	**Test MAPE**
Random Forest	mtry=50 splitrule=variance min.node.size=5	3.595297	5.278197	9.18%	3.746567	6.524895	14.75%
XGBoost	nrounds=500 max_depth=2 eta=0.1 gamma=0.1 colsample_bytree=0.6 min_child_weight=1 subsample=0.5	2.799788	5.520246	9.57%	3.868539	6.530456	14.86%

Table 6.2

Note: the important features for the Random Forest and XGBoosting were decided by taking all the features that have scores greater than 100 and 0.1 respectively.

As depicted on Table 6.2, the results obtained when all the features were considered, both the techniques suggest that prediction accuracy has increased but comparatively the random forest model with all the features shows better results with both less Test RMSE and Test MAPE. Despite all that, Test MAPE values seem to have reduced compared to all the regression models applied in this case. Moreover, the random forest and xgboost models considering all the features reported training MAPE values of 7.56% and 5.58% respectively, approving the fact that random forest model with all the features is less overfitted. Even though, models with the important features were fitted, the prediction accuracy has decreased with considerably higher Test MAPE values. Table below shows the variable importance plots of the two models with all the features considered.

Random Forest Model

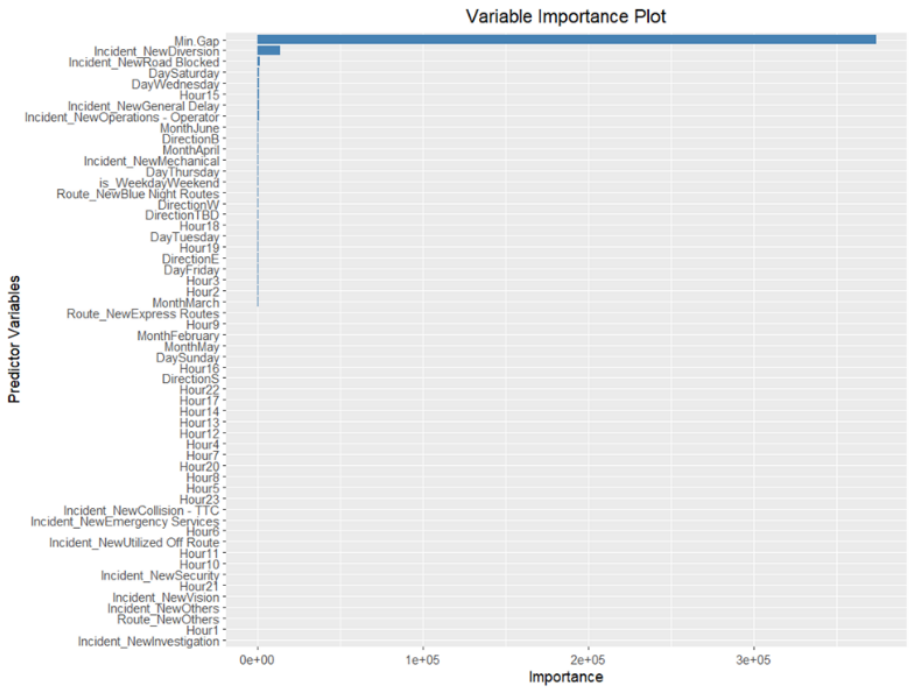

Figure 6.1

XGBoost Model

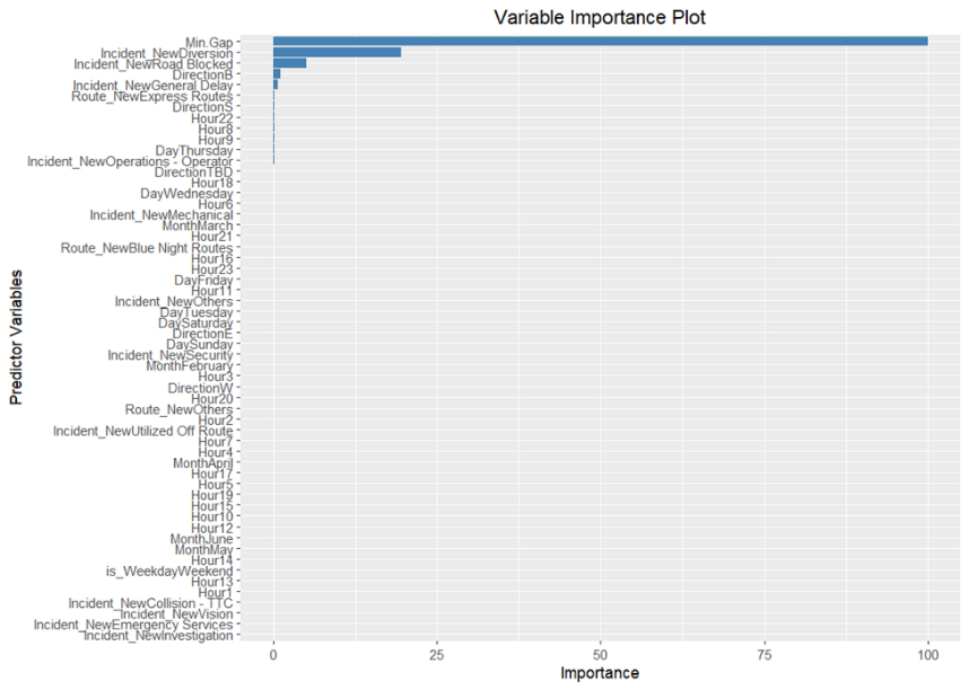

Figure 6.2

From both models we observe that the variable *Min.Gap* has resulted a dominant importance in predicting the delays, leaving majority of other variables negligible. This observation coincides with the result that we identified in the descriptive analysis where the scatterplot of *Min.Gap Vs Min.Delay* depicted a perfect positive linear relationship with a correlation coefficient of 0.96.

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, *Min.Delay* .

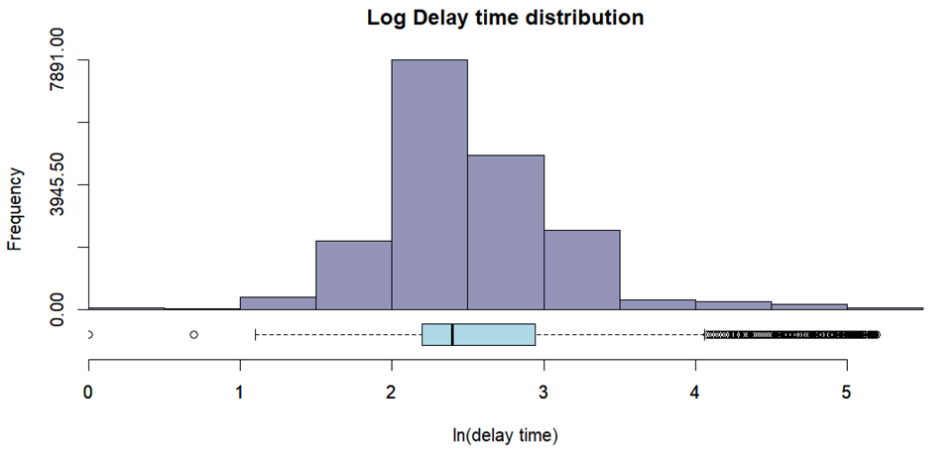

Figure 6.3

	Response – Min.Delay	Response – Log(Min.Delay)
Skewness	5.192685	0.6769573
Kurtosis	37.93428	5.319729

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**	**Test MAPE**
Ridge	30.85677	42.79989	35.59%
Lasso	25.99747	38.51643	25.42%
Elastic Net	33.03114	48.24348	35.75%

Table 6.3

Above results show that the log transformation has not been successful in fitting linear relationships between the response and predictors where the figure 6.4 confirms that the perfect positive positive linear relationship between Min.Delay and Min.Gap has become non-linear between ln(Min.Delay) and Min.Gap as it is identified as the most important predictor in this scenario. Hence we decided to shift to the random forest and xgboost methods from the regularization techniques in search of more accuracy.

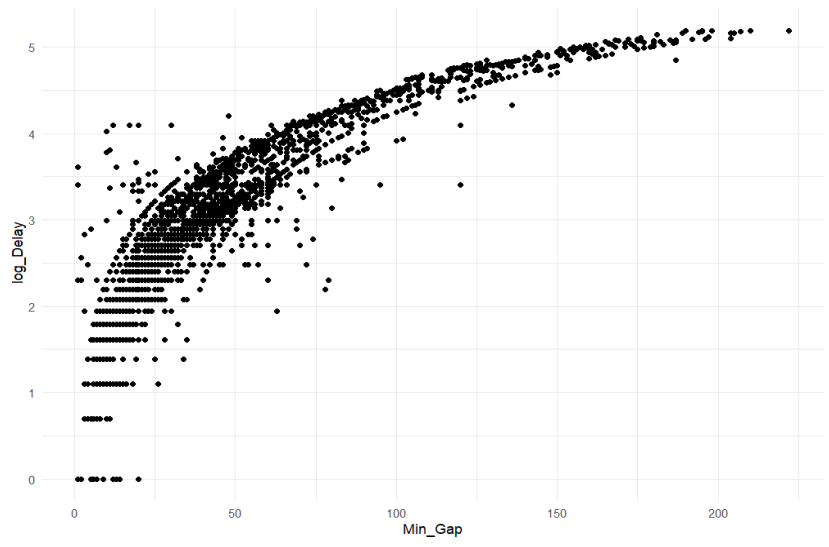

Figure 6.4

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**	**Test MAPE**	**Training RMSE**	**Test RMSE**	**Test MAPE**
Random Forest	mtry=35 splitrule=variance min.node.size=5	4.571086	6.377511	13.49%	2.654157	6.973025	14.82%
XGBoost	nrounds=500 max_depth=2 eta=0.1 gamma=0.1 colsample_bytree=0.5 min_child_weight=1 subsample=0.6	3.172071	6.110895	11.97%	3.450382	6.312643	13.28%

Table 6.4

Note: the important features for the Random Forest and XGBoosting were decided by taking all the features that have scores greater than 0.1.

As depicted on Table 6.4, the results obtained when all the features were considered, both the techniques suggest that prediction accuracy is high in terms of both RMSEs and Test MAPE but lower comparatively to the results we obtained for the random forest model and XGBoost models considering the response in original scale. Even though, models with the important features have increased the accuracy in the training data, the prediction accuracy has decreased related to the test data with a considerable amount of overfitting and higher Test MAPE values. Figure 6.5 and Figure 6.6 depict the variable importance plots relevant to this scenario with all features considered.

Random Forest Model

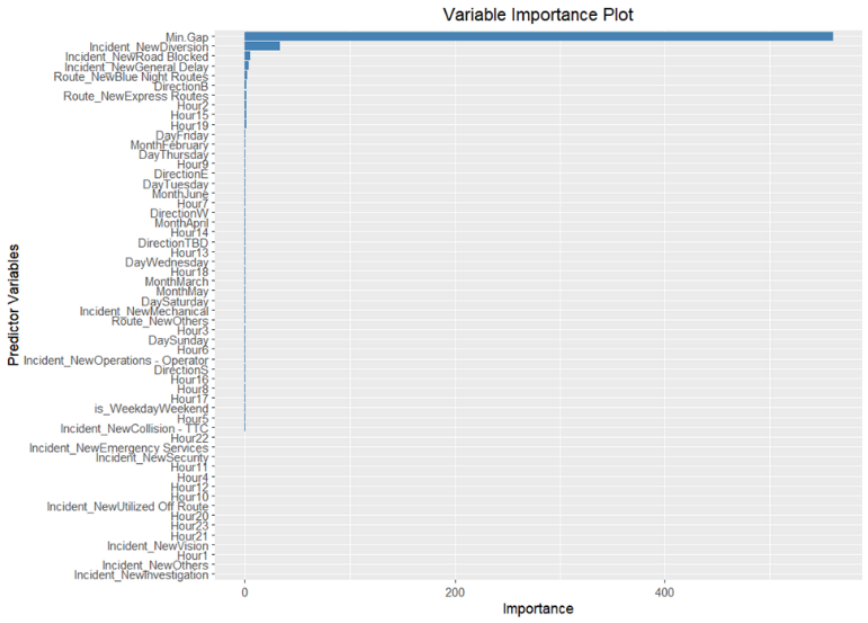

Figure 6.5

XGBoost Model

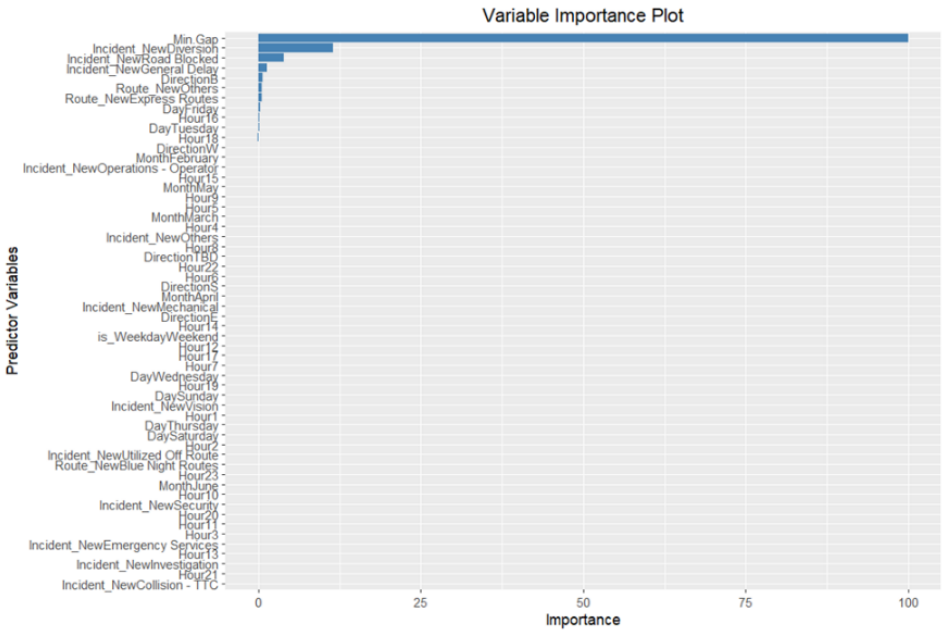

Figure 6.6

7. Issues Encountered and Proposed Solutions

- 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

- 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.

- When almost all the models didn't show vast differences in terms of training and test RMSEs, in order to identify which models are overfitted, we considered their training and test MAPEs, particularly for the models with considerably less training and test RMSEs which paved the path to select the best model.

8. Discussion and Conclusions

- Even though our initial distribution of the response variable was highly skewed to the right, the models which were fitted under the regularization techniques Ridge, lasso and Elastic Net resulted fair enough training and test RMSE values.

- It was evident that *Min.Gap* variable is dominant among all the variables in all the variable importance plots coinciding with the fact it shares a correlation of 0.96 with the response. Due to the above reason the results of the models with all the features and only the important features didn't result vast differences in terms of RMSE values.

- The results obtained in random forest model on the response in original scale considering all the features showed the highest accuracy regarding the test RMSE and test MAPE values which are 5.278197 ,9.18% respectively. Moreover, the obtained training MAPE was reported to be 7.56% which approves a lower overfitting of the model comparing with the test MAPE. The obtained results after log transformation relevant to the ensemble techniques also were fair enough but not better than the results obtained from the random forest model on the response variable.

- When all the models were compared, the Random Forest model applied on the original response considering all the features was identified as the best model which produces the most accurate predictions with less information loss. Figure 8.1 shows the partial dependency plot of Min.Gap which is the only numerical variable in the selected random forest model where it depicts the predicted delay time increases as the time gap increases in the model.

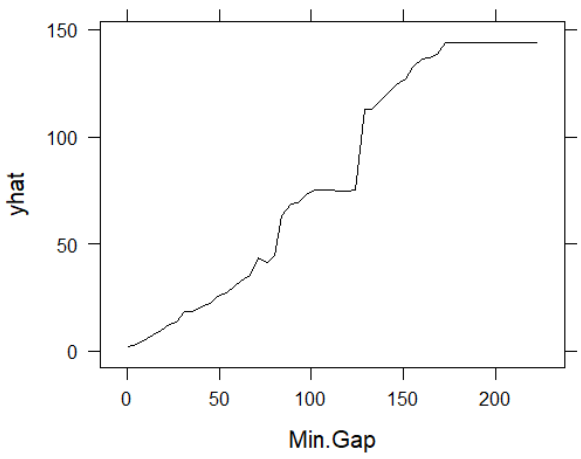

Figure 8.1

9. Appendix of the code

```r
data=read.csv("D:/3RD YEAR/SEMESTER II/ST3082/Project I Materials//ttc-bus-delay-data-2022.csv")
View(data)
head(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)

#Replacing "" with NA
data = replace(data, data=='', NA)
colSums(is.na(data))

#Removing records with delay=0
data=data[data$Min.Delay != 0, ]
nrow(data)

#Removing records with gap=0
data=data[data$Min.Gap != 0, ]
nrow(data)
```

```r
#Create Route_New column
x=c(table(data$Route))
which(data$Route=="RAD")
which(data$Route=="OTC")
data$Route[5619]=1000
data$Route[7346]=1000
data$Route[19679]=1000
data$Route[13053]=1001

Route_New=c()
for(i in 1:nrow(data)){
  if((!is.na(data$Route[i]))&& (as.numeric(substr(data$Route[i],1,4))>=7)&(as.numeric(substr(data$Route[
    Route_New[i]="Regular and limited service routes"

  }else if((!is.na(data$Route[i]))&&(as.numeric(substr(data$Route[i],1,4))>299)&(as.numeric(substr(data$
    Route_New[i]="Blue Night Routes"

  }else if((!is.na(data$Route[i]))&&(as.numeric(substr(data$Route[i],1,4))>399)&(as.numeric(substr(data$
    Route_New[i]="Community Routes"

  }else if((!is.na(data$Route[i]))&&(as.numeric(substr(data$Route[i],1,4))>899)&(as.numeric(substr(data$
    Route_New[i]="Express Routes"

  }else if(!is.na(data$Route[i])){
    Route_New[i]="Others"
  }else{
    Route_New[i]=NA
  }
}
table(Route_New)
data=cbind(data,Route_New)



#Create Hour column
Hour=c()
for(i in 1:length(data$Time)){
  Hour[i]=as.numeric(substr(data$Time[i],1,regexpr(":",data$Time[i])-1))
}
data=cbind(data,Hour)
```

```r
#Cleaning Data I
y=c()

for(i in 1:nrow(data)){
  if((!is.na(data$Route_New[i]))&&( (data$Route_New[i]=="Regular and limited service routes")&
                                    (data$is_Weekday[i]=="Weekday")&((data$Hour[i]>=1)&(data$Hour[i]<6))
    y=append(y,i)

  }else if((!is.na(data$Route_New[i]))&&( (data$Route_New[i]=="Regular and limited service routes")&
                                          (data$is_Weekday[i]=="Weekend")&((data$Hour[i]>=1)&(data$Hour[
    y=append(y,i)

  }else if((!is.na(data$Route_New[i]))&& ((data$Route_New[i]=="Blue Night Routes")&
                                          (data$is_Weekday[i]=="Weekday")&(data$Hour[i]<1)&(data$Hour[i]
    y=append(y,i)

  }else if((!is.na(data$Route_New[i]))&& ((data$Route_New[i]=="Blue Night Routes")&
                                          (data$is_Weekday[i]=="Weekend")&(data$Hour[i]<1)&(data$Hour[i]
    y=append(y,i)


  }else{
    y=append(y,"")
  }
}
y
y=y[y != ""]
length(y)
data=data %>%  filter(!row_number() %in% as.numeric((y)))
nrow(data)
colSums(is.na(data))
```

```r
#Create Month column
data = data %>%
  mutate(Month = str_extract(Date, "[:alpha:]+")
  )%>%

  mutate(
    Month = ifelse( str_detect(Date, "Jan"),"January", Month)
  ) %>%

  mutate(
    Month = ifelse( str_detect(Date, "Feb"),"February", Month)
  ) %>%
  mutate(
    Month = ifelse( str_detect(Date, "Mar"),"March", Month)
  ) %>%
  mutate(
    Month = ifelse( str_detect(Date, "Apr"),"April", Month)
  ) %>%
  mutate(
    Month = ifelse( str_detect(Date, "May"),"May", Month)
  ) %>%
  mutate(
    Month = ifelse( str_detect(Date, "Jun"),"June", Month)
  )


#Cleaning Direction column
table(data$Direction)
which(data$Direction=="/")
which(data$Direction=="2")
which(data$Direction=="6")
which(data$Direction=="D")
which(data$Direction=="I")
which(data$Direction=="J")
which(data$Direction=="Q")
data=data %>%  filter(!row_number() %in% c(76,19133,21555,6669,11507,152,4249,10326,13986))
nrow(data)
```

```r

#Create Incident_New column
table(Incident_New)
Incident_New=c()
for(i in 1:length(data$Incident)){
  if((data$Incident[i]=="Cleaning - Disinfection")|(data$Incident[i]=="Held By")|
     (data$Incident[i]=="Late Entering Service")){
    Incident_New[i]="Others"
  }else if(data$Incident[i]=="Road Blocked - NON-TTC Collision"){
    Incident_New[i]="Road Blocked"
  }else{
    Incident_New[i]=data$Incident[i]
  }


}
data=cbind(data,Incident_New)

#Create is_Weekday column
is_Weekday=c()
for(i in 1:nrow(data)){
  if((data$Day[i]=="Saturday") | (data$Day[i]=="Sunday") ) {
    is_Weekday[i]="Weekend"
  }else{
    is_Weekday[i]="Weekday"
  }
}
data=cbind(data,is_Weekday)
```

```r
#Factoring
data$Route_New = factor(data$Route_New,level=c("Regular and limited service routes","Blue Night Routes",
                                               "Express Routes","Others"))
data$Day = factor(data$Day,level=c("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday
data$Month = factor(data$Month,level=c("January","February","March","April","May","June"))
data$Incident_New = factor(data$Incident_New,level=c("Cleaning - Unsanitary","Diversion","General Delay"
                                                    "Collision - TTC","Emergency Services","Investigati
                                                    "Road Blocked","Utilized Off Route","Others"))
data$Direction=factor(data$Direction,level=c("N","S","E","W","B"))
data$is_Weekday=factor(data$is_Weekday,level=c("Weekday","Weekend"))
data$Hour=as.factor(data$Hour)


#Removing Vehicle Number column
data=subset(data,select=-Vehicle)
str(data)
sum(table(data$Route_New))
#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)

#Imputing missing values
colSums(is.na(trainset))
#Mode function.
getmode = function(v) {
  uniqv = unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

# Calculate the mode of Route variable.
result1 = getmode(v=trainset$Route_New)
print(result1)
```

```r
# Calculate the mode of Direction variable.
result2 = getmode(v=trainset$Direction)
print(result2)

table(trainset$Direction)


z1=which(is.na(trainset$Route_New))

h1=c()
w1=c()
for(i in 1:length(z1)){
  h1[i]=trainset$Hour[z1[i]]
  w1[i]=trainset$is_Weekday[z1[i]]
}

z2=which(is.na(testset$Route_New))


h2=c()
w2=c()
for(i in 1:length(z2)){
  h2[i]=testset$Hour[z2[i]]
  w2[i]=testset$is_Weekday[z2[i]]
}

trainset$Route_New[is.na(trainset$Route_New)] ="Regular and limited service routes"
trainset$Route_New[15943]="Blue Night Routes"
trainset$Route_New[17936]="Blue Night Routes"
```

```r
testset$Route_New[is.na(testset$Route_New)] ="Regular and limited service routes"
trainset$Direction=factor(trainset$Direction,level=c("N","S","E","W","B","TBD"))
testset$Direction=factor(testset$Direction,level=c("N","S","E","W","B","TBD"))
trainset$Direction[is.na(trainset$Direction)] ="TBD"
testset$Direction[is.na(testset$Direction)] ="TBD"


trainset=subset(trainset,select=-c(Date,Route,Time,Location,Incident))
testset=subset(testset,select=-c(Date,Route,Time,Location,Incident))



#Histogram and Boxplot
options(repr.plot.width=12,repr.plot.height=7)
hist_boxplot(trainset$Min.Delay,main="Delay time distribution",col="#9494b8",xlab="Dealy time(In minutes
boxplot.stats(trainset$Min.Delay)$stats
x=which(trainset$Min.Delay %in% boxplot.stats(trainset$Min.Delay)$out)
length(x)
```

```r
##############################PLS#####################################################


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

View(trainset_encoded)

x=trainset_encoded[,-8]
x[,8]=scale(x[,8], center = TRUE, scale = TRUE)
y=as.matrix(trainset_encoded[,8])
set.seed(100)
ModelPLS = pls(x,y,cv=5, info = "Delay time 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)
plot\IPScores(ModelPLS,ncomp=9, type = "h",show.label = FALSE)
summary(ModelPLS$coeffs)
plot(ModelPLS$coeffs,show.label = TRUE)
summary(ModelPLS$res$cal)
#Checking Outliers
Model0=setDistanceLimits(ModelPLS,lim.type = ModelPLS$lim.type,alpha=0.05)
plotXYResiduals(Model0,show.labels=FALSE,labels="indices")
# Identify outlier indices
# get row indices for outliers in calibration set
outliers = which(categorize(Model0, ModelPLS$res$cal) == "outlier")
length(outliers)
trainset_new=trainset_new %>%  filter(!row_number() %in% outliers)
str(trainset_new)
nrow(trainset_new)
```

```r
##########################Cluster Analysis#########################################################
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(8,9)]=scale(df_encoded[,c(8,9)], center = TRUE, scale = TRUE)
df_encoded[,-c(8,9)] = lapply(df_encoded[,-c(8,9)], 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)
View(df1)
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 2:8){
  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 = 3)
```

```r
# 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))

#############################################################################################
#Creating the model matrix
View(trainset_new)
data1=trainset_new


data1[,3]=scale(data1[,3], center = TRUE, scale = TRUE)
data1[,3]=as.numeric(data1[,3])
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(Min.Delay~., data = data1)[,-1]
#model_matrix = model.matrix(Min.Delay~., data = df)

# Separate the response variable from the predictors
y1 = data1$Min.Delay
#y1 = df$Min.Delay
is.matrix(model_matrix)
dim(model_matrix)
# Create a sequence of lambda values to test
lambda_seq = 10^seq(10, -2, length = 100)
library(glmnet)
```

```r
# 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_cv = glmnet(x = model_matrix, y=y1, alpha = 0, lambda = best_lambda2,standardize = FALSE)


y_hat_ridge = predict(model_cv, model_matrix)
library(Metrics)
rmse_m = rmse(y1, y_hat_ridge)
rmse_m
actual=y1
pred1=c(y_hat_ridge)


coef(model_cv ,s=best_lambda2)
#For the test set
data2=testset


data2[,3]=scale(data2[,3], center = TRUE, scale = TRUE)
data2[,3]=as.numeric(data2[,3])
View(data2)
nrow(data2)
model_matrix1 = model.matrix(Min.Delay~., data = data2)[,-1]

# Separate the response variable from the predictors
#y1 = data1$Min.Delay
y2 = data2$Min.Delay

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

```r
y_hat_ridge2 = predict(model_cv2, model_matrix1)

mape_m = mape(y2, y_hat_ridge2)
rmse_m = rmse(y2, y_hat_ridge2)
mape_m
rmse_m


#Lasso Regression
# Perform 10-fold cross-validation to select lambda ----------------------------

set.seed(100)
lasso_cv = cv.glmnet(x = model_matrix, y=y1, alpha = 1, lambda = lambda_seq, nfolds = 10,standardize=FAL
# 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_cv2,standardize = FALSE)


y_hat_lasso = predict(model_cv, model_matrix)

rmse_m = rmse(y1, y_hat_lasso)
rmse_m


#rsq_lasso_cv = cor(y, y_hat_cv)^2

coef(model_cv2 ,s=lambda_cv2)
```

```r
res = glmnet(x = model_matrix, y= y1, alpha = 1, lambda = lambda_seq,standardize = FALSE)
plot(res, xvar = "lambda")



#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)

mape_m = mape(y2, y_hat_lasso2)
rmse_m = rmse(y2, y_hat_lasso2)
mape_m
rmse_m

#Elastic Net Regression

set.seed(100)

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

elastic_net_model = train(Min.Delay ~ .,
                          data = data1,
                          method = "glmnet",
                          trControl = train_control)
```

```r

# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data1[,-2])
actual=y1
pred=c(y_hat_enet)
mape_m = mape(y1, pred)
rmse_m = rmse(y1, pred)
mape_m
rmse_m



#For the test set
#elastic_net_model1 = train(log_delay ~ .,data = data2,method = "glmnet",trControl = tr

# Check multiple R-squared
y_hat_enet = predict(elastic_net_model, data2[,-2])

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

```r
#Random Forest

set.seed(100)
indexes=sample(1:nrow(data1),0.1*nrow(data1))
data_new=data1[indexes,]

# Define the tuning grid
tune_grid = expand.grid(
  mtry =seq(20,50,by=5),
  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(
  Min.Delay ~ .,
  data = data_new,

  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[,-2])
actual=y1
pred=c(y_hat_enet)
mae_m = mae(y1, pred)
rmse_m = rmse(y1, pred)
mae_m
rmse_m
```

```r
#For the test set
y_hat_enet = predict(rf_model, data2[,-2])
actual=y2
pred=c(y_hat_enet )
mape_m = mape(y2, pred)
rmse_m = rmse(y2, 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()
```

```r
# Define the tuning grid
tune_grid = expand.grid(
  mtry =41,
  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(
  Min.Delay ~ .,
  data = data_new,

  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_imp[,-7])
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1, pred)
rmse_m
```

```r
#Xgboost
library(xgboost)
Xtrain = xgb.DMatrix(data = as.matrix(data1[,-2]),label=y1)
y_train = data1$sMin.Delay
Xtest = xgb.DMatrix(data = as.matrix(data2[,-2]),label=y2)
y_test = data2$Min.Delay

##############################################################################################

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 = 1
                  ))

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
```

```r
View(data1_encoded)
data1_imp = data1_encoded[, !(colnames(data1_encoded) %in% vars_to_remove)]
View(data1_imp)
colnames(data1_imp)
base_var=c("Route_NewRegular and limited service routes","DayMonday","MonthJanuary","In
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("Min.Delay", "Min.Gap")
# 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)
data_new=data1_imp[indexes,]
Xtrain = xgb.DMatrix(data = as.matrix(data1_imp[,-4]),label=y1)
y_train = data1$Min.Delay

set.seed(100)
```

```r

##############################################################################
df=var_imp[order(-var_imp$Importance),]
View(df)
df[c(1:41),]

nrow(df)

# Calculate number of variables to be removed

vars_to_remove = df$Variables[42:56]

# 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("Route_NewRegular and limited service routes","DayMonday","MonthJanuary","Incident_NewCleanin
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("Min.Delay", "Min.Gap")
# 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)
indexes=sample(1:nrow(data1_imp),0.1*nrow(data1_imp))
data_new=data1_imp[indexes,]
```

```r

#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)
data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
View(data2_imp)

data2_imp=as.data.frame(data2_imp)

# 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[,-7])

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

```r
y_hat_enet = predict(xgb_model, Xtest)

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




##############################################################################
# Calculate number of variables to be removed

vars_to_remove = df$Variables[23:56]

# Remove least important variables from data1

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

```r
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 = 1
                  ))

plot(xgb_model)
y_hat_enet = predict(xgb_model, Xtrain)
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m
rmse=sqrt(sum((actual -pred)^2)/nrow(data1))

#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)
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("Min.Delay", "Min.Gap")
# 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))
Xtest = xgb.DMatrix(data = data2_imp[,-4],label=y2)
y_test = data2$Min.Delay

y_hat_enet = predict(xgb_model, Xtest)
actual=y_test
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$Min.Delay,main="Delay time distribution",col="#9494b8",xlab="Dealy time(In min
boxplot.stats(trainset_new$Min.Delay)$stats
x=which(trainset_new$Min.Delay %in% boxplot.stats(trainset_new$Min.Delay)$out)
length(x)

options(repr.plot.width=12,repr.plot.height=7)
hist_boxplot(log(trainset_new$Min.Delay),main="Log Delay time distribution",col="#9494b8",xlab="ln(delay
boxplot.stats(log(trainset_new$Min.Delay))$stats
x=which(log(trainset_new$Min.Delay) %in% boxplot.stats(log(trainset_new$Min.Delay))$out)
length(x)
# Create a ggplot object
ggplot(trainset_new, aes(x = log(Min.Delay))) +

  # Add histogram layer
  geom_histogram(binwidth = 0.1, aes(y = ..density..), fill = "#9494b8", color = "black") +

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

  # Add x-axis label
  xlab("Log delay time ") +

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

  # Set theme
  theme_minimal()


ggplot(data=trainset_new,aes(x=Min.Gap,y=log(Min.Delay)))+
  geom_point()+

  theme_minimal()+

  labs(x="Min_Gap",y=" log_Delay")
```

```
#Skewness and kurtosis
skewness(trainset_new$Min.Delay)#>1 positively skewed
kurtosis(trainset_new$Min.Delay)
skewness(log(trainset_new$Min.Delay))#>1 positively skewed
kurtosis(log(trainset_new$Min.Delay))

log_delay=c(log(trainset_new$Min.Delay))
trainset_new=cbind(log_delay,trainset_new)
log_delay=c(log(testset$Min.Delay))
testset=cbind(log_delay,testset)
################################################################################################
#Creating the model matrix
View(trainset_new)
data1=trainset_new[,-3]
data1[,3]=scale(data1[,3], center = TRUE, scale = TRUE)
data1[,3]=as.numeric(data1[,3])
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_delay~., 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$log_delay
#y1 = df$Min.Delay
is.matrix(model_matrix)
dim(model_matrix)
# Create a sequence of lambda values to test
lambda_seq = 10^seq(10, -2, length = 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

pred2=c(y_hat_ridge2)

d_actual=trainset_new$Min.Delay
d_pred2=as.integer(exp(pred2))
rmse_m=rmse(d_actual,d_pred2)
rmse_m

coef(model_cv2 ,s=best_lambda2)
```

```
#For the test set
View(testset)
data2=testset[,-3]
data2[,3]=scale(data2[,3], center = TRUE, scale = TRUE)
data2[,3]=as.numeric(data2[,3])
View(data2)

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

# Separate the response variable from the predictors
#y1 = data1$Min.Delay
y2 = data2$log_delay

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$Min.Delay

d_pred2=as.integer(exp(pred2))
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)
set.seed(100)
lasso_cv = cv.glmnet(x = model_matrix, y=y1, alpha = 1, lambda = lambda_seq, nfolds = 10,standardize=FAL
# 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

pred2=c(y_hat_lasso2)

d_actual=trainset_new$Min.Delay

d_pred2=as.integer(exp(pred2))
rmse_m=rmse(d_actual,d_pred2)
rmse_m

#rsq_lasso_cv = cor(y, y_hat_cv)^2

coef(model_cv2 ,s=lambda_cv2)
# 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",label=TRUE,lw=2)
```

```
#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

pred2=c(y_hat_lasso2)

d_actual=testset$Min.Delay

d_pred2=as.integer(exp(pred2))
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)
View(data1)
#str(data1)
# Train the model
#elastic_net_model = train(log_delay ~ .,data = data1,method = "glmnet",preProcess = c("center", "scale"
elastic_net_model = train(log_delay ~ .,
                          data = data1,
                          method = "glmnet",
                          trControl = train_control)
```

```
pred=c(y_hat_enet)

d_actual=trainset_new$Min.Delay
d_pred=as.integer(exp(pred))
rmse_m=rmse(d_actual,d_pred)
rmse_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$Min.Delay
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


set.seed(100)
indexes=sample(1:nrow(data1),0.1*nrow(data1))
data_new=data1[indexes,]

# Define the tuning grid
tune_grid = expand.grid(
  mtry =seq(20,50,by=5),
  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_delay ~ .,
  data = data_new,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
  tuneGrid=tune_grid,
  importance="impurity"
)
# Get best hyperparameters
best_params = rf_model$bestTune
plot(rf_model)
#For the training set
y_hat_enet = predict(rf_model, data1[,-1])
actual=y1
pred=c(y_hat_enet)
d_actual=trainset_new$Min.Delay
d_pred=as.integer(exp(pred))
rmse_m = rmse(d_actual, d_pred)
rmse_m

#For the test set

y_hat_enet = predict(rf_model, data2[,-1])
actual=y2
pred=c(y_hat_enet )
d_actual=testset$Min.Delay
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()


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, fill = Imp
  geom_bar(stat = "identity", color = "black") +
  labs(title = "Variable Importance Plot", x = "Predictor Variables", y = "Importance") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_fill_gradientn(colors = c("lightblue", "darkblue"),
                       guide = "none", aesthetics = "fill") +
  coord_flip() +
  scale_y_log10()
```

```
##########################################################################################

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


nrow(df)

# Calculate number of variables to be removed

vars_to_remove = df$Variables[36:56]

# 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("Route_NewRegular and limited service routes","DayMonday","MonthJanuary","Incident_NewCleanin
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("Min.Delay", "Min.Gap")
# 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)
indexes=sample(1:nrow(data1_imp),0.1*nrow(data1_imp))
data_new=data1_imp[indexes,]
# Define the tuning grid
tune_grid = expand.grid(
  mtry =31,
  splitrule = "variance"
)
# Tune the hyperparameters using the tune() function
library(ranger)


# Perform 5-fold cross-validation
set.seed(100)
rf_model = train(
  Min.Delay ~ .,
  data = data_new,

  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_imp[,-7])
actual=y1
pred=c(y_hat_enet)
rmse_m = rmse(y1, pred)
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)
data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
View(data2_imp)
data2_imp=as.data.frame(data2_imp)

# 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[,-7])

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


##########################################################################################
#Xgboost
library(xgboost)
Xtrain = xgb.DMatrix(data = as.matrix(data1[,-2]),label=y1)
y_train = data1$log_delay
Xtest = xgb.DMatrix(data = as.matrix(data2[,-2]),label=y2)
y_test = data2$log_delay
#Xgboost
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 = 1
                  ))

plot(xgb_model)
y_hat_enet = predict(xgb_model, Xtrain)
actual=y1
pred=c(y_hat_enet)
d_actual=trainset_new$Min.Delay
d_pred=as.integer(exp(pred))

rmse_m = rmse(d_actual, d_pred)

rmse_m

#For the test set

y_hat_enet = predict(xgb_model, Xtest)

actual=y2
pred=c(y_hat_enet )
d_actual=testset$Min.Delay
d_pred=as.integer(exp(pred))


# 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))



##########################################################################
# Calculate number of variables to be removed

vars_to_remove = df$Variables[23:56]

# 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("Route_NewRegular and limited service routes","DayMonday","MonthJanuary","Incident_NewCleanin
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_delay", "Min.Gap")
# 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)
data_new=data1_imp[indexes,]
Xtrain = xgb.DMatrix(data = as.matrix(data1_imp[,-4]),label=y1)
y_train = data1$log_delay

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 = 1
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)
data2_imp = data2_imp[, !(colnames(data2_imp) %in% base_var)]
View(data2_imp)
str(data2_imp)
data2_imp=as.data.frame(data2_imp)

# 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))
Xtest = xgb.DMatrix(data = data2_imp[,-4],label=y2)
y_test = data2$log_delay

y_hat_enet = predict(xgb_model, Xtest)
actual=y2
pred=c(y_hat_enet)
rmse_m = rmse(y1,pred)
rmse_m
mape_m = mape(y1,pred)
mape_m
```