Commute – Office Transportation Predictive Modeling

Submission Date: Feb 02, 2020

Author: Ayush Jain
Mentor: Deepak Gupta

Table of Content

Table of Contents

1 Project Objective

This project requires you to understand what mode of transport employees prefers to commute to their office.

Based on the past details of the Employee, we will perform modelling which will help us predict whether the Employee will commute using his Personal Car or not.

The Data consumed is in the form of .csv with the name "Cars.csv". In this we will first investigate and Analyze the data to understand the insights and the readiness of the data for modelling.

The data includes employee information about their mode of transport as well as their personal and professional details like age, salary, work exp.

We are expected to predict whether or not an employee will use Car as a mode of transport. Also, which variables are a significant predictor behind this decision.

In this we will first investigate and Analyze the data to understand the insights and the readiness of the data for modelling.

We will further be performing the Data Modelling and Data Cleaning steps on the given data to ensure that the data is ready for Model Building.

Once the Data is ready, we will be building the below Models on the Data and Interpret the best Model that gives the best results.

- Logistic Regression Model
- K – Nearest Neighbor Model
- Naïve Bayes Model
- Bagging Model
- Boosting Model (eXtreme Gradient Boosting)

The data file contains 444 observations with 9 variables.

2 Exploratory Data Analysis – Step by step approach

Exploratory Data Analysis is one of the important phases in the data Analysis in understanding the significance and accuracy of the data. It usually consists of setting up the environment to work in R, loading the data and checking the validity of data loaded.

A Typical Data exploration activity consists of the following steps:

- Environment Set up and Data Import.
    - Install Necessary Package in R.

- Setting Up Working Directory.
- Reading Dataset in R.
- Checking for Outliers and Null Values in the Dataset
- Checking for Multicollinearity within the Independent variables.
- Performing Univariate Analysis on independent variables.
- Performing Bi-variate Analysis.
- Preparing the Data for Model Building.

- Variable Identification.

We shall follow these steps in exploring the provided dataset.

2.1 Environment Set up and Data Import

2.1.1 Deploying necessary Packages in R.

In this section, we will install and invoke the necessary Packages and Libraries that are going to be the part of our work throughout the project. Having all the packages at the same places increases code readability and Understandability.

```
# Deploying necessary Libraries to the Code.

library(corrplot)
library(DataExplorer)
library(ggplot2)
library(car)
library(caret)
library(caTools)
library(psych)
library(ggbiplot)
library(ipred)
library(e1071)
library(rpart)
library(DMwR)
```

2.1.2 Setting Up Working Directory.

Setting a working directory on starting of the R session makes importing and exporting data files and code files easier. Basically, working directory is the location/ folder on the PC where you have the data, codes etc. related to the project. This helps maintain the code readability and avoid unwanted errors.

```r
# Setting the Working Directory.

setwd("D:/Great Learning/Machine Learning/Project 5")
```

2.1.3 Reading Dataset in R.

The given dataset is in .xlsx format. Hence, the command 'read.xslx' from readxl package is used for importing the file.

```r
# Reading the dataset.

cars<- read.csv("Cars_edited.csv")
```

2.1.4 Performing basic Data checks.

This section of the report checks for the basic steps to ensure that the data is imported properly and also checks the Structure of the dataset and Summary to have the basic understanding of the Data.

```r
#Reading first 10 rows to ensure that the data is loaded correctly

head(cars)

##   Age Gender Engineer MBA Work.Exp Salary Distance license
## 1  28   Male        0   0        4   14.3      3.2       0
## 2  23 Female        1   0        4    8.3      3.3       0
## 3  29   Male        1   0        7   13.4      4.1       0
## 4  28 Female        1   1        5   13.4      4.5       0
## 5  27   Male        1   0        4   13.4      4.6       0
## 6  26   Male        1   0        4   12.3      4.8       1
##          Transport
## 1 Public Transport
## 2 Public Transport
## 3 Public Transport
## 4 Public Transport
## 5 Public Transport
## 6 Public Transport
```

```r
# Understanding the Structure of the Data.
str(cars)
```

```
## 'data.frame':    444 obs. of  9 variables:
##  $ Age      : int  28 23 29 28 27 26 28 26 22 27 ...
##  $ Gender   : Factor w/ 2 levels "Female","Male": 2 1 2 1 2 2 2 1 2 2 ...
##  $ Engineer : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ MBA      : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Work.Exp : int  4 4 7 5 4 4 5 3 1 4 ...
##  $ Salary   : num  14.3 8.3 13.4 13.4 13.4 12.3 14.4 10.5 7.5 13.5 ...
##  $ Distance : num  3.2 3.3 4.1 4.5 4.6 4.8 5.1 5.1 5.1 5.2 ...
##  $ license  : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ Transport: Factor w/ 3 levels "2Wheeler","Car",..: 3 3 3 3 3 3 1 3 3 3
...
```

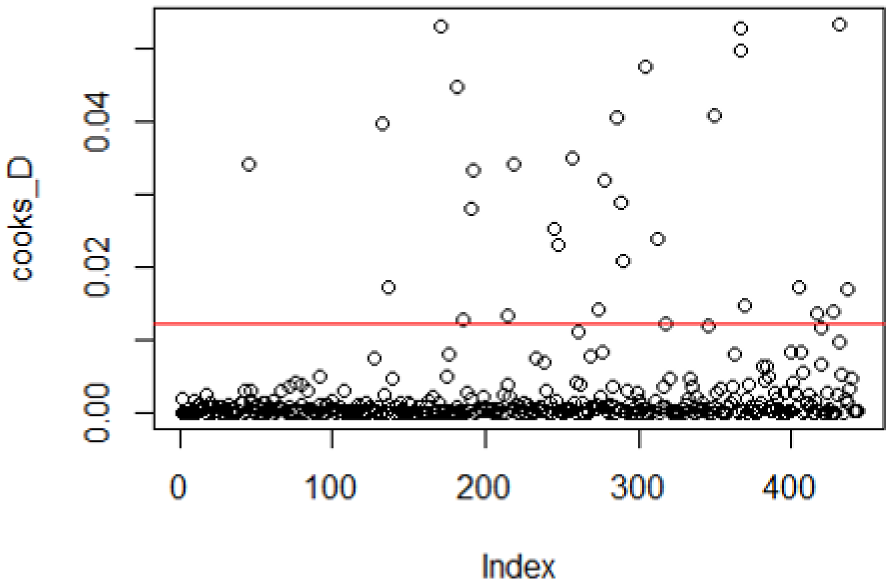
```r
# Checking the summary of the data.

summary(cars)
```

```
##       Age          Gender       Engineer           MBA        
##  Min.   :18.00   Female:128   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:25.00   Male  :316   1st Qu.:1.0000   1st Qu.:0.0000  
##  Median :27.00                Median :1.0000   Median :0.0000  
##  Mean   :27.75                Mean   :0.7545   Mean   :0.2528  
##  3rd Qu.:30.00                3rd Qu.:1.0000   3rd Qu.:1.0000  
##  Max.   :43.00                Max.   :1.0000   Max.   :1.0000  
##                                                NA's   :1       
##     Work.Exp        Salary         Distance        license      
##  Min.   : 0.0   Min.   : 6.50   Min.   : 3.20   Min.   :0.0000  
##  1st Qu.: 3.0   1st Qu.: 9.80   1st Qu.: 8.80   1st Qu.:0.0000  
##  Median : 5.0   Median :13.60   Median :11.00   Median :0.0000  
##  Mean   : 6.3   Mean   :16.24   Mean   :11.32   Mean   :0.2342  
##  3rd Qu.: 8.0   3rd Qu.:15.72   3rd Qu.:13.43   3rd Qu.:0.0000  
##  Max.   :24.0   Max.   :57.00   Max.   :23.40   Max.   :1.0000  
##                                                                 
##             Transport  
##  2Wheeler        : 83  
##  Car             : 61  
##  Public Transport:300  
##                        
## 
```

2.1.5 Checking for Outliers and Null Values in the Dataset.

Outliers: An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. Examination of the data for unusual observations that are far removed from the mass of data. These points are often referred to as outliers.

```r
# Performing Cooks Distance

cd_lm <- lm(as.numeric(cars$Transport)~.,data = cars)
cooks_D <- cooks.distance(cd_lm)

plot(cooks_D)
abline(h=4*mean(cooks_D,na.rm = TRUE),col="red")
```



We found few observations that are present as the Outliers in the dataset. We will not be performing any treatment on this as all the models that we will be building are immune to Outliers and have no impact on the performance.

Null Values: These are the missing values in the dataset that needs to be treated to get the proper accuracy of the Model.

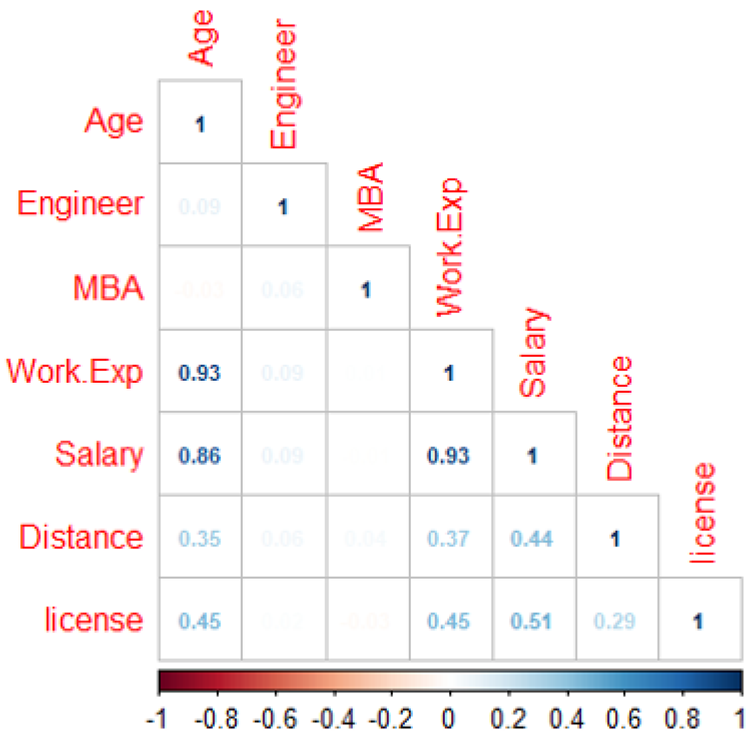
```r
# Checking for the Null Values and Treating it.

dim(cars)

## [1] 444   9

sum(is.na(cars))

## [1] 1

cars<- na.omit(cars)
```

We found 1 value missing in the Dataset which we removed.

2.1.6 Checking for Multicollinearity within the Independent variables.

Multicollinearity is a state of very high intercorrelations or inter-associations among the independent variables. It is therefore a type of disturbance in the data, and if present in the data the statistical inferences made about the data may not be reliable.

```
# Performing Correlation Matrix and Plotting it.

mat<- cor(cars[,-c(2,9)])
corrplot(mat,method = "number", type = "lower", number.cex = .70)
```



From the above Correlation plot, we found that there exists a high correlation between Work.Exp - Age, Work.Exp - Salary and Salary – Age. Which we will further be treating based on the VIF value achieved during Logistic Model.

2.1.7 Performing Univariate Analysis on independent variables.

Univariate analysis is perhaps the simplest form of statistical analysis. Like other forms of statistics, it can be inferential or descriptive. The key fact is that only one variable is involved.

For Numeric variables, default plot is histogram and boxplot while for Categorical variables it is Bar plot.

Histogram: A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable.

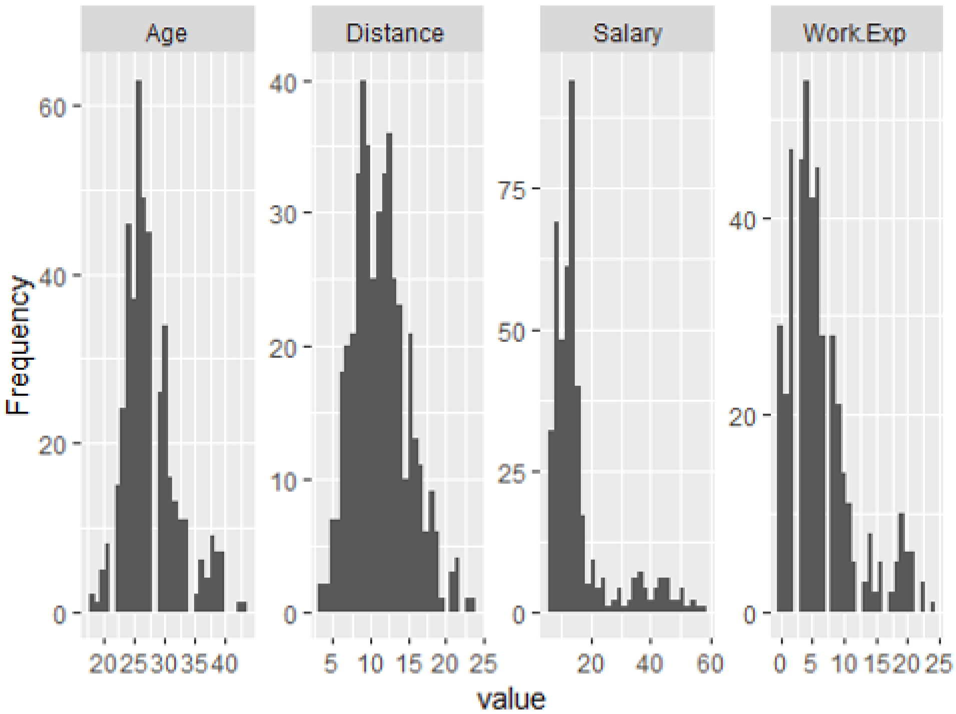
Boxplot: A box plot or boxplot is a method for graphically depicting groups of numerical data through their quartiles. Outliers may be plotted as individual points.

In the problem given, we will be using the above two plotting functions to perform the Univariate analysis on the dataset and identify any outliners present in the data.

Plotting the histogram for all the numeric variables in the dataset.

To analyze each variable, we plot the histogram for the variables.

```
plot_histogram(cars)
```



From the above Plot we can conclude the below:

- From the Histogram and Box plot we can observe that there are certain higher values; especially in ‘Age’, ’Work Experience’ and ‘Salary’.
- These values are well above the respective ‘mean’ in those categories.
- Age: The maximum frequency lies within the Age of 23-27 with a mean of 27.
- Distance: Maximum number of employees stays within the range of 7-12 km.

- Salary: Major number of employees falls in the Salary bracket of 1-15L per Annum.
- Work.Exp: The mean 'Work Experience' of the workforce is slightly less than 6 years but more than 5 years.

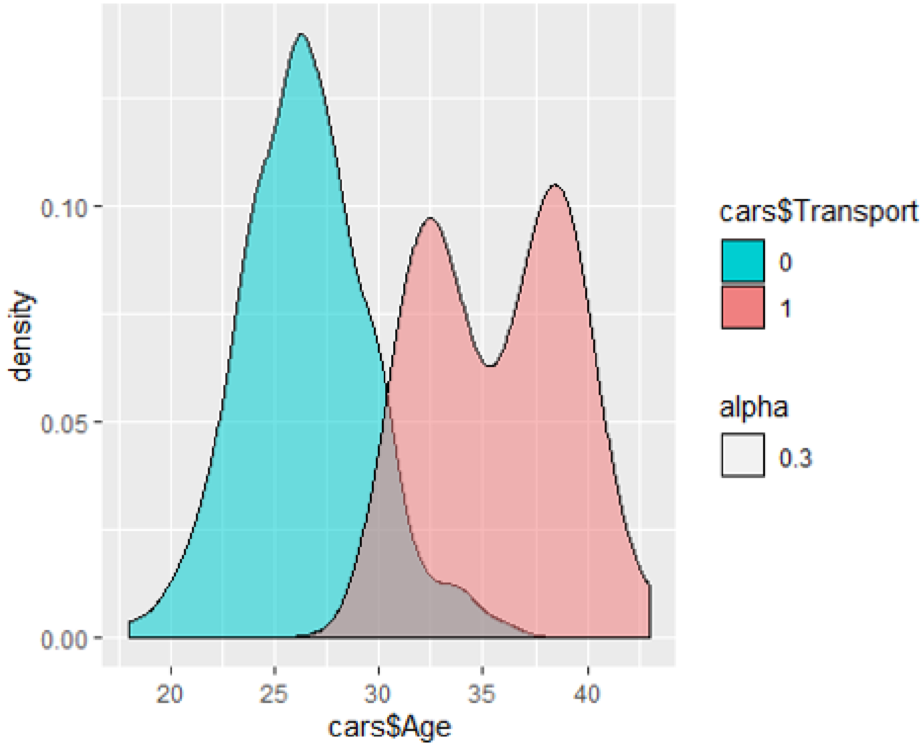
2.1.8 Performing Bi-variate Analysis.

Multivariate analysis is a set of techniques used for analysis of data sets that contain more than one variable, and the techniques are especially valuable when working with correlated variables. The techniques provide an empirical method for information extraction, regression, or classification.

For Multivariate analysis, the default plot is the Scatter Plot or Density Plot. We will be plotting the correlation between the different variables with Churn to understand the relation between the dependent variable Churn with the Independent variables.

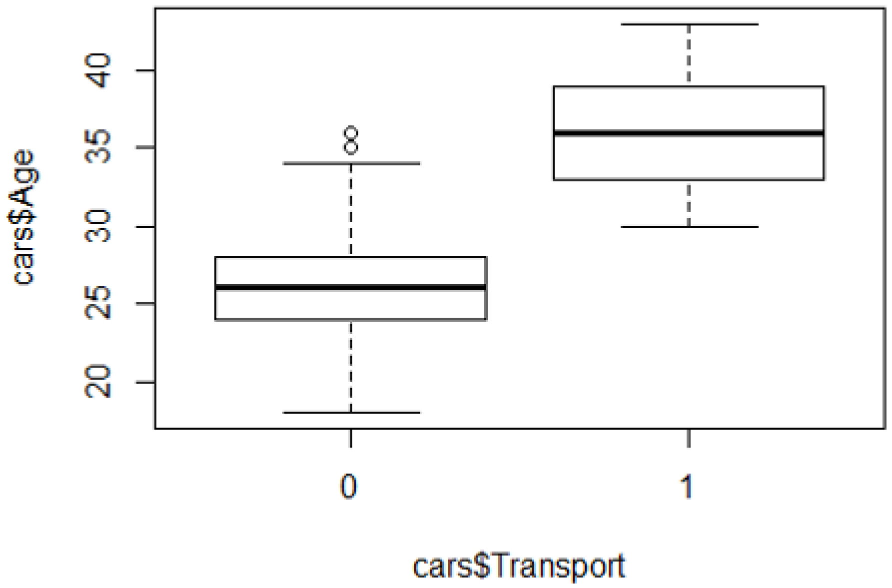
➢ Density Plot of Age with respect to Transport.

```
ggplot(cars, aes(x=cars$Age)) +
  geom_density(aes(fill =cars$Transport, alpha = 0.3)) +
  scale_color_manual(values = c("#868686FF", "#EFC000FF")) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```



➢ Box Plot of Age with respect to Transport.

```
boxplot(cars$Age~ cars$Transport)
```



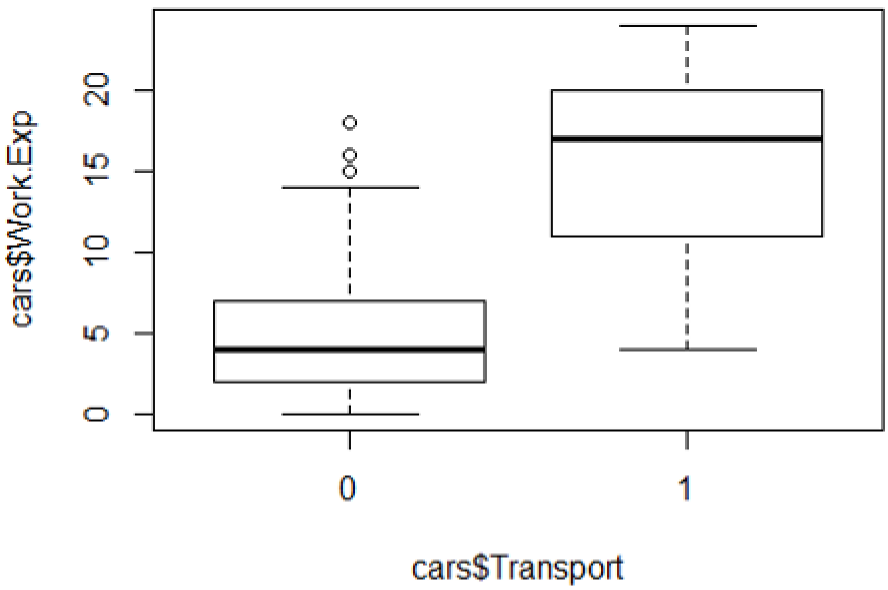
Age is varying from 27 to 47 with a mean of 27.75, median of 27 and range of 25. Since range
is high, we are going to consider it for upcoming analysis.

➢ Density Plot of Work.Exp with respect to Transport.

```
ggplot(cars, aes(x=cars$Work.Exp)) +
  geom_density(aes(fill =cars$Transport, alpha = 0.3)) +
  scale_color_manual(values = c("#868686FF", "#EFC000FF")) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```

➢ Box Plot of Work.Exp with respect to Transport.

```
boxplot(cars$Work.Exp~ cars$Transport)
```



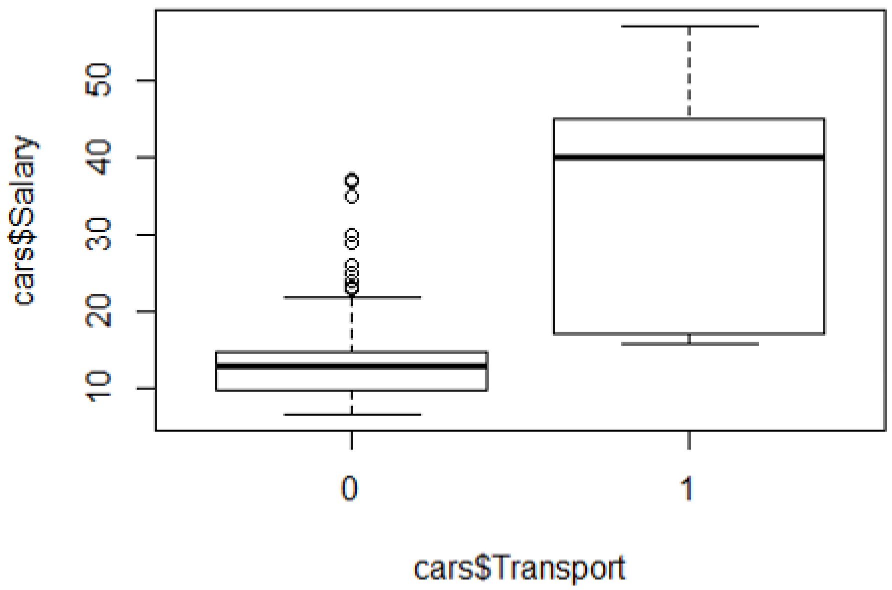
Work.Exp is varying from 0 to 24 with a mean of 6.3, median of 5 and range of 24. Since range is high, we are going to consider it for upcoming analysis.

➢ Density Plot of Salary with respect to Transport.

```
ggplot(cars, aes(x=cars$Salary)) +
  geom_density(aes(fill =cars$Transport, alpha = 0.3)) +
  scale_color_manual(values = c("#868686FF", "#EFC000FF")) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```

➢ Box Plot of Salary with respect to Transport.

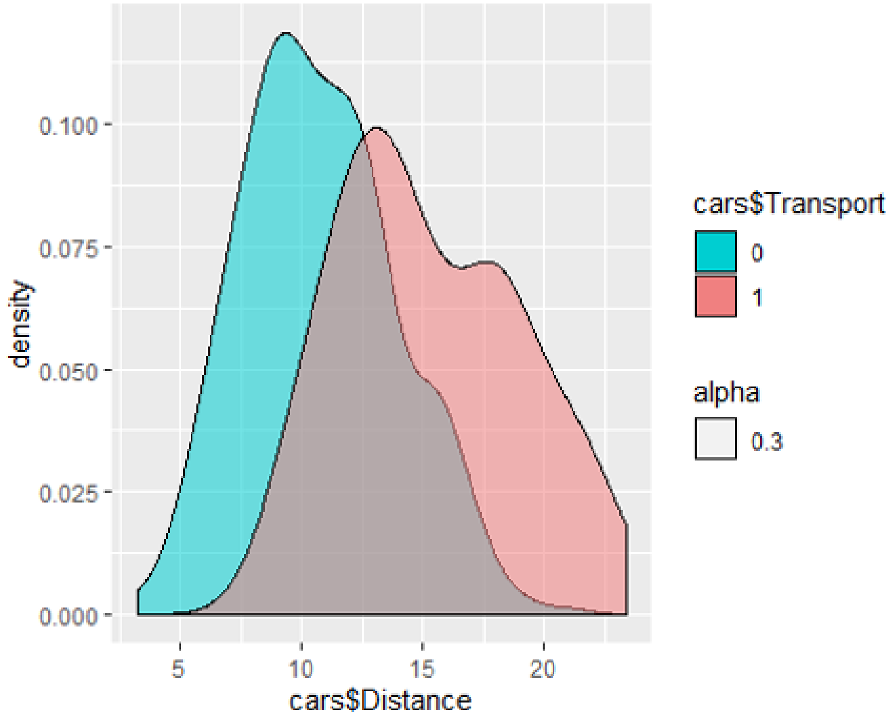
```
boxplot(cars$Salary~ cars$Transport)
```



Salary is varying from 6 to 57 with a mean of 16.24, median of 13.60 and range of 51. Since range is very high, we are going to consider it for upcoming analysis.

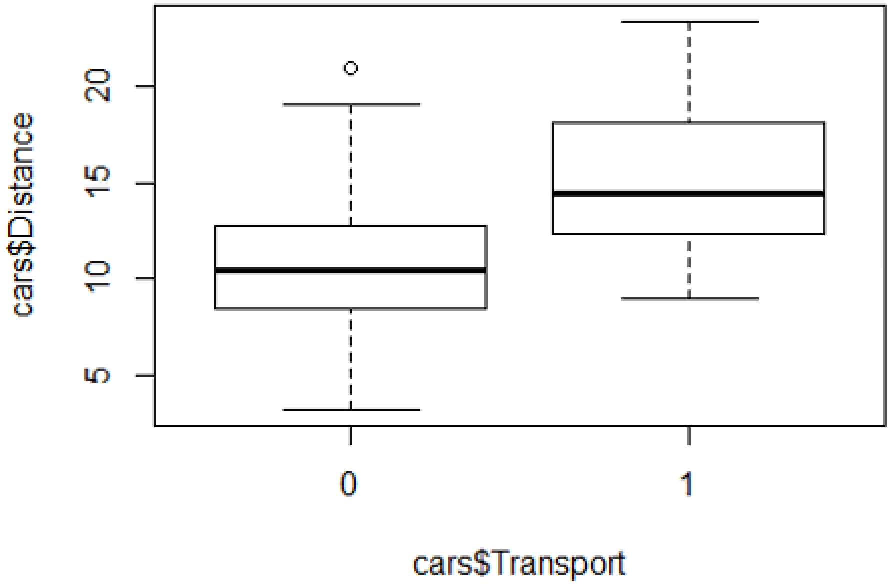
- Density Plot of Distance with respect to Transport.

```
ggplot(cars, aes(x=cars$Distance)) +
  geom_density(aes(fill =cars$Transport, alpha = 0.3)) +
  scale_color_manual(values = c("#868686FF", "#EFC000FF")) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```



➢ Box Plot of Distance with respect to Transport.

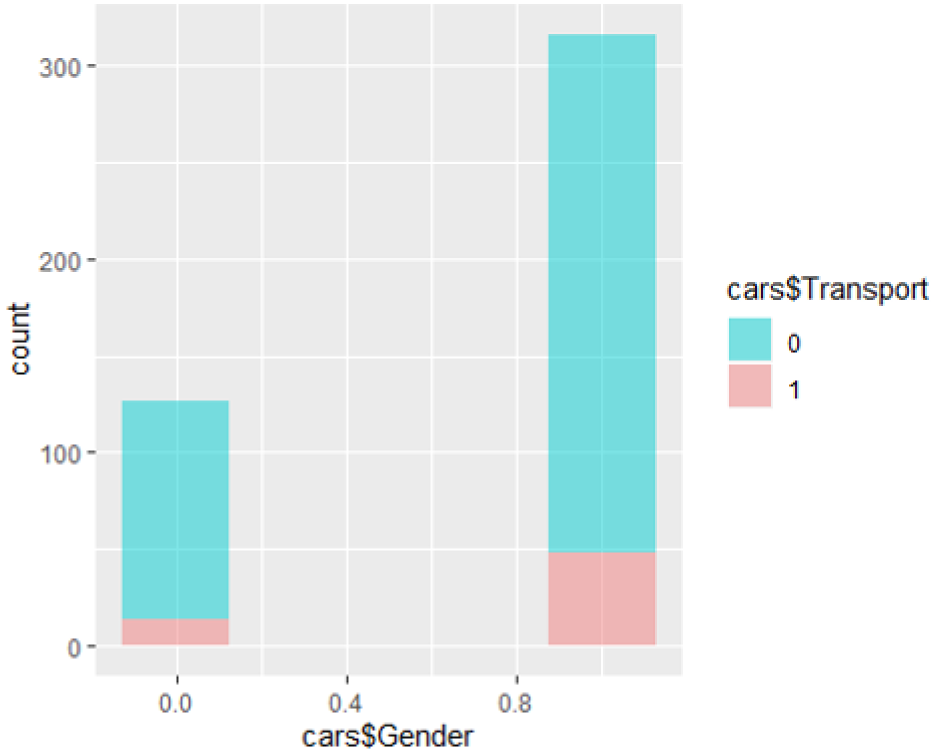
```
boxplot(cars$Distance~ cars$Transport)
```



Distance is varying from 3.20 to 23.40 with a mean of 11.32, median of 11 and range of 20. Since range is high, we are going to consider it for upcoming analysis.

➢ Bar Plot of Gender with respect to Transport.

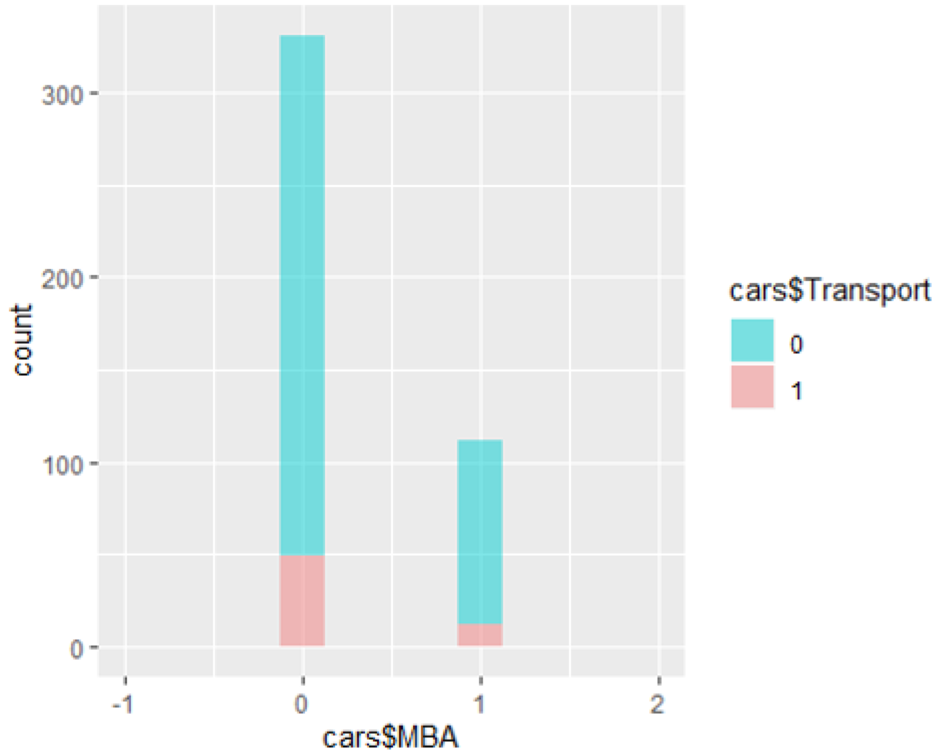
```
ggplot(cars,aes(x = cars$Gender, fill = cars$Transport)) +
  geom_bar(width = 0.25,alpha = 0.5) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```



Gender is a dichotomous variable that specifies the gender of user. From the graph it is clearly seen that car preference for Male is more than female. So, let’s explore its influence on car usage using proportion table too.

➢ Bar Plot of MBA with respect to Transport.

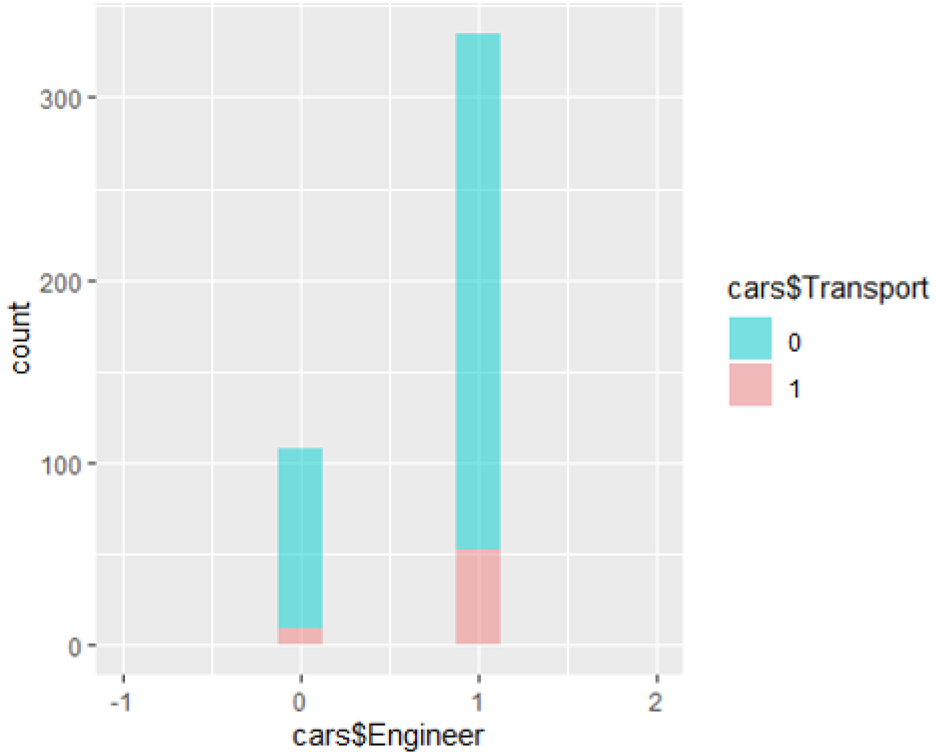
```
ggplot(cars,aes(x = cars$MBA, fill = cars$Transport)) +
  geom_bar(width = 0.25,alpha = 0.5) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen")) +
xlim(-1,2)
```



MBA is a categorical variable specifies the user holds the degree of MBA or not. From the graph it is clear that Non MBA are more inclined to Car Usage than MBA degree holders. Let's explore the proportion table to check the influence on Car Usage.

- Bar Plot of Engineer with respect to Transport.

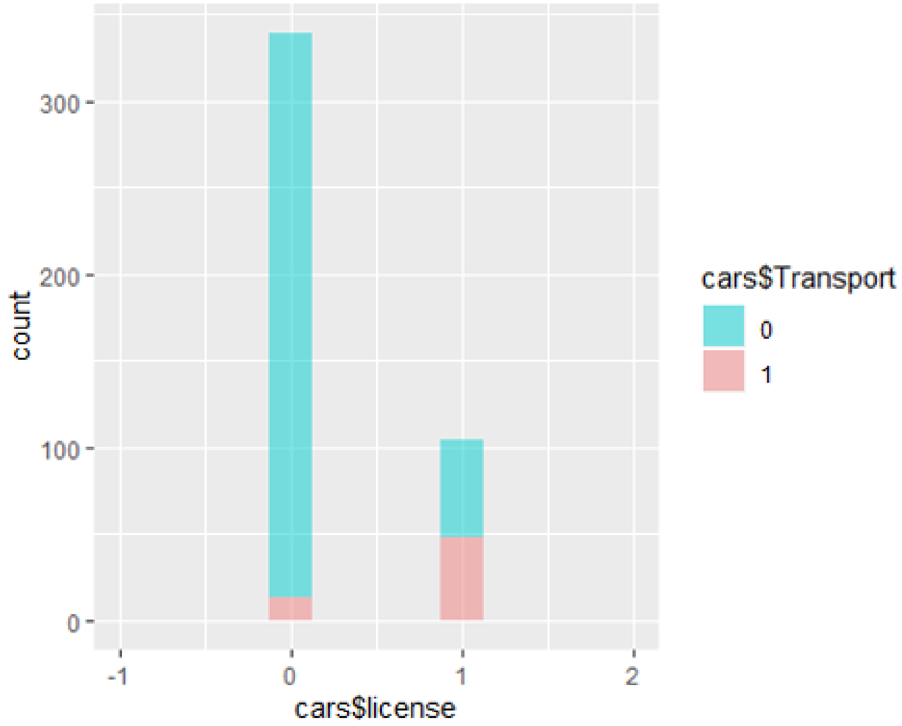
```
ggplot(cars,aes(x = cars$Engineer, fill = cars$Transport)) +
  geom_bar(width = 0.25,alpha = 0.5) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen")) +
xlim(-1,2)
```



Engineer is a categorical variable specifies whether user is Engineer or not. From the graph we can clearly see that Engineer is more inclined towards car usage than non engineer. Let's explore this into proportion table.

➢ Bar Plot of License with respect to Transport.

```
ggplot(cars,aes(x = cars$license, fill = cars$Transport)) +
  geom_bar(width = 0.25,alpha = 0.5) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen")) +
xlim(-1,2)
```



License is a categorical variable specifies the user has driving license or not. From the graph it is clearly seen that license holder users are more inclined to Car Usage. Let's explore the proportion table to check the influence on Car Usage.

2.1.9 Preparing the Data for Model Building.

In this part of the report we will preparing the data for Modelling purpose.

We here convert the Multi Class (Car, 2-Wheeler, Public) Dependent variable (Transport) to Binary Class Variable (0 and 1).

Also, we will be converting Gender into the Categorical variable (0 and 1).

```r
# Converting Dependent variables to 0 and 1
cars$Transport <- ifelse(cars$Transport == "Car",1,0)
cars$Gender <- ifelse(cars$Gender =="Female",0,1)
cars$Transport <- as.factor(cars$Transport)

str(cars)

## 'data.frame':    443 obs. of  9 variables:
##  $ Age      : int  28 23 29 28 27 26 28 26 22 27 ...
##  $ Gender   : num  1 0 1 0 1 1 1 0 1 1 ...
##  $ Engineer : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ MBA      : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Work.Exp : int  4 4 7 5 4 4 5 3 1 4 ...
##  $ Salary   : num  14.3 8.3 13.4 13.4 13.4 12.3 14.4 10.5 7.5 13.5 ...
##  $ Distance : num  3.2 3.3 4.1 4.5 4.6 4.8 5.1 5.1 5.1 5.2 ...
##  $ license  : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "na.action")= 'omit' Named int 145
##   ..- attr(*, "names")= chr "145"
```

Once we have converted Transport and Gender to Categorical Variables, we will now split the Data into Train and Test in 70-30 ratio.

```r
#Splitting the data into Train and Test with a Ratio of 70 and 30 resp.


set.seed(123)
index <- sample.split(cars, SplitRatio = .70)
trainData <- subset(cars,index==TRUE)
dim(trainData)

## [1] 296   9

testData <- subset(cars,index==FALSE)
dim(testData)

## [1] 147   9
```

After splitting the data, we Balance the Train data using SMOTE method. In this we balance the data to 50, 50 by increasing the Minority and reducing the Majority.

```
balanced_trainData <- SMOTE(Transport~., trainData, perc.over = 250,k =
5,perc.under =150)
str(balanced_trainData)

## 'data.frame':    258 obs. of  9 variables:
##  $ Age      : num  24 34 30 26 36 30 24 30 32 27 ...
##  $ Gender   : num  1 1 1 1 1 1 1 0 1 1 ...
##  $ Engineer : num  1 1 0 0 1 1 1 1 1 0 ...
##  $ MBA      : num  1 0 0 1 1 0 0 0 0 0 ...
##  $ Work.Exp : num  3 12 8 5 18 8 4 6 9 9 ...
##  $ Salary   : num  9.9 16.9 14.6 12.8 28.7 14.6 8.5 15.6 15.5 23.9 ...
##  $ Distance : num  10.9 16.6 6.1 13.9 10.4 7.1 7.5 11.9 5.5 14.1 ...
##  $ license  : num  0 0 0 0 1 0 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...

prop.table(table(balanced_trainData$Transport))

##
##   0   1
## 0.5 0.5

dim(balanced_trainData)

## [1] 258   9
```

Once the data is balanced, we will perform all the Modelling on this dataset.

2.2 Variable Identification

This section holds the Methods that are used during the Analysis of the problem. Below are the Functions that we have used for the Analysis.

- **setwd()**
  Set the working directory to dir.
- **read.csv()**
  Reads a file in table format and creates a data frame from it.
- **head()**
  Returns the first parts of a vector, matrix, table, data frame or function.
- **str()**
  Compactly display the internal Structure of an R object.
- **summary()**
  Summary is a generic function used to produce result summaries of the results of various model fitting functions.
- **sum()**
  Sum returns the sum of all the values present in its arguments.

- **colsum()**
  Form row and column sums and means for numeric arrays.

- **is.null()**
  NULL is often returned by expressions and functions whose value is undefined. is.null returns TRUE if its argument's value is NULL and FALSE otherwise.

- **na.omit()**
  This function removed the rows from the dataset that contains null values.
- **boxplot()**
  It is plotting technique, which is used to identify if there any outliners are present in the data.
- **plot_histogram()**
  Plot histogram for each continuous feature on a single area.
- **cor()**
  cor compute the variance of x and the covariance or correlation of x and y if these are vectors. If x and y are matrices then the covariances (correlations) between the columns of x and the columns of y are computed.
- **corrplot()**
  This is used to plot the correlation matrix for better visualization and presentation.
- **set.seed()**
  set.seed is the recommended way to specify seeds.
- **sample.split()**
  Split data from vector Y into two sets in predefined ratio while preserving relative ratios of different labels in Y. Used to split the data used during classification into train and test subsets.
- **cooks.distance()**
  Cooks Distance is the method to identify the Outliers in the data.
- **glm()**
  glm is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution. Here we will be building the Logistic Model using this.

- **vif()**
  Calculates variance-inflation and generalized variance-inflation factors for linear, generalized linear, and other models.
- **lrtest()**

lrtest is a generic function for carrying out likelihood ratio tests. The default method can be employed for comparing nested (generalized) linear models (see details below).

- **pR2()**
  Compute various pseudo-R2 measures for various GLMs
- **predict()**
  predict is a generic function for predictions from the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.
- **table()**
  table uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.
- **confusionmatrix()**
  Calculate the confusion matrix for the fitted values for a logistic regression model.
- **scale()**
  scale is generic function whose default method centers and/or scales the columns of a numeric matrix.
- **ifelse()**
  ifelse returns a value with the same shape as test which is filled with elements selected from either yes or no depending on whether the element of test is TRUE or FALSE.
- **SMOTE()**
  This function handles unbalanced classification problems using the SMOTE method. Namely, it can generate a new "SMOTEd" data set that addresses the class unbalance problem. Alternatively, it can also run a classification algorithm on this new data set and return the resulting model.
- **cbind()**
  Take a sequence of vector, matrix or data-frame arguments and combine by columns or rows, respectively. These are generic functions with methods for other R classes.
- **traincontrol()**
  Control the computational nuances of the train function
- **train()**
  This function sets up a grid of tuning parameters for a number of classification and regression routines, fits each model and calculates a resampling based performance measure.
- **Bagging()**
  Bagging for classification, regression and survival trees.

- **Xgboost()**
  xgb.train is an advanced interface for training an xgboost model. The xgboostfunction is a simpler wrapper for xgb.train.
- **naïveBayes()**
  Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

3 Conclusion

Once the Data is ready and divided into the Train and Test, also balanced and Scaled we will further be performing the given Models on the data and check which Model performs the best in Train and Test and we will also be performing the Performance measures to validate the model.

- Logistic Model
- K- Nearest Neighbor model
- Naïve Bayes Model
- Bagging Model
- Boosting Model (XG Boosting)

3.1 Logistic Model

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability between 0 and 1 and the sum adding to one.

3.1.1 **Building Logistic Model on Train Data**

```r
#Performing Logistic Model on Train data.

logit <- glm(balanced_trainData$Transport~., data = balanced_trainData,
family = "binomial")
summary(logit)

##
## Call:
## glm(formula = balanced_trainData$Transport ~ ., family = "binomial",
##     data = balanced_trainData)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.49329  -0.01478   0.00004   0.02165   2.04785
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -74.008422  15.472147  -4.783 1.72e-06 ***
## Age           2.355327   0.521234   4.519 6.22e-06 ***
## Gender       -0.851113   0.914716  -0.930 0.352129
## Engineer      1.506589   1.123968   1.340 0.180109
## MBA          -2.871473   1.016811  -2.824 0.004743 **
## Work.Exp     -1.003964   0.314222  -3.195 0.001398 **
## Salary        0.004538   0.073320   0.062 0.950650
## Distance      0.692678   0.181160   3.824 0.000132 ***
## license       2.443810   1.019817   2.396 0.016560 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 476.885  on 343  degrees of freedom
## Residual deviance:  64.557  on 335  degrees of freedom
## AIC: 82.557
##
## Number of Fisher Scoring iterations: 9
```

The first Model has been created with all the Independent Variables and we identified that there were only a few variables that are significant to the model.

We will further be checking the Multicollinearity in the variables using the Variance Inflation Factors aka VIF.

```r
vif(logit)

##    Gender  Engineer       MBA   license       Age  Work.Exp    Salary
##  3.833214  1.528132  1.437715  4.406819 27.341991 42.798370  7.509137
##  Distance
##  3.249967
```

We found that the variables “Work.Exp” is highly correlated to “Age” and “Salary” and hence needs to be treated.

We refine the Logistic model by removing the Work.Exp.

Model Refinement

We create the Logistic Model by Removing the Work Exp.

```
balanced_trainData1 <- balanced_trainData[,-7]
str(balanced_trainData1)

## 'data.frame':    344 obs. of  8 variables:
##  $ Age      : num  30 27 30 24 30 25 29 20 27 30 ...
##  $ Gender   : num  0 1 1 1 0 1 1 0 0 0 ...
##  $ Engineer : num  1 1 1 1 0 1 1 0 1 0 ...
##  $ MBA      : num  0 0 0 1 0 0 0 1 0 0 ...
##  $ Work.Exp : num  8 4 6 0 6 3 9 1 5 6 ...
##  $ Salary   : num  14.7 13.5 15.8 7.9 15.6 10.7 23.8 8.5 12.8 15.6 ...
##  $ license  : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...

testData1 <- testData[,-7]
str(testData1)

## 'data.frame':    133 obs. of  8 variables:
##  $ Age      : int  23 28 27 26 25 25 23 26 24 30 ...
##  $ Gender   : num  0 0 1 0 0 1 1 1 1 1 ...
##  $ Engineer : int  1 1 1 1 1 1 0 1 1 0 ...
##  $ MBA      : int  0 1 0 0 0 1 0 0 0 0 ...
##  $ Work.Exp : int  4 5 4 3 4 4 2 5 6 8 ...
##  $ Salary   : num  8.3 13.4 13.4 10.5 11.5 11.5 8.6 11.4 10.6 14.6 ...
##  $ license  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```r
logit_refined <- glm(Transport~.,
                     data = balanced_trainData1, family = "binomial")
summary(logit)
```

```
##
## Call:
## glm(formula = balanced_trainData$Transport ~ ., family = "binomial",
##     data = balanced_trainData)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.49329  -0.01478   0.00004   0.02165   2.04785
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -74.008422  15.472147  -4.783 1.72e-06 ***
## Age           2.355327   0.521234   4.519 6.22e-06 ***
## Gender       -0.851113   0.914716  -0.930 0.352129
## Engineer      1.506589   1.123968   1.340 0.180109
## MBA          -2.871473   1.016811  -2.824 0.004743 **
## Work.Exp     -1.003964   0.314222  -3.195 0.001398 **
## Salary        0.004538   0.073320   0.062 0.950650
## Distance      0.692678   0.181160   3.824 0.000132 ***
## license       2.443810   1.019817   2.396 0.016560 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 476.885  on 343  degrees of freedom
## Residual deviance:  64.557  on 335  degrees of freedom
## AIC: 82.557
##
## Number of Fisher Scoring iterations: 9
```

Validating the VIF on the refined Model

```r
vif(logit_refined)
```

```
##   Gender Engineer      MBA  license      Age   Salary Distance
## 1.913377 1.187279 1.176344 1.620455 1.680925 1.578101 1.203729
```

After removing Work.Exp, the VIF values are Scaled to 1 hence the Multicollinearity has been handled.

Further we will be performing all the Models on the new data i.e. the data with the handled Multicollinearity.

Performing Likelihood Ratio Test

```
# Calculating Likelihood Test
logit_likelihood <- lrtest(logit_refined)
logit_likelihood

## Likelihood ratio test
##
## Model 1: Transport ~ Gender + Engineer + MBA + license + Age + Salary +
##     Distance
## Model 2: Transport ~ 1
##   #Df   LogLik Df  Chisq Pr(>Chisq)
## 1   8  -41.111
## 2   1 -178.832 -7 275.44  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

After performing the Likelihood Ratio test on the Model, we receive the P Value as 2.2 e-16 which is extremely smaller than the alpha, hence the model is significant.

Calculating McFadden Psudo R-sq

```
# Calculating Psudo R Sq

logit_rsq <- pR2(logit_refined)
logit_rsq

##          llh      llhNull           G2     McFadden         r2ML
##  -45.7768485 -238.4426301  385.3315632    0.8080173    0.6737691
##         r2CU
##    0.8983588
```

The value for McFadden Psudo R Sq is 0.808, which is not in the Range of 0.2 – 0.4 making the Model an Overfit Model.

Checking for the Variable Importance

```
> varImp(logit_refined)
           Overall
Gender   3.294401
Engineer 2.044619
MBA      1.805957
license  3.072545
Age      3.950602
Salary   1.089769
Distance 2.059508
```

By performing variable Importance, we Identified that Gender Age and License plays an Important role and are more significant than the other Models.

3.1.2 Predicting Logistic Model on Train

Going forward, we will be performing the Prediction on Train Data to check the accuracy of the Model built.

```
# Predicting values on Train Data.

pred_logit_train <- predict(logit_refined, balanced_trainData)

pred_logit_train_class <- ifelse(pred_logit_train <.5,0,1)
head(pred_logit_train_class)

## 101  12 361 130 247 212
##   0   0   0   0   0   0

prop.table(table(pred_logit_train_class))

## pred_logit_train_class
##         0         1
## 0.5203488 0.4796512
```

Performing Confusion Matrix on the Predicted Values

```
# Creating Performance Matrix on Train

pred_logit_train_class<- as.factor(pred_logit_train_class)
caret::confusionMatrix(pred_logit_train_class,balanced_trainData$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 163  16
##          1   9 156
##
##                Accuracy : 0.9273
##                  95% CI : (0.8946, 0.9524)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8547
##
##  Mcnemar's Test P-Value : 0.2301
##
##             Sensitivity : 0.9477
##             Specificity : 0.9070
##          Pos Pred Value : 0.9106
##          Neg Pred Value : 0.9455
##              Prevalence : 0.5000
##          Detection Rate : 0.4738
##    Detection Prevalence : 0.5203
##       Balanced Accuracy : 0.9273
##
##        'Positive' Class : 0
##

# Performing Prediction on Test Data
```

As per the above Confusion Matrix, we have achieved the accuracy of 92.7% with a Sensitivity of 94.7% and Specificity of 90.7%.

Calculating the Base Line Accuracy on Train Data

```
# Calculating Baseline
prop.table(table(balanced_trainData$Transport))

##
##   0   1
## 0.5 0.5
```

The base line model explains the error rate that can occur if all the observations are predicted False. In case of Train Data, we achieved the Base Line Accuracy of 50%.

3.1.3 Predicting Logistic Model on Test

Now we will be predicting the values on the unseen dataset i.e. Test Data.

```
# Performing Prediction on Test Data

pred_logit_test <- predict(logit_refined, testData1, type = "response")
pred_logit_test_class <- ifelse(pred_logit_test <.5,0,1)
head(pred_logit_test_class)

##  2  4  5  8 11 16
##  0  0  0  0  0  0
```

Performing Confusion Matrix on the Predicted Values

```
# Creating performance Matrix on Test

pred_logit_test_class <- as.factor(pred_logit_test_class)
caret::confusionMatrix(pred_logit_test_class,testData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 111   3
##          1   4  15
##
##                Accuracy : 0.9474
##                  95% CI : (0.8946, 0.9786)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.0017
##
##                   Kappa : 0.7803
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.9652
##             Specificity : 0.8333
##          Pos Pred Value : 0.9737
##          Neg Pred Value : 0.7895
##              Prevalence : 0.8647
##          Detection Rate : 0.8346
##    Detection Prevalence : 0.8571
##       Balanced Accuracy : 0.8993
##
##        'Positive' Class : 0
##
```

As per the above Confusion Matrix, we have achieved the accuracy of 94.7% with a Sensitivity of 96.5% and Specificity of 83.3%.

3.2 K-Nearest Neighbor Model

k-nearest neighbor classification for test set from training set. For each row of the test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

3.2.1 Building KNN Model on Train

```
# Performing K- Nearest Neighbor
ctrl <- trainControl(method = "cv", number = 3)
knnModel <- train(Transport~.,data = balanced_trainData1,method = "knn",
                  trControl = ctrl,
                  tuneLength = 10)
knnModel$bestTune

##   k
## 3 9

summary(knnModel)

##             Length Class      Mode
## learn       2      -none-     list
## k           1      -none-     numeric
## theDots     0      -none-     list
## xNames      7      -none-     character
## problemType 1      -none-     character
## tuneValue   1      data.frame list
## obsLevels   2      -none-     character
## param       0      -none-     list
```

We achieve the maximum Accuracy at k = 9.

Now when we have built the KNN Model on Train data we will further be predicting the Model on Train data and build the Confusion Matrix to check for the Accuracy of the Model.

3.2.2 Performing Prediction and Creating Confusion Matrix on Train Data

```
# Predicting Value on Train data

pred_knn_Train <- predict(knnModel,balanced_trainData1)

# Creating Perfromance Matrix

caret::confusionMatrix(pred_knn_Train,balanced_trainData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 158   4
##          1  14 168
##
##                Accuracy : 0.9477
##                  95% CI : (0.9186, 0.9687)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8953
##
##  Mcnemar's Test P-Value : 0.03389
##
##             Sensitivity : 0.9186
##             Specificity : 0.9767
##          Pos Pred Value : 0.9753
##          Neg Pred Value : 0.9231
##              Prevalence : 0.5000
##          Detection Rate : 0.4593
##    Detection Prevalence : 0.4709
##       Balanced Accuracy : 0.9477
##
##        'Positive' Class : 0
##
```

From the above we've gained the Accuracy of 94.7% and Sensitivity of 91.8% and Specificity of 97.6%.

3.2.3 Performing Prediction and Creating Confusion Matrix on Test Data

```
# Predicting Values on Test data.

pred_knn_test <- predict(knnModel, testData1)

caret::confusionMatrix(pred_knn_test, testData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 110   2
##          1   5  16
##
##                Accuracy : 0.9474
##                  95% CI : (0.8946, 0.9786)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.0017
##
##                   Kappa : 0.7899
##
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.9565
##             Specificity : 0.8889
##          Pos Pred Value : 0.9821
##          Neg Pred Value : 0.7619
##              Prevalence : 0.8647
##          Detection Rate : 0.8271
##    Detection Prevalence : 0.8421
##       Balanced Accuracy : 0.9227
##
##        'Positive' Class : 0
##
```

From the prediction on Test Data, we achieve the accuracy of 94.7% with a Sensitivity of 95.6% and Specificity of 88.8%.

3.3 Naïve Bayes Model

A Naive Bayes classifier is a probabilistic machine learning model that’s used for classification task. The crux of the classifier is based on the Bayes theorem.

3.3.1 Building the Naïve Bayes Model on Train Data

```
# Performing Naive Bayes Model

NBModel <- naiveBayes(Transport~., data = balanced_trainData1)
summary(NBModel)

##           Length Class  Mode
## apriori   2      table  numeric
## tables    7      -none- list
## levels    2      -none- character
## isnumeric 7      -none- logical
## call      4      -none- call
```

3.3.2 Predicting the Model and Creating Performance Matrix on Train Data

```
# Prediction on Train data.

pred_nb_train <- predict(NBModel, balanced_trainData1)


# Creating Performance Matrix on Traindata
caret::confusionMatrix(pred_nb_train, balanced_trainData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 163  20
##          1   9 152
##
##                Accuracy : 0.9157
##                  95% CI : (0.8812, 0.9428)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8314
##
##  Mcnemar's Test P-Value : 0.06332
##
##             Sensitivity : 0.9477
##             Specificity : 0.8837
##          Pos Pred Value : 0.8907
##          Neg Pred Value : 0.9441
##              Prevalence : 0.5000
##          Detection Rate : 0.4738
##    Detection Prevalence : 0.5320
##       Balanced Accuracy : 0.9157
##
##        'Positive' Class : 0
##
```

From the above Model, we achieve the Accuracy of 91.5% with a Sensitivity of 94.8% and Specificity of 88.3%.

3.3.3 Predicting Model and Performing Confusion Matrix on Test Data

Now we will be predicting the values on the unseen data after performing on Train data.

```
# Prediction on Test data.

pred_nb_test <- predict(NBModel, testData1)

# Creating Performance Matrix on Test Data
caret::confusionMatrix(pred_nb_test, testData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 113   4
##          1   2  14
##
##                Accuracy : 0.9549
##                  95% CI : (0.9044, 0.9833)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.0005588
##
##                   Kappa : 0.7978
##
##  Mcnemar's Test P-Value : 0.6830914
##
##             Sensitivity : 0.9826
##             Specificity : 0.7778
##          Pos Pred Value : 0.9658
##          Neg Pred Value : 0.8750
##              Prevalence : 0.8647
##          Detection Rate : 0.8496
##    Detection Prevalence : 0.8797
##       Balanced Accuracy : 0.8802
##
##        'Positive' Class : 0
##
```

From the above Prediction, we achieve the Accuracy of 95.4% with a Sensitivity of 98.2% and Specificity of 77.7%

3.4 Bagging Model

Bagging (stands for Bootstrap Aggregating) is a way to decrease the variance of your prediction by generating additional data for training from your original dataset using combinations with repetitions to produce multisets of the same cardinality/size as your original data. By increasing the size of your training set you can't improve the model predictive force, but just decrease the variance, narrowly tuning the prediction to expected outcome.

3.4.1 Performing Bagging on the Train dataset.

```
# Performing Bagging Model

bagModel <- bagging(as.numeric(Transport)~.,data = balanced_trainData1,
                    control = rpart.control(maxdepth = 5, minsplit = 3))

summary(bagModel)

##         Length Class      Mode
## y       344    -none-     numeric
## X         7    data.frame list
## mtrees   25    -none-     list
## OOB       1    -none-     logical
## comb      1    -none-     logical
## call      4    -none-     call
```

3.4.2 Performing Prediction on Train Dataset and Creating Confusion Matrix.

```
#Predicting Model on Train data
pred_bag_train <- predict(bagModel, data= balanced_trainData1)
pred_bag_train1 <- ifelse(pred_bag_train<0.5,0,1)
pred_bag_train1 <- as.factor(pred_bag_train1)

caret::confusionMatrix(pred_bag_train1,balanced_trainData1$Transport)

## Warning in confusionMatrix.default(pred_bag_train1,
## balanced_trainData1$Transport): Levels are not in the same order for
## reference and data. Refactoring data to match.

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1 172 172
##
##                Accuracy : 0.5
##                  95% CI : (0.4459, 0.5541)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.5215
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0
##             Specificity : 1.0
##          Pos Pred Value : NaN
##          Neg Pred Value : 0.5
##              Prevalence : 0.5
##          Detection Rate : 0.0
##    Detection Prevalence : 0.0
##       Balanced Accuracy : 0.5
##
##        'Positive' Class : 0
```

From the above Prediction, we achieve the Accuracy of 0.5% with a Sensitivity of 0% and Specificity of 100%

3.4.3 Predicting values on Test data and performing Confusion Matrix

Now we will be predicting the values on the unseen data after performing on Train data.

```
# Predicting Model on Test data

pred_bag_test <- predict(bagModel,testData1)
pred_bag_test1 <- ifelse(pred_bag_test<0.4,0,1)
pred_bag_test1 <- as.factor(pred_bag_test1)

caret::confusionMatrix(pred_bag_test1,testData1$Transport)

## Warning in confusionMatrix.default(pred_bag_test1, testData1$Transport):
## Levels are not in the same order for reference and data. Refactoring data
## to match.

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1 115  18
##
##                Accuracy : 0.1353
##                  95% CI : (0.0822, 0.2054)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.1353
##              Prevalence : 0.8647
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
```

From the above Prediction, we achieve the Accuracy of 13.5% with a Sensitivity of 0% and Specificity of 100%

In the above code, we found that the Bagging is not performing well on the Train and Test data.

We will be creating a new Model using a different approach.

3.4.4 Recreating the Model using a different Approach and Predicting on Train Data

```
# Performing Bagging using different Approach
bagModel2 <- train(Transport~.,data = balanced_trainData1,
                   method = "treebag",
                   trControl = trainControl(method = "cv", number = 10),
                   nbagg= 200,
                   control = rpart.control(minsplit = 2,cp=0))
```

```
pred_bag2_train <- predict(bagModel2, data= balanced_trainData1)
#pred_bag2_train1 <- ifelse(pred_bag2_train<0.5,0,1)
pred_bag2_train1 <- as.factor(pred_bag2_train)

caret::confusionMatrix(pred_bag2_train1,balanced_trainData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 172   0
##          1   0 172
##
##                Accuracy : 1
##                  95% CI : (0.9893, 1)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : 0
##
```

From the prediction on new model, we found that the Model is predicting 100% accurately on the Train data with a Sensitivity of 100% and Specificity of 100%.

3.4.5 Predicting values on Test Data and Creating the Confusion Matrix

```
# Predicting Model on Test data

pred_bag2_test <- predict(bagModel2,testData1)
#pred_bag2_test1 <- ifelse(pred_bag_test<0.4,0,1)
pred_bag2_test1 <- as.factor(pred_bag2_test)

caret::confusionMatrix(pred_bag2_test1,testData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 113   2
##          1   2  16
##
##                Accuracy : 0.9699
##                  95% CI : (0.9248, 0.9917)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 3.661e-05
##
##                   Kappa : 0.8715
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9826
##             Specificity : 0.8889
##          Pos Pred Value : 0.9826
##          Neg Pred Value : 0.8889
##              Prevalence : 0.8647
##          Detection Rate : 0.8496
##    Detection Prevalence : 0.8647
##       Balanced Accuracy : 0.9357
##
##        'Positive' Class : 0
##
```

In Test data, the model is performing well with an Accuracy of 96.9% with a Sensitivity of 98.2% and Specificity of 88.8%.

3.5 Boosting Model (XG Boosting)

Boosting is a two-step approach, where one first uses subsets of the original data to produce a series of averagely performing models and then "boosts" their performance by combining them together using a particular cost function (=majority vote). Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models: every new subsets contains the elements that were (likely to be) misclassified by previous models.

3.5.1 Performing Boosting Model on the Train Data

Before creating the Model, we need to convert the dataset to the Matrix as the data consumed in this model is in the form of Matrix.

Features_train: A matrix with all the Independent Variables built on the Train Dataset.

Label_train: A matrix with the Dependent variable built on the test Dataset.

Features_test: A matrix with all the Independent Variables built on the Test Dataset.

```
features_train <- as.matrix(balanced_trainData1[,1:7])
#str(features_train)
label_train <- as.matrix(balanced_trainData1[,8])
features_test <- as.matrix(testData1[,1:7])
str(features_test)

##  num [1:133, 1:7] 23 28 27 26 25 25 23 26 24 30 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:133] "2" "4" "5" "8" ...
##   ..$ : chr [1:7] "Age" "Gender" "Engineer" "MBA" ...

xgbModel <- xgboost(
  data = features_train,
  label = label_train,
  eta = 1,
  max_depth = 100,
  min_child_weight = 3,
  nrounds = 1000,
  nfold = 10,
  objective = "binary:logistic",
  verbose = 0,
  early_stopping_rounds = 10)

summary(xgbModel)

##                 Length Class              Mode
## handle             1   xgb.Booster.handle externalptr
## raw             9001   -none-             raw
## best_iteration     1   -none-             numeric
## best_ntreelimit    1   -none-             numeric
## best_score         1   -none-             numeric
## niter              1   -none-             numeric
## evaluation_log     2   data.table         list
## call              18   -none-             call
## params             6   -none-             list
## callbacks          2   -none-             list
## feature_names      7   -none-             character
## nfeatures          1   -none-             numeric
```

Once the model is created, we will be predicting the Model on Train and Test data to check it's performance.

3.5.2 Predicting values on the Train data and creating the Confusion Matrix

```
# Performing Prediction on Train data.
pred_xgb_train <- predict(xgbModel, newdata = features_train)
pred_xgb_train1 <- ifelse(pred_xgb_train<.5,0,1)
pred_xgb_train1<- as.factor(pred_xgb_train1)
caret::confusionMatrix(pred_xgb_train1, balanced_trainData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 171   1
##          1   1 171
##
##                Accuracy : 0.9942
##                  95% CI : (0.9792, 0.9993)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9884
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9942
##             Specificity : 0.9942
##          Pos Pred Value : 0.9942
##          Neg Pred Value : 0.9942
##              Prevalence : 0.5000
##          Detection Rate : 0.4971
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9942
##
##        'Positive' Class : 0
##
```

In the above prediction on Train data, we have achieved the Accuracy of 99.4% with a Sensitivity of 99.4% and Specificity of 99.4%.

3.5.3 Prediction on Test Data and creating the Confusion Matrix

```
# Performing Prediction on Test data.
pred_xgb_test <- predict(xgbModel,newdata = features_test)

pred_xgb_test1 <- ifelse(pred_xgb_test<.5,0,1)
pred_xgb_test1<- as.factor(pred_xgb_test1)
caret::confusionMatrix(pred_xgb_test1, testData1$Transport)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction   0   1
##          0 114   2
##          1   1  16
##
##                Accuracy : 0.9774
##                  95% CI : (0.9355, 0.9953)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 6.803e-06
##
##                   Kappa : 0.9013
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9913
##             Specificity : 0.8889
##          Pos Pred Value : 0.9828
##          Neg Pred Value : 0.9412
##              Prevalence : 0.8647
##          Detection Rate : 0.8571
##    Detection Prevalence : 0.8722
##       Balanced Accuracy : 0.9401
##
##        'Positive' Class : 0
##
```

We achieved the Accuracy of 97.7.67% with a Sensitivity of 99.1% and Specificity of 88.8% on the above model when predicted on Test.

We will further be refining the Model using a different approach to create the Model to check if the Model perform any better on Train and Test.

3.5.4 **Recreating the Model on Train using different Approach and predicting the Values.**

```
# Performing XGBoosting using different Approach

carsxgb <- train(Transport~.,balanced_trainData1,
                 trControl = trainControl("cv", number = 2),method =
"xgbTree")


pred_xgb2_train <- predict(carsxgb, balanced_trainData1)
pred_xgb2_train <- as.factor(pred_xgb2_train)
caret::confusionMatrix(pred_xgb2_train, balanced_trainData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 172   0
##          1   0 172
##
##                Accuracy : 1
##                  95% CI : (0.9893, 1)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : 0
##
```

In the refined Model, we achieved the Accuracy of 100% with Sensitivity and Specificity of 100% on the Train dataset.

3.5.5 **Predicting the value on Test data and Creating the Confusion Matrix**

```
pred_xgb2_test <- predict(carsxgb, testData1)
pred_xgb2_test <- as.factor(pred_xgb2_test)
caret::confusionMatrix(pred_xgb2_test, testData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 113   1
##          1   2  17
##
##                Accuracy : 0.9774
##                  95% CI : (0.9355, 0.9953)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 6.803e-06
##
##                   Kappa : 0.9058
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9826
##             Specificity : 0.9444
##          Pos Pred Value : 0.9912
##          Neg Pred Value : 0.8947
##              Prevalence : 0.8647
##          Detection Rate : 0.8496
##    Detection Prevalence : 0.8571
##       Balanced Accuracy : 0.9635
##
##        'Positive' Class : 0
##
```

From the prediction on Test data, we got the accuracy of 97.7% with the Sensitivity of 98.2% and a specificity of 94.4%

3.6 Actionable Insights and Recommendations

Now, when we have built the Logistic Model, KNN Model, Naïve Bayes Model, Bagging Model and Boosting Model. Below is the Summary of all the Models and their Performance measures in a matrix format.

	Logistic Model		KNN		Naïve Bayes		Bagging		Boosting	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Accuracy	92.7	94.7	94.7	94.7	91.5	95.4	100	96.99	100	97.7
Sensitivity	94.7	96.5	91.8	95.6	94.7	98.2	100	98.26	100	98.2
Specificity	90.7	83.3	97.6	88.8	88.3	77.7	100	88.8	100	94.4

Considering the above Confusion Matrix created on all the Models, Boosting achieves the highest Accuracy of 100% in Train and 97.7% in Test.

In Boosting, Sensitivity for Train is achieved as 100% and 98.2% in Test, Specificity for Training is achieved as 100% while in Test it is 94.4% hence making it an efficient and most fit Model out of all the five Models.

At the end, we concluded that Salary, Age and Gender are the most Significant Variables making the high impact on the use of Car as a mode of Commute to Office. Hence the Employees with Higher Salary are Most likely to use Car. Also, Employees with the higher Experience are likely to use Car as the mode of Transport. From the data we also found that Men are more likely to use Car as the mode of Transport.

As per the performance measured, Boosting proves to be the most Significant and Fit Model.

4 Appendix A – Source Code

```
# Deploying necessary Libraries to the Code.

 library(corrplot)

##   corrplot   0.84
loaded

library(DataExplorer)
library(ggplot2)
library(car)

##    Loading    required    package:    carData

library(caret)

## Loading required package: lattice

library(caTools)
library(psych)

##

## Attaching package: 'psych'

## The following object is masked from 'package:car':

##

##     logit
```

```
## The following objects are masked from 'package:ggplot2':
##
##                    %+%,    alpha
library(ggbiplot)
## Loading required package: plyr
## Loading required package: scales
##
## Attaching package: 'scales'
## The following objects are masked from 'package:psych':
##
##     alpha, rescale
## Loading required package: grid
library(ipred)
library(e1071)
library(rpart)
library(DMwR)
## Warning: package 'DMwR' was built under R version 3.6.2
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
##
## Attaching package: 'DMwR'
## The following object is masked from 'package:plyr':
##
##                      join
library(xgboost)
## Warning: package 'xgboost' was built under R version 3.6.2 library(blorr)
## Warning: package 'blorr' was built under R version 3.6.2 library(lmtest)
## Warning: package 'lmtest' was built under R version 3.6.2
## Loading required package: zoo
## Warning: package 'zoo' was built under R version 3.6.2
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
library(pscl)
```

```
## Warning: package 'pscl' was built under R version 3.4.4
```

```
## Classes and Methods for R developed in the
## Political Science Computational Laboratory
## Department of Political Science
## Stanford University
## Simon Jackman
## hurdle and zeroinfl functions by Achim Zeileis
```

```r
# Setting the Working Directory.

setwd("D:/Great Learning/Machine Learning/Project 5")

# Reading the dataset.

cars<- read.csv("Cars_edited.csv")

str(cars)
```

```
## 'data.frame':    444 obs. of  9 variables:
##  $ Age      : int  28 23 29 28 27 26 28 26 22 27 ...
##  $ Gender   : Factor w/ 2 levels "Female","Male": 2 1 2 1 2 2 2 1 2 2 ...
##  $ Engineer : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ MBA      : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Work.Exp : int  4 4 7 5 4 4 5 3 1 4 ...
##  $ Salary   : num  14.3 8.3 13.4 13.4 13.4 12.3 14.4 10.5 7.5 13.5 ...
##  $ Distance : num  3.2 3.3 4.1 4.5 4.6 4.8 5.1 5.1 5.1 5.2 ...
##  $ license  : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ Transport: Factor w/ 3 levels "2Wheeler","Car",..: 3 3 3 3 3 3 1 3 3 3 ...
```

```r
summary(cars)
```

```
##       Age            Gender       Engineer           MBA        
##  Min.   :18.00   Female:128   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:25.00   Male  :316   1st Qu.:1.0000   1st Qu.:0.0000  
##  Median :27.00                Median :1.0000   Median :0.0000  
##  Mean   :27.75                Mean   :0.7545   Mean   :0.2528  
##  3rd Qu.:30.00                3rd Qu.:1.0000   3rd Qu.:1.0000  
##  Max.   :43.00                Max.   :1.0000   Max.   :1.0000  
##                                                NA's   :1       
##     Work.Exp        Salary         Distance        license      
##  Min.   : 0.0   Min.   : 6.50   Min.   : 3.20   Min.   :0.0000  
##  1st Qu.: 3.0   1st Qu.: 9.80   1st Qu.: 8.80   1st Qu.:0.0000  
##  Median : 5.0   Median :13.60   Median :11.00   Median :0.0000  
##  Mean   : 6.3   Mean   :16.24   Mean   :11.32   Mean   :0.2342  
##  3rd Qu.: 8.0   3rd Qu.:15.72   3rd Qu.:13.43   3rd Qu.:0.0000  
```

```
##  Max.   :24.0   Max.   :57.00   Max.   :23.40   Max.   :1.0000
##
##            Transport
##  2Wheeler        : 83
##  Car             : 61
##  Public Transport:300
##
##
##
##
```

```
dim(cars)
```

```
## [1] 444   9
```

```
cars<- na.omit(cars)
dim(cars)
```

```
## [1] 443   9
```

```
names(cars)
```

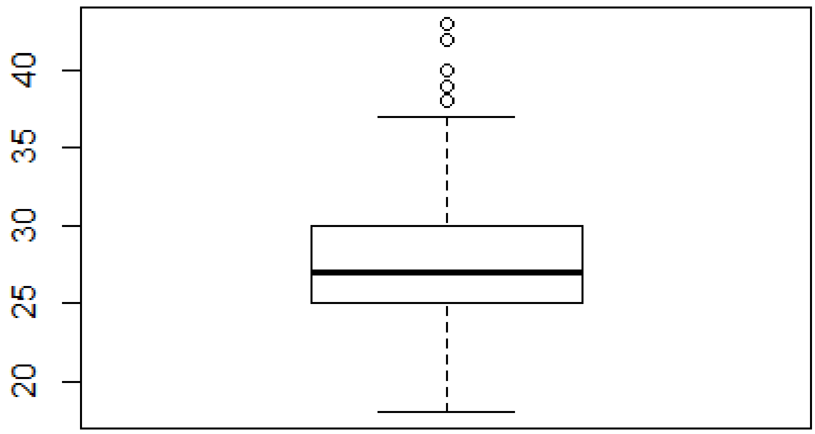
```
## [1] "Age"       "Gender"    "Engineer"  "MBA"       "Work.Exp"  "Salary"
## [7] "Distance"  "license"   "Transport"
```

```
# Converting Dependent variables to 0 and 1
cars$Transport <- ifelse(cars$Transport == "Car",1,0)
cars$Gender <- ifelse(cars$Gender =="Female",0,1)
cars$Transport <- as.factor(cars$Transport)

str(cars)
```

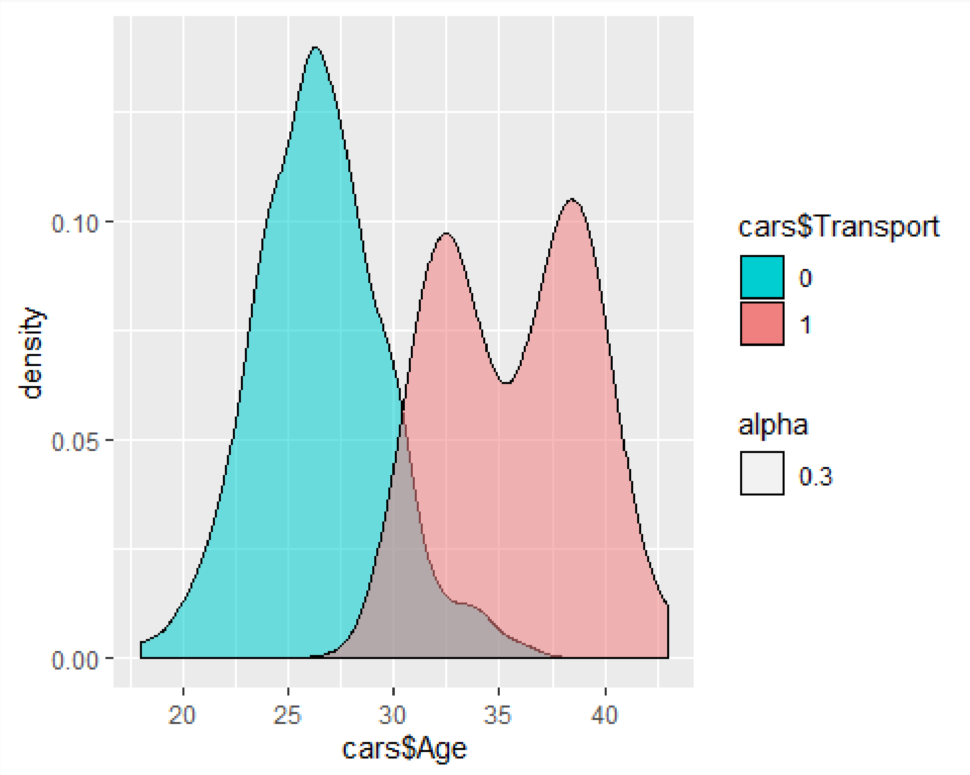
```
## 'data.frame':    443 obs. of  9 variables:
##  $ Age      : int  28 23 29 28 27 26 28 26 22 27 ...
##  $ Gender   : num  1 0 1 0 1 1 1 0 1 1 ...
##  $ Engineer : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ MBA      : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Work.Exp : int  4 4 7 5 4 4 5 3 1 4 ...
##  $ Salary   : num  14.3 8.3 13.4 13.4 13.4 12.3 14.4 10.5 7.5 13.5 ...
##  $ Distance : num  3.2 3.3 4.1 4.5 4.6 4.8 5.1 5.1 5.1 5.2 ...
##  $ license  : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "na.action")=Class 'omit'  Named int 145
##   .. ..- attr(*, "names")= chr "145"
```

```
plot_histogram(cars)
```

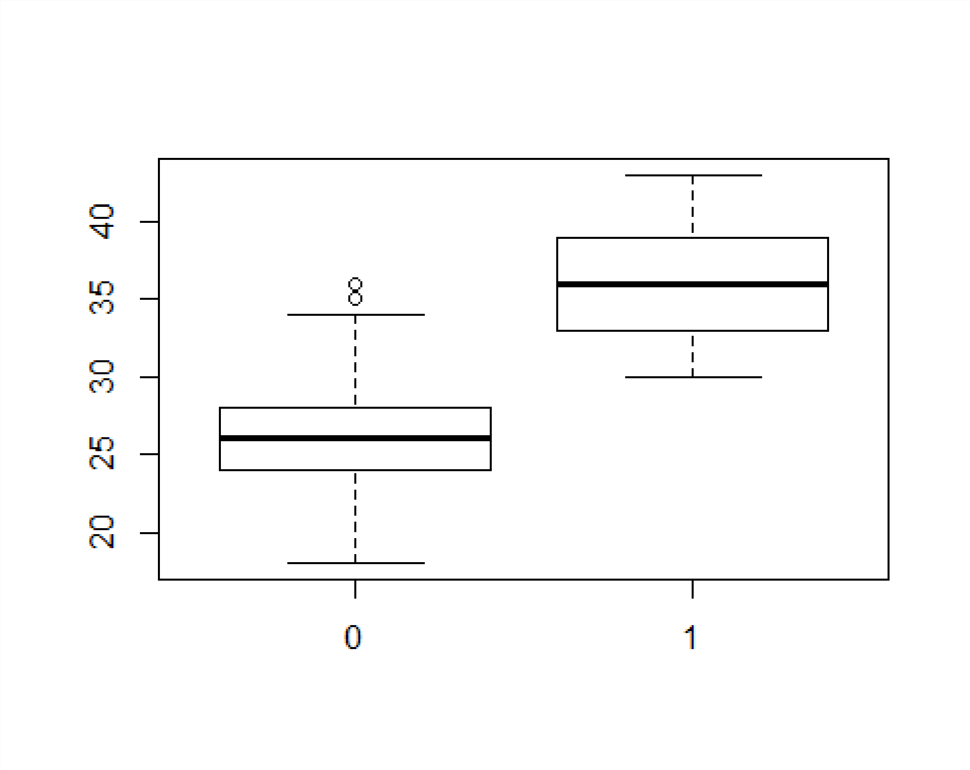

```
boxplot(cars$Age)
```



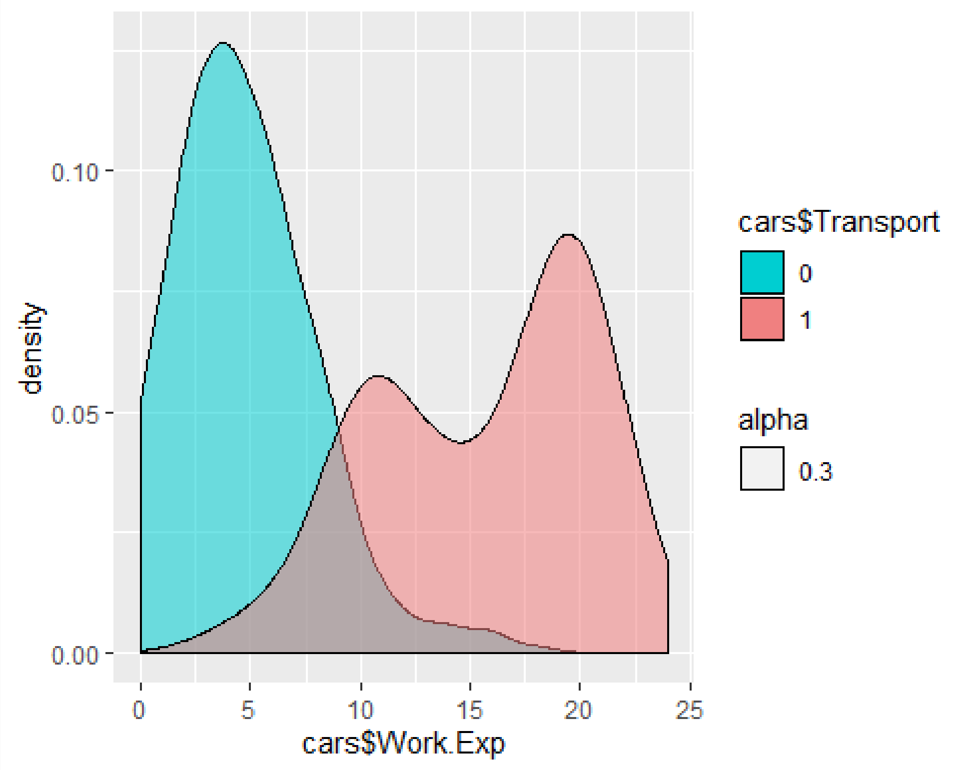
```
ggplot(cars, aes(x=cars$Age)) +
  geom_density(aes(fill =cars$Transport, alpha = 0.3)) +
  scale_color_manual(values = c("#868686FF", "#EFC000FF")) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```



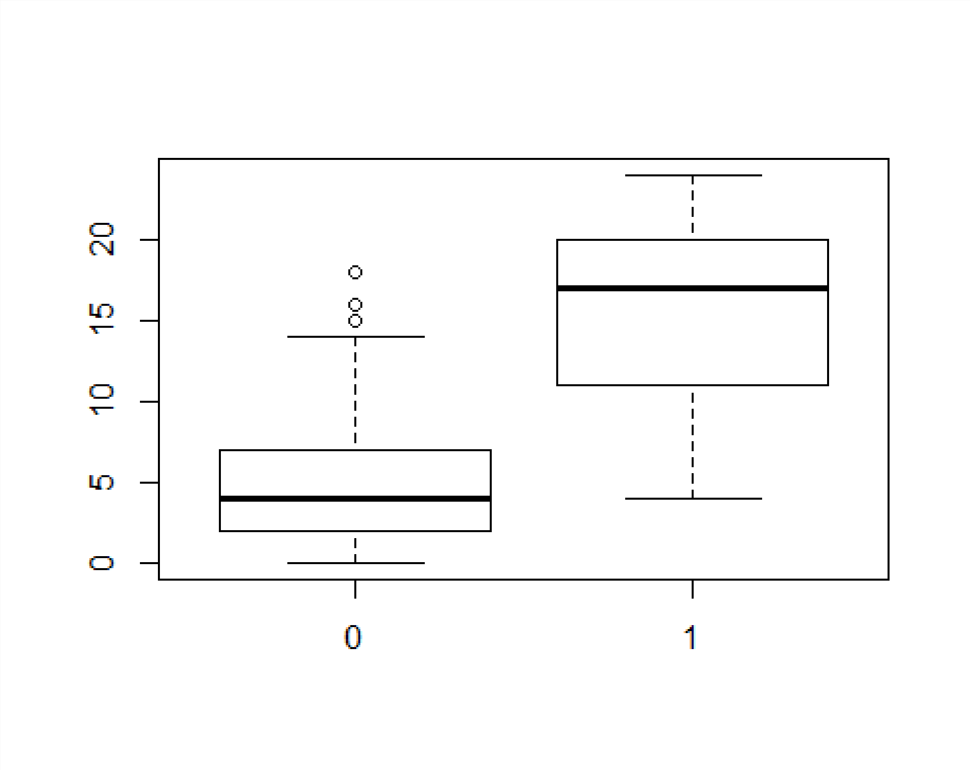

```
boxplot(cars$Age~ cars$Transport)
```



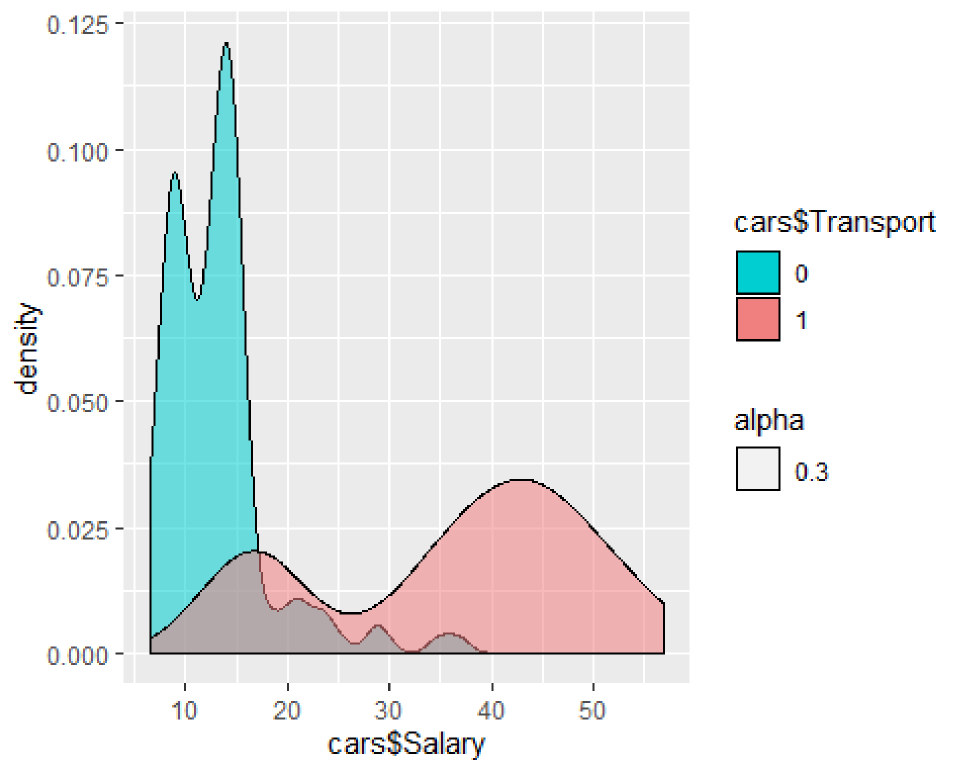

```
ggplot(cars, aes(x=cars$Work.Exp)) +
  geom_density(aes(fill =cars$Transport, alpha = 0.3)) +
  scale_color_manual(values = c("#868686FF", "#EFC000FF")) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```



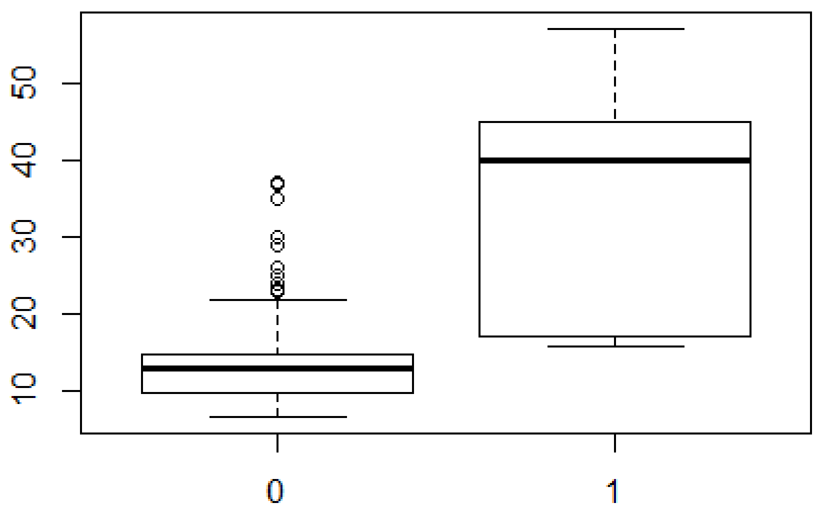

```
boxplot(cars$Work.Exp~ cars$Transport)
```




```
ggplot(cars, aes(x=cars$Salary)) +
  geom_density(aes(fill =cars$Transport, alpha = 0.3)) +
  scale_color_manual(values = c("#868686FF", "#EFC000FF")) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```

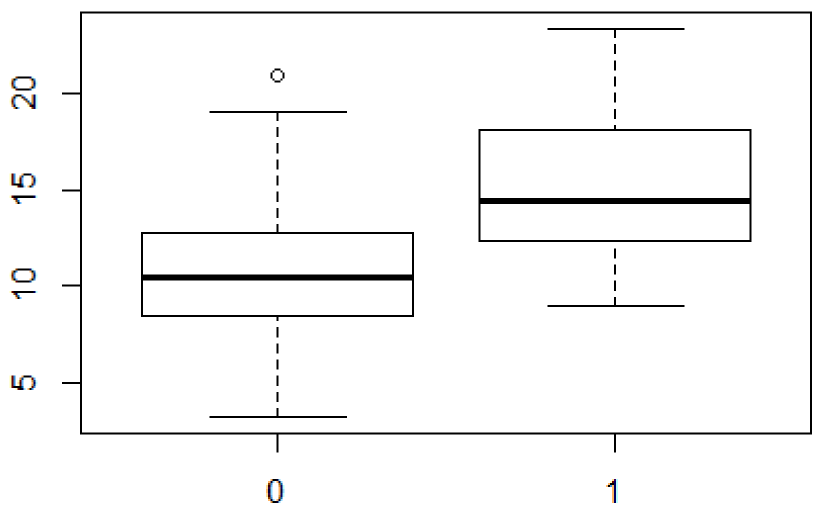



```
boxplot(cars$Salary~ cars$Transport)
```

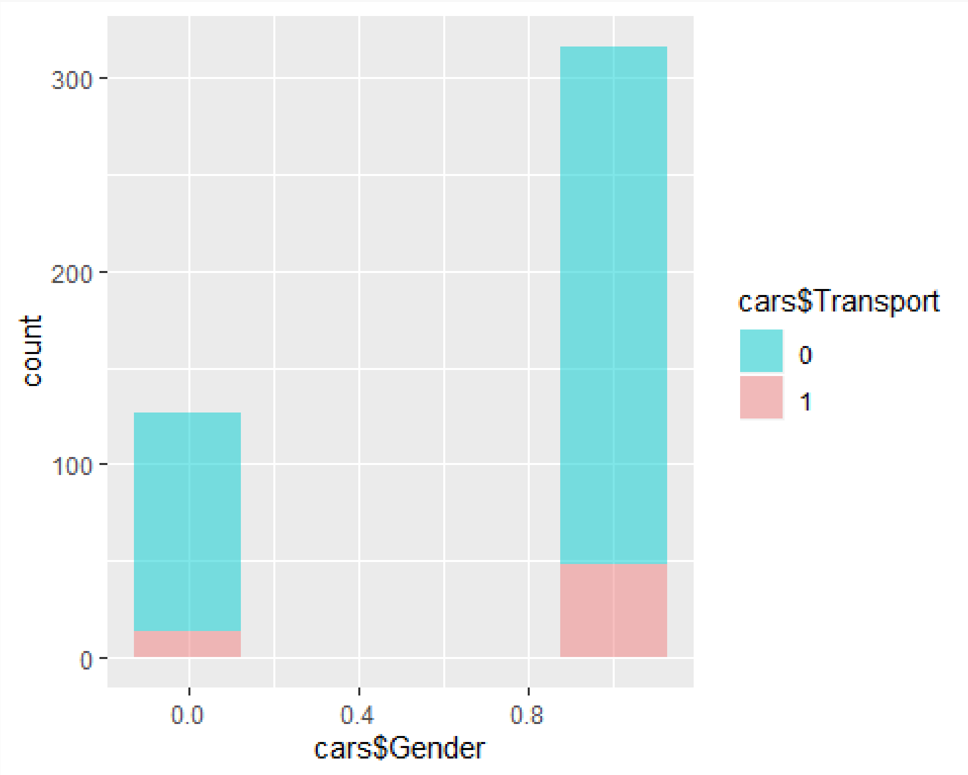



```
ggplot(cars, aes(x=cars$Distance)) +
  geom_density(aes(fill =cars$Transport, alpha = 0.3)) +
  scale_color_manual(values = c("#868686FF", "#EFC000FF")) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```

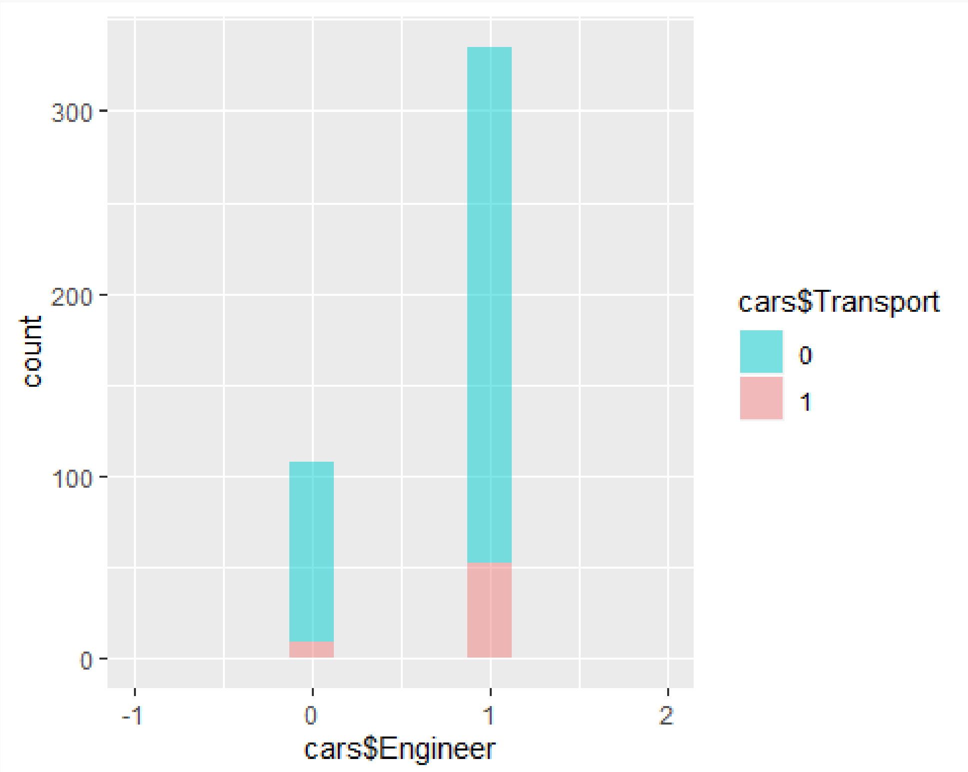

```
boxplot(cars$Distance~ cars$Transport)
```



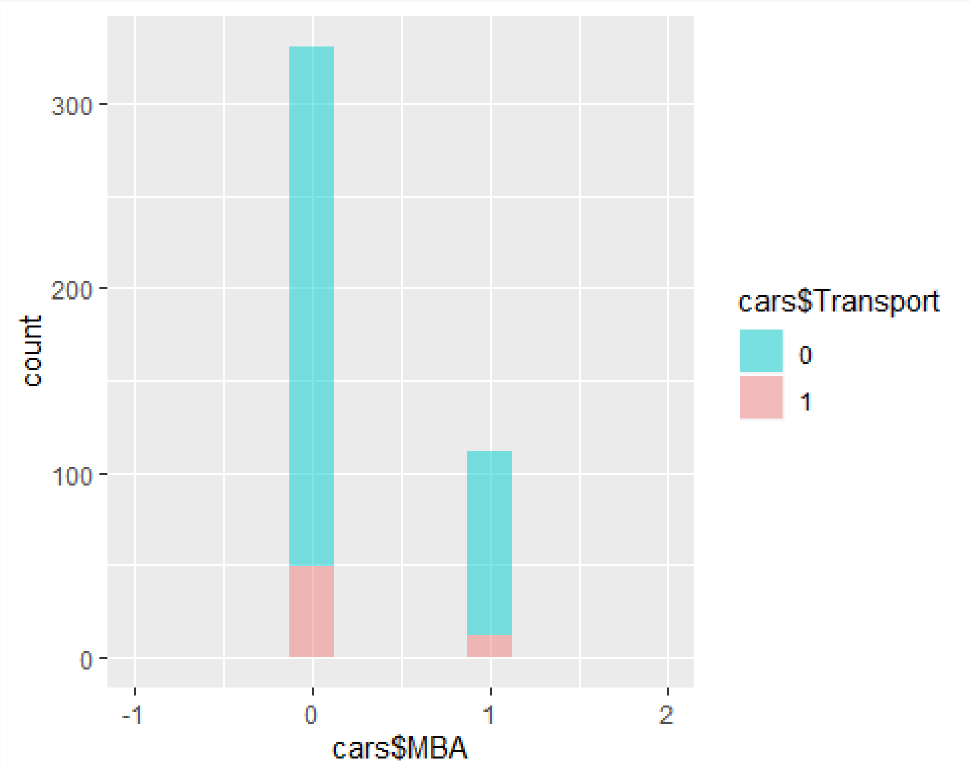

```
ggplot(cars,aes(x = cars$Gender, fill = cars$Transport)) +
  geom_bar(width = 0.25,alpha = 0.5) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen"))
```



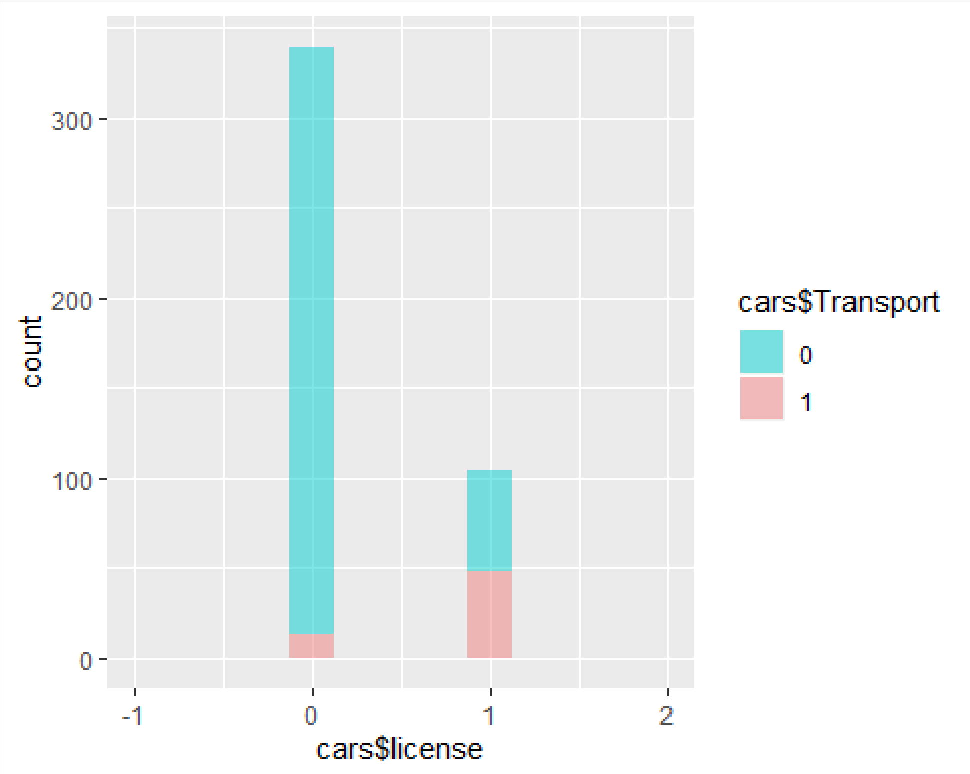

```
ggplot(cars,aes(x = cars$Engineer, fill = cars$Transport)) +
  geom_bar(width = 0.25,alpha = 0.5) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen")) + xlim(-1,2)
```




```
ggplot(cars,aes(x = cars$MBA, fill = cars$Transport)) +
  geom_bar(width = 0.25,alpha = 0.5) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen")) + xlim(-1,2)
```

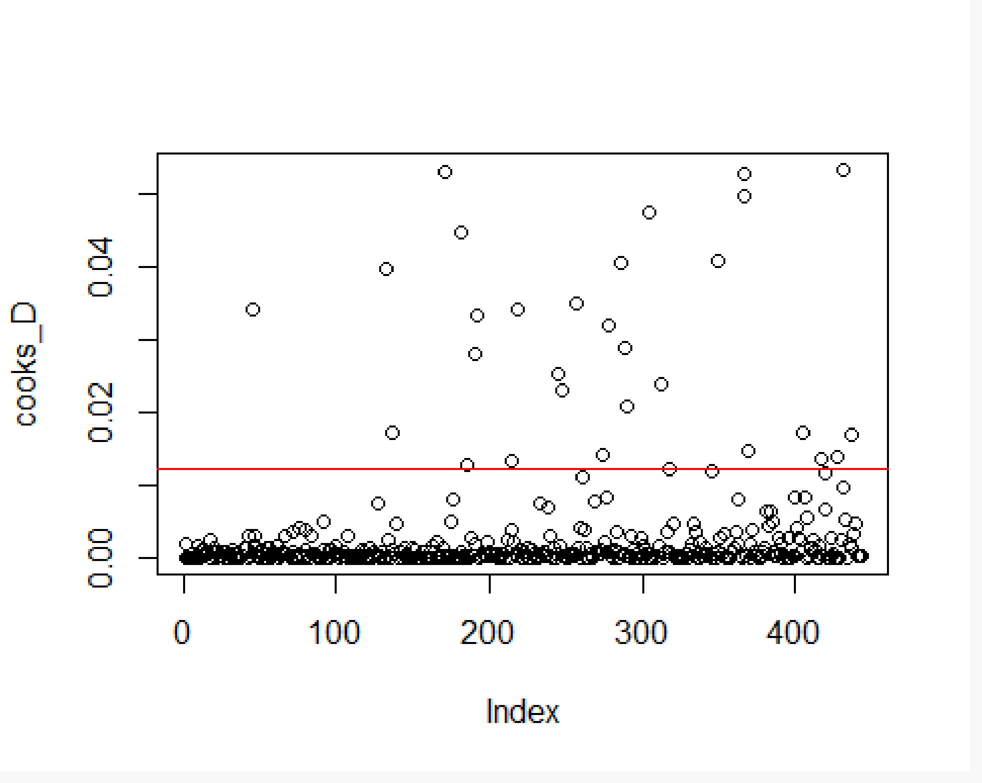



```
ggplot(cars,aes(x = cars$license, fill = cars$Transport)) +
  geom_bar(width = 0.25,alpha = 0.5) +
  scale_fill_manual(values = c("darkturquoise","lightcoral","lightgreen")) + xlim(-1,2)
```

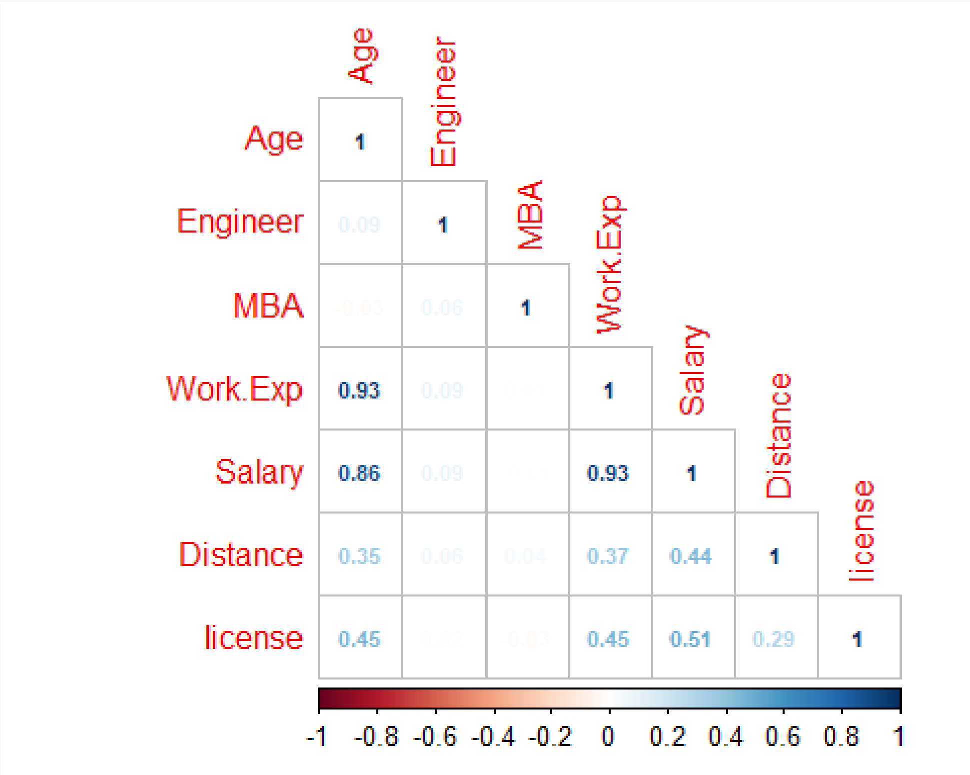



```
# Performing Cooks Distance

cd_lm <- lm(as.numeric(cars$Transport)~.,data = cars)
cooks_D <- cooks.distance(cd_lm)
```

```
plot(cooks_D)
abline(h=4*mean(cooks_D,na.rm = TRUE),col="red")
```




```
# Performing Correlation Matrix and Plotting it.

mat<- cor(cars[,-c(2,9)])
corrplot(mat,method = "number", type = "lower", number.cex = .70)
```




```
prop.table(table(cars$Transport))

##
##         0         1
## 0.8623025 0.1376975

#Splitting the data into Train and Test with a Ratio of 70 and 30 resp.
str(cars)

## 'data.frame':    443 obs. of  9 variables:
##  $ Age      : int  28 23 29 28 27 26 28 26 22 27 ...
##  $ Gender   : num  1 0 1 0 1 1 1 0 1 1 ...
##  $ Engineer : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ MBA      : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Work.Exp : int  4 4 7 5 4 4 5 3 1 4 ...
##  $ Salary   : num  14.3 8.3 13.4 13.4 13.4 12.3 14.4 10.5 7.5 13.5 ...
##  $ Distance : num  3.2 3.3 4.1 4.5 4.6 4.8 5.1 5.1 5.1 5.2 ...
##  $ license  : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "na.action")=Class 'omit'  Named int 145
##   .. ..- attr(*, "names")= chr "145"

set.seed(123)
index <- sample.split(cars$Transport, SplitRatio = .70)
trainData <- subset(cars,index==TRUE)
dim(trainData)

## [1] 310   9

testData <- subset(cars,index==FALSE)
dim(testData)
```

```
## [1] 133   9

balanced_trainData <- SMOTE(Transport~., trainData, perc.over = 350,k = 5,perc.under =13
4)
str(balanced_trainData)

## 'data.frame':    344 obs. of  9 variables:
##  $ Age      : num  30 27 30 24 30 25 29 20 27 30 ...
##  $ Gender   : num  0 1 1 1 0 1 1 0 0 0 ...
##  $ Engineer : num  1 1 1 1 0 1 1 0 1 0 ...
##  $ MBA      : num  0 0 0 1 0 0 0 1 0 0 ...
##  $ Work.Exp : num  8 4 6 0 6 3 9 1 5 6 ...
##  $ Salary   : num  14.7 13.5 15.8 7.9 15.6 10.7 23.8 8.5 12.8 15.6 ...
##  $ Distance : num  8.5 5.3 14.3 9.1 11.6 10.8 9.4 7.9 9.7 11.6 ...
##  $ license  : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...

prop.table(table(balanced_trainData$Transport))

##
##   0   1
## 0.5 0.5

dim(balanced_trainData)

## [1] 344   9


#Performing Logistic Model on Train data.

logit <- glm(balanced_trainData$Transport~., data = balanced_trainData, family = "binomi
al")
summary(logit)

##
## Call:
## glm(formula = balanced_trainData$Transport ~ ., family = "binomial",
##     data = balanced_trainData)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.49329  -0.01478   0.00004   0.02165   2.04785
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept) -74.008422  15.472147  -4.783 1.72e-06 ***
## Age           2.355327   0.521234   4.519 6.22e-06 ***
## Gender       -0.851113   0.914716  -0.930 0.352129
## Engineer      1.506589   1.123968   1.340 0.180109
## MBA          -2.871473   1.016811  -2.824 0.004743 **
## Work.Exp     -1.003964   0.314222  -3.195 0.001398 **
## Salary        0.004538   0.073320   0.062 0.950650
## Distance      0.692678   0.181160   3.824 0.000132 ***
## license       2.443810   1.019817   2.396 0.016560 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 476.885  on 343  degrees of freedom
## Residual deviance:  64.557  on 335  degrees of freedom
## AIC: 82.557
##
## Number of Fisher Scoring iterations: 9
```

```
vif(logit)
```

```
names(balanced_trainData)
```

```
## [1] "Age"       "Gender"    "Engineer"  "MBA"       "Work.Exp"  "Salary"
## [7] "Distance"  "license"   "Transport"
```

```
balanced_trainData1 <- balanced_trainData[,-7]
str(balanced_trainData1)
```

```
## 'data.frame':    344 obs. of  8 variables:
##  $ Age      : num  30 27 30 24 30 25 29 20 27 30 ...
##  $ Gender   : num  0 1 1 1 0 1 1 0 0 0 ...
##  $ Engineer : num  1 1 1 1 0 1 1 0 1 0 ...
##  $ MBA      : num  0 0 0 1 0 0 0 1 0 0 ...
##  $ Work.Exp : num  8 4 6 0 6 3 9 1 5 6 ...
##  $ Salary   : num  14.7 13.5 15.8 7.9 15.6 10.7 23.8 8.5 12.8 15.6 ...
##  $ license  : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
testData1 <- testData[,-7]
str(testData1)
```

```
## 'data.frame':    133 obs. of  8 variables:
##  $ Age      : int  23 28 27 26 25 25 23 26 24 30 ...
##  $ Gender   : num  0 0 1 0 0 1 1 1 1 1 ...
##  $ Engineer : int  1 1 1 1 1 1 0 1 1 0 ...
##  $ MBA      : int  0 1 0 0 0 1 0 0 0 0 ...
##  $ Work.Exp : int  4 5 4 3 4 4 2 5 6 8 ...
##  $ Salary   : num  8.3 13.4 13.4 10.5 11.5 11.5 8.6 11.4 10.6 14.6 ...
##  $ license  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
logit_refined <- glm(Transport~.,
                     data = balanced_trainData1, family = "binomial")
summary(logit)
```

```
##
## Call:
## glm(formula = balanced_trainData$Transport ~ ., family = "binomial",
##     data = balanced_trainData)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.49329  -0.01478   0.00004   0.02165   2.04785
##
## Coefficients:
```

```
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -74.008422  15.472147  -4.783 1.72e-06 ***
## Age           2.355327   0.521234   4.519 6.22e-06 ***
## Gender       -0.851113   0.914716  -0.930 0.352129
## Engineer      1.506589   1.123968   1.340 0.180109
## MBA          -2.871473   1.016811  -2.824 0.004743 **
## Work.Exp     -1.003964   0.314222  -3.195 0.001398 **
## Salary        0.004538   0.073320   0.062 0.950650
## Distance      0.692678   0.181160   3.824 0.000132 ***
## license       2.443810   1.019817   2.396 0.016560 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 476.885  on 343  degrees of freedom
## Residual deviance:  64.557  on 335  degrees of freedom
## AIC: 82.557
##
## Number of Fisher Scoring iterations: 9
```

```
vif(logit_refined)


# Calculating Likelihood Test
logit_likelihood <- lrtest(logit_refined)
logit_likelihood
```

```
## Likelihood ratio test
##
## Model 1: Transport ~ Age + Gender + Engineer + MBA + Work.Exp + Salary +
##     license
## Model 2: Transport ~ 1
##   #Df   LogLik Df  Chisq Pr(>Chisq)
## 1   8  -45.777
## 2   1 -238.443 -7 385.33  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Calculating Psudo R Sq

logit_rsq <- pR2(logit_refined)
logit_rsq
```

```
##          llh      llhNull           G2     McFadden         r2ML
##  -45.7768485 -238.4426301  385.3315632    0.8080173    0.6737691
##         r2CU
##    0.8983588
```

```
# Calculating Odds Ratio

logit_odds <- exp(logit_refined$coefficients)
print(logit_odds, digits = 10)
```

```
##     (Intercept)             Age          Gender        Engineer
## 2.294700471e-25 7.045711916e+00 4.027238819e-01 4.036941157e+00
```

```
##              MBA        Work.Exp          Salary         license
## 4.529030631e-02 5.484634778e-01 1.032239364e+00 5.227973298e+00

# Predicting values on Train Data.

pred_logit_train <- predict(logit_refined, balanced_trainData)

pred_logit_train_class <- ifelse(pred_logit_train <.5,0,1)
head(pred_logit_train_class)

## 101  12 361 130 247 212
##   0   0   0   0   0   0

prop.table(table(pred_logit_train_class))

## pred_logit_train_class
##         0         1
## 0.5203488 0.4796512

# Calculating Baseline
prop.table(table(balanced_trainData$Transport))

##
##   0   1
## 0.5 0.5

# Creating Performance Matrix on Train

pred_logit_train_class<- as.factor(pred_logit_train_class)
caret::confusionMatrix(pred_logit_train_class,balanced_trainData$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 163  16
##          1   9 156
##
##                Accuracy : 0.9273
##                  95% CI : (0.8946, 0.9524)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8547
##
##  Mcnemar's Test P-Value : 0.2301
##
##             Sensitivity : 0.9477
##             Specificity : 0.9070
##          Pos Pred Value : 0.9106
##          Neg Pred Value : 0.9455
##              Prevalence : 0.5000
##          Detection Rate : 0.4738
##    Detection Prevalence : 0.5203
##       Balanced Accuracy : 0.9273
##
```

```
##        'Positive' Class : 0
##

# Performing Prediction on Test Data

pred_logit_test <- predict(logit_refined, testData1, type = "response")
pred_logit_test_class <- ifelse(pred_logit_test <.5,0,1)
head(pred_logit_test_class)

##  2  4  5  8 11 16
##  0  0  0  0  0  0

# Creating performance Matrix on Test

pred_logit_test_class <- as.factor(pred_logit_test_class)
caret::confusionMatrix(pred_logit_test_class,testData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 111   3
##          1   4  15
##
##                Accuracy : 0.9474
##                  95% CI : (0.8946, 0.9786)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.0017
##
##                   Kappa : 0.7803
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.9652
##             Specificity : 0.8333
##          Pos Pred Value : 0.9737
##          Neg Pred Value : 0.7895
##              Prevalence : 0.8647
##          Detection Rate : 0.8346
##    Detection Prevalence : 0.8571
##       Balanced Accuracy : 0.8993
##
##        'Positive' Class : 0
##

# Performing K- Nearest Neighbor
ctrl <- trainControl(method = "cv", number = 3)
knnModel <- train(Transport~.,data = balanced_trainData1,method = "knn",
                  trControl = ctrl,
                  tuneLength = 10)
knnModel$bestTune

##   k
## 3 9
```

```r
summary(knnModel)

##             Length Class      Mode
## learn       2      -none-     list
## k           1      -none-     numeric
## theDots     0      -none-     list
## xNames      7      -none-     character
## problemType 1      -none-     character
## tuneValue   1      data.frame list
## obsLevels   2      -none-     character
## param       0      -none-     list

# Predicting Value on Train data

pred_knn_Train <- predict(knnModel,balanced_trainData1)

# Creating Perfromance Matrix

caret::confusionMatrix(pred_knn_Train,balanced_trainData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 158   4
##          1  14 168
##
##                Accuracy : 0.9477
##                  95% CI : (0.9186, 0.9687)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8953
##
##  Mcnemar's Test P-Value : 0.03389
##
##             Sensitivity : 0.9186
##             Specificity : 0.9767
##          Pos Pred Value : 0.9753
##          Neg Pred Value : 0.9231
##              Prevalence : 0.5000
##          Detection Rate : 0.4593
##    Detection Prevalence : 0.4709
##       Balanced Accuracy : 0.9477
##
##        'Positive' Class : 0
##

# Predicting Values on Test data.

pred_knn_test <- predict(knnModel, testData1)

caret::confusionMatrix(pred_knn_test, testData1$Transport)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 110   2
##          1   5  16
##
##                Accuracy : 0.9474
##                  95% CI : (0.8946, 0.9786)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.0017
##
##                   Kappa : 0.7899
##
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.9565
##             Specificity : 0.8889
##          Pos Pred Value : 0.9821
##          Neg Pred Value : 0.7619
##              Prevalence : 0.8647
##          Detection Rate : 0.8271
##    Detection Prevalence : 0.8421
##       Balanced Accuracy : 0.9227
##
##        'Positive' Class : 0
##
```

```
# Performing Naive Bayes Model

NBModel <- naiveBayes(Transport~., data = balanced_trainData1)
summary(NBModel)
```

```
##           Length Class  Mode
## apriori   2      table  numeric
## tables    7      -none- list
## levels    2      -none- character
## isnumeric 7      -none- logical
## call      4      -none- call
```

```
# Prediction on Train data.

pred_nb_train <- predict(NBModel, balanced_trainData1)


# Creating Performance Matrix on Traindata
caret::confusionMatrix(pred_nb_train, balanced_trainData1$Transport)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 163  20
##          1   9 152
##
```

```
##                Accuracy : 0.9157
##                  95% CI : (0.8812, 0.9428)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8314
##
##  Mcnemar's Test P-Value : 0.06332
##
##             Sensitivity : 0.9477
##             Specificity : 0.8837
##          Pos Pred Value : 0.8907
##          Neg Pred Value : 0.9441
##              Prevalence : 0.5000
##          Detection Rate : 0.4738
##    Detection Prevalence : 0.5320
##       Balanced Accuracy : 0.9157
##
##        'Positive' Class : 0
##
```

```
# Prediction on Test data.

pred_nb_test <- predict(NBModel, testData1)

# Creating Performance Matrix on Test Data
caret::confusionMatrix(pred_nb_test, testData1$Transport)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 113   4
##          1   2  14
##
##                Accuracy : 0.9549
##                  95% CI : (0.9044, 0.9833)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.0005588
##
##                   Kappa : 0.7978
##
##  Mcnemar's Test P-Value : 0.6830914
##
##             Sensitivity : 0.9826
##             Specificity : 0.7778
##          Pos Pred Value : 0.9658
##          Neg Pred Value : 0.8750
##              Prevalence : 0.8647
##          Detection Rate : 0.8496
##    Detection Prevalence : 0.8797
##       Balanced Accuracy : 0.8802
##
```

```
##        'Positive' Class : 0
##

# Performing Bagging Model

bagModel <- bagging(as.numeric(Transport)~.,data = balanced_trainData1,
                    control = rpart.control(maxdepth = 5, minsplit = 3))

summary(bagModel)

##        Length Class      Mode
## y      344    -none-     numeric
## X        7    data.frame list
## mtrees  25    -none-     list
## OOB      1    -none-     logical
## comb     1    -none-     logical
## call     4    -none-     call

bagModel$X

##             Age     Gender  Engineer        MBA  Work.Exp   Salary
## 101   30.00000 0.00000000 1.0000000 0.00000000  8.000000 14.70000
## 12    27.00000 1.00000000 1.0000000 0.00000000  4.000000 13.50000
## 361   30.00000 1.00000000 1.0000000 0.00000000  6.000000 15.80000
## 130   24.00000 1.00000000 1.0000000 1.00000000  0.000000  7.90000
## 247   30.00000 0.00000000 0.0000000 0.00000000  6.000000 15.60000
## 212   25.00000 1.00000000 1.0000000 0.00000000  3.000000 10.70000
## 150   29.00000 1.00000000 1.0000000 0.00000000  9.000000 23.80000
## 76    20.00000 0.00000000 0.0000000 1.00000000  1.000000  8.50000
## 166   27.00000 0.00000000 1.0000000 0.00000000  5.000000 12.80000
## 247.1 30.00000 0.00000000 0.0000000 0.00000000  6.000000 15.60000
## 205   27.00000 1.00000000 0.0000000 1.00000000  8.000000 20.70000
## 264   30.00000 0.00000000 1.0000000 0.00000000  6.000000 15.60000
## 297   30.00000 1.00000000 1.0000000 1.00000000  8.000000 14.70000
## 191   28.00000 1.00000000 1.0000000 0.00000000  3.000000 10.80000
## 152   22.00000 0.00000000 1.0000000 1.00000000  2.000000  8.50000
## 3     29.00000 1.00000000 1.0000000 0.00000000  7.000000 13.40000
## 1     28.00000 1.00000000 0.0000000 0.00000000  4.000000 14.30000
## 423   23.00000 1.00000000 0.0000000 0.00000000  3.000000  9.90000
## 45    33.00000 1.00000000 0.0000000 0.00000000 13.000000 36.60000
## 163   25.00000 1.00000000 1.0000000 0.00000000  1.000000  8.60000
## 290   24.00000 1.00000000 1.0000000 0.00000000  1.000000  7.70000
## 303   21.00000 1.00000000 0.0000000 0.00000000  3.000000  9.80000
## 369   26.00000 1.00000000 1.0000000 0.00000000  5.000000 12.80000
## 240   23.00000 1.00000000 1.0000000 0.00000000  4.000000 10.60000
## 307   27.00000 1.00000000 1.0000000 0.00000000  6.000000 12.80000
## 352   27.00000 1.00000000 0.0000000 0.00000000  9.000000 23.90000
## 241   28.00000 0.00000000 0.0000000 0.00000000  9.000000 23.80000
## 324   25.00000 0.00000000 0.0000000 0.00000000  2.000000  8.90000
## 12.1  27.00000 1.00000000 1.0000000 0.00000000  4.000000 13.50000
## 163.1 25.00000 1.00000000 1.0000000 0.00000000  1.000000  8.60000
## 370   34.00000 0.00000000 0.0000000 0.00000000 14.000000 28.80000
## 254   27.00000 0.00000000 1.0000000 0.00000000  3.000000 10.70000
## 389   26.00000 0.00000000 1.0000000 0.00000000  3.000000 10.80000
## 70    26.00000 1.00000000 1.0000000 1.00000000  4.000000 12.40000
```

```
## 123   26.00000 0.00000000 1.0000000 0.00000000  3.000000 10.80000
## 329   27.00000 0.00000000 1.0000000 1.00000000  4.000000 13.70000
## 422   23.00000 0.00000000 1.0000000 1.00000000  2.000000  9.00000
## 125   24.00000 1.00000000 1.0000000 0.00000000  4.000000 10.90000
## 1.1   28.00000 1.00000000 0.0000000 0.00000000  4.000000 14.30000
## 422.1 23.00000 0.00000000 1.0000000 1.00000000  2.000000  9.00000
## 60    24.00000 1.00000000 1.0000000 0.00000000  4.000000  8.50000
## 23    27.00000 0.00000000 1.0000000 0.00000000  9.000000 15.50000
## 235   24.00000 1.00000000 1.0000000 1.00000000  6.000000 11.60000
## 124   26.00000 1.00000000 1.0000000 0.00000000  4.000000 12.70000
## 206   28.00000 1.00000000 0.0000000 0.00000000  6.000000 13.90000
## 327   24.00000 1.00000000 1.0000000 0.00000000  4.000000 13.80000
## 30    30.00000 0.00000000 1.0000000 0.00000000  8.000000 14.60000
## 281   26.00000 0.00000000 1.0000000 0.00000000  2.000000  9.80000
## 399   20.00000 0.00000000 1.0000000 0.00000000  2.000000  9.00000
## 109   20.00000 1.00000000 1.0000000 0.00000000  2.000000  8.80000
## 98    24.00000 1.00000000 0.0000000 0.00000000  2.000000  8.70000
## 63    23.00000 1.00000000 1.0000000 0.00000000  1.000000  7.50000
## 35    30.00000 0.00000000 1.0000000 0.00000000  8.000000 14.60000
## 216   33.00000 1.00000000 1.0000000 1.00000000 14.000000 34.90000
## 192   34.00000 1.00000000 1.0000000 1.00000000 14.000000 36.90000
## 193   36.00000 1.00000000 1.0000000 1.00000000 18.000000 28.70000
## 377   25.00000 0.00000000 1.0000000 0.00000000  2.000000  8.80000
## 15    32.00000 1.00000000 1.0000000 0.00000000  9.000000 15.50000
## 81    22.00000 0.00000000 1.0000000 0.00000000  2.000000 11.70000
## 203   30.00000 1.00000000 1.0000000 1.00000000  8.000000 14.60000
## 94    24.00000 1.00000000 1.0000000 1.00000000  6.000000 10.60000
## 222   26.00000 0.00000000 1.0000000 0.00000000  4.000000 12.60000
## 189   29.00000 1.00000000 1.0000000 1.00000000  6.000000 14.60000
## 221   21.00000 0.00000000 0.0000000 0.00000000  3.000000  9.80000
## 57    26.00000 1.00000000 1.0000000 0.00000000  4.000000 12.50000
## 57.1  26.00000 1.00000000 1.0000000 0.00000000  4.000000 12.50000
## 162   26.00000 1.00000000 1.0000000 0.00000000  3.000000 10.50000
## 272   25.00000 0.00000000 1.0000000 0.00000000  5.000000 17.80000
## 256   29.00000 0.00000000 0.0000000 0.00000000  7.000000 13.60000
## 386   28.00000 0.00000000 0.0000000 1.00000000  9.000000 23.80000
## 332   27.00000 1.00000000 1.0000000 1.00000000  8.000000 21.80000
## 269   28.00000 1.00000000 1.0000000 1.00000000  7.000000 13.90000
## 330   27.00000 1.00000000 1.0000000 0.00000000  6.000000 12.90000
## 52    30.00000 0.00000000 1.0000000 0.00000000  8.000000 14.40000
## 98.1  24.00000 1.00000000 0.0000000 0.00000000  2.000000  8.70000
## 129   26.00000 0.00000000 1.0000000 1.00000000  3.000000 10.90000
## 160   26.00000 1.00000000 1.0000000 0.00000000  3.000000 10.60000
## 250   23.00000 1.00000000 1.0000000 0.00000000  0.000000  6.90000
## 330.1 27.00000 1.00000000 1.0000000 0.00000000  6.000000 12.90000
## 101.1 30.00000 0.00000000 1.0000000 0.00000000  8.000000 14.70000
## 333   26.00000 0.00000000 1.0000000 0.00000000  8.000000 20.90000
## 9     22.00000 1.00000000 1.0000000 0.00000000  1.000000  7.50000
## 256.1 29.00000 0.00000000 0.0000000 0.00000000  7.000000 13.60000
## 331   24.00000 1.00000000 0.0000000 1.00000000  2.000000  8.90000
## 181   25.00000 0.00000000 1.0000000 0.00000000  6.000000 11.60000
## 81.1  22.00000 0.00000000 1.0000000 0.00000000  2.000000 11.70000
## 10    27.00000 1.00000000 1.0000000 0.00000000  4.000000 13.50000
## 110   24.00000 1.00000000 1.0000000 0.00000000  6.000000 12.70000
```

```
## 386.1 28.00000 0.00000000 0.0000000 1.00000000  9.000000 23.80000
## 393   24.00000 0.00000000 1.0000000 0.00000000  1.000000  8.80000
## 211   25.00000 1.00000000 1.0000000 1.00000000  7.000000 13.60000
## 86    29.00000 1.00000000 1.0000000 0.00000000  6.000000 14.70000
## 192.1 34.00000 1.00000000 1.0000000 1.00000000 14.000000 36.90000
## 328   26.00000 1.00000000 0.0000000 0.00000000  4.000000 12.70000
## 160.1 26.00000 1.00000000 1.0000000 0.00000000  3.000000 10.60000
## 44    25.00000 1.00000000 1.0000000 0.00000000  3.000000 10.50000
## 109.1 20.00000 1.00000000 1.0000000 0.00000000  2.000000  8.80000
## 144   33.00000 1.00000000 1.0000000 1.00000000 11.000000 15.60000
## 101.2 30.00000 0.00000000 1.0000000 0.00000000  8.000000 14.70000
## 184   29.00000 1.00000000 1.0000000 0.00000000  6.000000 14.80000
## 146   25.00000 1.00000000 1.0000000 0.00000000  1.000000  8.60000
## 49    28.00000 0.00000000 0.0000000 1.00000000  5.000000 14.60000
## 19    23.00000 1.00000000 1.0000000 0.00000000  2.000000  8.50000
## 102   24.00000 1.00000000 0.0000000 1.00000000  2.000000  8.50000
## 408   32.00000 0.00000000 1.0000000 1.00000000  9.000000 15.90000
## 192.2 34.00000 1.00000000 1.0000000 1.00000000 14.000000 36.90000
## 187   35.00000 0.00000000 1.0000000 0.00000000 16.000000 28.70000
## 304   22.00000 1.00000000 1.0000000 0.00000000  0.000000  6.80000
## 267   30.00000 1.00000000 1.0000000 0.00000000  8.000000 14.80000
## 23.1  27.00000 0.00000000 1.0000000 0.00000000  9.000000 15.50000
## 281.1 26.00000 0.00000000 1.0000000 0.00000000  2.000000  9.80000
## 144.1 33.00000 1.00000000 1.0000000 1.00000000 11.000000 15.60000
## 376   22.00000 1.00000000 0.0000000 0.00000000  0.000000  6.80000
## 313   34.00000 1.00000000 1.0000000 1.00000000 15.000000 37.00000
## 56    28.00000 0.00000000 1.0000000 0.00000000  9.000000 21.70000
## 116   29.00000 1.00000000 1.0000000 0.00000000  9.000000 22.80000
## 51    23.00000 1.00000000 1.0000000 1.00000000  3.000000 11.70000
## 230   29.00000 1.00000000 1.0000000 0.00000000  5.000000 14.90000
## 217   32.00000 1.00000000 1.0000000 0.00000000 12.000000 15.70000
## 273   31.00000 1.00000000 1.0000000 0.00000000 10.000000 14.90000
## 122   28.00000 0.00000000 0.0000000 0.00000000 10.000000 19.70000
## 237   30.00000 1.00000000 1.0000000 0.00000000 10.000000 13.80000
## 422.2 23.00000 0.00000000 1.0000000 1.00000000  2.000000  9.00000
## 303.1 21.00000 1.00000000 0.0000000 0.00000000  3.000000  9.80000
## 36    27.00000 1.00000000 1.0000000 0.00000000  6.000000 12.60000
## 174   24.00000 1.00000000 1.0000000 0.00000000  0.000000  7.60000
## 25    24.00000 1.00000000 0.0000000 0.00000000  2.000000  8.50000
## 133   27.00000 1.00000000 0.0000000 0.00000000  7.000000 12.50000
## 298   24.00000 0.00000000 1.0000000 1.00000000  2.000000  8.70000
## 3.1   29.00000 1.00000000 1.0000000 0.00000000  7.000000 13.40000
## 312   30.00000 0.00000000 1.0000000 0.00000000  6.000000 15.80000
## 181.1 25.00000 0.00000000 1.0000000 0.00000000  6.000000 11.60000
## 292   27.00000 1.00000000 1.0000000 0.00000000  4.000000 13.80000
## 266   28.00000 1.00000000 1.0000000 1.00000000  6.000000 13.70000
## 221.1 21.00000 0.00000000 0.0000000 0.00000000  3.000000  9.80000
## 283   26.00000 1.00000000 1.0000000 0.00000000  3.000000 10.70000
## 259   21.00000 1.00000000 1.0000000 1.00000000  3.000000  9.90000
## 113   25.00000 0.00000000 1.0000000 0.00000000  3.000000 10.60000
## 42    25.00000 0.00000000 1.0000000 0.00000000  4.000000 11.50000
## 405   31.00000 1.00000000 1.0000000 0.00000000  8.000000 15.90000
## 299   27.00000 1.00000000 1.0000000 0.00000000  8.000000 20.70000
## 241.1 28.00000 0.00000000 0.0000000 0.00000000  9.000000 23.80000
```

```
## 49.1  28.00000 0.00000000 0.0000000 1.00000000  5.000000 14.60000
## 214   26.00000 0.00000000 1.0000000 1.00000000  4.000000 12.80000
## 102.1 24.00000 1.00000000 0.0000000 1.00000000  2.000000  8.50000
## 379   24.00000 1.00000000 0.0000000 0.00000000  0.000000  6.90000
## 6     26.00000 1.00000000 1.0000000 0.00000000  4.000000 12.30000
## 94.1  24.00000 1.00000000 1.0000000 1.00000000  6.000000 10.60000
## 52.1  30.00000 0.00000000 1.0000000 0.00000000  8.000000 14.40000
## 128   28.00000 0.00000000 1.0000000 0.00000000  5.000000 14.60000
## 293   26.00000 1.00000000 1.0000000 1.00000000  5.000000 12.70000
## 422.3 23.00000 0.00000000 1.0000000 1.00000000  2.000000  9.00000
## 286   27.00000 1.00000000 1.0000000 0.00000000  8.000000 20.70000
## 198   24.00000 1.00000000 1.0000000 1.00000000  1.000000  7.90000
## 168   27.00000 0.00000000 1.0000000 1.00000000  4.000000 13.80000
## 399.1 20.00000 0.00000000 1.0000000 0.00000000  2.000000  9.00000
## 49.2  28.00000 0.00000000 0.0000000 1.00000000  5.000000 14.60000
## 36.1  27.00000 1.00000000 1.0000000 0.00000000  6.000000 12.60000
## 377.1 25.00000 0.00000000 1.0000000 0.00000000  2.000000  8.80000
## 165   31.00000 1.00000000 0.0000000 1.00000000  7.000000 15.90000
## 18    26.00000 0.00000000 1.0000000 0.00000000  4.000000 12.30000
## 254.1 27.00000 0.00000000 1.0000000 0.00000000  3.000000 10.70000
## 184.1 29.00000 1.00000000 1.0000000 0.00000000  6.000000 14.80000
## 242   27.00000 1.00000000 1.0000000 0.00000000  6.000000 12.70000
## 105   23.00000 1.00000000 0.0000000 0.00000000  2.000000  8.80000
## 62    26.00000 1.00000000 0.0000000 0.00000000  4.000000 12.60000
## 47    21.00000 1.00000000 0.0000000 0.00000000  3.000000  9.50000
## 199   28.00000 1.00000000 1.0000000 0.00000000  5.000000 14.70000
## 214.1 26.00000 0.00000000 1.0000000 1.00000000  4.000000 12.80000
## 56.1  28.00000 0.00000000 1.0000000 0.00000000  9.000000 21.70000
## 70.1  26.00000 1.00000000 1.0000000 1.00000000  4.000000 12.40000
## 309   26.00000 1.00000000 1.0000000 0.00000000  4.000000 12.70000
## 127   39.00000 1.00000000 1.0000000 1.00000000 19.000000 38.90000
## 172   30.00000 1.00000000 1.0000000 0.00000000  4.000000 16.80000
## 177   36.00000 1.00000000 0.0000000 0.00000000 17.000000 39.00000
## 182   32.00000 0.00000000 1.0000000 1.00000000  9.000000 16.90000
## 219   33.00000 1.00000000 1.0000000 0.00000000 11.000000 16.70000
## 229   39.00000 1.00000000 1.0000000 0.00000000 19.000000 47.00000
## 249   32.00000 1.00000000 1.0000000 0.00000000  9.000000 16.90000
## 265   40.00000 1.00000000 1.0000000 0.00000000 21.000000 54.00000
## 277   33.00000 0.00000000 1.0000000 0.00000000 13.000000 36.00000
## 279   32.00000 1.00000000 0.0000000 0.00000000  9.000000 16.90000
## 287   33.00000 1.00000000 0.0000000 0.00000000 11.000000 17.00000
## 289   33.00000 1.00000000 1.0000000 0.00000000 10.000000 16.90000
## 306   34.00000 1.00000000 1.0000000 1.00000000 11.000000 17.00000
## 335   35.00000 0.00000000 1.0000000 0.00000000 15.000000 37.00000
## 337   38.00000 1.00000000 1.0000000 0.00000000 19.000000 54.00000
## 338   36.00000 0.00000000 1.0000000 0.00000000 18.000000 44.00000
## 351   32.00000 1.00000000 1.0000000 0.00000000 11.000000 15.80000
## 353   38.00000 1.00000000 1.0000000 0.00000000 19.000000 48.00000
## 355   40.00000 1.00000000 1.0000000 0.00000000 22.000000 51.00000
## 364   31.00000 1.00000000 1.0000000 0.00000000 12.000000 34.00000
## 367   32.00000 0.00000000 0.0000000 0.00000000 10.000000 15.90000
## 368   32.00000 0.00000000 1.0000000 1.00000000 10.000000 15.80000
## 373   34.00000 1.00000000 1.0000000 0.00000000 14.000000 45.00000
## 396   37.00000 1.00000000 1.0000000 1.00000000 18.000000 41.00000
```

```
## 401   39.00000 1.00000000 1.0000000 0.00000000 21.000000 40.90000
## 406   32.00000 0.00000000 1.0000000 0.00000000 14.000000 30.90000
## 411   40.00000 1.00000000 1.0000000 1.00000000 20.000000 41.90000
## 418   33.00000 1.00000000 1.0000000 0.00000000 14.000000 33.00000
## 420   34.00000 1.00000000 1.0000000 1.00000000 16.000000 36.00000
## 424   36.00000 0.00000000 1.0000000 0.00000000 17.000000 38.00000
## 425   39.00000 1.00000000 1.0000000 0.00000000 21.000000 46.00000
## 426   38.00000 1.00000000 1.0000000 0.00000000 18.000000 45.00000
## 427   40.00000 1.00000000 1.0000000 0.00000000 20.000000 48.00000
## 430   38.00000 1.00000000 1.0000000 0.00000000 19.000000 51.00000
## 431   42.00000 1.00000000 1.0000000 0.00000000 22.000000 55.00000
## 435   40.00000 1.00000000 1.0000000 0.00000000 22.000000 45.00000
## 436   37.00000 1.00000000 0.0000000 0.00000000 19.000000 42.00000
## 437   43.00000 1.00000000 1.0000000 1.00000000 24.000000 52.00000
## 439   34.00000 1.00000000 1.0000000 0.00000000 14.000000 38.00000
## 440   40.00000 1.00000000 1.0000000 0.00000000 20.000000 57.00000
## 441   38.00000 1.00000000 1.0000000 0.00000000 19.000000 44.00000
## 442   37.00000 1.00000000 1.0000000 0.00000000 19.000000 45.00000
## 443   37.00000 1.00000000 0.0000000 0.00000000 19.000000 47.00000
## 131   37.26706 1.00000000 1.0000000 1.00000000 18.133532 40.71958
## 2     42.04160 1.00000000 1.0000000 1.00000000 22.801998 48.86123
## 310   38.28389 1.00000000 1.0000000 1.00000000 18.641943 39.65192
## 4     31.57147 1.00000000 1.0000000 0.00000000  7.666758 16.74762
## 5     30.72159 1.00000000 1.0000000 0.00000000  6.405307 20.69660
## 610   30.70986 1.00000000 1.0000000 0.00000000  5.419718 16.82366
## 7     38.43548 1.00000000 0.8118274 0.00000000 18.623655 45.49462
## 8     36.15491 1.00000000 0.0000000 0.00000000 17.309823 40.23929
## 910   33.07582 1.00000000 0.0000000 0.00000000 11.151645 17.55603
## 1010  32.00000 0.00000000 1.0000000 0.53598337 11.320083 23.39623
## 11    33.16987 0.58493656 1.0000000 1.00000000 10.169873 16.95849
## 1210  32.00000 0.00000000 1.0000000 1.00000000  9.230097 16.64689
## 13    33.28285 1.00000000 1.0000000 0.00000000 11.848557 24.70472
## 14    33.09392 1.00000000 1.0000000 0.00000000 11.281750 18.70042
## 151   33.42743 1.00000000 1.0000000 0.00000000 12.282285 25.80422
## 16    39.66239 1.00000000 1.0000000 0.00000000 20.987157 49.64954
## 17    38.37285 1.00000000 1.0000000 0.00000000 19.000000 51.39002
## 183   39.21724 1.00000000 1.0000000 0.00000000 19.434487 48.52070
## 194   32.43033 1.00000000 1.0000000 0.00000000 10.075832 22.94618
## 20    32.61553 1.00000000 1.0000000 0.00000000 10.538819 23.39382
## 21    33.86536 1.00000000 1.0000000 0.00000000 13.663405 43.10834
## 22    40.00000 1.00000000 1.0000000 0.00000000 21.127706 53.61688
## 231   39.14210 1.00000000 1.0000000 0.00000000 20.142103 51.42631
## 24    40.00000 1.00000000 1.0000000 0.00000000 20.028562 48.17137
## 251   33.91957 0.00000000 1.0000000 0.00000000 13.919570 36.45979
## 26    32.82612 0.08694017 1.0000000 0.00000000 12.913060 35.82612
## 27    33.25310 0.25309899 1.0000000 0.00000000 13.253099 38.27789
## 28    32.68632 1.00000000 0.0000000 0.00000000 10.372650 16.96863
## 29    32.35361 1.00000000 0.3536061 0.00000000  9.707212 16.82928
## 301   33.14840 1.00000000 0.0000000 0.00000000 11.296801 23.24491
## 31    32.63455 1.00000000 0.0000000 0.00000000 10.269091 16.96345
## 32    32.46395 1.00000000 0.0000000 0.00000000  9.927893 16.94639
## 33    36.78014 1.00000000 0.0000000 0.00000000 18.560281 40.62588
## 34    33.00000 1.00000000 1.0000000 0.00000000 11.858855 24.38189
## 354   30.41968 1.00000000 1.0000000 0.00000000  4.839359 16.81399
```

```
## 362   33.00000 1.00000000 1.0000000 0.00000000 12.943597 28.74798
## 37    33.54524 1.00000000 1.0000000 0.54523837 10.545238 16.95452
## 38    33.93735 1.00000000 1.0000000 0.93735000 11.187950 18.00240
## 39    34.00000 1.00000000 1.0000000 1.00000000 12.505713 22.72171
## 40    35.31211 0.00000000 1.0000000 0.00000000 15.936338 39.18479
## 41    33.96239 0.00000000 1.0000000 0.00000000 13.962390 36.48120
## 421   34.09677 0.90323356 1.0000000 0.00000000 14.096766 44.22587
## 43    38.00000 1.00000000 1.0000000 0.00000000 19.000000 48.06553
## 444   39.87828 1.00000000 1.0000000 0.00000000 20.878277 54.00000
## 451   38.44704 1.00000000 1.0000000 0.00000000 19.000000 50.87074
## 46    35.92100 0.03949920 1.0000000 0.00000000 17.842003 44.03950
## 471   36.69744 0.34871949 1.0000000 0.00000000 18.348719 45.39488
## 48    34.92892 0.53554139 1.0000000 0.00000000 15.857834 44.53554
## 491   32.80095 1.00000000 1.0000000 0.00000000 11.000000 16.52085
## 50    32.00000 1.00000000 1.0000000 0.00000000  9.335784 16.71532
## 511   32.75311 1.00000000 1.0000000 0.00000000 11.000000 16.47780
## 521   38.00000 1.00000000 1.0000000 0.00000000 18.918533 47.75560
## 53    38.00000 1.00000000 1.0000000 0.00000000 18.197762 45.59329
## 54    38.32760 1.00000000 1.0000000 0.00000000 19.000000 47.67240
## 55    38.86123 1.00000000 1.0000000 0.00000000 20.291852 49.29185
## 561   40.00000 1.00000000 1.0000000 0.00000000 20.940572 49.41086
## 571   38.66853 1.00000000 1.0000000 0.00000000 20.002795 52.99721
## 58    31.50984 1.00000000 1.0000000 0.00000000 10.470471 25.28169
## 59    31.14313 1.00000000 1.0000000 0.00000000 12.095420 34.52481
## 601   33.30786 1.00000000 1.0000000 0.00000000 13.538571 37.07714
## 61    32.00000 0.65026150 0.0000000 0.00000000  9.349738 16.55026
## 621   32.39525 0.39525376 0.0000000 0.00000000 10.395254 16.33478
## 631   32.00000 0.00000000 0.5470234 0.54702342  9.452977 16.44702
## 64    34.21326 0.00000000 1.0000000 0.44668601 13.873198 28.08357
## 65    34.34985 0.00000000 1.0000000 0.41253863 14.112230 28.84164
## 66    34.84876 0.00000000 1.0000000 0.05041468 14.747927 35.93121
## 67    35.65322 1.00000000 1.0000000 0.00000000 16.066527 48.71975
## 68    32.29980 1.00000000 1.0000000 0.00000000 12.866532 38.76593
## 69    37.51470 1.00000000 1.0000000 0.00000000 17.514696 45.00000
## 701   37.85410 1.00000000 1.0000000 0.14590002 18.000000 44.41640
## 71    34.37815 1.00000000 1.0000000 1.00000000 16.252100 36.63025
## 72    37.84544 1.00000000 1.0000000 1.00000000 18.563623 41.25363
## 73    38.02262 1.00000000 1.0000000 0.00000000 18.067849 44.90727
## 74    39.00000 1.00000000 1.0000000 0.00000000 21.000000 43.58561
## 75    33.18577 1.00000000 1.0000000 0.00000000 14.216728 33.24459
## 761   32.08836 0.08836330 1.0000000 0.00000000 14.000000 31.08556
## 77    32.00000 0.00000000 0.4916294 0.00000000 11.966517 23.27444
## 78    32.89433 0.00000000 1.0000000 0.00000000 13.105665 35.46111
## 79    39.28831 1.00000000 1.0000000 1.00000000 19.525541 41.68649
## 80    39.77418 1.00000000 1.0000000 1.00000000 19.774182 41.22254
## 811   40.52195 1.00000000 1.0000000 1.00000000 20.695935 43.65724
## 82    32.70744 1.00000000 1.0000000 0.00000000 13.707436 33.14628
## 83    32.33800 1.00000000 1.0000000 0.00000000 13.338004 33.33100
## 84    36.77847 1.00000000 1.0000000 0.00000000 18.408218 37.97499
## 85    34.06598 1.00000000 1.0000000 1.00000000 16.043987 36.10997
## 861   34.00000 1.00000000 1.0000000 0.41606115 14.832122 37.16788
## 87    42.02707 1.00000000 1.0000000 1.00000000 23.135169 50.27034
## 88    37.95841 0.97920373 1.0000000 0.00000000 17.979204 44.85443
## 89    35.09660 0.00000000 1.0000000 0.00000000 15.193198 37.09660
```

```
## 90    34.44918 0.77540759 1.0000000 0.00000000 14.673777 38.00000
## 91    39.04211 1.00000000 1.0000000 0.00000000 21.042108 45.95789
## 92    39.27375 1.00000000 1.0000000 0.00000000 21.273751 45.72625
## 93    38.63760 1.00000000 1.0000000 0.00000000 20.275197 45.27520
## 941   38.75914 1.00000000 1.0000000 0.00000000 20.277406 45.75914
## 95    37.54208 1.00000000 1.0000000 0.00000000 18.457922 45.00000
## 96    38.00000 1.00000000 1.0000000 0.00000000 18.104079 44.89592
## 97    40.00000 1.00000000 1.0000000 0.00000000 21.907902 45.13815
## 981   38.36209 1.00000000 1.0000000 0.00000000 18.362089 45.54313
## 99    38.81232 1.00000000 1.0000000 0.00000000 19.406162 49.78152
## 100   37.11159 1.00000000 1.0000000 0.00000000 19.000000 45.66955
## 1011  38.09032 1.00000000 1.0000000 0.00000000 19.180644 50.54839
## 1021  37.25436 1.00000000 1.0000000 0.00000000 19.000000 46.52618
## 103   41.39896 1.00000000 1.0000000 0.00000000 21.398965 52.89638
## 104   39.74120 1.00000000 1.0000000 0.00000000 21.247068 48.22361
## 1051  40.09700 1.00000000 1.0000000 0.00000000 20.572748 53.09700
## 106   38.52676 1.00000000 1.0000000 0.00000000 19.790139 44.26338
## 107   38.18171 1.00000000 1.0000000 0.00000000 18.363414 45.00000
## 108   39.49709 1.00000000 1.0000000 0.00000000 21.497092 45.50291
## 1091  36.15444 1.00000000 0.0000000 0.00000000 17.308888 39.46333
## 1101  37.11733 1.00000000 0.1173312 0.00000000 19.000000 42.23466
## 111   37.00000 1.00000000 0.2352689 0.00000000 19.000000 42.70581
## 112   40.19306 1.00000000 1.0000000 1.00000000 20.257415 42.54997
## 1131  41.05883 1.00000000 1.0000000 1.00000000 21.411771 45.46472
## 114   42.93978 1.00000000 1.0000000 1.00000000 23.919704 51.79725
## 115   34.34511 1.00000000 1.0000000 0.00000000 14.345112 38.60395
## 1161  35.68174 1.00000000 1.0000000 0.00000000 16.102175 40.52261
## 117   37.22678 1.00000000 1.0000000 0.00000000 17.226778 43.64686
## 118   39.08115 1.00000000 1.0000000 0.00000000 19.540575 54.24345
## 119   40.00000 1.00000000 1.0000000 0.00000000 20.000000 50.39627
## 120   40.00000 1.00000000 1.0000000 0.00000000 21.662444 47.02534
## 121   38.95264 1.00000000 1.0000000 0.00000000 20.905289 45.90529
## 1221  38.00000 1.00000000 1.0000000 0.00000000 18.477664 44.52234
## 1231  37.52073 1.00000000 1.0000000 0.00000000 19.000000 44.47927
## 1241  37.03978 1.00000000 1.0000000 0.00000000 19.000000 45.23868
## 1251  38.61868 1.00000000 1.0000000 0.00000000 20.618683 45.00000
## 126   37.28371 1.00000000 1.0000000 0.00000000 19.000000 44.71629
## 1271  37.00000 1.00000000 0.1650259 0.00000000 19.000000 46.66995
## 1281  37.00000 1.00000000 0.0000000 0.00000000 19.000000 44.85693
## 1291  36.15507 1.00000000 0.0000000 0.00000000 17.310133 40.24053
##         license
## 101   0.0000000
## 12    1.0000000
## 361   0.0000000
## 130   0.0000000
## 247   0.0000000
## 212   0.0000000
## 150   0.0000000
## 76    0.0000000
## 166   0.0000000
## 247.1 0.0000000
## 205   0.0000000
## 264   0.0000000
## 297   0.0000000
```

```
## 191   1.0000000
## 152   0.0000000
## 3     0.0000000
## 1     0.0000000
## 423   0.0000000
## 45    1.0000000
## 163   0.0000000
## 290   1.0000000
## 303   0.0000000
## 369   0.0000000
## 240   0.0000000
## 307   0.0000000
## 352   0.0000000
## 241   0.0000000
## 324   0.0000000
## 12.1  1.0000000
## 163.1 0.0000000
## 370   0.0000000
## 254   0.0000000
## 389   0.0000000
## 70    0.0000000
## 123   0.0000000
## 329   0.0000000
## 422   0.0000000
## 125   0.0000000
## 1.1   0.0000000
## 422.1 0.0000000
## 60    0.0000000
## 23    0.0000000
## 235   1.0000000
## 124   1.0000000
## 206   0.0000000
## 327   0.0000000
## 30    0.0000000
## 281   0.0000000
## 399   0.0000000
## 109   1.0000000
## 98    0.0000000
## 63    0.0000000
## 35    0.0000000
## 216   0.0000000
## 192   1.0000000
## 193   1.0000000
## 377   0.0000000
## 15    0.0000000
## 81    0.0000000
## 203   0.0000000
## 94    1.0000000
## 222   0.0000000
## 189   1.0000000
## 221   0.0000000
## 57    0.0000000
## 57.1  0.0000000
## 162   1.0000000
```

```
## 272   0.0000000
## 256   0.0000000
## 386   0.0000000
## 332   0.0000000
## 269   0.0000000
## 330   0.0000000
## 52    0.0000000
## 98.1  0.0000000
## 129   0.0000000
## 160   0.0000000
## 250   0.0000000
## 330.1 0.0000000
## 101.1 0.0000000
## 333   0.0000000
## 9     0.0000000
## 256.1 0.0000000
## 331   0.0000000
## 181   0.0000000
## 81.1  0.0000000
## 10    0.0000000
## 110   0.0000000
## 386.1 0.0000000
## 393   0.0000000
## 211   0.0000000
## 86    0.0000000
## 192.1 1.0000000
## 328   0.0000000
## 160.1 0.0000000
## 44    0.0000000
## 109.1 1.0000000
## 144   0.0000000
## 101.2 0.0000000
## 184   0.0000000
## 146   0.0000000
## 49    0.0000000
## 19    0.0000000
## 102   1.0000000
## 408   0.0000000
## 192.2 1.0000000
## 187   0.0000000
## 304   0.0000000
## 267   0.0000000
## 23.1  0.0000000
## 281.1 0.0000000
## 144.1 0.0000000
## 376   1.0000000
## 313   1.0000000
## 56    0.0000000
## 116   0.0000000
## 51    0.0000000
## 230   0.0000000
## 217   0.0000000
## 273   0.0000000
## 122   0.0000000
```

```
## 237   0.0000000
## 422.2 0.0000000
## 303.1 0.0000000
## 36    0.0000000
## 174   0.0000000
## 25    0.0000000
## 133   0.0000000
## 298   0.0000000
## 3.1   0.0000000
## 312   0.0000000
## 181.1 0.0000000
## 292   0.0000000
## 266   0.0000000
## 221.1 0.0000000
## 283   1.0000000
## 259   0.0000000
## 113   0.0000000
## 42    0.0000000
## 405   0.0000000
## 299   0.0000000
## 241.1 0.0000000
## 49.1  0.0000000
## 214   0.0000000
## 102.1 1.0000000
## 379   0.0000000
## 6     1.0000000
## 94.1  1.0000000
## 52.1  0.0000000
## 128   0.0000000
## 293   0.0000000
## 422.3 0.0000000
## 286   0.0000000
## 198   0.0000000
## 168   0.0000000
## 399.1 0.0000000
## 49.2  0.0000000
## 36.1  0.0000000
## 377.1 0.0000000
## 165   0.0000000
## 18    0.0000000
## 254.1 0.0000000
## 184.1 0.0000000
## 242   0.0000000
## 105   0.0000000
## 62    0.0000000
## 47    0.0000000
## 199   1.0000000
## 214.1 0.0000000
## 56.1  0.0000000
## 70.1  0.0000000
## 309   0.0000000
## 127   1.0000000
## 172   0.0000000
## 177   1.0000000
```

```
## 182   0.0000000
## 219   1.0000000
## 229   1.0000000
## 249   1.0000000
## 265   1.0000000
## 277   1.0000000
## 279   1.0000000
## 287   1.0000000
## 289   0.0000000
## 306   0.0000000
## 335   1.0000000
## 337   1.0000000
## 338   1.0000000
## 351   1.0000000
## 353   1.0000000
## 355   1.0000000
## 364   1.0000000
## 367   0.0000000
## 368   1.0000000
## 373   1.0000000
## 396   1.0000000
## 401   0.0000000
## 406   0.0000000
## 411   1.0000000
## 418   0.0000000
## 420   1.0000000
## 424   1.0000000
## 425   1.0000000
## 426   1.0000000
## 427   1.0000000
## 430   1.0000000
## 431   1.0000000
## 435   1.0000000
## 436   1.0000000
## 437   1.0000000
## 439   1.0000000
## 440   1.0000000
## 441   1.0000000
## 442   1.0000000
## 443   1.0000000
## 131   1.0000000
## 2     1.0000000
## 310   1.0000000
## 4     0.5238226
## 5     0.0000000
## 610   0.0000000
## 7     1.0000000
## 8     1.0000000
## 910   1.0000000
## 1010  0.0000000
## 11    0.0000000
## 1210  0.2300969
## 13    1.0000000
## 14    1.0000000
```

```
## 151   1.0000000
## 16    1.0000000
## 17    1.0000000
## 183   1.0000000
## 194   1.0000000
## 20    1.0000000
## 21    1.0000000
## 22    1.0000000
## 231   1.0000000
## 24    1.0000000
## 251   1.0000000
## 26    1.0000000
## 27    1.0000000
## 28    1.0000000
## 29    1.0000000
## 301   1.0000000
## 31    1.0000000
## 32    1.0000000
## 33    1.0000000
## 34    0.0000000
## 354   0.0000000
## 362   0.0000000
## 37    0.0000000
## 38    0.0000000
## 39    0.3011425
## 40    1.0000000
## 41    1.0000000
## 421   1.0000000
## 43    1.0000000
## 444   1.0000000
## 451   1.0000000
## 46    1.0000000
## 471   1.0000000
## 48    1.0000000
## 491   1.0000000
## 50    1.0000000
## 511   1.0000000
## 521   1.0000000
## 53    1.0000000
## 54    1.0000000
## 55    1.0000000
## 561   1.0000000
## 571   1.0000000
## 58    1.0000000
## 59    1.0000000
## 601   1.0000000
## 61    0.6502615
## 621   0.3952538
## 631   0.0000000
## 64    1.0000000
## 65    1.0000000
## 66    1.0000000
## 67    1.0000000
## 68    1.0000000
```

```
## 69    1.0000000
## 701   1.0000000
## 71    1.0000000
## 72    1.0000000
## 73    0.9773836
## 74    0.5265901
## 75    0.0000000
## 761   0.0000000
## 77    0.0000000
## 78    0.8943346
## 79    1.0000000
## 80    1.0000000
## 811   1.0000000
## 82    0.1462821
## 83    0.3309978
## 84    0.0000000
## 85    1.0000000
## 861   1.0000000
## 87    1.0000000
## 88    1.0000000
## 89    1.0000000
## 90    1.0000000
## 91    1.0000000
## 92    1.0000000
## 93    1.0000000
## 941   1.0000000
## 95    1.0000000
## 96    1.0000000
## 97    1.0000000
## 981   1.0000000
## 99    1.0000000
## 100   1.0000000
## 1011  1.0000000
## 1021  1.0000000
## 103   1.0000000
## 104   1.0000000
## 1051  1.0000000
## 106   1.0000000
## 107   1.0000000
## 108   1.0000000
## 1091  1.0000000
## 1101  1.0000000
## 111   1.0000000
## 112   1.0000000
## 1131  1.0000000
## 114   1.0000000
## 115   1.0000000
## 1161  1.0000000
## 117   1.0000000
## 118   1.0000000
## 119   1.0000000
## 120   1.0000000
## 121   1.0000000
## 1221  1.0000000
```

```
## 1231  1.0000000
## 1241  1.0000000
## 1251  1.0000000
## 126   1.0000000
## 1271  1.0000000
## 1281  1.0000000
## 1291  1.0000000

#Predicting Model on Train data
pred_bag_train <- predict(bagModel, data= balanced_trainData1)
pred_bag_train1 <- ifelse(pred_bag_train<0.5,0,1)
pred_bag_train1 <- as.factor(pred_bag_train1)

caret::confusionMatrix(pred_bag_train1,balanced_trainData1$Transport)

## Warning in confusionMatrix.default(pred_bag_train1,
## balanced_trainData1$Transport): Levels are not in the same order for
## reference and data. Refactoring data to match.

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1 172 172
##
##                Accuracy : 0.5
##                  95% CI : (0.4459, 0.5541)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.5215
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0
##             Specificity : 1.0
##          Pos Pred Value : NaN
##          Neg Pred Value : 0.5
##              Prevalence : 0.5
##          Detection Rate : 0.0
##    Detection Prevalence : 0.0
##       Balanced Accuracy : 0.5
##
##        'Positive' Class : 0
##

# Predicting Model on Test data

pred_bag_test <- predict(bagModel,testData1)
pred_bag_test1 <- ifelse(pred_bag_test<0.4,0,1)
pred_bag_test1 <- as.factor(pred_bag_test1)

caret::confusionMatrix(pred_bag_test1,testData1$Transport)
```

```
## Warning in confusionMatrix.default(pred_bag_test1, testData1$Transport):
## Levels are not in the same order for reference and data. Refactoring data
## to match.

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   0   0
##          1 115  18
##
##                Accuracy : 0.1353
##                  95% CI : (0.0822, 0.2054)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.1353
##              Prevalence : 0.8647
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##

# Performing Bagging using different Approach
bagModel2 <- train(Transport~.,data = balanced_trainData1,
                   method = "treebag",
                   trControl = trainControl(method = "cv", number = 10),
                   nbagg= 200,
                   control = rpart.control(minsplit = 2,cp=0))

pred_bag2_train <- predict(bagModel2, data= balanced_trainData1)
#pred_bag2_train1 <- ifelse(pred_bag2_train<0.5,0,1)
pred_bag2_train1 <- as.factor(pred_bag2_train)

caret::confusionMatrix(pred_bag2_train1,balanced_trainData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 172   0
##          1   0 172
##
##                Accuracy : 1
##                  95% CI : (0.9893, 1)
##     No Information Rate : 0.5
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : 0
##
```

```
# Predicting Model on Test data

pred_bag2_test <- predict(bagModel2,testData1)
#pred_bag2_test1 <- ifelse(pred_bag_test<0.4,0,1)
pred_bag2_test1 <- as.factor(pred_bag2_test)

caret::confusionMatrix(pred_bag2_test1,testData1$Transport)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 113   2
##          1   2  16
##
##                Accuracy : 0.9699
##                  95% CI : (0.9248, 0.9917)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 3.661e-05
##
##                   Kappa : 0.8715
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9826
##             Specificity : 0.8889
##          Pos Pred Value : 0.9826
##          Neg Pred Value : 0.8889
##              Prevalence : 0.8647
##          Detection Rate : 0.8496
##    Detection Prevalence : 0.8647
##       Balanced Accuracy : 0.9357
##
##        'Positive' Class : 0
##
```

```r
# Performing boosting Model
str(balanced_trainData1)

## 'data.frame':    344 obs. of  8 variables:
##  $ Age      : num  30 27 30 24 30 25 29 20 27 30 ...
##  $ Gender   : num  0 1 1 1 0 1 1 0 0 0 ...
##  $ Engineer : num  1 1 1 1 0 1 1 0 1 0 ...
##  $ MBA      : num  0 0 0 1 0 0 0 1 0 0 ...
##  $ Work.Exp : num  8 4 6 0 6 3 9 1 5 6 ...
##  $ Salary   : num  14.7 13.5 15.8 7.9 15.6 10.7 23.8 8.5 12.8 15.6 ...
##  $ license  : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...

features_train <- as.matrix(balanced_trainData1[,1:7])
#str(features_train)
label_train <- as.matrix(balanced_trainData1[,8])
str(balanced_trainData1)

## 'data.frame':    344 obs. of  8 variables:
##  $ Age      : num  30 27 30 24 30 25 29 20 27 30 ...
##  $ Gender   : num  0 1 1 1 0 1 1 0 0 0 ...
##  $ Engineer : num  1 1 1 1 0 1 1 0 1 0 ...
##  $ MBA      : num  0 0 0 1 0 0 0 1 0 0 ...
##  $ Work.Exp : num  8 4 6 0 6 3 9 1 5 6 ...
##  $ Salary   : num  14.7 13.5 15.8 7.9 15.6 10.7 23.8 8.5 12.8 15.6 ...
##  $ license  : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Transport: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...

features_test <- as.matrix(testData1[,1:7])
str(features_test)

##  num [1:133, 1:7] 23 28 27 26 25 25 23 26 24 30 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:133] "2" "4" "5" "8" ...
##   ..$ : chr [1:7] "Age" "Gender" "Engineer" "MBA" ...

xgbModel <- xgboost(
  data = features_train,
  label = label_train,
  eta = 1,
  max_depth = 100,
  min_child_weight = 3,
  nrounds = 1000,
  nfold = 10,
  objective = "binary:logistic",
  verbose = 0,
  early_stopping_rounds = 10)

summary(xgbModel)

##                  Length Class              Mode
## handle              1   xgb.Booster.handle externalptr
## raw              9001   -none-             raw
## best_iteration      1   -none-             numeric
## best_ntreelimit     1   -none-             numeric
## best_score          1   -none-             numeric
```

```
## niter                1   -none-            numeric
## evaluation_log       2   data.table        list
## call                18   -none-            call
## params               6   -none-            list
## callbacks            2   -none-            list
## feature_names        7   -none-            character
## nfeatures            1   -none-            numeric
```

```
# Performing Prediction on Train data.
str(label_train)
```

```
##  chr [1:344, 1] "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" ...
```

```
pred_xgb_train <- predict(xgbModel, newdata = features_train)
pred_xgb_train1 <- ifelse(pred_xgb_train<.5,0,1)
pred_xgb_train1<- as.factor(pred_xgb_train1)
caret::confusionMatrix(pred_xgb_train1, balanced_trainData1$Transport)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   0   1
##          0 171   1
##          1   1 171
## 
##                Accuracy : 0.9942
##                  95% CI : (0.9792, 0.9993)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
## 
##                   Kappa : 0.9884
## 
##  Mcnemar's Test P-Value : 1
## 
##             Sensitivity : 0.9942
##             Specificity : 0.9942
##          Pos Pred Value : 0.9942
##          Neg Pred Value : 0.9942
##              Prevalence : 0.5000
##          Detection Rate : 0.4971
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9942
## 
##        'Positive' Class : 0
## 
```

```
# Performing Prediction on Test data.
str(features_test)
```

```
##  num [1:133, 1:7] 23 28 27 26 25 25 23 26 24 30 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:133] "2" "4" "5" "8" ...
##   ..$ : chr [1:7] "Age" "Gender" "Engineer" "MBA" ...
```

```r
pred_xgb_test <- predict(xgbModel,newdata = features_test)
predict(xgbModel,features_test)
```

```
##   [1] 0.005553108 0.002608013 0.005553108 0.005553108 0.005553108
##   [6] 0.002608013 0.005553108 0.005553108 0.005553108 0.005553108
##  [11] 0.005553108 0.005553108 0.002608013 0.108460173 0.005553108
##  [16] 0.013055790 0.002608013 0.002608013 0.005553108 0.005553108
##  [21] 0.005553108 0.022691119 0.005553108 0.005553108 0.002608013
##  [26] 0.005553108 0.005553108 0.005553108 0.005553108 0.005553108
##  [31] 0.221148297 0.005553108 0.005553108 0.013055790 0.002608013
##  [36] 0.101477616 0.005553108 0.005553108 0.038132731 0.005553108
##  [41] 0.931666434 0.038132731 0.002608013 0.113587439 0.005553108
##  [46] 0.005553108 0.005553108 0.005553108 0.002608013 0.053414010
##  [51] 0.992889643 0.002608013 0.002608013 0.005553108 0.005553108
##  [56] 0.005553108 0.002608013 0.101477616 0.005553108 0.027869817
##  [61] 0.027869817 0.006156276 0.013055790 0.005553108 0.005553108
##  [66] 0.002608013 0.002608013 0.005553108 0.005553108 0.013055790
##  [71] 0.990905762 0.005553108 0.002608013 0.005553108 0.221148297
##  [76] 0.005553108 0.013055790 0.057436731 0.960263252 0.152070001
##  [81] 0.002608013 0.511994004 0.013055790 0.005553108 0.005553108
##  [86] 0.969351947 0.013055790 0.006156276 0.002608013 0.992889643
##  [91] 0.013055790 0.005553108 0.005553108 0.033662852 0.972500682
##  [96] 0.002608013 0.002608013 0.005553108 0.005553108 0.005553108
## [101] 0.013055790 0.005553108 0.005553108 0.002608013 0.005553108
## [106] 0.637289286 0.005553108 0.982650340 0.005553108 0.005553108
## [111] 0.992889643 0.005553108 0.005553108 0.033662852 0.005553108
## [116] 0.013055790 0.948893011 0.101477616 0.002608013 0.005553108
## [121] 0.013055790 0.005553108 0.992889643 0.002608013 0.005553108
## [126] 0.002608013 0.983565152 0.254781544 0.992889643 0.006156276
## [131] 0.911611795 0.006156276 0.982650340
```

```r
pred_xgb_test1 <- ifelse(pred_xgb_test<.5,0,1)
pred_xgb_test1<- as.factor(pred_xgb_test1)
caret::confusionMatrix(pred_xgb_test1, testData1$Transport)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   0   1
##          0 114   2
##          1   1  16
## 
##                Accuracy : 0.9774
##                  95% CI : (0.9355, 0.9953)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 6.803e-06
## 
##                   Kappa : 0.9013
## 
##  Mcnemar's Test P-Value : 1
## 
##             Sensitivity : 0.9913
##             Specificity : 0.8889
##          Pos Pred Value : 0.9828
##          Neg Pred Value : 0.9412
```

```
##              Prevalence : 0.8647
##          Detection Rate : 0.8571
##    Detection Prevalence : 0.8722
##       Balanced Accuracy : 0.9401
##
##        'Positive' Class : 0
##
```

```
vec_xgb <- vector()
lr <- c(0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1)
md <- c(1,3,5,7,9,11,13,15)
nr <- c(2,25,50,75,100,500,1000)

for (i in nr){
  xgbModel_ref <- xgboost(
    data = features_train,
    label = label_train,
    eta = 0.1,
    max_depth = 100,
    min_child_weight = 3,
    nrounds = 10000,
    nfold = 10,
    objective = "binary:logistic",
    verbose = 0,
    early_stopping_rounds = 10)

  xgb.pred.class <- predict(xgbModel_ref,features_test)
  vec_xgb <- cbind(vec_xgb,sum(testData1$Transport==1 & xgb.pred.class > 0.5))
}
vec_xgb
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]   16   16   16   16   16   16   16
```

```
xgbModel1 <- xgboost(
  data = features_train,
  label = label_train,
  eta = 0.1,
  max_depth = 1,
  min_child_weight = 3,
  nrounds = 2,
  nfold = 10,
  objective = "binary:logistic",
  verbose = 1,
  early_stopping_rounds = 10)
```

```
## [1]  train-error:0.052326
## Will train until train_error hasn't improved in 10 rounds.
##
## [2]  train-error:0.052326
```

```
# Performing Prediction on Train data.
pred_xgb_ref_train <- predict(xgbModel1, newdata = features_train)
pred_xgb_ref_train1 <- ifelse(pred_xgb_ref_train<.5,0,1)
```

```
pred_xgb_ref_train1<- as.factor(pred_xgb_ref_train1)
caret::confusionMatrix(pred_xgb_train1, balanced_trainData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 171   1
##          1   1 171
##
##                Accuracy : 0.9942
##                  95% CI : (0.9792, 0.9993)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9884
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9942
##             Specificity : 0.9942
##          Pos Pred Value : 0.9942
##          Neg Pred Value : 0.9942
##              Prevalence : 0.5000
##          Detection Rate : 0.4971
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9942
##
##        'Positive' Class : 0
##

# Performing Prediction on Test data.

pred_xgb_ref_test <- predict(xgbModel1, newdata= features_test)
pred_xgb_ref_test1 <- ifelse(pred_xgb_ref_test<.5,0,1)
pred_xgb_ref_test1<- as.factor(pred_xgb_ref_test1)
caret::confusionMatrix(pred_xgb_ref_test1, testData1$Transport)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 110   1
##          1   5  17
##
##                Accuracy : 0.9549
##                  95% CI : (0.9044, 0.9833)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.0005588
##
##                   Kappa : 0.8238
##
##  Mcnemar's Test P-Value : 0.2206714
##
##             Sensitivity : 0.9565
```

```
##              Specificity : 0.9444
##           Pos Pred Value : 0.9910
##           Neg Pred Value : 0.7727
##               Prevalence : 0.8647
##           Detection Rate : 0.8271
##     Detection Prevalence : 0.8346
##        Balanced Accuracy : 0.9505
##
##         'Positive' Class : 0
##
```

```
# Performing XGBoosting using different Approach

carsxgb <- train(Transport~.,balanced_trainData1,
                 trControl = trainControl("cv", number = 2),method = "xgbTree")


pred_xgb2_train <- predict(carsxgb, balanced_trainData1)
pred_xgb2_train <- as.factor(pred_xgb2_train)
caret::confusionMatrix(pred_xgb2_train, balanced_trainData1$Transport)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 172   0
##          1   0 172
##
##                Accuracy : 1
##                  95% CI : (0.9893, 1)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : 0
##
```

```
pred_xgb2_test <- predict(carsxgb, testData1)
pred_xgb2_test <- as.factor(pred_xgb2_test)
caret::confusionMatrix(pred_xgb2_test, testData1$Transport)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   0   1
##          0 113   1
##          1   2  17
##
##                Accuracy : 0.9774
##                  95% CI : (0.9355, 0.9953)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 6.803e-06
##
##                   Kappa : 0.9058
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9826
##             Specificity : 0.9444
##          Pos Pred Value : 0.9912
##          Neg Pred Value : 0.8947
##              Prevalence : 0.8647
##          Detection Rate : 0.8496
##    Detection Prevalence : 0.8571
##       Balanced Accuracy : 0.9635
##
##        'Positive' Class : 0
##
```