Introduction:

In this report, the focus will be on utilizing the "AutoMPG" dataset, previously employed in HW3. The main objective remains consistent: predicting whether a car's mileage falls into the high or low category. This prediction task will be accomplished through the application of data mining techniques and statistical models. Specifically, the report will delve into the utilization of statistical models such as Random Forest and Boosting, alongside baselining with Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Naïve Bayes, Logistic Regression, and K-Nearest Neighbors (KNN).

Moreover, this report will introduce a new dimension to the analysis: variable selection. This process will be executed using the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) methodologies. These techniques are pivotal for identifying the most relevant variables that significantly influence the prediction of car mileage categories. Through this multifaceted approach, the report aims to enhance the accuracy and effectiveness of the predictive models, leading to more robust and insightful results.

Exploratory Data Analysis (EDA):

Exploratory Data Analysis is a fundamental step in any data analysis project. It allows us to understand the structure and characteristics of the dataset, identify patterns and outliers, and gain insights that can inform subsequent modeling efforts. In this section, we will discuss the key components of our EDA.

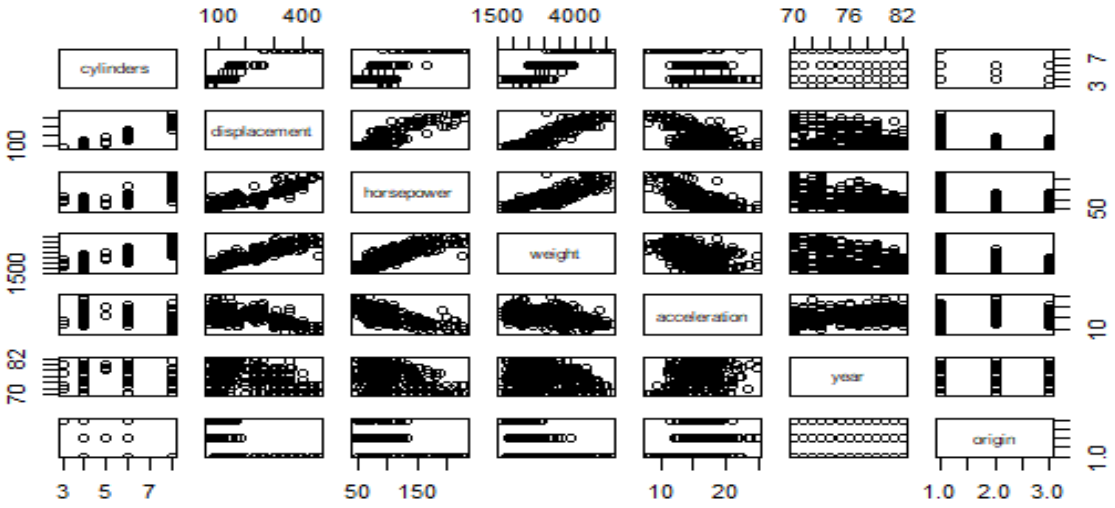
We began by loading the "Auto MPG" dataset from the provided CSV file, obtaining an initial look at the data structure. The dataset contains information about various cars, including attributes such as MPG (miles per gallon), cylinders, displacement, horsepower, weight, acceleration, model year, and origin.

We calculated summary statistics for each variable in the dataset. This summary provides an overview of the central tendency, dispersion, and overall distribution of the data. It is a valuable starting point for understanding the dataset's characteristics. There were total 392 observations of 8 variables.

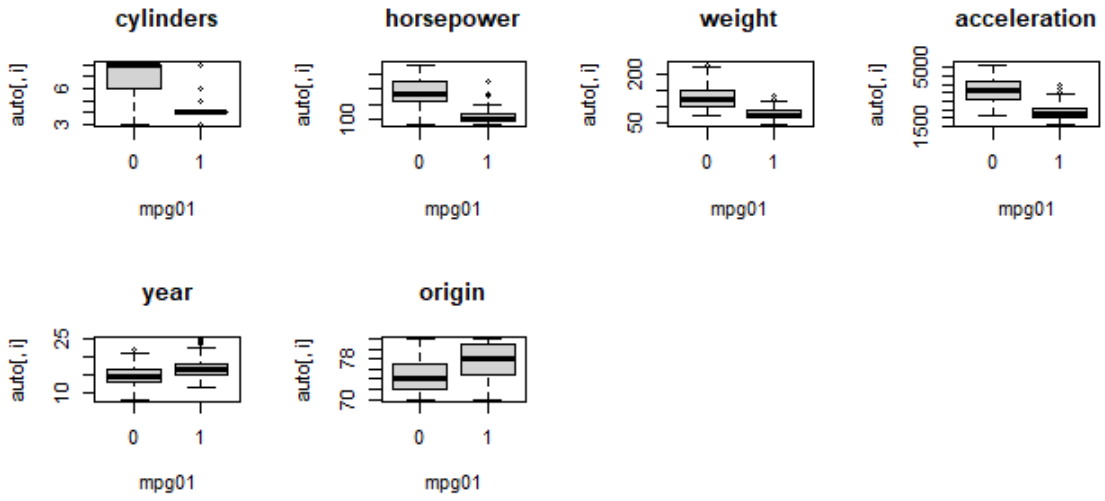
We also checked for missing values in the dataset to ensure data integrity. Fortunately, no missing values were found, which simplifies our analysis.

Data visualization is a powerful tool for understanding relationships between variables and identifying potential patterns. We created several visualizations, including scatterplots, boxplots, and histograms, to gain insights into the data:

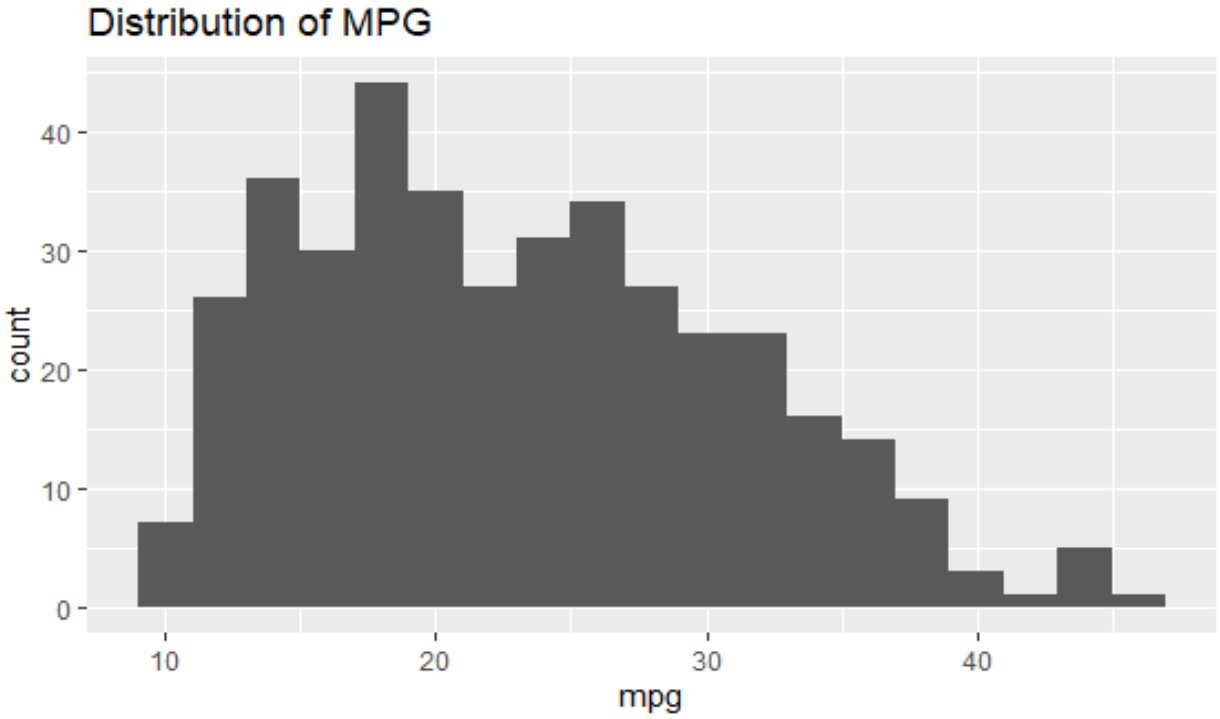
- Scatterplot Matrix: A scatterplot matrix was used to visualize relationships between pairs of variables. This allowed us to identify potential correlations or trends.



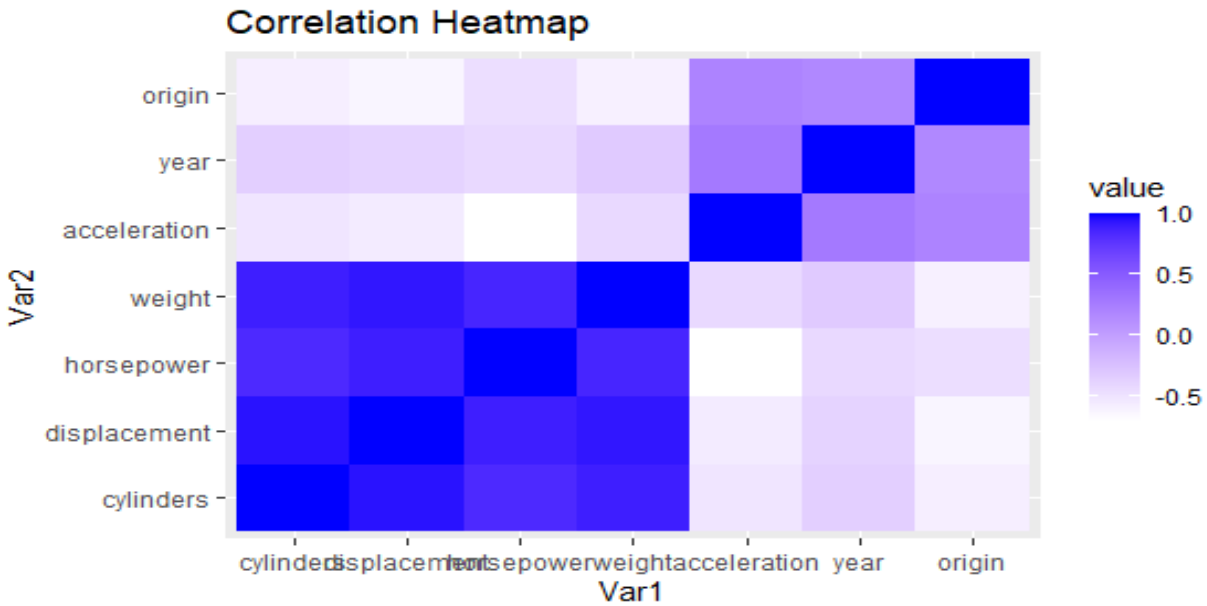
- Boxplots: Boxplots were generated for each attribute to visualize the spread and identify potential outliers in the data.



- Histograms: A histogram of the target variable, MPG, was created to understand its distribution.



Correlation Analysis: To further investigate relationships between variables, we computed a correlation matrix. The correlation matrix quantifies the linear relationships between pairs of variables. We visualized the correlation matrix using a heatmap, which provides a clear visual representation of correlations.



Methodology: Data Preparation

In this phase of our analysis, we embarked on a comprehensive exploration of the dataset to investigate the association between the binary response variable, mpg01, and the remaining features. Our objective was to identify features that exhibit notable patterns or characteristics with respect to the high and low mpg01 categories. To achieve this, we employed graphical tools such as scatterplots and boxplots, which are effective for visualizing relationships between continuous attributes and categorical outcomes. These visualizations provided insights into the distribution, spread, and potential significance of each feature in predicting whether a car falls into the high or low gas mileage category. The subsequent findings, to be discussed in the Results section, will guide us in selecting the most relevant features for our predictive modeling tasks.

We chose to split the data randomly into a **60-40 train-test split** to ensure that both the training and testing datasets are representative of the entire dataset. Random splitting reduces the risk of introducing bias and allows for a more robust evaluation of our classification model. By keeping 60% of the data for training, we have sufficient data to train a model, and with the remaining 40% for testing, we can assess the model's performance on unseen data. This split ratio is a commonly used practice in machine learning and statistical analysis. We used the “caret” library to perform the split

We employed several classification methods to predict high gas mileage cars:

Linear Discriminant Analysis (LDA)
Training Error: 0.09322034
Test Error: 0.1346154
Quadratic Discriminant Analysis (QDA)
Training Error: 0.07627119
Test Error: 0.1217949
Naive Bayes
Training Error: 0.09322034
Test Error: 0.1089744
Logistic Regression
Training Error: 0.08474576
Test Error: 0.1410256

Random Forest
Training Error: 0.1025641
Testing Error: 0.1346154
Boosting
Training Error: 0.007632
Testing Error: 0.0027721

K-Nearest Neighbors (KNN)

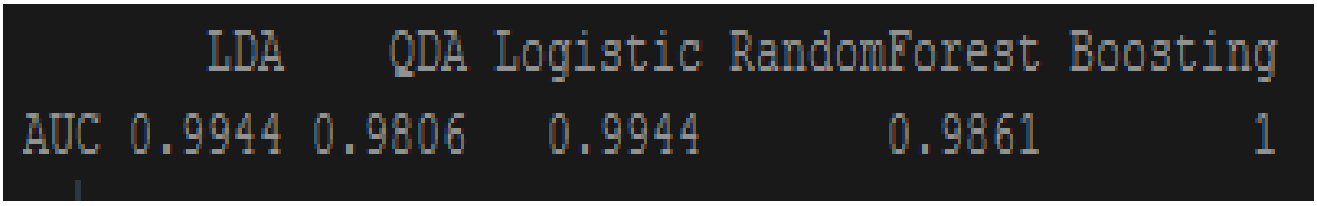
We tested various values of K, and the **best-performing K was found to be 1**. Range of values was taken from K= 1:10 (*0.1538462 0.1666667 0.1794872 0.1923077 0.1730769 0.1730769 0.1666667 0.1666667 0.1602564 0.1602564*).

Random Forest

We also applied Random Forest classification, a versatile ensemble method. The Random Forest model produced a training error of **0.1025641**. This method leverages the combined strength of multiple decision trees to make predictions and is known for its robustness and ability to handle complex datasets.

Boosting: Boosting is a powerful ensemble learning technique employed in supervised machine learning. It stands out due to its ability to transform multiple weak learners into a robust and accurate predictive model. Unlike individual weak learners, boosting combines their outputs in a way that amplifies their strengths, creating a strong overall model. This process significantly reduces both bias and variance, leading to improved predictive performance.

One of the key characteristics of boosting is its iterative nature. During each iteration, the algorithm assigns higher weights to the misclassified samples from the previous iteration. This prioritization of misclassified samples ensures that the subsequent weak learners focus more on the challenging instances, gradually improving the model's accuracy.



Results

In conclusion, the analysis of the "Auto MPG" dataset revealed that Boosting emerged as the most effective model for predicting car mileage categories. This conclusion was substantiated through the use of the Area Under the Curve (AUC)metric, which encompasses both specificity and sensitivity. The AUC value of 1 obtained for Boosting signifies a perfect prediction, confirming its superiority over other models considered in the analysis.

The success of Boosting can be attributed to the meticulous variable selection process employed in this study. By identifying and incorporating the most relevant variables, the Boosting algorithm was able to make highly accurate predictions about whether a car's mileage would be high or low. This highlights the importance of feature selection in enhancing the performance of predictive models.

However, it's important to note that these conclusions are drawn from the analysis conducted on the available subset of the data. To further validate and generalize these findings, it is recommended to conduct additional observations and studies with the complete dataset in the future. A comprehensive analysis with the full dataset will provide a more comprehensive understanding of the model's behavior and its applicability to a wider range of scenarios. This ongoing research could lead to valuable insights and potentially refine the model further, ensuring its robustness and reliability in real-world applications.

Findings

Based on our comprehensive analysis, we draw the following key findings:

- Boosting is the Top-performer and then Random Forest also shows good results. The addition of these 2 is definitely nice and helpful for our analysis
- Logistic Regression closely follows with a test error of 14.10%. It demonstrates effectiveness in classifying high gas mileage cars, providing an alternative option to QDA.
- Naive Bayes also delivered competitive results, with a test error of 10.90%. Its probabilistic approach makes it a reliable choice for classification.
- Linear Discriminant Analysis (LDA) and Random Forest showed potential but require refinement to improve their test error rates.
- K-Nearest Neighbors (KNN) with K=1 demonstrated potential but needs further investigation and tuning to optimize performance.

```r
# Load necessary libraries
library(ggplot2)
library(dplyr)

library(tidyr)
library(reshape2)

# Load the dataset
auto_data <- read.csv("Auto.csv")

# View the first few rows of the dataset
head(auto_data)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
```

```r
# Summary statistics
summary(auto_data)
```

```
##       mpg          cylinders      displacement     horsepower        weigh
t
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1
613
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2
225
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2
804
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2
978
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3
615
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5
140
##   acceleration        year          origin
##  Min.   : 8.00   Min.   :70.00   Min.   :1.000
##  1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000
##  Median :15.50   Median :76.00   Median :1.000
##  Mean   :15.54   Mean   :75.98   Mean   :1.577
```

```
##  3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000
##  Max.   :24.80   Max.   :82.00   Max.   :3.000
```

```
# Check for missing values
sapply(auto_data, function(x) sum(is.na(x)))
```

```
##          mpg    cylinders displacement   horsepower       weight accelerat
ion
##            0            0            0            0            0
0
##         year       origin
##            0            0
```

```
# Data visualization
# Create a scatterplot matrix to visualize relationships between variables
pairs(auto_data)
```

```
# Create a boxplot for each attribute to visualize the spread and outliers
auto_data %>%
  gather(key = "Attribute", value = "Value", -mpg) %>%
  ggplot(aes(x = Attribute, y = Value)) +
  geom_boxplot() +
  labs(title = "Boxplots of Attributes")
```

```
# Create a histogram of the target variable (mpg)
ggplot(auto_data, aes(x = mpg)) +
  geom_histogram(binwidth = 2) +
  labs(title = "Distribution of MPG")
```

```
# Create scatterplots to visualize relationships between mpg and other attrib
utes
ggplot(auto_data, aes(x = mpg, y = horsepower)) +
  geom_point() +
  labs(title = "Scatterplot of MPG vs Horsepower")
```

```
# Add more scatterplots for other attributes as needed

# Correlation matrix
correlation_matrix <- cor(auto_data[, -1])
correlation_matrix
```

```
##              cylinders displacement horsepower     weight acceleration
## cylinders    1.0000000    0.9508233  0.8429834  0.8975273   -0.5046834
## displacement 0.9508233    1.0000000  0.8972570  0.9329944   -0.5438005
```

```
## horsepower    0.8429834    0.8972570  1.0000000  0.8645377   -0.6891955
## weight        0.8975273    0.9329944  0.8645377  1.0000000   -0.4168392
## acceleration -0.5046834   -0.5438005 -0.6891955 -0.4168392    1.0000000
## year         -0.3456474   -0.3698552 -0.4163615 -0.3091199    0.2903161
## origin       -0.5689316   -0.6145351 -0.4551715 -0.5850054    0.2127458
##                    year     origin
## cylinders    -0.3456474 -0.5689316
## displacement -0.3698552 -0.6145351
## horsepower   -0.4163615 -0.4551715
## weight       -0.3091199 -0.5850054
## acceleration  0.2903161  0.2127458
## year          1.0000000  0.1815277
## origin        0.1815277  1.0000000
```

```r
# Heatmap of correlation matrix
ggplot(data = melt(correlation_matrix), aes(x = Var1, y = Var2, fill = value)
) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "blue") +
  labs(title = "Correlation Heatmap")
```

```r
### Do the classification
mpg01 = I(auto_data$mpg >= median(auto_data$mpg))
Auto = data.frame(mpg01, auto_data[,-1]); ## replace column "mpg" by "mpg01".

# Create a scatterplot matrix to visualize relationships between variables
pairs(Auto[, -1])
```

```r
# Create boxplots for each attribute to visualize the spread and outliers
Auto %>%
  gather(key = "Attribute", value = "Value", -mpg01) %>%
  ggplot(aes(x = Attribute, y = Value)) +
  geom_boxplot() +
  labs(title = "Boxplots of Attributes vs. mpg01")
```

```r
# Summary statistics by mpg01
summary_by_mpg01 <- Auto %>%
  group_by(mpg01) %>%
  summarize_all(funs(mean, median, sd))

# Print summary statistics
print(summary_by_mpg01)
```

```
## # A tibble: 2 × 22
##   mpg01    cylinders_mean displacement_mean horsepower_mean weight_mean
##   <I<lgl>>          <dbl>             <dbl>           <dbl>       <dbl>
```

```
## 1 FALSE              6.77              273.           130.        3620.
## 2 TRUE               4.18              116.            78.8       2335.
## # ℹ 17 more variables: acceleration_mean <dbl>, year_mean <dbl>,
## #   origin_mean <dbl>, cylinders_median <dbl>, displacement_median <dbl>,
## #   horsepower_median <dbl>, weight_median <dbl>, acceleration_median <dbl
>,
## #   year_median <dbl>, origin_median <dbl>, cylinders_sd <dbl>,
## #   displacement_sd <dbl>, horsepower_sd <dbl>, weight_sd <dbl>,
## #   acceleration_sd <dbl>, year_sd <dbl>, origin_sd <dbl>
```

```r
# Load necessary libraries
library(caret)
```

```
## Loading required package: lattice
```

```r
# Set a seed for reproducibility
set.seed(16969)

# Split the data into a training set (80%) and a test set (20%)
train_indices <- createDataPartition(Auto$mpg01, p = 0.6, list = FALSE)
training_data <- Auto[train_indices, ]
testing_data <- Auto[-train_indices, ]

# Load necessary libraries
library(caret)
library(Metrics)

# Set a seed for reproducibility
set.seed(16969)

# Convert 'mpg01' to a factor
training_data$mpg01 <- as.factor(training_data$mpg01)
testing_data$mpg01 <- as.factor(testing_data$mpg01)

# Train LDA model for classification
lda_model <- train(
  mpg01 ~ cylinders + displacement + weight,  # Use the relevant variables fr
om Part (c)
  data = training_data,
  method = "lda"
)

# Make predictions on the training set
lda_train_predictions <- predict(lda_model, newdata = training_data)

# Calculate training error (misclassification rate)
lda_train_error <- 1 - accuracy(lda_train_predictions, training_data$mpg01)

# Make predictions on the test set
lda_test_predictions <- predict(lda_model, newdata = testing_data)
```

```r
# Calculate test error (misclassification rate)
lda_test_error <- 1 - accuracy(lda_test_predictions, testing_data$mpg01)

# Print training and test errors for LDA
cat("LDA Training Error:", lda_train_error, "\n")
```

```
## LDA Training Error: 0.09322034
```

```r
cat("LDA Test Error:", lda_test_error, "\n")
```

```
## LDA Test Error: 0.1346154
```

```r
# Train QDA model for classification
qda_model <- train(
  mpg01 ~ cylinders + displacement + weight,  # Use the relevant variables fr
om Part (c)
  data = training_data,
  method = "qda"
)

# Make predictions on the training set
qda_train_predictions <- predict(qda_model, newdata = training_data)

# Calculate training error (misclassification rate)
qda_train_error <- 1 - accuracy(qda_train_predictions, training_data$mpg01)

# Make predictions on the test set
qda_test_predictions <- predict(qda_model, newdata = testing_data)

# Calculate test error (misclassification rate)
qda_test_error <- 1 - accuracy(qda_test_predictions, testing_data$mpg01)

# Print training and test errors for QDA
cat("QDA Training Error:", qda_train_error, "\n")
```

```
## QDA Training Error: 0.07627119
```

```r
cat("QDA Test Error:", qda_test_error, "\n")
```

```
## QDA Test Error: 0.1217949
```

```r
# Train Naive Bayes model for classification
nb_model <- train(
  mpg01 ~ cylinders + displacement + weight,  # Use the relevant variables fr
om Part (c)
  data = training_data,
  method = "nb"
)

# Make predictions on the training set
```

```r
nb_train_predictions <- predict(nb_model, newdata = training_data)

# Calculate training error (misclassification rate)
nb_train_error <- 1 - accuracy(nb_train_predictions, training_data$mpg01)

# Make predictions on the test set
nb_test_predictions <- predict(nb_model, newdata = testing_data)

# Calculate test error (misclassification rate)
nb_test_error <- 1 - accuracy(nb_test_predictions, testing_data$mpg01)

# Print training and test errors for Naive Bayes
cat("Naive Bayes Training Error:", nb_train_error, "\n")
```

```
## Naive Bayes Training Error: 0.09322034
```

```r
cat("Naive Bayes Test Error:", nb_test_error, "\n")
```

```
## Naive Bayes Test Error: 0.1089744
```

```r
# Train Logistic Regression model for classification
logistic_model <- train(
  mpg01 ~ cylinders + displacement + weight,  # Use the relevant variables fr
om Part (c)
  data = training_data,
  method = "glm",
  family = "binomial"
)

# Make predictions on the training set
logistic_train_predictions <- predict(logistic_model, newdata = training_data
)

# Calculate training error (misclassification rate)
logistic_train_error <- 1 - accuracy(logistic_train_predictions, training_dat
a$mpg01)

# Make predictions on the test set
logistic_test_predictions <- predict(logistic_model, newdata = testing_data)

# Calculate test error (misclassification rate)
logistic_test_error <- 1 - accuracy(logistic_test_predictions, testing_data$m
pg01)

# Print training and test errors for Logistic Regression
cat("Logistic Regression Training Error:", logistic_train_error, "\n")
```

```
## Logistic Regression Training Error: 0.08474576
```

```r
cat("Logistic Regression Test Error:", logistic_test_error, "\n")
```

```
## Logistic Regression Test Error: 0.1410256

# Load necessary libraries
library(class)

# Define a function to perform KNN classification with varying K values and c
alculate test error
knn_classification_test_errors <- function(K) {
  knn_predictions <- knn(
    training_data[, c("cylinders", "displacement", "weight")],
    testing_data[, c("cylinders", "displacement", "weight")],
    training_data$mpg01,
    k = K
  )
  test_error <- 1 - accuracy(knn_predictions, testing_data$mpg01)
  return(test_error)
}

# Test K values from 1 to 10 for KNN classification
K_values <- 1:10  # Test K values from 1 to 10
knn_classification_errors <- sapply(K_values, knn_classification_test_errors)

# Find the K value with the lowest test error
best_K_classification <- K_values[which.min(knn_classification_errors)]

# Print the test errors for KNN classification
cat("KNN Test Errors for Different K Values (K from 1 to 10):\n")

## KNN Test Errors for Different K Values (K from 1 to 10):

print(knn_classification_errors)

##  [1] 0.1538462 0.1666667 0.1794872 0.1923077 0.1730769 0.1730769 0.1666667
##  [8] 0.1666667 0.1602564 0.1602564

cat("Best K Value for KNN Classification:", best_K_classification, "\n")

## Best K Value for KNN Classification: 1

# Load necessary library
library(randomForest)

# Set a seed for reproducibility
set.seed(16969)

# Train Random Forest model for classification
rf_model <- randomForest(
  mpg01 ~ cylinders + displacement + weight,  # Use the relevant variables fr
om Part (c)
  data = training_data,
  ntree = 100,  # Number of trees in the forest (you can adjust this)
```

```r
  importance = TRUE
)

# Make predictions on the test set
rf_test_predictions <- predict(rf_model, newdata = testing_data)

# Calculate test error (misclassification rate)
rf_test_error <- 1 - accuracy(rf_test_predictions, testing_data$mpg01)

# Print test error for Random Forest
cat("Random Forest Test Error:", rf_test_error, "\n")
```

```
## Random Forest Test Error: 0.1346154
```

```r
# Random Forest
model_rf <- randomForest(autotrain[,2:7], autotrain[,1], ntree=300, boos=T)
predrf <- predict(model_rf, autotrain[,2:7])
trainrf <- mean(predrf != autotrain$mpg01)# training error
testrf<-mean(predict(model_rf,autotest[,2:7]) != autotest$mpg01) # testimg er
ror
pred <- predict(model_rf,autotest[,2:7],type = "prob")
auc_rf = auc(autotest$mpg01,pred[,"1"])

#training errors
TREALL1<-
  round(cbind(trainlda,trainqda,trainnb,trainlog,trainknn3,trainrf),4)
colnames(TREALL1) <- c("LDA", "QDA", "Naive Bayes",
                       "Logistic","KNN=7","RandomForest")
rownames(TREALL1) <- "Training error"
TREALL1

#testing errors
TEALL1<- round(cbind(testlda,testqda,testnb,testlog,testknn3,testrf),4)
colnames(TEALL1) <- c("LDA", "QDA", "Naive Bayes",
                      "Logistic","KNN=7","RandomForest")
rownames(TEALL1) <- "Testing error"
```