HW3

Sean Morris

2024-02-13

Contents


```
knitr::opts_chunk$set(echo = TRUE)
rm(list=ls())
setwd("~/Desktop/Statlearning/Homework/HW3")

library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Chapter 4 (Page 189) Q1 Q5 Q7 Q14 - Skip (f) - For (d) (e), (g), and (h): in addition to computing the overall test error, construct the confusion matrix - Add part (i): briefly comment on the efficacy of the 4 methods employed. Which did the best in terms of overall test error? Which did the best at reducing each type of classification error? Which do you think is best overall?

Question 4

Part A

Suppose that we have a set of observations, each with measurements on p = 1 feature, X. We assume that X is uniformly (evenly) distributed on [0, 1]. Associated with each observation is a response value. Suppose that we wish to predict a test observation's response using only observations that are within 10 % of the range of X closest to that test observation. For instance, in order to predict the response for a test observation with X = 0.6, we will use observations in the range [0.55,0.65]. On average, what fraction of the available observations will we use to make the prediction?

```
# Given that values below X = 0.05 and above X = 0.95 would fall outside of bounds:
# Need to Adjust for boundary effects

avg_frac = function(x) {
  if (x < 0.05) {
    return(x + 0.05)
  } else if (x > 0.95) {
    return(1.05 - x)
  } else {
    return(0.1)
  }
}

average = integrate(Vectorize(avg_frac), lower = 0, upper = 1)
ans.a = average$value

cat(ans.a*100, "%")
```

```
## 9.749969 %
```

Part B

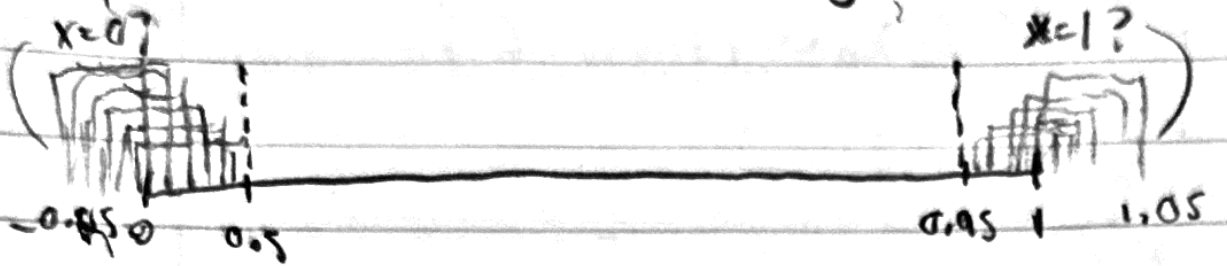
Now suppose that we have a set of observations, each with measurements on p = 2 features, X1 and X2. We assume that (X1, X2) are uniformly distributed on [0, 1] × [0, 1]. We wish to predict a test observation's response using only observations that are within 10 % of the range of X1 and within 10 % of the range of X2 closest to that test observation. For instance, in order to predict the response for a test observation with X1 = 0.6 and X2 = 0.35, we will use observations in the range [0.55, 0.65] for X1 and in the range [0.3,0.4] for X2. On average, what fraction of the available observations will we use to make the prediction?

```
# Given the uniform distribution and fixed range percentage selection:

ans.b = ans.a**2
cat(ans.b*100, "%")
```

```
## 0.950619 %
```

4) $X \sim \text{Unif}(0,1)$; 10% range



$$f_{\bar{X}}(x) \begin{cases} x < 0.5 \to 0.5 + x \\ 0.5 < x < .95 \to .10 \\ x > .95 \to 1.05 - x \end{cases} \xrightarrow{1\text{-grace}} 2\int_0^{0.5} 0.05 + x \, dx + .09$$

Seek

Part C

Now suppose that we have a set of observations on p = 100 features. Again the observations are uniformly distributed on each feature, and again each feature ranges in value from 0 to 1. We wish to predict a test observation's response using observations within the 10 % of each feature's range that is closest to that test observation. What fraction of the available observations will we use to make the prediction?

```
# Assuming uniform distributions for each feature
# Individual probabilities would remain 0.0975% on avg for each
# Thus,

ans.c = ans.a**100

cat(ans.c*100,"%")
```

```
## 7.949206e-100 %
```

Part D

Using your answers to parts (a)–(c), argue that a drawback of KNN when p is large is that there are very few training observations "near" any given test observation.

- As the number of features gets very large, the probability that any training observation falls within the threshold range of a given test observation increases exponentially.

Part E

Now suppose that we wish to make a prediction for a test observation by creating a p-dimensional hypercube centered around the test observation that contains, on average, 10 % of the train- ing observations. For p = 1,2, and 100, what is the length of each side of the hypercube? Comment on your answer.

- Assuming that our hyper cube contains on average 10% of the training observations, its volume therefore is 0.095 . Therefore, for any p-dimensional hyper cube, each length of the cube would be equal to 0.095^(1/p). For p = 1,2 and 100, the length of each side would be 0.095, 0.095^0.5, and 0.095^0.01 respectively. As we increase the number of parameters, the length of each side of the hyper cube will converge to the entire size span of our features.

Question 5

Part A

If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?

- If the Bayes decision boundary is linear, then one would expect LDA to perform better on the training and test sets as its underlying function is more well-defined to capture the true underlying decision boundary. QDA may perform better by finding more complex decision boundaries, however it also runs the risk of over fitting the data.

Part B

If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?

- If the Bayes decision boundary is non-linear, then one would expect QDA to outperform LDA on both the test and training set. QDA's underlying discriminant function allows it to better detect higher order decision boundaries that would otherwise be ignored by LDA.

Part C

In general, as the sample size n increases, do we expect the test prediction accuracy of QDA relative to LDA to improve, decline, or be unchanged? Why?

- As the sample size n increases, one would expect the test prediction of QDA relative to LDA to increase, as a larger sample size provides QDA with more data to estimate its more nuanced decision boundaries while simultaneously reducing its risk of over fitting (which is more common in smaller datasets).

Part E

True or False: Even if the Bayes decision boundary for a given problem is linear, we will probably achieve a superior test error rate using QDA rather than LDA because QDA is flexible enough to model a linear decision boundary. Justify your answer.

- This depends on the number of test observations fed into the QDA. If the sample size n is too low, then we run the risk of QDA over fitting the data - leading to untrustworthy classifications.

Question 7

Suppose that we wish to predict whether a given stock will issue a dividend this year ("Yes" or "No") based on X, last year's percent profit. We examine a large number of companies and discover that the mean value of X for companies that issued a dividend was X_1 = 10, while the mean for those that didn't was X_1 = 0. In addition, the variance of X for these two sets of companies was sig^2 = 36. Finally, 80 % of companies issued dividends. Assuming that X follows a normal distribution, predict the probability that a company will issue a dividend this year given that its percentage profit was X = 4 last year.

```
# Where 1 denotes dividend offered and 0 denotes dividend not offered
mu_1 = 10
mu_0 = 0
var_x = 36
p_1 = 0.8
p_0 = 0.2

# Expanding of law of total probability from notebook:
denom = (dnorm(4,mu_1,sqrt(var_x)) * p_1) +
        (dnorm(4,mu_0,sqrt(var_x)) * p_0)

numerator = (dnorm(4,mu_1,sqrt(var_x)) * p_1)

ans.7 = (numerator/denom)*100
cat(ans.7, "%")
```

```
## 75.18525 %
```

7)

Given:
- $\bar{X}_1 = \mu_1 = 10$; $\bar{X}_1 \sim N(10, 6)$
- $\bar{X}_0 = \mu_0 = 0$; $\bar{X}_0 \sim N(0, 6)$
- $\hat{\sigma}^2 = 36$
- $\hat{\sigma} = 6$
- $\pi_1 = 0.8$
- $\pi_0 = 0.2$

Find: $p(\pi_1 | X=4) = \dfrac{p(X=4|\pi_1) \cdot \pi_1}{p(X=4)}$

$p(X=4)$: LOTP (*)

(*) $p(X=4) = p(X=4|\pi_1) \cdot \pi_1 + p(X=4|\pi_0) \cdot \pi_2$

$p(X=4|\pi_1) \sim N(\mu_1, \hat{\sigma})$; $p(X=4|\pi_0) \sim N(\mu_0, \hat{\sigma})$

see R.

Question 14

Part A

```
rm(list = ls())

auto = read.csv("Auto.csv", header=TRUE)

auto$mpg01 = ifelse(auto$mpg >= median(auto$mpg), 1, 0)
#auto %>% relocate(mpg01)

auto$horsepower = as.numeric(auto$horsepower)
```

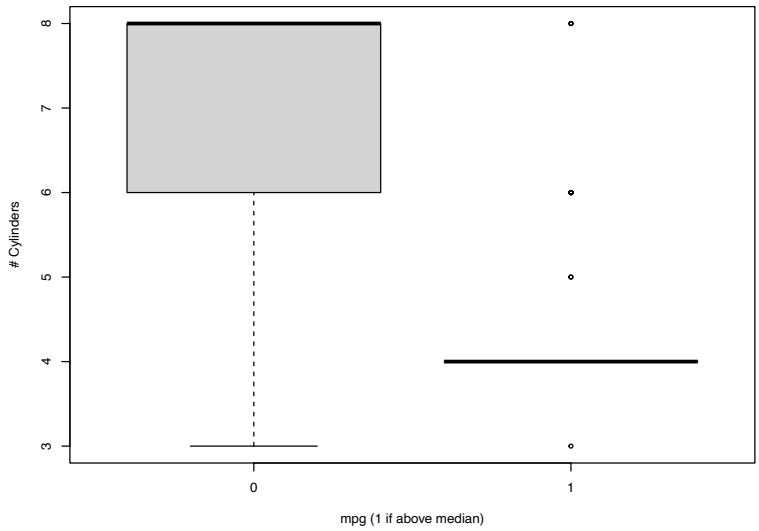
```
## Warning: NAs introduced by coercion
```

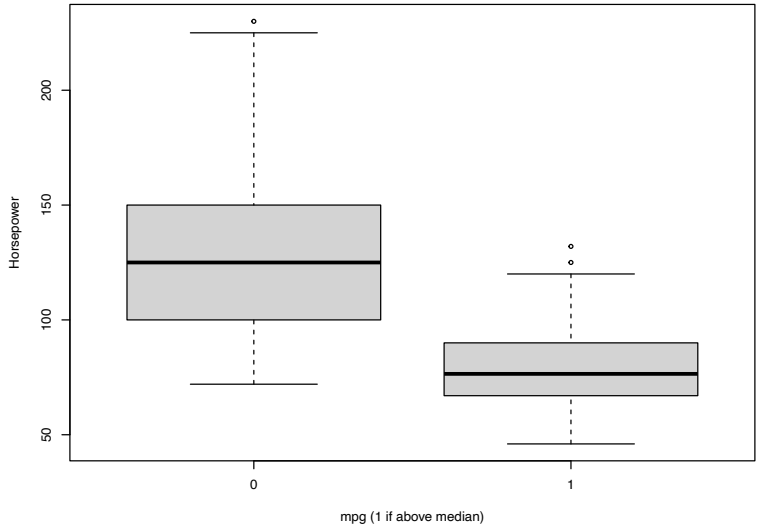
```
auto = na.omit(auto)
```

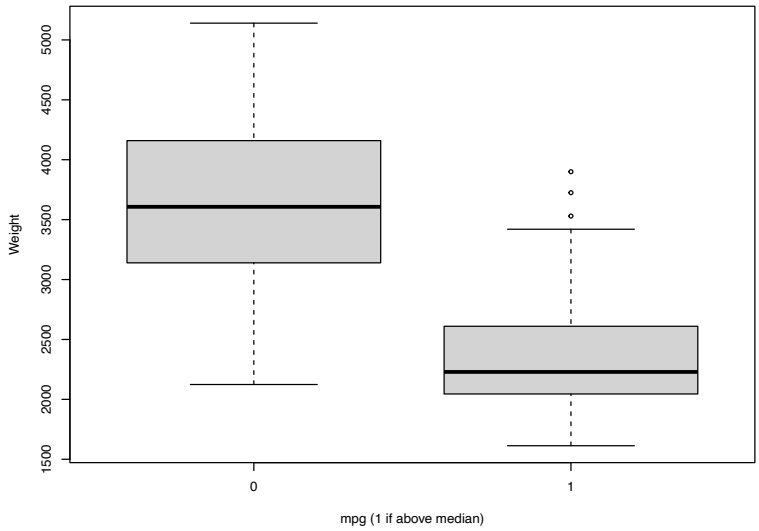
Part B

```
par(mfrow = c(3,1))

boxplot(auto$cylinders ~ auto$mpg01,
        xlab = "mpg (1 if above median)",
        ylab = "# Cylinders")
boxplot(as.numeric(auto$horsepower) ~ auto$mpg01,
        xlab = "mpg (1 if above median)",
        ylab = "Horsepower")
boxplot(auto$weight ~ auto$mpg01,
        xlab = "mpg (1 if above median)",
        ylab = "Weight")
```


Cylinders
3
4
5
6
7
8
0
1
mpg (1 if above median)

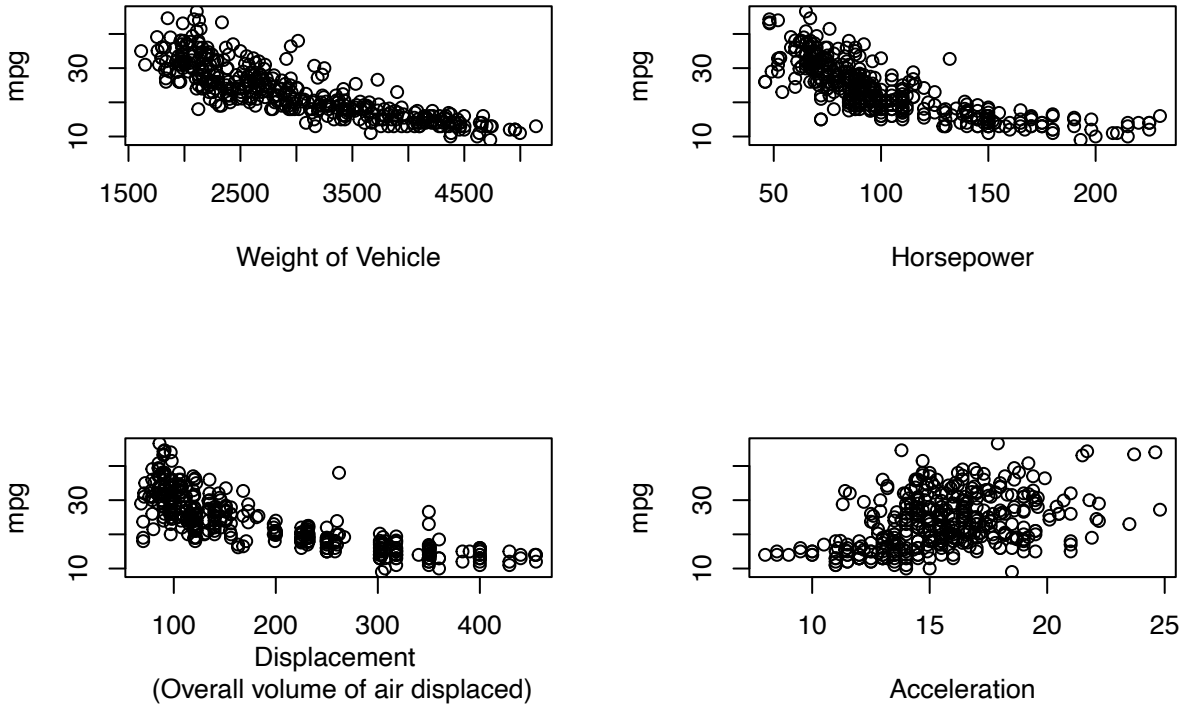

Horsepower
50
100
150
200
0
1
mpg (1 if above median)


Weight
1500
2000
2500
3000
3500
4000
4500
5000
0
1
mpg (1 if above median)

- Higher # cylinders associated with lower fuel efficiency
- Higher horsepower associated with lower fuel efficiency
- Heavier vehicles tend to have lower fuel efficiency

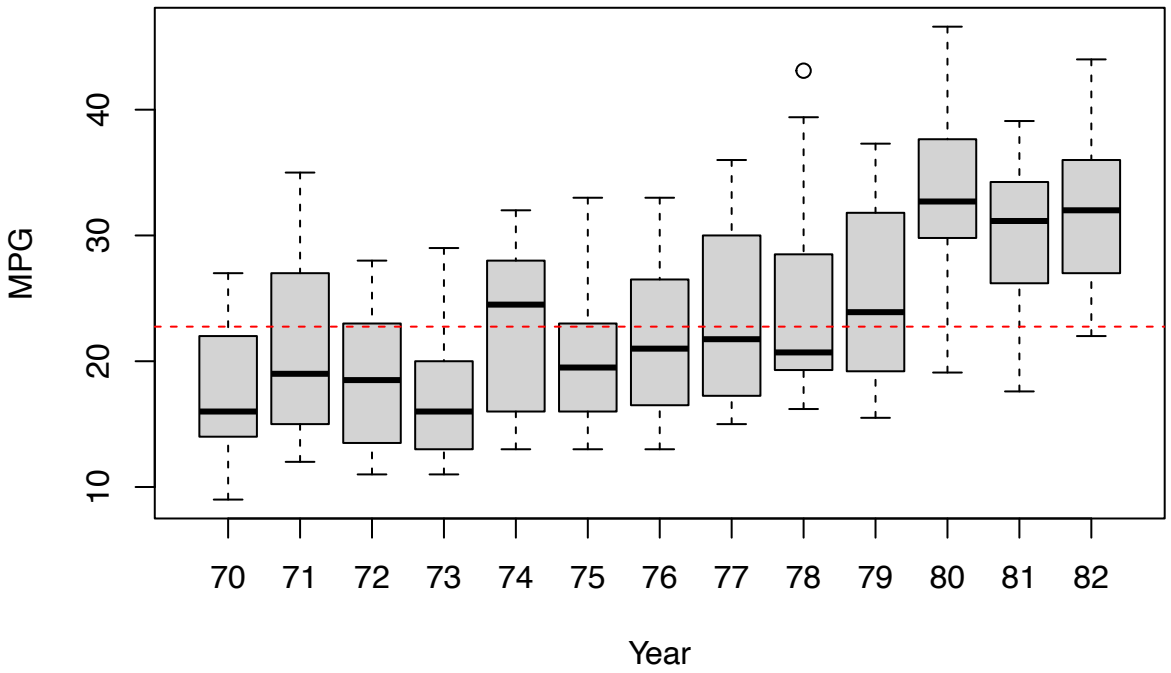
```
par(mfrow = c(2,2))

plot(auto$weight,auto$mpg,
     xlab = "Weight of Vehicle",
     ylab = "mpg")
plot(auto$horsepower,auto$mpg,
     xlab = "Horsepower",
     ylab = "mpg")
plot(auto$displacement,auto$mpg,
     xlab = "Displacement \n (Overall volume of air displaced)",
     ylab = "mpg")
plot(auto$acceleration,auto$mpg,
     xlab = "Acceleration",
     ylab = "mpg")
```



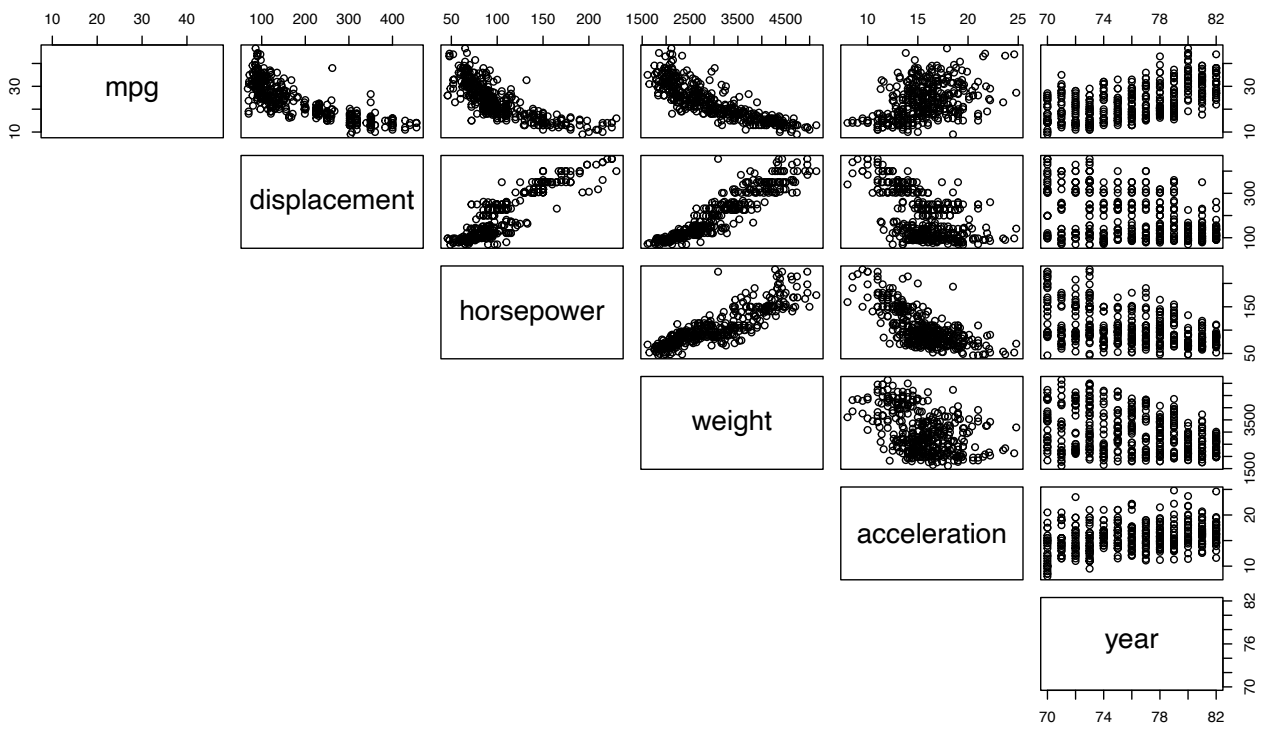
- Negative, slightly non-linear relationship between a vehicle's weight and its mpg. Initially negative, convex relationship between horsepower and mpg - efficiency appears to increase as horsepower exceeds 200. Negative and non-linear relationship between a vehicle's displacement and its mpg. Lastly, there does not appear any correlation between a vehicle's acceleration & its mpg.

```
boxplot(auto$mpg ~ auto$year,
        xlab = "Year",
        ylab = "MPG")
abline(h = median(auto$mpg), col="Red", lty=2)
```



- Fuel efficiency displays a gradual upwards trend until spike in 1980's. Consider adding another binary variable for cars produced before and after 1980.

```
pairs(auto[, c(1,3:7)], lower.panel = NULL)
```



Part C

```
library(MASS)
```

```
##
```

```
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
nrow(auto) # Checking number of rows
```

```
## [1] 392
```

```
sample = floor(0.75 * nrow(auto))

# For reproducability
set.seed(1789)
training_index = sort(sample(1:nrow(auto), size = sample))

# Splitting Data
auto_train = auto[training_index, ]
auto_test = auto[-training_index, ]

# Initializing model errors dataframe
model_errors = data.frame(model = character(), test_error = numeric(), stringsAsFactors = FALSE)
```

Part D

```
# Running LDA on training data
model_d = lda(mpg01 ~ displacement + horsepower + acceleration, data = auto_train)

# Running on test data
model_d.pred = predict(model_d, auto_test)

# Test Error
test_error = mean(model_d.pred$class != auto[-training_index, "mpg01"])
print(test_error)
```

```
## [1] 0.1122449
```

```
# Confusion matrix
confusion_matrix = table(Predicted = model_d.pred$class, Actual = auto_test$mpg01)
print(confusion_matrix)
```

```
##           Actual
## Predicted  0  1
##         0 40  1
##         1 10 47
```

```
# Appending to model_errors df
model_errors = rbind(model_errors, data.frame(model = "LDA", test_error = test_error))
```

Part E

```
rm(list = c("confusion_matrix", "test_error"))

# Running QDA on trainind data
model_e = qda(mpg01 ~ displacement + horsepower + acceleration, data = auto_train)
```

```r
# Running on test data
model_e.pred = predict(model_e, auto_test)

# Test Error
test_error = mean(model_e.pred$class != auto[-training_index, "mpg01"])
print(test_error)
```

```
## [1] 0.08163265
```

```r
# Confusion matrix
confusion_matrix = table(Predicted = model_e.pred$class, Actual = auto_test$mpg01)
print(confusion_matrix)
```

```
##           Actual
## Predicted  0  1
##         0 43  1
##         1  7 47
```

```r
# Appending to model_errors df
model_errors = rbind(model_errors, data.frame(model = "QDA", test_error = test_error))
```

Part F

```r
rm(list = c("confusion_matrix", "test_error"))

# Running logistic regression
model_f = glm(mpg01 ~ displacement + horsepower + acceleration, data = auto_train, family = binomial)

# Running on test data
model_f.probs = predict(model_f, auto_test, type = "response")

model_f.pred = rep(0, 98)
model_f.pred[model_f.probs > 0.5] = 1

# Test Error
test_error = mean(model_f.pred != auto[-training_index, "mpg01"])
print(test_error)
```

```
## [1] 0.1020408
```

```r
# Confusion matrix
confusion_matrix = table(Predicted = model_f.pred, Actual = auto_test$mpg01)
print(confusion_matrix)
```

```
##           Actual
## Predicted  0  1
##         0 42  2
##         1  8 46
```

```r
# Appending to model_errors df
model_errors = rbind(model_errors, data.frame(model = "Logistic", test_error = test_error))
```

Part G

```r
library(e1071)
rm(list = c("confusion_matrix", "test_error"))
```

```r
# Running Naive Bayes
model_g = naiveBayes(mpg01 ~ displacement + horsepower + acceleration, data = auto_train)

# Runnning on test data
model_g.pred = predict(model_g, auto_test)

# Test Error
test_error = mean(model_g.pred != auto[-training_index, "mpg01"])
print(test_error)
```

```
## [1] 0.122449
```

```r
# Confusion matrix
confusion_matrix = table(Predicted = model_g.pred, Actual = auto_test$mpg01)
print(confusion_matrix)
```

```
##          Actual
## Predicted  0  1
##         0 40  2
##         1 10 46
```

```r
# Appending to model_errors df
model_errors = rbind(model_errors, data.frame(model = "N. Bayes", test_error = test_error))
```

Part H

```r
library(class)
rm(list = c("confusion_matrix", "test_error"))

# Defining train and test matrices
train.X = cbind(auto$displacement, auto$horsepower, auto$acceleration)[training_index, ]
test.X = cbind(auto$displacement, auto$horsepower, auto$acceleration)[-training_index, ]
train.Y = cbind(auto$mpg01)[training_index,]

# Setting seed for reproducibility
set.seed(1789)

# Running on training data
model_h = knn(train.X, test.X, train.Y, k=3)

# Test Error
test_error = mean(model_h != auto[-training_index, "mpg01"])
print(test_error)
```

```
## [1] 0.09183673
```

```r
# Confusion matrix
confusion_matrix = table(Predicted = model_h, Actual = auto_test$mpg01)
print(confusion_matrix)
```

```
##          Actual
## Predicted  0  1
##         0 43  2
##         1  7 46
```

```r
# Appending to model_errors df
model_errors = rbind(model_errors, data.frame(model = "KNN", test_error = test_error))
```

Part I

```r
print(model_errors)
```

```
##      model test_error
## 1      LDA 0.11224490
## 2      QDA 0.08163265
## 3 Logistic 0.10204082
## 4 N. Bayes 0.12244898
## 5      KNN 0.09183673
```

- While QDA generated the lowest test error, KNN was also able to generate a low test error while reducing error types equally well. KNN only miss-classified one more above-median observation as a below-median observation than QDA did. Aside from that single miss classification, they performed equally well. I noticed also that all of the models miss-classified sub-median mpg automobiles for above-median mpg cars more frequently than the opposite case. In order to decide which model is actually the best, I would likely need to apply another method of specification like k-fold CV in order to gather more data from different model configurations, rather than solely operating off the train-test split that the questions used. However, going of of the above alone, I would argue the QDA is the best model for this task, as a few of the relationships between predictors and response variable's continuous counterpart are slightly non-linear - which may give QDA a leg up.
- While it was not asked of us to complete, I followed along with the textbook to implement the logistic regression model, and it appears to perform better than its LDA and naive bayes counterparts. I firmly believe that if I were to add a binary feature for years before and after 1980 to the glm model, then it would likely be the highest performing with all other modelling conditions held constant.