Q2.

(a)

```
library(ISLR)
data("Auto")
mpg01 <- ifelse(Auto$mpg > 25, 1, 0)
Auto <- data.frame(Auto, mpg01)
set.seed(123)
num_train <- nrow(Auto) * 0.8
inTrain <- sample(nrow(Auto), size = num_train)
training <- Auto[inTrain,]
testing <- Auto[-inTrain,]
logistic_model <- glm(mpg01 ~ displacement + horsepower + weight + cylinders, data = training)
summary(logistic_model)
```

```
##
## Call:
## glm(formula = mpg01 ~ displacement + horsepower + weight + cylinders,
##     data = training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.7775  -0.3107   0.1283   0.2552   0.8809
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.748e+00  1.383e-01  12.644  < 2e-16 ***
## displacement  9.839e-04  8.153e-04   1.207   0.2284
## horsepower   -1.213e-03  1.208e-03  -1.003   0.3164
## weight       -3.276e-04  6.586e-05  -4.974 1.09e-06 ***
## cylinders    -7.823e-02  3.710e-02  -2.108   0.0358 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1243352)
##
##     Null deviance: 75.834  on 312  degrees of freedom
## Residual deviance: 38.295  on 308  degrees of freedom
## AIC: 242.68
##
## Number of Fisher Scoring iterations: 2
```

Hence, only "weight" and "cylinders" are significant in the logistic regression. **(b)** For testing error

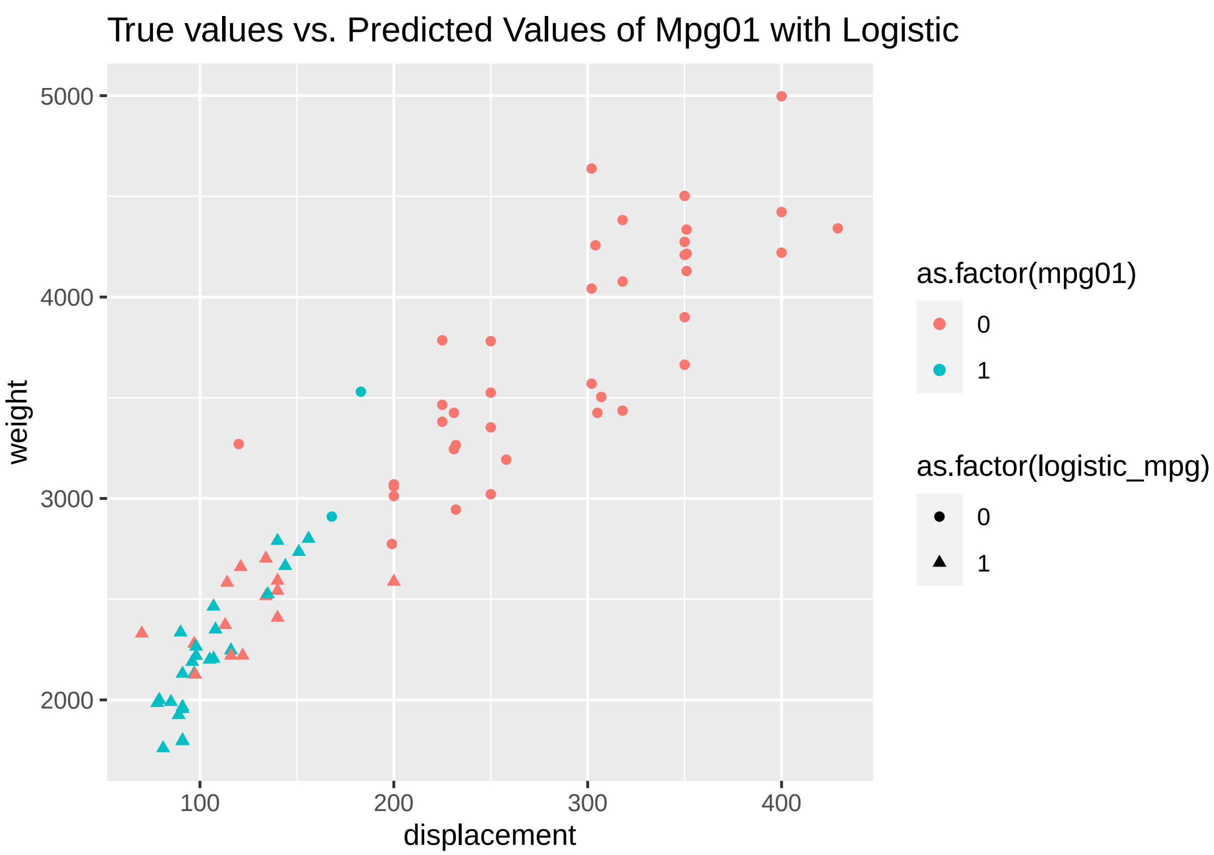
```
pred_test <- predict(logistic_model, testing)
testPrediction = rep("0", nrow(testing))
testPrediction[pred_test > .5] = "1"
testing$logistic_mpg = testPrediction
table(testPrediction, testing$mpg01, dnn = c("Predicted", "Actual"))
```

```
##          Actual
## Predicted  0  1
##         0 38  2
##         1 14 25
```

```r
round(mean(testPrediction != testing$mpg01), 2)
```

```
## [1] 0.2
```

```r
##Hence, the testing error is 0.2
ggplot(testing, aes(x=displacement, y=weight, color = as.factor(mpg01),shape = as.factor(logistic_mpg))
```



For training error

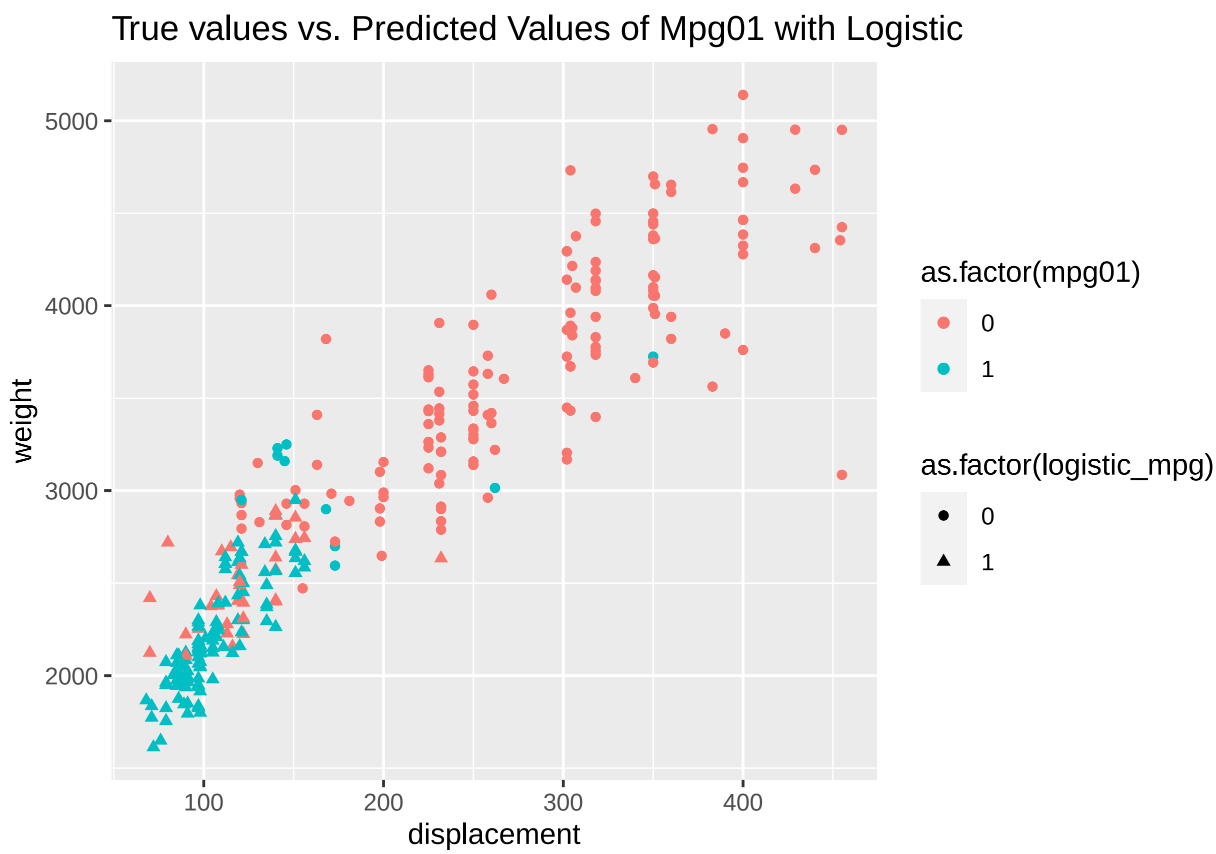
```r
pred_train <- predict(logistic_model, training)
trainPrediction = rep("0", nrow(training))
trainPrediction[pred_train > .5] = "1"
training$logistic_mpg = trainPrediction
table(trainPrediction, training$mpg01, dnn = c("Predicted", "Actual"))
```

```
##          Actual
## Predicted   0   1
##         0 149  10
##         1  35 119
```

```r
round(mean(trainPrediction != training$mpg01), 2)
```

```
## [1] 0.14
```

```r
##Hence, the training error is 0.14
ggplot(training, aes(x=displacement, y=weight, color = as.factor(mpg01),shape = as.factor(logistic_mpg)
```



(c)

$$\log \frac{p}{1-p} = 1.748 + 9.839\times 10^{-4} displacement - 1.213\times 10^{-3} horsepower - 3.276\times 10^{-4} weight - 7.823\times 10^{-2} cylinders \quad (1)$$

```
median(training$displacement)
```

```
## [1] 146
```

```
median(training$horsepower)
```

```
## [1] 95
```

```
median(training$weight)
```

```
## [1] 2807
```

```
median(training$cylinders)
```

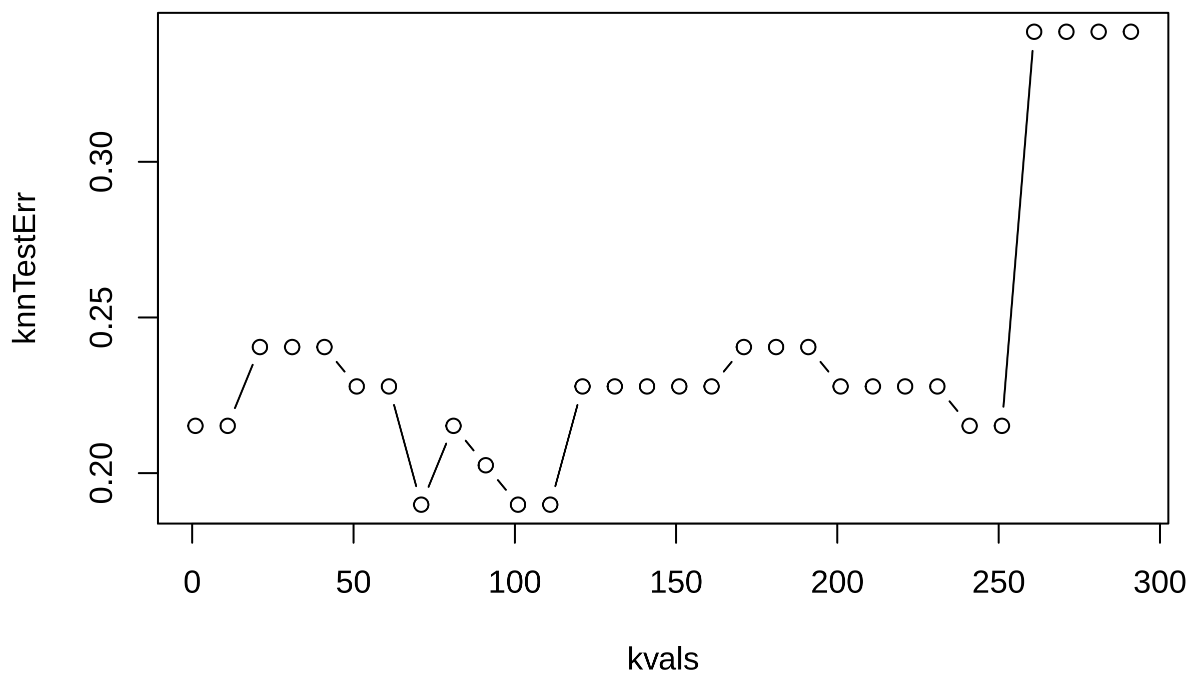
```
## [1] 4
```

Hence $odds = 0.408$, $p = 0.6$ **(d)**

```
trainX = as.matrix(training[c("displacement", "horsepower","weight","cylinders")])
testX = as.matrix(testing[c("displacement", "horsepower","weight","cylinders")])
set.seed(1)
kvals = seq(1, 300, 10)
```

testing error

```r
knnTestErr = vector(length = length(kvals))
for (i in 1:length(kvals)) {
knn.pred = knn(train = trainX, test = testX, cl = training$mpg01, k=kvals[i])
knnTestErr[i] = mean(knn.pred != testing$mpg01)
}
plot(knnTestErr ~ kvals, type = "b")
```

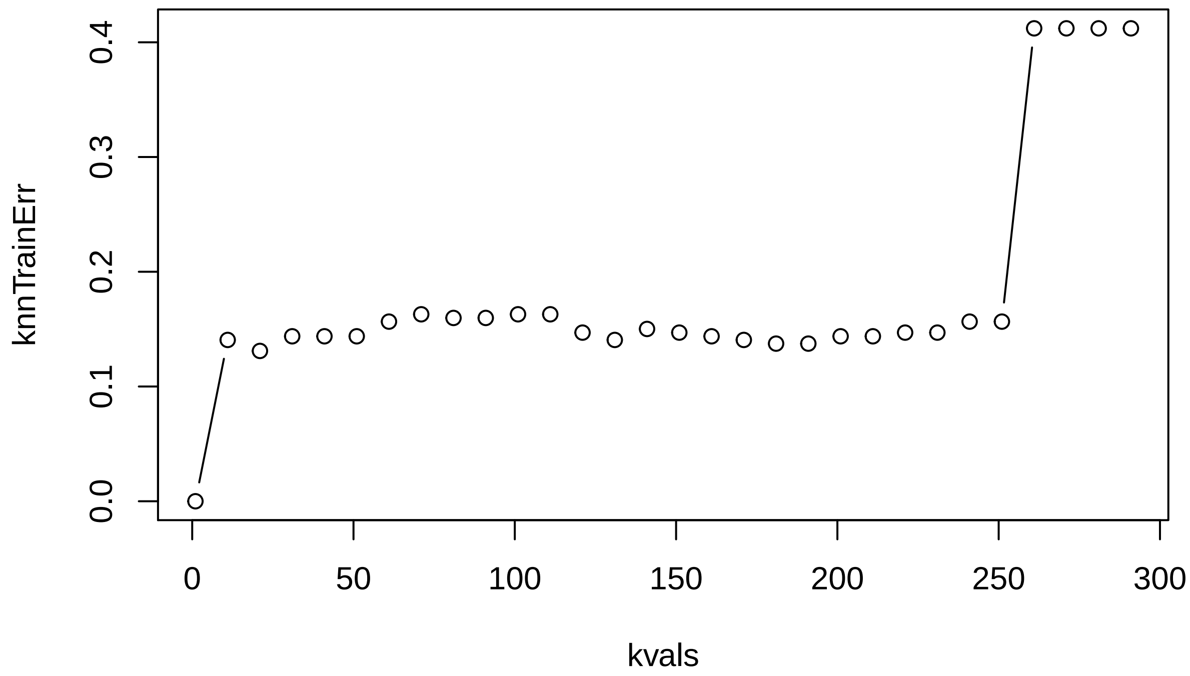



```r
knnTestErr ##The minimum value is obtained when k = 71
```

```
##  [1] 0.2151899 0.2151899 0.2405063 0.2405063 0.2405063 0.2278481 0.2278481
##  [8] 0.1898734 0.2151899 0.2025316 0.1898734 0.1898734 0.2278481 0.2278481
## [15] 0.2278481 0.2278481 0.2278481 0.2405063 0.2405063 0.2405063 0.2278481
## [22] 0.2278481 0.2278481 0.2278481 0.2151899 0.2151899 0.3417722 0.3417722
## [29] 0.3417722 0.3417722
```

training error

```r
kvals = seq(1, 300, 10)
knnTrainErr = vector(length = length(kvals))
for (i in 1:length(kvals)) {
knn.pred1 = knn(train = trainX, test = trainX, cl = training$mpg01, k=kvals[i])
knnTrainErr[i] = mean(knn.pred1 != training$mpg01)
}
plot(knnTrainErr ~ kvals, type = "b")
```



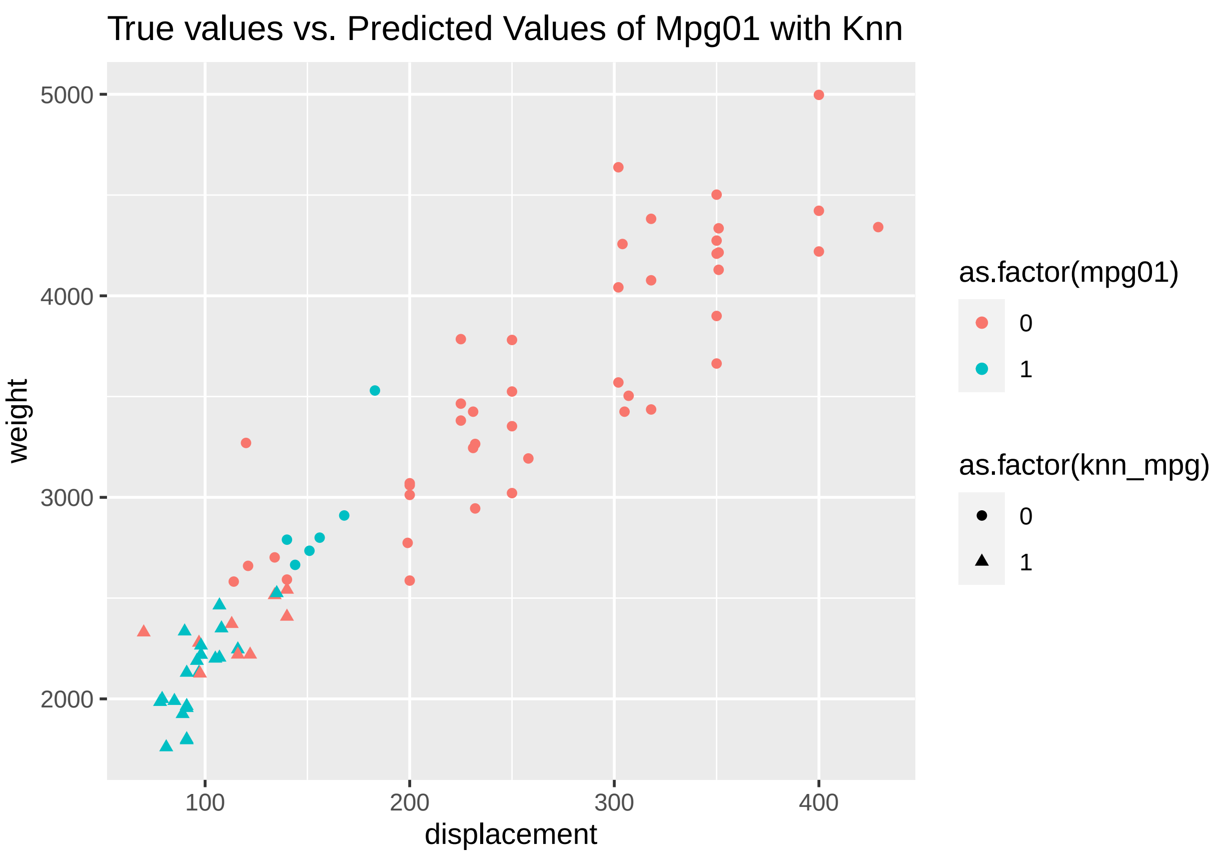

```
knnTrainErr  ##The minimum value is obtained when k = 1
```

```
##  [1] 0.0000000 0.1405751 0.1309904 0.1437700 0.1437700 0.1437700 0.1565495
##  [8] 0.1629393 0.1597444 0.1597444 0.1629393 0.1629393 0.1469649 0.1405751
## [15] 0.1501597 0.1469649 0.1437700 0.1405751 0.1373802 0.1373802 0.1437700
## [22] 0.1437700 0.1469649 0.1469649 0.1565495 0.1565495 0.4121406 0.4121406
## [29] 0.4121406 0.4121406
```

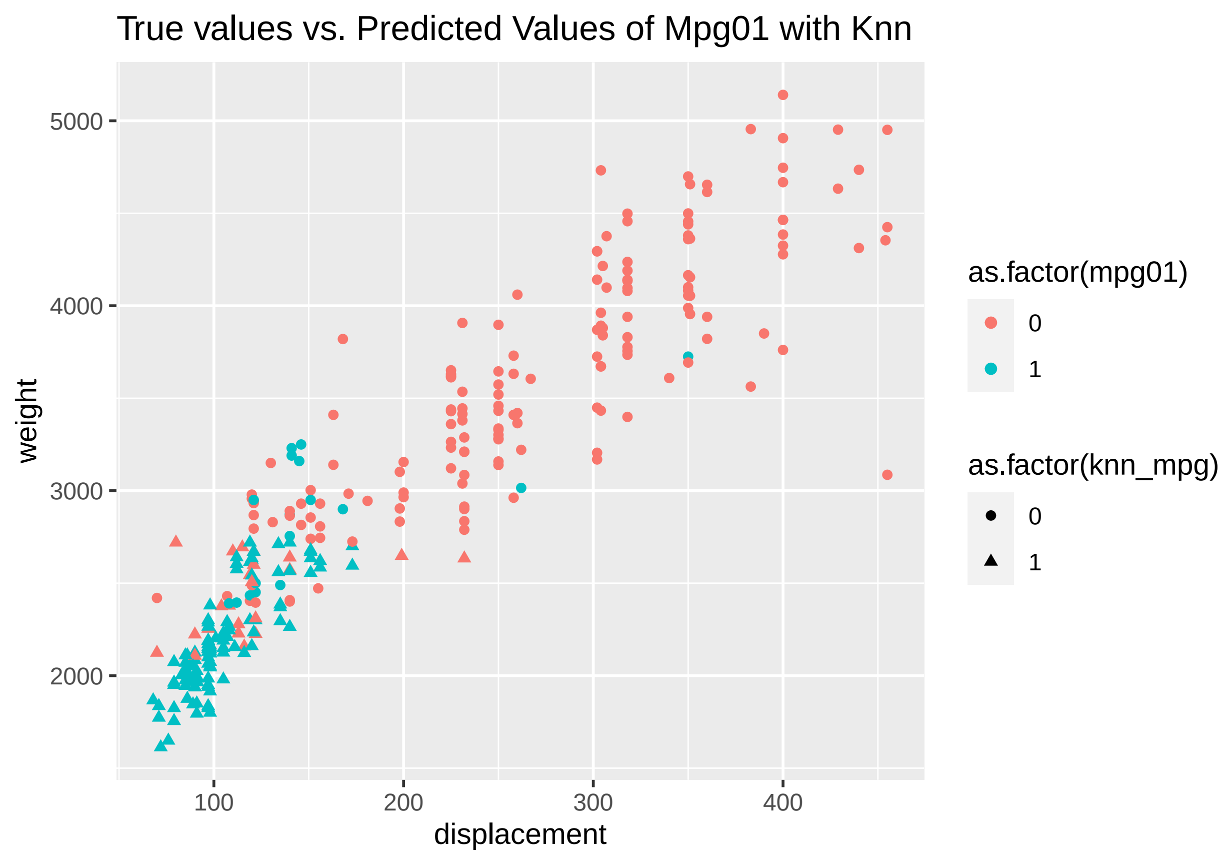
```
##except for k = 1m the minimum value is obtained when k = 21
```

(e)

```
knn.pred = knn(train = trainX, test = testX, cl = training$mpg01, k=71)
testing$knn_mpg = knn.pred
ggplot(testing, aes(x=displacement, y=weight, color = as.factor(mpg01),shape = as.factor(knn_mpg)))+geom
```




```
knn.pred1 = knn(train = trainX, test = trainX, cl = training$mpg01, k=21)
training$knn_mpg = knn.pred1
ggplot(training, aes(x=displacement, y=weight, color = as.factor(mpg01),shape = as.factor(knn_mpg)))+ge
```



(f)

```r
table(knn.pred, testing$mpg01, dnn = c("Predicted", "Actual"))
```

```
##          Actual
## Predicted  0  1
##         0 43  6
##         1  9 21
```

```r
round(mean(knn.pred != testing$mpg01), 2)
```

```
## [1] 0.19
```

```r
##Hence, the testing error is 0.19

table(knn.pred1, training$mpg01, dnn = c("Predicted", "Actual"))
```

```
##          Actual
## Predicted   0   1
##         0 160  17
##         1  24 112
```

```r
round(mean(knn.pred1 != training$mpg01), 2)
```

```
## [1] 0.13
```

```r
##Hence, the training error is 0.13
```

In this experiment, we choose k-value with a non-efficient for loop, which is similar to bubbling selection, the efficiency of this algorithm is $O(n)$, which is super slow and might cause extremely long estimation time.

However, if we have determined the range of k-value, we can use a more efficient algorithm to cut off the time into $Olog(n)$.

Also, normalization is also very important in Knn classification, if we ignored the normalization part it might jeopardize the estimation.

(g) Regarding the test error, QDA performs the best, and about the training error both QDA and Knn classification performs the best. Hence the the distribution of the data is non-linear, and the boundary between classes is quadratic.