Tom Meagher
DS 502 Homework 2
October 4, 2016

**1.** Section 4.7, page 168, question 1

**2.** Section 4.7, page 169-170, question 5

**a.**

If the Bayes decision boundary is linear, we expect LDA to perform better on the testing set because it is much less flexible and performs well in situations where having lower variance is important, like if Bayes decision boundary is linear. In contrast, QDA performs poorly well on the training set because it models the random error of the data better, but this will make it perform worse on the test data.

**b.**

If the Bayes decision boundary is non-linear, we expect QDA to perform better on the training and test sets because it has higher variance, which would approximate non-linearities in the data better. In contrast, LDA, as it is a linear method, would result in high bias across the data.

**c.**

As the sample size increases, we expect the test prediction accuracy of both QDA and LDA to increase. It depends which one will perform better—if the data is non-linear, then QDA should perform better. If the data is linear, then LDA will perform better. Also, if the sample size doesn't increase too much (stays relatively small), then LDA could perform better than QDA because QDA needs more samples to perform well.

**d.**

True, 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.

While the statement is true, you have to worry about overfitting. QDA will not only be able to model the linear decision boundary, but it can also capture some of the residual error. This will help it perform well on the training data and possibly the test data since it is more flexible than LDA. If QDA models too much of the residual error in the training data (overfit), it might perform much worse on the test data.

**3.** Section 4.7, page 170, question 8
Logistic regression: training error = 20%, testing error= 30%
KNN (k=1): training error = ?, testing error = ?, average error = 18%

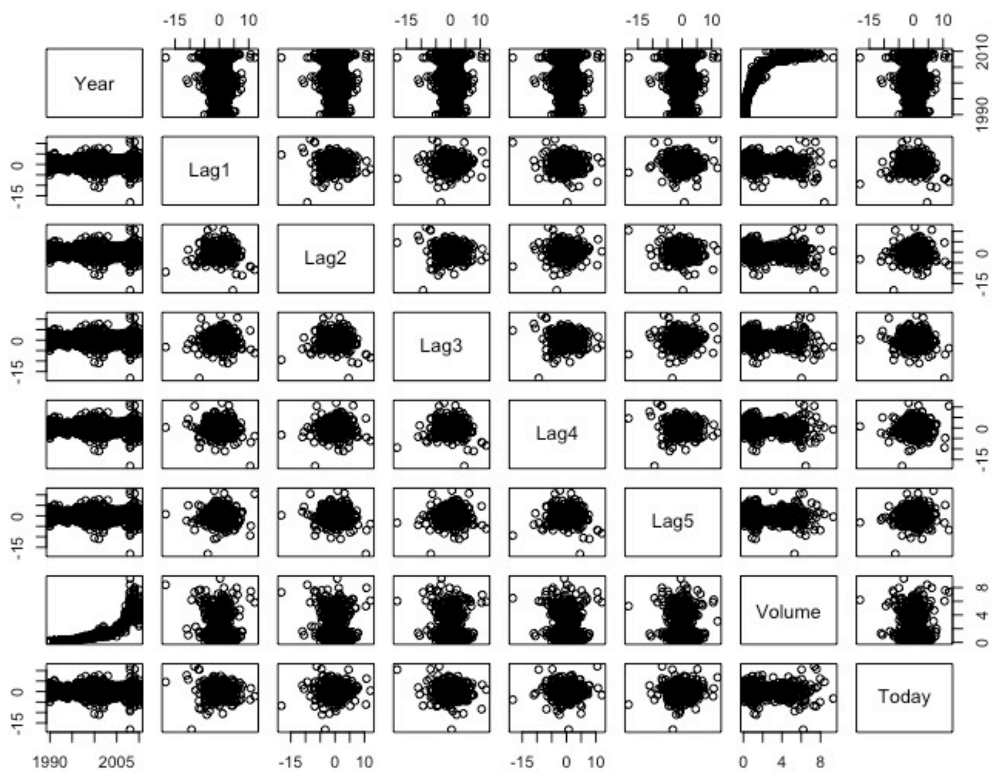
KNN with k=1 will classify all the data in the training set correctly (training error = 0%), but the test set will almost certainly have errors. With the knowledge that the training error for KNN is equal to zero, we can use a sample dataset to compare the testing error of the logistic regression and KNN. Assume there is a dataset with 1,000 samples so we will have 500 training and 500 testing samples. The overall error for KNN is 18%, which means there are 180 misclassified samples for KNN. Since the training error of KNN is equal to zero, then these 180 samples had to come from testing. We can calculate KNN's testing error to be 36% (180 misclassified samples / 500 testing samples). As a result, you can clearly see that logistic regression's testing error (30%) is better than KNN's (36%) so you would prefer logistic regression for classifying new observations in the problem.

**4.** Section 4.7, page171, question 10

**a.**

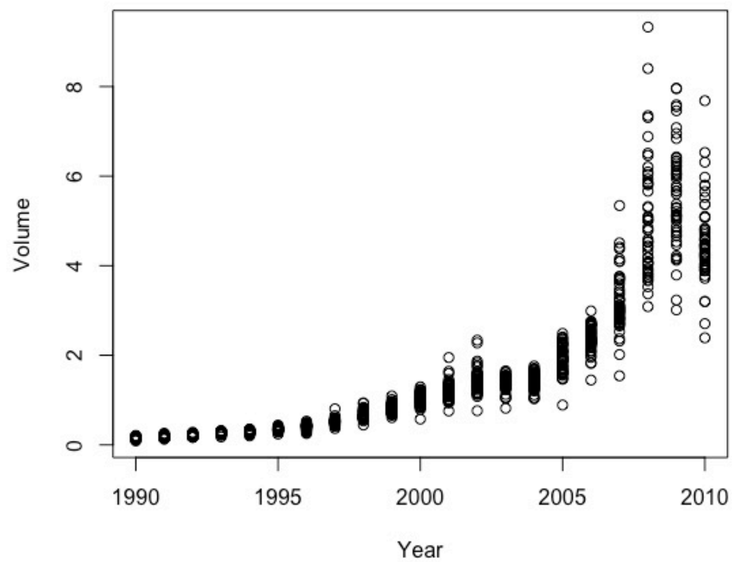
```
> summary(Weekly)
      Year           Lag1               Lag2               Lag3               Lag4         
 Min.   :1990   Min.   :-18.1950   Min.   :-18.1950   Min.   :-18.1950   Min.   :-18.1950  
 1st Qu.:1995   1st Qu.: -1.1540   1st Qu.: -1.1540   1st Qu.: -1.1580   1st Qu.: -1.1580  
 Median :2000   Median :  0.2410   Median :  0.2410   Median :  0.2410   Median :  0.2380  
 Mean   :2000   Mean   :  0.1506   Mean   :  0.1511   Mean   :  0.1472   Mean   :  0.1458  
 3rd Qu.:2005   3rd Qu.:  1.4050   3rd Qu.:  1.4090   3rd Qu.:  1.4090   3rd Qu.:  1.4090  
 Max.   :2010   Max.   : 12.0260   Max.   : 12.0260   Max.   : 12.0260   Max.   : 12.0260  
      Lag5              Volume            Today          Direction 
 Min.   :-18.1950   Min.   :0.08747   Min.   :-18.1950   Down:484  
 1st Qu.: -1.1660   1st Qu.:0.33202   1st Qu.: -1.1540   Up  :605  
 Median :  0.2340   Median :1.00268   Median :  0.2410             
 Mean   :  0.1399   Mean   :1.57462   Mean   :  0.1499             
 3rd Qu.:  1.4050   3rd Qu.:2.05373   3rd Qu.:  1.4050             
 Max.   : 12.0260   Max.   :9.32821   Max.   : 12.0260             
```

Based on an initial numerical summary of the data, there does not appear to be any pattern. All the lags (Lag1-Lag5) appear very consistent with each other. Further investigating, I looked at a scatterplot matrix of all the quantitative variables.




Unfortunately, it is very difficult to read—not only because there is a lot of data, but also because there aren't very main patterns to pick out. One pattern that looks promising though is the relationship between year and volume.

Exploring the relationship further, it appears that over the years, the trade volume has increased visually significantly.




Looking into the correlation matrix for the dataset, clearly confirms that there is a significant, positive relationship between year and volume, whereas all the other correlations are very small.

```
> cor(weekly_quantitative)
              Year         Lag1        Lag2        Lag3         Lag4         Lag5      Volume        Today
Year    1.00000000 -0.032289274 -0.03339001 -0.03000649 -0.031127923 -0.030519101  0.84194162 -0.032459894
Lag1   -0.03228927  1.000000000 -0.07485305  0.05863568 -0.071273876 -0.008183096 -0.06495131 -0.075031842
Lag2   -0.03339001 -0.074853051  1.00000000 -0.07572091  0.058381535 -0.072499482 -0.08551314  0.059166717
Lag3   -0.03000649  0.058635682 -0.07572091  1.00000000 -0.075395865  0.060657175 -0.06928771 -0.071243639
Lag4   -0.03112792 -0.071273876  0.05838153 -0.07539587  1.000000000 -0.075675027 -0.06107462 -0.007825873
Lag5   -0.03051910 -0.008183096 -0.07249948  0.06065717 -0.075675027  1.000000000 -0.05851741  0.011012698
Volume  0.84194162 -0.064951313 -0.08551314 -0.06928771 -0.061074617 -0.058517414  1.00000000 -0.033077783
Today  -0.03245989 -0.075031842  0.05916672 -0.07124364 -0.007825873  0.011012698 -0.03307778  1.000000000
```

**b.**
Yes, Lag1 appears to be statistically significant at the 0.05 level. The intercept of the model is also significant.

```
> glm.fit=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume, data=Weekly, family=binomial)
> summary(glm.fit)

Call:
glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
    Volume, family = binomial, data = Weekly)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6949  -1.2565   0.9913   1.0849   1.4579

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.26686    0.08593   3.106   0.0019 **
Lag1        -0.04127    0.02641  -1.563   0.1181
Lag2         0.05844    0.02686   2.175   0.0296 *
Lag3        -0.01606    0.02666  -0.602   0.5469
Lag4        -0.02779    0.02646  -1.050   0.2937
Lag5        -0.01447    0.02638  -0.549   0.5833
Volume      -0.02274    0.03690  -0.616   0.5377
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1496.2  on 1088  degrees of freedom
Residual deviance: 1486.4  on 1082  degrees of freedom
AIC: 1500.4

Number of Fisher Scoring iterations: 4
```

**c.**

```
glm.probs = predict(glm.fit, Weekly, type="response")
glm.pred = rep("Down", nrow(Weekly))
glm.pred[glm.probs > 0.50] = "Up"
table(glm.pred, Direction)
mean(glm.pred == Direction)
```

```
         Direction
glm.pred Down  Up
    Down   54  48
    Up    430 557
```

Based on the confusion matrix, the model does not fit the data very well. The model predicts correctly 54 downs and 557 ups in the market, but incorrectly predicts 430 downs and 48 ups. Overall, this means that the model predicts the market direction only 56.1% of the time on the training set. Training set predicts tend to be overly optimistic—when using a test set, performance will likely decrease.

**d.**

```
train = Year <= 2008
Weekly2 = Weekly[!train,]
glm.fit = glm(Direction~Lag2, family=binomial, data=Weekly, subset=train)
summary(glm.fit)
glm.probs = predict(glm.fit, Weekly2, type="response")
glm.pred = rep("Down",nrow(Weekly2))
glm.pred[glm.probs > 0.50] = "Up"
table(glm.pred, Weekly2$Direction)
mean(glm.pred == Weekly2$Direction)
```

```
glm.pred Down Up
    Down    9  5
    Up     34 56
```

Overall fraction of correct predictions = 0.625

This model appears to fit the data better, even though we used two completely separate training and testing datasets.

**e.**

```
library(MASS)
lda.fit = lda(Direction~Lag2, data=Weekly, subset=train)
lda.predict = predict(lda.fit, Weekly2)
lda.class = lda.predict$class
table(lda.class, Weekly2$Direction)
mean(lda.class == Weekly2$Direction)
```

```
lda.class Down Up
     Down    9  5
     Up     34 56
```

Overall fraction of correct predictions = 0.625

LDA performs identical to the previous logistic regression

**f.**

```
qda.fit = qda(Direction~Lag2, data=Weekly, subset=train)
qda.fit
qda.predict = predict(qda.fit, Weekly2)
qda.class = qda.predict$class
table(qda.class, Weekly2$Direction)
mean(qda.class == Weekly2$Direction)
```

```
qda.class Down Up
     Down    0  0
     Up     43 61
```

Overall fraction of correct predictions = 0.586

QDA performs poorly on the test data. It predicts all the down directions incorrectly, but it predicts all the up directions correctly. The model could be overfitting the up direction and is unable to approximate anything else.

**g.**

```
library(class)
train.X = Weekly[train, "Lag2", drop=F]
test.X = Weekly[!train, "Lag2", drop=F]
train.Direction = Weekly[train, "Direction", drop=T]
test.Direction = Weekly[!train, "Direction", drop=T]
set.seed(1)
knn.pred = knn(train.X, test.X, train.Direction, k=1)
table(knn.pred, test.Direction)
mean(knn.pred == test.Direction)
```

```
        test.Direction
knn.pred Down Up
    Down   21 30
    Up     22 31
```

Overall fraction of correct predictions = 0.5

KNN does not appear very effective at modeling the data at all.

**h.**

The Logistic Regression (d) and Linear Discriminate Analysis (e) appear to perform the best on the data. Quadratic Discriminant Analysis (f) appears to overfit the up direction in the data, while K-Nearest Neighbors looks completely random, almost like guessing could have performed better.

**i.**

Logistic regression with predictors: Lag2, Volume, and Lag2*Volume

```
glm.fit = glm(Direction~Lag2*Volume, family=binomial, data=Weekly, subset=train)
summary(glm.fit)
glm.probs = predict(glm.fit, Weekly2, type="response")
glm.pred = rep("Down", nrow(Weekly2))
glm.pred[glm.probs > 0.50] = "Up"
table(glm.pred, Weekly2$Direction)
mean(glm.pred == Weekly2$Direction)
```

```
glm.pred Down Up
    Down   20 25
    Up     23 36
```

Overall fraction of correct predictions = 0.538

Does not perform very well.

KNN with predictor: Lag2, and k=4

```
set.seed(1)
knn.pred = knn(train.X, test.X, train.Direction, k=4)
table(knn.pred, test.Direction)
mean(knn.pred == test.Direction)
```

```
         test.Direction
knn.pred Down Up
    Down   20 17
    Up     23 44
```

Overall fraction of correct predictions = 0.615

Performs much better than KNN with k=1, but does not perform as well as the original Logistic Regression or Linear Discriminant Analysis.

KNN with predictor: Lag2, Lag2^2, and k=4

```
train.X = cbind(Lag2, I(Lag2 ^ 2))[train,]
test.X = cbind(Lag2, I(Lag2 ^ 2))[!train,]
train.Direction = Direction [train]
set.seed(1)
knn.pred = knn(train.X, test.X, train.Direction, k=4)
table(knn.pred, test.Direction)
mean(knn.pred == test.Direction)
```

```
         test.Direction
knn.pred Down Up
    Down   20 17
    Up     23 44
```

Overall fraction of correct predictions = 0.615

Performs the same as the previous KNN model, even with a non-linear transformation on Lag2.

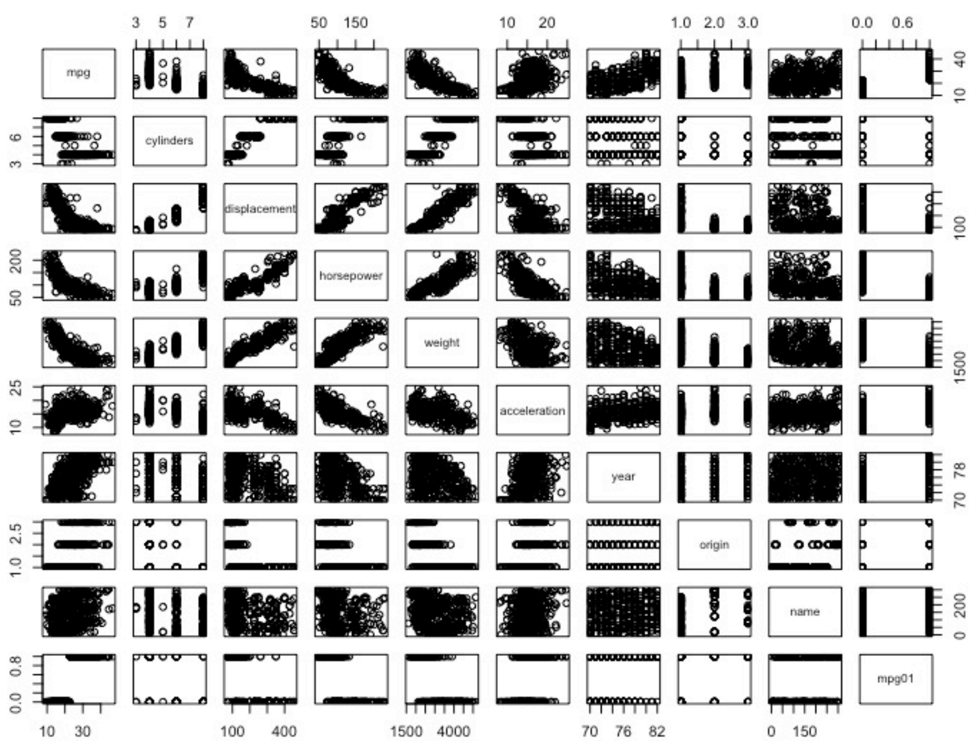
**5.** Section 4.7, page 171-172, question 11

**a.**

```
library(ISLR)
data(Auto)
Auto$mpg01 <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
```

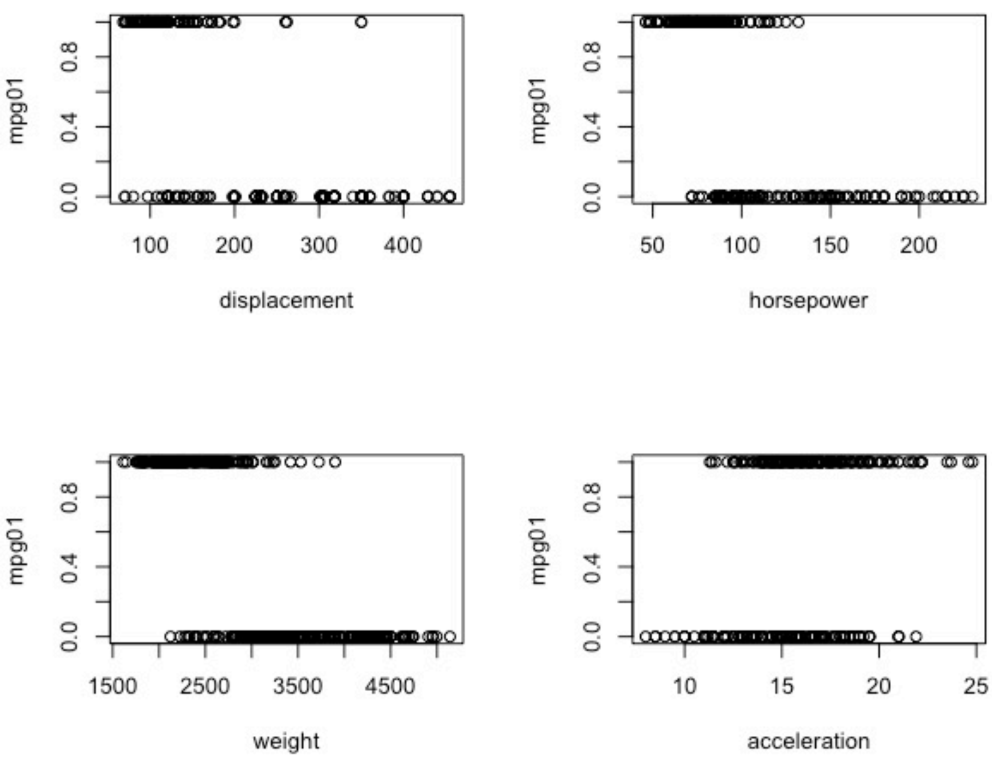
**b.**

```
pairs(Auto)
```



Displacement, horsepower, weight, and acceleration all see to be useful in predicting mpg01 because there is a pattern to the data that looks like it can be modeled through classification. Investigating further below, we can see the relationships better than in using a scatterplot matrix.

```
par(mfrow=c(2, 2))
plot(displacement, mpg01)
plot(horsepower, mpg01)
plot(weight, mpg01)
plot(acceleration, mpg01)
```




**c.**

```
set.seed(1)
rands <- rnorm(nrow(Auto))
test <- rands > quantile(rands, 0.75)
train <- !test
Auto.train <- Auto[train,]
Auto.test <- Auto[test,]
```

**d.**

```
library(MASS)
lda.fit = lda(mpg01~displacement+horsepower+weight+acceleration, data=Auto,
subset=train)
lda.fit
lda.predict = predict(lda.fit, Auto.test)
lda.class = lda.predict$class
table(lda.class, Auto.test$mpg01)
test_accuracy <- mean(lda.class == Auto.test$mpg01)
1 - test_accuracy
```

```
lda.class  0  1
        0 33  3
        1 11 51
```

The test accuracy of LDA is 0.8571429, while the test error is 0.1428571.

**e.**

```r
qda.fit = qda(mpg01~displacement+horsepower+weight+acceleration, data=Auto.train)
qda.fit
qda.predict = predict(qda.fit, Auto.test)
qda.class = qda.predict$class
table(qda.class, Auto.test$mpg01)
test_accuracy <- mean(qda.class == Auto.test$mpg01)
1 - test_accuracy
```

```
qda.class  0  1
        0 36  5
        1  8 49
```

The test accuracy of QDA is 0.8673469, while the test error is 0.1326531.

**f.**

```r
glm.fit = glm(mpg01~displacement+horsepower+weight+acceleration, family=binomial,
data=Auto.train)
summary(glm.fit)
glm.probs = predict(glm.fit, Auto.test, type="response")
glm.pred = rep(0, nrow(Auto.test))
glm.pred[glm.probs > 0.50] = 1
table(glm.pred, Auto.test$mpg01)
test_accuracy <- mean(glm.pred == Auto.test$mpg01)
1 - test_accuracy
```

```
glm.pred  0  1
       0 36  6
       1  8 48
```

The test accuracy of logistic regression is 0.8571429, while the test error is 0.1428571.

**g.**

```r
library(class)
set.seed(1)
train.knn = Auto.train[,c("displacement","horsepower","weight","acceleration")]
test.knn =  Auto.test[,c("displacement","horsepower","weight","acceleration")]
```

**k=1**

```r
knn.pred=knn(train.knn, test.knn, Auto.train$mpg01, k=1)
table(knn.pred, Auto.test$mpg01)
test_accuracy <- mean(knn.pred==Auto.test$mpg01)
1 - test_accuracy
```

```
knn.pred  0  1
       0 35  5
       1  9 49
```

The test accuracy of KNN (k=1) is 0.8571429, while the test error is 0.1428571.

**k=2**

```r
knn.pred=knn(train.knn, test.knn, Auto.train$mpg01, k=2)
table(knn.pred, Auto.test$mpg01)
test_accuracy <- mean(knn.pred==Auto.test$mpg01)
1 - test_accuracy
```

```
knn.pred  0  1
       0 34  5
       1 10 49
```

The test accuracy of KNN (k=2) is 0.8469388, while the test error is 0.1530612.

**k=3 performs the best on the data set**

```r
knn.pred=knn(train.knn, test.knn, Auto.train$mpg01, k=3)
table(knn.pred, Auto.test$mpg01)
test_accuracy <- mean(knn.pred==Auto.test$mpg01)
1 - test_accuracy
```

```
knn.pred  0  1
       0 36  4
       1  8 50
```

The test accuracy of KNN (k=3) is 0.877551, while the test error is 0.122449.

**k=4**

```r
knn.pred=knn(train.knn, test.knn, Auto.train$mpg01, k=4)
table(knn.pred, Auto.test$mpg01)
test_accuracy <- mean(knn.pred==Auto.test$mpg01)
1 - test_accuracy
```

```
knn.pred  0  1
       0 36  5
       1  8 49
```

The test accuracy of KNN (k=4) is 0.8673469, while the test error is 0.1326531.

**6.** Section 5.4, page 197, question 1

**7.** Section 5.4, page 197, question 2

**a.**

Bootstrap is sampling with replacement so even when we start with *n* samples, the sample size will be the same every time because the observation is returned to the sample *n*. The first bootstrap observation, *j*, is one of the *n* samples so there are *n* - 1 samples that are not *j*. Thus, we get the following:

**b.**

The second bootstrap observation has the same probably that it is not the *jth* observation from the original sample as **a** because we sampling with replacement!

**c.**

**d.**

```
prob_in_boot.fn = function (n) {
  return (1 - ((1 - 1/n)^n))
}
prob_in_boot.fn(5)
```

**0.67232**

**e.**

```
prob_in_boot.fn(100)
```

**0.6339677**

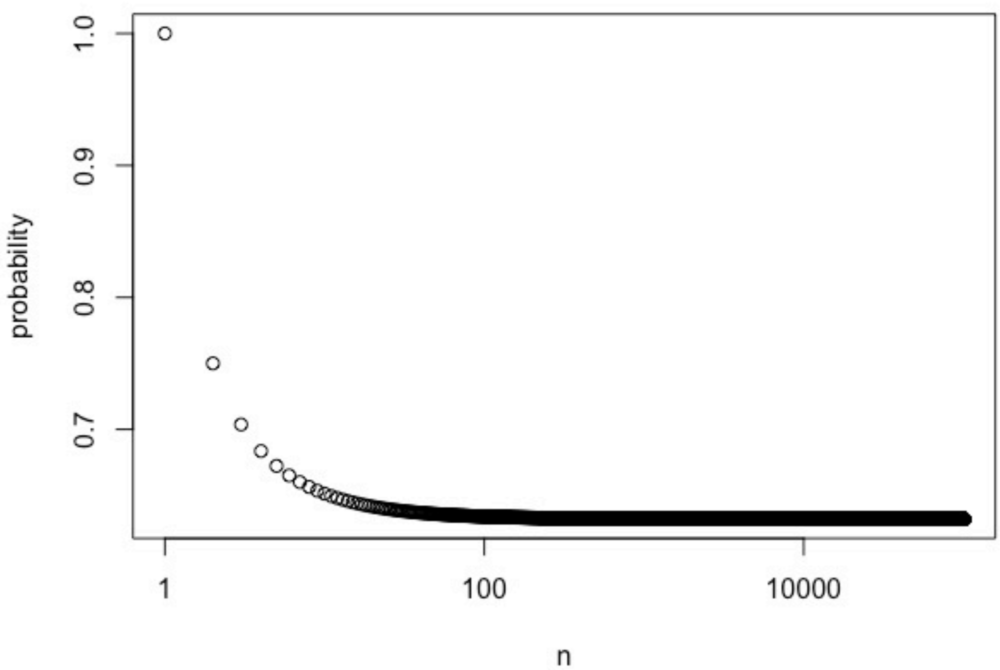
**f.**

```
prob_in_boot.fn(10000)
```

**0.632139**

**g.**

```
x = seq(1, 100000)
y = sapply(x, function (n) { prob_in_boot.fn(n) })
plot(x, y, xlab="n", ylab="probability", log="x")
```




As n increases, the probability decreases, until n is around 100. After that the probability stays pretty constant at 0.6. This is interesting because it looks like once n is larger than 100 (could be 100,000,000!), the probability that an observation is in the bootstrap sample is still around 60%.

**h.**

```
store = rep(NA, 10000)
for (i in 1:10000) {
  store[i] = sum(sample(1:100, rep=TRUE) == 4) > 0
}
mean(store)
```

**0.6364**

Similar to what we expected, based on the chart in part **g**, four is in the bootstrap sample 63% of the time. Part **g** hypothesized that if *n* is greater than 100, then the probability that an observation will be in the bootstrap sample is around 60% of the time. This holds true in this case, where our observation is four and *n* is equal to 10,000, and likely any other that meets the "sample size" condition.

**8.** Section 5.4, page 198, question 5

**a.**

```
library(ISLR)
data(Default)
attach(Default)
set.seed(1)
glm.fit = glm(default~income+balance, family="binomial", data=Default)
```

**b.**

```
set.seed(1)
train = sample(nrow(Default), nrow(Default)/2)
Default.train = Default[train,]
Default.test = Default[-train,]
glm.fit = glm(default~income+balance, family="binomial", data=Default.train)
glm.probs = predict(glm.fit, Default.test, type="response")
glm.pred = ifelse(glm.probs>.5, "Yes", "No")
mean(glm.pred != Default.test$default)
```

**Validation set error = 0.0286**

**c.**

```
set.seed(5)
train = sample(nrow(Default), nrow(Default)/2)
Default.train = Default[train,]
Default.test = Default[-train,]
glm.fit = glm(default~income+balance, family="binomial", data=Default.train)
glm.probs = predict(glm.fit, Default.test, type="response")
glm.pred = ifelse(glm.probs>.5, "Yes", "No")
mean(glm.pred != Default.test$default)
```

**Validation set error = 0.0246**

```
set.seed(25)
train = sample(nrow(Default), nrow(Default)/2)
Default.train = Default[train,]
Default.test = Default[-train,]
glm.fit = glm(default~income+balance, family="binomial", data=Default.train)
glm.probs = predict(glm.fit, Default.test, type="response")
glm.pred = ifelse(glm.probs>.5, "Yes", "No")
mean(glm.pred != Default.test$default)
```

**Validation set error = 0.0256**

```
set.seed(50)
train = sample(nrow(Default), nrow(Default)/2)
Default.train = Default[train,]
Default.test = Default[-train,]
glm.fit = glm(default~income+balance, family="binomial", data=Default.train)
glm.probs = predict(glm.fit, Default.test, type="response")
glm.pred = ifelse(glm.probs>.5, "Yes", "No")
mean(glm.pred != Default.test$default)
```

**Validation set error = 0.024**

Even though three different train/test splits are used, the results do not very much at all.

**d.**

```
set.seed(5)
train = sample(nrow(Default), nrow(Default)/2)
Default.train = Default[train,]
Default.test = Default[-train,]
glm.fit = glm(default~income+balance+student, family="binomial", data=Default.train)
glm.probs = predict(glm.fit, Default.test, type="response")
glm.pred = ifelse(glm.probs>.5, "Yes", "No")
mean(glm.pred != Default.test$default)
```

**Validation set error = 0.0256**

```
set.seed(25)
train = sample(nrow(Default), nrow(Default)/2)
Default.train = Default[train,]
Default.test = Default[-train,]
glm.fit = glm(default~income+balance+student, family="binomial", data=Default.train)
glm.probs = predict(glm.fit, Default.test, type="response")
glm.pred = ifelse(glm.probs>.5, "Yes", "No")
mean(glm.pred != Default.test$default)
```

**Validation set error = 0.0252**

```
set.seed(50)
train = sample(nrow(Default), nrow(Default)/2)
Default.train = Default[train,]
Default.test = Default[-train,]
glm.fit = glm(default~income+balance+student, family="binomial", data=Default.train)
glm.probs = predict(glm.fit, Default.test, type="response")
glm.pred = ifelse(glm.probs>.5, "Yes", "No")
mean(glm.pred != Default.test$default)
```

**Validation set error = 0.024**

The presence of the student dummy variable does not change much in terms of reducing the test error rate.

**9.** Section 5.4, page 199, question 6

**a.**

```
library(ISLR)
data(Default)
attach(Default)
set.seed(1)
glm.fit = glm(default~income+balance, family="binomial", data=Default)
summary(glm.fit)$coef[,1] # Same as coef(glm.fit)
```

```
> summary(glm.fit)$coef[,1]
  (Intercept)        income       balance
-1.154047e+01  2.080898e-05  5.647103e-03
```

**b.**

```
boot.fn = function (data, index) {
  coef(glm(default~income+balance, family="binomial", data=data, subset=index))
}
```

**c.**

```
library(boot)
boot(Default, boot.fn, 1000)
```

```
> boot(Default, boot.fn, 1000)

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Default, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
         original         bias     std. error
t1* -1.154047e+01 -2.085838e-02 4.267477e-01
t2*  2.080898e-05 -1.751077e-07 4.947051e-06
t3*  5.647103e-03  1.440310e-05 2.224863e-04
```

**d.**

The estimated standard errors from the glm() and boot.fn() functions are very close to each other, but not exactly the same. In fact, the glm() function estimates are higher than the boot.fn() estimates! This is likely because the linear assumption causes glm()'s errors to be a little higher than that of bootstrap. Since bootstrap is nonparametric this assumption is not made so it is more accurate.

```
> summary(glm.fit)$coef
                 Estimate   Std. Error    z value      Pr(>|z|)
(Intercept) -1.154047e+01 4.347564e-01 -26.544680 2.958355e-155
income       2.080898e-05 4.985167e-06   4.174178  2.990638e-05
balance      5.647103e-03 2.273731e-04  24.836280 3.638120e-136
```