ISLR | Chapter 4 Exercises

Marshall McQuillen

6/25/2018

Conceptual

1

$$P(X) \ = \ \frac{\epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}} \quad and \quad \epsilon^{\beta_0 \ + \ \beta_1 X_1} \ = \ \frac{P(X)}{1 \ - \ P(X)}$$

$$\epsilon^{\beta_0 \ + \ \beta_1 X_1} \ = \ \frac{\frac{\epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}}}{1 \ - \ \frac{\epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1+\epsilon^{\beta_0 \ + \ \beta_1 X_1}}}$$

$$\epsilon^{\beta_0 \ + \ \beta_1 X_1} \ = \ \frac{\frac{\epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}}}{\frac{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}} \ - \ \frac{\epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1+\epsilon^{\beta_0 \ + \ \beta_1 X_1}}}$$

$$\epsilon^{\beta_0 \ + \ \beta_1 X_1} \ = \ \frac{\frac{\epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}}}{\frac{1}{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}}}$$

$$\epsilon^{\beta_0 \ + \ \beta_1 X_1} \ = \ \frac{\epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}} \cdot \frac{1 \ + \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1}$$

$$\epsilon^{\beta_0 \ + \ \beta_1 X_1} \ = \ \frac{\epsilon^{\beta_0 \ + \ \beta_1 X_1}}{1}$$

$$\epsilon^{\beta_0 \ + \ \beta_1 X_1} \ = \ \epsilon^{\beta_0 \ + \ \beta_1 X_1}$$

2

Prove that maximizing the equation

$$p_k(x) \ = \ \frac{\pi_k \cdot \frac{1}{\sqrt{2\pi\sigma}} \cdot \epsilon^{\ - \ \frac{1}{2\sigma^2} \cdot \left(x \ - \ \mu_k^2\right)}}{\sum_{l=1}^{k} \pi_l \cdot \frac{1}{\sqrt{2\pi\sigma}} \cdot \epsilon^{\ - \ \frac{1}{2\sigma^2} \cdot \left(x \ - \ \mu_k^2\right)}}$$

for k, is equivalent to maximizing the equation

$$\delta_k(x) \ = \ \frac{x\mu_k}{\sigma^2} \ - \ \frac{\mu_k^3}{2\sigma^2} \ + \ log(\pi_k)$$

for k.

As the text says, taking the *log* of the first equation will get us started.

$$log\left(p_k(x)\right) \ = \ log\left(\frac{\pi_k \cdot \frac{1}{\sqrt{2\pi\sigma}} \cdot \epsilon^{\ - \ \frac{1}{2\sigma^2} \cdot \left(x \ - \ \mu_k^2\right)}}{\sum_{l=1}^{k} \pi_l \cdot \frac{1}{\sqrt{2\pi\sigma}} \cdot \epsilon^{\ - \ \frac{1}{2\sigma^2} \cdot \left(x \ - \ \mu_k^2\right)}}\right)$$

$$log\,(p_k(x)) \;=\; log(\pi_k) \;+\; log\left(\frac{1}{\sqrt{2\pi\sigma}}\right) \;-\; \frac{1}{2\sigma^2}\left(x \;-\; \mu_k^2\right) \;-\; log\left(\sum_{l=1}^{k}\pi_l\cdot\frac{1}{\sqrt{2\pi\sigma}}\cdot\epsilon^{\;-\;\frac{1}{2\sigma^2}\cdot\left(x\;-\;\mu_k^2\right)}\right)$$

Expanding the third term in the equation we get

$$log\,(p_k(x)) \;=\; log(\pi_k) + log\left(\frac{1}{\sqrt{2\pi\sigma}}\right) - \frac{1}{2\sigma^2}\left(x^2 \;-\; 2x\mu_k \;+\; \mu_k^2\right) - log\left(\sum_{l=1}^{k}\pi_l\cdot\frac{1}{\sqrt{2\pi\sigma}}\cdot\epsilon^{\;-\;\frac{1}{2\sigma^2}\cdot\left(x\;-\;\mu_k^2\right)}\right)$$

Once again expanding the third term, this time multiplying the $-\frac{1}{\sqrt{2\sigma 62}}$ with all terms within the parenthesis, we get

$$log\,(p_k(x)) \;=\; log(\pi_k) \;+\; log\left(\frac{1}{\sqrt{2\pi\sigma}}\right) \;-\; \frac{x^2}{2\sigma^2} \;+\; \frac{2x\mu_k}{2\sigma^2} \;-\; \frac{\mu_k^2}{2\sigma^2} \;-\; log\left(\sum_{l=1}^{k}\pi_l\cdot\frac{1}{\sqrt{2\pi\sigma}}\cdot\epsilon^{\;-\;\frac{1}{2\sigma^2}\cdot\left(x\;-\;\mu_k^2\right)}\right)$$

simplifying what is now the fourth term in equation, we get

$$log\,(p_k(x)) \;=\; log(\pi_k) \;+\; log\left(\frac{1}{\sqrt{2\pi\sigma}}\right) \;-\; \frac{x^2}{2\sigma^2} \;+\; \frac{x\mu_k}{\sigma^2} \;-\; \frac{\mu_k^2}{2\sigma^2} \;-\; log\left(\sum_{l=1}^{k}\pi_l\cdot\frac{1}{\sqrt{2\pi\sigma}}\cdot\epsilon^{\;-\;\frac{1}{2\sigma^2}\cdot\left(x\;-\;\mu_k^2\right)}\right)$$

Since we are maximizing the equation **over all possible classes of** k, *we can eliminate any terms that are indepedent of* k*.* This leaves us with

$$log\,(p_k(x)) \;=\; log(\pi_k) \;+\; \frac{x\mu_k}{\sigma^2} \;-\; \frac{\mu_k^2}{2\sigma^2}$$

which, when we rearrange the terms slightly, proves that maximizing the two equations are equivalent.

$$log\,(p_k(x)) \;=\; \frac{x\mu_k}{\sigma^2} \;-\; \frac{\mu_k^2}{2\sigma^2} \;+\; log(\pi_k) \;=\; \delta_k(x)$$

3

Following the exact same logic as the proof from question 2, we get to the point where we are looking to reduce the terms that are dependent on k. However, with Quadratic Discriminant Analysis, **we aren't making the assumption that there is a constant variance σ^2 across all classes of k.** Therefore, we can only eliminate the last term (normalizing constant), since all the other terms depend on k. Thus, the equation we are left with (shown below) is the final equation. The third term in the equation shows us that the entire formula does not change in a linear fashion as x changes.

$$\delta_k(x) \;=\; log(\pi_k) \;+\; log\left(\frac{1}{\sqrt{2\pi\sigma_k}}\right) \;-\; \frac{x^2}{2\sigma_k^2} \;+\; \frac{x\mu_k}{\sigma_k^2} \;-\; \frac{\mu_k^2}{2\sigma_k^2}$$

4

- **A**. Considering the fact that the variable X is Uniformly Distributed, if we use observations that are within 0.10 of a test observation x_i, then, on average we will be using 10% of the observations (ignoring cases where $0.05 \geq x_i \geq 0.95$ for the sake of simplicity).

- **B**. Given that we would like to use observations that are within 10% a test observation x_i on both predictors X_1 and X_2, both of which are Uniformly Distributed over [0, 1], then on average we would only be using $0.1^2 = 0.01$ 1% of our observations.
- **C**. Following the logical flow from the previous two question, if we have 100 predictors, all Uniformly Distributed over [0, 1], then the probability $P((x_{i,j} - 0.05) \geq X_j \geq (x_{i,j} + 0.05)) = 0.1$ for each predictor. Therefore, we would use roughly $0.1^{100} = 1.00E^{-100}$ percent of our observations.
- **D**. As we can see from parts **A - C**, as the number of predictors p increases, the number of observations that we would consider "near" our test observations x_i decreases drastically. Therefore, when using KNN for example, more likely than not there aren't going to be any observations that fit our definition of "near." More probable is that some of our observations will be "near" out test observation *with respect to some predictors*, and rather far away from our test observation with respect to other predictors, which then begs the question of whether we should use those "not-so-near" when predicting a quantitative or qualitative response of our test observation.
- **E**. If we would like to create a hypercube that has a sample space equal to 10% of the observations "nearest" to a test observation x_i, the length of each side of the hypercube will be equal to $0.1^{\frac{1}{p}}$, where p is the number of predictors. The intution behind this, **given that we want the "size" (for lack of a more dimension-expansive term) of our our sample space to remain constant at 0.1**, is as follows;
  - With p being equal to 1, this will just be a vector of length 0.1, meaning that roughly 10% of our observations would be within 10% of the range of X_1closest to the test observation x_i.
  - For 2 dimensions, aka 2 predictors, our sample space is an area, and the formula for area is $A = l^2$ (given that we want the length of both vectors to be the same). Thus, the length of our vectors is equal to $l = \sqrt{0.1} = 0.1^{\frac{1}{2}} = 0.32$. This means that roughly 32% of the observations would be within 10% of the range of **one** of the predictors X_1 *or* X_2 closest to the test observation, but still only 10% would be would be within 10% of the range of **both** predictors closest to the test observation.
  - We can see (below) that as our number of dimensions/predictors increases, in order for our sample space to remain constant at 0.1, the length of each vector of the hypercube asymptotically approaches 1. Since *each vector represents a predictor*, and **the length of each vector represents the percent of the observations that are within 10% of the range of ONE of the predictors X_j closest to the test observation**, we can see that as p increases, the percent of observations that are "close" to the test observation with respect to ONE of the predictors increases, yet the percent of observations that are "close" to the test observation with respect to ALL the predictors decreases.
  - In more broad terms, as the number of dimensions increases, the percent of observations that are "close" to one value of a test observation increases, but the percent of observations that are "close" to all values of a test observation decreases.

```r
# percent of observations that are close to ONE value of a test observation approaching 1
percents <- NULL
for (i in 1:100) {
    percent <- 0.1**(1/i)
    percents <- c(percents, percent)
}
print(percents[1:10])
```

```
##  [1] 0.1000000 0.3162278 0.4641589 0.5623413 0.6309573 0.6812921 0.7196857
##  [8] 0.7498942 0.7742637 0.7943282
```

5

- **A**. If the true Bayes decision boundary is linear, we would expect Linear Discriminant Analysis (LDA) to outperform Quadratic Discriminant Analysis (QDA) on both the training and test sets.
- **B**. If the true Bayes decision boundary is non-linear, we would expect QDA to outperform LDA on both the training and test sets.
- **C**. As the sample size n increases, we would expect QDA to outperform LDA. The reason for this is that when the sample size is small, using LDA reduces the variance of the classifier far more than QDA, making it more generalizable to new data. With larger data sets, since reducing the **bias** is more important that reducing the variance, QDA will usually produce a better model.
- **D**. The blanket statement that QDA will outperform LDA, or more generally, more flexible methods will outperform less flexible methods **all** the time is false. Broadly speaking, when modeling data, you are trying to model the true relationship between a set of variables and a target variable. Therefore, the **best** model will be the model that most closely follows this true relationship. So, when the Bayes Decision Boundary is is truly linear, then QDA will not outperform LDA because QDA assumes the decision boundary is quadratic.

6

Since Logistic Regression follows the linear model, where the output Y is the log odds of a success, defining the equation in terms of the problem context is step one.

$$Y \ = \ \beta_0 \ + \ \beta_1 X_1 \ + \ \beta_2 X_2$$

$$Recieve \ an \ A \ = \ intercept \ + \ \beta_1(hours \ studied) \ + \ \beta_2(undergrad \ GPA)$$

The estimates for β_0, β_1 *and* β_2 being equal to -6, 0.05 and 1, respectively. To get the *probability* that a student will recieve an A is a matter of plugging these coefficients into the equation, along with the values for hours studied and undergrad GPA for the student, and then converting the output into a probability.

$$Recieve \ an \ A \ = \ -6 \ + \ 0.05(hours \ studied) \ + \ 1(undergrad \ GPA)$$

- **A**.

$$Recieve \ an \ A \ = \ -6 \ + \ 0.05(40) \ + \ 1(3.5) \ = \ -0.5$$

$$-0.5 \ = \ Log \ Odds \ of \ Success$$

$$\epsilon^{-0.5} \ = Odds \ of \ Success$$

$$Probability(Success) \ = \ \frac{\epsilon^{-0.5}}{1 \ + \ \epsilon^{-0.5}} \ = \ 0.3775407 \ = \ 37.75\%$$

- **B**. To find the number of hours a student with a 3.5 GPA would need to have a 50% change of getting an A, re-working the equation is all that is needed.

$$Log\ Odds\ (A)\ =\ -6\ +\ 0.05(hours\ studied)\ +\ 1(3.5)$$

$$Odds\ (A)\ =\ \epsilon^{-6\ +\ 0.05(hours\ studied)\ +\ 3.5}$$

$$P(A)\ =\ \frac{\epsilon^{-6\ +\ 0.05(hours\ studied)\ +\ 3.5}}{1\ +\ \epsilon^{-6\ +\ 0.05(hours\ studied)\ +\ 3.5}}$$

$$0.5\ =\ \frac{\epsilon^{-6\ +\ 0.05(hours\ studied)\ +\ 3.5}}{1\ +\ \epsilon^{-6\ +\ 0.05(hours\ studied)\ +\ 3.5}}$$

$$0.5(1\ +\ \epsilon^{-6\ +\ 0.05(hours\ studied)\ +\ 3.5})\ = \epsilon^{-6\ +\ 0.05(hours\ studied)\ +\ 3.5}$$

$$0.5(1\ +\ \epsilon^{-2.5\ +\ 0.05(hours\ studied)})\ = \epsilon^{-2.5+\ 0.05(hours\ studied)}$$

$$0.5\ +\ 0.5(\epsilon^{-2.5\ +\ 0.05(hours\ studied)})\ = \epsilon^{-2.5+\ 0.05(hours\ studied)}$$

$$0.5\ +\ 0.5Z\ = Z\ where\ Z\ =\ \epsilon^{-2.5\ +\ 0.05(hours\ studied)}$$

$$0.5 = 0.5Z\ where\ Z\ =\ \epsilon^{-2.5\ +\ 0.05(hours\ studied)}$$

$$0.5 = 0.5(\epsilon^{-2.5\ +\ 0.05(hours\ studied)})$$

$$1 = \epsilon^{-2.5\ +\ 0.05(hours\ studied)}$$

$$log(1) = -2.5\ +\ 0.05(hours\ studied)$$

$$log(1)\ +\ 2.5 = 0.05(hours\ studied)$$

$$\frac{log(1)\ +\ 2.5}{0.05} = hours\ studied\ =\ 50$$

7

Outlining the problem, the goal is to find the probability of company X distributing a dividend given it had a 4% profit last year. This can be expressed as $P(\ Div \mid 4\ )$. Bayes' Theorem can then be rewritten in the context of the problem as:

$$P(\ Div \mid 4\ ) \ = \ \frac{P(\ 4 \mid Div\ )P(\ Div\ )}{P(4)}$$

$$P(\ Div \mid 4\ ) \ = \ \frac{P(\ 4 \mid Div\ )P(\ Div\ )}{P(\ 4 \mid Div\ )P(\ Div\ ) \ + \ P(\ 4 \mid No\ Div\ )P(\ No\ Div\ )}$$

The probability of a company returning a dividend $P(\ Div\ ) \ = \ 0.8$, therefore the probability that a company doesn't return a dividend is $P(\ No\ Div\ ) \ = \ 0.2$. To find the probability that a company had a 4% profit last year given it did (and did not) return a dividend, we use the mean % profit of those that did return a dividend (10) and those that did not (0), and the standard deviation that defines the distributions ($\sqrt{36} \ = \ 6$). Assuming a Normal distribution, the probability that a company had a 4% profit last year given they returned a dividend, $P(\ 4 \mid Div\ )$, is `dnorm(4, 10, 6) = 0.04032845`. The probability a company had a 4% profit given they did *not* return a dividend, $P(\ 4 \mid No\ Div\ )$, is `dnorm(4, 0, 6) = 0.05324133`. Putting all these numbers into the above equation:

$$P(\ Div \mid 4\ ) \ = \ \frac{(0.04032845)(0.8)}{(0.04032845)(0.8) \ + \ (0.05324133)(0.2)} \ = \ 0.7518525 \ = \ 75.2\%$$

8

Since the true evaluation of a model will always be on unseen data, otherwise known as the testing set, we need to compare the testing error rates of the two classification methods. The testing error rate for Logistic Regression is given as 30%.

To find the testing error rate using KNN where K = 1, a deconstruction of a weighted average is needed. However, since the training and testing data sets are equally divided (so their weights would both be 0.5), this will be equal to the arithmetic mean. When K = 1, the nearest neighbor to any given point is itself, and therefore the training error is 0%. So, to find *testing error* in $\bar{x} \ = \ \frac{training\ error\ +\ testing\ error}{2}$, when the known numbers are plugged in, $18 \ = \frac{0\ +\ testing\ error}{2}$, the testing error comes out to be 36%. Therefore, since 30% < 36%, using Logistic Regression would provide a lower error rate on new observations.

9

$$Odds \ = \ \frac{P}{1 \ - \ P} \quad and \quad Probability \ = \ \frac{Odds}{1 \ + \ Odds}$$

- **A**.

$$Odds \ = \ \frac{P}{1 \ - \ P}$$

$$0.37 \ = \ \frac{P}{1 \ - \ P}$$

$$0.37(1 \ - \ P) \ = \ P$$

$$0.37 \; - \; 0.37P \; = \; P$$

$$0.37 = \; 1.37P$$

$$\frac{0.37}{1.37} = \; P \; = \; 0.270073 \; = \; 27\%$$

- **B**.

$$P \; = \; \frac{Odds}{1 \; + \; Odds}$$

$$0.16 \; = \; \frac{Odds}{1 \; + \; Odds}$$

$$0.16(1 \; + \; Odds) \; = \; Odds$$

$$0.16 \; + \; 0.16(Odds) \; = \; Odds$$

$$0.16 \; = \; 0.84(Odds)$$

$$\frac{0.16}{0.84} \; = \; Odds \; = \; 0.1904762 \; = \; 0.19$$

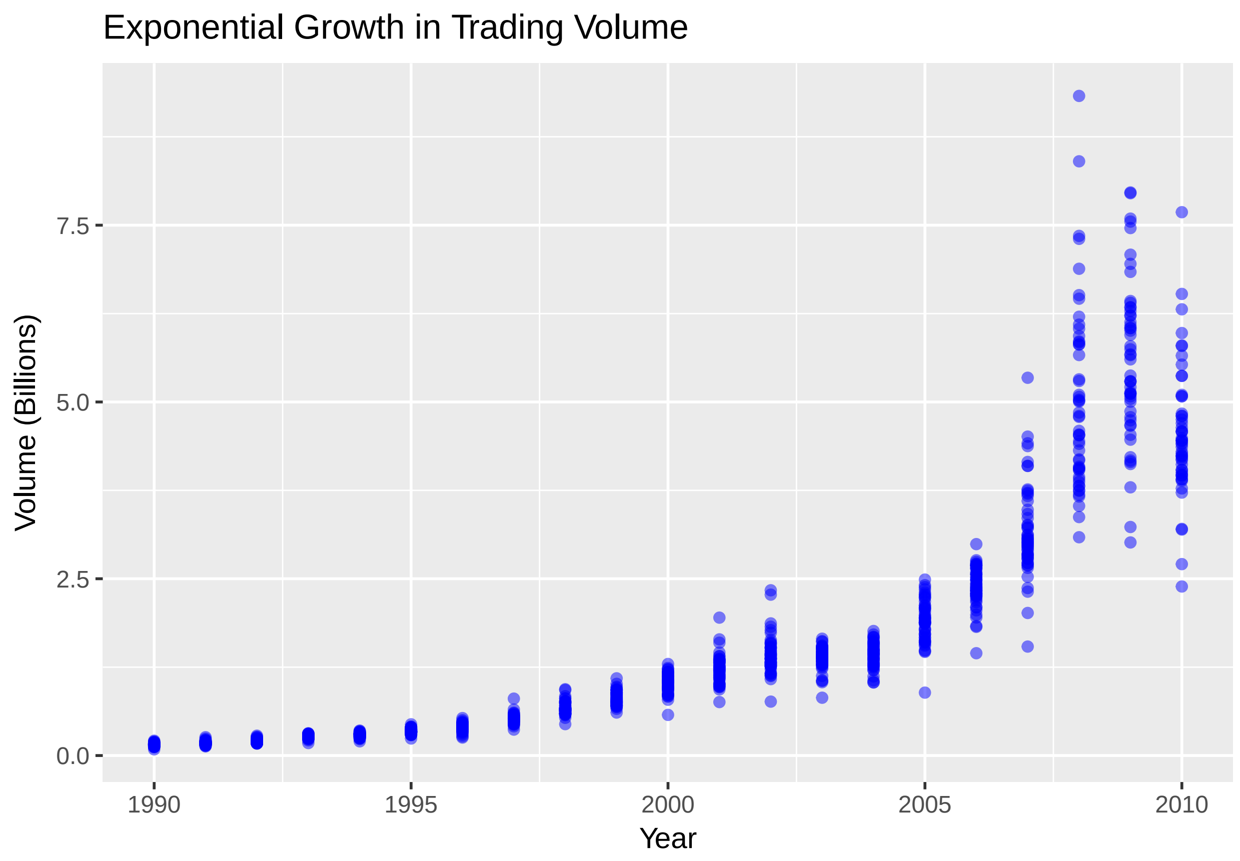
Applied

10

- **A**. Looking at the plot below, it is clear that the trend in trading volume is exponentially increasing over time. In addition, since each year has 52 data points associated with it, the spread of the data points as time goes on indicates an increase in the variance of trading volume from year to year. This is confirmed in the second plot, showing a large increase in the standard deviation of trading volume after 2005. Lastly by looking at a table of the counts of weeks that had a positive or negative return per year, there doesn't seem to be a noticeable pattern in the number of positive or negative week over time. This is confirmed with a Chi-squared Test of Independence, returning a P-Value of 0.4266

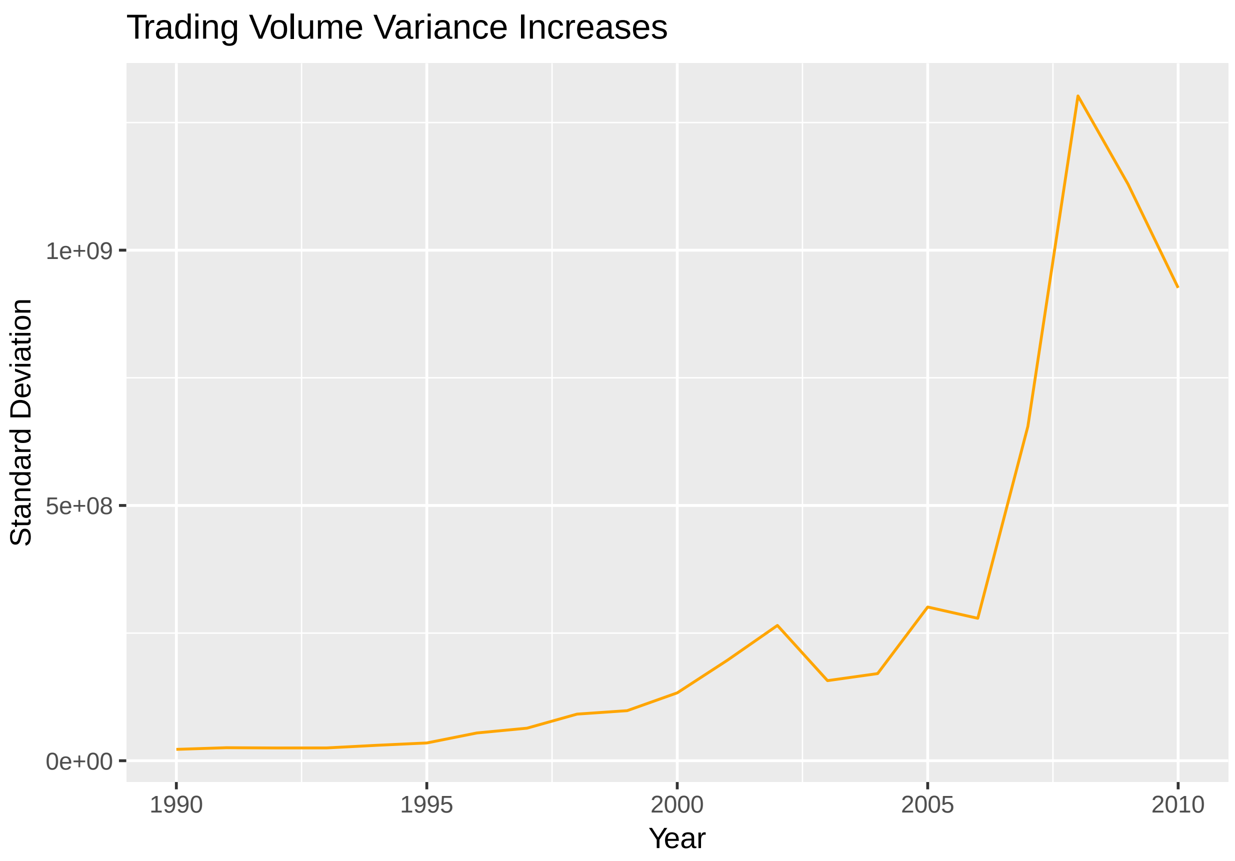
```r
suppressPackageStartupMessages(library(ISLR))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(e1071))

ggplot(data = Weekly, aes(x = Year, y = Volume)) +
    geom_point(color = 'blue', alpha = 0.5) +
    labs(x = 'Year',
         y = "Volume (Billions)",
         title = 'Exponential Growth in Trading Volume')
```



```
by_year <- Weekly %>% group_by(Year)
year_df <- as.data.frame(by_year %>% summarize(stand_dev = sd(Volume)*1000000000))

ggplot(year_df, aes(x = Year, y = stand_dev)) +
    geom_line(color = 'orange') +
    labs(x = 'Year',
         y = 'Standard Deviation',
         title = 'Trading Volume Variance Increases')
```



```r
direction_year_table <- with(Weekly, table(Direction, Year))
direction_year_table
```

```
##          Year
## Direction 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002
##      Down   23   21   23   21   25   16   21   24   19   22   29   27   30
##      Up     24   31   29   31   27   36   32   28   33   30   23   25   22
##          Year
## Direction 2003 2004 2005 2006 2007 2008 2009 2010
##      Down   18   24   22   23   24   29   23   20
##      Up     34   28   30   29   29   23   29   32
```

```r
chisq.test(direction_year_table)
```

```
##
##  Pearson's Chi-squared test
##
## data:  direction_year_table
## X-squared = 20.508, df = 20, p-value = 0.4266
```

- **B**. The only variable that has a statistically significant coefficient is Lag2, the percentage return two weeks previous to the present week.

```r
fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
           Weekly,
           family = 'binomial')
summary(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**. With a Sensitivity (recall) of 92.07%, it is fair to say that this model is pretty good at correctly classifying a positive return when it is in fact a week with a positive return. However, simply looking at the Confusion Matrix shows that this model **tends to simply predict a positive response no matter what**. For example, given that a week returned a negative return, this model would correctly classify that week as having a negative return only 11.15% of the time.

```r
week_copy <- Weekly
week_copy$predict_probabilities <- predict(fit, type = 'response')
week_copy$prediction <- 'Down'
week_copy$prediction[week_copy$predict_probabilities >= 0.5] <- 'Up'
week_copy$prediction <- as.factor(week_copy$prediction)
confusionMatrix(week_copy$prediction, week_copy$Direction, 'Up')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down  Up
##       Down   54  48
##       Up    430 557
##
##                Accuracy : 0.5611
##                  95% CI : (0.531, 0.5908)
##     No Information Rate : 0.5556
##     P-Value [Acc > NIR] : 0.369
##
##                   Kappa : 0.035
##  Mcnemar's Test P-Value : <2e-16
```

```
##
##             Sensitivity : 0.9207
##             Specificity : 0.1116
##          Pos Pred Value : 0.5643
##          Neg Pred Value : 0.5294
##              Prevalence : 0.5556
##          Detection Rate : 0.5115
##    Detection Prevalence : 0.9063
##       Balanced Accuracy : 0.5161
##
##        'Positive' Class : Up
##
```

- **D.**

```
train <- subset(Weekly, Year <= 2008)
test <- subset(Weekly, Year >= 2009)

fit <- glm(Direction ~ Lag2, train, family = 'binomial')

get_accuracy <- function(model, test_data=test) {
    test_data$probs <- predict(model, newdata = test_data, type = 'response')
    test_data$prediction <- 'Down'
    test_data$prediction[test_data$probs >= 0.5] <- 'Up'
    cf <- confusionMatrix(factor(test_data$prediction), factor(test_data$Direction), 'Up')
    return(list(cf$table, cf$overall[1]))
}
get_accuracy(fit)
```

```
## [[1]]
##           Reference
## Prediction Down Up
##       Down    9  5
##       Up     34 56
##
## [[2]]
## Accuracy
##    0.625
```

- **E.**

```
library(MASS)
```

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

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

```
lda_fit <- lda(Direction ~ Lag2, train)
test$prediction <- predict(lda_fit, newdata = test, type = 'response')$class
cf <- confusionMatrix(factor(test$prediction), factor(test$Direction), 'Up')
cf$table
```

```
##           Reference
## Prediction Down Up
##       Down    9  5
##       Up     34 56
```

```r
cf$overall[1]
```

```
## Accuracy
##    0.625
```

- **F**.

```r
library(MASS)
qda_fit <- qda(Direction ~ Lag2, train)
test$prediction <- predict(qda_fit, newdata = test, type = 'response')$class
cf <- confusionMatrix(factor(test$prediction), factor(test$Direction), 'Up')
```

```
## Warning in confusionMatrix.default(factor(test$prediction), factor(test
## $Direction), : Levels are not in the same order for reference and data.
## Refactoring data to match.
```

```r
cf$table
```

```
##           Reference
## Prediction Down Up
##       Down    0  0
##       Up     43 61
```

```r
cf$overall[1]
```

```
##  Accuracy
## 0.5865385
```

- **G**.

```r
library(class)
test <- subset(test, select = -prediction)
nearest_neighbors <- knn(data.frame(train$Lag2), data.frame(test$Lag2),
                         train$Direction)
cf <- confusionMatrix(factor(test$Direction), nearest_neighbors, 'Up')
cf$table
```

```
##           Reference
## Prediction Down Up
##       Down   21 22
##       Up     29 32
```

```r
cf$overall[1]
```

```
##  Accuracy
## 0.5096154
```

- **H**. Both Logistic Regression and Linear Discriminant Analysis had equal accuracy at 62.5%. **This is a good indication that the Bayes Decision Boundary is truly linear, in some manner**. However, it is important to note that Quadratic Discriminant Analysis simply classified all weeks as having a positive return and still got 58.65% accuracy. So, the "best" models in this scenario are only 4% better than a rather unintelligent model.

- **I**. Since there is some evidence that the Bayes Decision Boundary is somewhat linear, and there are only 8 possible variables with which Direction can be predicted, this provides a good opportunity to practice Best Subset Selection, introduced in Chapter 6 of ISLR. The leaps package has a function `regsubsets` that allows the user to perform multiple subset selection methods, including Best, Forward and Backward. Initially including all the variables led to *Today* being included in every model, however looking at the documentation for the Weekly data set shows that the *Today* column is "Percentage return for this week"; in other words a direct proxy for our response variable, Direction. Since this is some form of data leakage I will perform best subset selection without this variable. The results are shown below.

```r
library(leaps)
best_subset_selection <- regsubsets(Direction ~ . - Today,
                                    data = train,
                                    method = 'exhaustive',
                                    nvmax = dim(train)[2] - 2)
summary(best_subset_selection)
```

```
## Subset selection object
## Call: regsubsets.formula(Direction ~ . - Today, data = train, method = "exhaustive",
##     nvmax = dim(train)[2] - 2)
## 7 Variables  (and intercept)
##        Forced in Forced out
## Year       FALSE      FALSE
## Lag1       FALSE      FALSE
## Lag2       FALSE      FALSE
## Lag3       FALSE      FALSE
## Lag4       FALSE      FALSE
## Lag5       FALSE      FALSE
## Volume     FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: exhaustive
##          Year Lag1 Lag2 Lag3 Lag4 Lag5 Volume
## 1  ( 1 ) " "  "*"  " "  " "  " "  " "  " "
## 2  ( 1 ) " "  "*"  "*"  " "  " "  " "  " "
## 3  ( 1 ) " "  "*"  "*"  " "  " "  " "  "*"
## 4  ( 1 ) " "  "*"  "*"  " "  " "  "*"  "*"
## 5  ( 1 ) " "  "*"  "*"  " "  "*"  "*"  "*"
## 6  ( 1 ) " "  "*"  "*"  "*"  "*"  "*"  "*"
## 7  ( 1 ) "*"  "*"  "*"  "*"  "*"  "*"  "*"
```

```r
one_var <- as.formula("Direction ~ Lag1")
two_var <- as.formula("Direction ~ Lag1 + Lag2")
three_var <- as.formula("Direction ~ Lag1 + Lag2 + Volume")
four_var <- as.formula("Direction ~ Lag1 + Lag2 + Volume + Lag5")
five_var <- as.formula("Direction ~ Lag1 + Lag2 + Volume + Lag5 + Lag4")
six_var <- as.formula("Direction ~ Lag1 + Lag2 + Volume + Lag5 + Lag4 + Lag3")
seven_var <- as.formula("Direction ~ Lag1 + Lag2 + Volume + Lag5 + Lag4 + Lag3 + Year")

models <- list(one_var, two_var, three_var, four_var, five_var, six_var, seven_var)

results_df <- NULL
for (model in models) {
    # logistic regression
    lr_fit <- glm(model, train, family = 'binomial')
    lr_acc <- get_accuracy(lr_fit)[[2]]

    # LDA
    lda_fit <- lda(model, train)
    lda_y_hat <- predict(lda_fit, newdata = test, type = 'response')$class
    cf <- confusionMatrix(factor(lda_y_hat), factor(test$Direction), 'Up')
    lda_acc <- cf$overall[1]
    # QDA
    qda_fit <- qda(model, train)
    qda_y_hat <- predict(qda_fit, newdata = test, type = 'response')$class
    cf <- confusionMatrix(factor(qda_y_hat), factor(test$Direction), 'Up')
    qda_acc <- cf$overall[1]

    df <- data.frame("Logistic_Regression_Acc" = lr_acc,
                     "LDA_Acc" = lda_acc,
                     "QDA_Acc" = qda_acc)

    results_df <- rbind(results_df, df)
}
```

```
## Warning in confusionMatrix.default(factor(qda_y_hat), factor(test
## $Direction), : Levels are not in the same order for reference and data.
## Refactoring data to match.
```

```
rownames(results_df) <- c("One Variable Model",
                          "Two Variables Model",
                          "Three Variables Model",
                          "Four Variables Model",
                          "Five Variables Model",
                          "Six Variables Model",
                          "Seven Variables Model")

knitr::kable(results_df, col.names = c('Logistic Regression Accuracy',
                                       'LDA Accuracy',
                                       'QDA Accuracy'))
```

	Logistic Regression Accuracy	LDA Accuracy	QDA Accuracy
One Variable Model	0.5673077	0.5673077	0.5865385
Two Variables Model	0.5769231	0.5769231	0.5576923
Three Variables Model	0.5288462	0.5288462	0.4615385
Four Variables Model	0.4903846	0.5000000	0.4326923
Five Variables Model	0.4615385	0.4711538	0.4326923
Six Variables Model	0.4615385	0.4615385	0.4326923
Seven Variables Model	0.4519231	0.4519231	0.4711538

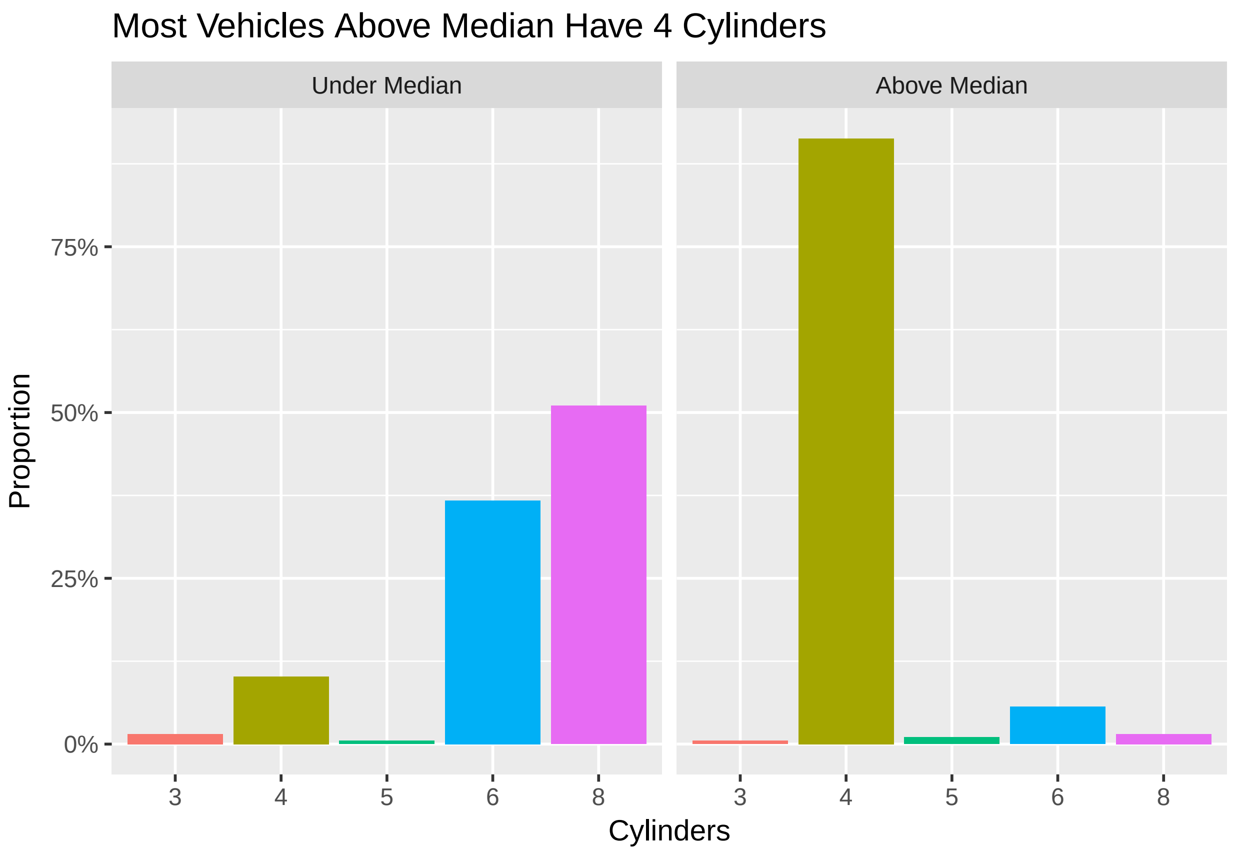
11

- **A.**

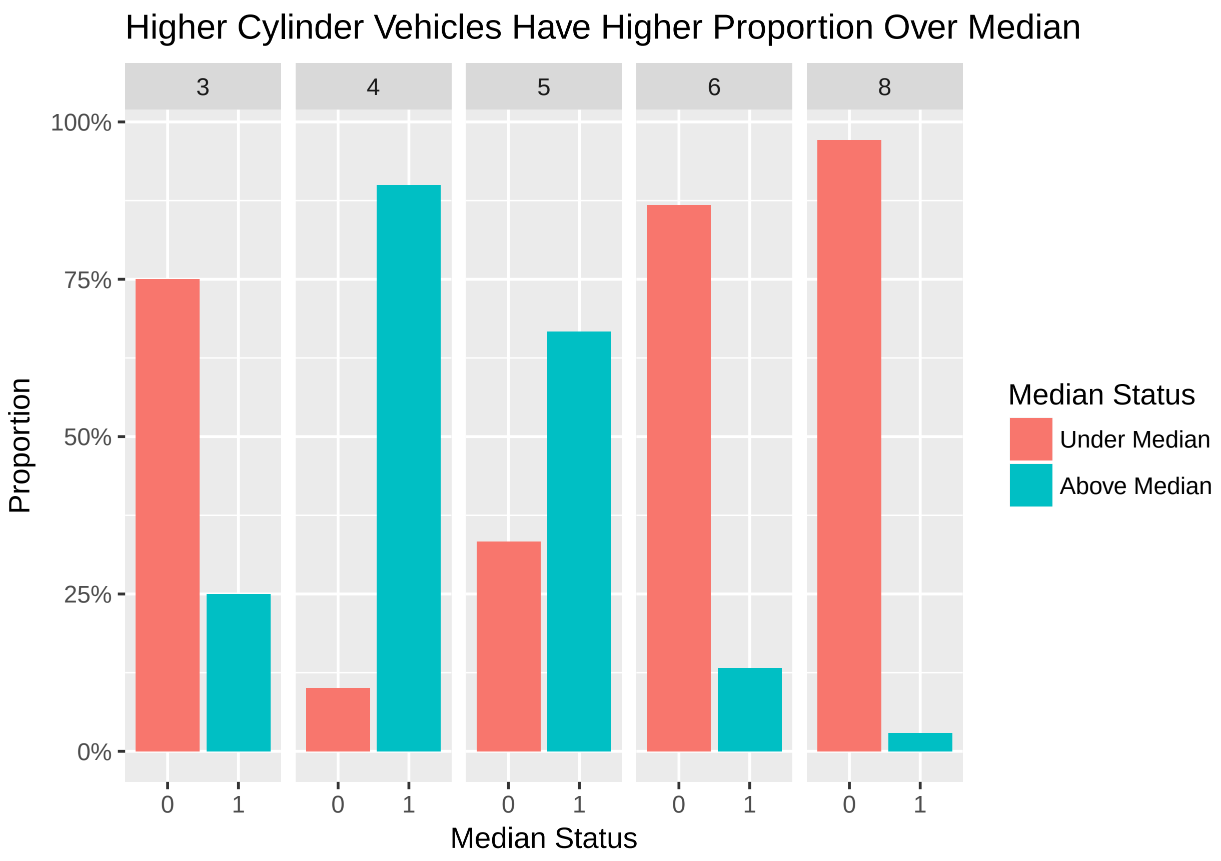
```
df <- Auto
df$mpg01 <- 0
df$mpg01[df$mpg > median(df$mpg)] <- 1
```

- **B.**

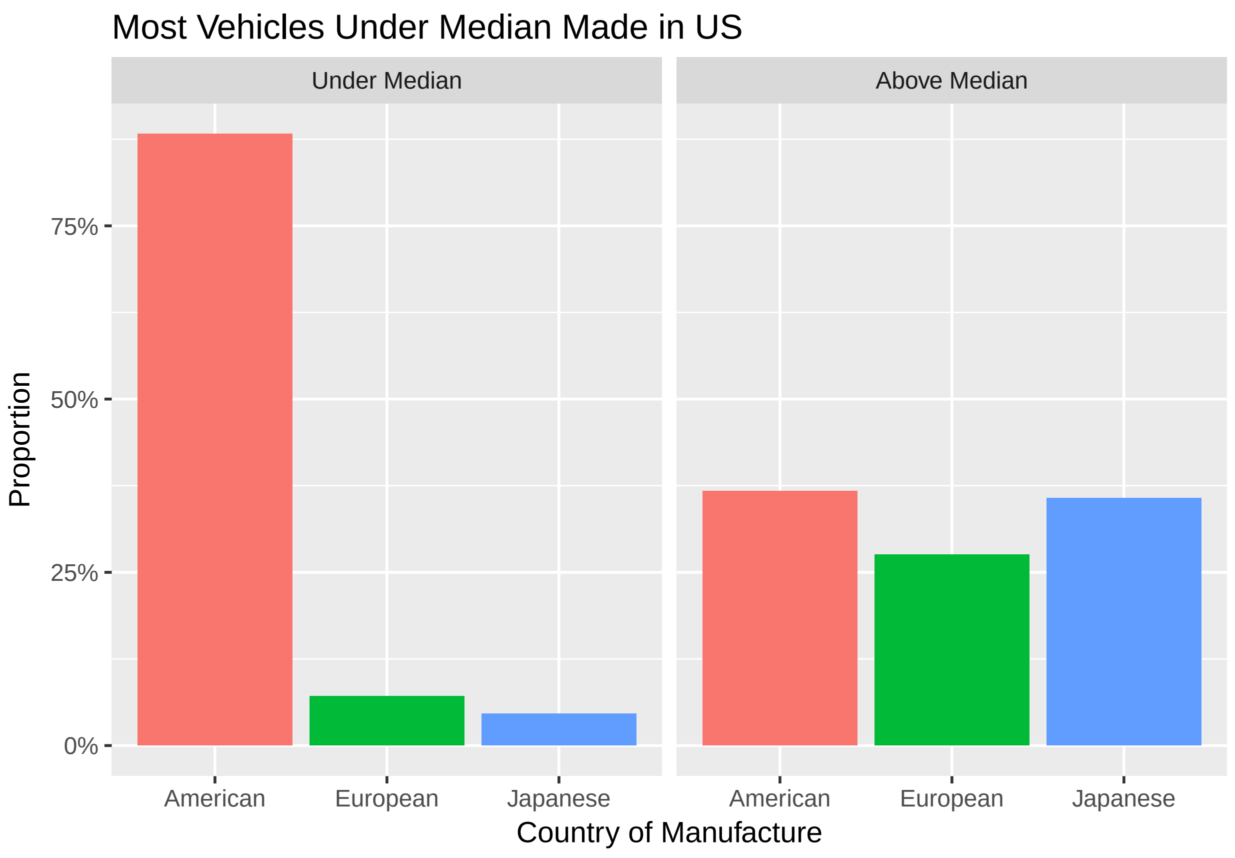
```
ggplot(df, aes(y = cylinders,
               x = factor(cylinders)) +
    geom_bar(aes(y=..prop.., group=mpg01,
                 fill = factor(..x..))) +
    facet_grid(~factor(mpg01, labels = c('Under Median','Above Median')) +
    scale_y_continuous(labels = scales::percent) +
    scale_fill_discrete(guide = FALSE) +
    labs(x = 'Cylinders',
         y = 'Proportion',
         title = "Most Vehicles Above Median Have 4 Cylinders")
```



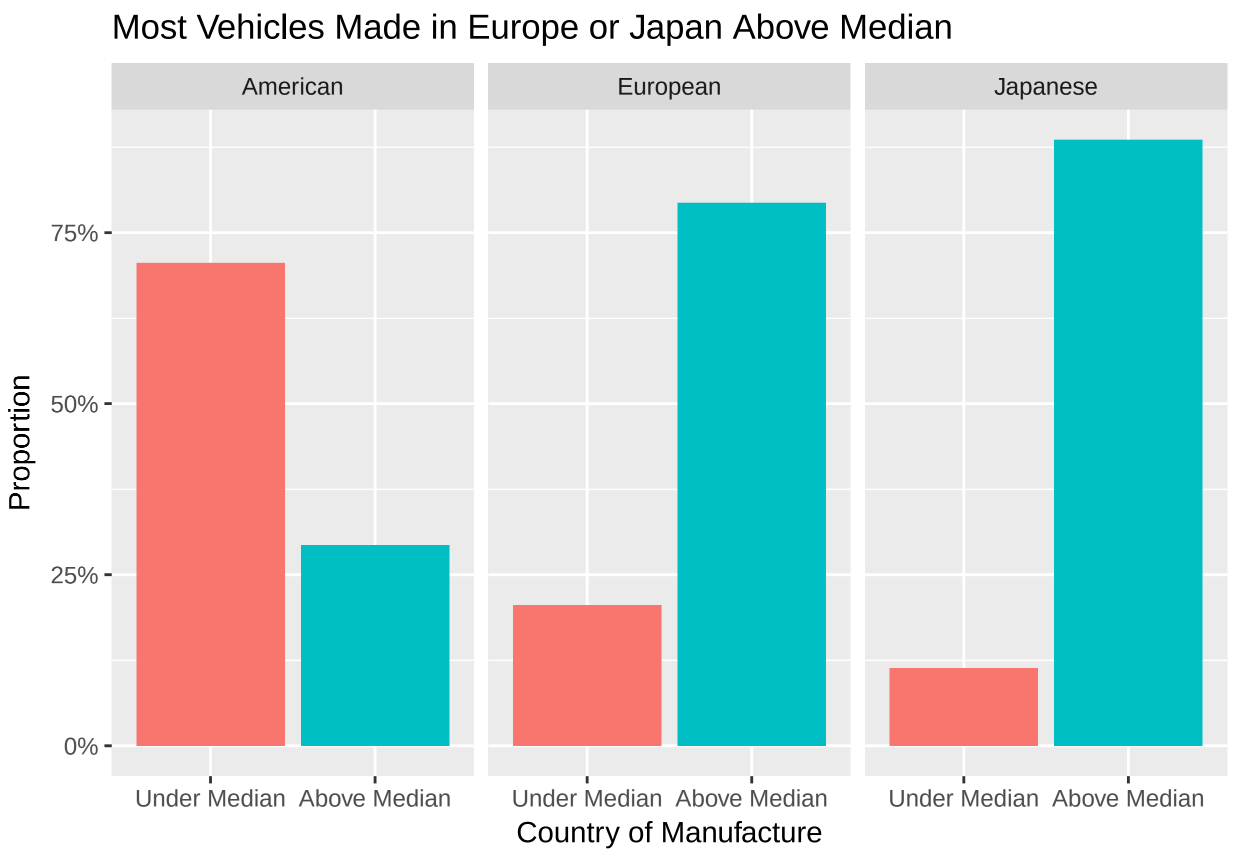
```
ggplot(df, aes(y = mpg01,
               x = factor(mpg01))) +
    geom_bar(aes(y=..prop.., group=cylinders,
                 fill = factor(..x..))) +
    facet_grid(~factor(cylinders)) +
    scale_y_continuous(labels = scales::percent) +
    scale_fill_discrete(name = "Median Status",
                        labels = c('Under Median','Above Median')) +
    labs(x = 'Median Status',
         y = 'Proportion',
         title = "Higher Cylinder Vehicles Have Higher Proportion Over Median")
```



```r
ggplot(df, aes(y = origin,
               x = factor(origin, labels = c('American','European','Japanese')))) +
    geom_bar(aes(y=..prop.., group=mpg01,
                 fill = factor(..x..))) +
    facet_grid(~factor(mpg01, labels = c('Under Median','Above Median'))) +
    scale_y_continuous(labels = scales::percent) +
    scale_fill_discrete(guide = FALSE) +
    labs(x = 'Country of Manufacture',
         y = 'Proportion',
         title = "Most Vehicles Under Median Made in US")
```



```
ggplot(df, aes(y = mpg01,
               x = factor(mpg01, labels = c('Under Median','Above Median')))) +
    geom_bar(aes(y=..prop.., group=origin,
                 fill = factor(..x..))) +
    facet_grid(~factor(origin, labels = c('American','European','Japanese'))) +
    scale_y_continuous(labels = scales::percent) +
    scale_fill_discrete(guide = FALSE) +
    labs(x = 'Country of Manufacture',
         y = 'Proportion', title =
             "Most Vehicles Made in Europe or Japan Above Median")
```



- **C**.

```r
set.seed(5)
sample_size <- floor(0.75*nrow(df))
train_idx <- sample(seq_len(nrow(df)), size = sample_size)
train <- df[train_idx,]
test <- df[-train_idx,]
```

- **D**. Using only the *Cylinders* and *Origin* columns, an accuracy of 90.82% is achieved, corresponding to a testing error rate of 9.18%.

```r
lda_model <- lda(mpg01 ~ cylinders + factor(origin), train)

discriminant_analysis_acc <- function(model, newdata=test) {
    y_hat <- predict(model, newdata = newdata, type = 'response')$class
    cf <- confusionMatrix(factor(y_hat), factor(newdata$mpg01), '1')
    model_accuracy <- cf$overall[1]
    print(cf$table)
    model_accuracy
}

discriminant_analysis_acc(lda_model)
```

```
##           Reference
## Prediction  0  1
##          0 48  3
##          1  6 41

##  Accuracy
## 0.9081633
```

- **E**. Performing QDA with the same variables as LDA, the same accuracy and error are returned, 90.82% and 9.18% respectively.

```
qda_model <- qda(mpg01 ~ cylinders + factor(origin), train)
discriminant_analysis_acc(qda_model)
```

```
##             Reference
## Prediction   0  1
##            0 48  3
##            1  6 41
```

```
##  Accuracy
## 0.9081633
```

- **F**. Logistic Regression has the same accuracy and error rate as QDA and LDA. **Its important to note that this is happening because the "model" is just a series of conditional probabilities**. For example, $P(\ MPG_{01}\ = 1 \,|\, Origin\ = US\ \&\ Cylinders\ = 8\ )\ = 0.016$ and thus the prediction for all vehicles that have 8 cylinders and were made in the US will be predicted to be below the median. Each combination of country of manufacture and number of cylinders will have its own predicted probability.

```
lr_model <- glm(mpg01 ~ cylinders + factor(origin), train, family = 'binomial')
probs <- predict(lr_model, newdata = test, type = 'response')
y_hat <- rep(0, length(probs))
y_hat[probs >= 0.5] <- 1
cf <- confusionMatrix(factor(y_hat), factor(test$mpg01), '1')
cf$table
```

```
##             Reference
## Prediction   0  1
##            0 48  3
##            1  6 41
```

```
cf$overall[1]
```

```
##  Accuracy
## 0.9081633
```

- **G**. Once again the accuracy and testing error rate are the same as the other algorithms, regardless of K.

```
knn_train <- cbind(train$cylinders, train$origin)
knn_test <- cbind(test$cylinders, test$origin)

nn_train <- function(k) {
    nearest_neighbors <- knn(data.frame(knn_train),
                             data.frame(knn_test),
                             train$mpg01,
                             k=k)
    cf <- confusionMatrix(nearest_neighbors, factor(test$mpg01), '1')
    print(cf$table)
    print(cf$overall[1])
}

nn_train(2)
```

```
##             Reference
## Prediction   0  1
##           0 48  3
##           1  6 41
##  Accuracy
## 0.9081633
```

```
nn_train(10)
```

```
##            Reference
## Prediction  0  1
##           0 48  3
##           1  6 41
##  Accuracy
## 0.9081633
```

```
nn_train(15)
```

```
##            Reference
## Prediction  0  1
##           0 48  3
##           1  6 41
##  Accuracy
## 0.9081633
```

12

- **A.**

```
Power <- function() {
    print(2^3)
}
Power()
```

```
## [1] 8
```

- **B.**

```
Power2 <- function(x, a) {
    print(x^a)
}
Power2(3,8)
```

```
## [1] 6561
```

- **C.**

```
Power2(10, 3)
```

```
## [1] 1000
```

```
Power2(8, 17)
```

```
## [1] 2.2518e+15
```

```
Power2(131, 3)
```

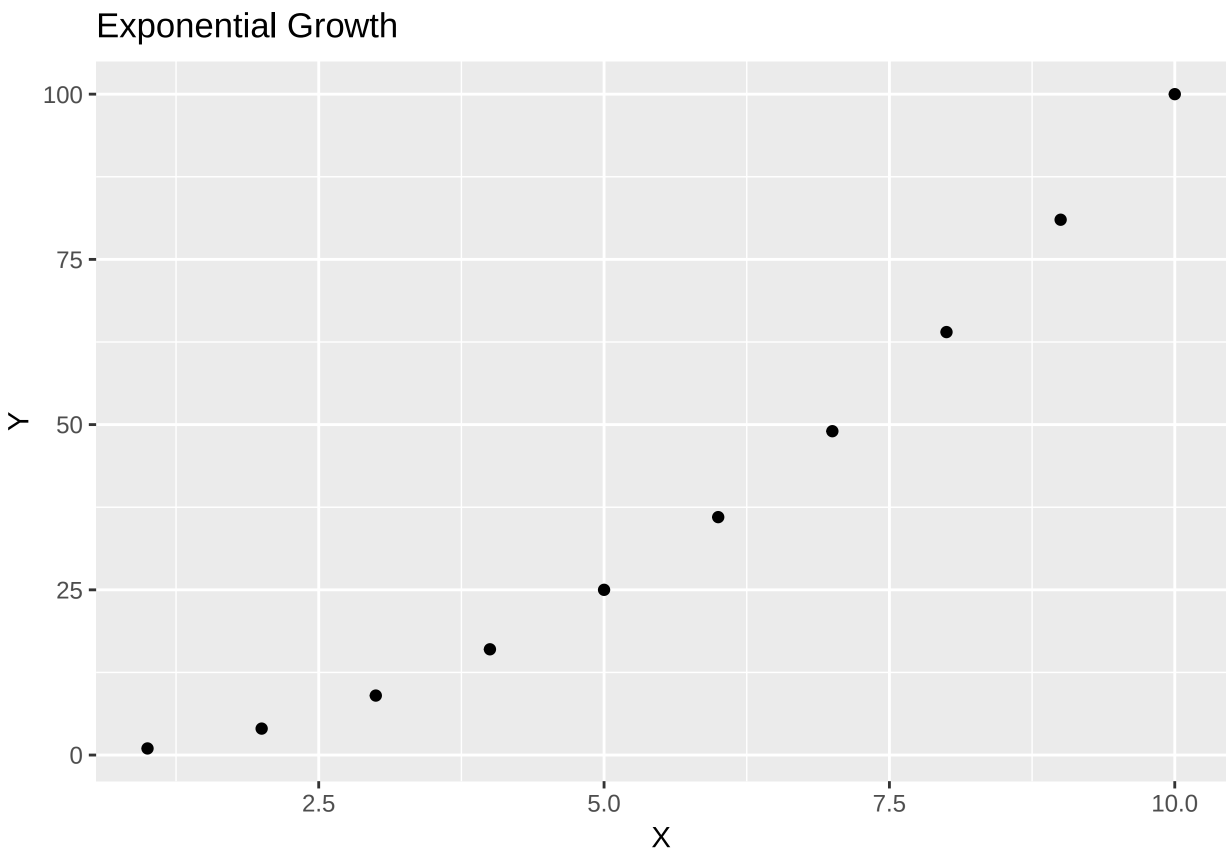
```
## [1] 2248091
```

- **D.**

```
Power3 <- function(x, a) {
    return(x^a)
}
```

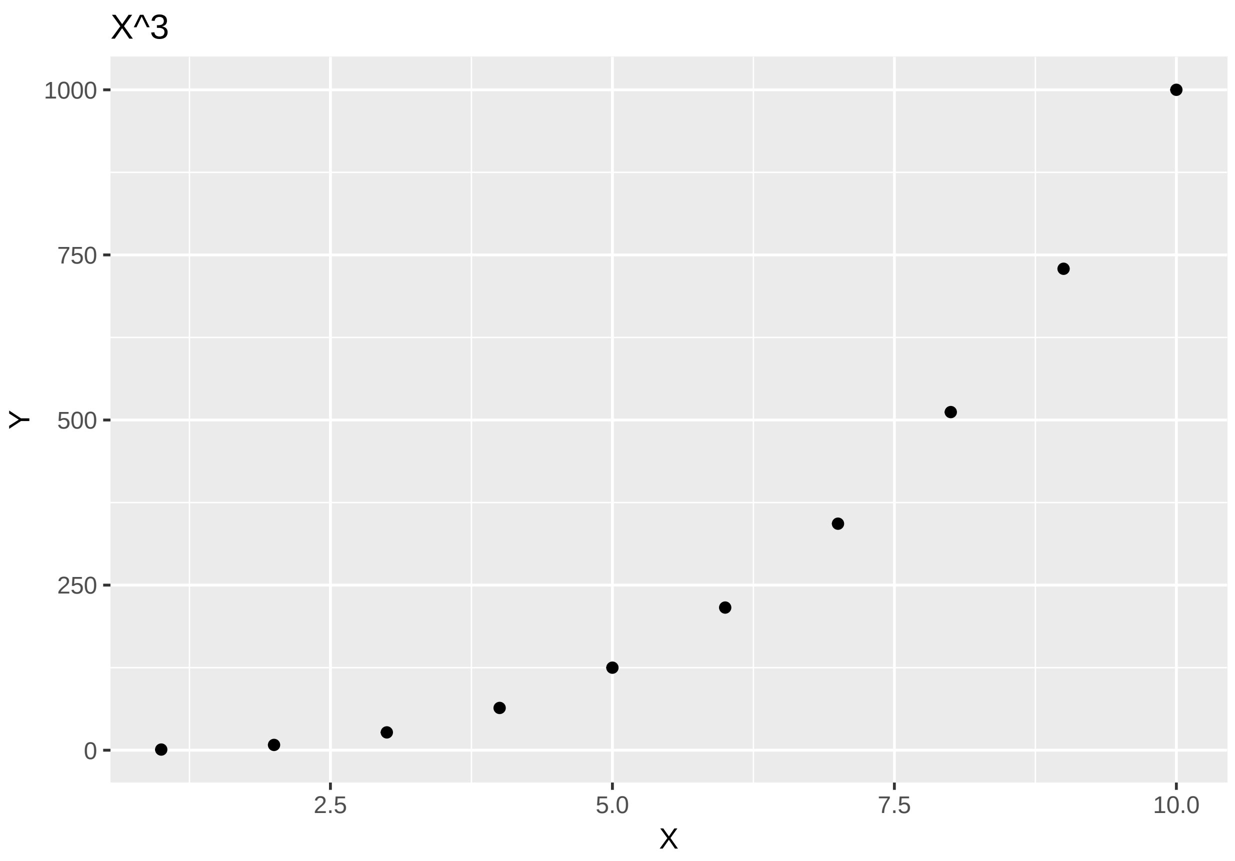
- **E**.

```
x <- 1:10
y <- Power3(x, 2)
qplot(x, y, xlab = 'X', ylab = 'Y', main = 'Exponential Growth')
```



- **F**.

```
PlotPower <- function(x_range, a) {
    y <- Power3(x_range, a)
    title <- paste("X^", as.character(a), sep = "")
    return(qplot(x_range, y, xlab = 'X', ylab = 'Y', main = title))
}
first_plot <- PlotPower(1:10, 3)
first_plot
```



13

After performing logistic regression with all the variables (excluding *crim* to prevent any data leakage), I removed the three variable with the least significant P-Values, rm (average number of rooms per dwelling), indus (proportion of non-retail business acres per town), and age (proportion of owner-occupied units built prior to 1940). This only reduced the accuracy from ~86% to 83.33%, not a bad tradeoff considering the reduction in dimension size. I used this same model for LDA, QDA and KNN with K ranging from 1 to 25. Both LDA and QDA outperformed logistic regression, however, KNN outperformed all other models. The plot on the last page shows a relatively consistent decrease in accuracy as K increases.

```r
library(MASS)
df <- Boston
df$crim01 <- 0
df$crim01[df$crim > median(df$crim)] <- 1

# train test split
set.seed(5)
sample_size <- floor(0.75*nrow(df))
train_idx <- sample(seq_len(nrow(df)), size = sample_size)
df.train <- df[train_idx,]
df.test <- df[-train_idx,]

# logistic regression
lr_model <- glm(crim01 ~ . - crim - rm - indus - age, df.train, family = 'binomial')
probs <- predict(lr_model, newdata = df.test, type = 'response')
y_hat <- 0
```

```r
y_hat[probs >= 0.5] <- 1
confusionMatrix(factor(y_hat), factor(df.test$crim01), '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  1  0
##          1 12 59
##
##                Accuracy : 0.8333
##                  95% CI : (0.727, 0.9108)
##     No Information Rate : 0.8194
##     P-Value [Acc > NIR] : 0.451990
##
##                   Kappa : 0.1202
##  Mcnemar's Test P-Value : 0.001496
##
##             Sensitivity : 1.00000
##             Specificity : 0.07692
##          Pos Pred Value : 0.83099
##          Neg Pred Value : 1.00000
##              Prevalence : 0.81944
##          Detection Rate : 0.81944
##    Detection Prevalence : 0.98611
##       Balanced Accuracy : 0.53846
##
##        'Positive' Class : 1
##
```

```r
# lda
lda_model <- lda(crim01 ~ . - crim - rm - indus - age, df.train)
y_hat <- predict(lda_model, newdata = df.test)$class
confusionMatrix(factor(y_hat), factor(df.test$crim01), '1')
```

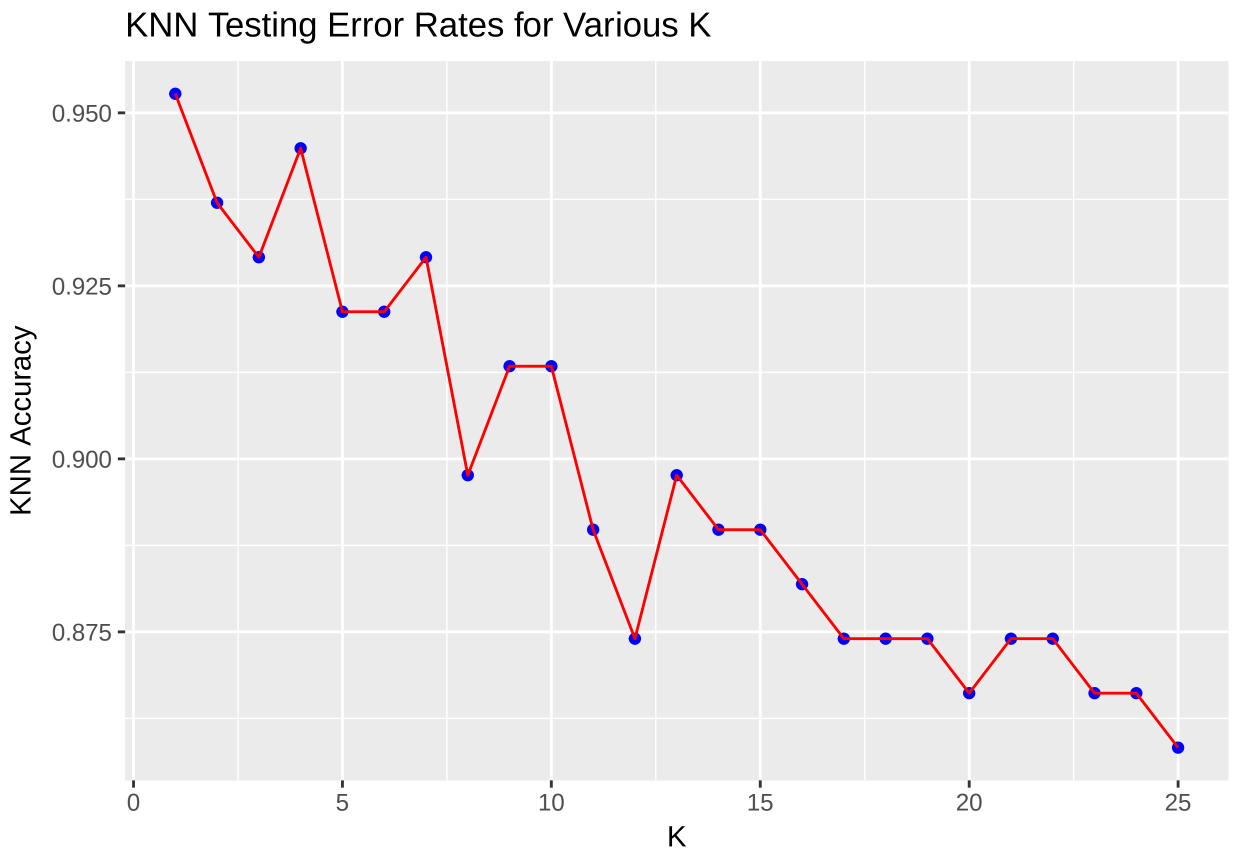
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 56 10
##          1  8 53
##
##                Accuracy : 0.8583
##                  95% CI : (0.7853, 0.9138)
##     No Information Rate : 0.5039
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7164
##  Mcnemar's Test P-Value : 0.8137
##
##             Sensitivity : 0.8413
##             Specificity : 0.8750
##          Pos Pred Value : 0.8689
##          Neg Pred Value : 0.8485
##               Prevalence : 0.4961
##           Detection Rate : 0.4173
##     Detection Prevalence : 0.4803
##        Balanced Accuracy : 0.8581
##
##         'Positive' Class : 1
##
```

```r
# qda
qda_model <- qda(crim01 ~ . - crim - rm - indus - age, df.train)
y_hat <- predict(qda_model, newdata = df.test)$class
confusionMatrix(factor(y_hat), factor(df.test$crim01), '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 58 13
##          1  6 50
##
##                Accuracy : 0.8504
##                  95% CI : (0.7763, 0.9075)
##     No Information Rate : 0.5039
##     P-Value [Acc > NIR] : 2.673e-16
##
##                   Kappa : 0.7005
##  Mcnemar's Test P-Value : 0.1687
##
##             Sensitivity : 0.7937
##             Specificity : 0.9062
##          Pos Pred Value : 0.8929
##          Neg Pred Value : 0.8169
##              Prevalence : 0.4961
##          Detection Rate : 0.3937
##    Detection Prevalence : 0.4409
##       Balanced Accuracy : 0.8500
##
##        'Positive' Class : 1
##
```

```r
# knn
knn_train <- subset(df.train, select = - c(crim, crim01, rm, indus, age))
knn_test <- subset(df.test, select = - c(crim, crim01, rm, indus, age))

knn_accuracy <- function(k_range) {
    accuracy_vector <- c()
    for (k in k_range) {
        nearest_neighbors <- knn(knn_train,
                                 knn_test,
                                 df.train$crim01,
                                 k = k)
        cf <- confusionMatrix(nearest_neighbors, factor(df.test$crim01), '1')
        accuracy_vector <- c(accuracy_vector, cf$overall[1])
    }
    return(accuracy_vector)
}
k_range <- 1:25
k_range_acc <- knn_accuracy(k_range)
results_df <- data.frame(k_range, k_range_acc)
ggplot(results_df, aes(x = k_range, y = k_range_acc)) +
    geom_point(color = 'blue') +
    geom_line(color = 'red') +
    labs(x = 'K', y = 'KNN Accuracy', title = 'KNN Testing Error Rates for Various K')
```


KNN Testing Error Rates for Various K
KNN Accuracy
K