STAT 639 HW 2

Jack Cunningham (jgavc@tamu.edu)

2/20/24

Homework 2

1)

We are tasked to show that the logistic function representation and logit representation for the logistic regression model are equivalent:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$

Manipulating numerator and denominator on right side:

$$p(X) = (1+e^{\beta_0+\beta_1 X})p(X)\frac{1}{e^{\beta_0+\beta_1 X}}p(X)$$

$$p(X) = p(X)(\frac{p(X)+e^{\beta_0+\beta_1 X}p(X)}{e^{\beta_0+\beta_1 X}})$$

Dividing both sides by $p(X)$.

$$1 = \frac{p(X)+e^{\beta_0+\beta_1 X}p(X)}{e^{\beta_0+\beta_1 X}}$$

$$e^{\beta_0+\beta_1 X} - e^{\beta_0+\beta_1 X}p(X) = p(X)$$

Factoring left side:

$$e^{\beta_0+\beta_1 X}(1-p(X)) = p(X)$$

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}$$

2)

a) We will use 1/10 of the available observations since we are looking at observations within 10% of our chosen X.

b) We have (X_1, X_2) uniformly distributed on $[0,1] \times [0,1]$. Since we want to look at observations within 10% on both X_1 and X_2 we are left with a fraction of: $(\frac{1}{10})(\frac{1}{10}) = \frac{1}{100}$.

c) Extending our previous answer, with $p = 100$ we will only use $(\frac{1}{10})^{100}$ of the available observations.

d) As p increases the fraction of observations within a given range is extremely small this leads us to either extend our range to find "like" observations or be left with very few observations that drive our prediction.

e) If we want to use, on average, 10% of our training observations to inform a prediction on a test observation we can treat the space utilized as a hyper-cube. The hyper-cube then has a volume of 0.1.

The function for the volume of a hyper-cube is $(x)^n$ where x is the length of each equal side and n is the number of sides.

Our objective is to solve for the length of each side, so $x = V^{1/n}$ where V is the volume.

For $p = 1$ we have a length of $x = (.1)^1 = .1$.

For $p = 2$ we have a length of $x = (.1)^{1/2} = .3162278$

For $p = 100$ we have a length of $x = (.1)^{1/100} = .9772372$

As p increases in order to get a certain amount of nearby values we need extend the range of acceptable values in each predictor. We can see that at high p, such as 100, the range for each predictor is .9772. This causes concern as the nearby values aren't nearby at all.

3)

We are given a training data set and apply two classification procedures:

1. Logistic regression with an error rate of 20% on training data and 30% on test data.
2. 1-nearest neighbors with an average error rate of 18% on training and test data.

With the information we are given the test error is the best metric for assessing model performance. In order to compare the performance of the two methods we need to obtain the test error rate of 1-nearest neighbors.

The nature of 1-nearest neighbors is assigning an observation to its closest observation. After the model is trained and we ask it to classify an already seen observation it will guess the correct class every time. Therefore the training error for 1-nearest neighbors will be 0%. Since we are given the average error rate of 18% we can easily find the test error rate, 36%.

We would choose logistic regression in this case as it has a lower error rate than 1-nearest neighbors.

4)

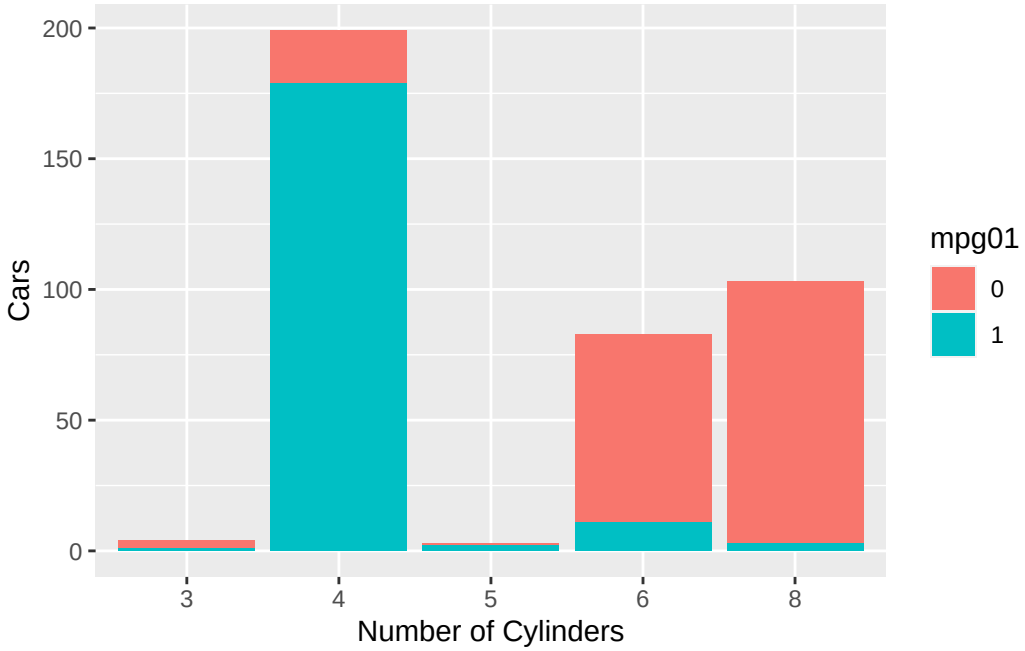
```
require(ISLR2)
```

```
Warning: package 'ISLR2' was built under R version 4.3.2
```

```
require(tidyverse)
```

```
Auto_mpg <- Auto |>
  mutate(mpg01 = as.factor(ifelse(mpg > median(mpg), 1, 0))) |>
  mutate(cylinders = as.factor(cylinders)) |>
  mutate(origin = as.factor(origin))
```

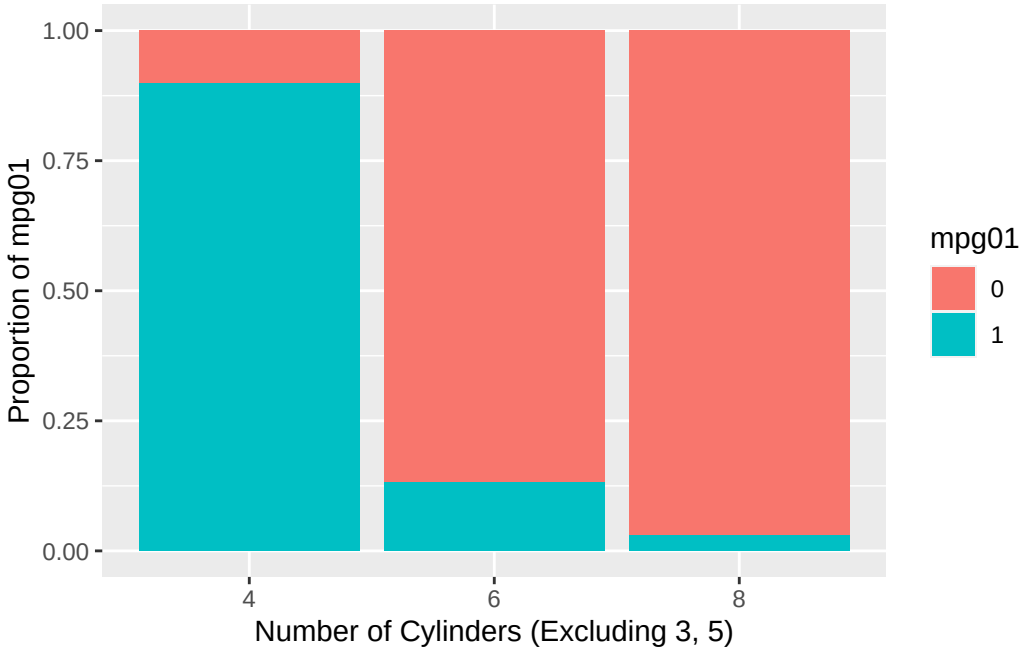
```
Auto_mpg |>
  ggplot(aes(x = cylinders, fill = mpg01)) +
  geom_bar() +
  labs(x = "Number of Cylinders",
       y = "Cars")
```



We can see that the number of cylinders appears to have a high impact on whether the mpg is above the median. We also see that the different cylinder cars are not equal, there are many 4-cylinder cars while very few 3 and 5 cylinder cars.

Therefore thinking back to our previous graph, let us disregard cylinders 3 and 5 due to their extremely low occurrences and compare the rates of MPG01.

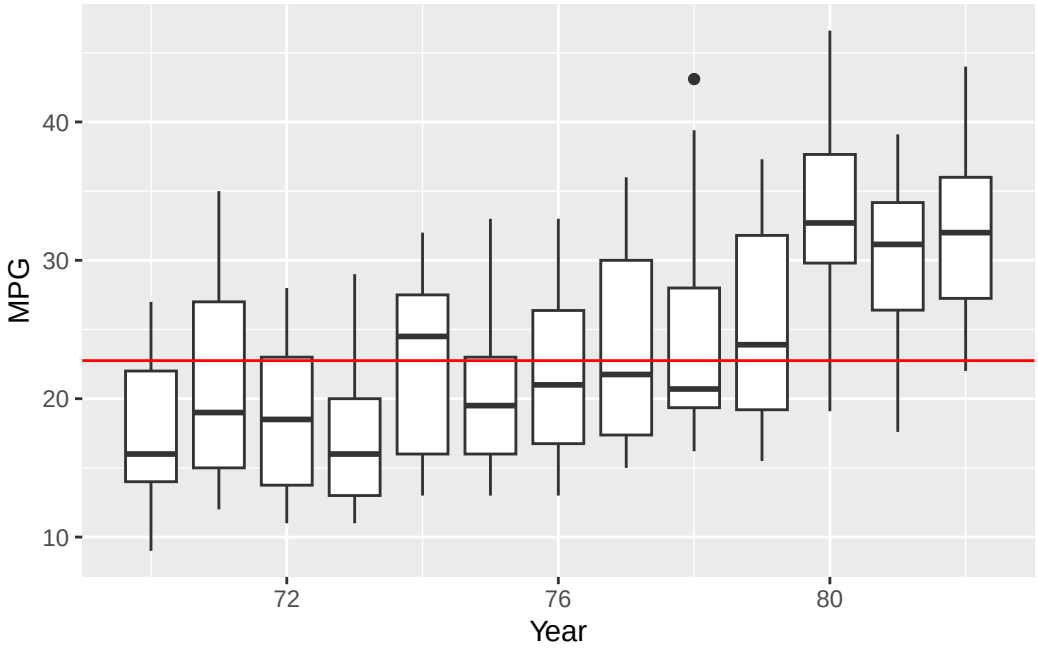
```
Auto_mpg |>
  filter(cylinders %in% c(4, 6, 8)) |>
  ggplot(aes(x = cylinders, fill = mpg01)) +
  geom_bar(position = "fill") +
  labs(x = "Number of Cylinders (Excluding 3, 5)",
       y = "Proportion of mpg01",)
```



Now we can see that the number of cylinders appear to be very related to whether a car is above the median MPG, the lower the cylinder the higher the mpg.

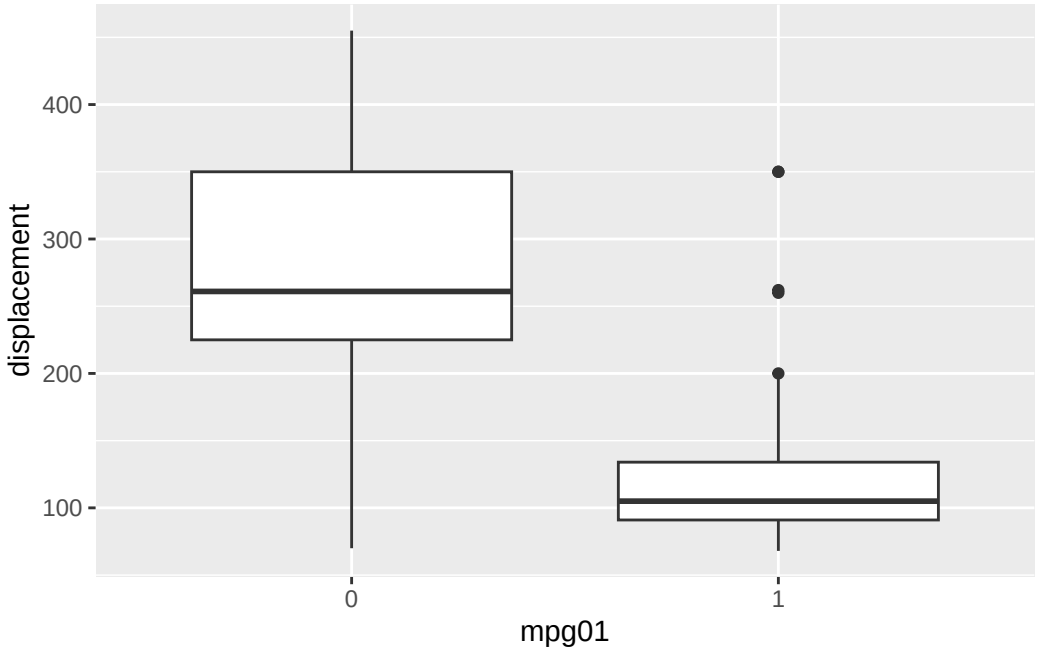
Let's now take a look at the relationship of mpg and the year of the car.

```
Auto_mpg |>
  ggplot(aes(x = year, y = mpg)) +
  geom_boxplot(aes(group = year)) +
  geom_hline(aes(yintercept = median(mpg)), col = "red") +
  labs(
    x = "Year",
    y = "MPG"
  )
```



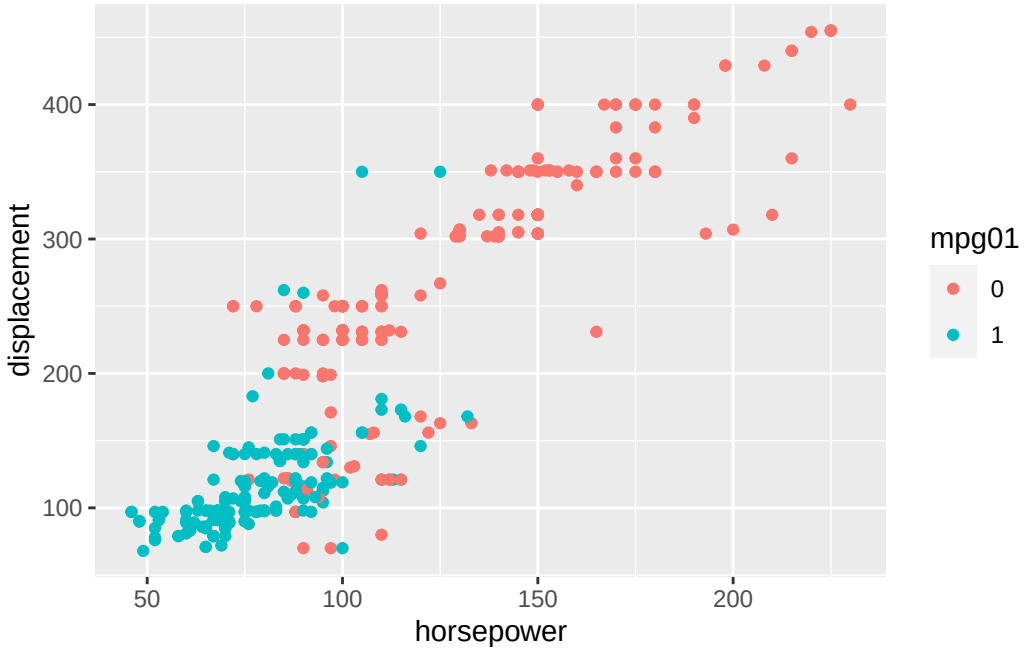
Here we can see that there is a positive correlation between year and miles per gallon. The red line is the median of mpg, we can see that those later years have samples almost entirely above the median. From this we can gather that year seems to be a useful feature to predict mpg01.

```
Auto_mpg |>
  ggplot(aes(x = mpg01, y = displacement)) +
  geom_boxplot()
```



The engine displacement also appears to have a large impact on whether the mpg is above or below the median, there appears to be large separation between the two classes on this feature. There is a high chance that the displacement feature is related to horsepower, let's investigate that.

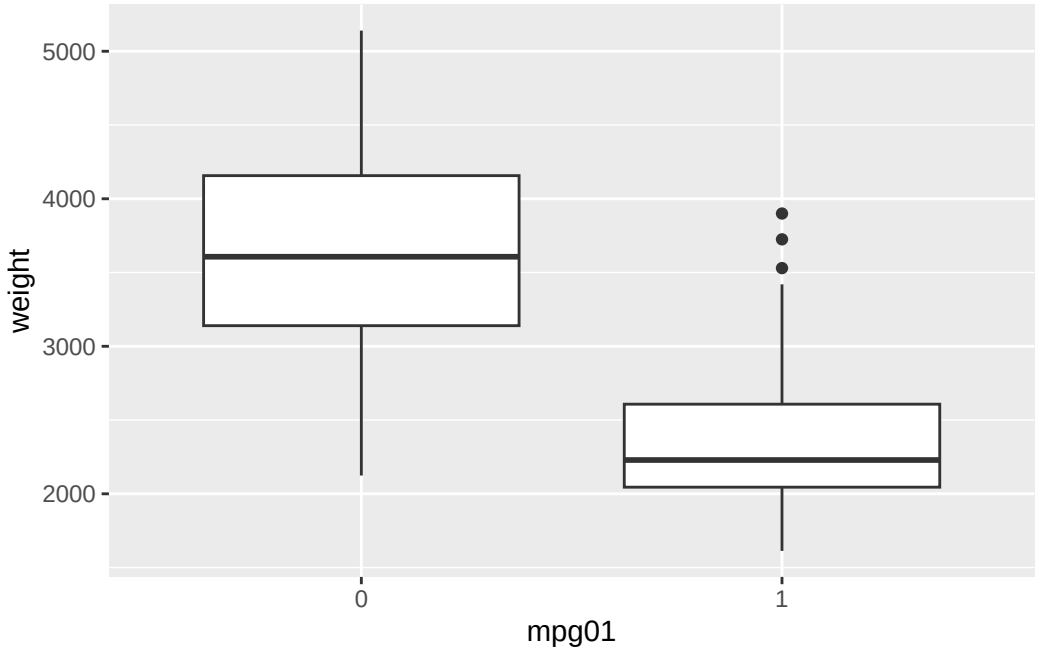
```
Auto_mpg |>
  ggplot(aes(x = horsepower, y = displacement)) +
  geom_point(aes(color = mpg01))
```



Here we can see that horsepower and displacement are closely related and the vehicles with above median mpg are low in both.

Let's see the relationship between weight and mpg_01.

```
Auto_mpg |>
  ggplot(aes(x = mpg01, y = weight)) +
  geom_boxplot()
```

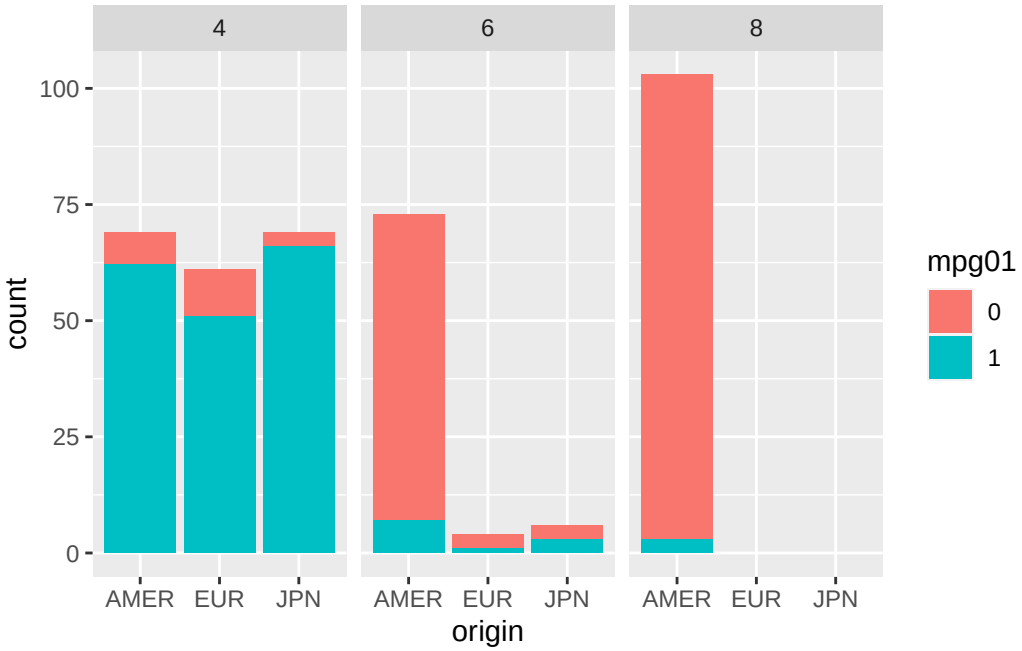


There also appears to be a relationship between mpg01 and the weight of the vehicle.

It is also worth taking a look to see whether the origin of the vehicle has an impact on the mpg of the vehicle. But it is also worth controlling for the type of vehicle created by the regions.

Let's use the number of cylinders of the car as a way of digging into how the origin affects the mpg of the vehicle.

```
Auto_mpg |>
  filter(cylinders %in% c(4,6,8)) |>
  ggplot(aes(x = origin, fill = mpg01)) +
  geom_bar() +
  scale_x_discrete(labels = c("1" = "AMER", "2" = "EUR", "3" = "JPN")) +
  facet_wrap(~cylinders)
```



Here we can see that Japan and Europe almost entirely create cars with 4 cylinders. America creates the most cars in the data set as well as nearly all the 6 and 8 cylinder cars.

From this we can see that it is hard to parse out the impact of the origin of the car after also controlling for the cylinders of the cars created.

An overall takeaway from this analysis is that MPG01 appears to have a strong relationship with nearly every feature but many features are correlated strongly with each other. Say horsepower and engine displacement or origin and cylinder. There is also high separation in the distribution of features for the two different classes. An example of this would be in weight and displacement.

c)

```
require(tidymodels)

set.seed(50)
auto_split <- initial_split(Auto_mpg, prop = 0.75)
auto_train_data <- training(auto_split)
auto_test_data <- testing(auto_split)

mpg01_test <- auto_test_data$mpg01
```

d)

```r
library(MASS)
auto_lda_fit <- lda(
  mpg01 ~ cylinders + displacement + horsepower + weight + acceleration + year,
  data = auto_train_data)
auto_lda_fit
```

```
Call:
lda(mpg01 ~ cylinders + displacement + horsepower + weight +
    acceleration + year, data = auto_train_data)

Prior probabilities of groups:
        0         1
0.4897959 0.5102041

Group means:
  cylinders4  cylinders5 cylinders6 cylinders8 displacement horsepower   weight
0     0.0625 0.000000000 0.36111111     0.5625     284.0833  134.02083 3678.174
1     0.9000 0.006666667 0.06666667     0.0200     118.3033   79.73333 2343.753
  acceleration     year
0     14.33403 74.46528
1     16.41600 77.67333

Coefficients of linear discriminants:
                      LD1
cylinders4    2.7894049618
cylinders5    3.4479357762
cylinders6    0.1655129960
cylinders8    0.7159273978
displacement -0.0021572034
horsepower    0.0004800910
weight       -0.0008018795
acceleration -0.0267374436
year          0.1042211758
```

```r
auto_lda_pred <- predict(auto_lda_fit, auto_test_data, type = "response")
auto_lda_pred_class <- auto_lda_pred$class

table(auto_lda_pred_class, mpg01_test)
```

```
                   mpg01_test
auto_lda_pred_class  0  1
                  0 40  1
                  1 12 45
```

We can calculate the error rate:

```
  mean(auto_lda_pred_class!=mpg01_test)
```

```
[1] 0.1326531
```

e)

```
  auto_qda_fit <- qda(
      mpg01 ~ displacement + horsepower + weight + acceleration + year,
    data = auto_train_data
  )
  auto_qda_fit
```

```
Call:
qda(mpg01 ~ displacement + horsepower + weight + acceleration +
    year, data = auto_train_data)

Prior probabilities of groups:
        0         1
0.4897959 0.5102041

Group means:
  displacement horsepower   weight acceleration     year
0     284.0833  134.02083 3678.174     14.33403 74.46528
1     118.3033   79.73333 2343.753     16.41600 77.67333
```

```
  auto_qda_pred_class <- predict(auto_qda_fit, auto_test_data)$class
  table(auto_qda_pred_class, mpg01_test)
```

```
                   mpg01_test
auto_qda_pred_class  0  1
                  0 38  4
                  1 14 42
```

We can calculate the error rate:

```
mean(auto_qda_pred_class!=mpg01_test)
```

```
[1] 0.1836735
```

f)

```
auto_logistic_fits <- glm(
  mpg01 ~ cylinders + displacement + horsepower + weight + acceleration + year,
  family = binomial,
  data = auto_train_data
)
summary(auto_logistic_fits)
```

```
Call:
glm(formula = mpg01 ~ cylinders + displacement + horsepower +
    weight + acceleration + year, family = binomial, data = auto_train_data)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.567e+01  8.088e+00  -1.938  0.05262 .
cylinders4    5.153e+00  1.734e+00   2.972  0.00296 **
cylinders5    1.695e+01  1.455e+03   0.012  0.99071
cylinders6    3.277e+00  2.024e+00   1.619  0.10544
cylinders8    7.950e+00  3.127e+00   2.543  0.01100 *
displacement -1.048e-02  1.161e-02  -0.903  0.36668
horsepower   -7.678e-02  3.446e-02  -2.228  0.02589 *
weight       -3.314e-03  1.475e-03  -2.247  0.02463 *
acceleration -3.212e-01  2.140e-01  -1.501  0.13342
year          4.609e-01  1.024e-01   4.499 6.81e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 407.448  on 293  degrees of freedom
Residual deviance:  88.773  on 284  degrees of freedom
AIC: 108.77
```

```
Number of Fisher Scoring iterations: 14
```

```
auto_logistic_probs <- predict(auto_logistic_fits, auto_test_data, type = "response")
```

```
auto_logistic_pred <- rep(FALSE, 98)
auto_logistic_pred[auto_logistic_probs > .5] <- TRUE
table(auto_logistic_pred, mpg01_test)
```

```
                mpg01_test
auto_logistic_pred  0  1
             FALSE 41  1
             TRUE  11 45
```

We can calculate the error rate:

```
mean(auto_logistic_pred != mpg01_test)
```

```
[1] 1
```

```
require(e1071)
```

```
Warning: package 'e1071' was built under R version 4.3.2
```

```
auto_nb_fit <- naiveBayes(
  mpg01 ~ cylinders + displacement + horsepower + weight + acceleration + year,
  data = auto_train_data
)
auto_nb_fit
```

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
```

```
        0         1
0.4897959 0.5102041

Conditional probabilities:
   cylinders
Y             3           4           5           6           8
  0 0.013888889 0.062500000 0.000000000 0.361111111 0.562500000
  1 0.006666667 0.900000000 0.006666667 0.066666667 0.020000000

   displacement
Y       [,1]     [,2]
  0 284.0833 86.38655
  1 118.3033 41.24981

   horsepower
Y        [,1]     [,2]
  0 134.02083 37.62931
  1  79.73333 16.88916

   weight
Y       [,1]     [,2]
  0 3678.174 651.6271
  1 2343.753 390.9530

   acceleration
Y       [,1]     [,2]
  0 14.33403 2.682321
  1 16.41600 2.599641

   year
Y       [,1]     [,2]
  0 74.46528 3.113052
  1 77.67333 3.724572
```

```
auto_nb_pred <- predict(auto_nb_fit, auto_test_data)
table(auto_nb_pred, mpg01_test)
```

```
            mpg01_test
auto_nb_pred  0  1
           0 38  0
           1 14 46
```

We can calculate the error rate below:

```
mean(auto_nb_pred != mpg01_test)
```

```
[1] 0.1428571
```

h)

```
library(class)
auto_train_x = auto_train_data |>
  dplyr::select(cylinders, displacement, horsepower, weight, acceleration, year)

auto_test_x = auto_test_data |>
  dplyr::select(cylinders, displacement, horsepower, weight, acceleration, year)

auto_train_mpg01 = auto_train_data |>
  dplyr::select(mpg01)
```

```
auto_knn_pred_1 <- knn(train = auto_train_x,
                       test = auto_test_x,
                       cl = auto_train_data$mpg01,
                       k = 1)
table(auto_knn_pred_1, mpg01_test)
```

```
              mpg01_test
auto_knn_pred_1  0  1
              0 40  5
              1 12 41
```

We can calculate error:

```
mean(auto_knn_pred_1 != mpg01_test)
```

```
[1] 0.1734694
```

Lets try again with k = 5:

```r
auto_knn_pred_5 <- knn(train = auto_train_x,
                       test = auto_test_x,
                       cl = auto_train_data$mpg01,
                       k = 5)
table(auto_knn_pred_5, mpg01_test)
```

```
              mpg01_test
auto_knn_pred_5  0  1
              0 41  5
              1 11 41
```

We can calculate error:

```r
mean(auto_knn_pred_5 != mpg01_test)
```

```
[1] 0.1632653
```

Lets try again with k = 10:

```r
auto_knn_pred_10 <- knn(train = auto_train_x,
                       test = auto_test_x,
                       cl = auto_train_data$mpg01,
                       k = 10)
table(auto_knn_pred_10, mpg01_test)
```

```
               mpg01_test
auto_knn_pred_10  0  1
               0 39  4
               1 13 42
```

We can calculate error:

```r
mean(auto_knn_pred_10 != mpg01_test)
```

```
[1] 0.1734694
```

Now lets try with an extremely large k, k = 50.

```
auto_knn_pred_50 <- knn(train = auto_train_x,
                        test = auto_test_x,
                        cl = auto_train_data$mpg01,
                        k = 50)
table(auto_knn_pred_50, mpg01_test)
```

```
                mpg01_test
auto_knn_pred_50  0  1
               0 37  3
               1 15 43
```

We can calculate error:

```
mean(auto_knn_pred_50 != mpg01_test)
```

```
[1] 0.1836735
```

From the four K's we tested K = 5 appears to have done the best. There seems to be a sweet-spot between low and high values of k that allow for the best performance of the KNN model. This tracks with our previous understanding of the trade-offs of bias and variance, the point where both are minimized with likely not be with low ks or extremely high ks.

5)

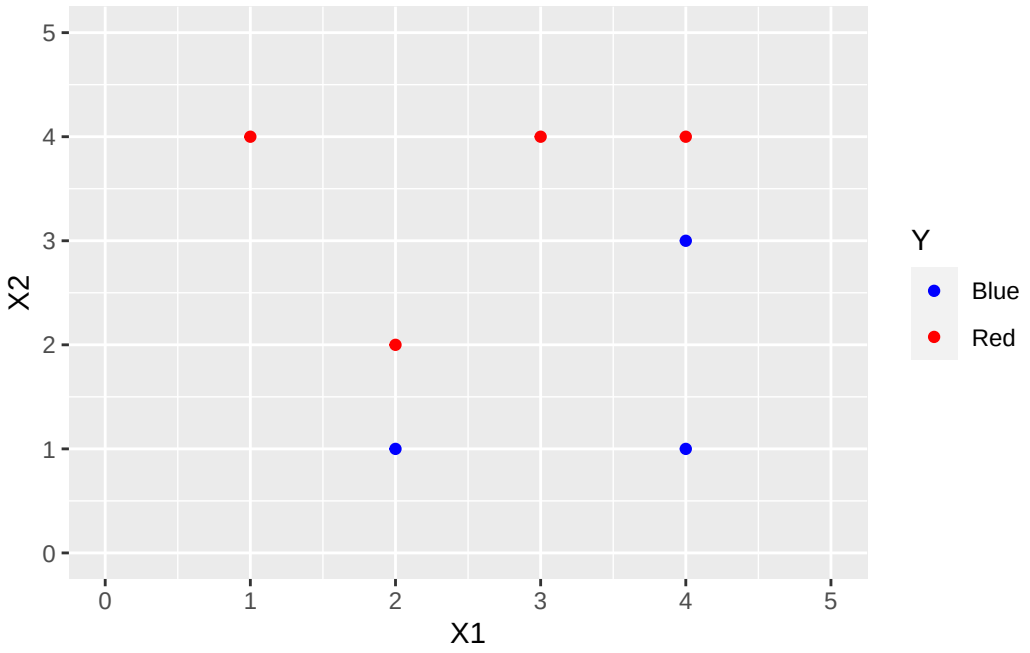
```
toy_svm_df <- tribble(
  ~Obs, ~X1, ~X2, ~Y,
  1,     3,   4,  "Red",
  2,     2,   2,  "Red",
  3,     4,   4,  "Red",
  4,     1,   4,  "Red",
  5,     2,   1,  "Blue",
  6,     4,   3,  "Blue",
  7,     4,   1,  "Blue"

)

toy_svm_df <- toy_svm_df |>
  mutate(Y = as.factor(Y))
```

a)

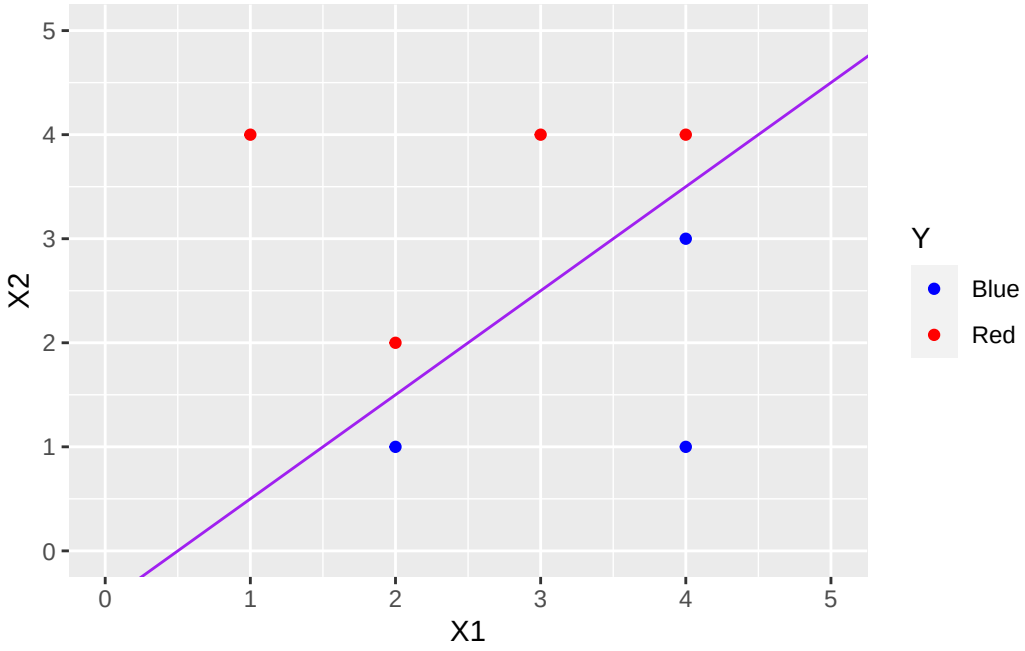
```r
observations <- toy_svm_df |>
  ggplot(aes(x = X1, y = X2, color = Y)) +
  geom_point() +
  scale_color_manual(values = c("Red" = "red", "Blue" = "blue")) +
  scale_x_continuous(limits = c(0, 5)) +
  scale_y_continuous(limits = c(0, 5))
observations
```



b)

We can find the optimal line by inspection, using the closest two points at $(2, 1), (2, 2)$ and $(4, 3), (4, 4)$ respectively.

```r
optimal_seper_classifier <- observations +
  geom_abline(intercept = -.5, slope  = 1, col = "purple")
optimal_seper_classifier
```

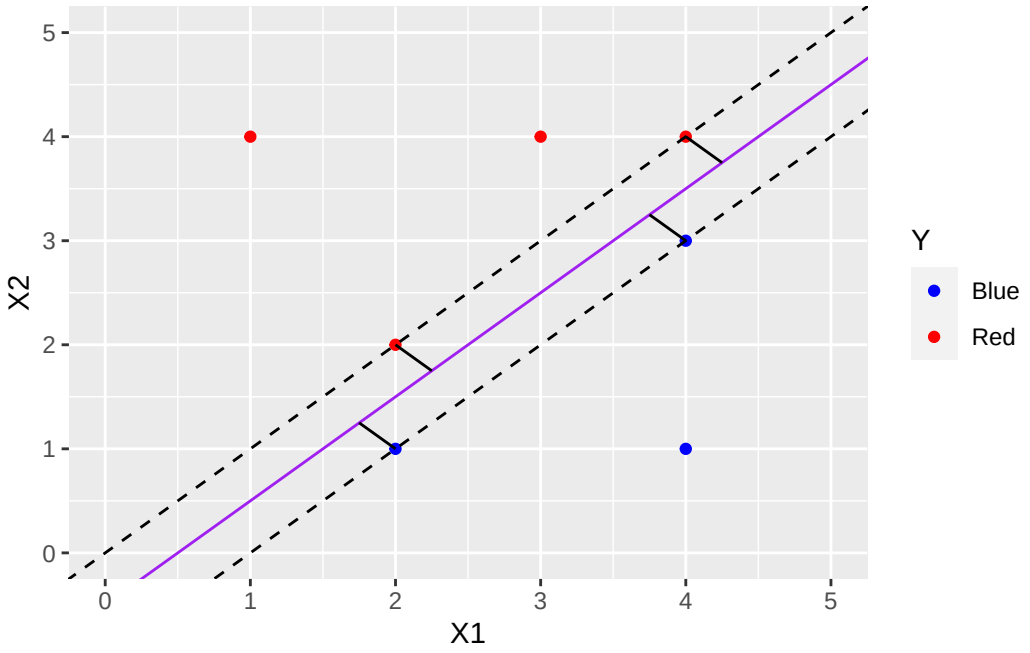


c)

The equation of our optimal separating hyper-plane is $-.5+1X_1+1X_2 = 0$. So our classification rule is red if $-.5 + 1X_1 + 1X_2 > 0$ and blue if $-.5 + 1X_1 + 1X_2 < 0$.

d)

```
optimal_seper_classifier +
  annotate("segment", x = c(2,2,4,4), xend = c(1.75,2.25,3.75,4.25),
           y = c(1,2,3,4), yend = c(1.25,1.75,3.25,3.75)) +
  geom_abline(intercept = c(-1,0), slope = c(1,1), linetype = 2)
```



e)

The support vectors are $(2,1),(2,2),(4,3),(4,4)$.

f)

The seventh observation is not a support vector, unless it is moved into the margin it will not change the maximal margin hyper-plane.