Ben Howell

Ayanna Fisher

Matthew Bradley

Hayley Zorkic

SDS 323 – HW 3

Ben Howell

Question 14 (A, B, C, F, H)

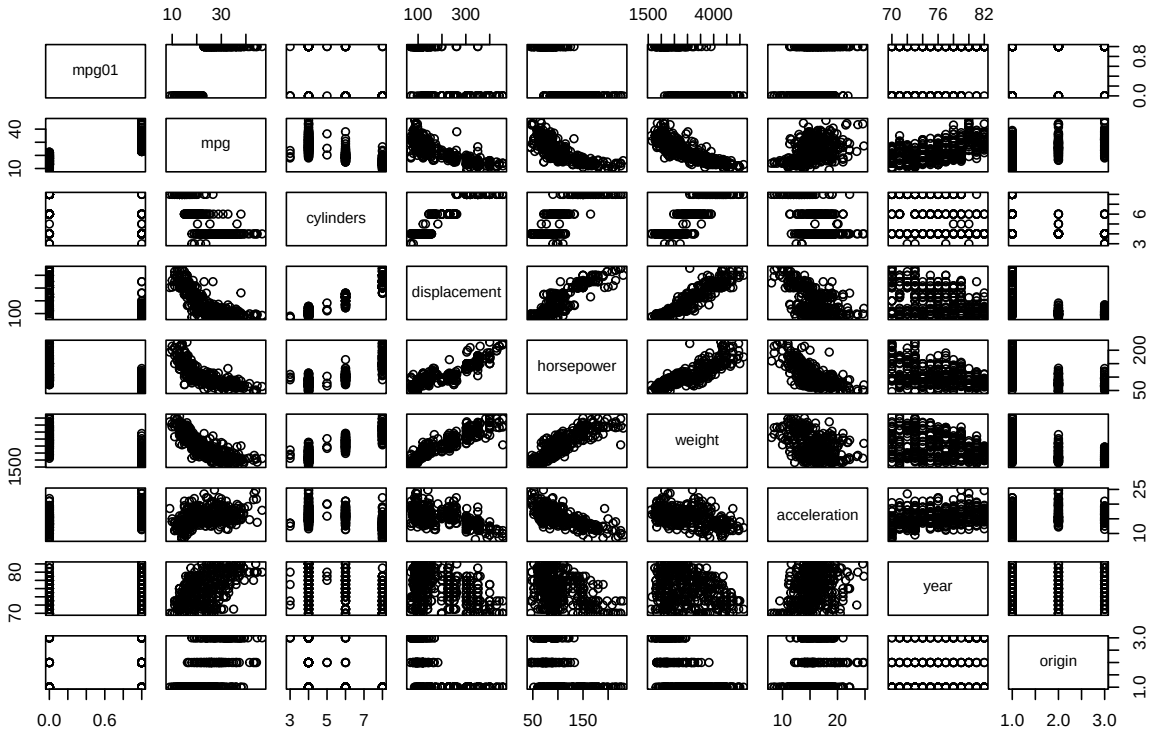
```
require(tidyverse)
require(ISLR2)
```

```
head(Auto)
```

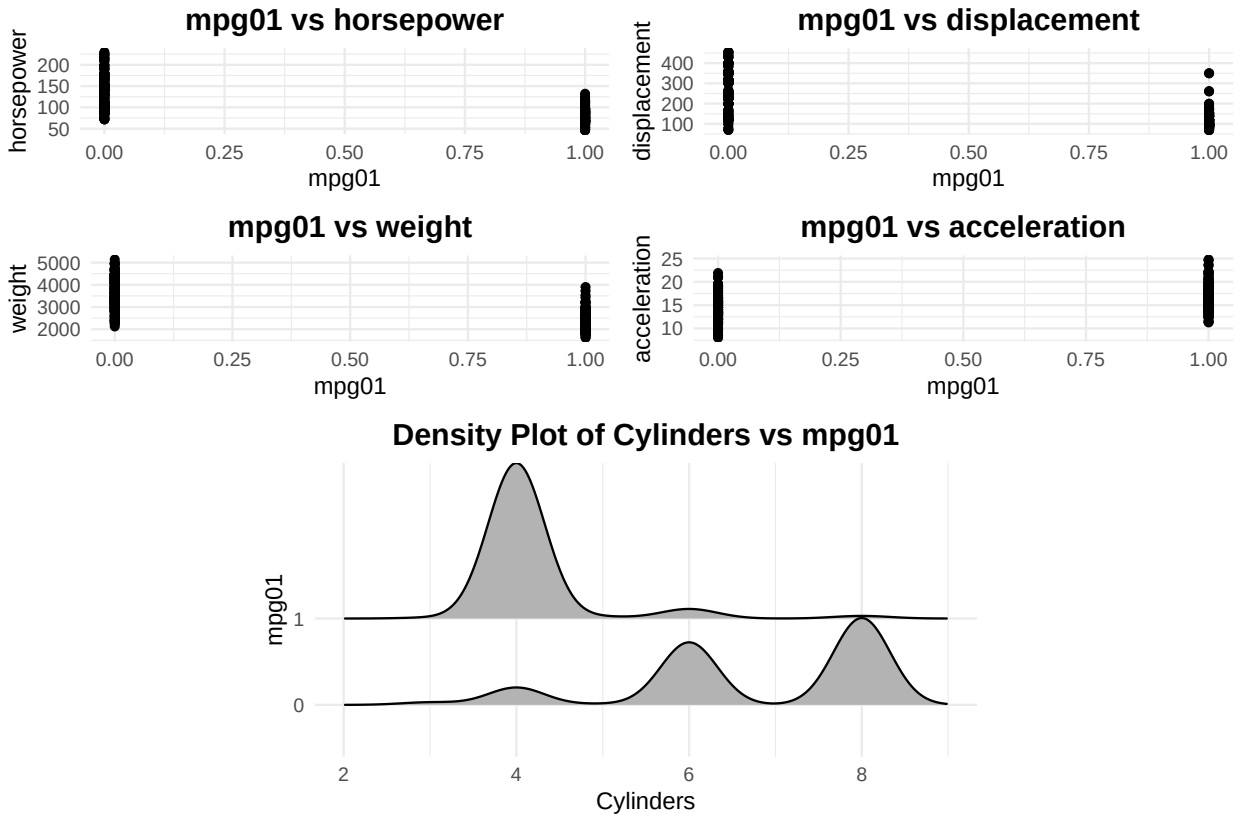
```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                       name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

```
df <- Auto %>%
  dplyr::mutate(mpg01 = ifelse(mpg >= median(Auto$mpg, na.rm = TRUE), 1, 0))

# hist(df$mpg01)
```

```
pairs(df %>%
        dplyr::select(mpg01, mpg, cylinders, displacement,
                      horsepower, weight, acceleration, year, origin))
```



Well, clearly the best predictor of `mpg01` is going to be `mpg`, but it doesn't make sense to include `mpg` as a variable in a model attempting to predict `mpg01`. Beyond that, it looks like horsepower, weight, displacement, and acceleration may have some impact on `mpg01`.



We can see a semblance of linear relationships between `mpg01` and the top four variables. When we compare `mpg01` to `cylinders`, we see that there appears to be a heavy concentration of high `mpg` vehicles with only 4 cylinders, whereas low `mpg` vehicles (a `mpg01` $= 0$) appear to be concentrated at 6 or 8 cylinders.

Before we run the model, we first split the data into training and testing datasets using the `caret` library.

```
set.seed(123)
ti <- caret::createDataPartition(df$mpg01, p = 0.75, list = FALSE)

train_data = df[ti, ]
test_data = df[-ti, ]
```

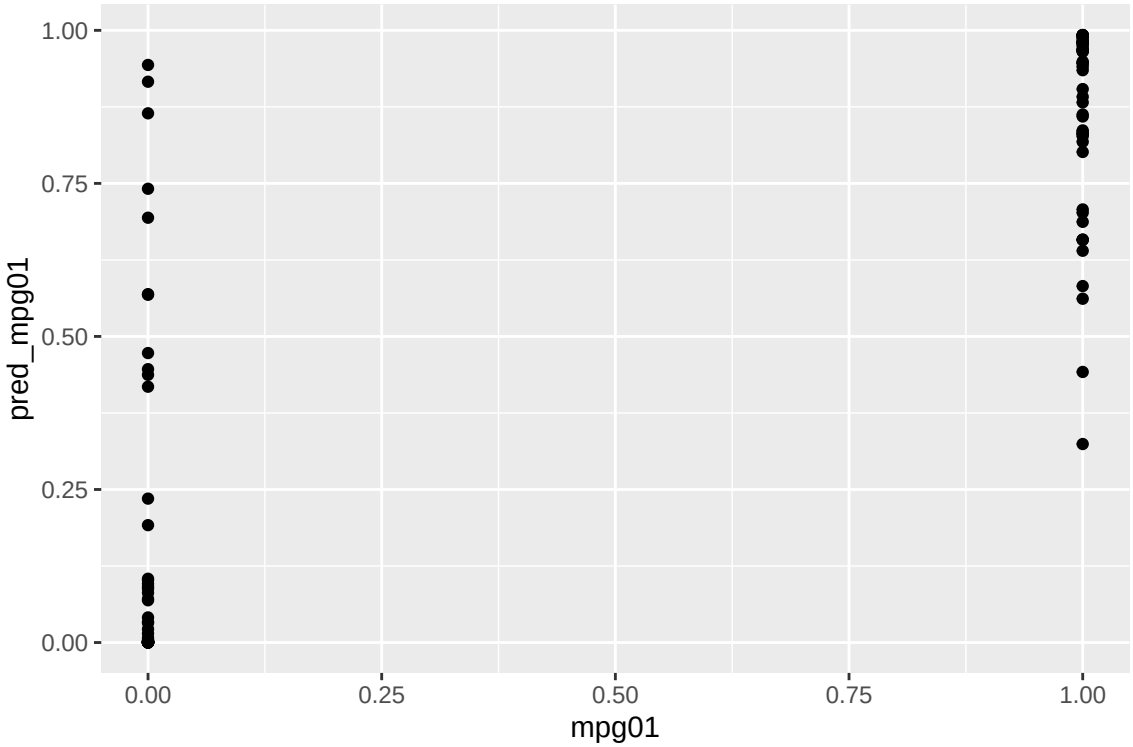
```
log_model <- glm(mpg01 ~ cylinders + displacement + horsepower + weight + acceleration,
                 data = train_data, family = "binomial")
summary(log_model)
```

```
##
## Call:
## glm(formula = mpg01 ~ cylinders + displacement + horsepower +
##     weight + acceleration, family = "binomial", data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1515  -0.2190   0.0473   0.3623   3.3259
##
## Coefficients:
```

```
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  11.097855   3.038713   3.652  0.00026 ***
## cylinders     0.117531   0.386559   0.304  0.76109
## displacement -0.015151   0.009201  -1.647  0.09963 .
## horsepower   -0.042977   0.022694  -1.894  0.05826 .
## weight       -0.001817   0.001018  -1.786  0.07415 .
## acceleration  0.011349   0.142132   0.080  0.93636
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 407.57  on 293  degrees of freedom
## Residual deviance: 160.11  on 288  degrees of freedom
## AIC: 172.11
##
## Number of Fisher Scoring iterations: 7
```

With our data split into training and testing data sets, we see that `displacement` and `horsepower` are significant variables (using a 5% level of significance). Variables like `weight` and `cylinders` provide some value, but are not statistically significant at a reasonable level, and `acceleration` provides no value.

```
test_data$pred_mpg01 <- predict(log_model, newdata = test_data, type = "response")

test_data %>%
  ggplot() +
  geom_point(aes(x = mpg01, y = pred_mpg01))
```



```
test_data <- test_data %>%
  dplyr::mutate(pred_binary = ifelse(pred_mpg01 >= 0.5, 1, 0))

acc <- mean(test_data$mpg01 == test_data$pred_binary)
table(test_data$mpg01, test_data$pred_binary)
```

```
##
##      0  1
##   0 42  7
##   1  2 47
```

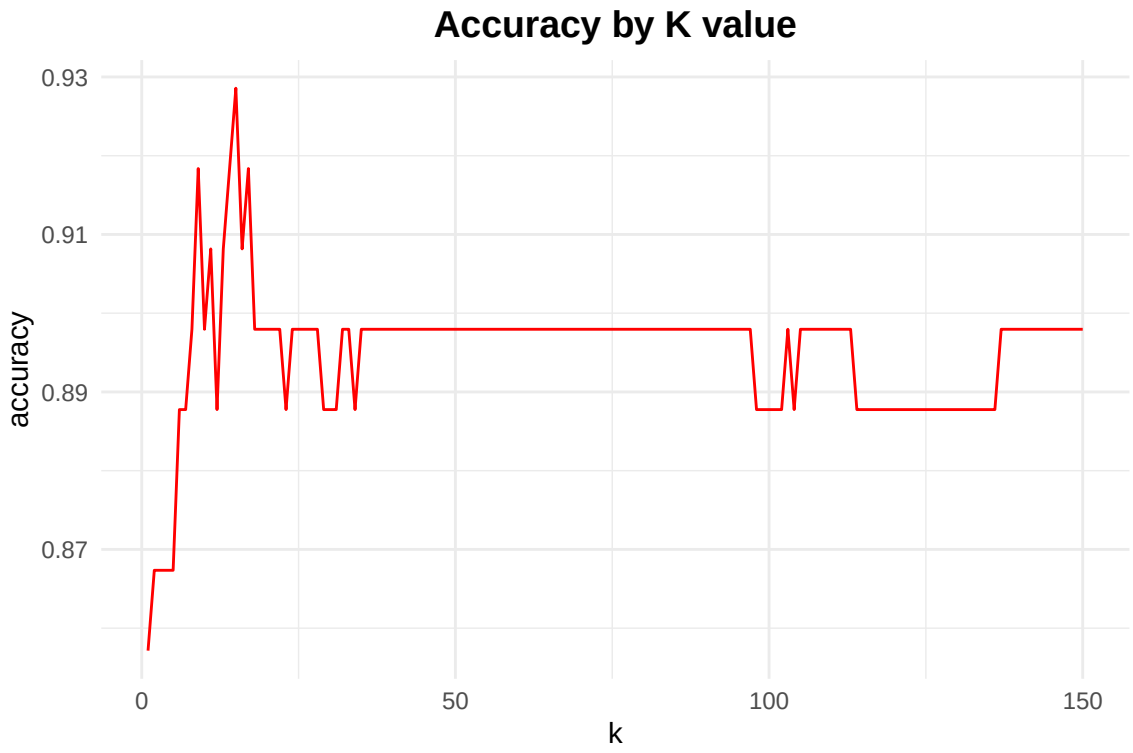
By running a confusion matrix on the predicted `mpg01` probabilities, we can see that 90.8 of the predictions were correct. The greatest error occurs when the `mpg01` $= 0$ in reality, but was predicted to be greater than 1.

```
require(FNN)
```

```
## Loading required package: FNN
```

```
set.seed(123)

mpg_train <- train_data %>%
  dplyr::select(cylinders, displacement, horsepower, weight, acceleration)
mpg_test <- test_data %>%
  dplyr::select(cylinders, displacement, horsepower, weight, acceleration)

train_label <- train_data %>%
  dplyr::select(mpg01)
test_label <- test_data %>%
  dplyr::select(mpg01)

knn_acc <- list()

for (z in 1:150) {
  knn_model <- knn(train = mpg_train, test = mpg_test, cl = train_data$mpg01, k = z)
  knn_table <- table(knn_model, test_data$mpg01)
  k_a <- sum(diag(knn_table))/ sum(knn_table)
  k_df <- data.frame(k_a, z)
  colnames(k_df) <- c("accuracy", "k")

  knn_acc[[z]] <- k_df

  # print(z)
}

knn_acc <- dplyr::bind_rows(knn_acc) %>%
  # dplyr::mutate(order = row_number()) %>%
  arrange(desc(accuracy))

knn_acc %>%
  ggplot() +
  geom_line(aes(x = k, y = accuracy), col = "red") +
```

```
  labs(title = "Accuracy by K value") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14))
```




```
n <- floor(min(knn_acc$accuracy, na.rm = TRUE) * 100)
l <- ceiling(max(knn_acc$accuracy, na.rm = TRUE) * 100)

m <- knn_acc %>%
  dplyr::filter(accuracy == max(accuracy, na.rm = TRUE)) %>%
  dplyr::filter(k == max(k)) %>%
  pull(k)
```

We see that the accuracy values are between 85% and 93%. The highest accuracy rate of 92.86% is achieved at K = 15. The accuracy of 92.86% beats the logistic regression's accuracy of 90.8% by 2.06% points.

HW3_Q2

Q15 in Section 4.8 of ISLR

##15a. Write a function, Power(), that prints out the result of raising 2 ##to the 3rd power. Use the print() function to output the result.

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

print(Power())
```

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

##15b. Create a new function, Power2(), that allows you to pass any ##two numbers, x and a, and prints out the value of x^a

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

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

##15c. Using the Power2() function that you just wrote, compute 103, ##817, and 1313

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

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

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

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

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

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

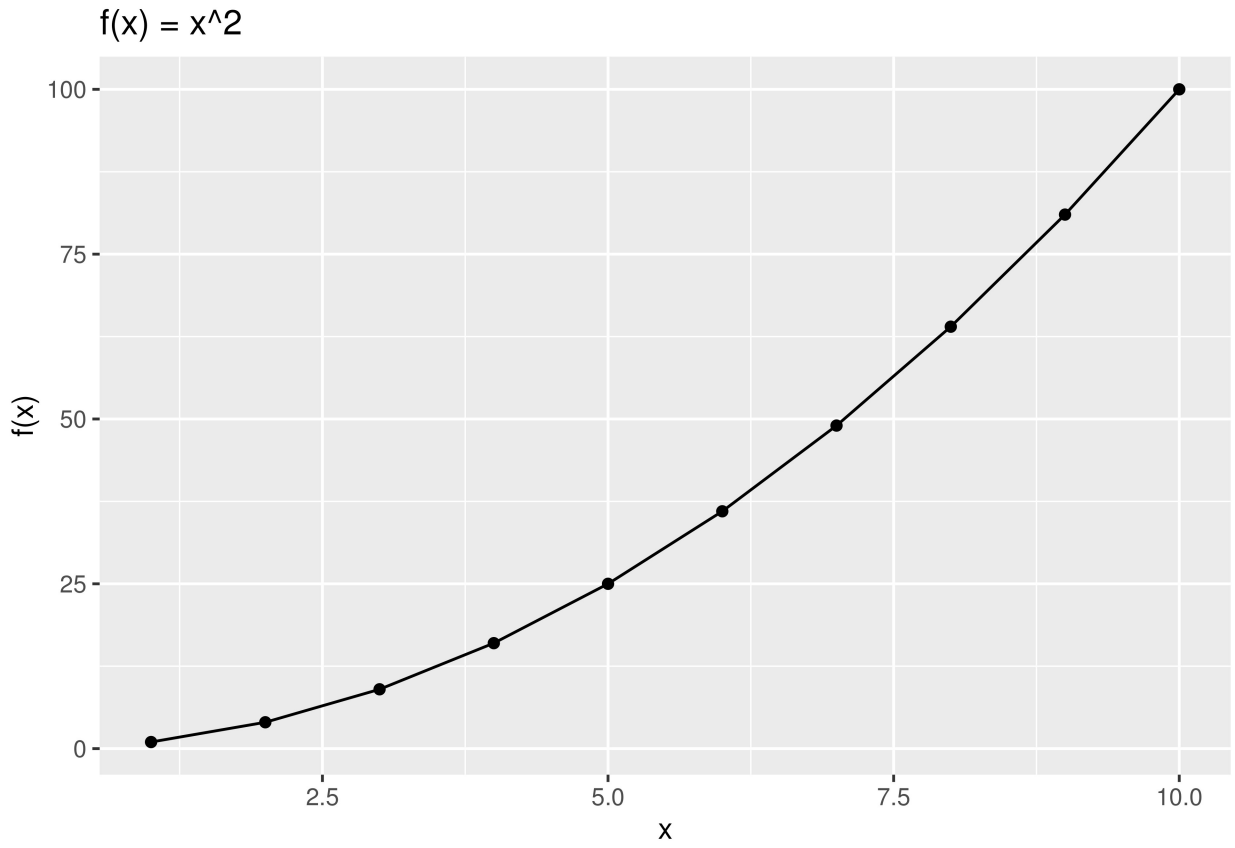
##15d. Now create a new function, Power3(), that actually returns the ##result x^a as an R object

```
Power3 <- function(x, a){
  res <- x^a
  return(res)
}
Power3(3,8)
```

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

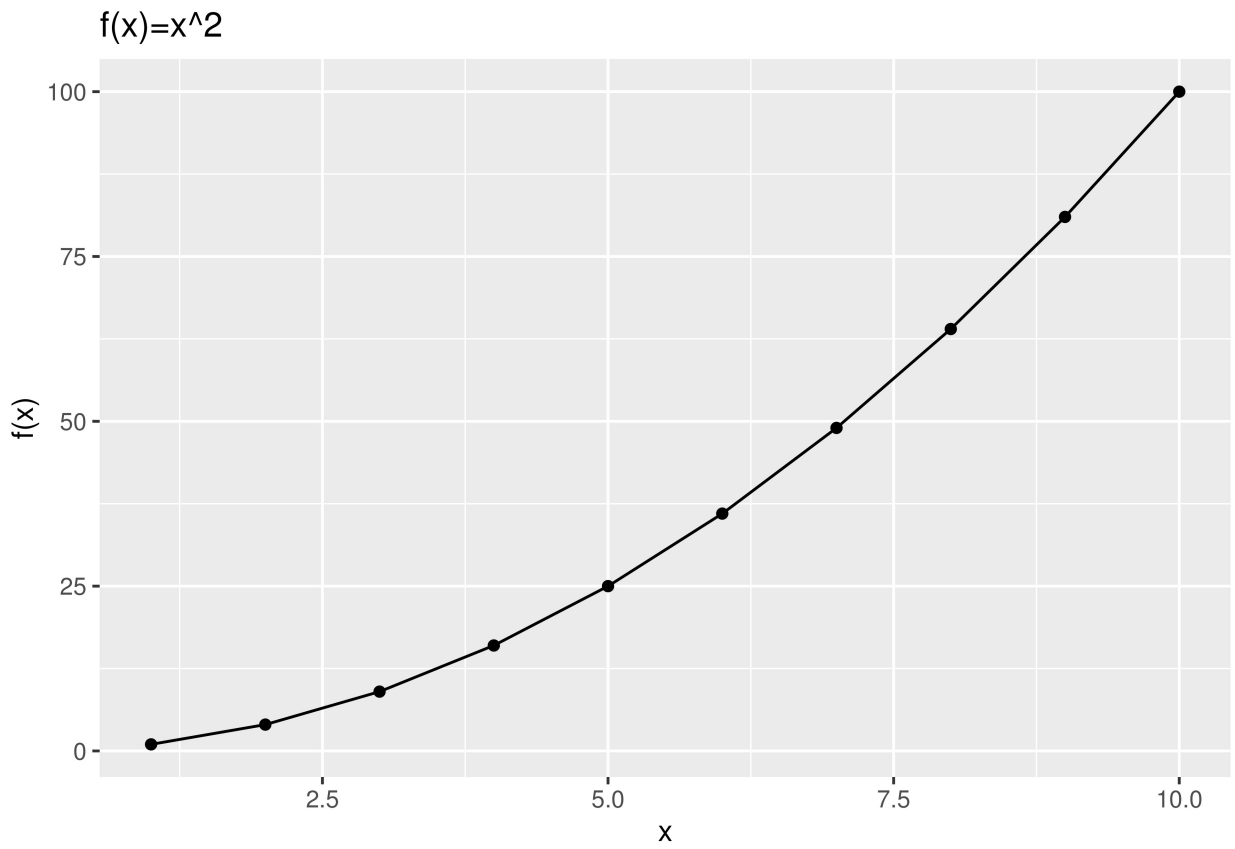
##15e. using the Power3() function, create a plot of f(x) = x2. ##The x-axis should display a range of integers from 1 to 10, and ##the y-axis should display x2. Label, title, consider log-scale

```
x <-  1:10
ggplot(data.frame(x=x, y = Power3(x, 2)), aes(x = x, y = y)) +
  geom_line() + geom_point() + xlab("x") + ylab("f(x)") +
  ggtitle("f(x) = x^2")
```



##15f. Create a function, PlotPower(), that allows you to create a plot ##of x against x^a for a fixed a and for a range of values of x

```
PlotPower <- function(x,a){
ggplot(data.frame(x=x, y = Power3(x, 2)), aes(x = x, y = y)) +
  geom_line() + geom_point() + labs(x = "x", y = "f(x)",
                                    title = "f(x)=x^2")
}
PlotPower(1:10, 3)
```


f(x)=x^2
100
75
50
25
0
f(x)
2.5
5.0
7.5
10.0
x

3. Question 16 in Section 4.8

(Only use logistic regression and KNN models for this question)

Using the Boston data set, fit classification models in order to predict whether a given census tract has a crime rate above or below the me- dian. Explore logistic regression, LDA, naive Bayes, and KNN models using various subsets of the predictors. Describe your findings. Hint: You will have to create the response variable yourself, using the variables that are contained in the Boston data set

Grab our data, one-hot encode the crime variable such that it tells us if its ABOVE or BELOW the average crime rate:

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.1.2
```

```
attach(Boston)
library(FNN)
```

```
## Warning: package 'FNN' was built under R version 4.1.2
```

```
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio lstat medv
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3  4.98 24.0
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8  9.14 21.6
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8  4.03 34.7
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7  2.94 33.4
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7  5.33 36.2
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7  5.21 28.7
```

```
set.seed(1)

# one-hot encode Crime Variable. 1=above median, 0=below median
Boston$crim <- factor(ifelse(Boston$crim > median(Boston$crim), 1, 0))
head(Boston)
```

```
##   crim zn indus chas   nox    rm  age    dis rad tax ptratio lstat medv
## 1    0 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3  4.98 24.0
## 2    0  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8  9.14 21.6
## 3    0  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8  4.03 34.7
## 4    0  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7  2.94 33.4
## 5    0  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7  5.33 36.2
## 6    0  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7  5.21 28.7
```

As you can see the 'crim' column changed from a series of continuous numbers into binary 0/1 based on their relationship to the median 'crim' value.

Split data into testing and training:

```
sample_size <- floor(0.75 * nrow(Boston))
subset <- sample(seq_len(nrow(Boston)), size = sample_size)
Boston_train <- Boston[subset, ] # get the training set
Boston_test <- Boston[-subset, ] # get the test set

print(paste0("Rows in boston: ", nrow(Boston)))
```

```
## [1] "Rows in boston: 506"
```

```
print(paste0("Rows in test set: ", nrow(Boston_test)))
```

```
## [1] "Rows in test set: 127"
```

```
print(paste0("Rows in training set: ", nrow(Boston_train)))
```

```
## [1] "Rows in training set: 379"
```

```
print(paste0("Rows in Training + Testing: ",sum(nrow(Boston_test),nrow(Boston_train))))
```

```
## [1] "Rows in Training + Testing: 506"
```

Now we have a training set where the training set is 75% of the data and the testing set is 25% of the data.

Logistic Regression

Set up a logistic regression model with all variables:

```
# Glm = generalized linear model, "family = binomial" means that we consider logistic regression
my_glm <- glm(crim~ . , data = Boston_train, family = "binomial")
summary(my_glm)
```

```
##
## Call:
## glm(formula = crim ~ ., family = "binomial", data = Boston_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1974  -0.1381  -0.0003   0.0030   3.6668
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -47.765298   7.762319  -6.153 7.58e-10 ***
## zn           -0.116529   0.046515  -2.505  0.01224 *
## indus        -0.084808   0.054927  -1.544  0.12259
## chas          0.111021   0.798530   0.139  0.88943
## nox          59.769363  10.154042   5.886 3.95e-09 ***
## rm           -0.312253   0.855574  -0.365  0.71514
## age           0.027033   0.014222   1.901  0.05733 .
## dis           1.234418   0.302872   4.076 4.59e-05 ***
```

```
## rad           0.646595   0.179666   3.599  0.00032 ***
## tax          -0.005976   0.003077  -1.942  0.05209 .
## ptratio       0.281197   0.146327   1.922  0.05464 .
## lstat         0.115201   0.056457   2.041  0.04130 *
## medv          0.195255   0.083889   2.328  0.01994 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 525.40  on 378  degrees of freedom
## Residual deviance: 151.01  on 366  degrees of freedom
## AIC: 177.01
##
## Number of Fisher Scoring iterations: 9
```

Compute the classification accuracy from training data:

```
# Build logistic regression model
predicted_glm_train <- predict(my_glm, Boston_train, type = "response")
yhat_predict_train <- ifelse(predicted_glm_train > 0.5, 1, 0)

# Create a table
table_Boston_train <- table(y = Boston_train$crim, yhat = yhat_predict_train)
table_Boston_train
```

```
##    yhat
## y     0   1
##   0 178  12
##   1  16 173
```

```
# Classification accuracy
accuracy_Boston_train <- sum(diag(table_Boston_train))/ sum(table_Boston_train)
accuracy_Boston_train
```

```
## [1] 0.9261214
```

The training Accuracy is 93%.

Compute the classification accuracy from the testing data:

```
# Compute the classification accuracy from test data
predicted_glm_test <- predict(my_glm, Boston_test, type = "response")
yhat_predict_test <- ifelse(predicted_glm_test > 0.5, 1, 0)

table_Boston_test <- table(y = Boston_test$crim, yhat = yhat_predict_test)
table_Boston_test
```

```
##    yhat
## y    0  1
##   0 51 12
##   1  5 59
```

```r
# Classification accuracy from test data
accuracy_Boston_test <- sum(diag(table_Boston_test))/ sum(table_Boston_test)
accuracy_Boston_test
```

```
## [1] 0.8661417
```

The testing Accuracy is 87%.

K nearest Neighbors

Omit NAs from the data and re-split:

```r
# Remove missing data from data
Boston <- na.omit(Boston)

sample_size <- floor(0.75 * nrow(Boston))
subset <- sample(seq_len(nrow(Boston)), size = sample_size)
Boston_train <- Boston[subset, ] # get the training set
Boston_test <- Boston[-subset, ] # get the test set

print(paste0("Rows in boston: ", nrow(Boston)))
```

```
## [1] "Rows in boston: 506"
```

```r
print(paste0("Rows in test set: ", nrow(Boston_test)))
```

```
## [1] "Rows in test set: 127"
```

```r
print(paste0("Rows in training set: ", nrow(Boston_train)))
```

```
## [1] "Rows in training set: 379"
```

```r
print(paste0("Rows in Training + Testing: ",sum(nrow(Boston_test),nrow(Boston_train))))
```

```
## [1] "Rows in Training + Testing: 506"
```

Fit a KNN Classifier with K=1

```r
# We first try with K = 1
my_knn <- knn(Boston_train[, -1],
              Boston_test[,-1],
              Boston_train$crim,
              k = 1)
table_knn <- table(my_knn, Boston_test$crim)

accuracy_Boston_test <- sum(diag(table_knn))/ sum(table_knn)
accuracy_Boston_test
```

```
## [1] 0.9448819
```

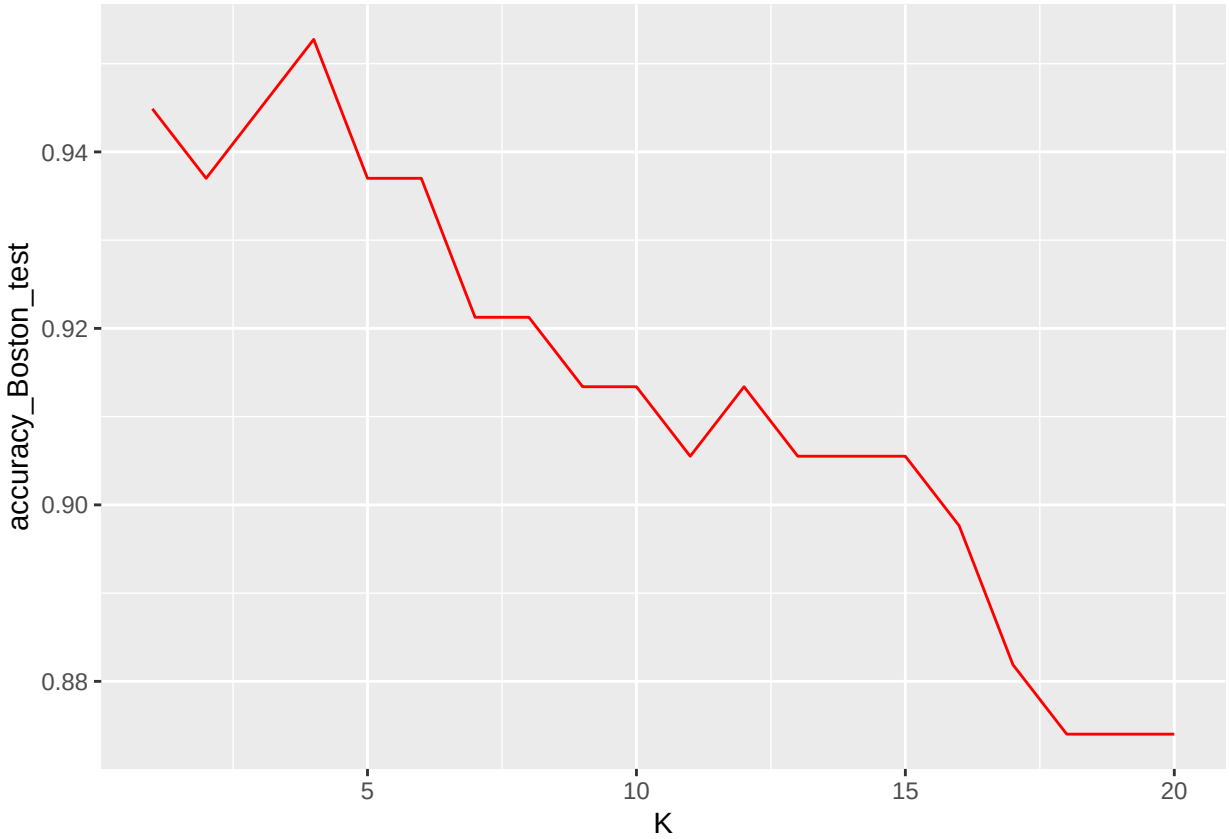
Our test accuracy is 90%.

Now lets see which K will give us the highest testing accuracy:

```
# We plot the accuracy for different values of K
M = 20 # the number of possible values of K
K <- seq(1, M)

accuracy_Boston_test <- rep(0,M)

for (i in 1:M)
{
  my_knn <- knn(Boston_train[,-1], Boston_test[,-1], Boston_train$crim, k = K[i])
  table_knn <- table(my_knn, Boston_test$crim)

  accuracy_Boston_test[i] <- sum(diag(table_knn))/ sum(table_knn)
}
max(accuracy_Boston_test)
```

```
## [1] 0.9527559
```

```
accuracy_Boston_test <- as.data.frame(accuracy_Boston_test)

ggplot(accuracy_Boston_test,
       aes(x = K,
           y = accuracy_Boston_test)) +
      geom_line(col = "red")
```



The best value for K is around 1 or 2!

```
accuracy_Boston_test[1,]
```

```
## [1] 0.9448819
```

```
accuracy_Boston_test[2,]
```

```
## [1] 0.9370079
```

The test accuracy of K=1 and K=2 is 90%.

Bonus Questions

Question 4:

a: On average, we will use 10% ($\frac{1}{10}$) of available observations to make the prediction. This is because X is uniformly distributed, so in any given range of values such as the range [0.55,0.65], that makes up 10% of the values, we will most likely see 10% of the available observations.

b: On average, we will use 1% ($\frac{1}{100}$) of available observations to make the prediction. The logic follows from part a, because X1 and X2 are uniformly distributed, but now, on average, we will only use 10% of X1 observations and 10% of X2 observations. Because we are looking for observations that are within 10% of X1 and 10% of X2, we can multiply X1 and X2 to find the probability of a point being in both of these ranges. \$0.1*0.1 = 0.01 = \$1%.

c: This follows the same logic as part b, but in part b we used 0.1^2, and here we will use 0.1^{100} (because p =100, and we want observations within 10% of all 100 of these features). So our likliehood is 0.1^{100} or $\frac{1}{10^{100}}$.

d: As we can see in part A-C, as p increases, the amount of observations used decreases exponentially, so that when p gets very large, there are an extremely small fraction of observations that are close. This will lead to nearest neighbors that are not actually near each other because if we define being near eachother as being within 10% of eachother in each dimension, we will have hardly any neighbors when we look at all of the dimensions together.

e: For p=1, the hypercube would need to cover 10% of the length to capture 10% of the data points, so its length would be 0.1. For p = 2, the hypercube (square) area would need to cover 10% of the area, so $length * width = 0.1 = length^2$, so the side lengths would need to be $0.1^{\frac{1}{2}} = 0.316$. For a p = 100 hypercube, its volume would be found by multiplying all 100 sides, and setting this equal to .1, to find the side lengths when volume covers 10% of observation. In this case, $0.1 = length^{100}$. So $0.1^{1/100} = 0.977$ = length of each side. As we can see, the length needed to cover 10% of the data increases as p increases, showing that the "nearest neighbors" at high p values are not actually very near to each other, when compared to low p values.

Question 12:

a:

p = $\frac{exp(B_0+B_1x)}{1+exp(B_0+B_1x)}$

odds = $\frac{p}{1-p}$

1 - p = $1 - \frac{exp(B_0+B_1x)}{1+exp(B_0+B_1x)}$

$\frac{p}{1-p}$ simplifies to $exp(B_0 + B_1x)$

Answer:

$ln(exp(B_0 + B_1x)) = B_0 + B_1x$

b:

p = $\frac{exp(\alpha_{orange0}+\alpha_{orange1}x)}{exp(\alpha_{orange0}+\alpha_{orange1}x)+exp(\alpha_{apple0}+\alpha_{apple1}x)}$

1 - p = $1 - \frac{exp(\alpha_{orange0}+\alpha_{orange1}x)}{exp(\alpha_{orange0}+\alpha_{orange1}x)+exp(\alpha_{apple0}+\alpha_{apple1}x)}$

$odds = \frac{p}{1-p} = \frac{exp(\alpha_{orange0}+\alpha_{orange1}x)}{exp(\alpha_{apple0})+\alpha_{apple1}x}$

Answer:

ln(odds) = $\frac{\alpha_{orange0}+\alpha_{orange1}x}{\alpha_{apple0}+\alpha_{applee1}x}$

c:

We do not know the exact coefficient estimates but we can set

$\frac{\alpha_{orange0}}{\alpha_{apple0}} = B_0 = 2$

and

$\frac{\alpha_{orange1}}{\alpha_{apple1}} = B_1 =$ -1

d:

$B_0 = \frac{\alpha_{orange0}}{\alpha_{apple0}} = \frac{1.2}{3} = 0.4$

$B_1 = \frac{\alpha_{orange1}}{\alpha_{apple1}} = \frac{-2}{.6} =$ -3.333

e:

I expect them to agree 100% of the time. Softmax is used for multiclass data, whereas logistic regression is used for binary data Because this example has only two classes (it is binary), the softmax and logistic approaches should return the same predicted labels (this is why we can even predict one model's coefficients from the other model as seen above).