Exercise 2

KNN Practice

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.2

## -- Attaching packages -------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v stringr 1.4.0
## v tidyr   1.0.2     v forcats 0.4.0
## v readr   1.3.1

## Warning: package 'ggplot2' was built under R version 3.5.2

## Warning: package 'tibble' was built under R version 3.5.2

## Warning: package 'tidyr' was built under R version 3.5.2

## Warning: package 'purrr' was built under R version 3.5.2

## Warning: package 'stringr' was built under R version 3.5.2

## Warning: package 'forcats' was built under R version 3.5.2

## -- Conflicts ----------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(FNN)
```

```
## Warning: package 'FNN' was built under R version 3.5.2
```

```
sclass = read.csv("sclass.csv", header = T)

#split into two data sets: trim = 350 & 65 AMG
sclass_350 = sclass %>%
  filter(trim == 350)

sclass_65 <- sclass %>%
  filter(trim == "65 AMG")
```

```
#350 group
set.seed(126)
```

```r
#create train and test set sizes
N350 = nrow(sclass_350)
N_train350 = floor(0.8*N350)
N_test350 = N350-N_train350

#create train set by sampling then use rest of data for test set
train_ind350 = sort(sample.int(N350, N_train350, replace = FALSE))

S_train350 = sclass_350[train_ind350,]
S_test350 = sclass_350[-train_ind350,]

X_train350 = select(S_train350, mileage)
Y_train350 = select(S_train350, price)
X_test350 = select(S_test350, mileage)
Y_test350 = select(S_test350, price)

#use different values of K to run KNN
knn5_350 = knn.reg(train = X_train350, test = X_test350, y = Y_train350, k=5)
knn10_350 = knn.reg(train = X_train350, test = X_test350, y = Y_train350, k=10)
knn15_350 = knn.reg(train = X_train350, test = X_test350, y = Y_train350, k=15)
knn20_350 = knn.reg(train = X_train350, test = X_test350, y = Y_train350, k=20)
knn50_350 = knn.reg(train = X_train350, test = X_test350, y = Y_train350, k=50)
knn75_350 = knn.reg(train = X_train350, test = X_test350, y = Y_train350, k=75)
knn100_350 = knn.reg(train = X_train350, test = X_test350, y = Y_train350, k=100)

#define function to calculate rmse
rmse = function(y, ypred) {
  sqrt(mean(data.matrix(y-ypred)^2))
}

ypred_knn5_350 = knn5_350$pred
ypred_knn10_350 = knn10_350$pred
ypred_knn15_350 = knn15_350$pred
ypred_knn20_350 = knn20_350$pred
ypred_knn50_350 = knn50_350$pred
ypred_knn75_350 = knn75_350$pred
ypred_knn100_350 = knn100_350$pred

rmse(Y_test350, ypred_knn5_350)
```

```
## [1] 10745.33
```

```r
rmse(Y_test350, ypred_knn10_350)
```

```
## [1] 10716.15
```

```r
rmse(Y_test350, ypred_knn15_350)
```

```
## [1] 10660.44
```

```r
rmse(Y_test350, ypred_knn20_350)
```

```
## [1] 10951.32
```

```r
rmse(Y_test350, ypred_knn50_350)
```

```
## [1] 11899.03
```

```r
rmse(Y_test350, ypred_knn75_350)
```

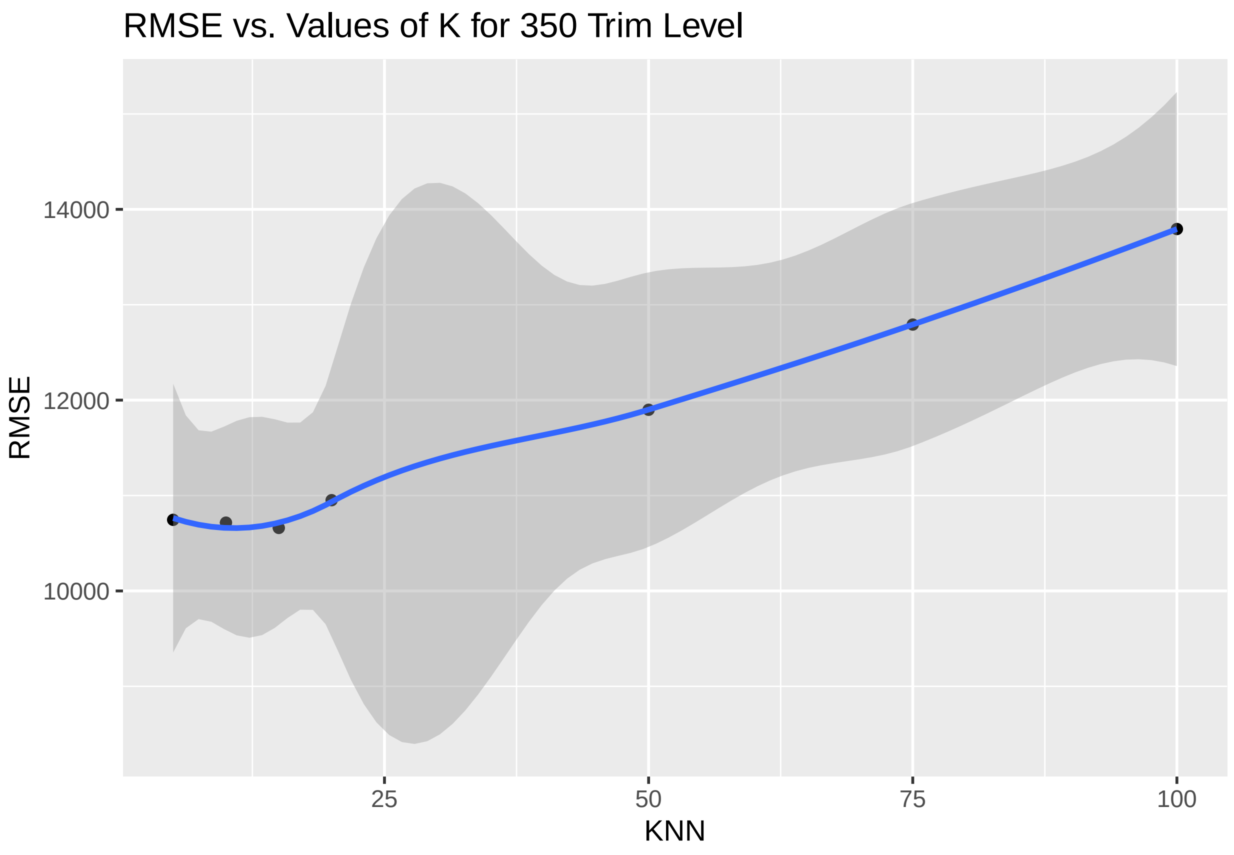
```
## [1] 12792.12
```

```r
rmse(Y_test350, ypred_knn100_350)
```

```
## [1] 13793.8
```

```r
#create new data set with KNN and RMSE values
rmse350 = data.frame(KNN = c(5, 10, 15, 20, 50, 75, 100), RMSE = c(rmse(Y_test350, ypred_knn5_350), rms

#plot KNN values and RMSE values to find where curve bottoms out
ggplot(data = rmse350) +
  geom_point(mapping = aes(x = KNN, y = RMSE)) +
  ggtitle("RMSE vs. Values of K for 350 Trim Level") +
  geom_smooth(mapping = aes(x = KNN, y = RMSE))
```

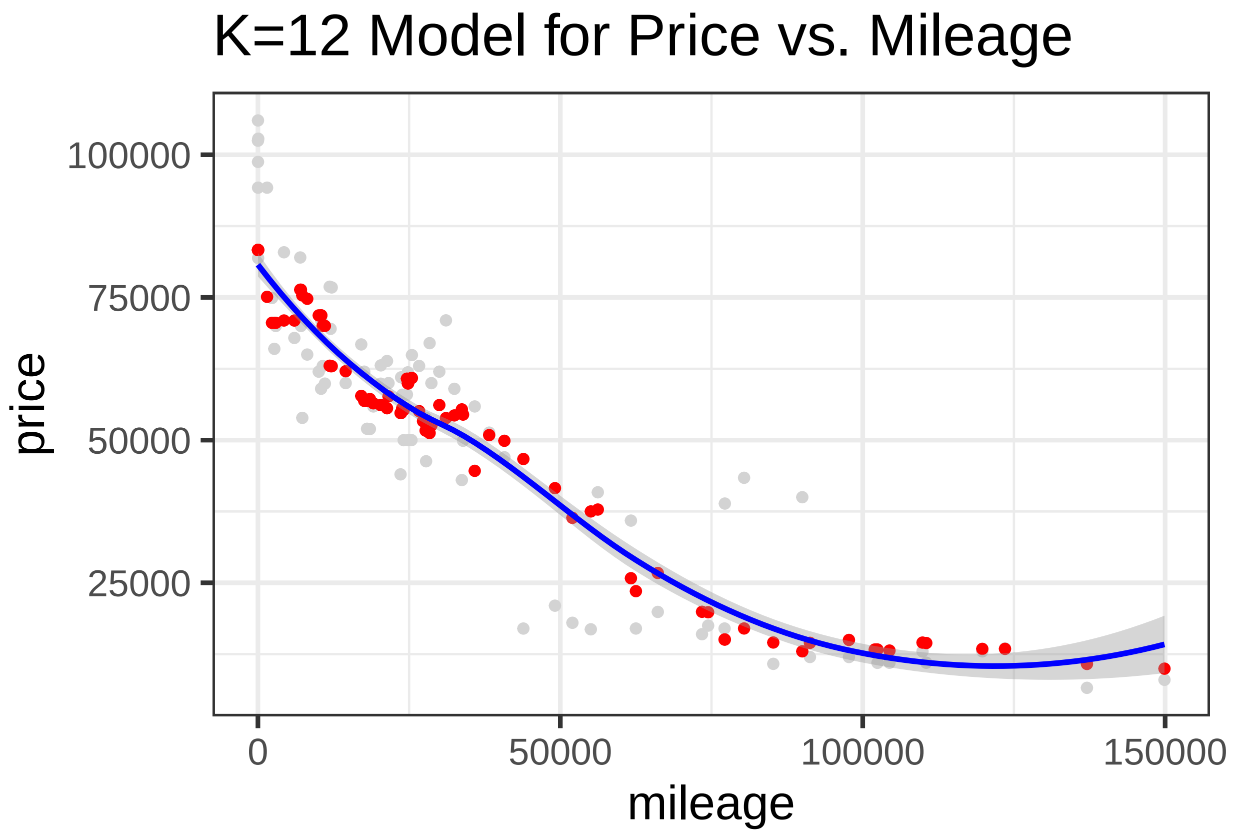
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```r
#lowest RMSE value at about K = 12, attach predictions to test data
knn12_350 = knn.reg(train = X_train350, test = X_test350, y = Y_train350, k=12)
ypred_knn12_350 = knn12_350$pred
S_test350$ypred_knn12_350 = ypred_knn12_350

#plot fitted model
plot350 = ggplot(data = S_test350) +
  geom_point(mapping = aes(x = mileage, y = price), color='lightgrey') +
  theme_bw(base_size=18)
```

```r
plot350 + geom_point(aes(x = mileage, y = ypred_knn12_350), color='red') +
  geom_smooth(aes(x = mileage, y = ypred_knn12_350), color='blue') +
  ggtitle("K=12 Model for Price vs. Mileage")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```r
set.seed(127)
#65 AMG group
N65 = nrow(sclass_65)
N_train65 = floor(0.8*N65)
N_test65 = N65-N_train65

train_ind65 = sample.int(N65, N_train65, replace = FALSE)

S_train65 = sclass_65[train_ind65,]
S_test65 = sclass_65[-train_ind65,]

X_train65 = select(S_train65, mileage)
Y_train65 = select(S_train65, price)
X_test65 = select(S_test65, mileage)
Y_test65 = select(S_test65, price)

knn5_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=5)
knn10_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=10)
knn15_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=15)
knn20_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=20)
knn50_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=50)
```

```r
knn75_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=75)
knn100_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=100)
knn200_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=200)

ypred_knn5_65 = knn5_65$pred
ypred_knn10_65 = knn10_65$pred
ypred_knn15_65 = knn15_65$pred
ypred_knn20_65 = knn20_65$pred
ypred_knn50_65 = knn50_65$pred
ypred_knn75_65 = knn75_65$pred
ypred_knn100_65 = knn100_65$pred
ypred_knn200_65 = knn200_65$pred

rmse(Y_test65, ypred_knn5_65)
```

```
## [1] 18668.81
```

```r
rmse(Y_test65, ypred_knn10_65)
```

```
## [1] 15334.08
```

```r
rmse(Y_test65, ypred_knn15_65)
```

```
## [1] 15249.93
```

```r
rmse(Y_test65, ypred_knn20_65)
```

```
## [1] 15448.43
```

```r
rmse(Y_test65, ypred_knn50_65)
```

```
## [1] 18309.37
```

```r
rmse(Y_test65, ypred_knn75_65)
```

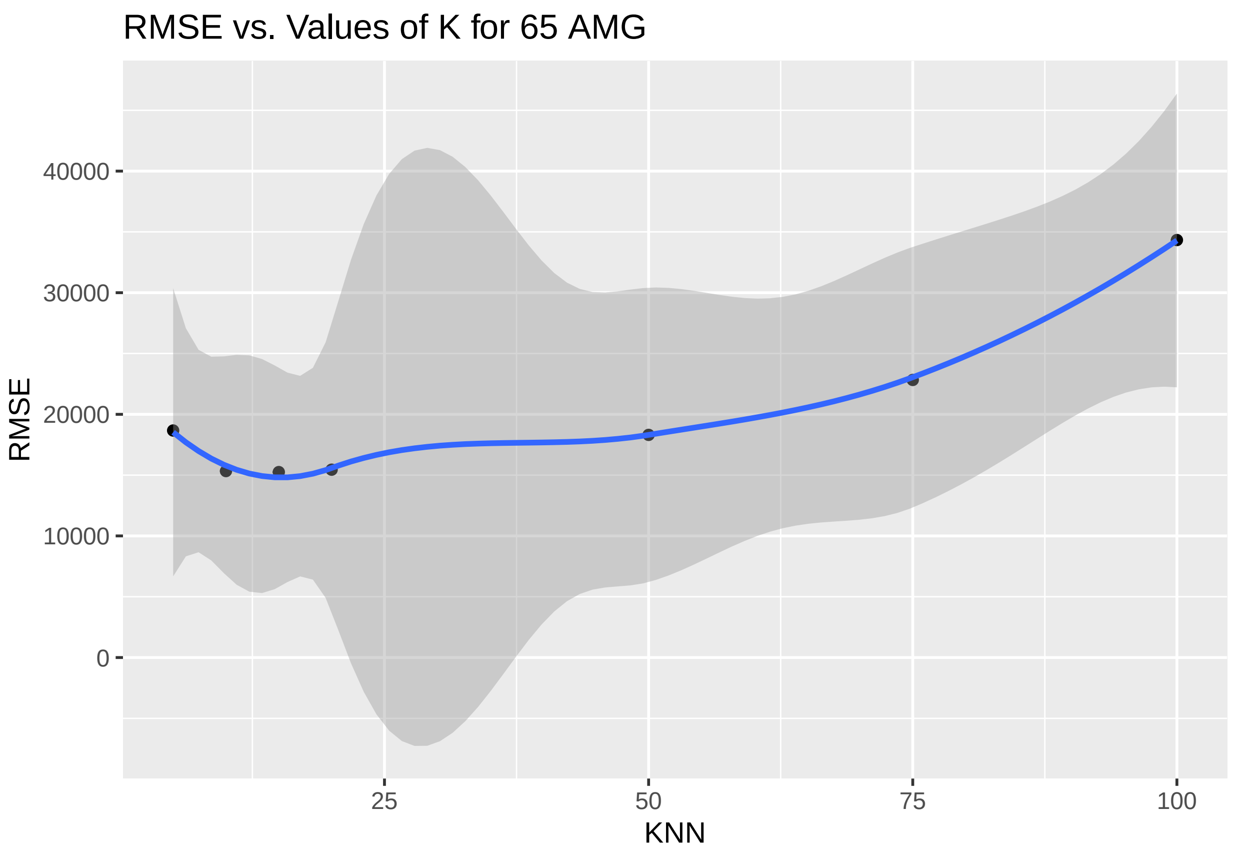
```
## [1] 22827.4
```

```r
rmse(Y_test65, ypred_knn100_65)
```

```
## [1] 34330.85
```

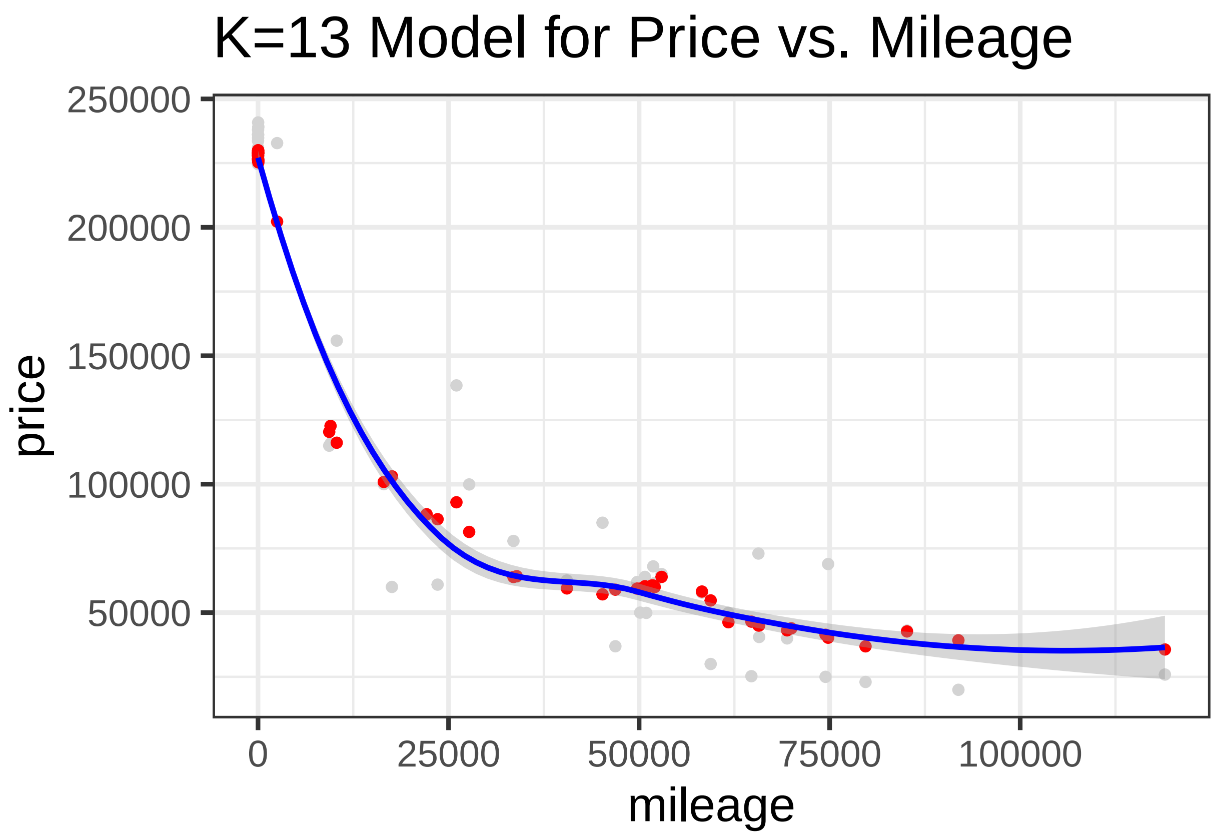
```r
rmse65 = data.frame(KNN = c(5, 10, 15, 20, 50, 75, 100), RMSE = c(rmse(Y_test65, ypred_knn5_65), rmse(Y

ggplot(data = rmse65) +
  geom_point(mapping = aes(x = KNN, y = RMSE)) +
  ggtitle("RMSE vs. Values of K for 65 AMG") +
  geom_smooth(mapping = aes(x = KNN, y = RMSE))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```r
#lowest RMSE value at about K = 13, attach predictions to test data
knn13_65 = knn.reg(train = X_train65, test = X_test65, y = Y_train65, k=13)
ypred_knn13_65 = knn13_65$pred
S_test65$ypred_knn13_65 = ypred_knn13_65

#plot fitted model
plot65 = ggplot(data = S_test65) +
  geom_point(mapping = aes(x = mileage, y = price), color='lightgrey') +
  theme_bw(base_size=18)

plot65 + geom_point(aes(x = mileage, y = ypred_knn13_65), color='red') +
  geom_smooth(aes(x = mileage, y = ypred_knn13_65), color='blue') +
  ggtitle("K=13 Model for Price vs. Mileage")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```
N65
```

```
## [1] 292
```

```
N350
```

```
## [1] 416
```

65 trim yields a larger optimal value of K. 350 trim had an optimal value of K equal to 12, while trim 65 had an optimal value of 13. The two values are nearly the same. The optimal k values varied based on the random sampling of the sample.int function. The small difference between the two can be attributed to the bias-variance tradeoff. The 350 trim data set had a K that is more likely to have memorized random noise and has higher variance, compared to the 65 trim that has slightly lower variance and higher bias. Since the two numbers are so close, the difference is likely negligiable. The sizes of the data sets could have influenced the optimal K value, but it is not likely since they only differ by 1.

Saratoga House Prices

```
library(tidyverse)
library(mosaic)
```

```
## Warning: package 'mosaic' was built under R version 3.5.2

## Loading required package: lattice

## Loading required package: ggformula

## Warning: package 'ggformula' was built under R version 3.5.2

## Loading required package: ggstance

## Warning: package 'ggstance' was built under R version 3.5.2

##
```

```
## Attaching package: 'ggstance'

## The following objects are masked from 'package:ggplot2':
##
##     geom_errorbarh, GeomErrorbarh

##
## New to ggformula?  Try the tutorials:
##  learnr::run_tutorial("introduction", package = "ggformula")
##  learnr::run_tutorial("refining", package = "ggformula")

## Loading required package: mosaicData

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by this.
##
## Note: If you use the Matrix package, be sure to load it BEFORE loading mosaic.

##
## Attaching package: 'mosaic'

## The following object is masked from 'package:Matrix':
##
##     mean

## The following object is masked from 'package:purrr':
##
##     cross

## The following object is masked from 'package:ggplot2':
##
##     stat

## The following objects are masked from 'package:dplyr':
##
##     count, do, tally

## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median, prop.test,
##     quantile, sd, t.test, var

## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```r
library(FNN)
library(foreach)
```

```
## Warning: package 'foreach' was built under R version 3.5.2
```

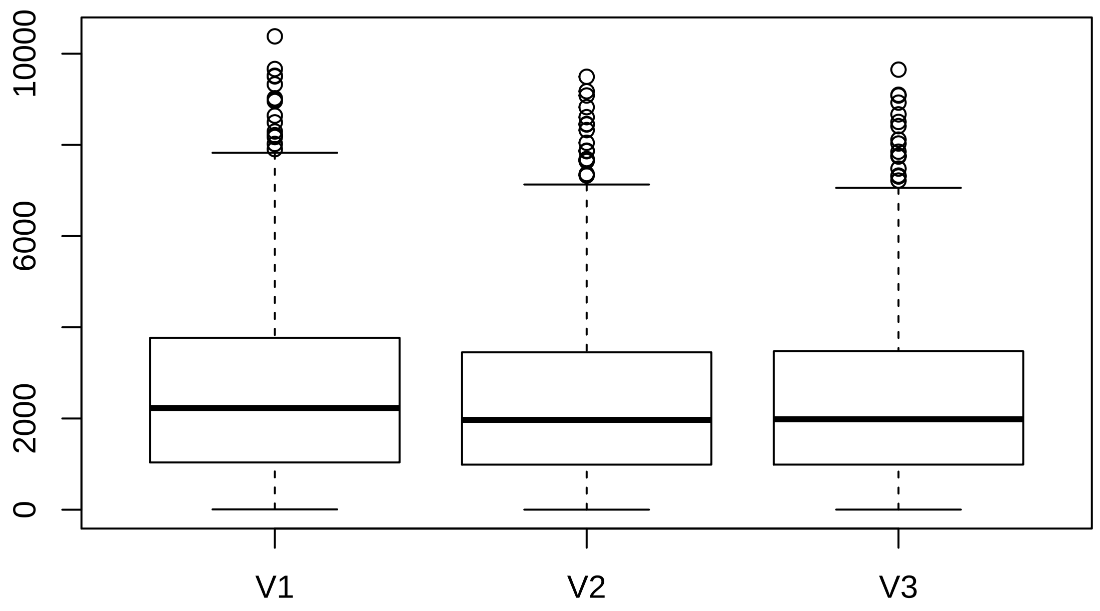
```
##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
data(SaratogaHouses)
#View(SaratogaHouses)

set.seed(50)

#HAND-BUILD
# easy averaging over train/test splits
n = nrow(SaratogaHouses)
n_train = round(0.8*n)  # round to nearest integer
n_test = n - n_train

rmse = function(y, yhat) {
  sqrt(mean(y-yhat)^2)
}

rmse_vals = do(1000)*{

  # re-split into train and test cases with the same sample sizes
  train_cases = sample.int(n, n_train, replace=FALSE)
  test_cases = setdiff(1:n, train_cases)
  saratoga_train = SaratogaHouses[train_cases,]
  saratoga_test = SaratogaHouses[test_cases,]

  # Fit to the training data
  lm_medium = lm(price ~ lotSize + age + livingArea + pctCollege + bedrooms
               + fireplaces + bathrooms + rooms + heating + fuel + centralAir,
               data=SaratogaHouses)

  lm_new1 = lm(price ~ lotSize + age + livingArea + pctCollege + bedrooms
             + fireplaces + bathrooms + rooms + heating + fuel + centralAir
             + landValue,
             data=SaratogaHouses)

  lm_new2 = lm(price ~ lotSize + age + livingArea + pctCollege + bedrooms
                 + fireplaces + bathrooms + rooms + heating + fuel + centralAir
                 + landValue + lotSize*livingArea,
                 data=SaratogaHouses)



  # Predictions out of sample
  yhat_test1 = predict(lm_medium, saratoga_test)
  yhat_test2 = predict(lm_new1, saratoga_test)
  yhat_test3 = predict(lm_new2, saratoga_test)
  c(rmse(saratoga_test$price, yhat_test1),
    rmse(saratoga_test$price, yhat_test2),
    rmse(saratoga_test$price, yhat_test3)
    )
}

rmse_vals
```

```
colMeans(rmse_vals)
```

```
##       V1       V2       V3
## 2633.600 2386.631 2386.610
```

```
boxplot(rmse_vals)
```



```
#KNN MODELS

# construct the training and test-set feature matrices
# note the "-1": this says "don't add a column of ones for the intercept"
Xtrain = model.matrix(price ~ lotSize + age + livingArea + pctCollege + bedrooms
                      + fireplaces + bathrooms + rooms + heating + fuel + centralAir
                      + landValue - 1, data=saratoga_train)
Xtest = model.matrix(price ~ lotSize + age + livingArea + pctCollege + bedrooms
                     + fireplaces + bathrooms + rooms + heating + fuel + centralAir
                     + landValue - 1, data=saratoga_test)
# training and testing set responses
ytrain = saratoga_train$price
ytest = saratoga_test$price
# now rescale:
scale_train = apply(Xtrain, 2, sd) # calculate std dev for each column
Xtilde_train = scale(Xtrain, scale = scale_train)
Xtilde_test = scale(Xtest, scale = scale_train) # use the training set scales!

head(Xtrain, 2)
```

```
##      lotSize age livingArea pctCollege bedrooms fireplaces bathrooms rooms
## 1225    0.24  17       1908         62        4          1       3.5     9
## 756     0.52  13        840         38        2          0       1.0     4
##      heatinghot air heatinghot water/steam heatingelectric fuelelectric fueloil
## 1225              1                      0               0            0       0
## 756               1                      0               0            0       0
##      centralAirNo landValue
## 1225            0     60200
## 756             1      4500
```

```
head(Xtilde_train, 2) %>% round(3)
```

```
##      lotSize    age livingArea pctCollege bedrooms fireplaces bathrooms  rooms
## 1225  -0.370 -0.363      0.248      0.623    1.030      0.698     2.417  0.871
## 756    0.004 -0.502     -1.472     -1.660   -1.433     -1.092    -1.363 -1.313
##      heatinghot air heatinghot water/steam heatingelectric fuelelectric fueloil
## 1225          0.735                 -0.469          -0.454       -0.465  -0.373
## 756           0.735                 -0.469          -0.454       -0.465  -0.373
##      centralAirNo landValue
## 1225       -1.326     0.742
## 756         0.754    -0.853
```

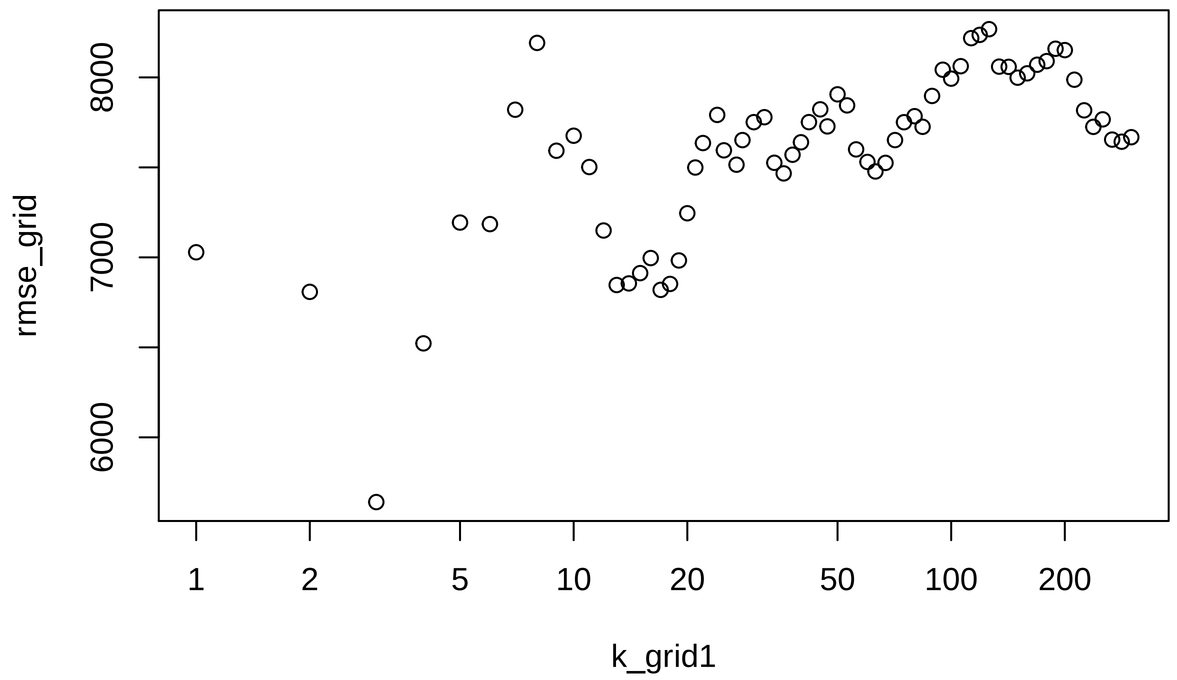
```
K = 10
# fit the model
knn_model = knn.reg(Xtilde_train, Xtilde_test, ytrain, k=K)
# calculate test-set performance
rmse(ytest, knn_model$pred)
```

```
## [1] 7675.831
```

```
rmse(ytest, yhat_test2) # from the linear model with the same features
```

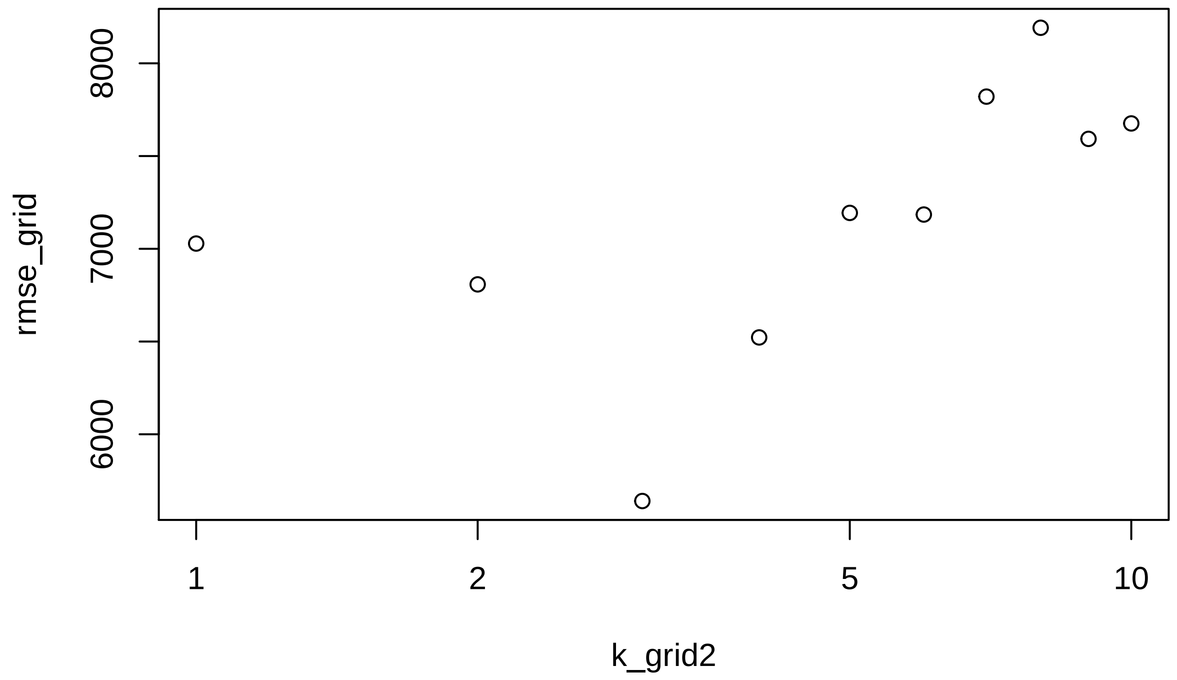
```
## [1] 119.9046
```

```
k_grid1 = exp(seq(log(1), log(300), length=100)) %>% round %>% unique
rmse_grid = foreach(K = k_grid1, .combine='c') %do% {
  knn_model = knn.reg(Xtilde_train, Xtilde_test, ytrain, k=K)
  rmse(ytest, knn_model$pred)
}
plot(k_grid1, rmse_grid, log='x') + abline(h=rmse(ytest, yhat_test2))
```



```
## integer(0)
```

```
#NARROW IT DOWN
k_grid2 = 1:10 %>% round %>% unique
rmse_grid = foreach(K = k_grid2, .combine='c') %do% {
  knn_model = knn.reg(Xtilde_train, Xtilde_test, ytrain, k=K)
  rmse(ytest, knn_model$pred)
}
plot(k_grid2, rmse_grid, log='x') + abline(h=rmse(ytest, yhat_test2))
```



```
## integer(0)
```

```r
which.min(rmse_grid)
```

```
## [1] 3
```

```r
min(rmse_grid)
```

```
## [1] 5640.279
```

```r
#The K-value is 3
```

Using our model, we concluded that adding landValue and lotSize*livingArea makes the model better than the "medium" model, which included lotSize, age, livingArea, pctCollege, bedrooms, fireplaces, bathrooms, rooms, heating, fuel, and centralAir. We thought landValue is crucial because depending on the land Value, the Saratoga house prices would increase or decrease. Moreover, we thought the interaction between the lotSize and living Area would decrease the means of the RMSE. We have tested 3 different models; the ???medium???, ???medium??? with the landValue, and ???medium??? with landValue and lotSize*livingArea. The mean of the "medium" model resulted in 2633.6, ???medium??? with the landValue had 2386.631, and ???medium??? with landValue and lotSize*livingArea had 2386. 61, which was similar to the 2nd model. We can conclude that the ???medium??? with landValue and lotSize*livingArea model was the best fit. After running the KNN model, we concluded that k = 3 is the best for this model.

Predicting When Articles Go Viral

```r
##LINEAR REGRESSION

##LINEAR REGRESSION
news <- read.csv("online_news.csv")
View(news)

set.seed(50)
NW = nrow(news)
NW_train = floor(0.8*NW) #floor means round number
NW_test = NW - NW_train

# randomly sample a set of data points to include in the training set
train_ind = sample.int(NW, NW_train, replace=FALSE)


# Define the training and testing set
DNW_train = news[train_ind,] #comma blank means all of the columns
DNW_test = news[-train_ind,]

z <- lm(formula = shares ~ num_hrefs, data = DNW_train)
yhat_testz = predict(z, DNW_test)

w <- lm(formula = shares ~ data_channel_is_world, data = DNW_train)
yhat_testw = predict(w, DNW_test)

x <- lm(formula = shares ~ average_token_length + data_channel_is_world + num_hrefs + n_tokens_title + 
yhat_testx = predict(x, DNW_test)

# Root mean squared prediction error
rmse = function(y, yhat) { sqrt( mean( (y - yhat)^2 ) )}
rmse(DNW_test$shares, yhat_testz)
```

```
## [1] 8298.898
```

```
rmse(DNW_test$shares, yhat_testw)
```

```
## [1] 8296.555
```

```
rmse(DNW_test$shares, yhat_testx) #smallest RMSE
```

```
## [1] 8262.774
```

```
#Confusion Matrix
viralx <- ifelse(predict(x, DNW_test) > 1400, 1, 0)
testviral<-ifelse(DNW_test$shares > 1400,1,0)
confusion_x = table(y = testviral, yhat = viralx)
confusion_x
```

```
##    yhat
## y      0    1
##   0   31 3982
##   1    9 3907
```

```
#Error Rates
totalcount<-36+4014+19+3860
overall_error_rate_x<-(19+4014)/totalcount
TPRx<-3860/(19+3860)
FPRx<-4014/(36+4014)
overall_error_rate_x
```

```
## [1] 0.5086392
```

```
TPRx
```

```
## [1] 0.9951018
```

```
FPRx
```

```
## [1] 0.9911111
```

Regression Approach

After setting the seed to 50 and defining the training and testing set, we tested out various linear regression. There were some variables that gave a high root mean squared prediction error which also means a low accuracy. First, we tested num_hrefts and shares which gave a root mean squared prediction error of 12144.99. Then, we tried to use multiple variables for smaller root mean squared prediction error which gives a measurement of error of the model in predicting the data. Second, we tested data_channel_is_world and it gave a root mean squared prediction error of 12142.06. Finally, we tested out average_token_length, data_channel_is_world, num_hrefs, n_tokens_title, num_videos, num_keywords, num_imgs, weekday_is_friday, weekday_is_saturday, and n_tokens_content with shares and got the lowest root mean squared prediction error, 12120.03. Therefore we concluded that this model is the best model for predicting shares. We then looked at the confusion matrix that this model produced. From this matrix, we were able to find the overall error rate, true positive rate, and false positive rate to be 50.86%, 99.51%, and 99.11%, respectively. Overall, this shows that our model is only slightly better than flipping a coin in predicting if an article goes viral. Additionally, it seems like this model also predicts that an article will most likely be viral, which might explain the high true positive rate and false positive rate.

```
##CLASSIFICATION METHOD


words <- read.csv("online_news.csv")
words$viral <- rep(0,nrow(words))
words$viral <- ifelse(words$shares > 1400, 1, 0)
#View(words)
```

```r
# Make a train-test split
set.seed(50)
NW = nrow(words)
NW_train = floor(0.8*NW) #floor means round number
NW_test = NW - NW_train

# randomly sample a set of data points to include in the training set
train_ind = sample.int(NW, NW_train, replace=FALSE)

# Define the training and testing set
DNW_train = words[train_ind,] #comma blank means all of the columns
DNW_test = words[-train_ind,]

##LOGISTIC REGRESSION

viralm1<-glm(viral~ num_hrefs+ n_tokens_content+ avg_negative_polarity, family=binomial, data=DNW_train

viralm2<-glm(viral~ num_imgs*num_videos, family=binomial, data=DNW_train)

viralm3<-glm(viral~weekday_is_monday+
              weekday_is_saturday+self_reference_max_shares + num_videos*n_tokens_content + max_negativ

viralm4<-glm(viral~num_hrefs+avg_negative_polarity+avg_positive_polarity, family = binomial, data = DNW

summary(viralm1)
```

```
##
## Call:
## glm(formula = viral ~ num_hrefs + n_tokens_content + avg_negative_polarity,
##     family = binomial, data = DNW_train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.303  -1.140  -1.087   1.207   1.318
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)           -1.772e-01  2.791e-02  -6.349 2.16e-10 ***
## num_hrefs              1.732e-02  1.200e-03  14.438  < 2e-16 ***
## n_tokens_content       1.121e-05  2.653e-05   0.423   0.6727
## avg_negative_polarity  1.669e-01  8.986e-02   1.858   0.0632 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43961  on 31714  degrees of freedom
## Residual deviance: 43682  on 31711  degrees of freedom
## AIC: 43690
##
## Number of Fisher Scoring iterations: 4
```

```r
summary(viralm2)
```

```
##
## Call:
## glm(formula = viral ~ num_imgs * num_videos, family = binomial,
##     data = DNW_train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.905  -1.137  -1.129   1.216   1.226
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -0.1140807  0.0135389  -8.426  < 2e-16 ***
## num_imgs             0.0176913  0.0014844  11.918  < 2e-16 ***
## num_videos           0.0078156  0.0028026   2.789  0.00529 **
## num_imgs:num_videos -0.0008251  0.0004716  -1.750  0.08018 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43961  on 31714  degrees of freedom
## Residual deviance: 43805  on 31711  degrees of freedom
## AIC: 43813
##
## Number of Fisher Scoring iterations: 4
```

```r
summary(viralm3)
```

```
##
## Call:
## glm(formula = viral ~ weekday_is_monday + weekday_is_saturday +
##     self_reference_max_shares + num_videos * n_tokens_content +
##     max_negative_polarity, family = binomial, data = DNW_train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.894  -1.129  -1.077   1.217   1.822
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -2.431e-01  2.533e-02  -9.595  < 2e-16 ***
## weekday_is_monday          -4.010e-02  3.021e-02  -1.327  0.18436
## weekday_is_saturday         9.523e-01  5.125e-02  18.581  < 2e-16 ***
## self_reference_max_shares   3.772e-06  4.304e-07   8.765  < 2e-16 ***
## num_videos                  1.259e-02  4.554e-03   2.764  0.00571 **
## n_tokens_content            1.836e-04  2.594e-05   7.077 1.47e-12 ***
## max_negative_polarity      -2.346e-01  1.226e-01  -1.914  0.05565 .
## num_videos:n_tokens_content -1.229e-05  3.793e-06  -3.241  0.00119 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 43961  on 31714  degrees of freedom
## Residual deviance: 43383  on 31707  degrees of freedom
## AIC: 43399
##
## Number of Fisher Scoring iterations: 4
```

```r
summary(viralm4)
```

```
##
## Call:
## glm(formula = viral ~ num_hrefs + avg_negative_polarity + avg_positive_polarity,
##     family = binomial, data = DNW_train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.294  -1.143  -1.069   1.204   1.357
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)           -0.260461   0.042021  -6.198 5.71e-10 ***
## num_hrefs              0.017052   0.001111  15.345  < 2e-16 ***
## avg_negative_polarity  0.227380   0.092725   2.452  0.01420 *
## avg_positive_polarity  0.304920   0.113827   2.679  0.00739 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43961  on 31714  degrees of freedom
## Residual deviance: 43675  on 31711  degrees of freedom
## AIC: 43683
##
## Number of Fisher Scoring iterations: 4
```

```r
#model 3 has the lowest AIC

#TESTING

total <- 3040+1010+2552+1327
phat_test_viralm1 = predict(viralm1, DNW_test, type='response')
yhat_test_viralm1 = ifelse(phat_test_viralm1 > 0.5, 1, 0)
confusion_out_logit1 = table(y = DNW_test$viral, yhat = yhat_test_viralm1)
confusion_out_logit1
```

```
##    yhat
## y      0    1
##   0 3053  960
##   1 2695 1221
```

```r
error1 <- (2552+1010)/total

phat_test_viralm2 = predict(viralm2, DNW_test, type='response')
yhat_test_viralm2 = ifelse(phat_test_viralm2 > 0.5, 1, 0)
confusion_out_logit2 = table(y = DNW_test$viral, yhat = yhat_test_viralm2)
confusion_out_logit2
```

```
##    yhat
## y      0    1
##   0 3216  797
##   1 2846 1070
```

```
error2 <- (2745+837)/total

phat_test_viralm3 = predict(viralm3, DNW_test, type='response')
yhat_test_viralm3 = ifelse(phat_test_viralm3 > 0.5, 1, 0)
confusion_out_logit3 = table(y = DNW_test$viral, yhat = yhat_test_viralm3)
confusion_out_logit3
```

```
##    yhat
## y      0    1
##   0 3436  577
##   1 2980  936
```

```
error3 <- (2850+607)/total

phat_test_viralm4 = predict(viralm4, DNW_test, type='response')
yhat_test_viralm4 = ifelse(phat_test_viralm4 > 0.5, 1, 0)
confusion_out_logit4 = table(y = DNW_test$viral, yhat = yhat_test_viralm4)
confusion_out_logit4
```

```
##    yhat
## y      0    1
##   0 3034  979
##   1 2648 1268
```

```
error4 <- (2523+1041)/total

error1
```

```
## [1] 0.449237
```

```
error2
```

```
## [1] 0.4517594
```

```
error3
```

```
## [1] 0.4359945
```

```
error4
```

```
## [1] 0.4494892
```

```
#viralm3 is the best model with the lowest error rate

#Other Error Rates
confusion_out_logit3
```

```
##    yhat
## y      0    1
##   0 3436  577
##   1 2980  936
```

```
TPR3<-1029/(2850+1029)
FPR3<-607/(3443+607)
TPR3
```

```
## [1] 0.2652746
```

```
FPR3
```

```
## [1] 0.1498765
```

After approaching this problem from a linear regression standpoint, we then tried a logistic regression standpoint. We created a ???Viral??? variable that assigned articles as viral if they had an amount of shares greater than 1400. Next, we also set the seed to 50 and split the data into a train and test set. From there, we created multiple logistic regression models to find a model that would be able to give some insight into predicting if an article were to go viral or not. We then evaluated these models by looking at the confusion matrix and AIC that each model produced. From these two metrics, we saw that our third model (which predicted viral from weekday_is_monday, weekday_is_saturday, self_reference_max_shares, num_videos*n_tokens_content, and max_negative_polarity) had the lowest AIC (43455) and overall error rate (43.59%). We chose the variables for this model because we had a hunch that these variables could be useful in predicting if an article went viral. Since we determined that the third model was the best, we proceeded to find other error rates. From this, we found that this model had a true positive rate of 26.52% and a false positive rate of 14.98%.

After completing both approaches, it can be seen that creating a threshold first and then regressing second produced better results. Although the true positive rate decreased, we were able to reduce our overall error rate and false positive rate. This shows that our model did not just try to classify everything as viral, unlike our initial linear model; which is most likely the reason that creating a threshold first is more beneficial to avoid such issues.