excercise2

KNN PRACTICE

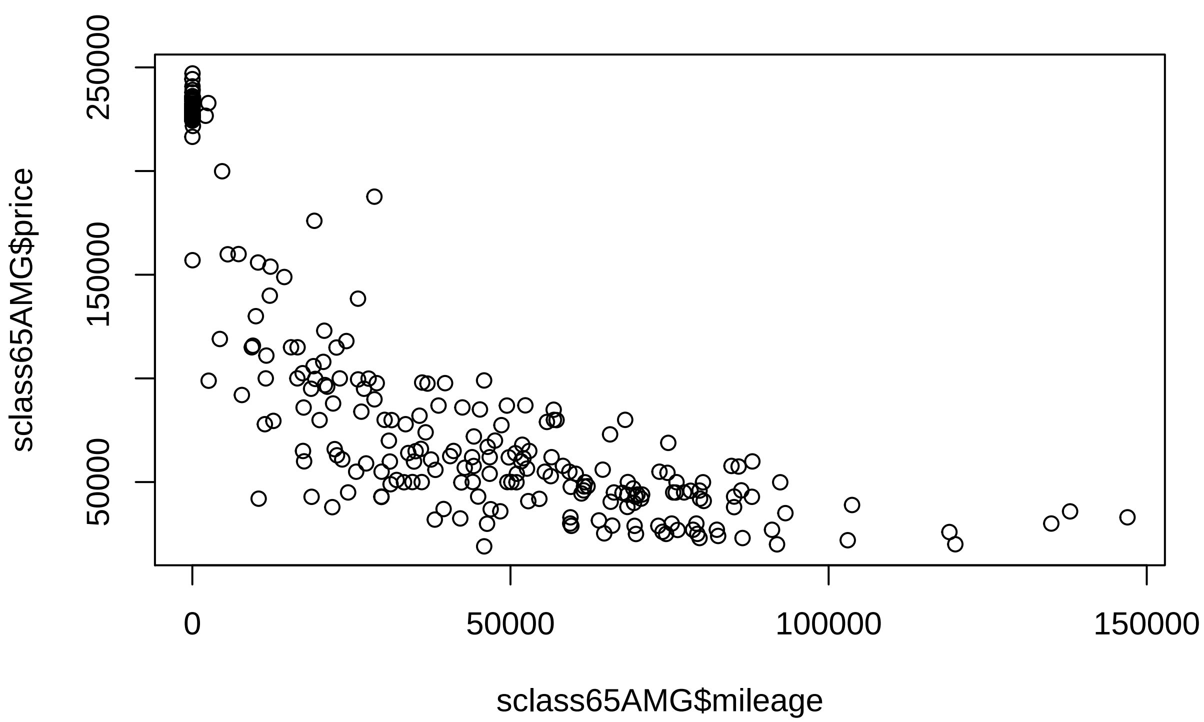
```r
library(tidyverse)
```

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

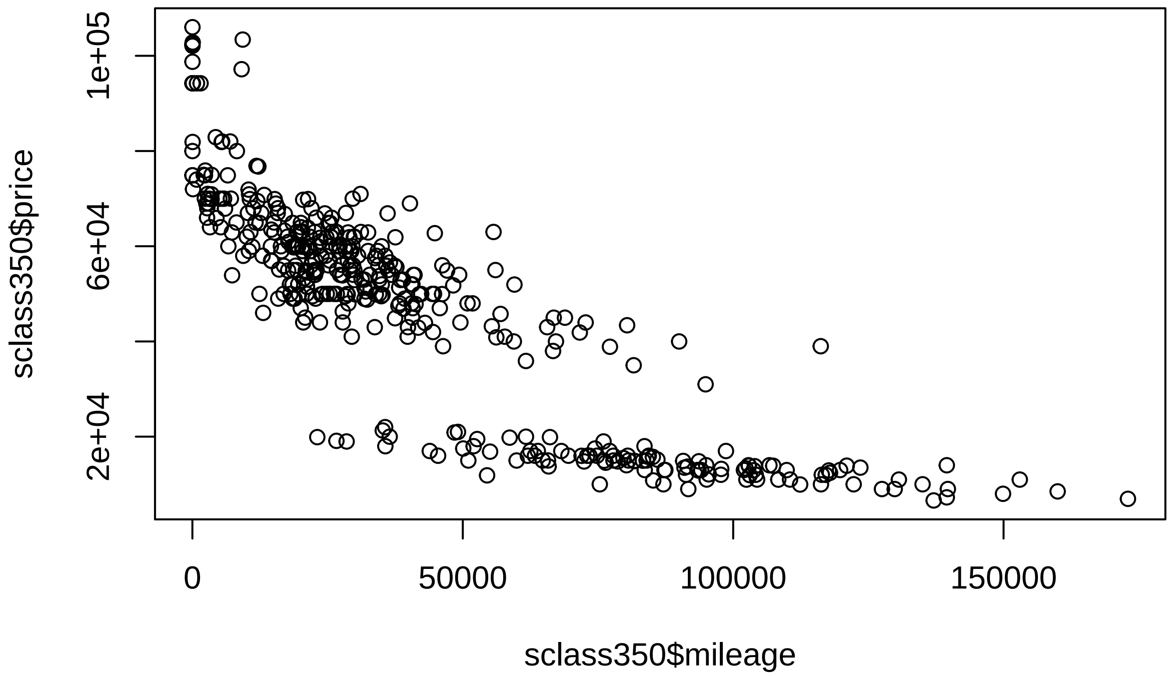
```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

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

```r
sclass <- read.csv("~/Documents/R/SDS 323/SDS323-master/data/sclass.csv")

# trim subsets
sclass350 <- dplyr::select(subset(sclass, trim == "350"), mileage, price)
sclass65AMG <- dplyr::select(subset(sclass, trim == "65 AMG"), mileage, price)

# plot price vs mileage for each trim
plot(sclass65AMG$mileage, sclass65AMG$price)
```




```r
plot(sclass350$mileage, sclass350$price)
```



Looking at the two plots, the data seems to indicate different relationships between mileage and price for each S Class. Hence using a sepereate KNN regression model for each S Class seems to be a good idea.

```r
library(FNN)
library(mosaic)
```

```
## Loading required package: lattice

## Loading required package: ggformula

## Loading required package: ggstance

##
## 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

## Registered S3 method overwritten by 'mosaic':
##   method                           from
##   fortify.SpatialPolygonsDataFrame ggplot2
```

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

## 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: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
```

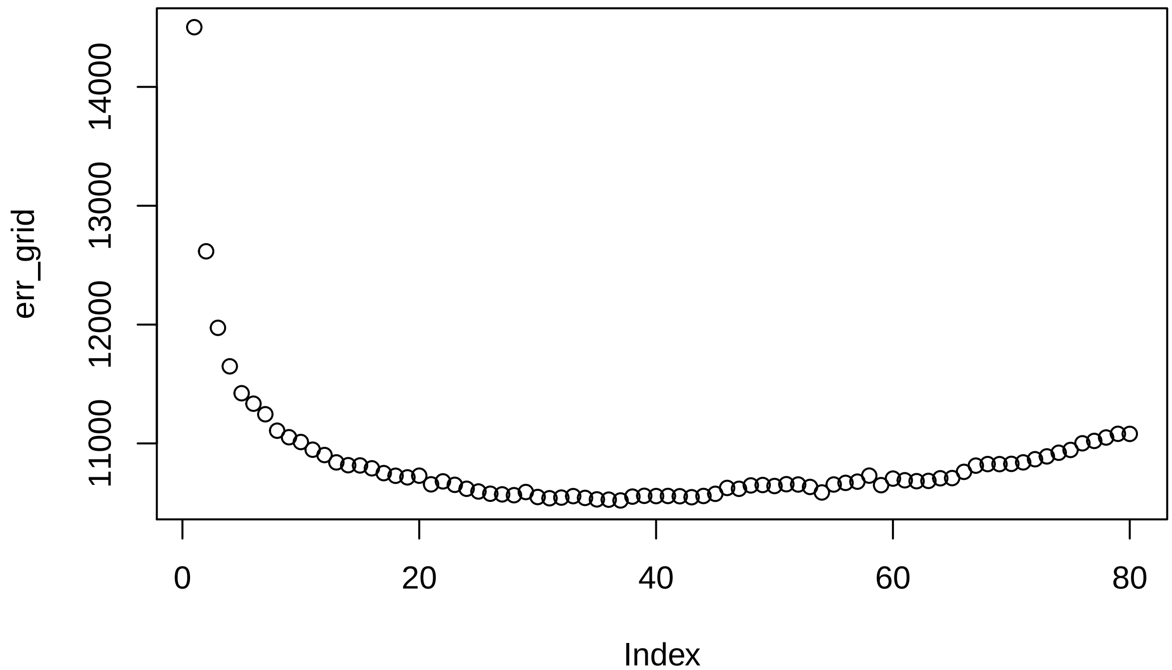
```
library(doMC)
```

```
## Loading required package: foreach

##
## Attaching package: 'foreach'

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

## Loading required package: iterators

## Loading required package: parallel
```

```
library(boot)
```

```
##
## Attaching package: 'boot'

## The following object is masked from 'package:mosaic':
##
##     logit

## The following object is masked from 'package:lattice':
##
##     melanoma
```

```r
options(`mosaic:parallelMessage` = FALSE)
set.seed(100)
set.rseed(100)

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

# KNN regression for S Class 350
X <- dplyr::select(sclass350, -price)
y <- sclass350$price
n <- nrow(sclass350)
train_ind <- n * 0.8

k_grid <- seq(1,80,by=1)
err_grid <- foreach(k = k_grid, .combine = 'c') %do% {
  out = do(500)*{
    # test/train split
    train_ind <- sample.int(nrow(sclass350), 0.8*nrow(sclass350))
    X_train <- data.frame(X[train_ind,])
    X_test <- data.frame(X[-train_ind,])
    y_train <- y[train_ind]
    y_test <- y[-train_ind]

    # scale train and test feature by the sd of train features
    scale_factors <- apply(X_train, 2, sd, na.rm = TRUE)
    X_train_sc <- scale(X_train, scale = scale_factors)
    X_test_sc <- scale(X_test, scale = scale_factors)

    model <- knn.reg(train = X_train_sc, test = X_test_sc, y = y_train, k = k)

    rmse(y_test, model$pred)
  }
  mean(out$result)
}
plot(err_grid)
```




```
# index of optimal K and RMSE
which.min(err_grid)
```

```
## [1] 37
```

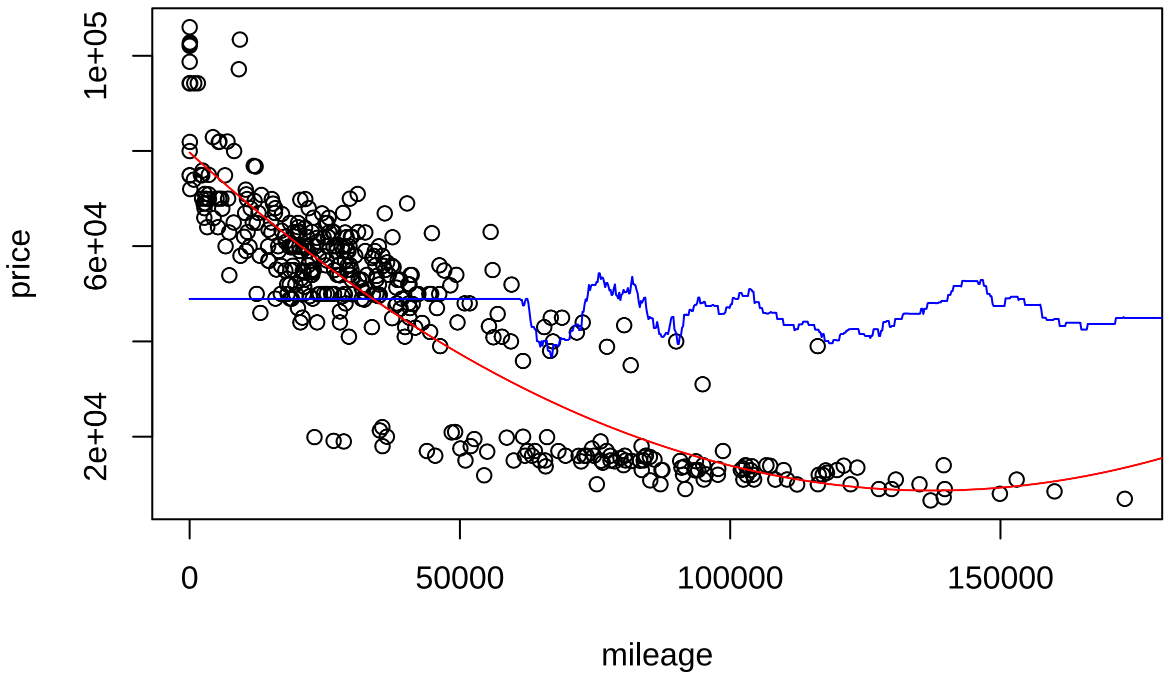
```
min(err_grid)
```

```
## [1] 10520.23
```

```
# plot optimal K
scale_factors <- apply(X, 2, sd, na.rm = TRUE)
X_test <- seq(0, 200000, length.out = 1000)
X_train_sc <- scale(X_train, scale = scale_factors)
X_test_sc <- scale(X_test, scale = scale_factors)
best <- knn.reg(train = X_train_sc, test = X_test_sc, y = y, k = which.min(err_grid))

# quadratic model for comparisson
lm <- glm(price ~ poly(x = mileage, degree = 2), data = sclass350)
lm.cv <- cv.glm(data = sclass350, lm)

# rmse of quadratic model
sqrt(lm.cv$delta[1])
```

```
## [1] 10238.95
```

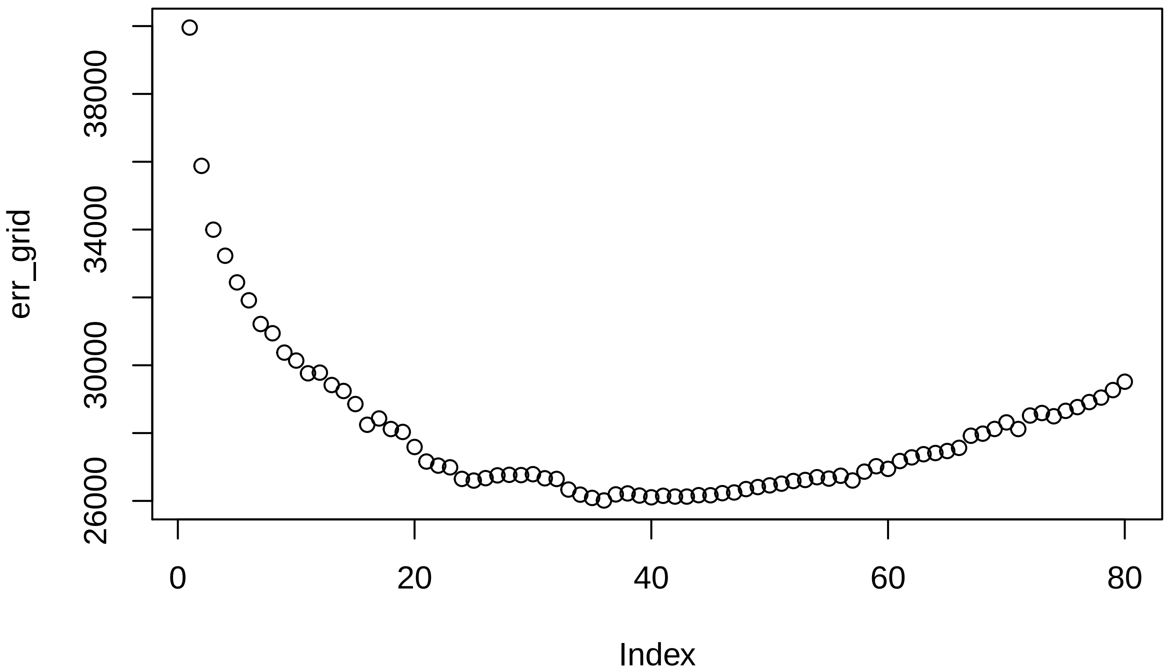
```
df1 <- data.frame(X_test, best$pred)
df2 <- data.frame(X_test, predict(lm, newdata = data.frame(mileage = X_test)))
plot(price ~ mileage, data = sclass350, col = "black")
lines(df1, col = "blue")
lines(df2, col = "red")
```



The optimal value for K is 37, which happens to be odd, and has an RMSE of 10520.23.

```
library(doMC)
set.seed(100)
set.rseed(100)
# KNN regression for S Class 65 AMG

X <- dplyr::select(sclass65AMG, -price)
y <- sclass65AMG$price

n <- nrow(sclass65AMG)
train_n <- n*0.8

k_grid <- seq(1,80,by=1)
err_grid <- foreach(k = k_grid, .combine = 'c') %do% {
  out = do(500)*{
    # make test/train split
    train_ind <- sample.int(n, train_n)
    X_train <- data.frame(X[train_ind,])
    X_test <- data.frame(X[-train_ind,])
    y_train <- y[train_ind]
    y_test <- y[-train_ind]

    # scale train and test feature by the sd of train features
    scale_factors <- apply(X_train, 2, sd, na.rm = TRUE)
    X_train_sc <- scale(X_train, scale = scale_factors)
    X_test_sc <- scale(X_test, scale = scale_factors)

    model <- knn.reg(train = X_train_sc, test = X_test_sc, y = y_train, k = k)

    rmse(y_test, model$pred)
  }
  mean(out$result)
```

```r
}
plot(err_grid)
```




```r
# optimal K and RMSE
which.min(err_grid)
```

```
## [1] 36
```

```r
min(err_grid)
```

```
## [1] 26012.75
```

```r
# optimal odd K and RMSE
odd <- err_grid[c(TRUE, FALSE)]
2*which.min(odd)-1
```

```
## [1] 35
```

```r
min(odd)
```

```
## [1] 26087.95
```

```r
# RMSE difference
min(err_grid) - min(odd)
```

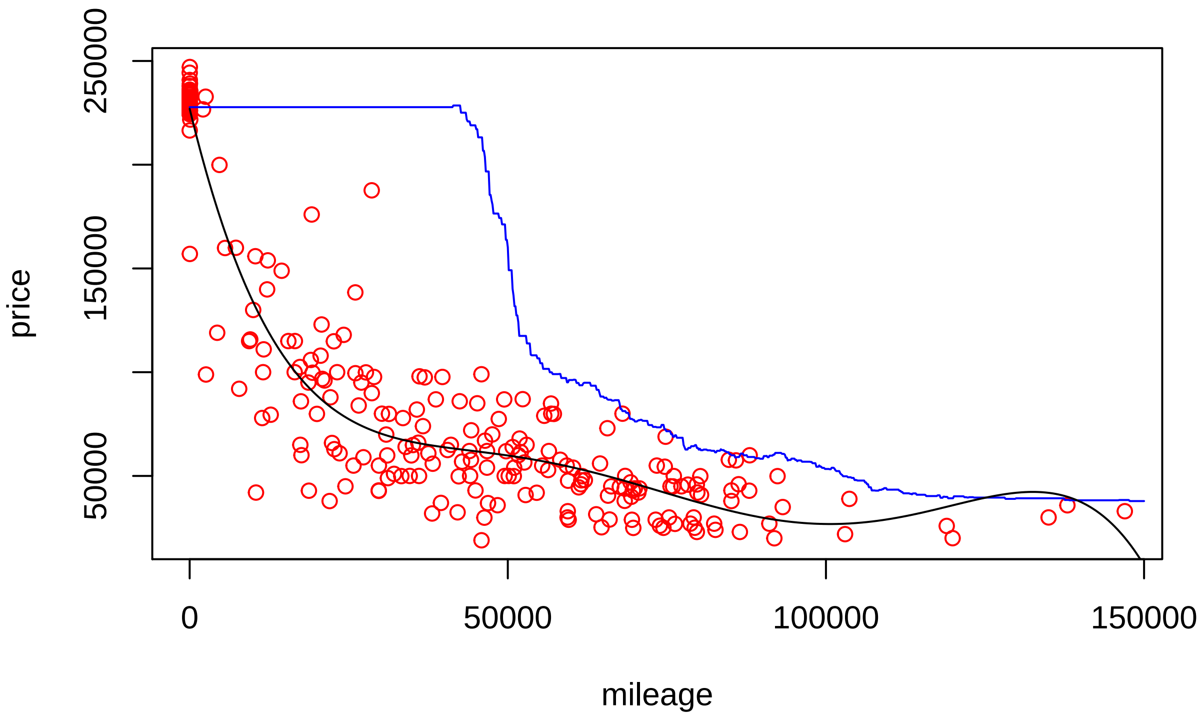
```
## [1] -75.20318
```

```r
lm <- glm(price ~ poly(x = mileage, degree = 5), data = sclass65AMG)
lm.cv <- cv.glm(sclass65AMG, lm)
sqrt(lm.cv$delta[1])
```

```
## [1] 21194.57
```

```r
X_test <- seq(0, 150000, length.out = 1000)
scale_factors <- apply(X, 2, sd, na.rm = TRUE)
X_train_sc <- scale(X, scale = scale_factors)
X_test_sc <- scale(X_test, scale = scale_factors)
```

```r
best <- knn.reg(X_train_sc, X_test_sc, y = y, k = which.min(err_grid))

df1 <- data.frame(X_test, best$pred)
df2 <- data.frame(X_test, predict(lm, newdata = data.frame(mileage = X_test)))

plot(price ~ mileage, data = sclass65AMG, col = "red")
lines(df1, col = "blue")
lines(df2, col = "black")
```



The model for 350 trim has the larger optimal K, this is likely because the data set is larger than that of 65 AMG trim. Therefore the model can average more data points without losing accuracy. The optimal K value is 36 and the optimal odd K is 35, the RMSE difference between the two is ~75. Because the two values for K are neighbors and the graph doesnt seem volatile, I think it is safe to use K = 36 as the optimal value for K. It is interesting to note that for both trims a simple polynomial model out preformed the KNN regression model, and is substantially easier to implement.

SARATOGA HOUSE PRICES

```r
set.seed(100)
set.rseed(100)
library(mosaic)
library(boot)
library(lm.beta)
data(SaratogaHouses)

summary(SaratogaHouses)
```

```
##      price            lotSize             age           landValue
##  Min.   :  5000   Min.   : 0.0000   Min.   :  0.00   Min.   :   200
##  1st Qu.:145000   1st Qu.: 0.1700   1st Qu.: 13.00   1st Qu.: 15100
##  Median :189900   Median : 0.3700   Median : 19.00   Median : 25000
##  Mean   :211967   Mean   : 0.5002   Mean   : 27.92   Mean   : 34557
##  3rd Qu.:259000   3rd Qu.: 0.5400   3rd Qu.: 34.00   3rd Qu.: 40200
```

```
##  Max.   :775000   Max.   :12.2000   Max.   :225.00   Max.   :412600
##    livingArea     pctCollege       bedrooms       fireplaces       bathrooms
##  Min.   : 616   Min.   :20.00   Min.   :1.000   Min.   :0.0000   Min.   :0.0
##  1st Qu.:1300   1st Qu.:52.00   1st Qu.:3.000   1st Qu.:0.0000   1st Qu.:1.5
##  Median :1634   Median :57.00   Median :3.000   Median :1.0000   Median :2.0
##  Mean   :1755   Mean   :55.57   Mean   :3.155   Mean   :0.6019   Mean   :1.9
##  3rd Qu.:2138   3rd Qu.:64.00   3rd Qu.:4.000   3rd Qu.:1.0000   3rd Qu.:2.5
##  Max.   :5228   Max.   :82.00   Max.   :7.000   Max.   :4.0000   Max.   :4.5
##      rooms                    heating           fuel
##  Min.   : 2.000   hot air         :1121   gas     :1197
##  1st Qu.: 5.000   hot water/steam: 302   electric: 315
##  Median : 7.000   electric        : 305   oil     : 216
##  Mean   : 7.042
##  3rd Qu.: 8.250
##  Max.   :12.000
##                sewer      waterfront newConstruction centralAir
##  septic           : 503   Yes:  15   Yes:  81        Yes: 635
##  public/commercial:1213   No :1713   No :1647        No :1093
##  none             :  12
##
##
##
```

```r
# baseline medium model with 11 main effects
lm_medium <- glm(price ~ lotSize + age + livingArea + pctCollege + bedrooms +
        fireplaces + bathrooms + rooms + heating + fuel + centralAir, data=SaratogaHouses)
summary(lm_medium)
```

```
##
## Call:
## glm(formula = price ~ lotSize + age + livingArea + pctCollege +
##     bedrooms + fireplaces + bathrooms + rooms + heating + fuel +
##     centralAir, data = SaratogaHouses)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -232296   -40021    -7679    28919   527748
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             28627.732  12224.396   2.342 0.019302 *
## lotSize                  9350.452   2421.118   3.862 0.000117 ***
## age                        47.547     65.149   0.730 0.465600
## livingArea                 91.870      5.033  18.253  < 2e-16 ***
## pctCollege                296.508    165.531   1.791 0.073428 .
## bedrooms               -15630.719   2885.084  -5.418 6.89e-08 ***
## fireplaces                985.061   3385.478   0.291 0.771112
## bathrooms               22006.971   3821.764   5.758 1.00e-08 ***
## rooms                    3259.119   1093.631   2.980 0.002922 **
## heatinghot water/steam  -9429.795   4738.934  -1.990 0.046765 *
## heatingelectric         -3609.986  14009.898  -0.258 0.796689
## fuelelectric           -12094.122  13792.538  -0.877 0.380686
## fueloil                 -8873.140   5395.649  -1.644 0.100257
## centralAirNo           -17112.819   3922.489  -4.363 1.36e-05 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 4393644476)
##
##     Null deviance: 1.6736e+13  on 1727  degrees of freedom
## Residual deviance: 7.5307e+12  on 1714  degrees of freedom
## AIC: 43287
##
## Number of Fisher Scoring iterations: 2
```

```
# K=10 CV
lm_medium.cv <- cv.glm(data = SaratogaHouses, lm_medium, K = 10)

# RMSE of the medium model
sqrt(lm_medium.cv$delta[1])
```

```
## [1] 66672.84
```

```
# my model
lm1 <- glm(price ~ lotSize + livingArea + pctCollege + bedrooms + bathrooms + rooms + centralAir + wate
lm1.cv <- cv.glm(data = SaratogaHouses, lm1, K = 10)

# RMSE of my model
sqrt(lm1.cv$delta[1])
```

```
## [1] 64900.97
```

```
# RMSE improvement on the medium model
sqrt(lm_medium.cv$delta[1]) - sqrt(lm1.cv$delta[1])
```

```
## [1] 1771.873
```

```
lm <- lm(price ~ lotSize + livingArea + pctCollege + bedrooms + bathrooms + rooms + centralAir + waterf
lm.beta(lm)
```

```
##
## Call:
## lm(formula = price ~ lotSize + livingArea + pctCollege + bedrooms +
##     bathrooms + rooms + centralAir + waterfront, data = SaratogaHouses)
##
## Standardized Coefficients::
##  (Intercept)      lotSize   livingArea   pctCollege     bedrooms    bathrooms
##   0.00000000   0.05741375   0.56569250   0.04993521  -0.10582911   0.14661605
##        rooms centralAirNo waterfrontNo
##   0.08359965  -0.10446242  -0.15881708
```

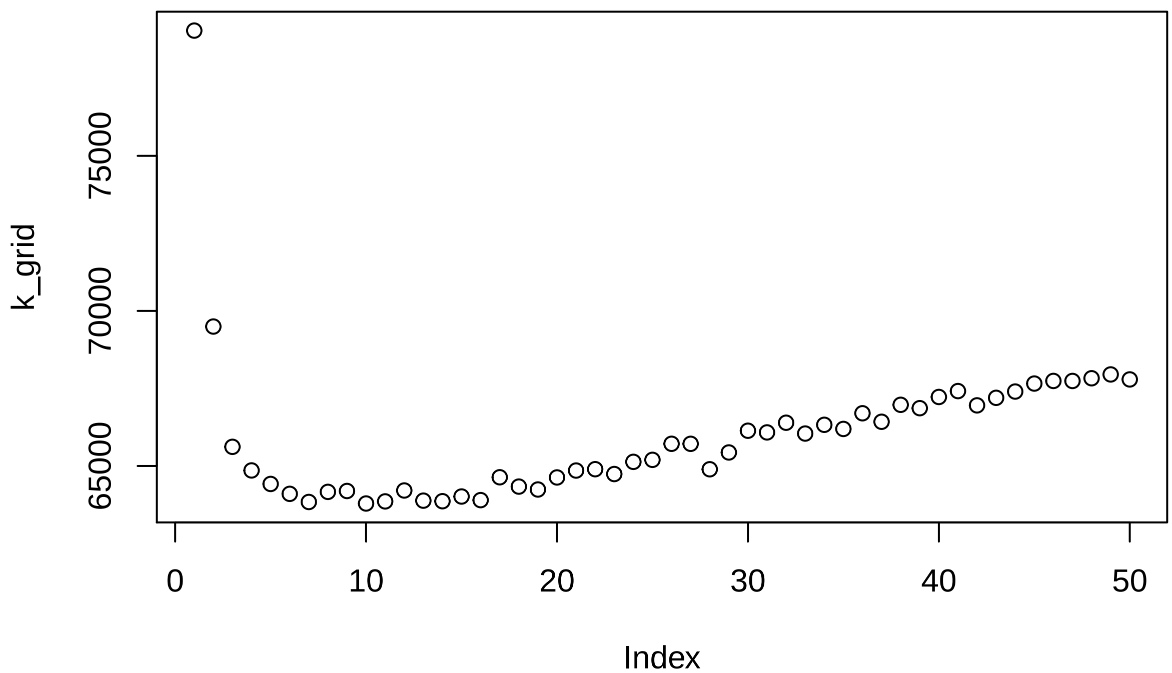
Adding the waterfront feature and removing age, fireplaces, heating, fuel, and centralAir from the medium model decreases the RMSE by about $1408. I added the waterfront feature because in my experience, waterfront properties are more expensive than their more land-locked partners. I chose to remove many of the features from the medium model because their coefficients were not statistically significantly non-zero. These features may not have been statistically significant, because they do not provide additional information for the model, or because of collinearity. In that case adding the interaction terms would helpd resolve this issue. But for this assignment, simply removing those featuers led to a significant improvement in the model's accuracy. Using the standardized coefficients we can see that livingArea, waterfront, and bathrooms have a large impact on the predictions of our model.

```
library(FNN)
library(caret)
```

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:mosaic':
##
##     dotPlot

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(mosaic)
library(doMC) # parallel computing
set.seed(100)
set.rseed(100)
data(SaratogaHouses)

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

# one-hot encode the categorical variables
dmy <- dummyVars("~.", data = SaratogaHouses)

# create data frame with new variables
data <- data.frame(predict(dmy, newdata=SaratogaHouses))

# dont use price
X <- data[,2:ncol(data)]
y <- data$price
n <- nrow(data)
train_n <- n * 0.8

# knn regression
k_grid <- seq(1,50,by=1)
k7 <- rep(0, 500)
for(k in k_grid) {
  err <- rep(0, 500)

  for(i in 1:length(err)) {
    train_ind <- sample.int(n, train_n)
    X_train <- X[train_ind,]
    X_test <- X[-train_ind,]
    y_train <- y[train_ind]
    y_test <- y[-train_ind]

    # scale train and test feature by the sd of train features
    scale_factors <- apply(X_train, 2, sd, na.rm = TRUE)
    X_train_sc <- scale(X_train, scale = scale_factors)
    X_test_sc <- scale(X_test, scale = scale_factors)

    model <- knn.reg(train = X_train_sc, test = X_test_sc, y = y_train, k = k)

    err[i] <- rmse(y_test, model$pred)
    if(k == 7) k7[i] <- err[i]
```

```
  }
  k_grid[k] <- mean(err)
}

# plot the test average RMSE
plot(k_grid)
```




```
# find the optimal K value
which.min(err)
```

```
## [1] 127
```

```
min(err)
```

```
## [1] 54498.67
```

```
# improvement on the hand crafted model
sqrt(lm1.cv$delta[1]) - min(err)
```

```
## [1] 10402.3
```

```
# t test
t.test(k7)
```

```
##
##  One Sample t-test
##
## data:  k7
## t = 288.41, df = 499, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  63404.86 64274.65
## sample estimates:
## mean of x
##  63839.75
```

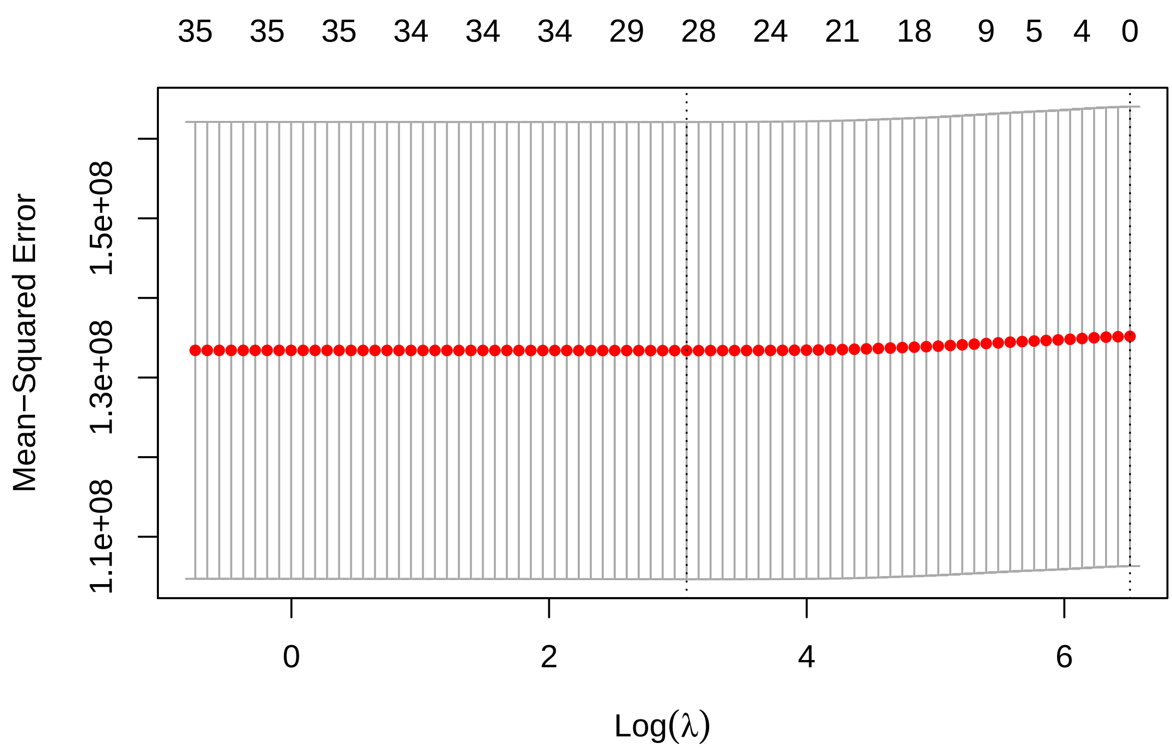
The optimal value for K is 7 with an RMSE of $63772.17, which is an improvement of about $1128.80 over my hand crafted model. The 95% confidence interval for the RMSE is (63404, 64274). -REPORT- Our model has an expected RMSE of $63772, meaning it is likely that our predictions for house price will regularly be off by about $60000. This is likely to have a large impact on the tax rate we charge each household, however I do not think it will have a large impact on the tax bracket of each house. Because of the large in-accruary of our model, I think it is important to factor in past evaluations of the house when deciding what tax rate to charge each household.

PREDICTING WHEN ARTICLES GO VIRAL

```r
library(glmnet)
```

```
## Loaded glmnet 3.0-2
```

```r
library(tidyverse)
library(boot)
library(caret)
set.seed(100)

news <- read.csv("~/Documents/R/SDS 323/SDS323-master/data/online_news.csv")
news <- dplyr::select(news, -url)

# calculate our null prediction
null_prediction <- sum(news$shares>1400)/nrow(news)

# seperate the data into features and response
X <- model.matrix(shares ~. -1, data=news)
y <- news$shares

# find the optimal lambda for our data set
cv.lasso1 <- cv.glmnet(X, y, alpha = 1, family = "gaussian", nfolds = 10)
plot(cv.lasso1)
```



```r
# K=10 CV
K <- 10
fold_id <- rep_len(1:K, nrow(X))
fold_id <- sample(fold_id)

acc<- rep(0, K)
tp<- rep(0, K)
fp<- rep(0, K)
for(i in 1:K) {
  train_set <- which(fold_id != i)
  X_train <- X[train_set,]
  X_test <- X[-train_set,]
  y_train <- y[train_set]
  y_test <- y[-train_set]
  y_test <- ifelse(y_test > 1400, 1, 0)

  l1 <- glmnet(X_train, y_train, alpha = 1, family = "gaussian", lambda = cv.lasso1$lambda.min)
  y_hat <- predict(l1, X_test)
  y_hat <- ifelse(y_hat > 1400, 1, 0)
  cm <- confusionMatrix(factor(y_hat), factor(y_test))$table
  cm
  acc[i] <- (cm[1] + cm[4])/sum(cm)
  tp[i] <- cm[4]/(cm[3] + cm[4])
  fp[i] <- cm[2]/(cm[1] + cm[2])
}
# accuracy
mean(acc)
```

```
## [1] 0.49662
```

```r
# true positive rate
mean(tp)
```

```
## [1] 0.9960667
```

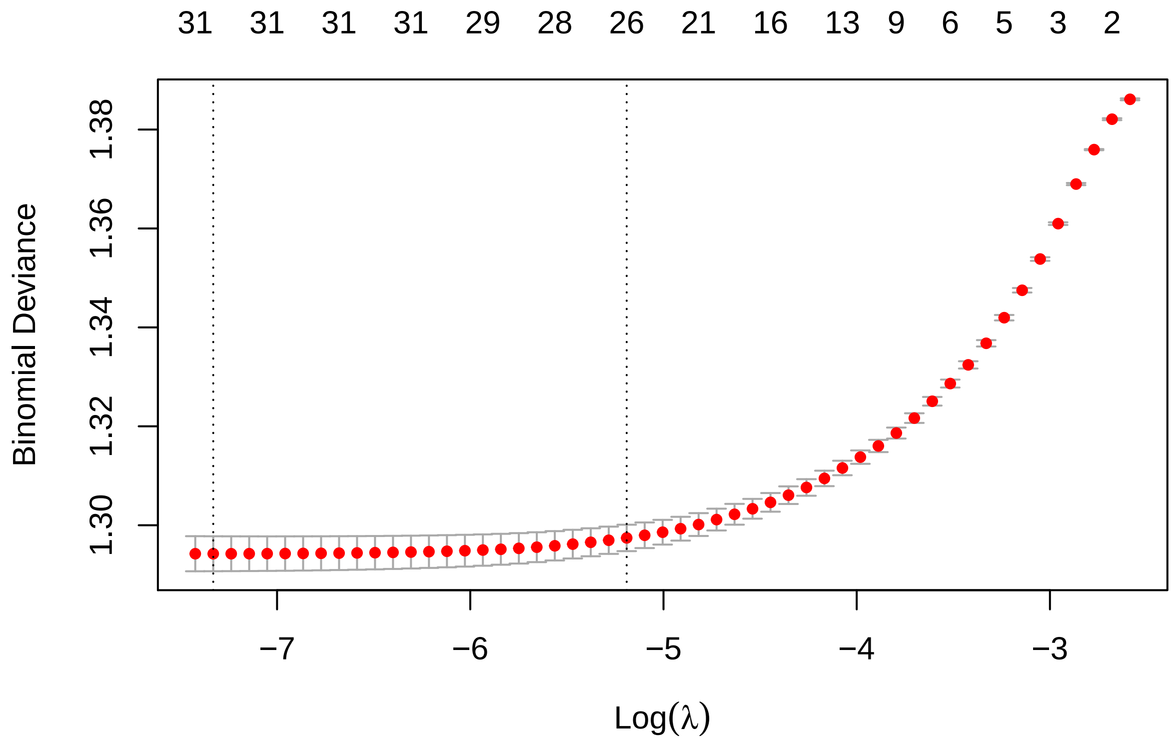
```r
# false positive rate
mean(fp)
```

```
## [1] 0.9898802
```

Training our model before thresholding y leads to a model that is less accurate than the no informaion rate, meaning it is less accurate than simply using the sample proportion of viral articles to predict viral status.

Now we preform the same analysis, but we threshold y first.

```r
set.seed(100)
viral <- news$shares
viral <- ifelse(viral > 1400, 1, 0)
y <- viral

# find the optimal lambda for the data set
X <- model.matrix(shares ~. -1, data=news)
cv.lasso2 <- cv.glmnet(X, y, alpha = 1, family = "binomial", nfolds = 10)
plot(cv.lasso2)
```




```r
# K=10 CV
K <- 10
l2_err <- rep(0, K)
fold_id <- rep_len(1:K, nrow(X))
fold_id <- sample(fold_id)

for(i in 1:K) {
  train_set <- which(fold_id != i)
  X_train <- X[train_set,]
  X_test <- X[-train_set,]
  y_train <- y[train_set]
  y_test <- y[-train_set]

  l2 <- glmnet(X_train, y_train, alpha = 1, family = "binomial", lambda = cv.lasso2$lambda.min)
  y_hat <- predict(l2, X_test)
  y_hat <- sapply(y_hat, function(x){ifelse(x > 0.5, 1, 0)})

  l2_err[i] <- mean(y_hat == y_test)
}
mean(l2_err)
```

```
## [1] 0.5907576
```

```r
# logistic model using every variable
X <- dplyr::select(news, -shares)

# K fold CV
acc<- rep(0, K)
tp<- rep(0, K)
fp<- rep(0, K)
```

```r
fold_id <- rep_len(1:K, nrow(X))
fold_id <- sample(fold_id)
for(k in 1:K) {
  train_set <- which(fold_id != k)
  X_train <- X[train_set,]
  X_test <- X[-train_set,]
  y_train <- y[train_set]
  y_test <- y[-train_set]

  model <- lm(shares ~., data = data.frame(X_train, shares = y_train))
  y_hat <- predict(model, newdata = X_test)
  y_hat <- ifelse(y_hat > 0.5, 1, 0)
  cm <- confusionMatrix(factor(y_hat), factor(y_test))$table
  acc[i] <- (cm[1] + cm[4])/sum(cm)
  tp[i] <- cm[4]/(cm[4] + cm[3])
  fp[i] <- cm[2]/(cm[1] + cm[2])
}
```

```
## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(model, newdata = X_test): prediction from a rank-deficient
## fit may be misleading
```

```r
# accuracy
mean(acc)
```

```
## [1] 0.06291625
```

```r
# true positive rate
mean(tp)
```

```
## [1] 0.06321897
```

```r
# false positive rate
mean(fp)
```

```
## [1] 0.03738648
```

KNN regression code.

```r
# library(FNN)
# library(matrixStats)
#
# colScale <- function(x,
#     center = TRUE,
#     scale = TRUE,
#     add_attr = TRUE,
#     rows = NULL,
#     cols = NULL) {
#
#     if (!is.null(rows) && !is.null(cols)) {
#         x <- x[rows, cols, drop = FALSE]
#     } else if (!is.null(rows)) {
#         x <- x[rows, , drop = FALSE]
#     } else if (!is.null(cols)) {
#         x <- x[, cols, drop = FALSE]
#     }
#
#   ################
#   # Get the column means
#   ################
#     cm = colMeans(x, na.rm = TRUE)
#   ################
#   # Get the column sd
#   ################
#     if (scale) {
#         csd = colSds(x, center = cm)
#     } else {
#         # just divide by 1 if not
#         csd = rep(1, length = length(cm))
#     }
#     if (!center) {
#         # just subtract 0
#         cm = rep(0, length = length(cm))
#     }
#     x = t( (t(x) - cm) / csd )
#     if (add_attr) {
#         if (center) {
#             attr(x, "scaled:center") <- cm
#         }
#         if (scale) {
#             attr(x, "scaled:scale") <- csd
#         }
#     }
#     return(x)
# }
#
```

```
# # KNN regression with 1se optimal lambda coefficients
# X <- model.matrix(shares ~. -1, data = news)
# l2 <- glmnet(X, y, alpha = 1, family = "binomial", lambda = cv.lasso2$lambda.1se)
#
# # get non zero coefficients
# lasso_coefs <- rownames(coef(l2))[coef(l2)[,1] == 0]
#
#
# X <- dplyr::select(news, -c(shares, lasso_coefs))
# sample_ind <- sample.int(nrow(X), 1000)
# sX <- X[sample_ind,]
# sy <- y[sample_ind]
#
#
# n <- nrow(X)
# train_n <- n*0.8
# k_grid <- seq(40,100, by = 1)
# for(k in k_grid) {
#   err <- rep(0, 5)
#   for(i in 1:length(err)) {
#     train_ind <- sample.int(n, train_n)
#     X_train <- X[train_ind,]
#     X_test <- X[-train_ind,]
#     y_train <- y[train_ind]
#     y_test <- y[-train_ind]
#
#     #scale_factors <- apply(X_train, 2, sd, na.rm = TRUE)
#     X_train_sc <- colScale(X_train, center = FALSE, scale = TRUE)
#     X_test_sc <- colScale(X_test, center = FALSE, scale = TRUE)
#     #X_train_sc <- scale(X_train, scale = scale_factors)
#     #X_test_sc <- scale(X_test, scale = scale_factors)
#
#     model <- knn.reg(X_train_sc, X_test_sc, y_train, k)
#     y_hat <- model$pred
#     y_hat <- ifelse(model$pred > 0.5, 1, 0)
#
#     err[i] <- mean(y_test == y_hat)
#     print(i)
#   }
#
#   k_grid[k] <- mean(err)
# }
#
# library(ggplot2)
# df <- data.frame(x = seq(1,100,by = 1), y = k_grid)
# ggplot(data = df) + xlim(40,100) + ylim(0.6,0.65) + geom_point(aes(x = x, y = y), data= df)
# plot(k_grid)
#
# which.max(k_grid)
# View(k_grid)
# k_grid[which.max(k_grid)]
```

Running the regression before thresholding y lead to 49.7% accuracy, slightly above the null prediction of 49.3%, which is the percentage of "viral" articles in the data set. Thresholding y before running the regression increased the accuracy to 59% an improvement of about 10% over our other model. We can see why this may be by looking at the MSE vs log(lambda) plots for each model. In the first plot, we can see there is minimal change in MSE for varying values for log(lambda), so much so that the difference in MSE corresponding to the min and max values for log(lambda) is negligable. More importantly, the confidence intervals are extremely large and thus it is possible the "optimal" value of lambda is not actually optimal. These observations are in stark contrast to those of the plot for model 2 (thresholding y before regression). In these plots, the confidence intervals are smaller and the plots look more similar to what we should expect to see. I also ran a logistic regression model using every parameter (excluding URL) and that saw an improvement to 63.7% accuracy. I also attempted to run a KNN regression, but with my computer's limited resources and the large data set (about 40000*36 data points), I could not complete the regression in any reasonable amount of time. I only used the non-zero coefficients from the 1 se lambda calculated from my previous lasso regression, I knew I may be throwing away information, but I hoped cutting out ~10 parameters would drastically increase the speed of training the model. This seemed to work, but it certainly wasn't enough to make the computation practical in a reasonable about of time. My next idea was to randomly sample a smaller portion of the data set and train the model on that data set. However this was unreliable, as the optimal K would change greately for different sample. I could fix this by increasing the sample size, but then that defeats the purpose because I run into the original problem. Finally I decided to implement a faster scale method, because from my testing that seemed to be one of the major resource drains in training the model (that along with calculating the distances for each point). I found a column scale function on the internet that seemed to be faster and implemented it. It greately increased the speed of the training algorithm. I trained the model over the entire data set 5 times for each K between 40 and 100 (the previous models made me confident that the optimal K was at least greater than 40), this took about 2 hours and was about 63.3% accurate for the optimal K. But to gain confidence in the model I would have to train it more than 5 times (most likely several hundred) for each value of K, which was still completely impractical. So I decided to scrap the idea of running a KNN regression model, but I left my code if you wanted to take a look.