# Exercise 2 Report

James Barr

3/30/2020

## KNN Practice: S-Class

### Question: How can we better predict price, given mileage, for two sub-models of the Mercedes S-Class

Figuring out the true value of a used car is a common problem for dealerships, consumers, and websites like Cars.com that attempt to give consumers accurate information on prices. For some cars, this task is harder than others. The Mercedes Benz S-Class is especially difficult to accurately value because there are so many sub-models of the S-Class. For this report, we will focus on two specific sub-models: S-Class 65 AMG and S-Class 350. Our goal is to build a predictive model for price, given mileage, for each of the sub models.

### The Data:

The data used from this report comprise almost 30k records, all taking place in the year 2014. Every listing in the data set was pulled from the secondary market. The data set includes 17 fields, but for this analysis, we will only be focusing on three fields: trim, mileage, and price. Future expansions of this report could go further and also include any of the other fields such as number of features, displacement, fuel type, year, or geographic location.

### Method: K-Nearest Neighbors

For this report, I will be using a K-Nearest Neighbors approach to building a predictive model. Simply put, K-Nearest Neighbors is a technique that looks at similar data points to predict the result (in this case price) of an unknown data point. For example, in this problem, if we are seeking to determine the price of a car with 100k miles, K-Nearest Neighbors will look at multiple previous sales of cars that had close to 100k miles to determine price.

First, I load the necessary libraries and the data set:

```r
sclass = read.csv("~/Desktop/Spring 2020 School/SDS 323/Exercise2/sclass.csv")
{
  library(tinytex)
  library(tidyverse)
  library(mosaic)
  library(FNN)
  library(foreach)
  library(class)
}
```

The first thing that I did for this data set was select only the relevant variables (price & mileage), then split it by the two sub models (65AMG & 350) into their own data sets.

```r
sclassX = dplyr::select(sclass,price, mileage)

sclass350 = sclassX[which(sclass$trim == "350"), ]
sclass65AMG = sclassX[which(sclass$trim == "65 AMG"), ]
```

The tricky part with K-Nearest Neighbors is determining what value $K$ to use. Essentially, how many similar data points to compare with. To determine the best value of $K$, the value of K that minimizes out of sample root mean squared error (RMSE), I tested $K$ values 1 through 29 to determine what value $K$ created the best model in terms of RMSE.

**Mercedes S-Class 350**

Then, I split the data so that I would have a test set and a training set.

```r
N = nrow(sclass350)
N_train = floor(0.8*N)
N_test = N - N_train
```

Then, I added in a helper function (HT/ Dr. James Scott) to help me later when I need to compute RMSE

```r
rmse = function(y, ypred) {
  sqrt(mean(data.matrix((y-ypred)^2)))
}
```

After that, I was ready to run my loop function, which ran 29 test models 500 times each and computed the RMSE for each trial of each model. The reason that I ran it 500 times was to reduce the Monte Carlo Variability that might result from random splitting of data into training sets and test sets.

```r
knn_rmse_vals = do(500)*{

  #Train Test Splits
  train_ind = sample.int(N, N_train, replace=FALSE)
  D_train = sclass350[train_ind,]
  D_test = sclass350[-train_ind,]

  X_trn_350 = D_train['mileage']
  y_trn_350 = D_train['price']
  X_tst_350 = D_test['mileage']
  y_tst_350 = D_test['price']

  #Test A Bunch of KNN Models
  {
  knn350_001 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=1)
  knn350_002 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=)
  knn350_003 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=3)
  knn350_004 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=4)
  knn350_005 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=5)
  knn350_006 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=6)
  knn350_007 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=7)
```

```
knn350_008 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=8)
knn350_009 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=9)
knn350_010 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=10)
knn350_011 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=11)
knn350_012 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=12)
knn350_013 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=13)
knn350_014 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=14)
knn350_015 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=15)
knn350_016 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=16)
knn350_017 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=17)
knn350_018 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=18)
knn350_019 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=19)
knn350_020 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=20)
knn350_021 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=21)
knn350_022 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=22)
knn350_023 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=23)
knn350_024 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=24)
knn350_025 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=25)
knn350_026 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=26)
knn350_027 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=27)
knn350_028 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=28)
knn350_029 = knn.reg(train = X_trn_350, test = X_tst_350, y = y_trn_350,k=29)
}
#Pull Prediction values
{
ypred350_1 = knn350_001$pred
ypred350_2 = knn350_002$pred
ypred350_3 = knn350_003$pred
ypred350_4 = knn350_004$pred
ypred350_5 = knn350_005$pred
ypred350_6 = knn350_006$pred
ypred350_7 = knn350_007$pred
ypred350_8 = knn350_008$pred
ypred350_9 = knn350_009$pred
ypred350_10 = knn350_010$pred
ypred350_11 = knn350_011$pred
ypred350_12 = knn350_012$pred
ypred350_13 = knn350_013$pred
ypred350_14 = knn350_014$pred
ypred350_15 = knn350_015$pred
ypred350_16 = knn350_016$pred
ypred350_17 = knn350_017$pred
ypred350_18 = knn350_018$pred
ypred350_19 = knn350_019$pred
ypred350_20 = knn350_020$pred
ypred350_21 = knn350_021$pred
ypred350_22 = knn350_022$pred
ypred350_23 = knn350_023$pred
ypred350_24 = knn350_024$pred
ypred350_25 = knn350_025$pred
ypred350_26 = knn350_026$pred
ypred350_27 = knn350_027$pred
ypred350_28 = knn350_028$pred
```

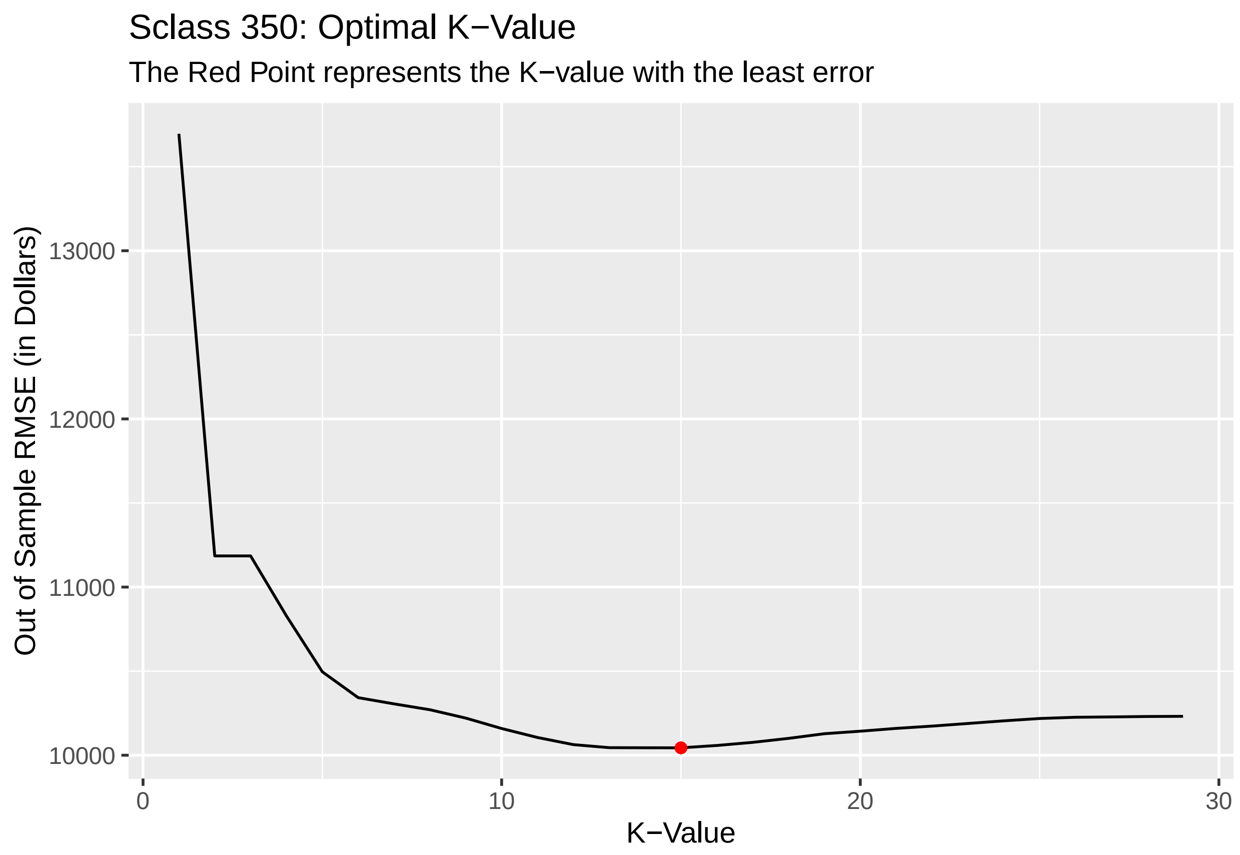
```
  ypred350_29 = knn350_029$pred
  }

  #Get RMSE for each
  {
  c(rmse(y_tst_350, ypred350_1),
  rmse(y_tst_350, ypred350_2),
  rmse(y_tst_350, ypred350_3),
  rmse(y_tst_350, ypred350_4),
  rmse(y_tst_350, ypred350_5),
  rmse(y_tst_350, ypred350_6),
  rmse(y_tst_350, ypred350_7),
  rmse(y_tst_350, ypred350_8),
  rmse(y_tst_350, ypred350_9),
  rmse(y_tst_350, ypred350_10),
  rmse(y_tst_350, ypred350_11),
  rmse(y_tst_350, ypred350_12),
  rmse(y_tst_350, ypred350_13),
  rmse(y_tst_350, ypred350_14),
  rmse(y_tst_350, ypred350_15),
  rmse(y_tst_350, ypred350_16),
  rmse(y_tst_350, ypred350_17),
  rmse(y_tst_350, ypred350_18),
  rmse(y_tst_350, ypred350_19),
  rmse(y_tst_350, ypred350_20),
  rmse(y_tst_350, ypred350_21),
  rmse(y_tst_350, ypred350_22),
  rmse(y_tst_350, ypred350_23),
  rmse(y_tst_350, ypred350_24),
  rmse(y_tst_350, ypred350_25),
  rmse(y_tst_350, ypred350_26),
  rmse(y_tst_350, ypred350_27),
  rmse(y_tst_350, ypred350_28),
  rmse(y_tst_350, ypred350_29))
  }

}
```

Then, I began to summarize and organize the results of that loop.

```
#summary of loop function findings
mean_rmse350 = colMeans(knn_rmse_vals)

#helper to create table
k_vals350 = c(1:29)
rmse_table350 = data_frame(k_vals350, mean_rmse350)
```

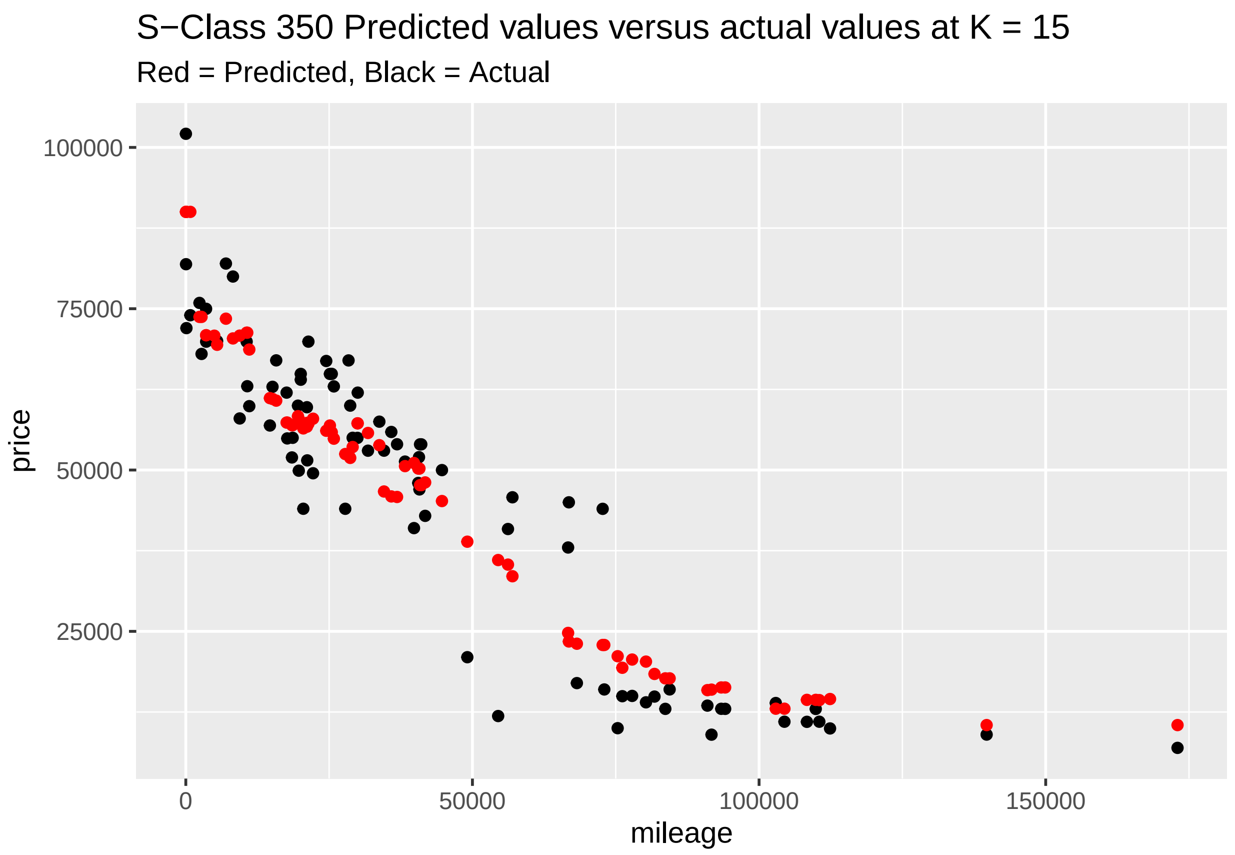
```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

Then, my data was ready to be put into a plot that exhibited the optimal value of K for this data set.



At optimal K value K $= 15$, here is a plot of the KNN predicted values versus the actual values.

```
p350_test = ggplot(data = D_test) +
  geom_point(mapping = aes(x = mileage, y = price))

KNN_Graph_350 = p350_test + geom_point(aes(x = mileage, y = ypred350_15), color='red') +
  labs(title = 'S-Class 350 Predicted values versus actual values at K = 15',
       subtitle = 'Red = Predicted, Black = Actual')

KNN_Graph_350
```



**S-Class 65 AMG**

Now, I will do the exact same thing to the S-Class 65 AMG data set that I did to the S-Class 350 data set. First, I need to split my data into a training sized set and a testing sized set.

```
N65AMG = nrow(sclass65AMG)
N_train65AMG = floor(0.8*N65AMG)
N_test65AMG = N65AMG - N_train65AMG
```

Note: I do not need to recreate my RMSE helper function because I already did it above.

Next, I run the same loop that also test 29 models 500 times each, but this time it is for the 65 AMG data set.

```
knn_rmse_vals65AMG = do(500)*{

  #Train Test Splits
  train_ind65AMG = sample.int(N65AMG, N_train65AMG, replace=FALSE)
  D_train65AMG = sclass65AMG[train_ind65AMG,]
  D_test65AMG = sclass65AMG[-train_ind65AMG,]

  X_trn_65AMG = D_train65AMG['mileage']
  y_trn_65AMG = D_train65AMG['price']
  X_tst_65AMG = D_test65AMG['mileage']
  y_tst_65AMG = D_test65AMG['price']
```

```r
#Test A Bunch of KNN Models
{
  knn65AMG_001 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=1)
  knn65AMG_002 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=)
  knn65AMG_003 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=3)
  knn65AMG_004 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=4)
  knn65AMG_005 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=5)
  knn65AMG_006 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=6)
  knn65AMG_007 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=7)
  knn65AMG_008 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=8)
  knn65AMG_009 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=9)
  knn65AMG_010 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=10)
  knn65AMG_011 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=11)
  knn65AMG_012 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=12)
  knn65AMG_013 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=13)
  knn65AMG_014 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=14)
  knn65AMG_015 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=15)
  knn65AMG_016 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=16)
  knn65AMG_017 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=17)
  knn65AMG_018 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=18)
  knn65AMG_019 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=19)
  knn65AMG_020 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=20)
  knn65AMG_021 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=21)
  knn65AMG_022 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=22)
  knn65AMG_023 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=23)
  knn65AMG_024 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=24)
  knn65AMG_025 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=25)
  knn65AMG_026 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=26)
  knn65AMG_027 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=27)
  knn65AMG_028 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=28)
  knn65AMG_029 = knn.reg(train = X_trn_65AMG, test = X_tst_65AMG, y = y_trn_65AMG,k=29)
}
#Pull Prediction values
{
  ypred65AMG_1 = knn65AMG_001$pred
  ypred65AMG_2 = knn65AMG_002$pred
  ypred65AMG_3 = knn65AMG_003$pred
  ypred65AMG_4 = knn65AMG_004$pred
  ypred65AMG_5 = knn65AMG_005$pred
  ypred65AMG_6 = knn65AMG_006$pred
  ypred65AMG_7 = knn65AMG_007$pred
  ypred65AMG_8 = knn65AMG_008$pred
  ypred65AMG_9 = knn65AMG_009$pred
  ypred65AMG_10 = knn65AMG_010$pred
  ypred65AMG_11 = knn65AMG_011$pred
  ypred65AMG_12 = knn65AMG_012$pred
  ypred65AMG_13 = knn65AMG_013$pred
  ypred65AMG_14 = knn65AMG_014$pred
  ypred65AMG_15 = knn65AMG_015$pred
  ypred65AMG_16 = knn65AMG_016$pred
  ypred65AMG_17 = knn65AMG_017$pred
  ypred65AMG_18 = knn65AMG_018$pred
  ypred65AMG_19 = knn65AMG_019$pred
```

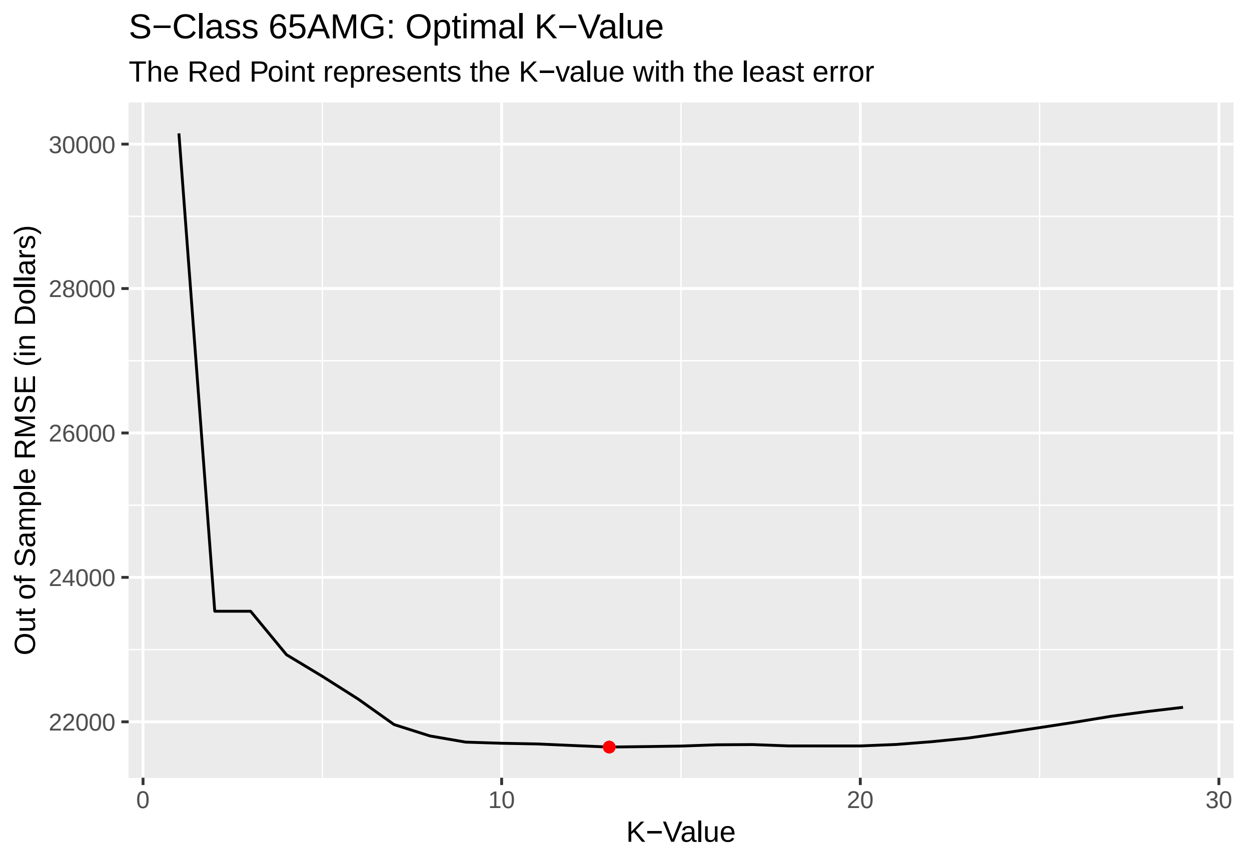
```
    ypred65AMG_20 = knn65AMG_020$pred
    ypred65AMG_21 = knn65AMG_021$pred
    ypred65AMG_22 = knn65AMG_022$pred
    ypred65AMG_23 = knn65AMG_023$pred
    ypred65AMG_24 = knn65AMG_024$pred
    ypred65AMG_25 = knn65AMG_025$pred
    ypred65AMG_26 = knn65AMG_026$pred
    ypred65AMG_27 = knn65AMG_027$pred
    ypred65AMG_28 = knn65AMG_028$pred
    ypred65AMG_29 = knn65AMG_029$pred
  }

  #Get RMSE for each
  {
    c(rmse(y_tst_65AMG, ypred65AMG_1),
      rmse(y_tst_65AMG, ypred65AMG_2),
      rmse(y_tst_65AMG, ypred65AMG_3),
      rmse(y_tst_65AMG, ypred65AMG_4),
      rmse(y_tst_65AMG, ypred65AMG_5),
      rmse(y_tst_65AMG, ypred65AMG_6),
      rmse(y_tst_65AMG, ypred65AMG_7),
      rmse(y_tst_65AMG, ypred65AMG_8),
      rmse(y_tst_65AMG, ypred65AMG_9),
      rmse(y_tst_65AMG, ypred65AMG_10),
      rmse(y_tst_65AMG, ypred65AMG_11),
      rmse(y_tst_65AMG, ypred65AMG_12),
      rmse(y_tst_65AMG, ypred65AMG_13),
      rmse(y_tst_65AMG, ypred65AMG_14),
      rmse(y_tst_65AMG, ypred65AMG_15),
      rmse(y_tst_65AMG, ypred65AMG_16),
      rmse(y_tst_65AMG, ypred65AMG_17),
      rmse(y_tst_65AMG, ypred65AMG_18),
      rmse(y_tst_65AMG, ypred65AMG_19),
      rmse(y_tst_65AMG, ypred65AMG_20),
      rmse(y_tst_65AMG, ypred65AMG_21),
      rmse(y_tst_65AMG, ypred65AMG_22),
      rmse(y_tst_65AMG, ypred65AMG_23),
      rmse(y_tst_65AMG, ypred65AMG_24),
      rmse(y_tst_65AMG, ypred65AMG_25),
      rmse(y_tst_65AMG, ypred65AMG_26),
      rmse(y_tst_65AMG, ypred65AMG_27),
      rmse(y_tst_65AMG, ypred65AMG_28),
      rmse(y_tst_65AMG, ypred65AMG_29))
  }

}
```

Next, I summarize and organize the results of that loop.

```
#summary of loop function findings
mean_rmse65AMG = colMeans(knn_rmse_vals65AMG)

#helper to create table
```

```
k_vals65AMG = c(1:29)
rmse_table65AMG = data_frame(k_vals65AMG, mean_rmse65AMG)
```

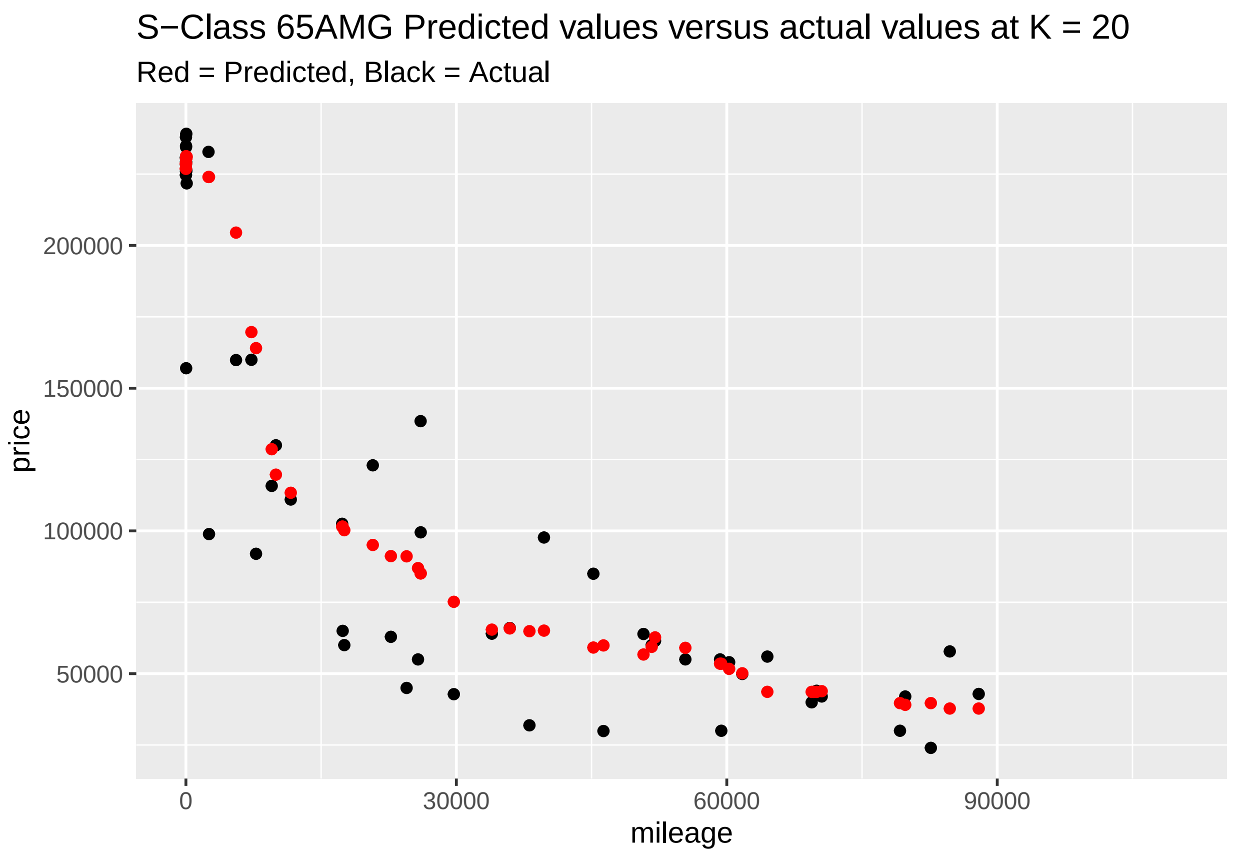
Then, I have everything that I need to create a graph that will show the optimal value of K.



At optimal K value K = 20, here is a plot of the KNN predicted values versus the actual values.

```
p65AMG_test = ggplot(data = D_test65AMG) +
  geom_point(mapping = aes(x = mileage, y = price))

KNN_Graph_65AMG = p65AMG_test + geom_point(aes(x = mileage, y = ypred65AMG_20), color='red') +
  xlim(-1,110000) +
  labs(title = 'S-Class 65AMG Predicted values versus actual values at K = 20',
       subtitle = 'Red = Predicted, Black = Actual')

KNN_Graph_65AMG
```

```
## Warning: Removed 1 rows containing missing values (geom_point).

## Warning: Removed 1 rows containing missing values (geom_point).
```



## Conclusion

In conclusion, even a simple K-Nearest Neighbors provides a solid technique for predicting the price of a car given mileage. I was rather surprised that the S-Class 65AMG had a larger optimal K-Value than the S-Class 350. I figured that given the fact that there were so many fewer observations in the 65AMG set, I assumed that the 65AMG set would have less. However, upon looking at the fitted graphs, it becomes clear that the 65AMG set has lots of variance within the data points, so a high K-value proves very useful for creating a smooth curve that does a reasonably good job of predicting price given mileage.

## Saratoga House Prices

### Question: How can we best predict house prices given a large set of variables.

Companies like Zillow have made an art form out of using data science to predict home prices. However, the value of a home matters to more people than just buyers and sellers; it also matters to the local taxing authorities. Property values directly affect the taxation of the homeowners and the revenue of the state, county, and/or city.

### The Data

The data set has over 1,700 records of house prices and their features in Saratoga, New York. The data set has 16 different variables, some categorical and some numerical.

## Methods

### Linear Regression

Linear Regression is a methodology that allows the model-makers to input what variables they think are important in predicting results. One disadvantage of this methodology is that the model-maker basically needs to make their best guess as to what variables are important and what interactions of those variables they think are important. Including every possible interaction would be a waste of computational power and would likely find connections that do not make much sense. One advantage of this method is that linear regressions is generally pretty easy to interpret and see the direct impact of certain components on the target variable (price).

### K-Nearest Neighbors

Simply put, K-Nearest Neighbors is a technique that looks at similar data points to predict the result (in this case price) of an unknown data point. For example, in this problem, if we are seeking to determine the price of a house based on several factors, K-Nearest Neighbors will look at the known prices of houses that are the most similar based on the factors selected to estimate the unknown price.

### Setting up for analyzation

```
#Load data
data(SaratogaHouses)

#Necessary Constants
n = nrow(SaratogaHouses)
n_train = round(0.8*n)  # round to nearest integer
n_test = n - n_train

#Helper Function for calculating rmse (the amount that our estimates are off by)
rmse = function(y, yhat) {
  sqrt( mean( (y - yhat)^2 ) )
}
```

## Results

###Linear Regression Here is the model that was pre-provided that we are trying to improve upon:

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

I added a bunch of interactions and variables that I think will matter. Here is what I came up that I thought would improve upon the given model:

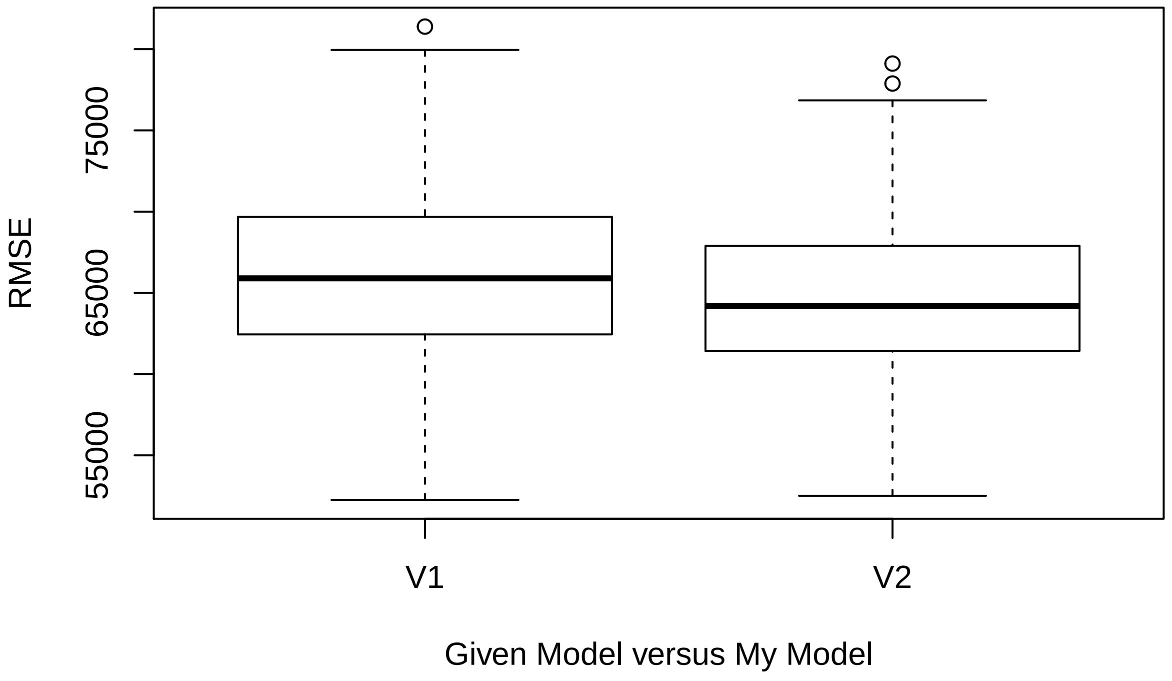
```
lm_mine = lm(price ~ lotSize + lotSize*waterfront + age + livingArea + pctCollege + bedrooms*bathrooms
               fireplaces + bathrooms + rooms + heating + fuel + centralAir + rooms*fireplaces, data=Sa
```

Now let's test using Root Mean Squared Error to see if it actually did better. Note, this loops tests both functions 250 times to get lessen the random sample variation that could have thrown our results off.

```r
lm_rmse_vals = do(250)*{

  # 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
  lm1 = update(lm_medium, data = saratoga_train)
  lmME = update(lm_mine, data = saratoga_train)

  # Predictions out of sample
  yhat_test1 = predict(lm1, saratoga_test)
  yhat_testME = predict(lmME, saratoga_test)

  c(rmse(saratoga_test$price, yhat_test1),
    rmse(saratoga_test$price, yhat_testME))
}
linear_comp = colMeans(lm_rmse_vals)
```

Now, here is a box-plot that compares the two models in terms of RMSE (lower is better):

```r
# My model made a clear improvement over the given model
boxplot(lm_rmse_vals, xlab ='Given Model versus My Model', ylab = 'RMSE')
```



**K Nearest Neighbors**

First, I need to convert the categorical variables (like heating) into dummy codes so that KNN can process their impact.

```
#Recreate categorical variables as dummy codes
SaratogaHouses$is_waterfront = ifelse(SaratogaHouses$waterfront == 'Yes', 1, 0)
SaratogaHouses$is_gas = ifelse(SaratogaHouses$fuel == 'gas', 1, 0)
SaratogaHouses$is_oil = ifelse(SaratogaHouses$fuel == 'oil', 1, 0)
SaratogaHouses$is_hotair = ifelse(SaratogaHouses$heating == 'hot air', 1, 0)
SaratogaHouses$is_steam = ifelse(SaratogaHouses$heating == 'hot water/steam', 1, 0)
SaratogaHouses$is_CentralAir = ifelse(SaratogaHouses$centralAir == 'Yes', 1, 0)
```

Then I need to rename my variables of interest into something that is easy to call:

```
X = dplyr::select(SaratogaHouses, lotSize, is_waterfront, age, livingArea,
                    pctCollege, bedrooms, bathrooms, fireplaces,
                    rooms, is_steam, is_hotair, is_gas, is_oil, is_CentralAir)
y = SaratogaHouses$price
```

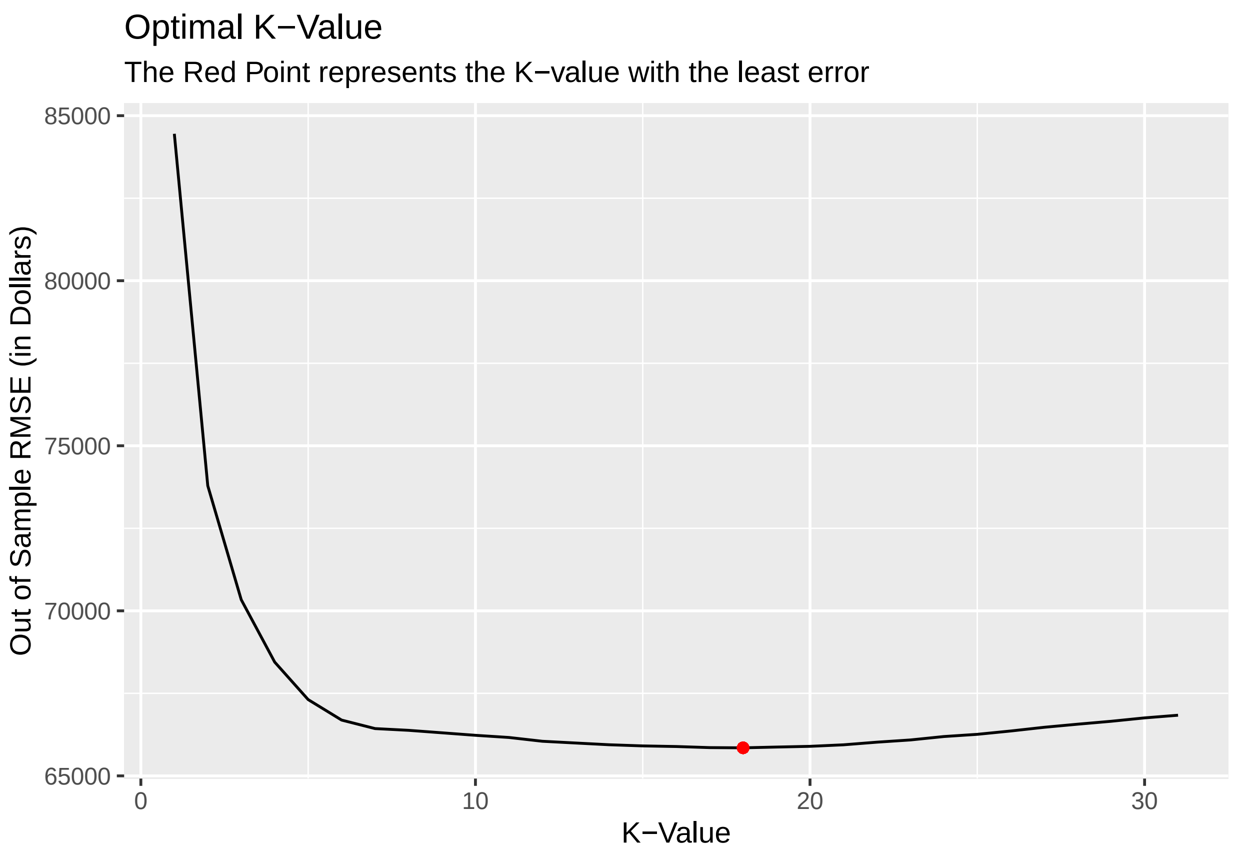
Next, I need to make the actual K-Nearest Neighbors regression and be careful to properly scale factors such that variables with large magnitudes (like land size) don't confuse the regression and totally overrun smaller magnitude variables that are also important (like number of bedrooms). I'll do so by scaling the variables such that the regression will focus on standard deviation of the variable (how unusual it is compared to the average). This program will help us figure out what K value works best.

```
rmse_vals = do(250)*{
    train_ind = sample.int(n, n_train)
    X_train = X[train_ind,]
    X_test = X[-train_ind,]
    y_train = y[train_ind]
    y_test = y[-train_ind]

    # scale the training set features
    scale_factors = apply(X_train, 2, sd)
    X_train_sc = scale(X_train, scale=scale_factors)

    # scale test set using same factors
    X_test_sc = scale(X_test, scale=scale_factors)

    #Fit to training data
   kn = {
    k1 = knn.reg(X_train_sc, X_test_sc, y_train, k=1)
    k2 = knn.reg(X_train_sc, X_test_sc, y_train, k=2)
    k3 = knn.reg(X_train_sc, X_test_sc, y_train, k=3)
    k4 = knn.reg(X_train_sc, X_test_sc, y_train, k=4)
    k5 = knn.reg(X_train_sc, X_test_sc, y_train, k=5)
    k6 = knn.reg(X_train_sc, X_test_sc, y_train, k=6)
    k7 = knn.reg(X_train_sc, X_test_sc, y_train, k=7)
    k8 = knn.reg(X_train_sc, X_test_sc, y_train, k=8)
    k9 = knn.reg(X_train_sc, X_test_sc, y_train, k=9)
    k10 = knn.reg(X_train_sc, X_test_sc, y_train, k=10)
    k11 = knn.reg(X_train_sc, X_test_sc, y_train, k=11)
    k12 = knn.reg(X_train_sc, X_test_sc, y_train, k=12)
    k13 = knn.reg(X_train_sc, X_test_sc, y_train, k=13)
    k14 = knn.reg(X_train_sc, X_test_sc, y_train, k=14)
    k15 = knn.reg(X_train_sc, X_test_sc, y_train, k=15)
    k16 = knn.reg(X_train_sc, X_test_sc, y_train, k=16)
```

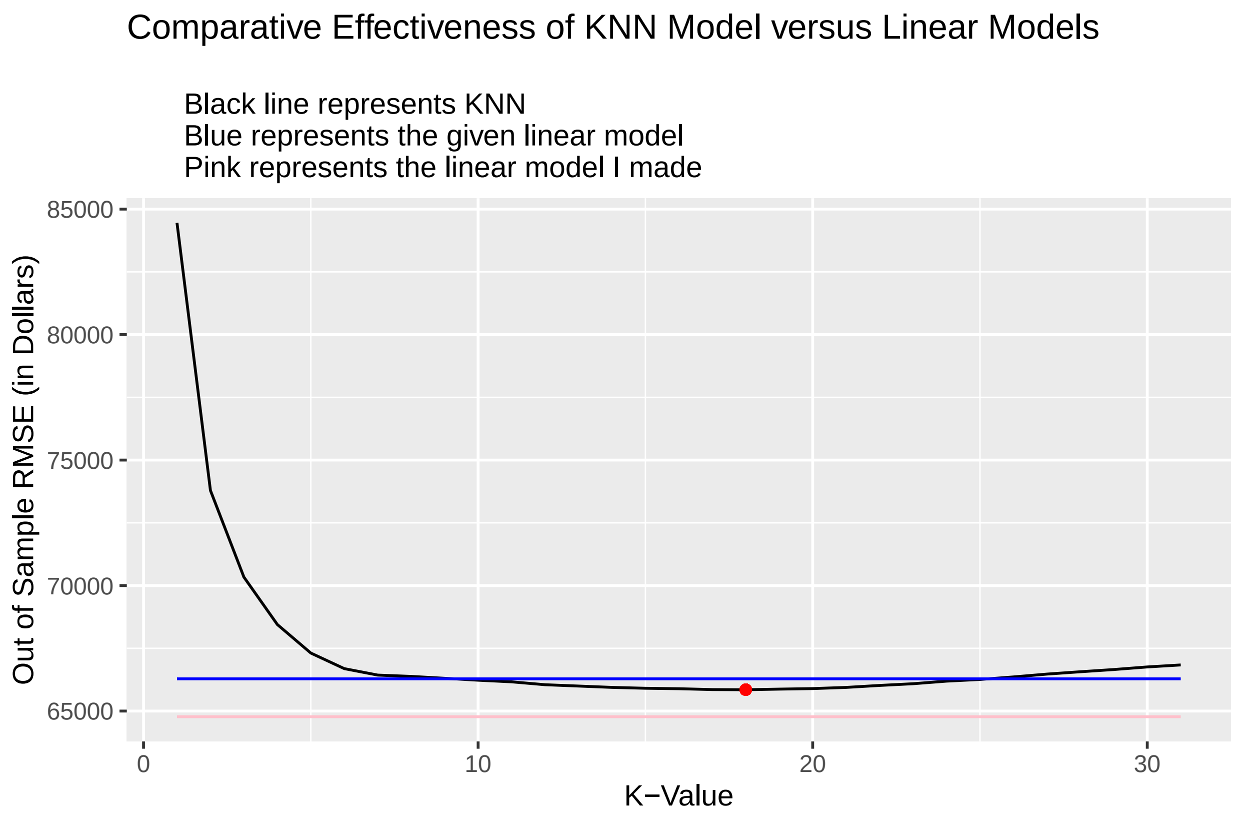
```
  k17 = knn.reg(X_train_sc, X_test_sc, y_train, k=17)
  k18 = knn.reg(X_train_sc, X_test_sc, y_train, k=18)
  k19 = knn.reg(X_train_sc, X_test_sc, y_train, k=19)
  k20 = knn.reg(X_train_sc, X_test_sc, y_train, k=20)
  k21 = knn.reg(X_train_sc, X_test_sc, y_train, k=21)
  k22 = knn.reg(X_train_sc, X_test_sc, y_train, k=22)
  k23 = knn.reg(X_train_sc, X_test_sc, y_train, k=23)
  k24 = knn.reg(X_train_sc, X_test_sc, y_train, k=24)
  k25 = knn.reg(X_train_sc, X_test_sc, y_train, k=25)
  k26 = knn.reg(X_train_sc, X_test_sc, y_train, k=26)
  k27 = knn.reg(X_train_sc, X_test_sc, y_train, k=27)
  k28 = knn.reg(X_train_sc, X_test_sc, y_train, k=28)
  k29 = knn.reg(X_train_sc, X_test_sc, y_train, k=29)
  k30 = knn.reg(X_train_sc, X_test_sc, y_train, k=30)
  k31 = knn.reg(X_train_sc, X_test_sc, y_train, k=31)
  }



  # pull prediction values
 ypred =  {
  ypred_k1 = k1$pred
  ypred_k2 = k2$pred
  ypred_k3 = k3$pred
  ypred_k4 = k4$pred
  ypred_k5 = k5$pred
  ypred_k6 = k6$pred
  ypred_k7 = k7$pred
  ypred_k8 = k8$pred
  ypred_k9 = k9$pred
  ypred_k10 = k10$pred
  ypred_k11 = k11$pred
  ypred_k12 = k12$pred
  ypred_k13 = k13$pred
  ypred_k14 = k14$pred
  ypred_k15 = k15$pred
  ypred_k16 = k16$pred
  ypred_k17 = k17$pred
  ypred_k18 = k18$pred
  ypred_k19 = k19$pred
  ypred_k20 = k20$pred
  ypred_k21 = k21$pred
  ypred_k22 = k22$pred
  ypred_k23 = k23$pred
  ypred_k24 = k24$pred
  ypred_k25 = k25$pred
  ypred_k26 = k26$pred
  ypred_k27 = k27$pred
  ypred_k28 = k28$pred
  ypred_k29 = k29$pred
  ypred_k30 = k30$pred
  ypred_k31 = k31$pred
  }
```

```
    #rmse vals

       c(
      rmse(y_test, ypred_k1),
      rmse(y_test, ypred_k2),
      rmse(y_test, ypred_k3),
      rmse(y_test, ypred_k4),
      rmse(y_test, ypred_k5),
      rmse(y_test, ypred_k6),
      rmse(y_test, ypred_k7),
      rmse(y_test, ypred_k8),
      rmse(y_test, ypred_k9),
      rmse(y_test, ypred_k10),
      rmse(y_test, ypred_k11),
      rmse(y_test, ypred_k12),
      rmse(y_test, ypred_k13),
      rmse(y_test, ypred_k14),
      rmse(y_test, ypred_k15),
      rmse(y_test, ypred_k16),
      rmse(y_test, ypred_k17),
      rmse(y_test, ypred_k18),
      rmse(y_test, ypred_k19),
      rmse(y_test, ypred_k20),
      rmse(y_test, ypred_k21),
      rmse(y_test, ypred_k22),
      rmse(y_test, ypred_k23),
      rmse(y_test, ypred_k24),
      rmse(y_test, ypred_k25),
      rmse(y_test, ypred_k26),
      rmse(y_test, ypred_k27),
      rmse(y_test, ypred_k28),
      rmse(y_test, ypred_k29),
      rmse(y_test, ypred_k30),
      rmse(y_test, ypred_k31)
    )
}
k_vals = c(1:31)
mean_rmse = colMeans(rmse_vals)

rmse_table = data_frame(x= k_vals, y=mean_rmse)

best_k = rmse_table[which.min(mean_rmse),]
```

Now, let's create a plot to visualize how well different K values do:



Now that we have the optimal value of K, we can compare the best K-Nearest Neighbors plot to the earlier Linear Models:



urprisingly, the linear model that I built actually did a better job of predicting the true house price than the KNN Model

### Conclusion

In the end, the best model was the hand-built linear model. The hand built linear model was usually within 65,000 dollars of the exact home price, which was about 1,000 dollars better than the runner up model, a K-Nearest Neighbors regression with $K = 14$. Therefore, when it comes to determining the proper valuation of a property for tax purposes, the tax authority's best option is to use the linear model that I built.

## Predicting Whether or not an Article will go Viral

### Question: Can we create a model that will give us a better idea of whether or not an article will go viral?

*Viral* in this context means that the article will be shared more than 1400 times. Obviously news outlets have a desire for their articles to be read by as many people as possible, but articles that meet this definition of *viral* are especially important to companies like Mashable because it impacts the amount of money they receive from advertisers.

### The Data

The data for this analysis comes from Mashable and has almost 40k records which are individual articles, their shares, and 36 features that describe the article. For a full description of what the features are click here (H/T Dr. James Scott). The summary is that they range from counts of how long the articles and titles are to ratings on how positive or negative the articles/titles sentiments are.

```
online_news = read.csv("~/Desktop/Spring 2020 School/SDS 323/Exercise2/online_news.csv")
```

## The Method

### Regress with KNN then Threshold

The first method that we will use approaches the problem as a K-Nearest Neighbors Regression, that will give us estimated shares based on the included features. Once it gives us an estimated number of shares, we can then threshold those numbers and say that the articles estimated to have over 1400 shares will go viral and those with less than 1400 shares will not go viral.

First, here is all the necessary libraries, some useful constants, and a helper function:

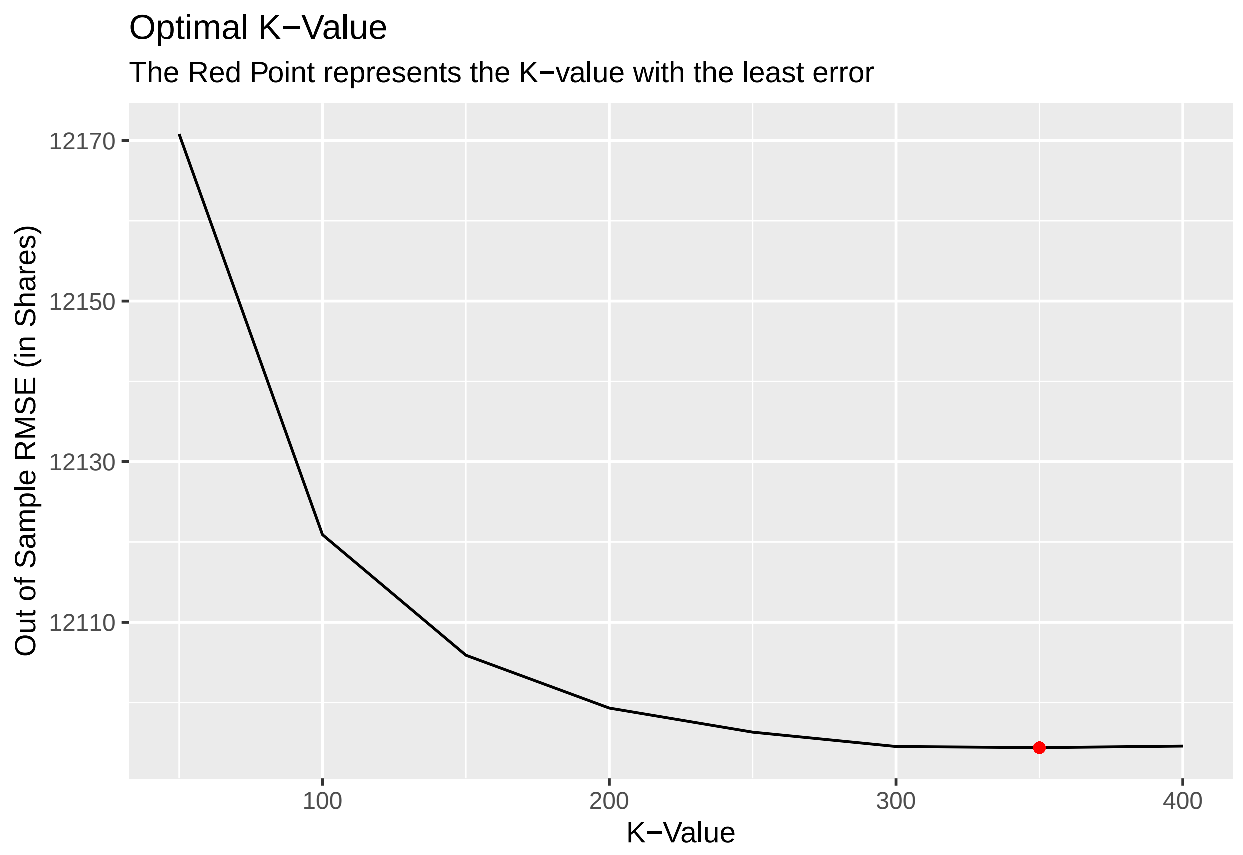
```
#Necessary Libraries
{
library(mosaic)
library(tidyverse)
library(FNN)
}

#useful constants
n = nrow(online_news)
n_train = round(0.8*n)  # round to nearest integer
n_test = n - n_train

#RMSE Helper function to test how well the KNN models do
rmse = function(y, yhat) {
  sqrt( mean( (y - yhat)^2 ) )
}

#Variable Selection
MashX = dplyr::select(online_news, title_subjectivity,
                      abs_title_sentiment_polarity, is_weekend,
                      self_reference_avg_sharess, n_tokens_content,
                      num_imgs, num_videos)

Mashy = online_news$shares
```

Then, we need to decide what value $K$ to use for the K-Nearest Neighbors Regression. To do that, I have made a loop that will test eight different values of $K$ 50 times each. The K-value with the lowest RMSE is the $K$ that does the best job predicting actual article shares.

```
rmse_vals = do(50)*{

  # re-split into train and test cases with the same sample sizes
  train_ind = sample.int(n, n_train, replace=FALSE)
  trainX = MashX[train_ind,]
  testX = MashX[-train_ind,]
  trainy = Mashy[train_ind]
  testy = Mashy[-train_ind]

  #Scale the different factors
  scale_factor = apply(trainX,2,sd)
  scale_trainX = scale(trainX, scale_factor)
  scale_testX = scale(testX, scale_factor)

   # Fit to the training data

  k50 = knn.reg(train = scale_trainX, test = scale_testX, y = trainy, k = 50)
  k100 = knn.reg(train = scale_trainX, test = scale_testX, y = trainy, k = 100)
  k150 = knn.reg(train = scale_trainX, test = scale_testX, y = trainy, k = 150)
  k200 = knn.reg(train = scale_trainX, test = scale_testX, y = trainy, k = 200)
  k250 = knn.reg(train = scale_trainX, test = scale_testX, y = trainy, k = 250)
  k300 = knn.reg(train = scale_trainX, test = scale_testX, y = trainy, k = 300)
  k350 = knn.reg(train = scale_trainX, test = scale_testX, y = trainy, k = 350)
  k400 = knn.reg(train = scale_trainX, test = scale_testX, y = trainy, k = 400)

  # Predictions out of sample
  k50pred = k50$pred
  k100pred = k100$pred
  k150pred = k150$pred
  k200pred = k200$pred
  k250pred = k250$pred
  k300pred = k300$pred
  k350pred = k350$pred
  k400pred = k400$pred

  c(rmse(k50pred, testy),
    rmse(k100pred, testy),
    rmse(k150pred, testy),
    rmse(k200pred, testy),
    rmse(k250pred, testy),
    rmse(k300pred, testy),
    rmse(k350pred, testy),
    rmse(k400pred, testy)
    )

# noticeable improvement over the starting point!
}
mean_rmse = colMeans(rmse_vals)

k_vals = c(50,100,150,200,250,300,350,400)

rmse_table = data.frame(x = k_vals, y = mean_rmse)
```

Now that I've created that loop, here is a plot of its results:

```
K_Selection_Graph =
  rmse_table %>%
  ggplot +
  geom_path(mapping = aes(x = k_vals, y = mean_rmse))+
  geom_point(aes(x=k_vals[which.min(mean_rmse)], y = min(mean_rmse)),color = 'red') +
  labs(x = 'K-Value', y = 'Out of Sample RMSE (in Shares)', title = 'Optimal K-Value', subtitle = 'The I

K_Selection_Graph
```



Then, now that we have figured out the optimal K-value, we can threshold at that K-value to create a confusion matrix. I looped this code 100 times to minimize Monte Carlo Variability.

```
viral = ifelse(online_news$shares> 1400, 1,0)

  out = do(50)*{
    train_ind = sample.int(n, n_train, replace=FALSE)
    trainX = MashX[train_ind,]
    testX = MashX[-train_ind,]
    ytrain = viral[train_ind]
    ytest = viral[-train_ind]

    #Scale the different factors
    scale_factor = apply(trainX,2,sd)
    scale_trainX = scale(trainX, scale_factor)
    scale_testX = scale(testX, scale_factor)

    # Fit KNN models (notice the odd values of K)
    knn_try = class::knn(train=scale_trainX, test= scale_testX, cl=ytrain, k=k_vals[which.min(mean_rmse)



    # Calculating classification errors
    sum(knn_try != ytest)/n_test
  }
```

Now let's get the key summary statistics:

```
  overall_error_rate01 = mean(out$result)
  confusion_matrix01 = table(ytest, knn_try)
  true_postitive_rate01 = confusion_matrix01[2,2]/sum(ytest)
  false_positive_rate01 = confusion_matrix01[1,2]/(confusion_matrix01[1,1] +confusion_matrix01[1,2])
```

### Overall Error Rate - Regress First

```
## [1] 0.4098499
```

### Confusion Matrix - Regress First

```
##      knn_try
## ytest    0    1
##     0 2854 1129
##     1 2146 1800
```

### True Positive Rate - Regress First

```
## [1] 0.4561581
```

### False Positive Rate - Regress First

```
## [1] 0.2834547
```

### Classification Approach

Classification is another way to solve this problem, imagining that all that matters is whether the article is one thing or the other. In this case, all we care about is whether an article is *viral* or *not viral*. In this case, it also saves a lot of time and computational power because it requires less steps.

The only new things that we have to do to get this to run is a new loop, this time just for classification.

```
k_grid02 = seq(51, 401, by=50)
err_grid02 = foreach(k = k_grid02,  .combine='c') %do% {
  out02 = do(50)*{
    train_ind02 = sample.int(n, n_train, replace=FALSE)
    trainX02 = MashX[train_ind02,]
    testX02 = MashX[-train_ind02,]
    ytrain02 = viral[train_ind02]
    ytest02 = viral[-train_ind02]

    #Scale the different factors
    scale_factor02 = apply(trainX02,2,sd)
    scale_trainX02 = scale(trainX02, scale_factor)
    scale_testX02 = scale(testX02, scale_factor)

    # Fit KNN models (notice the odd values of K)
    knn_try02 = class::knn(train=scale_trainX02, test= scale_testX02, cl=ytrain02, k=k)
    # Calculating classification errors
    sum(knn_try02 != ytest02)/n_test
  }
  mean(out02$result)
}
```

Then, similar to the other approach, we can pull the key summary statistics from there:

### Overall Error Rate - Classification Method

```
## [1] 0.4103317
```

### Confusion Matrix - Classification Method

```
##        knn_try02
## ytest02    0    1
##       0 2850 1153
##       1 2109 1817
```

### True Positive Rate - Classification Method

```
## [1] 0.462812
```

### False Positive Rate - Classification Method

```
## [1] 0.288034
```

### IMPORTANT NOTE: What If I just guessed whether or not an article would go viral?

You'll recall that we defined *viral* as more than 1400 shares. Well, it just so happens that the median amount of shares for this data set is **also** 1400 shares. Therefore, 50% of the articles published went viral, and 50% did not.

```
median(online_news$shares)
```

```
## [1] 1400
```

Knowing this, if I just guessed that all of the articles would go viral, I would be correct **50%** of the time; this means that my overall error rate would also be 50%.

Conversely, if I guessed that none of the articles would go viral, I would also have an error rate of 50%.

## Conclusion

There is almost no difference in the performance of the two different sets of approaches. This is likely because both approaches relied on the same kind of approach (K-Nearest Neighbors), and both approaches are fundamentally trying to do the same thing - get an idea on the number of shares that an article will receive. They use different mechanisms, and the classification approach was definitely easier to create, but in terms of performance they performed extremely similarly.

Somewhat unfortunately, our best model only made a modest (less than 8%) improvement over the most naive model (a guess) in terms of error rate. That being said, a modest improvement is still an improvement and we are better off than where we started.