Project 2 Final

Adam Kikuta

2022-12-06

Problem

For this project, we were tasked with helping Jacob Kawalski launch a real estate business and understand the market.

Objective

We set out to predict the prices of new homes based on the selling prices of previous homes that were up for sale.

Describing the Data

The data set provides details about the year the house was built, amount of bedrooms/bathrooms, square feet, view, zip code, etc. The three of us all took out the ID, day, day of week, latitude, and longitude variables because we did not think they were relevant in predicting house prices. Then, individually we selected variables from those that were left to make our predictions. Lastly, we normalized and factorized the variables, so the code can run smoothly.

KNN Model

Load Data

```
df <- read.csv("house_5.csv", header = TRUE)
df$new_price <- ifelse(df$price <= mean(df$price), "low", "high")
AD_df <- df [ , -c(1:2)]
AD_df2 <- AD_df[ , -c(3:4, 20:21)]
AD_df2 <- drop_na(AD_df2)

head(AD_df)
```

```
##   Year Month Day day_of_week   price bedrooms bathrooms sqft_living sqft_lot
## 1 2014     7  29           2 3100000        5      5.25        5090    23669
## 2 2014     8  12           2  305000        4      2.25        2050    12581
## 3 2014     6   2           1  585083        5      2.75        2910    36250
```

```
## 4 2015     2  20           5 1050000        4      3.00        3080    10757
## 5 2015     3   9           1  325000        3      2.25        1480    97138
## 6 2015     4   6           1 1040000        3      1.75        1900     9375
##   floors waterfront view condition grade sqft_above sqft_basement yr_built
## 1    2.0          0    0         3    12       5090             0     2006
## 2    2.0          0    0         4     7       2050             0     1978
## 3    1.0          0    0         3     8       1590          1320     1977
## 4    2.0          0    3         5     8       3080             0     1961
## 5    1.5          0    0         3     7       1480             0     1984
## 6    1.0          0    1         4     8       1330           570     1941
##   yr_renovated zipcode     lat     long new_price
## 1            0   98004 47.6297 -122.216      high
## 2            0   98002 47.3215 -122.204       low
## 3            0   98075 47.5916 -122.076      high
## 4            0   98006 47.5671 -122.159      high
## 5            0   98010 47.3317 -121.927       low
## 6            0   98115 47.6821 -122.273      high
```

```
str(AD_df2)
```

```
## 'data.frame':    15053 obs. of  18 variables:
##  $ Year         : int  2014 2014 2014 2015 2015 2015 2014 2014 2014 2014 ...
##  $ Month        : int  7 8 6 2 3 4 11 6 12 9 ...
##  $ price        : num  3100000 305000 585083 1050000 325000 ...
##  $ bedrooms     : int  5 4 5 4 3 3 4 3 6 3 ...
##  $ bathrooms    : num  5.25 2.25 2.75 3 2.25 1.75 2 1.75 2.5 2.5 ...
##  $ sqft_living  : int  5090 2050 2910 3080 1480 1900 2280 1960 3370 1484 ...
##  $ sqft_lot     : int  23669 12581 36250 10757 97138 9375 7200 6380 15625 1761 ...
##  $ floors       : num  2 2 1 2 1.5 1 1 1 1 3 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 3 0 1 0 0 0 0 ...
##  $ condition    : int  3 4 3 5 3 4 4 4 3 3 ...
##  $ grade        : int  12 7 8 8 7 8 7 7 8 7 ...
##  $ sqft_above   : int  5090 2050 1590 3080 1480 1330 2280 980 1770 1484 ...
##  $ sqft_basement: int  0 0 1320 0 0 570 0 980 1600 0 ...
##  $ yr_built     : int  2006 1978 1977 1961 1984 1941 1956 1939 1964 2003 ...
##  $ yr_renovated : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98004 98002 98075 98006 98010 98115 98133 98115 98166 98115 ...
##  $ new_price    : chr  "high" "low" "high" "high" ...
```

To begin, I loaded the data set. In order to run the kNN model, I needed to create an additional column for "new_price", which would ultimately take the "price" variable from the original data set then categorize it as either "low" or "high". From there, I dropped the unnecessary variables, which in this case were the first 4 columns and the last two before our newly created column. Because a kNN model cannot run with missing values, I added a code to drop any missing values.

Factorizing Data

```
str(AD_df2)
```

```
## 'data.frame':    15053 obs. of  18 variables:
```

```
##  $ Year         : int  2014 2014 2014 2015 2015 2015 2014 2014 2014 2014 ...
##  $ Month        : int  7 8 6 2 3 4 11 6 12 9 ...
##  $ price        : num  3100000 305000 585083 1050000 325000 ...
##  $ bedrooms     : int  5 4 5 4 3 3 4 3 6 3 ...
##  $ bathrooms    : num  5.25 2.25 2.75 3 2.25 1.75 2 1.75 2.5 2.5 ...
##  $ sqft_living  : int  5090 2050 2910 3080 1480 1900 2280 1960 3370 1484 ...
##  $ sqft_lot     : int  23669 12581 36250 10757 97138 9375 7200 6380 15625 1761 ...
##  $ floors       : num  2 2 1 2 1.5 1 1 1 1 3 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 3 0 1 0 0 0 0 ...
##  $ condition    : int  3 4 3 5 3 4 4 4 3 3 ...
##  $ grade        : int  12 7 8 8 7 8 7 7 8 7 ...
##  $ sqft_above   : int  5090 2050 1590 3080 1480 1330 2280 980 1770 1484 ...
##  $ sqft_basement: int  0 0 1320 0 0 570 0 980 1600 0 ...
##  $ yr_built     : int  2006 1978 1977 1961 1984 1941 1956 1939 1964 2003 ...
##  $ yr_renovated : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98004 98002 98075 98006 98010 98115 98133 98115 98166 98115 ...
##  $ new_price    : chr  "high" "low" "high" "high" ...
```

```
AD_df2$Month <- as.factor(AD_df2$Month)
AD_df2$waterfront <- factor(AD_df2$waterfront,
                            levels = c("0", "1"),
                            labels = c("No", "Yes"))
AD_df2$new_price <- factor(AD_df2$new_price,
                            levels = c("low", "high"),
                            labels = c("low", "high"))
AD_df2$zipcode <- as.factor(AD_df2$zipcode)
str(AD_df2)
```

```
## 'data.frame':    15053 obs. of  18 variables:
##  $ Year         : int  2014 2014 2014 2015 2015 2015 2014 2014 2014 2014 ...
##  $ Month        : Factor w/ 12 levels "1","2","3","4",..: 7 8 6 2 3 4 11 6 12 9 ...
##  $ price        : num  3100000 305000 585083 1050000 325000 ...
##  $ bedrooms     : int  5 4 5 4 3 3 4 3 6 3 ...
##  $ bathrooms    : num  5.25 2.25 2.75 3 2.25 1.75 2 1.75 2.5 2.5 ...
##  $ sqft_living  : int  5090 2050 2910 3080 1480 1900 2280 1960 3370 1484 ...
##  $ sqft_lot     : int  23669 12581 36250 10757 97138 9375 7200 6380 15625 1761 ...
##  $ floors       : num  2 2 1 2 1.5 1 1 1 1 3 ...
##  $ waterfront   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ view         : int  0 0 0 3 0 1 0 0 0 0 ...
##  $ condition    : int  3 4 3 5 3 4 4 4 3 3 ...
##  $ grade        : int  12 7 8 8 7 8 7 7 8 7 ...
##  $ sqft_above   : int  5090 2050 1590 3080 1480 1330 2280 980 1770 1484 ...
##  $ sqft_basement: int  0 0 1320 0 0 570 0 980 1600 0 ...
##  $ yr_built     : int  2006 1978 1977 1961 1984 1941 1956 1939 1964 2003 ...
##  $ yr_renovated : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : Factor w/ 70 levels "98001","98002",..: 4 2 39 6 9 50 58 50 64 50 ...
##  $ new_price    : Factor w/ 2 levels "low","high": 2 1 2 2 1 2 1 2 2 1 ...
```

Next, I went ahead and factorized the variables such as Month, waterfront, new_price, and zipcode. For our waterfront and new_price variable, I had to factorize those ones slightly differently in order for them to make sense. For waterfront, it was either "yes" or "no" because they either did or didn't have a waterfront view. In our original data set, new_price was a character. Through our different models, we wanted to see whether the price would be high or low, which is why we coded it that way. We wouldn't have been able to run the model if we kept it the way that it was because it could not categorize them.

Training-Validation Set

```
set.seed(666)

AD_train_index <- sample(1:nrow(AD_df2), 0.7 * nrow(AD_df2))
AD_valid_index <- setdiff(1:nrow(AD_df2), AD_train_index)

AD_train <- AD_df2[AD_train_index, ]
AD_valid <- AD_df2[AD_valid_index, ]

nrow(AD_train)
```

```
## [1] 10537
```

```
nrow(AD_valid)
```

```
## [1] 4516
```

We set the seed so that we would get the same results each time we ran the code. We decided to do a 70-30 split for our training-validation set.

Normalize Training and Validation Set

```
AD_train_norm <- AD_train
AD_valid_norm <- AD_valid

AD_norm_values <- preProcess(AD_train[, c(3, 5, 8)],
                          method = c("center",
                                     "scale"))
AD_train_norm[, c(3, 5, 8)] <- predict(AD_norm_values,
                                AD_train[, c(3, 5, 8)])

AD_valid_norm[, c(3, 5, 8)] <- predict(AD_norm_values,
                                 AD_valid[, c(3, 5, 8)])
str(AD_valid_norm)
```

```
## 'data.frame':    4516 obs. of  18 variables:
##  $ Year          : int  2014 2015 2014 2014 2014 2014 2014 2014 2015 2014 ...
##  $ Month         : Factor w/ 12 levels "1","2","3","4",..: 6 4 12 9 11 7 10 6 4 12 ...
##  $ price         : num  0.1273 1.367 0.0861 -0.4725 1.2275 ...
##  $ bedrooms      : int  5 3 6 3 5 3 3 3 2 4 ...
##  $ bathrooms     : num  0.824 -0.474 0.499 0.499 1.473 ...
##  $ sqft_living   : int  2910 1900 3370 1484 4115 1970 1800 2050 1100 2410 ...
##  $ sqft_lot      : int  36250 9375 15625 1761 7910 4058 4800 4185 1114 6770 ...
##  $ floors        : num  -0.92 -0.92 -0.92 2.776 0.928 ...
```

```
##  $ waterfront   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ view         : int  0 1 0 0 0 0 2 0 0 0 ...
##  $ condition    : int  3 4 3 3 3 3 4 3 3 4 ...
##  $ grade        : int  8 8 8 7 9 7 9 9 8 7 ...
##  $ sqft_above   : int  1590 1330 1770 1484 4115 1970 900 2050 900 1220 ...
##  $ sqft_basement: int  1320 570 1600 0 0 0 900 0 200 1190 ...
##  $ yr_built     : int  1977 1941 1964 2003 2014 2004 1927 2011 2009 1924 ...
##  $ yr_renovated : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : Factor w/ 70 levels "98001","98002",..: 39 50 64 50 30 35 57 52 55 43 ...
##  $ new_price    : Factor w/ 2 levels "low","high": 2 2 2 1 2 1 2 2 2 2 ...
```

```r
head(AD_valid_norm)
```

```
##    Year Month       price bedrooms  bathrooms sqft_living sqft_lot     floors
## 3  2014     6  0.12725124        5  0.8237190        2910    36250 -0.9197922
## 6  2015     4  1.36697273        3 -0.4743969        1900     9375 -0.9197922
## 9  2014    12  0.08614765        6  0.4991901        3370    15625 -0.9197922
## 10 2014     9 -0.47251009        3  0.4991901        1484     1761  2.7758609
## 12 2014    11  1.22752631        5  1.4727770        4115     7910  0.9280344
## 15 2014     7 -0.32535147        3  0.4991901        1970     4058  0.9280344
##    waterfront view condition grade sqft_above sqft_basement yr_built
## 3          No    0         3     8       1590          1320     1977
## 6          No    1         4     8       1330           570     1941
## 9          No    0         3     8       1770          1600     1964
## 10         No    0         3     7       1484             0     2003
## 12         No    0         3     9       4115             0     2014
## 15         No    0         3     7       1970             0     2004
##    yr_renovated zipcode new_price
## 3             0   98075      high
## 6             0   98115      high
## 9             0   98166      high
## 10            0   98115       low
## 12            0   98053      high
## 15            0   98065       low
```

In our data set, we had numerical variables, so we had to normalize them to put them on the same scale. These variables were price, bathrooms, and floors.

Load in New Customers and Normalize

```r
new_cust <- read.csv("house_test_5.csv")
new_cust_clean <- drop_na(new_cust)
AD_new_cust <- new_cust_clean[ , -c(1:2, 5:6, 21:22)]
head(AD_new_cust)
```

```
##   Year Month bedrooms bathrooms sqft_living sqft_lot floors waterfront view
## 1 2014     8        3       1.0        1010     7500    1.0          0    0
## 2 2014     6        3       2.0        1200     5029    1.0          0    0
## 3 2014     7        3       2.5        1420      814    2.0          0    0
## 4 2014    10        2       1.5        1240     3873    1.0          0    0
## 5 2015     3        4       1.0        1980     4560    1.5          0    0
```

```
## 6 2014    11        4       2.0        2200      3060     1.0           0    0
##   condition grade sqft_above sqft_basement yr_built yr_renovated zipcode
## 1         4     7       1010             0     1975            0   98074
## 2         3     6        880           320     1937            0   98115
## 3         3     8       1140           280     2008            0   98136
## 4         4     6        860           380     1909            0   98116
## 5         3     7       1980             0     1920            0   98103
## 6         3     7       1100          1100     1908         2000   98118
```

```
str(AD_new_cust)
```

```
## 'data.frame':    20 obs. of  16 variables:
##  $ Year         : int  2014 2014 2014 2014 2015 2014 2014 2015 2015 2014 ...
##  $ Month        : int  8 6 7 10 3 11 6 4 3 11 ...
##  $ bedrooms     : int  3 3 3 2 4 4 4 2 5 4 ...
##  $ bathrooms    : num  1 2 2.5 1.5 1 2 2.5 2 2.75 2.75 ...
##  $ sqft_living  : int  1010 1200 1420 1240 1980 2200 2240 1680 3100 4270 ...
##  $ sqft_lot     : int  7500 5029 814 3873 4560 3060 9826 6194 5298 25807 ...
##  $ floors       : num  1 1 2 1 1.5 1 1 1 2 2 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  4 3 3 4 3 3 4 3 3 3 ...
##  $ grade        : int  7 6 8 6 7 7 7 8 7 11 ...
##  $ sqft_above   : int  1010 880 1140 860 1980 1100 1370 1680 3100 4270 ...
##  $ sqft_basement: int  0 320 280 380 0 1100 870 0 0 0 ...
##  $ yr_built     : int  1975 1937 2008 1909 1920 1908 1988 2004 2007 1996 ...
##  $ yr_renovated : int  0 0 0 0 0 2000 0 0 0 0 ...
##  $ zipcode      : int  98074 98115 98136 98116 98103 98118 98023 98053 98065 98004 ...
```

```
AD_new_cust[, c(2,16)] <- lapply(AD_new_cust[, c(2,16)], as.factor)
AD_new_cust$waterfront <- factor(AD_new_cust$waterfront,
                          levels = c("0", "1"),
                          labels = c("No", "Yes"))
str(AD_new_cust)
```

```
## 'data.frame':    20 obs. of  16 variables:
##  $ Year         : int  2014 2014 2014 2014 2015 2014 2014 2015 2015 2014 ...
##  $ Month        : Factor w/ 10 levels "1","3","4","6",..: 6 4 5 8 2 9 4 3 2 9 ...
##  $ bedrooms     : int  3 3 3 2 4 4 4 2 5 4 ...
##  $ bathrooms    : num  1 2 2.5 1.5 1 2 2.5 2 2.75 2.75 ...
##  $ sqft_living  : int  1010 1200 1420 1240 1980 2200 2240 1680 3100 4270 ...
##  $ sqft_lot     : int  7500 5029 814 3873 4560 3060 9826 6194 5298 25807 ...
##  $ floors       : num  1 1 2 1 1.5 1 1 1 2 2 ...
##  $ waterfront   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ view         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  4 3 3 4 3 3 4 3 3 3 ...
##  $ grade        : int  7 6 8 6 7 7 7 8 7 11 ...
##  $ sqft_above   : int  1010 880 1140 860 1980 1100 1370 1680 3100 4270 ...
##  $ sqft_basement: int  0 320 280 380 0 1100 870 0 0 0 ...
##  $ yr_built     : int  1975 1937 2008 1909 1920 1908 1988 2004 2007 1996 ...
##  $ yr_renovated : int  0 0 0 0 0 2000 0 0 0 0 ...
##  $ zipcode      : Factor w/ 17 levels "98004","98023",..: 8 12 17 13 11 15 2 5 6 1 ...
```

```r
AD_norm_values <- preProcess(AD_new_cust[, c(4, 7)],
                             method = c("center",
                                        "scale"))
AD_new_cust_norm <- predict(AD_norm_values, AD_new_cust)
AD_new_cust_norm
```

```
##    Year Month bedrooms  bathrooms sqft_living sqft_lot     floors waterfront
## 1  2014     8        3 -2.0339854        1010     7500 -0.8786161         No
## 2  2014     6        3 -0.2259984        1200     5029 -0.8786161         No
## 3  2014     7        3  0.6779951        1420      814  0.6494119         No
## 4  2014    10        2 -1.1299919        1240     3873 -0.8786161         No
## 5  2015     3        4 -2.0339854        1980     4560 -0.1146021         No
## 6  2014    11        4 -0.2259984        2200     3060 -0.8786161         No
## 7  2014     6        4  0.6779951        2240     9826 -0.8786161         No
## 8  2015     4        2 -0.2259984        1680     6194 -0.8786161         No
## 9  2015     3        5  1.1299919        3100     5298  0.6494119         No
## 10 2014    11        4  1.1299919        4270    25807  0.6494119         No
## 11 2014     7        3  0.6779951        1800    11034  0.6494119         No
## 12 2014     9        4  0.6779951        2095     4400 -0.1146021         No
## 13 2015     1        3  0.6779951        2420     7500 -0.8786161         No
## 14 2014     6        4 -0.2259984        1570     9415  0.6494119         No
## 15 2014    12        3  0.2259984        1370     1533  2.1774398         No
## 16 2014     9        4  1.5819887        2720    11740 -0.8786161         No
## 17 2014    11        4 -0.6779951        1720     3050 -0.1146021         No
## 18 2015     3        4  0.6779951        3100     7807  0.6494119         No
## 19 2014     7        3 -0.6779951        1630     1526  2.1774398         No
## 20 2015     3        3 -0.6779951        1790    87213 -0.8786161         No
##    view condition grade sqft_above sqft_basement yr_built yr_renovated zipcode
## 1     0         4     7       1010             0     1975            0   98074
## 2     0         3     6        880           320     1937            0   98115
## 3     0         3     8       1140           280     2008            0   98136
## 4     0         4     6        860           380     1909            0   98116
## 5     0         3     7       1980             0     1920            0   98103
## 6     0         3     7       1100          1100     1908         2000   98118
## 7     0         4     7       1370           870     1988            0   98023
## 8     0         3     8       1680             0     2004            0   98053
## 9     0         3     7       3100             0     2007            0   98065
## 10    0         3    11       4270             0     1996            0   98004
## 11    0         4     8       1800             0     1987            0   98072
## 12    0         5     8       1295           800     1910            0   98116
## 13    2         4     8       1210          1210     1944            0   98117
## 14    0         4     7       1570             0     1984            0   98092
## 15    0         3     8       1370             0     2009            0   98133
## 16    0         5     9       2720             0     1957            0   98040
## 17    0         5     7       1040           680     1929            0   98116
## 18    0         3     9       3100             0     2003            0   98034
## 19    0         3     8       1630             0     2014            0   98103
## 20    0         4     7       1790             0     1974            0   98077
```

We then loaded in the new data set and dropped any of the missing values. From there, we also removed the unnecessary variables that we did before so that both of our data sets matched in terms of containing the same variables. After that, we factorized the same variables that we did with our original data set and normalized the same variables as well.

kNN Model, k = 3

Train Model

```
AD_knn_model_k3 <- caret::knn3(new_price ~ Year + Month + bedrooms + bathrooms + sqft_living + sqft_lot
                               data = AD_train_norm, k = 3)
AD_knn_model_k3
```

```
## 3-nearest neighbor model
## Training set outcome distribution:
##
##  low high
## 6710 3827
```

Predict Training Set

```
AD_knn_pred_k3_train <- predict(AD_knn_model_k3,
                                newdata = AD_train_norm,
                                type = "class")
head(AD_knn_pred_k3_train)
```

```
## [1] low  low  low  low  low  high
## Levels: low high
```

Evaluate

```
confusionMatrix(AD_knn_pred_k3_train, as.factor(AD_train_norm[, 18]), positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  low high
##       low  6150  877
##       high  560 2950
##
##                Accuracy : 0.8636
##                  95% CI : (0.8569, 0.8701)
##     No Information Rate : 0.6368
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6998
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7708
##             Specificity : 0.9165
##          Pos Pred Value : 0.8405
```

```
##          Neg Pred Value : 0.8752
##              Prevalence : 0.3632
##          Detection Rate : 0.2800
##    Detection Prevalence : 0.3331
##       Balanced Accuracy : 0.8437
##
##        'Positive' Class : high
##
```

Looking at the confusion matrix, we can see that the model has a strong accuracy score at .8636, indicating that the model actually predicted about 86% of the records in the training data. Both the sensitivity (.7708) and specificity (.9165) are relatively high, indicating that the recall rate is very good, and it is able to accurately predict a true positive, as well as accurately predict true negatives.

kNN Model, k = 5

Train Model

```
AD_knn_model_k5 <- caret::knn3(new_price ~ Year + Month + bedrooms + bathrooms + sqft_living + sqft_lot
                             data = AD_train_norm, k = 5)
AD_knn_model_k5
```

```
## 5-nearest neighbor model
## Training set outcome distribution:
##
##  low high
## 6710 3827
```

Predict Training Set

```
AD_knn_pred_k5_train <- predict(AD_knn_model_k5,
                               newdata = AD_train_norm,
                               type = "class")
head(AD_knn_pred_k5_train)
```

```
## [1] low  low  low  low  low  high
## Levels: low high
```

Evaluate

```
confusionMatrix(AD_knn_pred_k5_train, as.factor(AD_train_norm[, 18]), positive = "high")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction  low high
```

```
##       low  6038 1075
##       high  672 2752
##
##                Accuracy : 0.8342
##                  95% CI : (0.827, 0.8413)
##     No Information Rate : 0.6368
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6333
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7191
##             Specificity : 0.8999
##          Pos Pred Value : 0.8037
##          Neg Pred Value : 0.8489
##              Prevalence : 0.3632
##          Detection Rate : 0.2612
##    Detection Prevalence : 0.3250
##       Balanced Accuracy : 0.8095
##
##        'Positive' Class : high
##
```

Looking at the confusion matrix, we can see that the model has a strong accuracy score at .8342, indicating that the model actually predicted about 83% of the records in the training data. Both the sensitivity (.7191) and specificity (.8999) are relatively high, indicating that the recall rate is very good, and it is able to accurately predict a true positive, as well as accurately predict true negatives.

kNN Model, k = 7

Train Model

```
AD_knn_model_k7 <- caret::knn3(new_price ~ Year + Month + bedrooms + bathrooms + sqft_living + sqft_lot
                            data = AD_train_norm, k = 7)
AD_knn_model_k7
```

```
## 7-nearest neighbor model
## Training set outcome distribution:
##
##  low high
## 6710 3827
```

Predict Training Set

```
AD_knn_pred_k7_train <- predict(AD_knn_model_k7,
                             newdata = AD_train_norm,
                             type = "class")
head(AD_knn_pred_k7_train)
```

```
## [1] low low low low low low
## Levels: low high
```

Evaluate

```r
confusionMatrix(AD_knn_pred_k7_train, as.factor(AD_train_norm[, 18]), positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  low high
##       low  6002 1178
##       high  708 2649
##
##                Accuracy : 0.821
##                  95% CI : (0.8136, 0.8283)
##     No Information Rate : 0.6368
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6026
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6922
##             Specificity : 0.8945
##          Pos Pred Value : 0.7891
##          Neg Pred Value : 0.8359
##              Prevalence : 0.3632
##          Detection Rate : 0.2514
##    Detection Prevalence : 0.3186
##       Balanced Accuracy : 0.7933
##
##        'Positive' Class : high
##
```

Looking at the confusion matrix, we can see that the model has a strong accuracy score at .821, indicating that the model actually predicted about 82% of the records in the training data. Both the sensitivity (.6922) and specificity (.8945) are relatively high, indicating that the recall rate is very good, and it is able to accurately predict a true positive, as well as accurately predict true negatives.

Predict Validation Set, k = 3

```r
AD_knn_pred_k3_valid <- predict(AD_knn_model_k3,
                             newdata = AD_valid_norm,
                             type = "class")
head(AD_knn_pred_k3_valid)
```

```
## [1] low  low  high low  high high
## Levels: low high
```

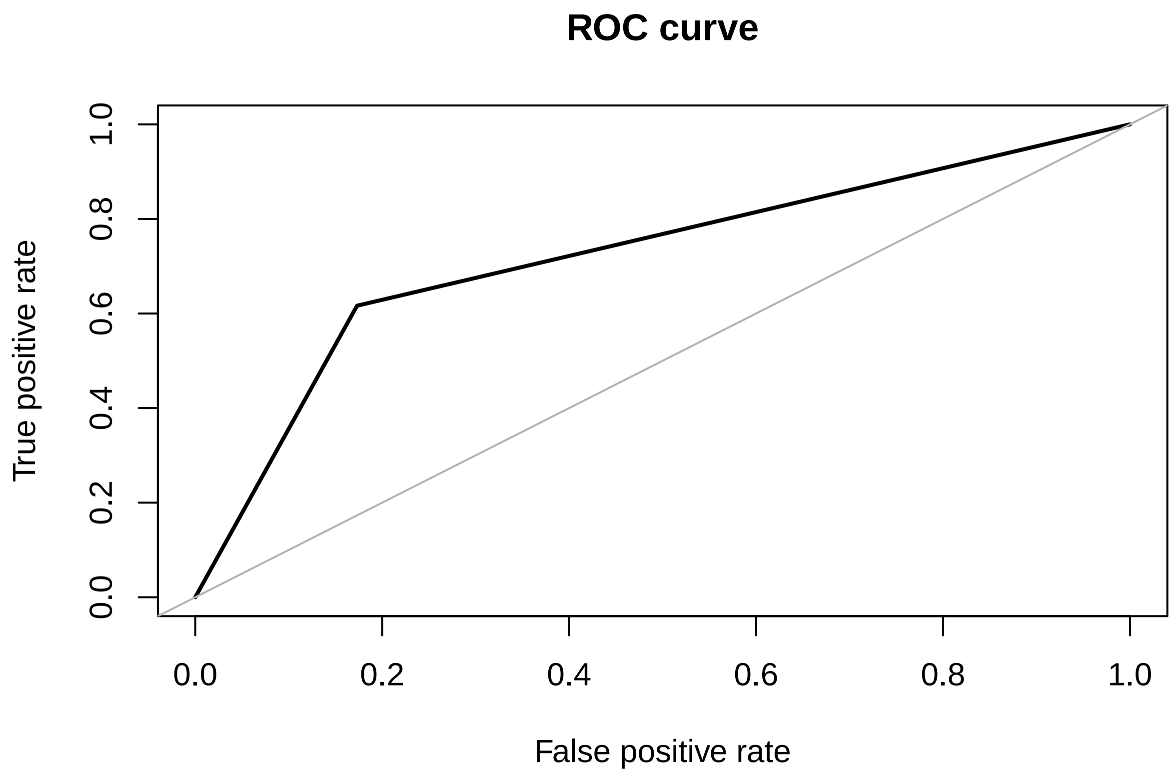
Evaluate

```
confusionMatrix(AD_knn_pred_k3_valid, as.factor(AD_valid_norm[, 18]),
                positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  low high
##       low  2346  644
##       high  491 1035
##
##                Accuracy : 0.7487
##                  95% CI : (0.7358, 0.7613)
##     No Information Rate : 0.6282
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4518
##
##  Mcnemar's Test P-Value : 6.429e-06
##
##             Sensitivity : 0.6164
##             Specificity : 0.8269
##          Pos Pred Value : 0.6782
##          Neg Pred Value : 0.7846
##              Prevalence : 0.3718
##          Detection Rate : 0.2292
##    Detection Prevalence : 0.3379
##       Balanced Accuracy : 0.7217
##
##        'Positive' Class : high
##
```

```
library(ROSE)

ROSE::roc.curve(AD_valid_norm$new_price,
                AD_knn_pred_k3_valid)
```




```
## Area under the curve (AUC): 0.722
```

The AUC is fairly high at .722, which means that the model is good at distinguishing between the positive and negative classes.

Predict Validation Set, k = 5

```
AD_knn_pred_k5_valid <- predict(AD_knn_model_k5,
                               newdata = AD_valid_norm,
                               type = "class")
head(AD_knn_pred_k5_valid)
```

```
## [1] low  low  high low  high high
## Levels: low high
```

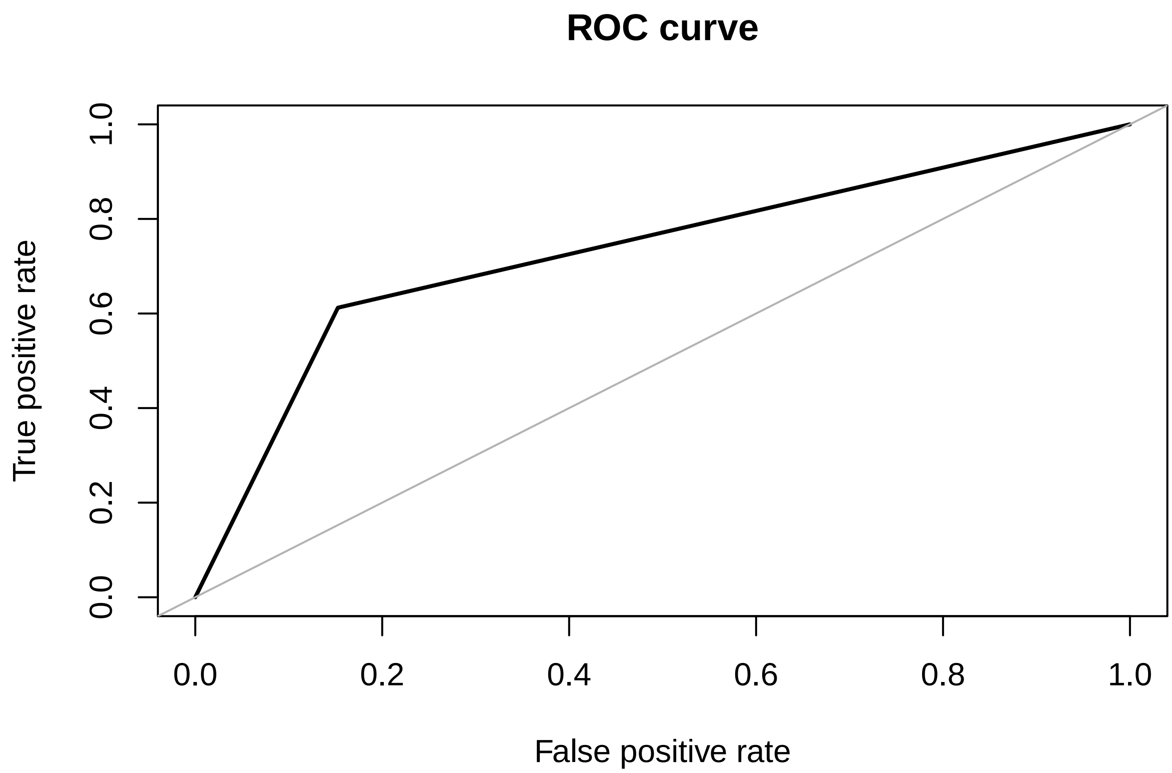
Evaluate

```
confusionMatrix(AD_knn_pred_k5_valid, as.factor(AD_valid_norm[, 18]),
                positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  low high
##       low  2404  651
##       high  433 1028
##
##                Accuracy : 0.76
##                  95% CI : (0.7472, 0.7724)
##     No Information Rate : 0.6282
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4722
##
##  Mcnemar's Test P-Value : 4.372e-11
##
##             Sensitivity : 0.6123
##             Specificity : 0.8474
##          Pos Pred Value : 0.7036
##          Neg Pred Value : 0.7869
##              Prevalence : 0.3718
##          Detection Rate : 0.2276
##    Detection Prevalence : 0.3235
##       Balanced Accuracy : 0.7298
##
##        'Positive' Class : high
##
```

```r
library(ROSE)

ROSE::roc.curve(AD_valid_norm$new_price,
                AD_knn_pred_k5_valid)
```




```
## Area under the curve (AUC): 0.730
```

The AUC is fairly high at .730, which means that the model is good at distinguishing between the positive and negative classes.

Predict Validation Set, k = 7

```
AD_knn_pred_k7_valid <- predict(AD_knn_model_k7,
                                newdata = AD_valid_norm,
                                type = "class")
head(AD_knn_pred_k7_valid)
```

```
## [1] low  low  high low  high high
## Levels: low high
```

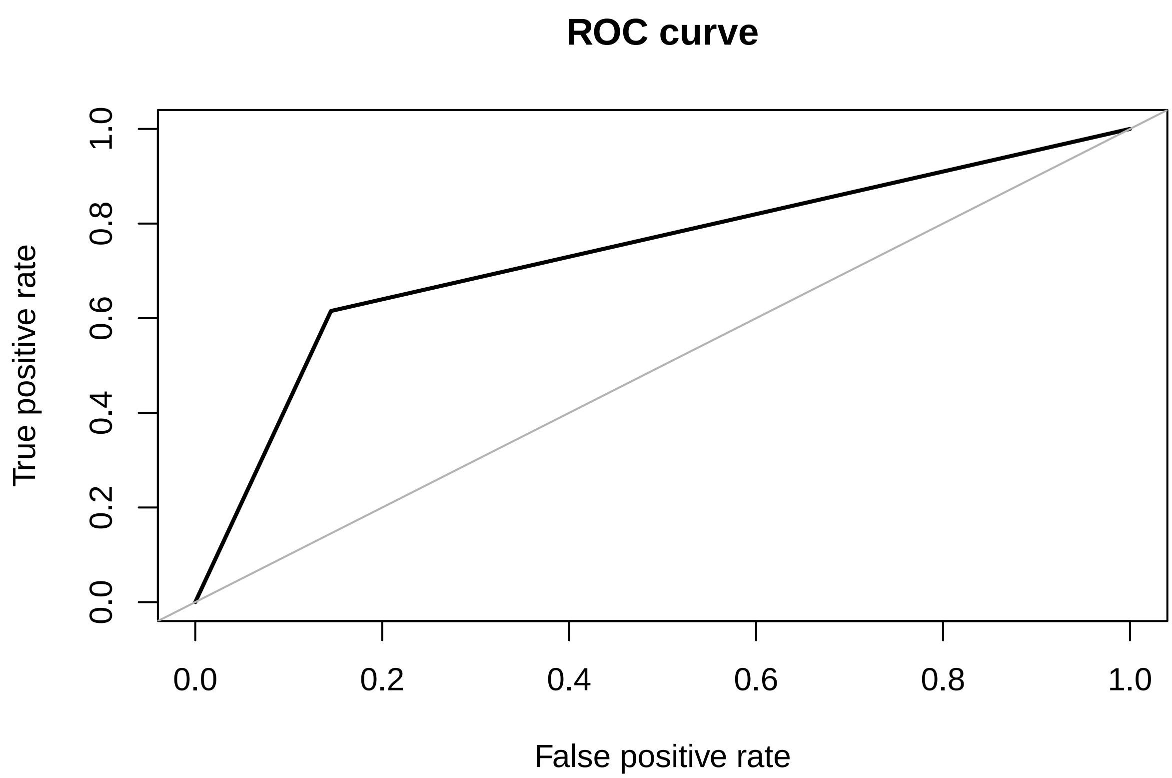
Evaluate

```
confusionMatrix(AD_knn_pred_k7_valid, as.factor(AD_valid_norm[, 18]),
                positive = "high")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  low high
##       low  2425  646
##       high  412 1033
##
##                Accuracy : 0.7657
##                  95% CI : (0.7531, 0.778)
##     No Information Rate : 0.6282
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4838
##
##  Mcnemar's Test P-Value : 7.876e-13
##
##             Sensitivity : 0.6152
##             Specificity : 0.8548
##          Pos Pred Value : 0.7149
##          Neg Pred Value : 0.7896
##              Prevalence : 0.3718
##          Detection Rate : 0.2287
##    Detection Prevalence : 0.3200
##       Balanced Accuracy : 0.7350
##
##        'Positive' Class : high
##
```

```
library(ROSE)

ROSE::roc.curve(AD_valid_norm$new_price,
                AD_knn_pred_k7_valid)
```




```
## Area under the curve (AUC): 0.735
```

The AUC is fairly high at .735, which means that the model is good at distinguishing between the positive and negative classes.

```
new_cust_predict <- predict(AD_knn_model_k5, newdata = AD_new_cust_norm,
                            type = "class")
new_cust_predict
```

```
##  [1] low  low  low  low  low  high low  low  low  high low  low  low  low  low
## [16] low  low  high high low
## Levels: low high
```

Regression Model

Load in Data

```
NUhousing_df <- read.csv("house_5.csv", header = TRUE)
head(NUhousing_df)
```

```
##   X         id Year Month Day day_of_week   price bedrooms bathrooms
```

```
## 1 1 3025059093 2014     7  29           2 3100000        5      5.25
## 2 2 7349400610 2014     8  12           2  305000        4      2.25
## 3 3 7527410080 2014     6   2           1  585083        5      2.75
## 4 4 7855000325 2015     2  20           5 1050000        4      3.00
## 5 5  421079105 2015     3   9           1  325000        3      2.25
## 6 6 8925100390 2015     4   6           1 1040000        3      1.75
##   sqft_living sqft_lot floors waterfront view condition grade sqft_above
## 1        5090    23669    2.0          0    0         3    12       5090
## 2        2050    12581    2.0          0    0         4     7       2050
## 3        2910    36250    1.0          0    0         3     8       1590
## 4        3080    10757    2.0          0    3         5     8       3080
## 5        1480    97138    1.5          0    0         3     7       1480
## 6        1900     9375    1.0          0    1         4     8       1330
##   sqft_basement yr_built yr_renovated zipcode     lat     long
## 1             0     2006            0   98004 47.6297 -122.216
## 2             0     1978            0   98002 47.3215 -122.204
## 3          1320     1977            0   98075 47.5916 -122.076
## 4             0     1961            0   98006 47.5671 -122.159
## 5             0     1984            0   98010 47.3317 -121.927
## 6           570     1941            0   98115 47.6821 -122.273
```

Drop Unneccessary Variables and Missing Values

```
NUhousing_df_1 <- NUhousing_df[, -c(1:2, 5:6, 22:23)]
NUhousing_df_1 <- drop_na(NUhousing_df_1)
```

Dropping unnecessary variables such as ID, Day of Month, and Day of Week since domain knowledge tells us that these variables are not very useful. Additionally, dropping lattitute and longitude since zipcode is a better predicter of location than these variables.

Change Variable Types

```
str(NUhousing_df_1)
```

```
## 'data.frame':    15053 obs. of  17 variables:
##  $ Year         : int  2014 2014 2014 2015 2015 2015 2014 2014 2014 2014 ...
##  $ Month        : int  7 8 6 2 3 4 11 6 12 9 ...
##  $ price        : num  3100000 305000 585083 1050000 325000 ...
##  $ bedrooms     : int  5 4 5 4 3 3 4 3 6 3 ...
##  $ bathrooms    : num  5.25 2.25 2.75 3 2.25 1.75 2 1.75 2.5 2.5 ...
##  $ sqft_living  : int  5090 2050 2910 3080 1480 1900 2280 1960 3370 1484 ...
##  $ sqft_lot     : int  23669 12581 36250 10757 97138 9375 7200 6380 15625 1761 ...
##  $ floors       : num  2 2 1 2 1.5 1 1 1 1 3 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 3 0 1 0 0 0 0 ...
##  $ condition    : int  3 4 3 5 3 4 4 4 3 3 ...
##  $ grade        : int  12 7 8 8 7 8 7 7 8 7 ...
##  $ sqft_above   : int  5090 2050 1590 3080 1480 1330 2280 980 1770 1484 ...
##  $ sqft_basement: int  0 0 1320 0 0 570 0 980 1600 0 ...
```

```
##  $ yr_built     : int  2006 1978 1977 1961 1984 1941 1956 1939 1964 2003 ...
##  $ yr_renovated : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98004 98002 98075 98006 98010 98115 98133 98115 98166 98115 ...
```

```
NUhousing_df_1[, c(2,9,17)] <- lapply(NUhousing_df_1[, c(2,9,17)], as.factor)
str(NUhousing_df_1)
```

```
## 'data.frame':    15053 obs. of  17 variables:
##  $ Year         : int  2014 2014 2014 2015 2015 2015 2014 2014 2014 2014 ...
##  $ Month        : Factor w/ 12 levels "1","2","3","4",..: 7 8 6 2 3 4 11 6 12 9 ...
##  $ price        : num  3100000 305000 585083 1050000 325000 ...
##  $ bedrooms     : int  5 4 5 4 3 3 4 3 6 3 ...
##  $ bathrooms    : num  5.25 2.25 2.75 3 2.25 1.75 2 1.75 2.5 2.5 ...
##  $ sqft_living  : int  5090 2050 2910 3080 1480 1900 2280 1960 3370 1484 ...
##  $ sqft_lot     : int  23669 12581 36250 10757 97138 9375 7200 6380 15625 1761 ...
##  $ floors       : num  2 2 1 2 1.5 1 1 1 1 3 ...
##  $ waterfront   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ view         : int  0 0 0 3 0 1 0 0 0 0 ...
##  $ condition    : int  3 4 3 5 3 4 4 4 3 3 ...
##  $ grade        : int  12 7 8 8 7 8 7 7 8 7 ...
##  $ sqft_above   : int  5090 2050 1590 3080 1480 1330 2280 980 1770 1484 ...
##  $ sqft_basement: int  0 0 1320 0 0 570 0 980 1600 0 ...
##  $ yr_built     : int  2006 1978 1977 1961 1984 1941 1956 1939 1964 2003 ...
##  $ yr_renovated : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : Factor w/ 70 levels "98001","98002",..: 4 2 39 6 9 50 58 50 64 50 ...
```

Factorizing month, waterfront, and zipcode since these integers represent a certain value.

Training Validation Split

```
set.seed(666)

NUtrain_index <- sample(1:nrow(NUhousing_df_1), 0.6 * nrow(NUhousing_df_1))
NUvalid_index <- setdiff(1:nrow(NUhousing_df_1), NUtrain_index)

NUtrain_df <- NUhousing_df_1[NUtrain_index, ]
NUvalid_df <- NUhousing_df_1[NUvalid_index, ]

nrow(NUtrain_df)
```

```
## [1] 9031
```

```
nrow(NUvalid_df)
```

```
## [1] 6022
```

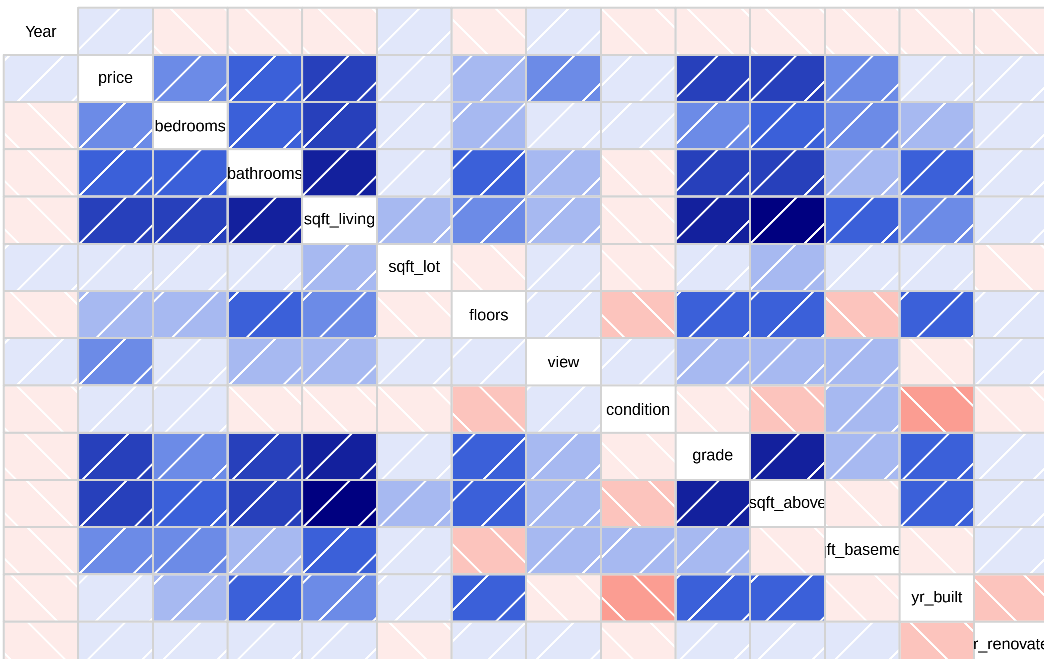
##Correlation Matrix

```
library(corrgram)
```

```
##
## Attaching package: 'corrgram'

## The following object is masked from 'package:lattice':
##
##     panel.fill
```

```
corrgram(NUtrain_df)
```



The variables most strongly correlated with price are bedrooms, bathrooms, sqft living, grade, sqft above, sqft basement. Square foot living is strongly correlated to square foot above, so I will drop sqft above since it is encompassed by square foot living, which is the square footage of the whole home. Since bathrooms and sqft living are correlated, and living seems more important, I will drop bathrooms.

Variable Set 1

I will first run a regression with all the variables of interest, including the categorical variables not present in the correlation matrix.

```
names(NUhousing_df_1)
```

```
##  [1] "Year"          "Month"         "price"         "bedrooms"
##  [5] "bathrooms"     "sqft_living"   "sqft_lot"      "floors"
##  [9] "waterfront"    "view"          "condition"     "grade"
## [13] "sqft_above"    "sqft_basement" "yr_built"      "yr_renovated"
## [17] "zipcode"
```

```
NUreg_model1 <- lm(price ~ Year + bedrooms + sqft_living + grade + sqft_basement + waterfront  + zipcode
                   data = NUtrain_df)
summary(NUreg_model1)
```

```
##
## Call:
## lm(formula = price ~ Year + bedrooms + sqft_living + grade +
##     sqft_basement + waterfront + zipcode, data = NUtrain_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -931736  -78416   -4048   65041 4482599
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -6.390e+07  7.764e+06  -8.231  < 2e-16 ***
## Year           3.153e+04  3.854e+03   8.182 3.18e-16 ***
## bedrooms      -3.104e+04  2.508e+03 -12.375  < 2e-16 ***
## sqft_living    2.268e+02  4.054e+00  55.941  < 2e-16 ***
## grade          4.658e+04  2.618e+03  17.792  < 2e-16 ***
## sqft_basement -2.786e+01  5.078e+00  -5.485 4.24e-08 ***
## waterfront1    8.730e+05  2.122e+04  41.141  < 2e-16 ***
## zipcode98002   3.713e+04  2.345e+04   1.584 0.113292
## zipcode98003  -1.649e+04  2.129e+04  -0.775 0.438602
## zipcode98004   7.953e+05  2.118e+04  37.558  < 2e-16 ***
## zipcode98005   3.182e+05  2.552e+04  12.469  < 2e-16 ***
## zipcode98006   3.265e+05  1.916e+04  17.043  < 2e-16 ***
## zipcode98007   2.543e+05  2.660e+04   9.560  < 2e-16 ***
## zipcode98008   2.919e+05  2.168e+04  13.466  < 2e-16 ***
## zipcode98010   5.781e+04  3.415e+04   1.693 0.090475 .
## zipcode98011   9.691e+04  2.408e+04   4.024 5.76e-05 ***
## zipcode98014   1.038e+05  2.584e+04   4.018 5.92e-05 ***
## zipcode98019   7.343e+04  2.459e+04   2.987 0.002829 **
## zipcode98022   6.155e+04  2.252e+04   2.733 0.006282 **
## zipcode98023  -2.683e+04  1.848e+04  -1.452 0.146594
## zipcode98024   1.950e+05  3.022e+04   6.452 1.16e-10 ***
## zipcode98027   1.720e+05  1.961e+04   8.771  < 2e-16 ***
## zipcode98028   1.255e+05  2.152e+04   5.830 5.72e-09 ***
## zipcode98029   1.967e+05  2.113e+04   9.312  < 2e-16 ***
## zipcode98030  -3.967e+03  2.122e+04  -0.187 0.851736
## zipcode98031   1.791e+04  2.113e+04   0.847 0.396835
## zipcode98032   3.460e+04  3.045e+04   1.136 0.255900
## zipcode98033   3.866e+05  1.945e+04  19.876  < 2e-16 ***
## zipcode98034   2.093e+05  1.833e+04  11.419  < 2e-16 ***
## zipcode98038   1.554e+04  1.822e+04   0.853 0.393577
## zipcode98039   1.338e+06  3.992e+04  33.515  < 2e-16 ***
## zipcode98040   5.802e+05  2.197e+04  26.409  < 2e-16 ***
## zipcode98042   6.481e+03  1.829e+04   0.354 0.723116
```

```
## zipcode98045   1.003e+05  2.304e+04   4.353 1.36e-05 ***
## zipcode98052   2.288e+05  1.821e+04  12.559  < 2e-16 ***
## zipcode98053   1.851e+05  1.964e+04   9.423  < 2e-16 ***
## zipcode98055   5.192e+04  2.087e+04   2.488 0.012858 *
## zipcode98056   9.057e+04  1.974e+04   4.588 4.54e-06 ***
## zipcode98058   3.453e+04  1.877e+04   1.840 0.065856 .
## zipcode98059   7.814e+04  1.886e+04   4.142 3.47e-05 ***
## zipcode98065   8.905e+04  2.228e+04   3.996 6.48e-05 ***
## zipcode98070  -1.390e+04  2.912e+04  -0.477 0.633213
## zipcode98072   1.496e+05  2.194e+04   6.816 9.95e-12 ***
## zipcode98074   1.641e+05  1.941e+04   8.458  < 2e-16 ***
## zipcode98075   1.661e+05  1.997e+04   8.319  < 2e-16 ***
## zipcode98077   1.080e+05  2.274e+04   4.751 2.06e-06 ***
## zipcode98092  -3.598e+04  2.044e+04  -1.760 0.078369 .
## zipcode98102   5.543e+05  3.254e+04  17.032  < 2e-16 ***
## zipcode98103   3.399e+05  1.794e+04  18.944  < 2e-16 ***
## zipcode98105   5.113e+05  2.365e+04  21.622  < 2e-16 ***
## zipcode98106   1.394e+05  2.026e+04   6.879 6.43e-12 ***
## zipcode98107   3.773e+05  2.168e+04  17.401  < 2e-16 ***
## zipcode98108   1.418e+05  2.451e+04   5.786 7.44e-09 ***
## zipcode98109   5.084e+05  2.752e+04  18.475  < 2e-16 ***
## zipcode98112   6.281e+05  2.173e+04  28.899  < 2e-16 ***
## zipcode98115   3.511e+05  1.801e+04  19.491  < 2e-16 ***
## zipcode98116   3.361e+05  2.049e+04  16.401  < 2e-16 ***
## zipcode98117   3.358e+05  1.855e+04  18.102  < 2e-16 ***
## zipcode98118   1.975e+05  1.874e+04  10.539  < 2e-16 ***
## zipcode98119   5.247e+05  2.360e+04  22.233  < 2e-16 ***
## zipcode98122   3.681e+05  2.163e+04  17.019  < 2e-16 ***
## zipcode98125   2.097e+05  1.915e+04  10.951  < 2e-16 ***
## zipcode98126   2.253e+05  2.036e+04  11.064  < 2e-16 ***
## zipcode98133   1.680e+05  1.873e+04   8.968  < 2e-16 ***
## zipcode98136   2.814e+05  2.145e+04  13.120  < 2e-16 ***
## zipcode98144   3.107e+05  2.032e+04  15.291  < 2e-16 ***
## zipcode98146   1.398e+05  2.116e+04   6.608 4.12e-11 ***
## zipcode98148   6.813e+04  4.053e+04   1.681 0.092785 .
## zipcode98155   1.731e+05  1.899e+04   9.116  < 2e-16 ***
## zipcode98166   1.028e+05  2.203e+04   4.665 3.13e-06 ***
## zipcode98168   8.188e+04  2.212e+04   3.702 0.000215 ***
## zipcode98177   2.659e+05  2.124e+04  12.520  < 2e-16 ***
## zipcode98178   8.060e+04  2.126e+04   3.791 0.000151 ***
## zipcode98188   7.375e+04  2.641e+04   2.793 0.005237 **
## zipcode98198   2.611e+04  2.121e+04   1.231 0.218270
## zipcode98199   4.096e+05  2.107e+04  19.436  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 169100 on 8955 degrees of freedom
## Multiple R-squared:  0.7917, Adjusted R-squared:  0.7899
## F-statistic: 453.7 on 75 and 8955 DF,  p-value: < 2.2e-16
```

Predicting Training Set 1

```
library(forecast)
NUreg_model_pred_train <- predict(NUreg_model1,
                                  NUtrain_df)
accuracy(NUreg_model_pred_train, NUtrain_df$price)
```

```
##                      ME     RMSE      MAE       MPE     MAPE
## Test set -5.176522e-06 168407.5 102566.1 -1.233898 21.16796
```

```
sd(NUtrain_df$price)
```

```
## [1] 368977.3
```

Since the standard deviation of price is $368,977 and the RMSE is only 168,407 which is about half of one standard deviation of price, this indicates this model may not have much error and will be good at predictions.

Predicting Validation Set 1

```
NUreg_model_pred_valid1 <- predict(NUreg_model1,
                                  NUvalid_df)
accuracy(NUreg_model_pred_valid1, NUvalid_df$price)
```

```
##                 ME     RMSE      MAE       MPE     MAPE
## Test set -725.4836 166515.7 103099.2 -1.511658 21.19915
```

The validation set RMSE is slightly lower that the training set, which is unlikely but could be due to random chance. The validation set's RMSE being low is a good sign that the model is good at predicting other data sets.

Regression Model 2

For this regression model, I only included the numerical variables of interest and excluded the categorical variables to see if fewer variables would improve our error rate.

```
NUreg_model2 <- lm(price ~ bedrooms + sqft_living + grade + sqft_basement,
                      data = NUtrain_df)
summary(NUreg_model2)
```

```
##
## Call:
## lm(formula = price ~ bedrooms + sqft_living + grade + sqft_basement,
##     data = NUtrain_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -1116478  -134125   -25560    95951  4571207
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -4.957e+05  2.313e+04  -21.43   <2e-16 ***
## bedrooms      -4.547e+04  3.575e+03  -12.72   <2e-16 ***
## sqft_living    1.928e+02  5.587e+00   34.51   <2e-16 ***
## grade          9.967e+04  3.563e+03   27.98   <2e-16 ***
## sqft_basement  8.004e+01  6.837e+00   11.71   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 247400 on 9026 degrees of freedom
## Multiple R-squared:  0.5505, Adjusted R-squared:  0.5503
## F-statistic:  2763 on 4 and 9026 DF,  p-value: < 2.2e-16
```

Predict Training Set 2

```
NUreg_model_pred_train2 <- predict(NUreg_model2,
                                NUtrain_df)
accuracy(NUreg_model_pred_train2, NUtrain_df$price)
```

```
##                      ME     RMSE      MAE       MPE     MAPE
## Test set -9.391321e-09 247370.2 161052.8 -10.66545 33.23633
```

Predicting Validation Set

```
NUreg_model_pred_valid2 <- predict(NUreg_model2,
                                NUvalid_df)
accuracy(NUreg_model_pred_valid2, NUvalid_df$price)
```

```
##                ME     RMSE      MAE       MPE     MAPE
## Test set 4056.96 246953.5 161553.4 -10.25075 32.85206
```

Since the RMSE for the training and validation set of regression 2 is higher than regression 1, this shows that regression 1 may be the more accurate model.

Evaluating Model - Regression 1

```
library(car)
vif(NUreg_model1)
```

```
##                   GVIF Df GVIF^(1/(2*Df))
## Year          1.010741  1        1.005356
## bedrooms      1.684553  1        1.297903
## sqft_living   4.472966  1        2.114939
```

```
## grade         3.076134  1        1.753891
## sqft_basement 1.581951  1        1.257756
## waterfront    1.077855  1        1.038198
## zipcode       1.894463 69        1.004641
```

Square foot of living and grade seem to have some multicollinearity, but the other variables have low VIF values which is good. This makes sense because houses with higher square foot would have a higher grade since this is a desirable trait.

Homoskedasticity

```r
library(lmtest)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
bptest(NUreg_model1)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  NUreg_model1
## BP = 1146.4, df = 75, p-value < 2.2e-16
```

Since the p-value is less than .05, we have sufficient evidence that there is not heteroskedasticity in the model.

Predicting Prices of New Houses

```r
NUnew_record <- read.csv("house_test_5.csv", header = TRUE)
NUnew_record<- NUnew_record[, -c(1:2, 5:6, 21:22)]
NUnew_record[, c(2,8,16)] <- lapply(NUnew_record[, c(2,8,16)], as.factor)

NUreg_model_pred_new <- predict(NUreg_model1,
                                newdata = NUnew_record, interval = "confidence")
NUreg_model_pred_new
```

```
##          fit      lwr      upr
## 1   233856.2 208065.1 259647.3
## 2   408427.4 387058.4 429796.3
## 3   482863.7 451654.9 514072.6
```

```
## 4   431831.2  402766.9  460895.4
## 5   630129.7  608094.3  652165.0
## 6   475409.3  451207.7  499611.0
## 7   266586.1  243393.9  289778.2
## 8   515919.2  489255.0  542583.4
## 9   602252.6  567738.7  636766.4
## 10 1759723.9 1728435.8 1791012.0
## 11  445034.7  412715.9  477353.5
## 12  645137.0  616500.4  673773.5
## 13  769730.6  745482.8  793978.5
## 14  129715.5  101092.6  158338.3
## 15  365946.0  341996.5  389895.6
## 16 1099886.0 1067599.0 1132173.0
## 17  516845.7  488127.7  545563.6
## 18  846710.8  823130.6  870290.9
## 19  596834.1  576048.4  617619.8
## 20  386195.3  351439.1  420951.5
```

The confidence interval indicates that there is a 95% chance of the true price being between the lower and upper bound for each of the houses. Since our model had a pretty small error rate compared to the standard deviation, we can rely on these predictions to a certain extent.

Predict new house prices with best model

We decided that the regression model was the best model because it provides us with a range of prices for the homes, instead of simply "high" or "low." We chose regression model 1 because it has a lower RMSE for both the training and validation sets, showing that it should predict other data sets more accurately. Our model predicted that we can be 95% confident that the true price of the houses will be between the lower and upper bounds.