Person 1(Group 1) - Similarity Regression

Dinesh Angadipeta

3/19/2023

Data set

```
houseData <- read.csv("kc_house_data.csv")
houseData <- subset(houseData, select = -id)
houseData <- subset(houseData, select = -date)
houseData <- subset(houseData, select = -sqft_basement)
```

For this notebook we will be using the data set of house sales in King County, USA between May 2014 and May 2015. The source for the original Kaggle page of the data set is here. We are removing the id and date columns for easier data exploration since those columns are not necessary for our goal. The basement sqft column is removed since

```
set.seed(1)
sample <- sample(c(TRUE, FALSE), nrow(houseData), replace=TRUE, prob=c(0.8,0.2))
train  <- houseData[sample, ]
test   <- houseData[!sample, ]
dim(train)
```

```
## [1] 17227    18
```

```
dim(test)
```

```
## [1] 4386   18
```

Here we are dividing into a 80/20 train/test.

Statistical Data Exploration

```


names(train)
```

names function

```
## [1] "price"         "bedrooms"      "bathrooms"     "sqft_living"
## [5] "sqft_lot"      "floors"        "waterfront"    "view"
## [9] "condition"     "grade"         "sqft_above"    "yr_built"
## [13] "yr_renovated"  "zipcode"       "lat"           "long"
## [17] "sqft_living15" "sqft_lot15"
```

Here we are listing the names of the variables in the data set. This helps plan out what variables will be useful for data exploration.

```
str(train)
```

str function

```
## 'data.frame':    17227 obs. of  18 variables:
##  $ price        : num  221900 538000 180000 510000 291850 ...
##  $ bedrooms     : int  3 3 2 3 3 3 3 3 2 3 ...
##  $ bathrooms    : num  1 2.25 1 2 1.5 1 2.5 2.5 1 1 ...
##  $ sqft_living  : int  1180 2570 770 1680 1060 1780 1890 3560 1160 1430 ...
##  $ sqft_lot     : int  5650 7242 10000 8080 9711 7470 6560 9796 6000 19901 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 1.5 ...
##  $ 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  3 3 3 3 3 3 3 3 4 4 ...
##  $ grade        : int  7 7 6 8 7 7 7 8 7 7 ...
##  $ sqft_above   : int  1180 2170 770 1680 1060 1050 1890 1860 860 1430 ...
##  $ yr_built     : int  1955 1951 1933 1987 1963 1960 2003 1965 1942 1927 ...
##  $ yr_renovated : int  0 1991 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98178 98125 98028 98074 98198 98146 98038 98007 98115 98028 ...
##  $ lat          : num  47.5 47.7 47.7 47.6 47.4 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: int  1340 1690 2720 1800 1650 1780 2390 2210 1330 1780 ...
##  $ sqft_lot15   : int  5650 7639 8062 7503 9711 8113 7570 8925 6000 12697 ...
```

Here we are using the "str" function to see how the data set is structured.

```
colSums(is.na(train))
```

colSums function using is.na

```
##         price      bedrooms     bathrooms   sqft_living      sqft_lot
##             0             0             0             0             0
##        floors    waterfront          view     condition         grade
##             0             0             0             0             0
##    sqft_above      yr_built  yr_renovated       zipcode           lat
##             0             0             0             0             0
##          long sqft_living15    sqft_lot15
##             0             0             0
```

Here we are looking at the number of missing values in each of the variables of the data set. This can cause problems with the missing data values, however this data set shows no instances of NA data.

```r
dim(train)
```

dim function

```
## [1] 17227    18
```

As used before when creating the test and training data, the dim function helps how the number of rows and columns.

```r
head(train)
```

head function

```
##    price bedrooms bathrooms sqft_living sqft_lot floors waterfront view
## 1 221900        3      1.00        1180     5650      1          0    0
## 2 538000        3      2.25        2570     7242      2          0    0
## 3 180000        2      1.00         770    10000      1          0    0
## 5 510000        3      2.00        1680     8080      1          0    0
## 8 291850        3      1.50        1060     9711      1          0    0
## 9 229500        3      1.00        1780     7470      1          0    0
##   condition grade sqft_above yr_built yr_renovated zipcode     lat     long
## 1         3     7       1180     1955            0   98178 47.5112 -122.257
## 2         3     7       2170     1951         1991   98125 47.7210 -122.319
## 3         3     6        770     1933            0   98028 47.7379 -122.233
## 5         3     8       1680     1987            0   98074 47.6168 -122.045
## 8         3     7       1060     1963            0   98198 47.4095 -122.315
## 9         3     7       1050     1960            0   98146 47.5123 -122.337
##   sqft_living15 sqft_lot15
## 1          1340       5650
## 2          1690       7639
## 3          2720       8062
## 5          1800       7503
## 8          1650       9711
## 9          1780       8113
```

The head function helps look at the first 6 rows.

```r
summary(train)
```

summary function

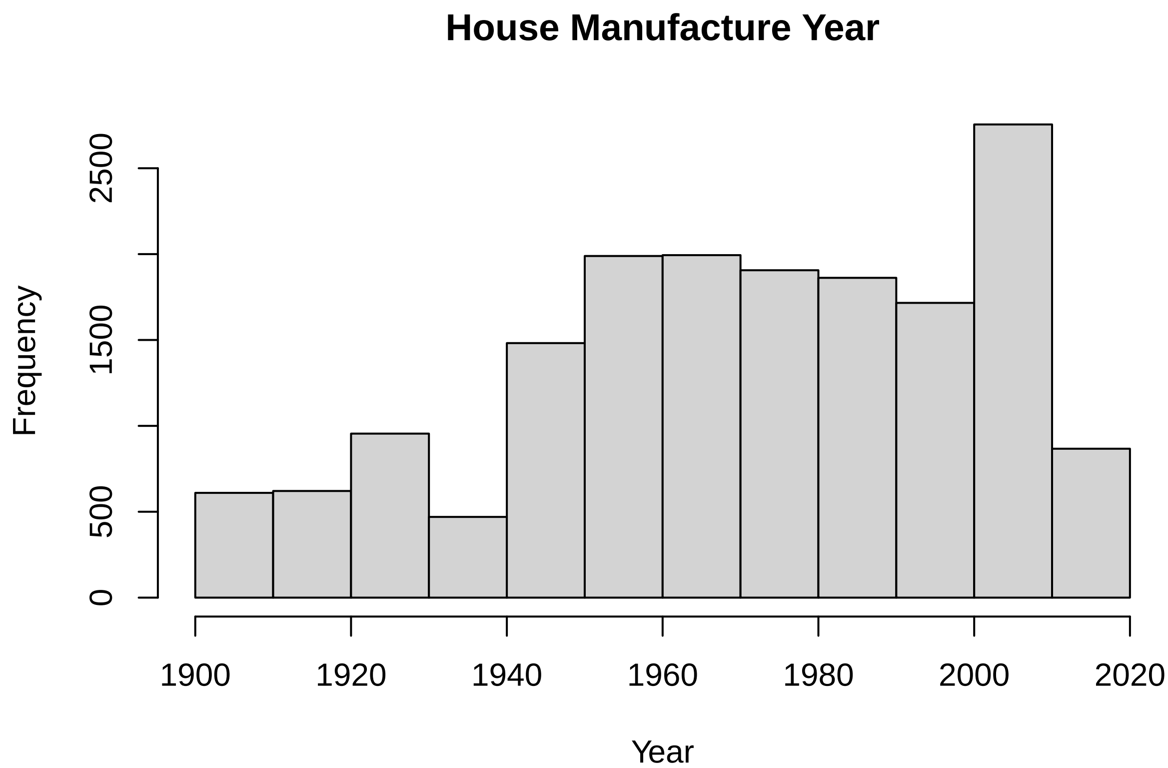
```
##      price           bedrooms       bathrooms      sqft_living
##  Min.   :  75000   Min.   : 0.00   Min.   :0.000   Min.   :  380
##  1st Qu.: 320000   1st Qu.: 3.00   1st Qu.:1.500   1st Qu.: 1420
##  Median : 450000   Median : 3.00   Median :2.250   Median : 1910
```

```
##  Mean   : 538657   Mean   : 3.37   Mean   :2.111   Mean   : 2078
##  3rd Qu.: 643202   3rd Qu.: 4.00   3rd Qu.:2.500   3rd Qu.: 2550
##  Max.   :7700000   Max.   :11.00   Max.   :8.000   Max.   :12050
##     sqft_lot           floors        waterfront            view
##  Min.   :    520   Min.   :1.000   Min.   :0.000000   Min.   :0.0000
##  1st Qu.:   5050   1st Qu.:1.000   1st Qu.:0.000000   1st Qu.:0.0000
##  Median :   7620   Median :1.500   Median :0.000000   Median :0.0000
##  Mean   :  15099   Mean   :1.496   Mean   :0.007082   Mean   :0.2342
##  3rd Qu.:  10723   3rd Qu.:2.000   3rd Qu.:0.000000   3rd Qu.:0.0000
##  Max.   :1651359   Max.   :3.500   Max.   :1.000000   Max.   :4.0000
##    condition          grade          sqft_above      yr_built
##  Min.   :1.000   Min.   : 3.000   Min.   : 380   Min.   :1900
##  1st Qu.:3.000   1st Qu.: 7.000   1st Qu.:1190   1st Qu.:1951
##  Median :3.000   Median : 7.000   Median :1560   Median :1975
##  Mean   :3.409   Mean   : 7.656   Mean   :1790   Mean   :1971
##  3rd Qu.:4.000   3rd Qu.: 8.000   3rd Qu.:2220   3rd Qu.:1997
##  Max.   :5.000   Max.   :13.000   Max.   :8860   Max.   :2015
##   yr_renovated        zipcode           lat             long
##  Min.   :   0.00   Min.   :98001   Min.   :47.16   Min.   :-122.5
##  1st Qu.:   0.00   1st Qu.:98033   1st Qu.:47.47   1st Qu.:-122.3
##  Median :   0.00   Median :98065   Median :47.57   Median :-122.2
##  Mean   :  85.96   Mean   :98078   Mean   :47.56   Mean   :-122.2
##  3rd Qu.:   0.00   3rd Qu.:98118   3rd Qu.:47.68   3rd Qu.:-122.1
##  Max.   :2015.00   Max.   :98199   Max.   :47.78   Max.   :-121.3
##  sqft_living15    sqft_lot15
##  Min.   : 399   Min.   :   651
##  1st Qu.:1480   1st Qu.:  5100
##  Median :1840   Median :  7620
##  Mean   :1986   Mean   : 12682
##  3rd Qu.:2360   3rd Qu.: 10097
##  Max.   :6210   Max.   :858132
```

This function gives an overview of the statistics of each variable.

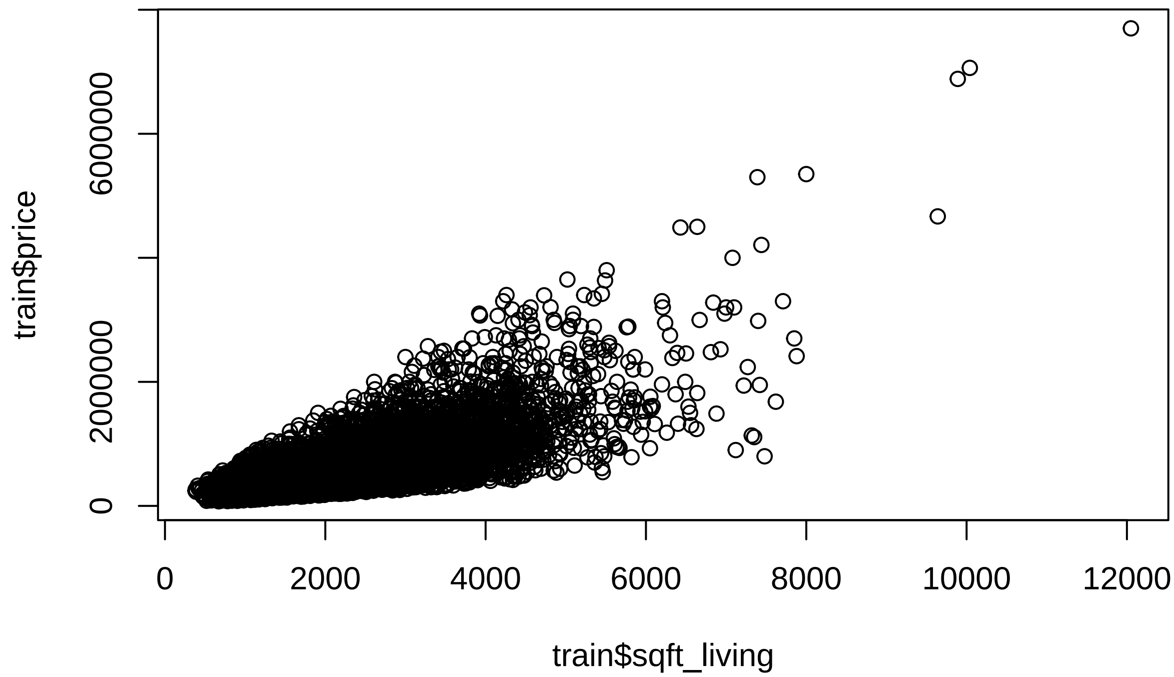
Graphical Data Exploration

```r
options(scipen=5)
hist(train$yr_built, main = "House Manufacture Year", xlab = "Year")
```



This graph shows how many houses were built in their respective years. This helps to show how old the houses are in the area and to show if there is new house development in the area.

```
options(scipen=5)
plot(train$sqft_living, train$price)
```



This graph helps to show the linear relation between the amount of livable square feet in a house and the price of the house.

Linear Regression Model

```
lm1 <- lm(price~., data = train)
summary(lm1)
```

```
##
## Call:
## lm(formula = price ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1284084   -98805    -8806    77663  4306502
##
## Coefficients:
##                    Estimate    Std. Error t value Pr(>|t|)
## (Intercept)   8534230.18471 3260938.83337   2.617  0.00888 **
## bedrooms       -38003.02111    2189.58448 -17.356  < 2e-16 ***
## bathrooms       45782.81145    3608.03949  12.689  < 2e-16 ***
## sqft_living       148.84585       4.90053  30.373  < 2e-16 ***
## sqft_lot            0.09445       0.05094   1.854  0.06376 .
## floors           2818.77103    4009.50513   0.703  0.48205
```

```
## waterfront     563739.75600   19845.31201  28.407  < 2e-16 ***
## view            54977.72256    2390.10249  23.002  < 2e-16 ***
## condition       23908.83581    2612.78557   9.151  < 2e-16 ***
## grade           93722.32514    2407.79701  38.925  < 2e-16 ***
## sqft_above         36.71980       4.88253   7.521 5.72e-14 ***
## yr_built        -2646.07593      81.08220 -32.634  < 2e-16 ***
## yr_renovated       17.97476       4.03507   4.455 8.46e-06 ***
## zipcode          -606.66552      36.75671 -16.505  < 2e-16 ***
## lat            603409.63529   11958.82670  50.457  < 2e-16 ***
## long          -219473.17878   14492.52817 -15.144  < 2e-16 ***
## sqft_living15      18.63577       3.84512   4.847 1.27e-06 ***
## sqft_lot15         -0.31850       0.08160  -3.903 9.52e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 200100 on 17209 degrees of freedom
## Multiple R-squared:  0.6996, Adjusted R-squared:  0.6993
## F-statistic:  2357 on 17 and 17209 DF,  p-value: < 2.2e-16
```

This code segment builds a simple linear regression model. We can see that the R squared is around -.7, which is not a bad value, showing that a decent model can be made. The high RSE is a byproduct of the large price values used in the data set.

```r
pred <- predict(lm1, newdata = test)
correlation <- cor(pred, test$price)
print(paste("correlation:", correlation))
```

Linear Regression Predictions

```
## [1] "correlation: 0.836521467061676"
```

```r
mse <- mean((pred-test$price)^2)
print(paste("mse:",mse))
```

```
## [1] "mse: 42406837222.5279"
```

```r
rmse<-sqrt(mse)
print(paste("rmse:",rmse))
```

```
## [1] "rmse: 205929.204394442"
```

These results for the linear regression prediction are not bad, yielding a high correlation along with mse in relation to the data.

kNN Regression

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

Here we are loading in the caret package needed for kNN Regression.

```
fit <- knnreg(train[,2:18],train[,1],k=3)
pred2 <- predict(fit, test[,2:18])
cor_knn1 <- cor(pred2, test$price)
mse_knn1 <- mean((pred2 - test$price)^2)
print(paste("cor=", cor_knn1))
```

```
## [1] "cor= 0.698168598311517"
```

```
print(paste("mse=", mse_knn1))
```

```
## [1] "mse= 73323097315.6671"
```

Using kNN regression without scaling the data in the beginning, we can see that the results were not as good, yielding a lower correlation.

Scaling the Data Here we are scaling both the train and test data on the means of the training set.

```
train_scaled <- train[, 2:18]
means <- sapply(train_scaled, mean)
stdvs <- sapply(train_scaled, sd)
train_scaled <- scale(train_scaled, center=means, scale=stdvs)
test_scaled <- scale(test[, 2:18], center=means, scale=stdvs)
```

```
fit <- knnreg(train_scaled, train$price, k=3)
pred3 <- predict(fit, test_scaled)
cor_knn2 <- cor(pred3, test$price)
mse_knn2 <- mean((pred3 - test$price)^2)
print(paste("cor=", cor_knn2))
```

Using the Scaled Data

```
## [1] "cor= 0.890643823492618"
```

```
print(paste("mse=", mse_knn2))
```

```
## [1] "mse= 29318578394.2228"
```

```
print(paste("rmse=", sqrt(mse_knn2)))
```

```
## [1] "rmse= 171226.687155428"
```

Now kNN has a higher correlation, so it will be better if we find a better k to use.

```
cor_k <- rep(0, 20)
mse_k <- rep(0, 20)
i <- 1
for (k in seq(1, 39, 2)){
  fit_k <- knnreg(train_scaled,train$price, k=k)
  pred_k <- predict(fit_k, test_scaled)
  cor_k[i] <- cor(pred_k, test$price)
  mse_k[i] <- mean((pred_k - test$price)^2)
  print(paste("k=", k, cor_k[i], mse_k[i]))
  i <- i + 1
}
```

Finding the best k value

```
## [1] "k= 1 0.850127134641021 42490950890.8532"
## [1] "k= 3 0.890643823492618 29318578394.2228"
## [1] "k= 5 0.900171692194114 26916049067.9147"
## [1] "k= 7 0.905129923763618 25959299059.7394"
## [1] "k= 9 0.900692868754319 27161011153.5765"
## [1] "k= 11 0.902096159920442 27189076587.009"
## [1] "k= 13 0.902834066034209 27303594340.4065"
## [1] "k= 15 0.903722204763674 27254678169.0108"
## [1] "k= 17 0.903425242485932 27562836374.8526"
## [1] "k= 19 0.902626903197281 27934616853.6686"
## [1] "k= 21 0.902072508794464 28321237706.8861"
## [1] "k= 23 0.90227924781175 28392614198.9658"
## [1] "k= 25 0.902340975010014 28603081599.3423"
## [1] "k= 27 0.901318654625594 29040069481.1903"
## [1] "k= 29 0.899929737846045 29501567073.7645"
## [1] "k= 31 0.899504142191231 29779428694.8103"
## [1] "k= 33 0.898214231237188 30240265084.8884"
## [1] "k= 35 0.897996853746183 30478833859.074"
## [1] "k= 37 0.897141186686049 30778678187.0016"
## [1] "k= 39 0.895587703063501 31266691282.9151"
```

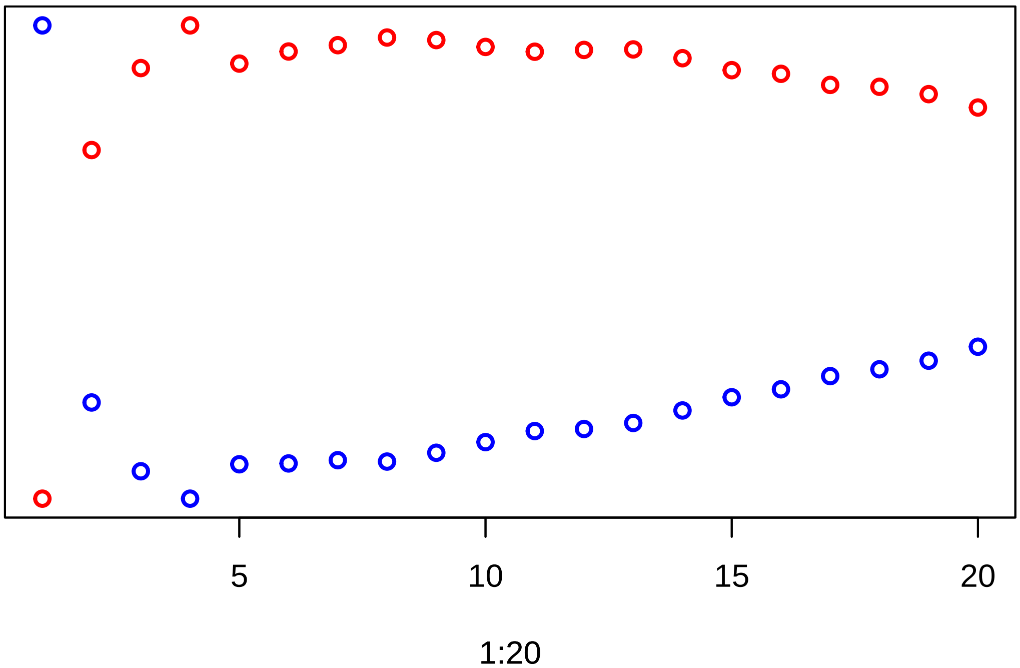
```
plot(1:20, cor_k, lwd=2, col='red', ylab="", yaxt='n')
par(new=TRUE)
plot(1:20, mse_k, lwd=2, col='blue', labels=FALSE, ylab="", yaxt='n')
```

```
## Warning in plot.window(...): "labels" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "labels" is not a graphical parameter
```

```
## Warning in box(...): "labels" is not a graphical parameter

## Warning in title(...): "labels" is not a graphical parameter
```



Looking at the graph, we can see that when k = 4, the correlation which is shown in red is higher than the mse which is shown in blue.

```
which.min(mse_k)
```

```
## [1] 4
```

```
which.max(cor_k)
```

```
## [1] 4
```

This code here helps to verify what was deduced.

```
fit <- knnreg(train_scaled, train$price, k=4)
pred4 <- predict(fit, test_scaled)
cor_knn3 <- cor(pred4, test$price)
mse_knn3 <- mean((pred4 - test$price)^2)
print(paste("cor=", cor_knn3))
```

Using k=4 for kNN Regression

```
## [1] "cor= 0.897511918248958"
```

```
print(paste("mse=", mse_knn3))
```

```
## [1] "mse= 27532079918.9509"
```

Decision Tree Regression

```
library(tree)

library(MASS)
```

Here we are installing the necessary libraries for using decision trees.

```
tree1 <- tree(price~., data = train)
summary(tree1)
```

Using Tree

```
##
## Regression tree:
## tree(formula = price ~ ., data = train)
## Variables actually used in tree construction:
## [1] "grade"       "lat"          "sqft_living" "long"
## Number of terminal nodes:  12
## Residual mean deviance:  40490000000 = 6.971e+14 / 17220
## Distribution of residuals:
##     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1665000    -99640    -24990         0     70360   2416000
```

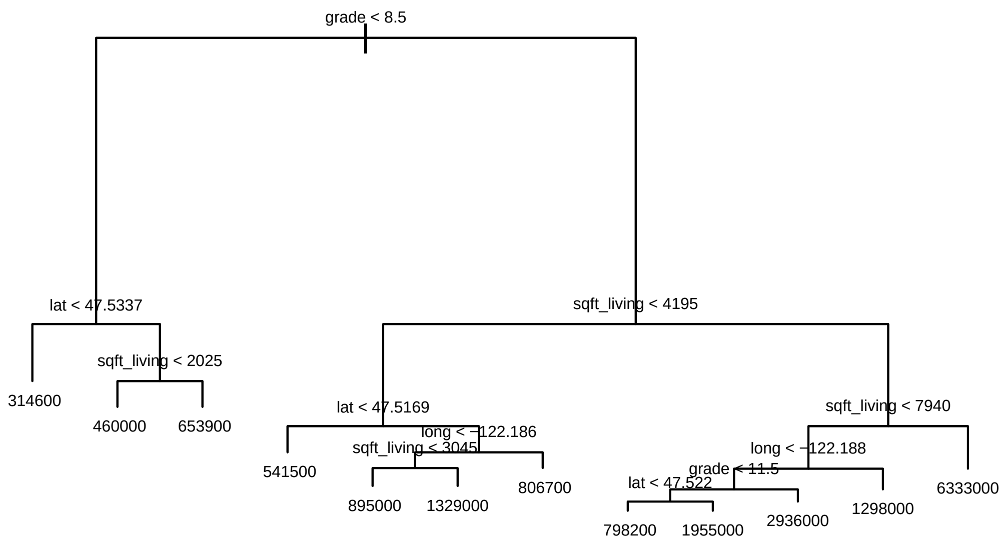
```
pred <- predict(tree1, newdata=test)
print(paste('correlation:', cor(pred, test$price)))
```

```
## [1] "correlation: 0.799856802049253"
```

```
rmse_tree <- sqrt(mean((pred-test$price)^2))
print(paste('rmse:', rmse_tree))
```

```
## [1] "rmse: 229416.192423866"
```

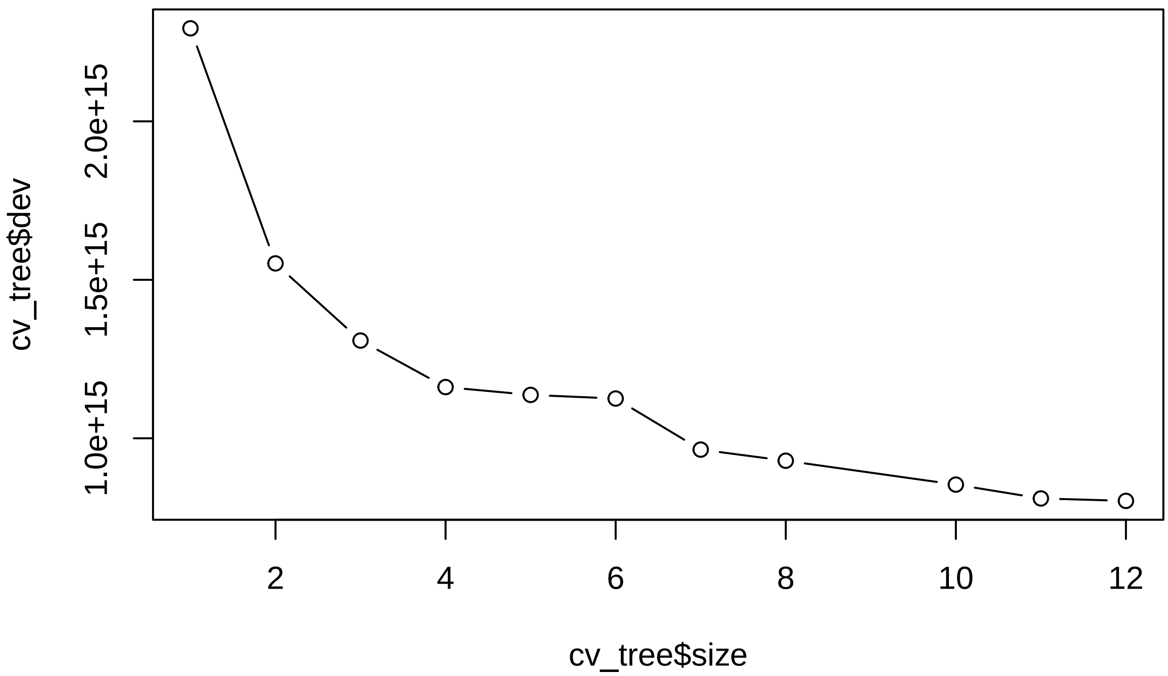
```
plot(tree1)
text(tree1, cex=0.5, pretty=0)
```



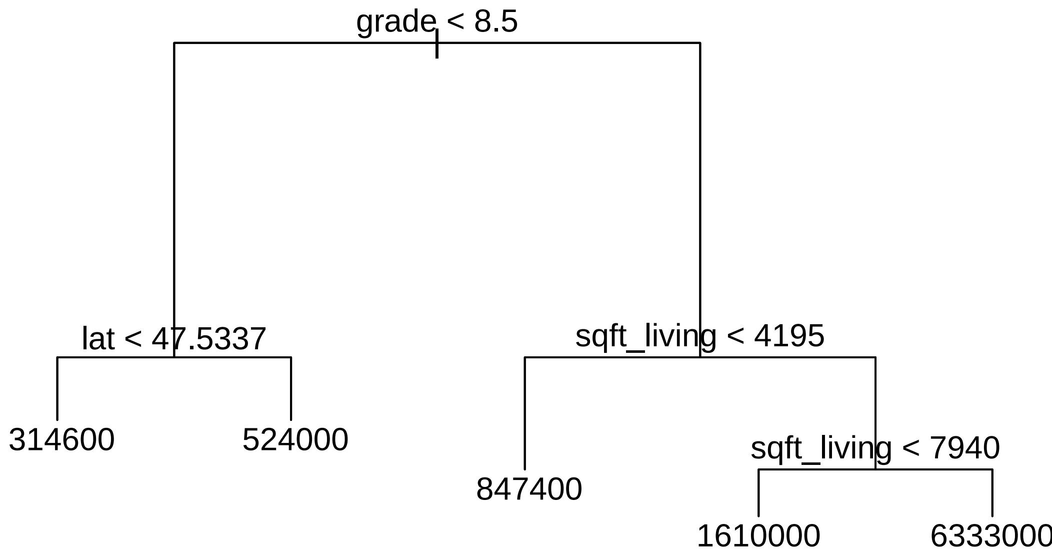
Here we can see that the correlation is not as good as the linear regression model.

Cross Validating Due to there being a deviance in the tree sizes, it is better if we work to use a smaller set of nodes for less variance.

```
cv_tree <- cv.tree(tree1)
plot(cv_tree$size, cv_tree$dev, type='b')
```




```
tree_pruned <- prune.tree(tree1, best=5)
plot(tree_pruned)
text(tree_pruned, pretty=0)
```



Pruning the Tree

Here the tree is pruned to the 5 best values to reduce variance with the previous out liars.

```
pred_pruned <- predict(tree_pruned, newdata=test)
cor_pruned <- cor(pred_pruned, test$price)
rmse_pruned <- rmse_pruned <- sqrt(mean((pred_pruned-test$price)^2))
print(paste('correlation of pruned tree:', cor_pruned))
```

Using the Pruned Tree to test

```
## [1] "correlation of pruned tree: 0.700377679512745"
```

```
print(paste('rmse of pruned tree:', rmse_pruned))
```

```
## [1] "rmse of pruned tree: 273528.945735084"
```

In this case, the pruned tree actually gave a worse correlation than the unpruned tree. While the tree is simpler to read and understand, it still yielded weaker results.

```
library(randomForest)
```

Random Forest

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

Here we have installed the library necessary for using random forests.

```r
rf <- randomForest(price~., data = train, importance=TRUE)
rf
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = train, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 16443404835
##                     % Var explained: 87.65
```

```r
pred_rf <- predict(rf, newdata=test)
cor_rf <- cor(pred_rf, test$price)
print(paste('corr:', cor_rf))
```

```
## [1] "corr: 0.936190489273414"
```

```r
rmse_rf <- sqrt(mean((pred_rf-test$price)^2))
print(paste('rmse:', rmse_rf))
```

```
## [1] "rmse: 132707.74412173"
```

Here we predict on the random forest and can see that it yields a much higher results than before.

```r
bag <- randomForest(price~., data = train, mtry=13)
bag
```

Predicting after bagging

```
##
## Call:
##  randomForest(formula = price ~ ., data = train, mtry = 13)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 16284853572
##                     % Var explained: 87.77
```

```
pred_bag <- predict(bag, newdata=test)
cor_bag <- cor(pred_bag, test$price)
rmse_bag <- sqrt(mean((pred_bag-test$price)^2))
print(paste('corr:', cor_bag))
```

```
## [1] "corr: 0.935166418754952"
```

```
print(paste('rmse:', rmse_bag))
```

```
## [1] "rmse: 133124.254423774"
```

Here we can see that predicting with bagging have slightly lower results than the random forest.

Analysis

From analyzing the results, we can see that the Decision Tree yields the highest correlation with the data at a correlation of almost 0.95 while linear regression yielded only a 0.84 correlation. kNN regression also yielded a higher correlation than linear regression, showing how both kNN Regression and Decision Trees are both more suited to analyze this data set. The reason for this could be that calculating the prices from all the variables and the similarity in most of the variables were more suited for both of the similarity algorithms rather than a simple linear regression.