Housing Price Regression

Ronen Reouveni

9/10/2020

Introduction

Real estate markets have long been dominated by real estate agents making decisions about the value of a property. Although agents often base their estimates on comps, the dollar amount a property was sold for near to the one in question, the process can be arbitrary. It is very difficult in this setting for a human to ascertain how much an additional bathroom will add in value. Or how much a corner home vs street home will differ in cost. Often times it is not possible to find a comparison property that is similar to the property in question. This leads to properties being sold and/or purchased for amounts that are potentially significantly different to their true value. These discrepancies are part of the reason that a very experienced investor can find properties that are undervalued. They are able to notice and understand when a listed price is not taking into consideration some underlying benefit a property has compared to how it is priced. Furthermore, buyers can easily lose money by overspending because a comparison property was in fact better than the property they purchased for the same price. There is a need for a more consistent approach to predicting a properties sale value.

Improving how properties are priced has many benefits. Stakeholders include buyers, sellers, real estate agents, and investors. However, the manner in which a solution is implemented may benefit some parties more than others. For example, if one party had the true value and the others did not they could easily exploit this. From a buyer's perspective, knowing the true value would allow a party to easily hone in on properties that are undervalued and avoid properties that are overvalued. From a seller's perspective they would know what a fair price would be for the property. Furthermore, there is the situation in which everyone will have access to this information. One of the most beneficial uses for a real estate agent is pricing the home. If this becomes obsolete due to technological advances then a key real estate agent skill is no longer needed. They will only be beneficial for writing offers and finding the properties to begin with. A critical point of any technological innovation is who will have access to it and who will not.

There is a significant opportunity for technology to make drastic improvements to this problem. Zillow, an online real estate firm, now gives an estimate range for the value of a home. Not coincidentally, this estimate is more often than not the listed price of the property. There are many other companies leveraging analytics to estimate the value of a home. Some of these firms are focused on investors. For example, finding undervalued properties and purchasing them. Then they allow high net worth investors to buy pieces of a real estate portfolio. However, there is still a large need for this to be improved. As technology and innovation improve, the accuracy of sophisticated housing price predictions also improves. Competitive models should be available to be used by simple buyers and sellers. Facilitating more honesty in the real estate market will only help bring about positive change in how deals are made and property changes hands.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```r
library(Metrics)
library(randomForestExplainer)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```r
library(xgboost)
library(randomcoloR)
library(FactoMineR)
library(MASS)
library(plyr)
library(ade4)
```

```
##
## Attaching package: 'ade4'
```

```
## The following object is masked from 'package:FactoMineR':
##
##     reconst
```

```r
library(arules)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'arules'
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```r
library(arulesCBA)
library(e1071)
library(reshape2)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```r
library(ggpubr)
```

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

```
##
## Attaching package: 'ggplot2'
```

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

```
##
## Attaching package: 'ggpubr'

## The following object is masked from 'package:plyr':
##
##     mutate
```

```
library(polycor)
library(robustHD)
```

```
## Loading required package: perry

## Loading required package: parallel

## Loading required package: robustbase

##
## Attaching package: 'perry'

## The following object is masked from 'package:Metrics':
##
##     mape
```

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

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

## The following object is masked from 'package:arules':
##
##     size
```

```
library(Rfast)
```

```
## Loading required package: Rcpp

## Loading required package: RcppZiggurat

##
## Attaching package: 'Rfast'

## The following objects are masked from 'package:robustbase':
##
##     colMedians, rowMedians

## The following object is masked from 'package:Metrics':
##
##     auc
```

```
library(glmnet)
```

```
## Loaded glmnet 4.0-2
```

```
library(rpart)
library(rattle)
```

```
## Loading required package: tibble
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
##
## Attaching package: 'rattle'
```

```
## The following object is masked from 'package:xgboost':
##
##     xgboost
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(curl)
library(cluster)
library(wordspace)
library(dendextend)
```

```
##
## ----------------------
## Welcome to dendextend version 1.14.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ----------------------
```

```
##
## Attaching package: 'dendextend'

## The following object is masked from 'package:rpart':
##
##     prune

## The following object is masked from 'package:ggpubr':
##
##     rotate

## The following object is masked from 'package:stats':
##
##     cutree
```

```
library(FNN)
```

```
##
## Attaching package: 'FNN'

## The following objects are masked from 'package:Rfast':
##
##     knn, knn.cv
```

```
library(arulesViz)
```

```
## Loading required package: grid

## Registered S3 method overwritten by 'seriation':
##   method         from
##   reorder.hclust gclus
```

```
library(klaR)
library(packcircles)
```

```
##
## Attaching package: 'packcircles'

## The following object is masked from 'package:MASS':
##
##     bacteria
```

Analysis and Models

About the Data

The data consists of information on homes sold in various neighborhoods in Iowa. The data contains 1,460 rows of observations that contain the sale price and 1,459 that do not. These two files are combined so that

data pre-processing is applied to all observations. The final submission to Kaggle is the sale price for the 1,459 observations where the sale price was not given. The data starts with 79 predictors and about 14,000 missing values. The Kaggle competition for this data set is measured in root mean squared log error, rmsle, and therefore that is the metric that will be used for evaluation.

```
#set seed for reproducabiliy and load data
set.seed(5948)
housingData <- read.csv("/Users/ronenreouveni/Desktop/projData/train.csv")
housingData_test <- read.csv("/Users/ronenreouveni/Desktop/projData/test.csv")

#remove ID and bring sale price into its own vector
housingData <- housingData[,-1]
targetValue <- housingData$SalePrice
housingData <- housingData[,-which(colnames(housingData)=='SalePrice')]
housingData_test <- housingData_test[,-1]

#combine both data files into one data frame to facilitate pre processing
fullFrame <- rbind(housingData,housingData_test)
```

The unique function can be used to get an idea of the data and also find any values that do not belong.

```
#check for mistakes and get a sense
unique(fullFrame$MSSubClass)
```

```
##  [1]  60  20  70  50 190  45  90 120  30  85  80 160  75 180  40 150
```

```
unique(fullFrame$OverallQual)
```

```
##  [1]  7  6  8  5  9  4 10  3  1  2
```

```
unique(fullFrame$OverallCond)
```

```
## [1] 5 8 6 7 4 2 3 9 1
```

```
unique(fullFrame$BedroomAbvGr)
```

```
## [1] 3 4 1 2 0 5 6 8
```

```
unique(fullFrame$TotRmsAbvGrd)
```

```
##  [1]  8  6  7  9  5 11  4 10 12  3  2 14 13 15
```

```
unique(fullFrame$Fireplaces)
```

```
## [1] 0 1 2 3 4
```

```
unique(fullFrame$FullBath)
```

```
## [1] 2 1 3 0 4
```

```r
unique(fullFrame$GarageCars)
```

```
## [1]  2  3  1  0  4  5 NA
```

```r
unique(fullFrame$BsmtFullBath)
```

```
## [1]  1  0  2  3 NA
```

```r
unique(fullFrame$BsmtHalfBath)
```

```
## [1]  0  1  2 NA
```

```r
unique(fullFrame$HalfBath)
```

```
## [1] 1 0 2
```

```r
unique(fullFrame$KitchenAbvGr)
```

```
## [1] 1 2 3 0
```

```r
table(fullFrame$Neighborhood)
```

```
##
## Blmngtn Blueste  BrDale BrkSide ClearCr CollgCr Crawfor Edwards Gilbert  IDOTRR
##      28      10      30     108      44     267     103     194     165      93
## MeadowV Mitchel   NAmes NoRidge NPkVill NridgHt  NWAmes OldTown  Sawyer SawyerW
##      37     114     443      71      23     166     131     239     151     125
## Somerst StoneBr   SWISU  Timber Veenker
##     182      51      48      72      24
```

```r
#load factors as factors
fullFrame$MSSubClass <- as.factor(fullFrame$MSSubClass)
fullFrame$OverallQual <- as.factor(fullFrame$OverallQual)
fullFrame$OverallCond <- as.factor(fullFrame$OverallCond)
fullFrame$BedroomAbvGr <- as.factor(fullFrame$BedroomAbvGr)
fullFrame$TotRmsAbvGrd <- as.factor(fullFrame$TotRmsAbvGrd)
fullFrame$Fireplaces <- as.factor(fullFrame$Fireplaces)
fullFrame$FullBath <- as.factor(fullFrame$FullBath)
fullFrame$GarageCars <- as.factor(fullFrame$GarageCars)
fullFrame$BsmtFullBath <- as.factor(fullFrame$BsmtFullBath)
fullFrame$BsmtHalfBath <- as.factor(fullFrame$BsmtHalfBath)
fullFrame$HalfBath <- as.factor(fullFrame$HalfBath)
fullFrame$BedroomAbvGr <- as.factor(fullFrame$BedroomAbvGr)
fullFrame$KitchenAbvGr <- as.factor(fullFrame$KitchenAbvGr)
```

```r
#make sure all years are present and month sold is 12 months
unique(fullFrame$YrSold)
```

```
## [1] 2008 2007 2006 2009 2010
```

```
unique(fullFrame$MoSold)
```

```
##  [1]  2  5  9 12 10  8 11  4  1  7  3  6
```

```
#make sure year built/remod all makes sense
summary(fullFrame$YearBuilt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1872    1954    1973    1971    2001    2010
```

```
summary(fullFrame$YearRemodAdd)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1950    1965    1993    1984    2004    2010
```

```
#notice one value is 2207
summary(fullFrame$GarageYrBlt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    1895    1960    1979    1978    2002    2207     159
```

```
#assume its 2007
fullFrame$GarageYrBlt[which(fullFrame$GarageYrBlt>2011)] <- 2007
summary(fullFrame$GarageYrBlt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    1895    1960    1979    1978    2002    2010     159
```

```
#Make time variables factors
fullFrame$YearRemodAdd <- as.factor(fullFrame$YearRemodAdd)
fullFrame$YrSold <- as.factor(fullFrame$YrSold)
fullFrame$MoSold <- as.factor(fullFrame$MoSold)
fullFrame$YearBuilt <- as.factor(fullFrame$YearBuilt)
fullFrame$GarageYrBlt <- as.factor(as.character(fullFrame$GarageYrBlt))
```

The following time-based predictors have two many values to leave in as a factor; it will have too many levels. Therefore, discretization is used.

```
#discretize time based entries with many values
fullFrame$YearBuilt <- discretize(as.numeric(as.character(fullFrame$YearBuilt)), breaks = 10)
fullFrame$YearRemodAdd <- suppressWarnings(discretize(
                            as.numeric(as.character(fullFrame$YearRemodAdd)),
                            breaks = 10))
fullFrame$GarageYrBlt <- discretize(as.numeric(as.character(fullFrame$GarageYrBlt)), breaks = 10)
```

Simple function to give the number of missing values per predictor.

```r
#lets see how many NA's we are dealing with
findMissing <- function(x) {
  i <- 1
  while (i < ncol(x)+1){
    print(paste(colnames(x)[i],length(x[,i][!complete.cases(x[,i])])))
    i <- i + 1
  }
}

findMissing(fullFrame)
```

```
## [1] "MSSubClass 0"
## [1] "MSZoning 4"
## [1] "LotFrontage 486"
## [1] "LotArea 0"
## [1] "Street 0"
## [1] "Alley 2721"
## [1] "LotShape 0"
## [1] "LandContour 0"
## [1] "Utilities 2"
## [1] "LotConfig 0"
## [1] "LandSlope 0"
## [1] "Neighborhood 0"
## [1] "Condition1 0"
## [1] "Condition2 0"
## [1] "BldgType 0"
## [1] "HouseStyle 0"
## [1] "OverallQual 0"
## [1] "OverallCond 0"
## [1] "YearBuilt 0"
## [1] "YearRemodAdd 0"
## [1] "RoofStyle 0"
## [1] "RoofMatl 0"
## [1] "Exterior1st 1"
## [1] "Exterior2nd 1"
## [1] "MasVnrType 24"
## [1] "MasVnrArea 23"
## [1] "ExterQual 0"
## [1] "ExterCond 0"
## [1] "Foundation 0"
## [1] "BsmtQual 81"
## [1] "BsmtCond 82"
## [1] "BsmtExposure 82"
## [1] "BsmtFinType1 79"
## [1] "BsmtFinSF1 1"
## [1] "BsmtFinType2 80"
## [1] "BsmtFinSF2 1"
## [1] "BsmtUnfSF 1"
## [1] "TotalBsmtSF 1"
## [1] "Heating 0"
## [1] "HeatingQC 0"
## [1] "CentralAir 0"
## [1] "Electrical 1"
```

```
## [1] "X1stFlrSF 0"
## [1] "X2ndFlrSF 0"
## [1] "LowQualFinSF 0"
## [1] "GrLivArea 0"
## [1] "BsmtFullBath 2"
## [1] "BsmtHalfBath 2"
## [1] "FullBath 0"
## [1] "HalfBath 0"
## [1] "BedroomAbvGr 0"
## [1] "KitchenAbvGr 0"
## [1] "KitchenQual 1"
## [1] "TotRmsAbvGrd 0"
## [1] "Functional 2"
## [1] "Fireplaces 0"
## [1] "FireplaceQu 1420"
## [1] "GarageType 157"
## [1] "GarageYrBlt 159"
## [1] "GarageFinish 159"
## [1] "GarageCars 1"
## [1] "GarageArea 1"
## [1] "GarageQual 159"
## [1] "GarageCond 159"
## [1] "PavedDrive 0"
## [1] "WoodDeckSF 0"
## [1] "OpenPorchSF 0"
## [1] "EnclosedPorch 0"
## [1] "X3SsnPorch 0"
## [1] "ScreenPorch 0"
## [1] "PoolArea 0"
## [1] "PoolQC 2909"
## [1] "Fence 2348"
## [1] "MiscFeature 2814"
## [1] "MiscVal 0"
## [1] "MoSold 0"
## [1] "YrSold 0"
## [1] "SaleType 1"
## [1] "SaleCondition 0"
```

Many of the missing values have meaning. According to the data dictionary majority of NA's indicate zero or none. However, many are truly NA's where the value is unknown. One of the largest is Lot Frontage. Instead of arbitrarily filling in the missing values or even using the median value for each neighborhood it is possible to use machine learning.

Naive Bayes will be used to fill in the missing Lot Frontage values. Lot Frontage is first discretized then Naive Bayes is used to predict which bucket a missing value would fall into. The median value of the predicted bucket is then used to fill in the missing value.

486 missing Lot Frontage values are replaced with this process.

```
#remove NA columns to simplify
fullFrame_NA1 <- fullFrame[,-c(which(colnames(fullFrame)=="FireplaceQu")
                               ,which(colnames(fullFrame)=="PoolQC")
                               ,which(colnames(fullFrame)=="Fence")
                               ,which(colnames(fullFrame)=="MiscFeature")
                               ,which(colnames(fullFrame)=="PoolArea")
```

```
                                ,which(colnames(fullFrame)=="Alley")
)]


sum(is.na(fullFrame$LotFrontage))
```

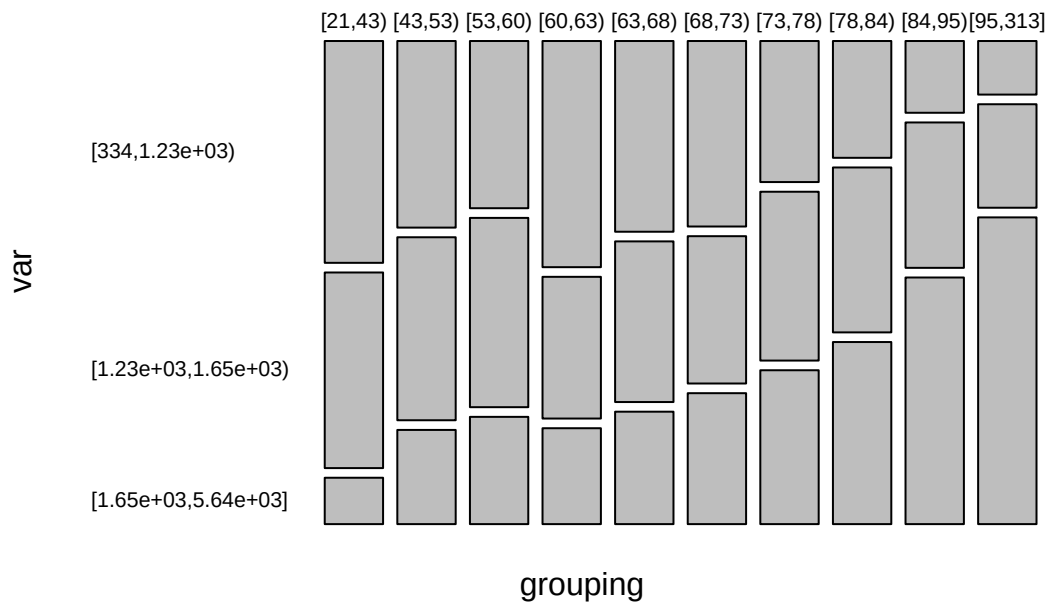
```
## [1] 486
```

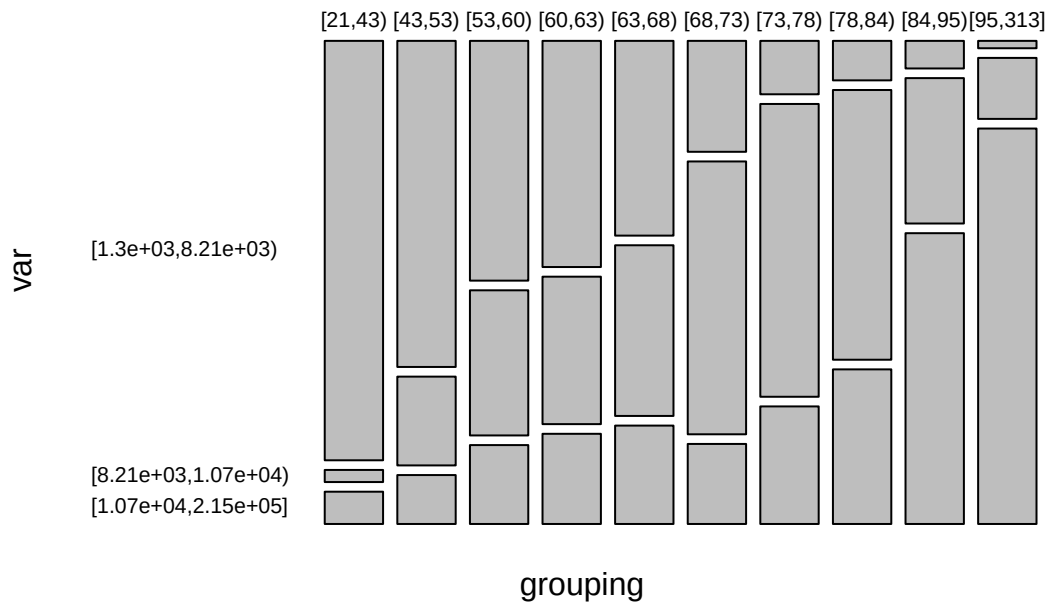
```
#Use NB to fill missing Lot Frontage values
fullFrame_NA1$LotFrontage <- discretize(fullFrame_NA1$LotFrontage, breaks = 10)
NBfullFrame <- fullFrame_NA1
fullFrame_NA1 <- suppressWarnings(discretizeDF(fullFrame_NA1))

#NaiveBayes replacement method
nbaLots <- naiveBayes(LotFrontage ~ ., data = fullFrame_NA1)


#Predict values and revalue levels
NBApreds <- predict(nbaLots, fullFrame_NA1[which(is.na(fullFrame_NA1$LotFrontage)),])
NBApreds <- revalue(NBApreds, c("[21,43)" = 32, "[43,53)"= 48, "[53,60)"=56,
                                "[60,63)"=62, "[63,68)"=65,
                                "[68,73)"= 70, "[73,78)"=76 ,'[78,84)'=83,
                                '[84,95)'=90, '[95,313]' = 100))
NBApreds <- as.numeric(levels(NBApreds))[NBApreds]

#get indicies of missing locations
lfNA <- which(is.na(fullFrame$LotFrontage))

#use a loop to fill in the missing values
k <- 1
for (i in lfNA) {
  fullFrame$LotFrontage[i] <- NBApreds[k]
  k <- k + 1
}

#There are no more NA's
sum(is.na(fullFrame$LotFrontage))
```

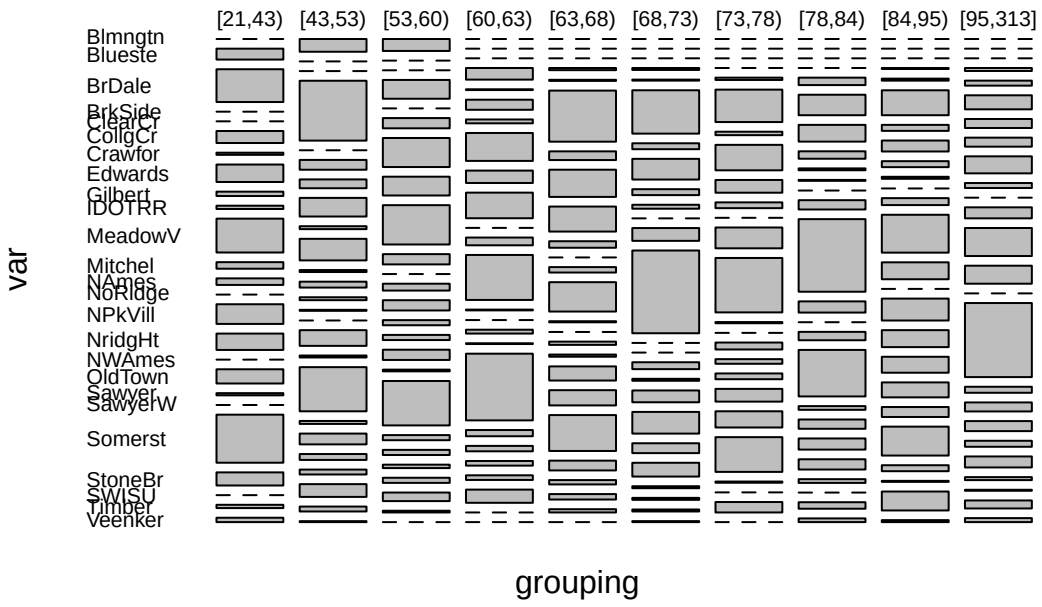
```
## [1] 0
```

The model is trained again to show some visualizations. Three mosaic plots are shown, one for GrLivArea, one for LotArea, and one for Neighborhood. All of these are the discretized versions.

```
nbViz <- NaiveBayes(LotFrontage ~ GrLivArea + LotArea + Neighborhood,
                    data = fullFrame_NA1, usekernel = TRUE)
plot(nbViz,las = 1 )
```


Naive Bayes Plot
[21,43)
[43,53)
[53,60)
[60,63)
[63,68)
[68,73)
[73,78)
[78,84)
[84,95)
[95,313]
[334,1.23e+03)
[1.23e+03,1.65e+03)
[1.65e+03,5.64e+03]
var
grouping


Naive Bayes Plot
[21,43)
[43,53)
[53,60)
[60,63)
[63,68)
[68,73)
[73,78)
[78,84)
[84,95)
[95,313]
[1.3e+03,8.21e+03)
[8.21e+03,1.07e+04)
[1.07e+04,2.15e+05]
var
grouping



Identifying which columns are numeric and which are categorical is essential for upcoming pre processing and visualization.

```
#split dataframe by type
ints <- c()
cats <- c()
for (i in 1:ncol(fullFrame)) {
  ifelse((is.numeric(fullFrame[,i])) ,ints[i] <- i ,cats[i] <- i)
}

ints <- sort(ints)
cats <- sort(cats)
```

Fill in remaining NA's with the mode.

```
#simple function to find mode
modeFunc <- function(x) {
  uni <- unique(x)
  uni[which.max(tabulate(match(x, uni)))]
}

#fill wiht obvious value
fullFrame$Functional[is.na(fullFrame$Functional)] <- 'Typ'
fullFrame$Electrical[is.na(fullFrame$Electrical)] <- 'SBrkr'
fullFrame$KitchenQual[is.na(fullFrame$KitchenQual)] <- 'TA'

#use mode function to fill
fullFrame$Exterior1st[is.na(fullFrame$Exterior1st)] <- modeFunc(fullFrame$Exterior1st)
fullFrame$Exterior2nd[is.na(fullFrame$Exterior2nd)] <- modeFunc(fullFrame$Exterior2nd)
fullFrame$SaleType[is.na(fullFrame$SaleType)] <- modeFunc(fullFrame$SaleType)
```

```
#fill missing garage area with 0
fullFrame$GarageArea[is.na(fullFrame$GarageArea)] <- 0


#replace missing zoning values by mode grouped by neighberhood
moveFrame <- aggregate(fullFrame$MSZoning, by=list(Neighborhood=fullFrame$Neighborhood),
                       FUN=modeFunc)
fullFrame <- join(fullFrame, moveFrame, by = 'Neighborhood')
fullFrame$MSZoning[is.na(fullFrame$MSZoning)] <- fullFrame$x[is.na(fullFrame$MSZoning)]

#remove the added redundent column
fullFrame <- fullFrame[,-ncol(fullFrame)]
```

According to the data dictionary the remaining missing values are all truly 0 or none. For example, NA for fence means there is no fence on the property. Two simple for loops are able to fill in the rest of the missing values with 0 or none.

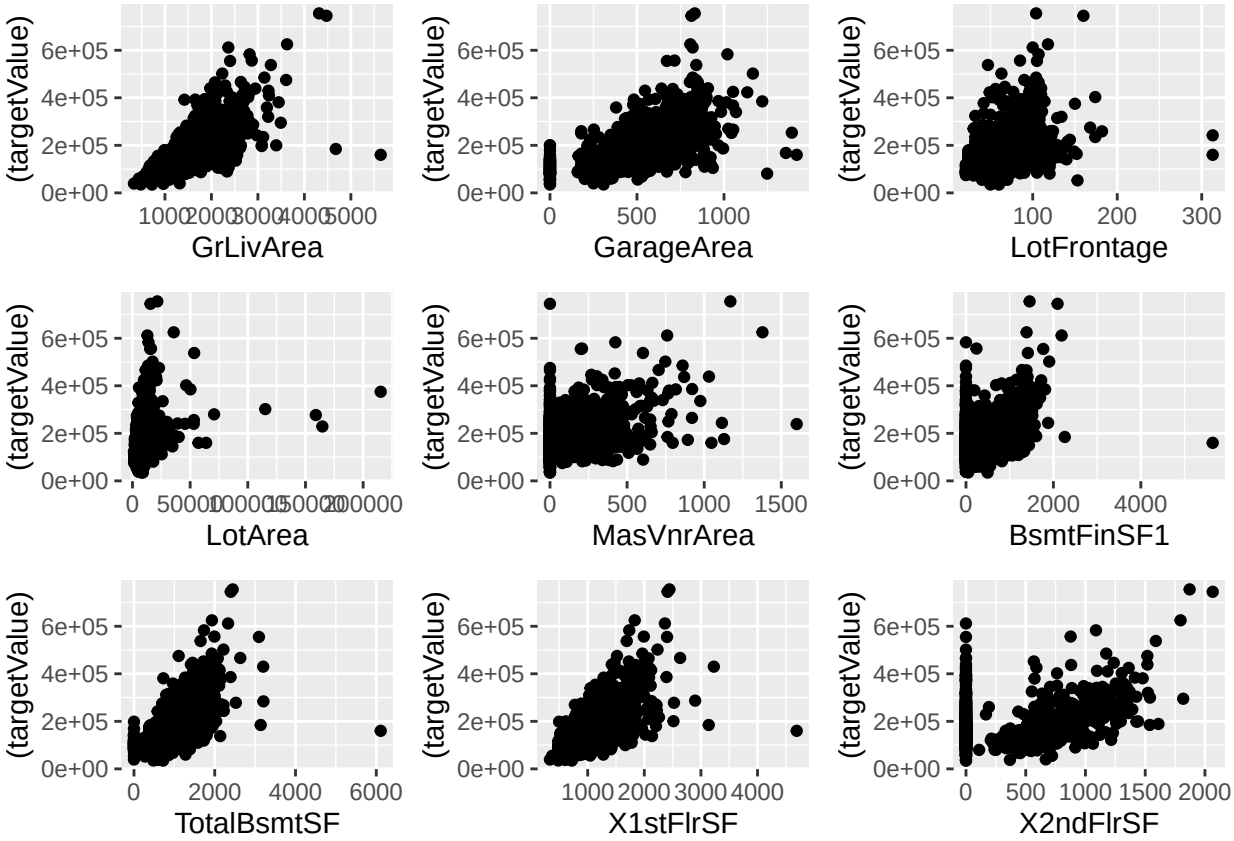
```
for (i in ints) {
  fullFrame[which(is.na(fullFrame[,i])),i] <- 0
}

for (i in cats) {
  fullFrame[,i] <- as.character(fullFrame[,i])
  fullFrame[which(is.na(fullFrame[,i])),i] <- 'none'
  fullFrame[,i] <- as.factor(fullFrame[,i])
}

sum(is.na(fullFrame))
```

```
## [1] 0
```

Next, select variables are chosen to plot against sale price. This led to the discovery of many outliers but in all cases removing them only lowered the accuracy.

```
#resplit the data frame
ints <- c()
cats <- c()
for (i in 1:ncol(fullFrame)) {
  ifelse((is.numeric(fullFrame[,i])) ,ints[i] <- i ,cats[i] <- i)
}

ints <- sort(ints)
cats <- sort(cats)

#create new dataframe with only the target value containing values
fullFrame_GraphScat <- cbind(fullFrame[1:length(targetValue),ints], targetValue)

a <- ggplot(fullFrame_GraphScat, aes(x=GrLivArea, y=(targetValue))) +
  geom_point()

b <- ggplot(fullFrame_GraphScat, aes(x=GarageArea, y=(targetValue))) +
  geom_point()
```

```
e <- ggplot(fullFrame_GraphScat, aes(x=LotFrontage, y=(targetValue))) +
  geom_point()

f <- ggplot(fullFrame_GraphScat, aes(x=LotArea, y=(targetValue))) +
  geom_point()

h <- ggplot(fullFrame_GraphScat, aes(x=MasVnrArea, y=(targetValue))) +
  geom_point()

i <- ggplot(fullFrame_GraphScat, aes(x=BsmtFinSF1, y=(targetValue))) +
  geom_point()

j <- ggplot(fullFrame_GraphScat, aes(x=TotalBsmtSF, y=(targetValue))) +
  geom_point()

k <- ggplot(fullFrame_GraphScat, aes(x=X1stFlrSF, y=(targetValue))) +
  geom_point()

l <- ggplot(fullFrame_GraphScat, aes(x=X2ndFlrSF, y=(targetValue))) +
  geom_point()

#display the viz
ggarrange(a,b,e,f,h,i,j,k,l)
```



To check that there are not major differences between the Kaggle train and test data a Random Forest is used to compare the two sets. A label is added to each data set indicating which set it comes from. Then a

Random Forest is used to try and predict which set an observation comes from. If this can be done with a high accuracy then there are major differences between the two sets.

Since the accuracy of this classification model is only 50% it means that the two data sets are similar. Although the model cannot predict which data set an observation comes from importance can still be visualized. If there was an issue this would illuminate which predictors were responsible.

This model is used in the pre-processing section because it drives if more pre-processing is needed to correct the issue between the two data sets. Once it is confirmed that there is no issue, feature engineering is safely possible and then moving on to using models to predict sales price.

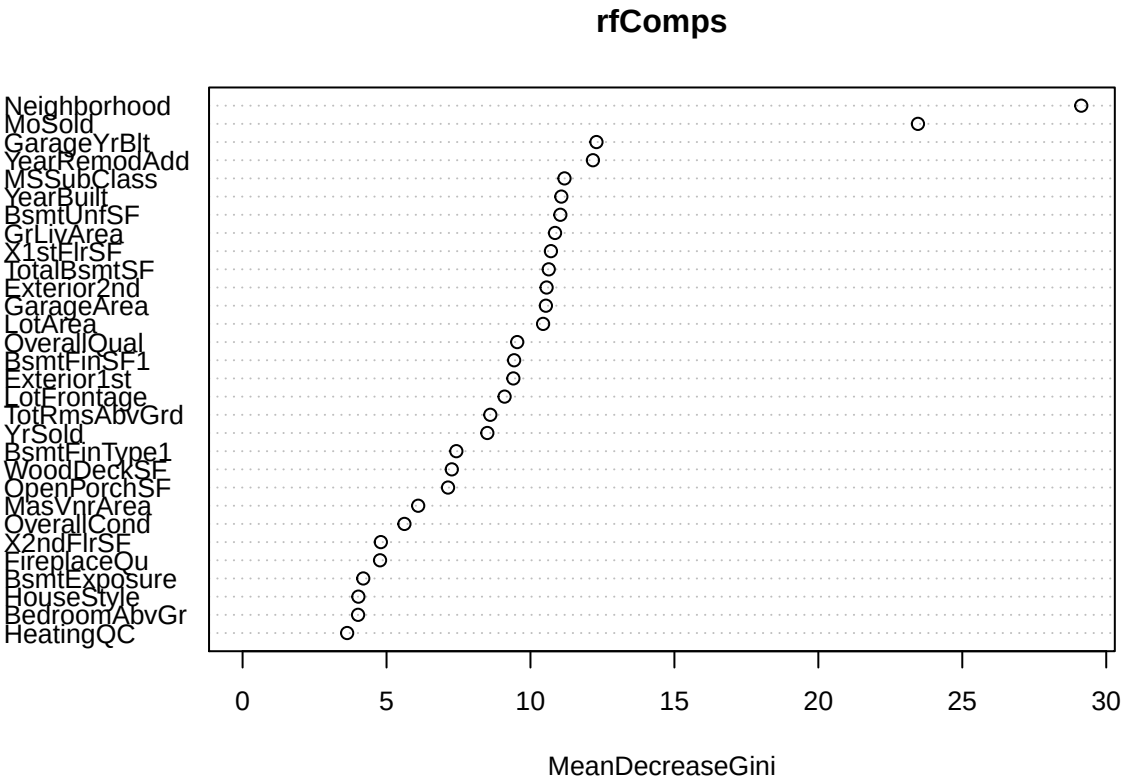
```
#This shows that the data comes from the same distribution,
#otherwise a RF could classify better than 50/50
dataTrain <- fullFrame[1:length(targetValue),]
dataTest <- fullFrame[(length(targetValue)+1):nrow(fullFrame),]
dataTrain$labs <- rep(0, nrow(dataTrain))
dataTest$labs <- rep(1, nrow(dataTest))
dataFull <- rbind(dataTrain, dataTest)
dataFull$labs <- as.factor(dataFull$labs)
idx <- sample(1:nrow(dataFull), .25*nrow(dataFull))
rfComps <- randomForest(dataFull[idx,-ncol(dataFull)], dataFull$labs[idx])
predsComp <- predict(rfComps, dataFull[-idx,-ncol(dataFull)])
table(predsComp, dataFull$labs[-idx])
```

```
##
## predsComp   0   1
##         0 360 360
##         1 736 734
```

```
mean(predsComp == dataFull$labs[-idx])
```

```
## [1] 0.4995434
```

```
varImpPlot(rfComps, sort = TRUE, cex = .85)
```



13 new features were created to improve the models.

1. TotalSf: the sum of all square foot predictors
2. T_bathrooms: the total number of bathrooms in a home
3. Tot_out: the total outside deck and porch area
4. Sf_Quality: an interaction between TotalSf and Quality
5. lotSizeRatio: TotalSf / by LotArea
6. QualCond: interaction between Quality and Condition
7. Price_sf: an interaction of median price / square foot in each neighborhood and overall quality
8. Price_SF_land: median price / square foot of land in each neighborhood
9. Price_SF_land_: an interaction of median price / square foot of land in each neighborhood and overall quality
10. Is_pool: binary value if there is a pool
11. Is_2stories: binary value if there is 2 stories
12. Is_garage: binary value if there is a garage
13. Is_bsmt: binary value if there is a basement

```
ncol(fullFrame)
```

```
## [1] 79
```

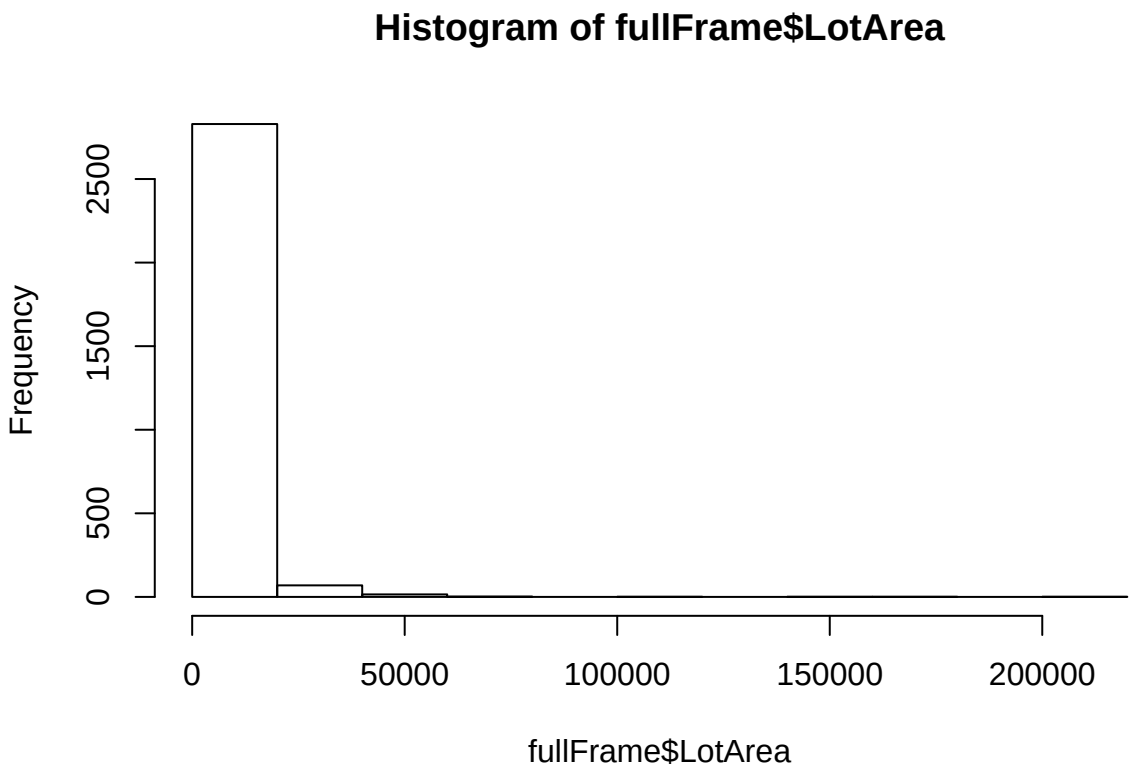
```
fullFrame$TotalSf <- fullFrame$TotalBsmtSF + fullFrame$X1stFlrSF + fullFrame$X2ndFlrSF

fullFrame$T_bathrooms <- as.numeric(fullFrame$FullBath) +
                        .5*as.numeric(fullFrame$HalfBath) +
                          as.numeric(fullFrame$BsmtFullBath) +
                          .5*as.numeric(fullFrame$BsmtHalfBath)
fullFrame$T_bathrooms <- as.factor(fullFrame$T_bathrooms)
```

```
fullFrame$Tot_out <- fullFrame$WoodDeckSF + fullFrame$OpenPorchSF + fullFrame$EnclosedPorch +
                          fullFrame$X3SsnPorch + fullFrame$ScreenPorch


fullFrame$Sf_Quality <-fullFrame$TotalSf^as.integer(fullFrame$OverallQual)

fullFrame$lotSizeRatio <-fullFrame$TotalSf/fullFrame$LotArea

fullFrame$QualCond <- as.integer(fullFrame$OverallCond)^as.integer(fullFrame$OverallQual)

Price_SF <- as.numeric((targetValue))/(fullFrame$GrLivArea[1:length(targetValue)])
buildData <- data.frame(fullFrame$Neighborhood[1:length(targetValue)], Price_SF)
priceKey <- as.data.frame(tapply(buildData$Price_SF,
                                 buildData$fullFrame.Neighborhood.1.length.targetValue.., median))
priceKey <- cbind(priceKey, rownames(priceKey))
colnames(buildData) <- c("1",'2')
colnames(priceKey) <- c("2", "Neighborhood")
fullFrame_intermediate <- join(fullFrame,priceKey, by="Neighborhood")
fullFrame$P_sf <- fullFrame_intermediate[,ncol(fullFrame_intermediate)]
fullFrame$P_sf <- fullFrame$P_sf*as.integer(fullFrame$OverallQual)


Price_SF_land <- as.numeric((targetValue[1:length(targetValue)]
                             ))/(fullFrame$LotArea[1:length(targetValue)])
buildData <- data.frame(fullFrame$Neighborhood[1:length(targetValue)], Price_SF_land)
priceKey <- as.data.frame(tapply(buildData$Price_SF_land,
                                 buildData$fullFrame.Neighborhood.1.length.targetValue.., median))
priceKey <- cbind(priceKey, rownames(priceKey))
colnames(buildData) <- c("1",'2')
colnames(priceKey) <- c("2", "Neighborhood")
fullFrame_intermediate <- join(fullFrame,priceKey, by="Neighborhood")
fullFrame$Price_SF_land <- fullFrame_intermediate[,ncol(fullFrame_intermediate)]

fullFrame$Price_SF_land_ <- fullFrame$Price_SF_land^as.integer(fullFrame$OverallQual)

fullFrame$Is_pool <- as.factor(ifelse(fullFrame$PoolArea > 0, 1, 0))

fullFrame$Is_2stories <- as.factor(ifelse(fullFrame$X2ndFlrSF > 0, 1, 0))

fullFrame$Is_garage <- as.factor(ifelse(fullFrame$GarageArea > 0, 1, 0))

fullFrame$Is_bsmt <- as.factor(ifelse(fullFrame$TotalBsmtSF > 0, 1, 0))
ncol(fullFrame)
```

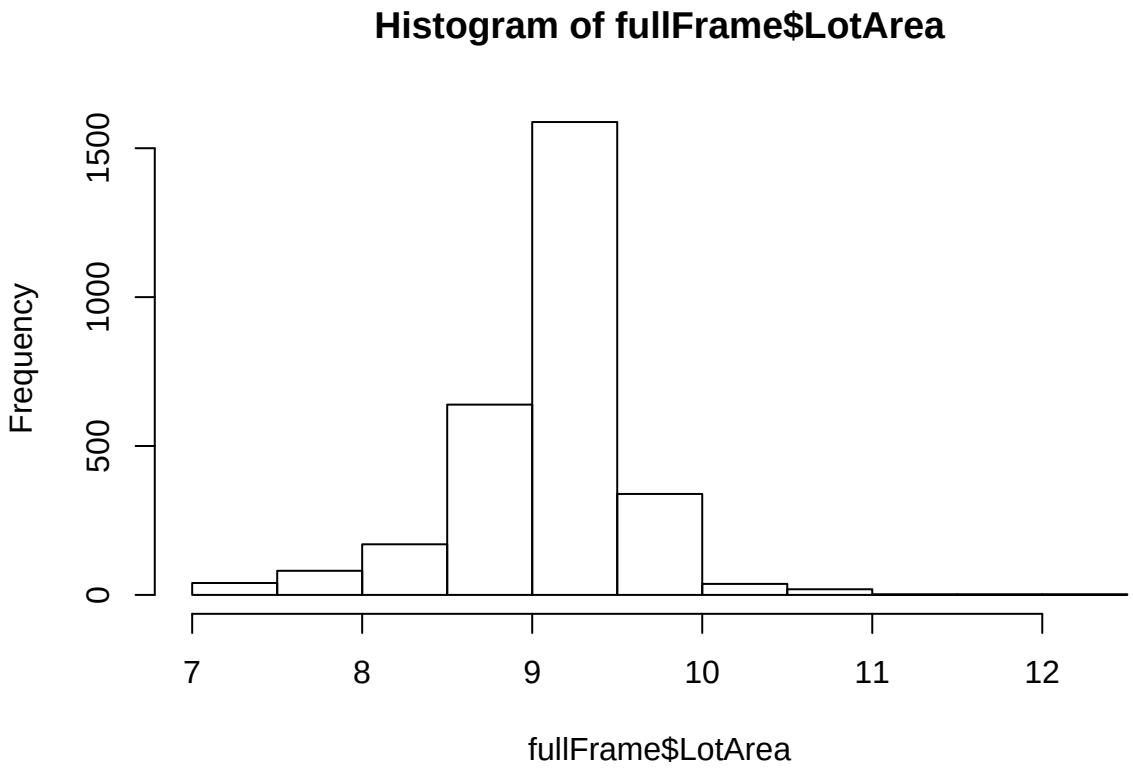
```
## [1] 92
```

There are some highly skewed predictors and taking the log1p of these predictors improved accuracy. A histogram of Lot Area before and after the transformation highlights how well the skewness is improved. The log1p is used because there are so many instances of 0.

```
hist(fullFrame$LotArea)
```

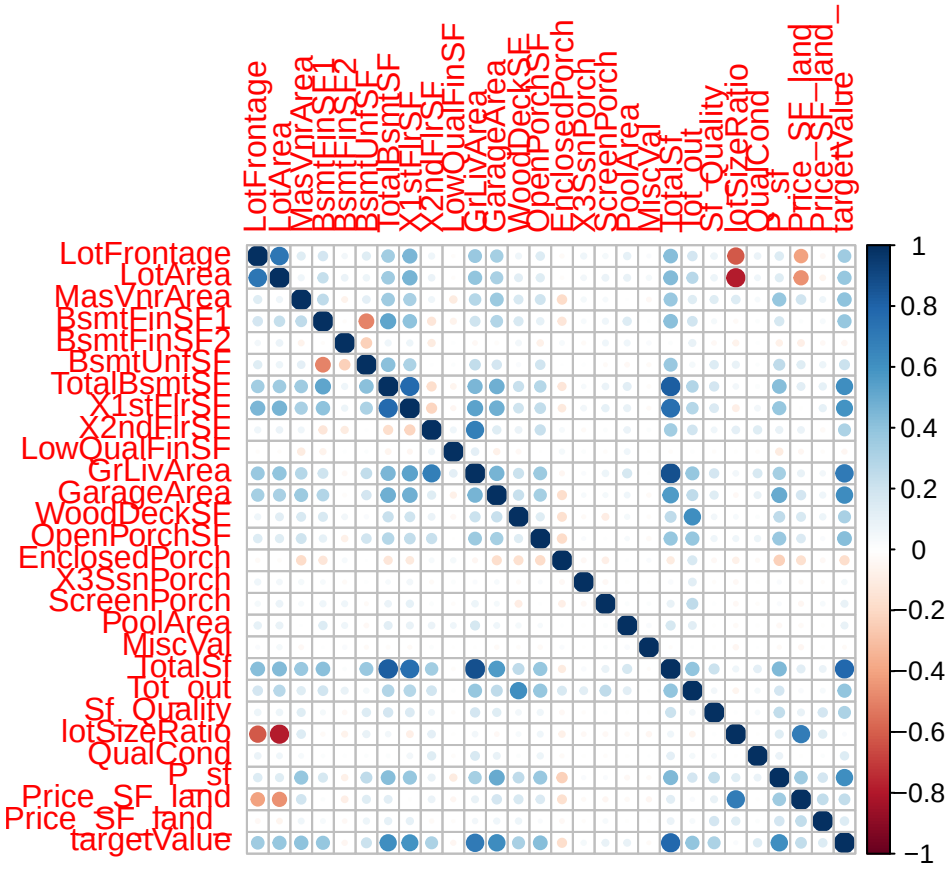



```
fullFrame$LowQualFinSF <- log1p(fullFrame$LowQualFinSF)
fullFrame$LotArea <- log1p(fullFrame$LotArea)
fullFrame$BsmtFinSF2 <- log1p(fullFrame$BsmtFinSF2)
fullFrame$ScreenPorch <- log1p(fullFrame$ScreenPorch)
fullFrame$X3SsnPorch <- log1p(fullFrame$X3SsnPorch)
fullFrame$EnclosedPorch <- log1p(fullFrame$EnclosedPorch)
fullFrame$MasVnrArea <- log1p(fullFrame$MasVnrArea)
fullFrame$OpenPorchSF <- log1p(fullFrame$OpenPorchSF)
fullFrame$WoodDeckSF <- log1p(fullFrame$WoodDeckSF)
fullFrame$LotFrontage <- log1p(fullFrame$LotFrontage)
fullFrame$X1stFlrSF <- log1p(fullFrame$X1stFlrSF)

hist(fullFrame$LotArea)
```



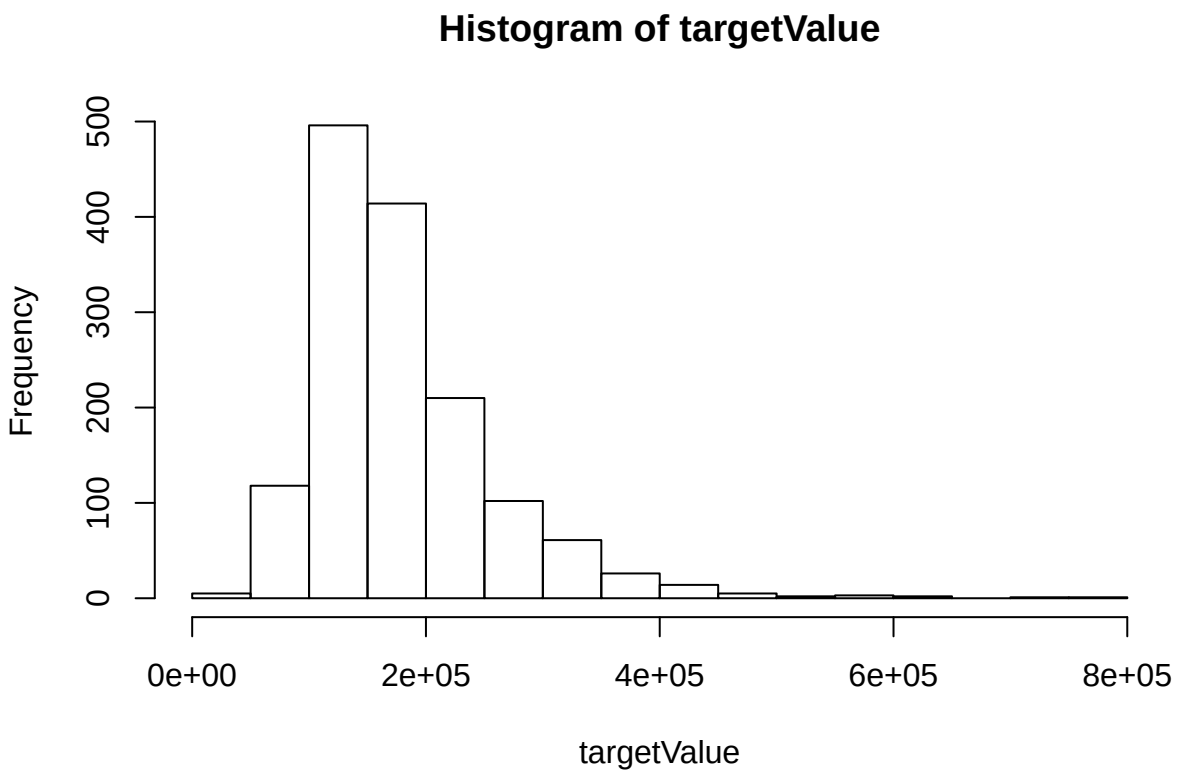
The correlation matrix highlights the fact that there is a high correlation between predictors. This means that models that handle multicollinearity well should be used in the final predictions.

```
#Resplit the data frame
ints <- c()
cats <- c()
for (i in 1:ncol(fullFrame)) {
  ifelse((is.numeric(fullFrame[,i])) ,ints[i] <- i ,cats[i] <- i)
}

ints <- sort(ints)
cats <- sort(cats)

intGraphs <- cbind(fullFrame[1:length(targetValue),ints], targetValue)

#correlations
cores <- cor(intGraphs)
corrplot(cores, method="circle")
```

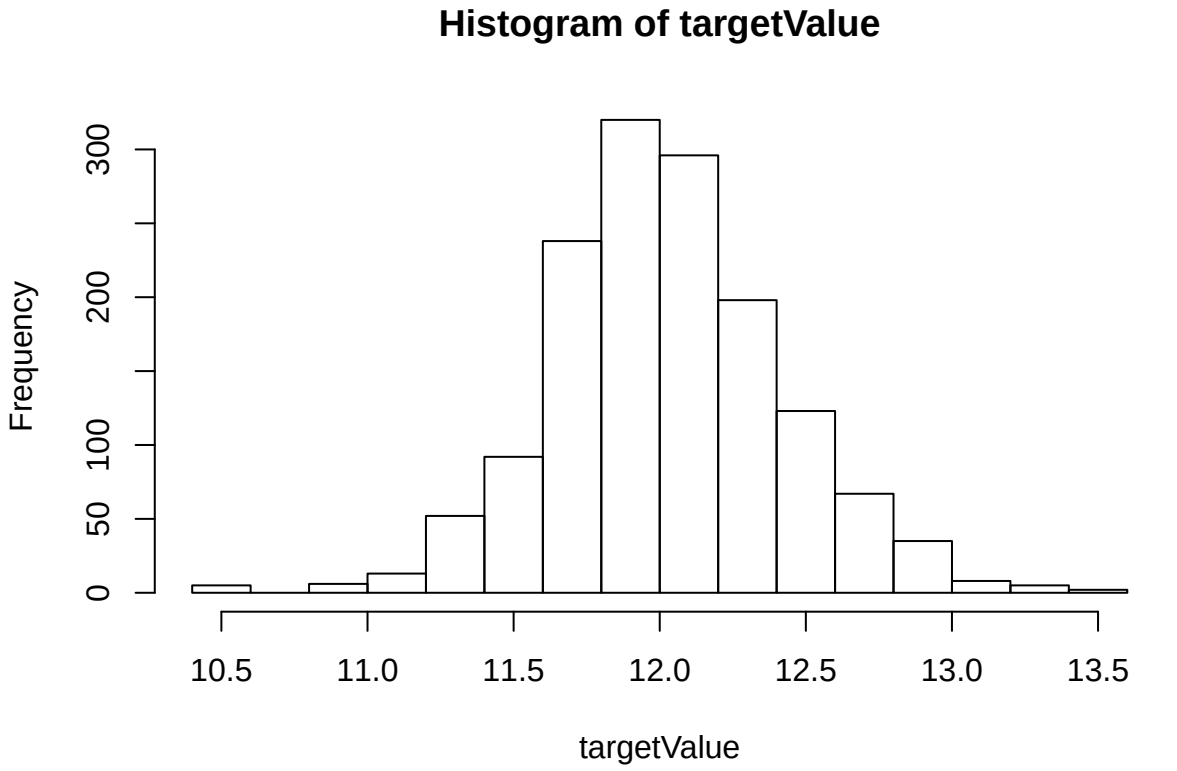


Sale price is much more normal after taking log1p.

```
hist(targetValue)
```



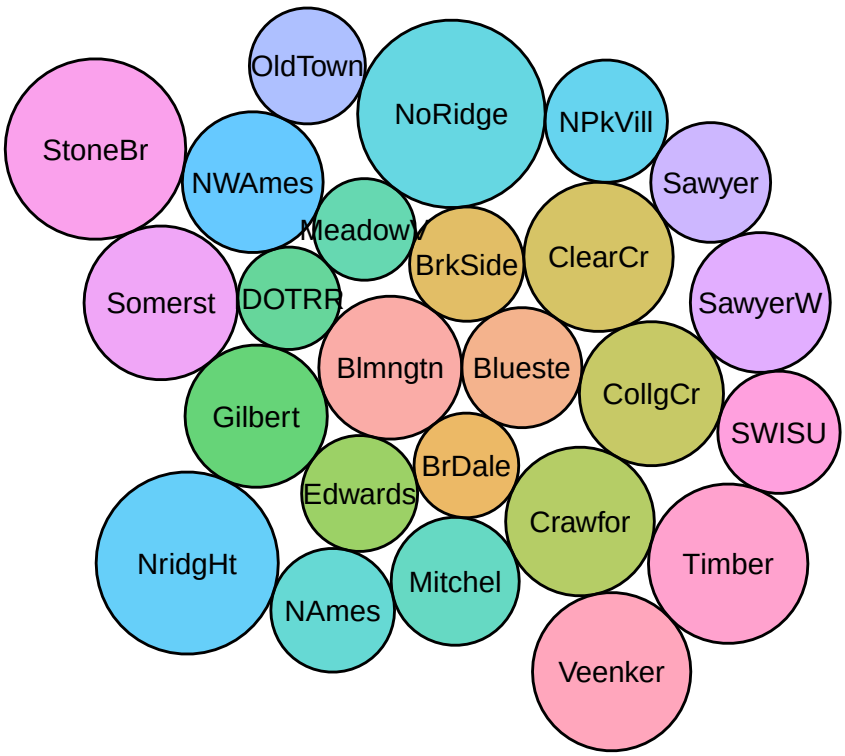

```
targetValue <- log1p(targetValue)
hist(targetValue)
```



At this point the pre-processing is completed. All missing values are filled, all transformations are made and all new features are engineered. These bubble plots highlight how the data changes when looking at sale price.

It is very easy to see which neighborhoods are the highest value and which are the lowest. The neighborhood visualization in particular shows that while there are large differences between some neighborhoods, like NoRidge and Edwards. Many neighborhoods are very similar like Blueste and BrkSide.

```
#https://www.r-graph-gallery.com/305-basic-circle-packing-with-one-level.html
Alldata <- data.frame(group = fullFrame$Neighborhood[1:1460],value = expm1(targetValue))
groupings <- data.frame(tapply(Alldata$value,Alldata$group,  mean))
data <- cbind(groupings, rownames(groupings))
colnames(data) <- c('value', 'group')
rownames(data) <- NULL
packing <- circleProgressiveLayout(data$value, sizetype='area')
data <- cbind(data, packing)
dat.gg <- circleLayoutVertices(packing, npoints=50)
# Make the plot
ggplot() +
  # Make the bubbles
  geom_polygon(data = dat.gg, aes(x, y, group = id, fill=as.factor(id)), colour = "black", alpha = 0.6)
  # Add text in the center of each bubble + control its size
  geom_text(data = data, aes(x, y,  label = group)) +
  scale_size_continuous(range = c(1,4)) +
  # General theme:
  theme_void() +
  theme(legend.position="none") +
  ggtitle('Mean price by neighborhood') +
  coord_equal()
```

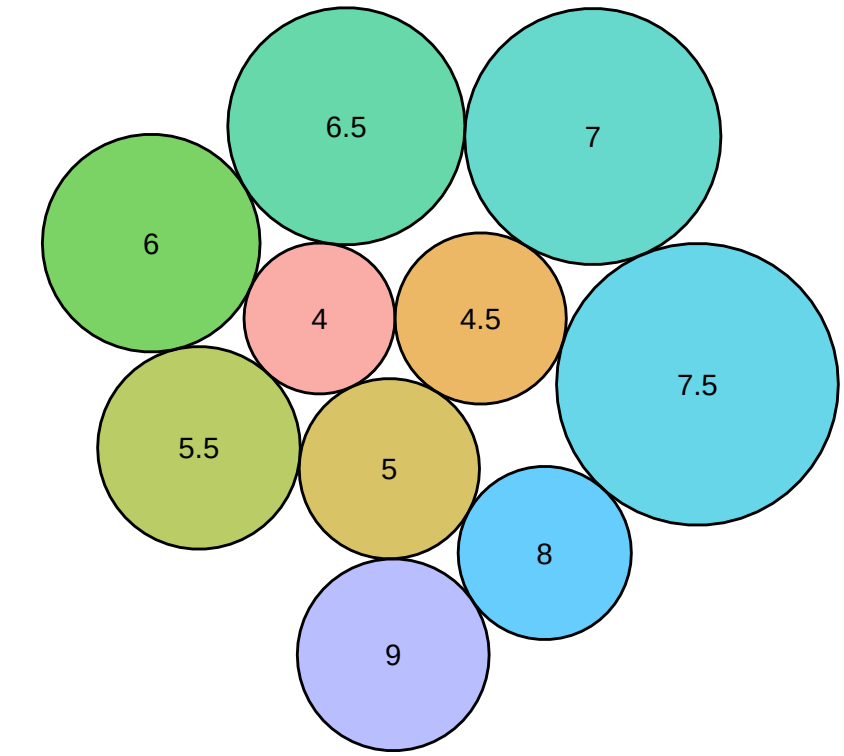


```
Alldata <- data.frame(group = fullFrame$T_bathrooms[1:1460],value = expm1(targetValue))
groupings <- data.frame(tapply(Alldata$value,Alldata$group,  mean))
data <- cbind(groupings, rownames(groupings))
colnames(data) <- c('value', 'group')
rownames(data) <- NULL
packing <- circleProgressiveLayout(data$value, sizetype='area')
```

```
## Warning in circleProgressiveLayout(data$value, sizetype = "area"): missing and/
## or non-positive sizes will be ignored
```

```
data <- cbind(data, packing)
dat.gg <- circleLayoutVertices(packing, npoints=50)
# Make the plot
ggplot() +
  # Make the bubbles
  geom_polygon(data = dat.gg, aes(x, y, group = id, fill=as.factor(id)), colour = "black", alpha = 0.6)
  # Add text in the center of each bubble + control its size
  geom_text(data = data, aes(x, y, label = group)) +
  scale_size_continuous(range = c(1,4)) +
  # General theme:
  theme_void() +
  theme(legend.position="none") +
  ggtitle('Mean price by number of bathrooms') +
  coord_equal()
```

```
## Warning: Removed 3 rows containing missing values (geom_text).
```



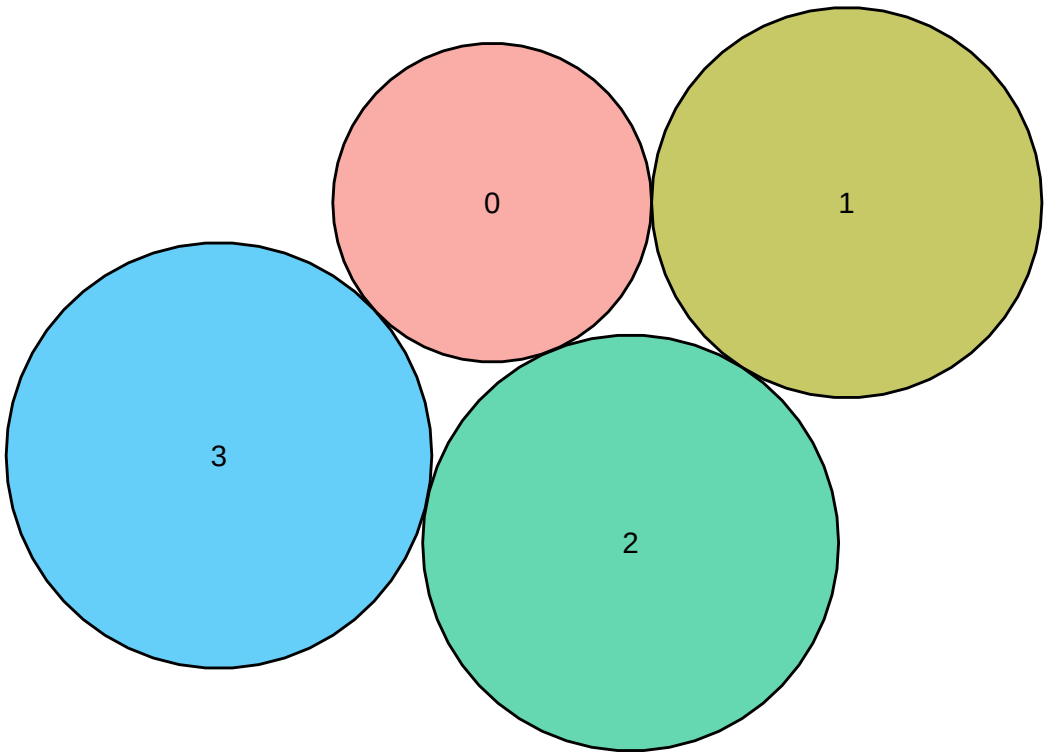
```
Alldata <- data.frame(group = fullFrame$Fireplaces[1:1460],value = expm1(targetValue))
groupings <- data.frame(tapply(Alldata$value,Alldata$group,  mean))
data <- cbind(groupings, rownames(groupings))
colnames(data) <- c('value', 'group')
rownames(data) <- NULL
packing <- circleProgressiveLayout(data$value, sizetype='area')
```

```
## Warning in circleProgressiveLayout(data$value, sizetype = "area"): missing and/
## or non-positive sizes will be ignored
```

```
data <- cbind(data, packing)
dat.gg <- circleLayoutVertices(packing, npoints=50)
# Make the plot
ggplot() +
  # Make the bubbles
  geom_polygon(data = dat.gg, aes(x, y, group = id, fill=as.factor(id)), colour = "black", alpha = 0.6)
  # Add text in the center of each bubble + control its size
  geom_text(data = data, aes(x, y, label = group)) +
  scale_size_continuous(range = c(1,4)) +
  # General theme:
  theme_void() +
  theme(legend.position="none") +
  ggtitle('Mean price by number of fireplaces') +
  coord_equal()
```

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

Mean price by number of fireplaces



```
Alldata <- data.frame(group = fullFrame$YearBuilt[1:1460],value = expm1(targetValue))
groupings <- data.frame(tapply(Alldata$value,Alldata$group,  mean))
data <- cbind(groupings, rownames(groupings))
colnames(data) <- c('value', 'group')
rownames(data) <- NULL
packing <- circleProgressiveLayout(data$value, sizetype='area')
data <- cbind(data, packing)
dat.gg <- circleLayoutVertices(packing, npoints=50)
# Make the plot
ggplot() +
  # Make the bubbles
  geom_polygon(data = dat.gg, aes(x, y, group = id, fill=as.factor(id)), colour = "black", alpha = 0.6)
  # Add text in the center of each bubble + control its size
  geom_text(data = data, aes(x, y, label = group)) +
  scale_size_continuous(range = c(1,4)) +
  # General theme:
  theme_void() +
  theme(legend.position="none") +
  ggtitle('Mean price by year built') +
  coord_equal()
```

Models

Association Rule Mining

Association Rule Mining is able to find rules within the data. These rules highlight various patterns. A new data frame is created to mine for association rules. Columns need to be removed to avoid redundant

rules. For example, GarageCars needs to be removed because without doing so a frequent and strong rule is GarageCars being 0 means GarageArea will be 0. Since this is trivial certain predictors are removed to avoid this. Sale price is discretized into three buckets. Low, medium, and high value homes. The data frame is then converted into transactions. First the algorithm mines for general rules and the top 10 rules for lift, confidence, and support are shown. Association Rule Mining is then used to find patterns that dictate which one of those buckets will an observation fall into. Sorting those rules by support gave the most interesting rules and those are the ones that are visualized.

```
armTransition <- fullFrame[,-c(89,90,91,92
                               ,which(colnames(fullFrame)=="GarageFinish")
                               ,which(colnames(fullFrame)=="GarageCars")
                               ,which(colnames(fullFrame)=="GarageQual")
                               ,which(colnames(fullFrame)=="GarageCond")
                               ,which(colnames(fullFrame)=="GarageYrBlt")
                                ,which(colnames(fullFrame)=="MSSubClass")
                                ,which(colnames(fullFrame)=="SaleType")
)]

armTransition <- armTransition[1:1460,]
armTransition$targs <- targetValue
armTransition$targs <- discretize(armTransition$targs, breaks = 3, labels = c('low', 'medium', 'high'))
armFrame <- suppressWarnings(as(armTransition, "transactions"))


rules_1 <- apriori(armFrame,
                   parameter = list(support=.25, confidence=.99, minlen = 3),
                   control=list(verbose = FALSE))

high_support <- sort(rules_1, by="support", decreasing=TRUE)
inspect(head(high_support,10))
```

```
##      lhs                          rhs                      support confidence coverage lift count
## [1]  {BsmtFinSF2=[0,7.3],
##       MiscVal=[0,1.55e+04]}   => {PoolArea=[0,738]}            1          1        1    1  1460
## [2]  {PoolArea=[0,738],
##       MiscVal=[0,1.55e+04]}   => {BsmtFinSF2=[0,7.3]}          1          1        1    1  1460
## [3]  {BsmtFinSF2=[0,7.3],
##       PoolArea=[0,738]}       => {MiscVal=[0,1.55e+04]}        1          1        1    1  1460
## [4]  {BsmtFinSF2=[0,7.3],
##       MiscVal=[0,1.55e+04]}   => {ScreenPorch=[0,6.18]}        1          1        1    1  1460
## [5]  {ScreenPorch=[0,6.18],
##       MiscVal=[0,1.55e+04]}   => {BsmtFinSF2=[0,7.3]}          1          1        1    1  1460
## [6]  {BsmtFinSF2=[0,7.3],
##       ScreenPorch=[0,6.18]}   => {MiscVal=[0,1.55e+04]}        1          1        1    1  1460
## [7]  {BsmtFinSF2=[0,7.3],
##       MiscVal=[0,1.55e+04]}   => {EnclosedPorch=[0,6.32]}      1          1        1    1  1460
## [8]  {EnclosedPorch=[0,6.32],
##       MiscVal=[0,1.55e+04]}   => {BsmtFinSF2=[0,7.3]}          1          1        1    1  1460
## [9]  {BsmtFinSF2=[0,7.3],
##       EnclosedPorch=[0,6.32]} => {MiscVal=[0,1.55e+04]}        1          1        1    1  1460
## [10] {BsmtFinSF2=[0,7.3],
##       MiscVal=[0,1.55e+04]}   => {X3SsnPorch=[0,6.23]}         1          1        1    1  1460
```

```
high_lift <- sort(rules_1, by="lift", decreasing=TRUE)
inspect(head(high_lift,10))
```

```
##      lhs                    rhs                    support confidence  coverage     lift count
## [1]  {BsmtFinType1=Unf,
##       BsmtFullBath=0}    => {BsmtFinSF1=[0,24)} 0.2842466          1 0.2842466 3.029046   415
## [2]  {BldgType=1Fam,
##       BsmtFinType1=Unf} => {BsmtFinSF1=[0,24)} 0.2554795          1 0.2554795 3.029046   373
## [3]  {BsmtFinType1=Unf,
##       BsmtFinType2=Unf} => {BsmtFinSF1=[0,24)} 0.2945205          1 0.2945205 3.029046   430
## [4]  {ExterCond=TA,
##       BsmtFinType1=Unf} => {BsmtFinSF1=[0,24)} 0.2616438          1 0.2616438 3.029046   382
## [5]  {LandContour=Lvl,
##       BsmtFinType1=Unf} => {BsmtFinSF1=[0,24)} 0.2664384          1 0.2664384 3.029046   389
## [6]  {BsmtCond=TA,
##       BsmtFinType1=Unf} => {BsmtFinSF1=[0,24)} 0.2657534          1 0.2657534 3.029046   388
## [7]  {BsmtFinType1=Unf,
##       Electrical=SBrkr} => {BsmtFinSF1=[0,24)} 0.2602740          1 0.2602740 3.029046   380
## [8]  {BsmtFinType1=Unf,
##       Functional=Typ}    => {BsmtFinSF1=[0,24)} 0.2760274          1 0.2760274 3.029046   403
## [9]  {BsmtFinType1=Unf,
##       CentralAir=Y}      => {BsmtFinSF1=[0,24)} 0.2636986          1 0.2636986 3.029046   385
## [10] {Alley=none,
##       BsmtFinType1=Unf} => {BsmtFinSF1=[0,24)} 0.2602740          1 0.2602740 3.029046   380
```

```
high_conf <- sort(rules_1, by="confidence", decreasing=TRUE)
inspect(head(high_conf,10))
```

```
##      lhs                                rhs                                 support confidence  cover
## [1]  {OverallQual=6,
##       Heating=GasA}                  => {Sf_Quality=[3.02e+20,7.25e+26)} 0.2506849          1 0.250
## [2]  {OverallQual=6,
##       RoofMatl=CompShg}              => {Sf_Quality=[3.02e+20,7.25e+26)} 0.2520548          1 0.252
## [3]  {Condition2=Norm,
##       OverallQual=6}                 => {Sf_Quality=[3.02e+20,7.25e+26)} 0.2554795          1 0.255
## [4]  {OverallQual=6,
##       PoolQC=none}                   => {Sf_Quality=[3.02e+20,7.25e+26)} 0.2541096          1 0.254
## [5]  {Street=Pave,
##       OverallQual=6}                 => {Sf_Quality=[3.02e+20,7.25e+26)} 0.2554795          1 0.255
## [6]  {Utilities=AllPub,
##       OverallQual=6}                 => {Sf_Quality=[3.02e+20,7.25e+26)} 0.2554795          1 0.255
## [7]  {OverallQual=6,
##       Sf_Quality=[3.02e+20,7.25e+26)} => {MiscVal=[0,1.55e+04]}          0.2561644          1 0.256
## [8]  {OverallQual=6,
##       MiscVal=[0,1.55e+04]}          => {Sf_Quality=[3.02e+20,7.25e+26)} 0.2561644          1 0.256
## [9]  {OverallQual=6,
##       Sf_Quality=[3.02e+20,7.25e+26)} => {BsmtFinSF2=[0,7.3]}            0.2561644          1 0.256
## [10] {OverallQual=6,
##       BsmtFinSF2=[0,7.3]}            => {Sf_Quality=[3.02e+20,7.25e+26)} 0.2561644          1 0.256
```

```
rulesTargs <- apriori(armFrame, parameter = list(support=.035, confidence = 1),
                      control=list(verbose = FALSE),
```

```
                  appearance = list(rhs = c('targs=low','targs=medium','targs=high')))



rules_lift <- sort(rulesTargs, by="lift", decreasing=TRUE)
inspect(head(rules_lift,10))
```

```
##      lhs                                   rhs            support confidence   coverage     lift count
## [1]  {GarageType=none,
##       TotalSf=[334,2.16e+03),
##       P_sf=[99.6,723)}                  => {targs=low} 0.03630137          1 0.03630137 2.997947    53
## [2]  {MasVnrType=None,
##       GarageType=none,
##       TotalSf=[334,2.16e+03)}           => {targs=low} 0.03698630          1 0.03698630 2.997947    54
## [3]  {MasVnrArea=[0,4.55),
##       GarageType=none,
##       TotalSf=[334,2.16e+03)}           => {targs=low} 0.03698630          1 0.03698630 2.997947    54
## [4]  {CentralAir=N,
##       GarageArea=[0,400),
##       P_sf=[99.6,723)}                  => {targs=low} 0.03972603          1 0.03972603 2.997947    58
## [5]  {CentralAir=N,
##       Sf_Quality=[334,3.02e+20),
##       Price_SF_land_=[13.1,3.82e+07)} => {targs=low} 0.03972603          1 0.03972603 2.997947    58
## [6]  {CentralAir=N,
##       WoodDeckSF=[0,4.95),
##       Sf_Quality=[334,3.02e+20)}        => {targs=low} 0.03972603          1 0.03972603 2.997947    58
## [7]  {LotFrontage=[3.09,4.13),
##       OverallQual=4,
##       TotalSf=[334,2.16e+03)}           => {targs=low} 0.03767123          1 0.03767123 2.997947    55
## [8]  {LotFrontage=[3.09,4.13),
##       OverallQual=4,
##       WoodDeckSF=[0,4.95)}              => {targs=low} 0.04041096          1 0.04041096 2.997947    59
## [9]  {LotConfig=Inside,
##       OverallQual=4,
##       TotalSf=[334,2.16e+03)}           => {targs=low} 0.05000000          1 0.05000000 2.997947    73
## [10] {OverallQual=4,
##       TotalBsmtSF=[0,861),
##       FullBath=1}                       => {targs=low} 0.03698630          1 0.03698630 2.997947    54
```

```
rules_supp <- sort(rulesTargs, by="support", decreasing=TRUE)
inspect(head(rules_supp,10))
```

```
##      lhs                                       rhs             support confidence   coverage     lift
## [1]  {BsmtFullBath=1,
##       TotalSf=[2.82e+03,1.18e+04],
##       Sf_Quality=[7.25e+26,1.11e+37],
##       Price_SF_land_=[1.46e+10,7.64e+15]} => {targs=high} 0.10342466          1 0.10342466 2.943548
## [2]  {BsmtFullBath=1,
##       TotalSf=[2.82e+03,1.18e+04],
##       P_sf=[934,1.58e+03],
##       Price_SF_land_=[1.46e+10,7.64e+15]} => {targs=high} 0.10205479          1 0.10205479 2.943548
## [3]  {BsmtFullBath=1,
```

```
##       TotalSf=[2.82e+03,1.18e+04],
##       QualCond=[3.91e+05,3.87e+08],
##       Price_SF_land_=[1.46e+10,7.64e+15]} => {targs=high} 0.10205479          1 0.10205479 2.943548
## [4]  {Fireplaces=1,
##       OpenPorchSF=[3.93,6.31],
##       TotalSf=[2.82e+03,1.18e+04],
##       QualCond=[3.91e+05,3.87e+08]}       => {targs=high} 0.10205479          1 0.10205479 2.943548
## [5]  {BsmtFullBath=1,
##       Functional=Typ,
##       TotalSf=[2.82e+03,1.18e+04],
##       Price_SF_land_=[1.46e+10,7.64e+15]} => {targs=high} 0.10136986          1 0.10136986 2.943548
## [6]  {ExterCond=TA,
##       HalfBath=1,
##       TotalSf=[2.82e+03,1.18e+04],
##       Sf_Quality=[7.25e+26,1.11e+37]}     => {targs=high} 0.10136986          1 0.10136986 2.943548
## [7]  {Foundation=PConc,
##       BsmtFullBath=1,
##       TotalSf=[2.82e+03,1.18e+04],
##       P_sf=[934,1.58e+03]}                => {targs=high} 0.10136986          1 0.10136986 2.943548
## [8]  {Fireplaces=1,
##       OpenPorchSF=[3.93,6.31],
##       TotalSf=[2.82e+03,1.18e+04],
##       Sf_Quality=[7.25e+26,1.11e+37]}     => {targs=high} 0.09931507          1 0.09931507 2.943548
## [9]  {Foundation=PConc,
##       BsmtFullBath=1,
##       TotalSf=[2.82e+03,1.18e+04],
##       Price_SF_land_=[1.46e+10,7.64e+15]} => {targs=high} 0.09794521          1 0.09794521 2.943548
## [10] {LotFrontage=[4.38,5.75],
##       HeatingQC=Ex,
##       GrLivArea=[1.66e+03,5.64e+03],
##       Sf_Quality=[7.25e+26,1.11e+37]}     => {targs=high} 0.09794521          1 0.09794521 2.943548
```

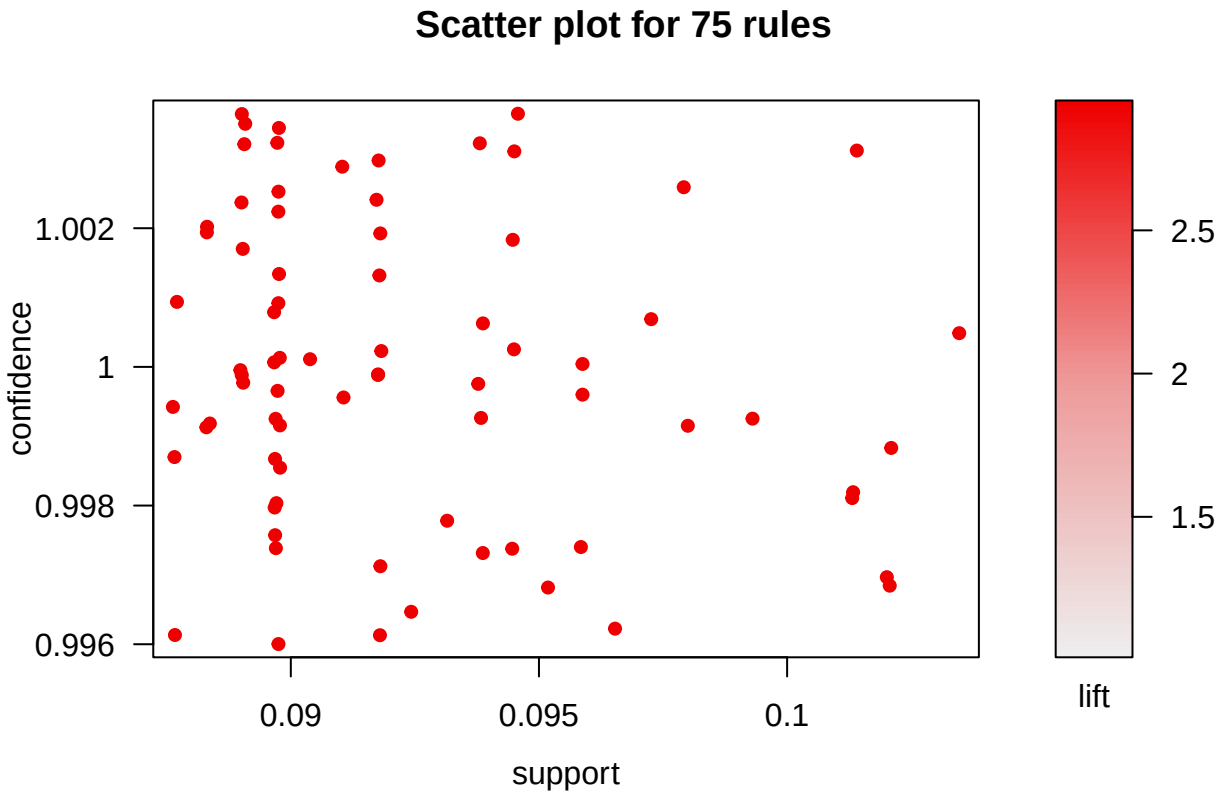
```
rules_conf <- sort(rulesTargs, by="confidence", decreasing=TRUE)
inspect(head(rules_conf,10))
```

```
##      lhs                                    rhs             support confidence   coverage     lift c
## [1]  {Neighborhood=NridgHt,
##       TotalSf=[2.82e+03,1.18e+04]}        => {targs=high} 0.04246575          1 0.04246575 2.943548
## [2]  {Neighborhood=NridgHt,
##       GrLivArea=[1.66e+03,5.64e+03]}      => {targs=high} 0.03561644          1 0.03561644 2.943548
## [3]  {Neighborhood=NridgHt,
##       GarageArea=[540,1.42e+03]}          => {targs=high} 0.04383562          1 0.04383562 2.943548
## [4]  {Neighborhood=NridgHt,
##       BldgType=1Fam}                      => {targs=high} 0.03767123          1 0.03767123 2.943548
## [5]  {KitchenQual=Ex,
##       Price_SF_land_=[1.46e+10,7.64e+15]} => {targs=high} 0.04726027          1 0.04726027 2.943548
## [6]  {TotalBsmtSF=[1.16e+03,6.11e+03],
##       T_bathrooms=6.5}                    => {targs=high} 0.04109589          1 0.04109589 2.943548
## [7]  {Neighborhood=NridgHt,
##       FireplaceQu=Gd,
##       GarageArea=[540,1.42e+03]}          => {targs=high} 0.03630137          1 0.03630137 2.943548
## [8]  {Neighborhood=NridgHt,
##       MasVnrArea=[4.55,7.38],
##       TotalSf=[2.82e+03,1.18e+04]}        => {targs=high} 0.03972603          1 0.03972603 2.943548
```

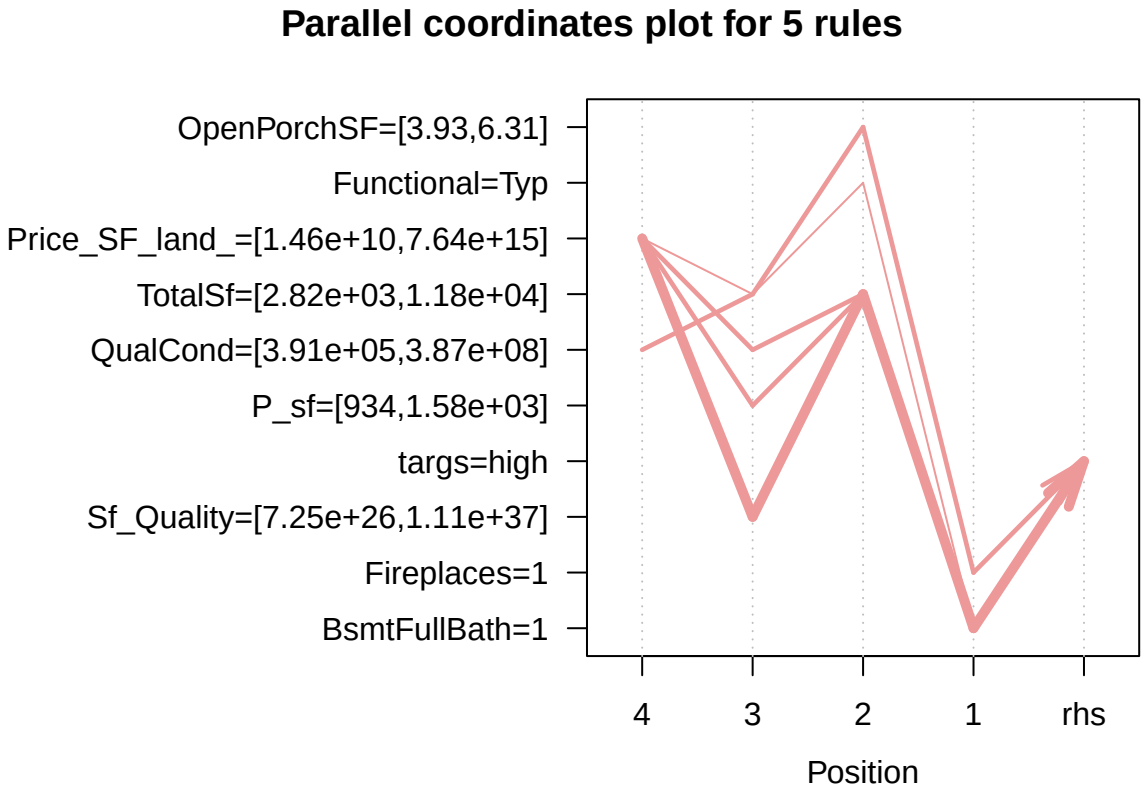
```
## [9]  {Neighborhood=NridgHt,
##       MasVnrArea=[4.55,7.38],
##       GarageArea=[540,1.42e+03]}          => {targs=high} 0.04109589          1 0.04109589 2.943548
## [10] {Neighborhood=NridgHt,
##       TotalSf=[2.82e+03,1.18e+04],
##       Price_SF_land_=[1.46e+10,7.64e+15]} => {targs=high} 0.03698630          1 0.03698630 2.943548
```

```
plot(rules_supp[1:75])
```

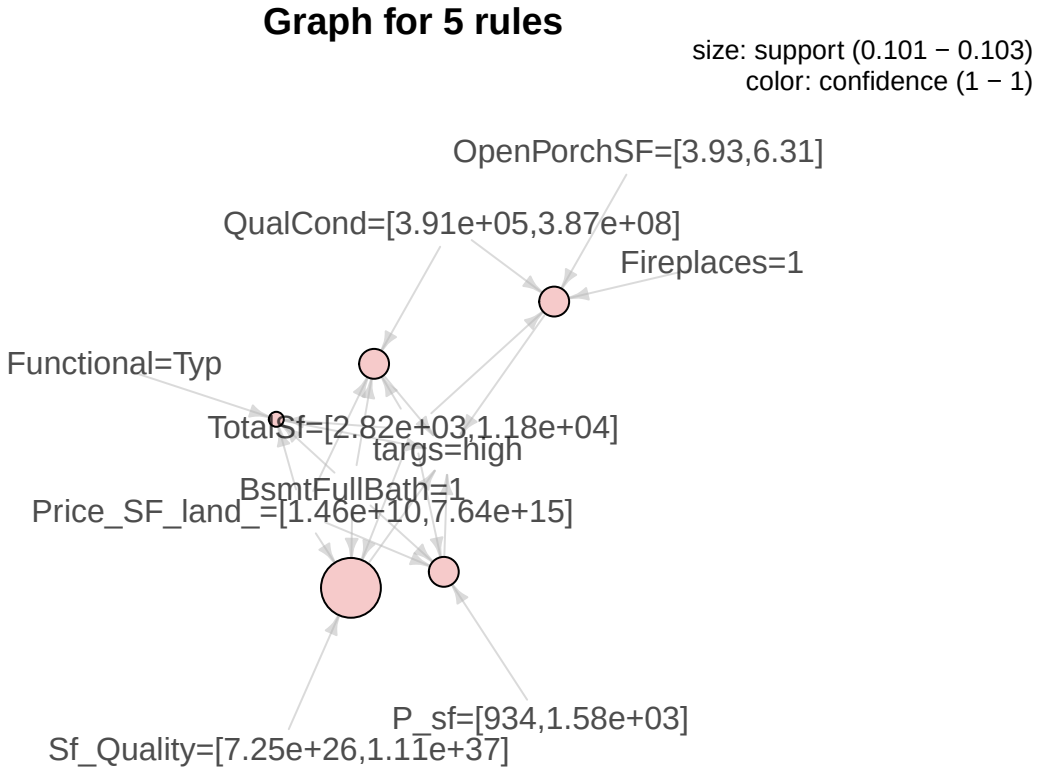
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



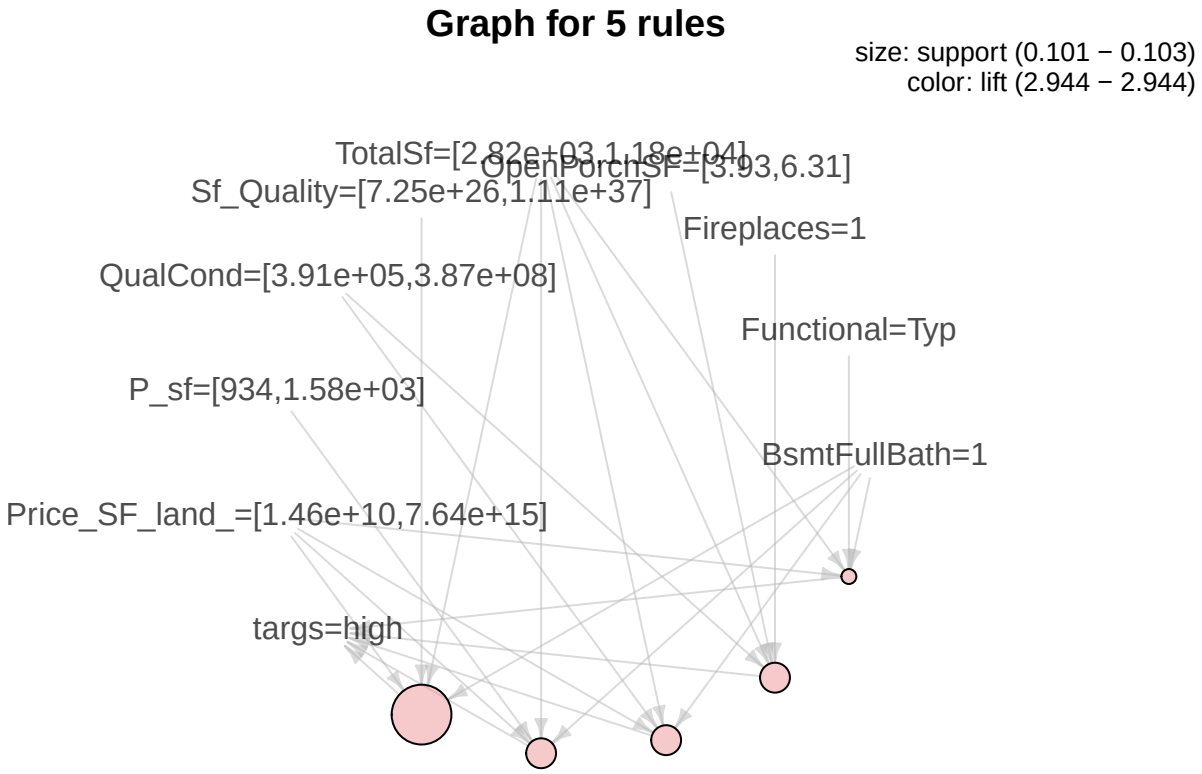

```
plot(rules_supp[1:5], method="paracoord", control=list(reorder=TRUE))
```




```
plot(rules_supp[1:5],method="graph",shading="confidence")
```



```
plot(rules_supp[1:5], method="graph", control=list(layout=igraph::in_circle()))
```

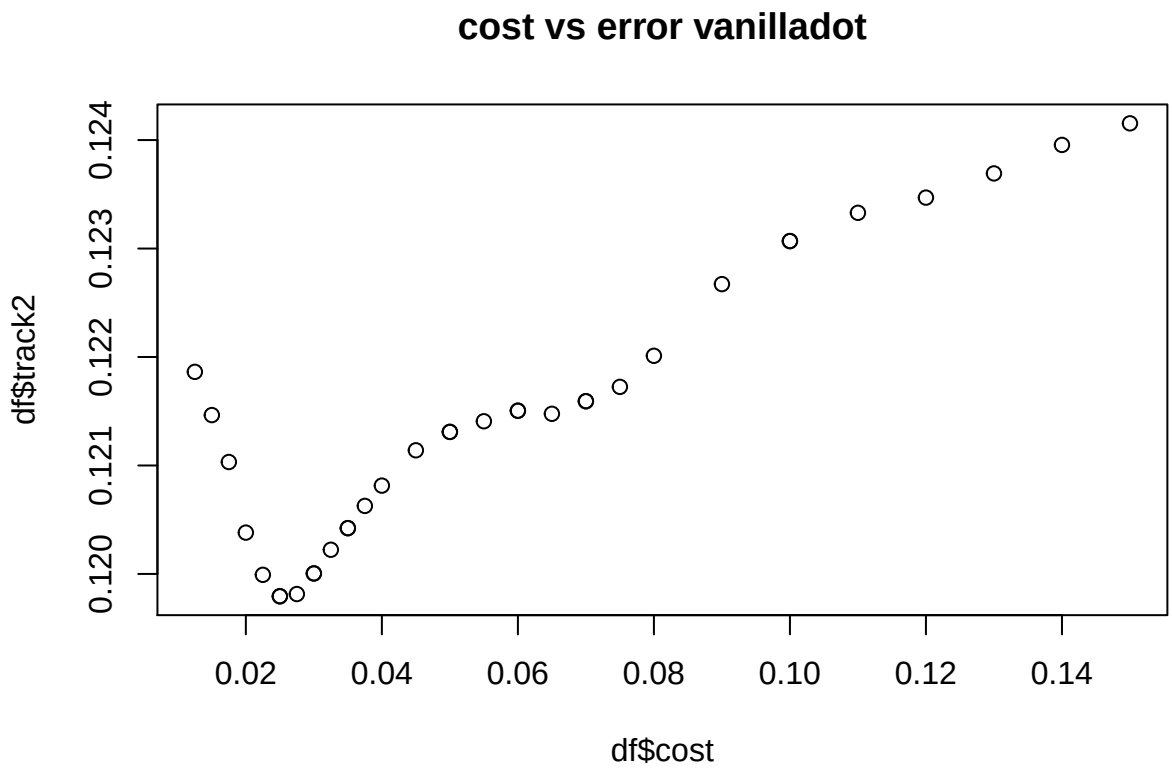


Support Vector Regression

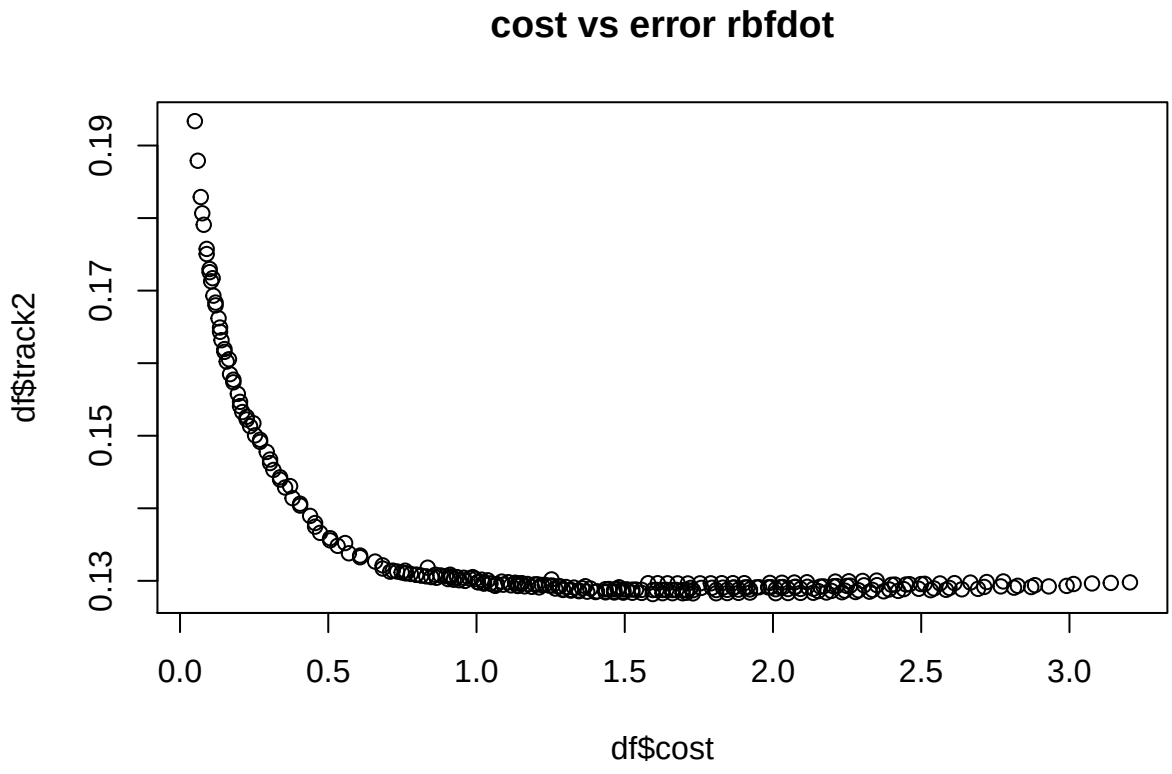
Support Vector Machine will attempt to find linear separability between observations. This section features a custom recursive cost tuning algorithm. It recursively passes in a vector of costs that are around the previous rounds best cost. It repeats this process until the same cost has been selected twice in a row or it has tried 25 costs. The function also creates a visualization of different costs and their respective error rate. A large benefit is being able to call this custom function and also pass in various kernels and regression types. Vanilladot, or the linear kernel, radial kernel, and laplacedot kernel are all tuned. The best is selected by using cross validation with optimal cost selected.

```
svmFrame <- cbind(fullFrame[1:length(targetValue),], targetValue)
svmTestFrame <- fullFrame[(length(targetValue)+1):nrow(fullFrame),]


indexes <- sample(1:length(targetValue), .75*length((1:length(targetValue))))

#taken from hadley wickam
quiet <- function(x) {
  sink(tempfile())
  on.exit(sink())
  invisible(force(x))
}

#function that tunes cost, takes a starting cost value, kernel, and type
FinetuneKSVMcost <- function(values, kern, svmType = 'eps-svr') {
  set.seed(5948)
  if (length(values) == 1) {
```

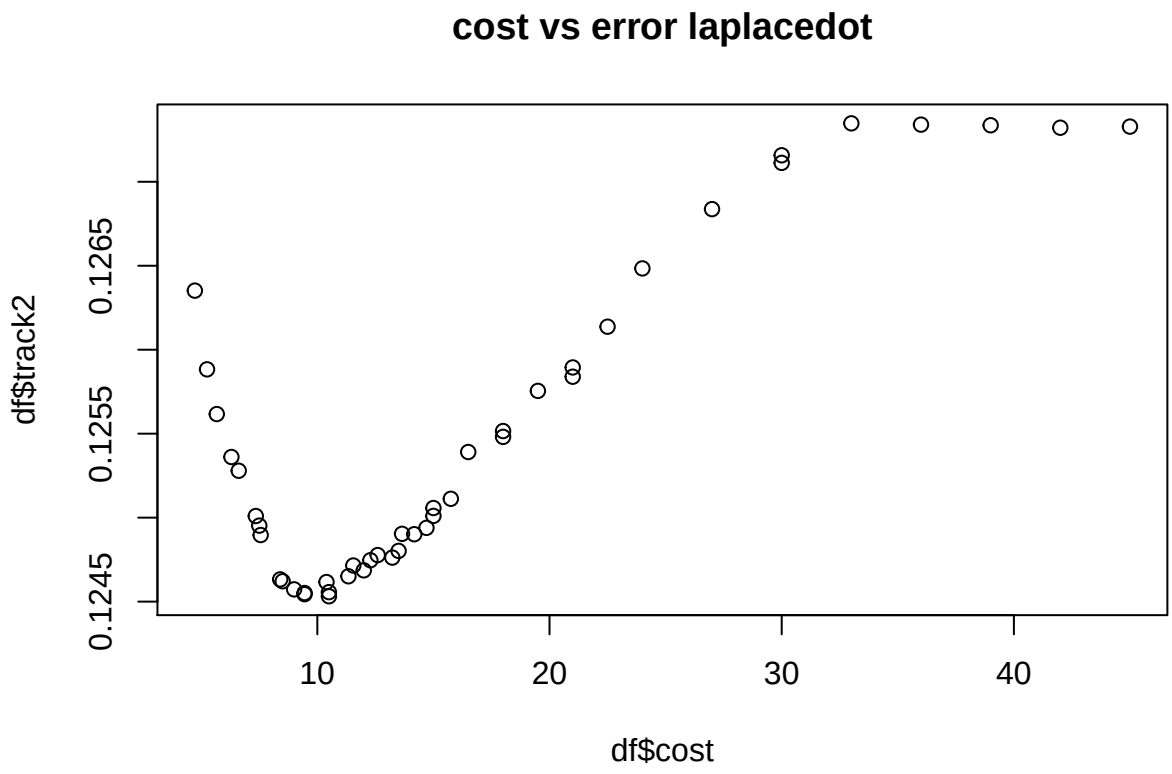
```
    track <- c()
    cost <- c()
    track2 <- c()
  }
  scoreVector <- c()
  for (i in values) {
    ksvmModel <- quiet(ksvm(x = targetValue ~ .,data = svmFrame[indexes,], kernel = kern, C = i, type =
    preds_ <- predict(ksvmModel,svmFrame[-indexes,])
    preds <- expm1(preds_)
    test3 <- expm1(targetValue[-indexes])
    scoreNow <- rmsle(preds ,test3)
    scoreVector <- c(scoreVector,scoreNow)
  }
  curr <- values[which.min(scoreVector)]
  newCosts <- seq(curr*.5, curr*1.5, curr/10)
  cost <<- c(cost, values)
  track2 <<- c(track2, scoreVector)
  track <<- c(track, values[which.min(scoreVector)])
  if (length(track)>2) {
    if (length(unique(track)) != length(track) || length(track)>25) {
      final <- c(values[which.min(scoreVector)], scoreVector[which.min(scoreVector)])
      df <- data.frame(cost,track2)
      df <- df[order(track2),]
      plot(df$cost, df$track2, main = paste('cost vs error', kern))
      return(final)
    }
  }
  FinetuneKSVMcost(newCosts, kern, svmType)
}

fineCost <- FinetuneKSVMcost(.1, 'vanilladot')
```



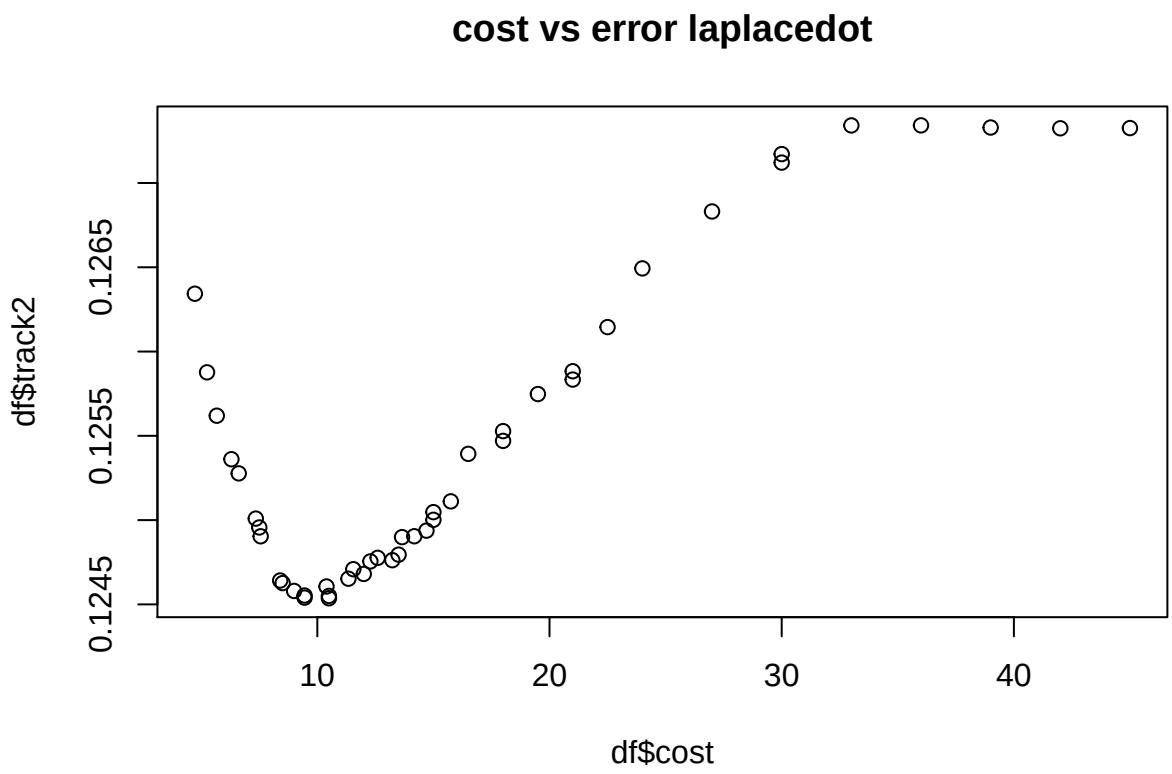

```
fineCost2 <- FinetuneKSVMcost(.1, 'rbfdot')
```



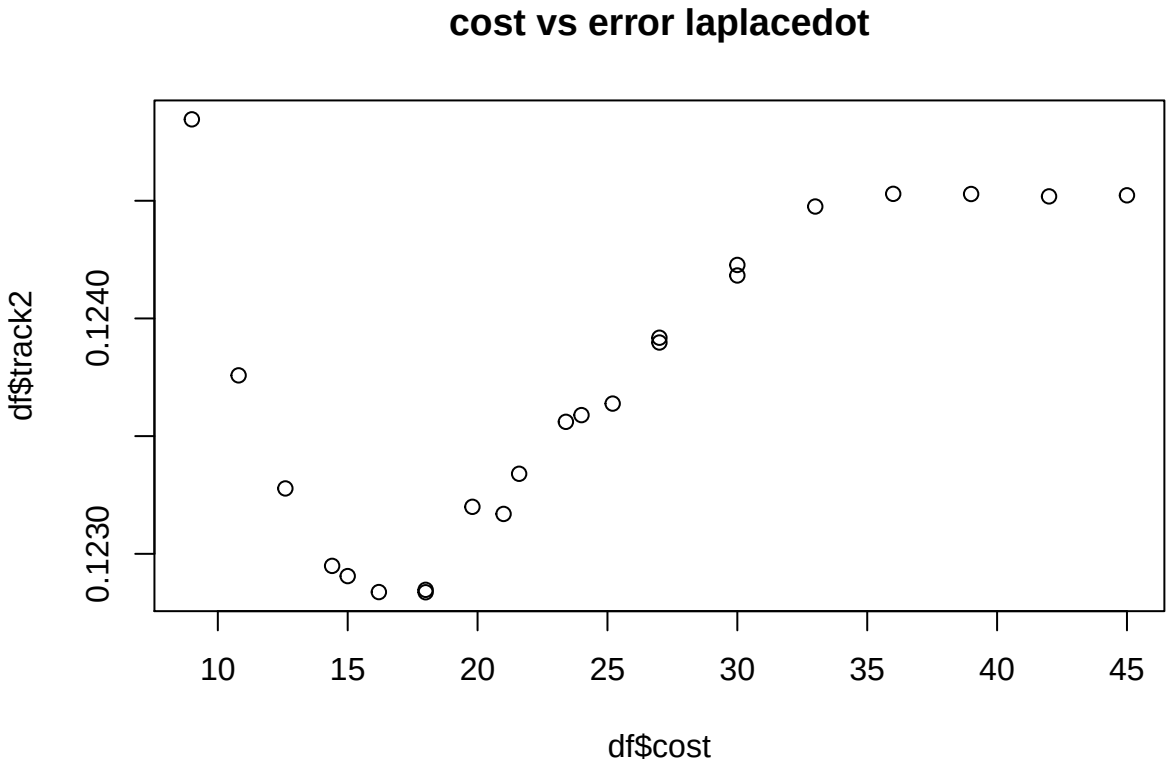

```
fineCost6 <- FinetuneKSVMcost(30, 'laplacedot')
```




```
fineCostype_1 <- FinetuneKSVMcost(30, 'laplacedot', 'eps-bsvr')
```




```
fineCostype_2 <- FinetuneKSVMcost(30, 'laplacedot', 'nu-svr')
```

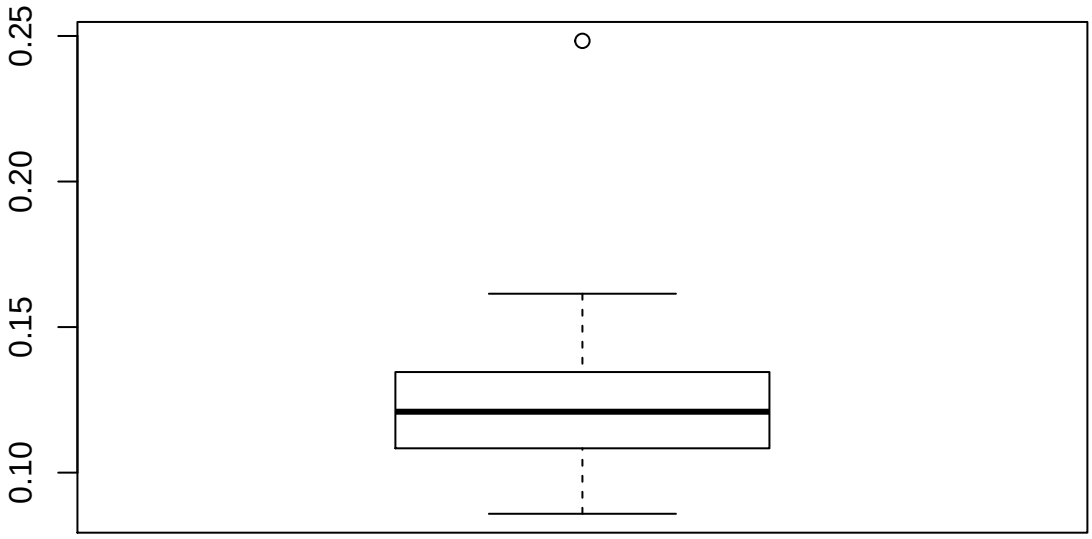


Linear kernel and laplacedot kernel both performed much better than radial did. Therefore, both of those kernels will be cross validated and then the best one will be used to make predictions. Each one of the 10 cross validated models will be used to make predictions. Then the mean prediction across all 10 folds is used to assess accuracy.

Although the linear kernel seemed to be the best from the previous analysis, laplacedot is more general and ends up over fitting less. It performs better with cross validation than the linear kernel did. This makes sense because the linear kernel would be impacted more by multicollinearity. The optimal cost from the tuning algorithm is passed into the cross validation scheme.

```
N <- nrow(svmFrame)

kfolds <- 10
set.seed(5948)

holdout <- split(sample(1:N), 1:kfolds)
Id <- 1:nrow(svmTestFrame)
SVMaggs <- data.frame(Id)
AllResultsSVM <- c()
for (k in 1:kfolds) {

  ksvmModel <- quiet(ksvm(x = targetValue ~ . , data = svmFrame[-holdout[[k]],]
                          , kernel = "vanilladot"
                          , C = fineCost[1]
                          , type = 'eps-svr'))
  preds_ <- predict(ksvmModel,svmFrame[holdout[[k]],])
  preds <- expm1(preds_)
  test3 <- expm1(targetValue[holdout[[k]]])
  score3 <- rmsle(preds ,test3)
  AllResultsSVM <- c(AllResultsSVM,score3)
```

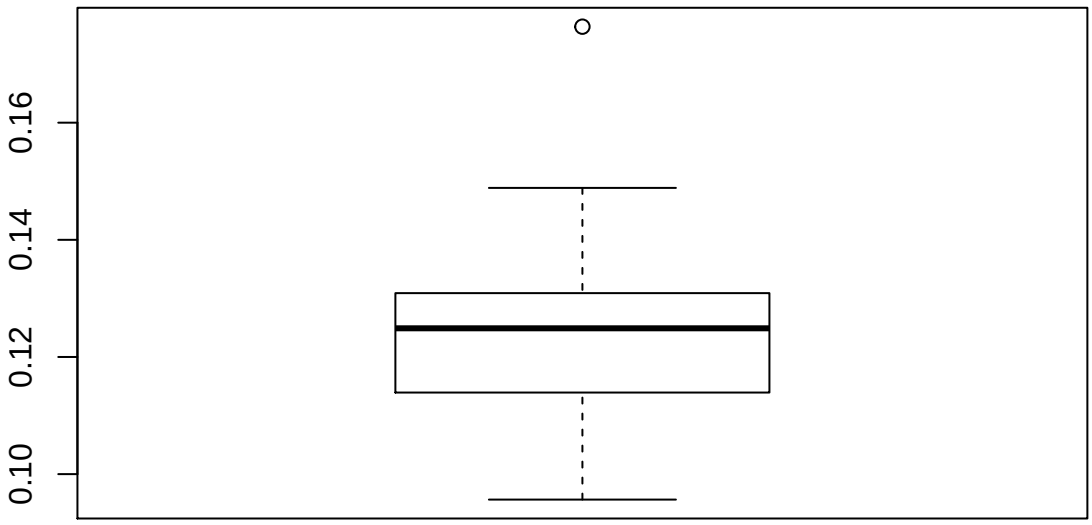
```
}

boxplot(AllResultsSVM)
```




```
mean(AllResultsSVM)
```

```
## [1] 0.1324978
```

```
N <- nrow(svmFrame)

kfolds <- 10
set.seed(5948)

holdout <- split(sample(1:N), 1:kfolds)
Id <- 1:nrow(svmTestFrame)
SVMaggs <- data.frame(Id)
AllResultsSVM <- c()
for (k in 1:kfolds) {

  ksvmModel <- quiet(ksvm(x = targetValue ~ . , data = svmFrame[-holdout[[k]],]
                          , kernel = "laplacedot"
                          , C = (fineCostype_2[1])
                          , type = 'nu-svr'))
  preds_ <- predict(ksvmModel,svmFrame[holdout[[k]],])
  preds <- expm1(preds_)
  test3 <- expm1(targetValue[holdout[[k]]])
  score3 <- rmsle(preds ,test3)
  AllResultsSVM <- c(AllResultsSVM,score3)

  kagPreds <- predict(ksvmModel ,svmTestFrame)
  newCol <- expm1(kagPreds)
  SVMaggs <- cbind(SVMaggs , newCol)
}

boxplot(AllResultsSVM)
```




```
mean(AllResultsSVM)
```

```
## [1] 0.1273599
```

Predictions on the Kaggle testing data are made and averaged across all folds. They are then stored into a data frame that will later be used.

```
SVMaggs_sub <- SVMaggs[,-1]
SVMaggs_sub$preFinal <- rowSums(SVMaggs_sub)
SVMaggs_sub$preFinal <- (SVMaggs_sub$preFinal)/kfolds
```

Decision Trees

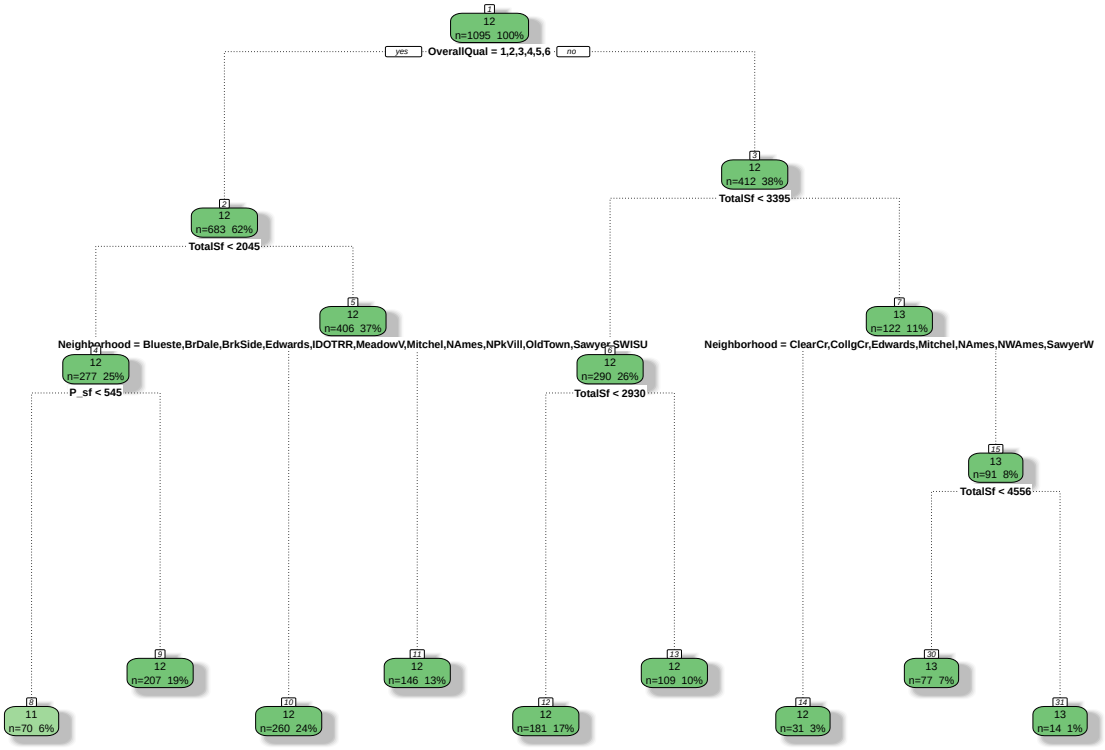
A decision tree asks sequential questions about the data and then comes to a prediction about an observation. To make predictions for sale price 3 trees are grown. First it uses the default pruning. Then it does not prune the tree at all. Finally the tree is pruned using the optimal values from the cross validated error rate observed from the unpruned tree. This produces the best results. However, none of the results are strong enough to be used in the submission on Kaggle. For the regression models, only the default pruned tree is visualized because of the size of the other trees.

Multiple trees are also grown to try and classify which neighborhood an observation comes from. Both of these trees are visualized.

```
treeFrame <- svmFrame

#default pruning
tree1reg <- rpart(targetValue ~ ., treeFrame[indexes,])
tree1regPreds <- predict(tree1reg, treeFrame[-indexes,])
rmsle(expm1(targetValue[-indexes]), expm1(tree1regPreds))
```

```
## [1] 0.1966314
```

```
fancyRpartPlot(tree1reg)
```



Rattle 2020−Sep−11 11:40:38 ronenreouveni

```
#no tuning
tree2 <- rpart(targetValue ~ ., treeFrame[indexes,],
               control = rpart.control(minbucket = 1, minsplit=1, cp=-1)
               , model = T)
rsq.rpart(tree2)
```

```
##
## Regression tree:
## rpart(formula = targetValue ~ ., data = treeFrame[indexes, ],
##     model = T, control = rpart.control(minbucket = 1, minsplit = 1,
##         cp = -1))
##
## Variables actually used in tree construction:
##  [1] Alley          BedroomAbvGr   BldgType       BsmtCond       BsmtExposure
##  [6] BsmtFinSF1     BsmtFinSF2     BsmtFinType1   BsmtFinType2   BsmtHalfBath
## [11] BsmtQual       BsmtUnfSF      Condition1     Electrical     EnclosedPorch
## [16] Exterior1st    Exterior2nd    Fence          FireplaceQu    Fireplaces
## [21] FullBath       GarageArea     GarageCars     GarageCond     GarageFinish
## [26] GarageType     GarageYrBlt    GrLivArea      HalfBath       HeatingQC
## [31] KitchenQual    LandContour    LandSlope      LotArea        LotConfig
## [36] LotFrontage    LotShape       lotSizeRatio   MasVnrArea     MasVnrType
## [41] MiscVal        MoSold         MSSubClass     MSZoning       Neighborhood
## [46] OpenPorchSF    OverallCond    OverallQual    P_sf           PavedDrive
## [51] Price_SF_land  Price_SF_land_ QualCond       SaleCondition  SaleType
## [56] Sf_Quality     T_bathrooms    Tot_out        TotalBsmtSF    TotalSf
## [61] TotRmsAbvGrd   WoodDeckSF     X1stFlrSF      X2ndFlrSF      YearBuilt
## [66] YearRemodAdd   YrSold
##
```

```
## Root node error: 174.98/1095 = 0.1598
##
## n= 1095
##
##               CP nsplit  rel error  xerror     xstd
## 1     4.4395e-01      0 1.0000e+00 1.00289 0.051667
## 2     1.1688e-01      1 5.5605e-01 0.58325 0.032078
## 3     1.0910e-01      2 4.3917e-01 0.45918 0.026998
## 4     3.6009e-02      3 3.3007e-01 0.36263 0.021305
## 5     3.0597e-02      4 2.9406e-01 0.33151 0.020364
## 6     2.9240e-02      5 2.6347e-01 0.31282 0.019268
## 7     1.2232e-02      6 2.3423e-01 0.25877 0.016749
## 8     1.0482e-02      7 2.2199e-01 0.25285 0.016334
## 9     1.0296e-02      8 2.1151e-01 0.24901 0.015909
## 10    9.8718e-03      9 2.0122e-01 0.24916 0.015926
## 11    9.2999e-03     10 1.9134e-01 0.24926 0.016043
## 12    7.5387e-03     11 1.8204e-01 0.24417 0.016010
## 13    7.5042e-03     12 1.7450e-01 0.23373 0.015192
## 14    6.3150e-03     13 1.6700e-01 0.23105 0.015066
## 15    5.0558e-03     14 1.6069e-01 0.22239 0.014466
## 16    4.3104e-03     15 1.5563e-01 0.21215 0.013558
## 17    4.2308e-03     16 1.5132e-01 0.21709 0.014547
## 18    3.7486e-03     17 1.4709e-01 0.21701 0.014605
## 19    3.6445e-03     18 1.4334e-01 0.21711 0.014603
## 20    3.5077e-03     19 1.3970e-01 0.21439 0.014437
## 21    3.2229e-03     20 1.3619e-01 0.21469 0.014757
## 22    3.1854e-03     21 1.3296e-01 0.21820 0.015165
## 23    2.9807e-03     22 1.2978e-01 0.21867 0.015282
## 24    2.8370e-03     23 1.2680e-01 0.22109 0.015539
## 25    2.7947e-03     24 1.2396e-01 0.22077 0.015560
## 26    2.7911e-03     25 1.2117e-01 0.22176 0.015593
## 27    2.7871e-03     27 1.1558e-01 0.22138 0.015597
## 28    2.3228e-03     28 1.1280e-01 0.22416 0.016031
## 29    2.3051e-03     29 1.1047e-01 0.23029 0.016814
## 30    2.2443e-03     30 1.0817e-01 0.23158 0.017055
## 31    2.2359e-03     31 1.0593e-01 0.23165 0.017046
## 32    1.9969e-03     32 1.0369e-01 0.23485 0.017115
## 33    1.8379e-03     33 1.0169e-01 0.23184 0.017046
## 34    1.8125e-03     34 9.9855e-02 0.23066 0.017031
## 35    1.8052e-03     35 9.8042e-02 0.23102 0.017134
## 36    1.7553e-03     36 9.6237e-02 0.23021 0.017096
## 37    1.6169e-03     37 9.4482e-02 0.23320 0.017279
## 38    1.5529e-03     38 9.2865e-02 0.23308 0.017235
## 39    1.5431e-03     40 8.9759e-02 0.23350 0.017096
## 40    1.5183e-03     41 8.8216e-02 0.23375 0.017099
## 41    1.4900e-03     42 8.6698e-02 0.23336 0.017090
## 42    1.3741e-03     44 8.3718e-02 0.23228 0.017074
## 43    1.3684e-03     45 8.2344e-02 0.23169 0.017127
## 44    1.2545e-03     46 8.0975e-02 0.23236 0.017052
## 45    1.2232e-03     47 7.9721e-02 0.23398 0.017107
## 46    1.2204e-03     48 7.8498e-02 0.23524 0.017310
## 47    1.2156e-03     49 7.7277e-02 0.23530 0.017313
## 48    1.1786e-03     50 7.6062e-02 0.23458 0.017287
## 49    1.1759e-03     51 7.4883e-02 0.23404 0.017278
```

```
## 50    1.1678e-03     52 7.3707e-02 0.23404 0.017278
## 51    1.1327e-03     53 7.2539e-02 0.23473 0.017303
## 52    1.1200e-03     54 7.1407e-02 0.23408 0.017317
## 53    1.0894e-03     55 7.0287e-02 0.23366 0.017298
## 54    1.0779e-03     56 6.9197e-02 0.23420 0.017311
## 55    1.0523e-03     57 6.8120e-02 0.23385 0.017307
## 56    1.0339e-03     58 6.7067e-02 0.23442 0.017308
## 57    1.0230e-03     59 6.6033e-02 0.23700 0.017344
## 58    1.0161e-03     60 6.5010e-02 0.23712 0.017341
## 59    9.4452e-04     61 6.3994e-02 0.23598 0.017155
## 60    9.4165e-04     62 6.3050e-02 0.23376 0.016825
## 61    8.8993e-04     63 6.2108e-02 0.23158 0.016722
## 62    8.6166e-04     64 6.1218e-02 0.23208 0.016695
## 63    8.3151e-04     65 6.0357e-02 0.23166 0.016605
## 64    8.0587e-04     66 5.9525e-02 0.23230 0.016592
## 65    7.5391e-04     67 5.8719e-02 0.23408 0.016696
## 66    7.3364e-04     68 5.7965e-02 0.23793 0.016786
## 67    7.1307e-04     69 5.7232e-02 0.23904 0.016811
## 68    7.1206e-04     70 5.6519e-02 0.23991 0.016813
## 69    6.9055e-04     71 5.5806e-02 0.23958 0.016807
## 70    6.8686e-04     72 5.5116e-02 0.24031 0.016859
## 71    6.7875e-04     73 5.4429e-02 0.24035 0.016854
## 72    6.6554e-04     74 5.3750e-02 0.24072 0.016953
## 73    6.5018e-04     75 5.3085e-02 0.24048 0.016940
## 74    6.4716e-04     76 5.2435e-02 0.24051 0.016940
## 75    6.4370e-04     77 5.1787e-02 0.24093 0.016943
## 76    6.4139e-04     78 5.1144e-02 0.24261 0.016974
## 77    6.2580e-04     79 5.0502e-02 0.24220 0.016976
## 78    6.1835e-04     80 4.9877e-02 0.24256 0.016973
## 79    6.0005e-04     81 4.9258e-02 0.24235 0.016980
## 80    5.9429e-04     82 4.8658e-02 0.24334 0.016999
## 81    5.9240e-04     83 4.8064e-02 0.24342 0.016998
## 82    5.8295e-04     84 4.7471e-02 0.24409 0.017024
## 83    5.8173e-04     85 4.6889e-02 0.24685 0.017439
## 84    5.7666e-04     86 4.6307e-02 0.24783 0.017698
## 85    5.7314e-04     87 4.5730e-02 0.24829 0.017707
## 86    5.5743e-04     88 4.5157e-02 0.24802 0.017402
## 87    5.5423e-04     89 4.4600e-02 0.24870 0.017414
## 88    5.4156e-04     90 4.4045e-02 0.24883 0.017440
## 89    5.3312e-04     91 4.3504e-02 0.24896 0.017435
## 90    5.2617e-04     92 4.2971e-02 0.24909 0.017434
## 91    5.1825e-04     93 4.2444e-02 0.24885 0.017403
## 92    5.0345e-04     94 4.1926e-02 0.24845 0.017213
## 93    5.0001e-04     95 4.1423e-02 0.24780 0.017203
## 94    4.8997e-04     96 4.0923e-02 0.24842 0.017210
## 95    4.8839e-04     97 4.0433e-02 0.24855 0.017208
## 96    4.8653e-04     98 3.9944e-02 0.24851 0.017208
## 97    4.8539e-04     99 3.9458e-02 0.24851 0.017208
## 98    4.7202e-04    100 3.8972e-02 0.24869 0.017204
## 99    4.3912e-04    101 3.8500e-02 0.24856 0.017237
## 100   4.3596e-04    102 3.8061e-02 0.24953 0.017282
## 101   4.2899e-04    103 3.7625e-02 0.24939 0.017272
## 102   4.2558e-04    104 3.7196e-02 0.24941 0.017187
## 103   4.2088e-04    105 3.6771e-02 0.24944 0.017193
```

```
## 104   4.1933e-04    106 3.6350e-02 0.24961 0.017200
## 105   4.0430e-04    107 3.5931e-02 0.25078 0.017261
## 106   3.9240e-04    108 3.5526e-02 0.25263 0.017356
## 107   3.8428e-04    109 3.5134e-02 0.25378 0.017401
## 108   3.8359e-04    110 3.4750e-02 0.25491 0.017388
## 109   3.7912e-04    111 3.4366e-02 0.25598 0.017405
## 110   3.7504e-04    112 3.3987e-02 0.25611 0.017412
## 111   3.7262e-04    113 3.3612e-02 0.25590 0.017418
## 112   3.6637e-04    114 3.3239e-02 0.25557 0.017403
## 113   3.6185e-04    115 3.2873e-02 0.25704 0.017415
## 114   3.5818e-04    117 3.2149e-02 0.25704 0.017415
## 115   3.5233e-04    118 3.1791e-02 0.25718 0.017410
## 116   3.4269e-04    119 3.1439e-02 0.25801 0.017433
## 117   3.3756e-04    120 3.1096e-02 0.25805 0.017448
## 118   3.3640e-04    121 3.0758e-02 0.25840 0.017455
## 119   3.2907e-04    122 3.0422e-02 0.25929 0.017477
## 120   3.2874e-04    123 3.0093e-02 0.25940 0.017487
## 121   3.2428e-04    124 2.9764e-02 0.25937 0.017487
## 122   3.2384e-04    125 2.9440e-02 0.25989 0.017492
## 123   3.1901e-04    126 2.9116e-02 0.26006 0.017491
## 124   3.1646e-04    127 2.8797e-02 0.26015 0.017490
## 125   3.1494e-04    128 2.8481e-02 0.26015 0.017490
## 126   3.1452e-04    129 2.8166e-02 0.25995 0.017488
## 127   3.1333e-04    130 2.7851e-02 0.25995 0.017488
## 128   3.0878e-04    131 2.7538e-02 0.26020 0.017488
## 129   3.0649e-04    132 2.7229e-02 0.26029 0.017485
## 130   2.9777e-04    133 2.6923e-02 0.26166 0.017528
## 131   2.9497e-04    134 2.6625e-02 0.26292 0.017591
## 132   2.8293e-04    135 2.6330e-02 0.26395 0.017586
## 133   2.8164e-04    136 2.6047e-02 0.26412 0.017579
## 134   2.7745e-04    137 2.5765e-02 0.26400 0.017579
## 135   2.7697e-04    138 2.5488e-02 0.26320 0.017542
## 136   2.7501e-04    139 2.5211e-02 0.26320 0.017542
## 137   2.6586e-04    141 2.4661e-02 0.26362 0.017532
## 138   2.6212e-04    142 2.4395e-02 0.26271 0.017489
## 139   2.5662e-04    143 2.4133e-02 0.26350 0.017576
## 140   2.5538e-04    144 2.3876e-02 0.26368 0.017577
## 141   2.5509e-04    145 2.3621e-02 0.26342 0.017516
## 142   2.5442e-04    146 2.3366e-02 0.26323 0.017518
## 143   2.4526e-04    147 2.3111e-02 0.26340 0.017517
## 144   2.3640e-04    148 2.2866e-02 0.26262 0.017470
## 145   2.3626e-04    149 2.2630e-02 0.26312 0.017501
## 146   2.3518e-04    150 2.2393e-02 0.26314 0.017502
## 147   2.3364e-04    151 2.2158e-02 0.26340 0.017502
## 148   2.3078e-04    152 2.1925e-02 0.26322 0.017503
## 149   2.2935e-04    153 2.1694e-02 0.26331 0.017503
## 150   2.2811e-04    154 2.1464e-02 0.26372 0.017513
## 151   2.2662e-04    155 2.1236e-02 0.26370 0.017515
## 152   2.2538e-04    156 2.1010e-02 0.26315 0.017505
## 153   2.2397e-04    157 2.0784e-02 0.26308 0.017506
## 154   2.2375e-04    158 2.0560e-02 0.26331 0.017507
## 155   2.1951e-04    159 2.0337e-02 0.26384 0.017511
## 156   2.1627e-04    160 2.0117e-02 0.26376 0.017499
## 157   2.0686e-04    162 1.9685e-02 0.26515 0.017540
```

```
## 158   2.0644e-04    163 1.9478e-02 0.26379 0.017516
## 159   2.0163e-04    164 1.9271e-02 0.26326 0.017519
## 160   1.9675e-04    165 1.9070e-02 0.26316 0.017523
## 161   1.9326e-04    166 1.8873e-02 0.26267 0.017501
## 162   1.9129e-04    167 1.8680e-02 0.26258 0.017500
## 163   1.8667e-04    168 1.8488e-02 0.26419 0.017629
## 164   1.8434e-04    169 1.8302e-02 0.26341 0.017618
## 165   1.8209e-04    170 1.8117e-02 0.26379 0.017621
## 166   1.8164e-04    171 1.7935e-02 0.26399 0.017622
## 167   1.8023e-04    172 1.7754e-02 0.26408 0.017621
## 168   1.7721e-04    173 1.7573e-02 0.26401 0.017619
## 169   1.7196e-04    175 1.7219e-02 0.26417 0.017628
## 170   1.6580e-04    176 1.7047e-02 0.26544 0.017659
## 171   1.6539e-04    177 1.6881e-02 0.26594 0.017668
## 172   1.6423e-04    178 1.6716e-02 0.26594 0.017668
## 173   1.6279e-04    179 1.6552e-02 0.26486 0.017513
## 174   1.6268e-04    180 1.6389e-02 0.26487 0.017513
## 175   1.6248e-04    181 1.6226e-02 0.26487 0.017513
## 176   1.6239e-04    182 1.6064e-02 0.26487 0.017513
## 177   1.6221e-04    183 1.5901e-02 0.26487 0.017513
## 178   1.6096e-04    184 1.5739e-02 0.26479 0.017514
## 179   1.5796e-04    185 1.5578e-02 0.26445 0.017505
## 180   1.5757e-04    186 1.5420e-02 0.26414 0.017506
## 181   1.5653e-04    187 1.5263e-02 0.26411 0.017507
## 182   1.5369e-04    188 1.5106e-02 0.26396 0.017503
## 183   1.5264e-04    189 1.4952e-02 0.26386 0.017503
## 184   1.5256e-04    190 1.4800e-02 0.26394 0.017503
## 185   1.5040e-04    191 1.4647e-02 0.26350 0.017497
## 186   1.4660e-04    192 1.4497e-02 0.26257 0.017453
## 187   1.4489e-04    193 1.4350e-02 0.26274 0.017453
## 188   1.4283e-04    194 1.4205e-02 0.26289 0.017462
## 189   1.4228e-04    195 1.4062e-02 0.26291 0.017462
## 190   1.3902e-04    196 1.3920e-02 0.26382 0.017582
## 191   1.3862e-04    197 1.3781e-02 0.26383 0.017582
## 192   1.3825e-04    198 1.3642e-02 0.26378 0.017582
## 193   1.3766e-04    199 1.3504e-02 0.26357 0.017575
## 194   1.3503e-04    200 1.3367e-02 0.26370 0.017665
## 195   1.3362e-04    201 1.3232e-02 0.26439 0.017786
## 196   1.3321e-04    202 1.3098e-02 0.26459 0.017785
## 197   1.3244e-04    203 1.2965e-02 0.26477 0.017797
## 198   1.3216e-04    204 1.2832e-02 0.26459 0.017795
## 199   1.2695e-04    205 1.2700e-02 0.26501 0.017814
## 200   1.2613e-04    206 1.2573e-02 0.26560 0.017845
## 201   1.2359e-04    208 1.2321e-02 0.26578 0.017806
## 202   1.2337e-04    209 1.2197e-02 0.26680 0.017863
## 203   1.2248e-04    210 1.2074e-02 0.26685 0.017863
## 204   1.2130e-04    211 1.1951e-02 0.26694 0.017867
## 205   1.1880e-04    212 1.1830e-02 0.26669 0.017867
## 206   1.1822e-04    213 1.1711e-02 0.26609 0.017769
## 207   1.1743e-04    214 1.1593e-02 0.26617 0.017769
## 208   1.1436e-04    215 1.1476e-02 0.26614 0.017771
## 209   1.1199e-04    216 1.1361e-02 0.26611 0.017781
## 210   1.0995e-04    217 1.1249e-02 0.26504 0.017656
## 211   1.0700e-04    218 1.1139e-02 0.26547 0.017700
```

```
## 212   1.0578e-04    219 1.1032e-02 0.26528 0.017699
## 213   1.0426e-04    220 1.0927e-02 0.26532 0.017720
## 214   1.0394e-04    221 1.0822e-02 0.26550 0.017726
## 215   1.0330e-04    222 1.0718e-02 0.26552 0.017726
## 216   1.0037e-04    223 1.0615e-02 0.26613 0.017765
## 217   1.0028e-04    224 1.0515e-02 0.26605 0.017769
## 218   9.9777e-05    225 1.0414e-02 0.26606 0.017769
## 219   9.9263e-05    226 1.0315e-02 0.26657 0.017777
## 220   9.8072e-05    227 1.0215e-02 0.26647 0.017777
## 221   9.7461e-05    228 1.0117e-02 0.26657 0.017776
## 222   9.6516e-05    229 1.0020e-02 0.26659 0.017775
## 223   9.4736e-05    230 9.9234e-03 0.26684 0.017779
## 224   9.4586e-05    231 9.8287e-03 0.26718 0.017790
## 225   9.3858e-05    232 9.7341e-03 0.26706 0.017790
## 226   9.3440e-05    233 9.6402e-03 0.26685 0.017786
## 227   9.3098e-05    234 9.5468e-03 0.26695 0.017785
## 228   9.2412e-05    235 9.4537e-03 0.26689 0.017785
## 229   9.2023e-05    236 9.3613e-03 0.26685 0.017785
## 230   9.1863e-05    237 9.2692e-03 0.26685 0.017785
## 231   9.1767e-05    238 9.1774e-03 0.26685 0.017785
## 232   8.9872e-05    239 9.0856e-03 0.26696 0.017785
## 233   8.7102e-05    240 8.9957e-03 0.26766 0.017788
## 234   8.7051e-05    241 8.9086e-03 0.26806 0.017812
## 235   8.6656e-05    242 8.8216e-03 0.26799 0.017813
## 236   8.5495e-05    243 8.7349e-03 0.26821 0.017818
## 237   8.4839e-05    244 8.6494e-03 0.26775 0.017805
## 238   8.4282e-05    245 8.5646e-03 0.26785 0.017804
## 239   8.2721e-05    246 8.4803e-03 0.26762 0.017801
## 240   8.1686e-05    247 8.3976e-03 0.26754 0.017798
## 241   8.0857e-05    248 8.3159e-03 0.26798 0.017812
## 242   8.0729e-05    249 8.2351e-03 0.26791 0.017813
## 243   8.0569e-05    250 8.1543e-03 0.26803 0.017812
## 244   8.0324e-05    251 8.0738e-03 0.26804 0.017812
## 245   8.0130e-05    252 7.9934e-03 0.26803 0.017812
## 246   8.0078e-05    253 7.9133e-03 0.26846 0.017818
## 247   7.9935e-05    254 7.8332e-03 0.26845 0.017818
## 248   7.9193e-05    255 7.7533e-03 0.26845 0.017818
## 249   7.7000e-05    256 7.6741e-03 0.26799 0.017629
## 250   7.6766e-05    257 7.5971e-03 0.26799 0.017628
## 251   7.6604e-05    258 7.5203e-03 0.26767 0.017621
## 252   7.6247e-05    260 7.3671e-03 0.26769 0.017621
## 253   7.5805e-05    261 7.2909e-03 0.26796 0.017641
## 254   7.4761e-05    262 7.2151e-03 0.26781 0.017639
## 255   7.1814e-05    263 7.1403e-03 0.26908 0.017666
## 256   7.1415e-05    264 7.0685e-03 0.26869 0.017651
## 257   7.0309e-05    265 6.9971e-03 0.26868 0.017649
## 258   6.9921e-05    266 6.9268e-03 0.26866 0.017650
## 259   6.9430e-05    267 6.8568e-03 0.26869 0.017648
## 260   6.8952e-05    268 6.7874e-03 0.26862 0.017649
## 261   6.7306e-05    269 6.7185e-03 0.26851 0.017656
## 262   6.6678e-05    270 6.6512e-03 0.26913 0.017658
## 263   6.5248e-05    271 6.5845e-03 0.26916 0.017658
## 264   6.5190e-05    272 6.5192e-03 0.26940 0.017668
## 265   6.4921e-05    273 6.4540e-03 0.26955 0.017668
```

```
## 266   6.3936e-05    274 6.3891e-03 0.26991 0.017691
## 267   6.3714e-05    275 6.3252e-03 0.27012 0.017694
## 268   6.2283e-05    276 6.2615e-03 0.26985 0.017696
## 269   5.9202e-05    277 6.1992e-03 0.26944 0.017683
## 270   5.8877e-05    278 6.1400e-03 0.26968 0.017681
## 271   5.8839e-05    279 6.0811e-03 0.26974 0.017681
## 272   5.8747e-05    280 6.0223e-03 0.26972 0.017681
## 273   5.8478e-05    281 5.9635e-03 0.26968 0.017682
## 274   5.6753e-05    283 5.8466e-03 0.26986 0.017680
## 275   5.5027e-05    284 5.7898e-03 0.27026 0.017686
## 276   5.4254e-05    285 5.7348e-03 0.27015 0.017682
## 277   5.4156e-05    286 5.6805e-03 0.27010 0.017682
## 278   5.4062e-05    287 5.6264e-03 0.27010 0.017682
## 279   5.3838e-05    288 5.5723e-03 0.27007 0.017682
## 280   5.3793e-05    289 5.5185e-03 0.27007 0.017682
## 281   5.3749e-05    290 5.4647e-03 0.27007 0.017682
## 282   5.3731e-05    291 5.4109e-03 0.27005 0.017683
## 283   5.1871e-05    292 5.3572e-03 0.27040 0.017682
## 284   5.1193e-05    293 5.3053e-03 0.27078 0.017672
## 285   5.0735e-05    294 5.2541e-03 0.27110 0.017675
## 286   5.0125e-05    295 5.2034e-03 0.27144 0.017677
## 287   4.7527e-05    296 5.1533e-03 0.27170 0.017672
## 288   4.7435e-05    297 5.1058e-03 0.27164 0.017673
## 289   4.6554e-05    298 5.0583e-03 0.27158 0.017674
## 290   4.6126e-05    299 5.0118e-03 0.27166 0.017675
## 291   4.5716e-05    300 4.9656e-03 0.27179 0.017675
## 292   4.5412e-05    301 4.9199e-03 0.27178 0.017679
## 293   4.4589e-05    302 4.8745e-03 0.27199 0.017687
## 294   4.3289e-05    303 4.8299e-03 0.27135 0.017627
## 295   4.3192e-05    304 4.7866e-03 0.27151 0.017632
## 296   4.2971e-05    305 4.7434e-03 0.27157 0.017632
## 297   4.2758e-05    306 4.7005e-03 0.27142 0.017631
## 298   4.2659e-05    307 4.6577e-03 0.27142 0.017631
## 299   4.2564e-05    308 4.6151e-03 0.27141 0.017631
## 300   4.2558e-05    309 4.5725e-03 0.27143 0.017631
## 301   4.1879e-05    310 4.5299e-03 0.27143 0.017631
## 302   4.1836e-05    311 4.4881e-03 0.27151 0.017631
## 303   4.1498e-05    312 4.4462e-03 0.27155 0.017631
## 304   4.0322e-05    313 4.4047e-03 0.27163 0.017643
## 305   4.0130e-05    314 4.3644e-03 0.27169 0.017642
## 306   3.9719e-05    315 4.3243e-03 0.27169 0.017642
## 307   3.8523e-05    316 4.2845e-03 0.27177 0.017644
## 308   3.8342e-05    317 4.2460e-03 0.27129 0.017599
## 309   3.7681e-05    318 4.2077e-03 0.27130 0.017599
## 310   3.7188e-05    319 4.1700e-03 0.27137 0.017599
## 311   3.7091e-05    320 4.1328e-03 0.27137 0.017599
## 312   3.7010e-05    321 4.0957e-03 0.27135 0.017599
## 313   3.6733e-05    322 4.0587e-03 0.27140 0.017612
## 314   3.6164e-05    323 4.0220e-03 0.27138 0.017613
## 315   3.6089e-05    324 3.9858e-03 0.27106 0.017600
## 316   3.6055e-05    325 3.9497e-03 0.27106 0.017600
## 317   3.5971e-05    326 3.9137e-03 0.27116 0.017600
## 318   3.5549e-05    327 3.8777e-03 0.27116 0.017600
## 319   3.5219e-05    328 3.8422e-03 0.27137 0.017608
```

```
## 320   3.4903e-05    329 3.8069e-03 0.27146 0.017609
## 321   3.4571e-05    330 3.7720e-03 0.27141 0.017609
## 322   3.4434e-05    331 3.7375e-03 0.27131 0.017604
## 323   3.4410e-05    332 3.7030e-03 0.27131 0.017604
## 324   3.4287e-05    333 3.6686e-03 0.27140 0.017604
## 325   3.4074e-05    334 3.6343e-03 0.27140 0.017604
## 326   3.2818e-05    335 3.6003e-03 0.27126 0.017602
## 327   3.2792e-05    336 3.5674e-03 0.27130 0.017602
## 328   3.2759e-05    337 3.5346e-03 0.27130 0.017602
## 329   3.2728e-05    338 3.5019e-03 0.27130 0.017602
## 330   3.2698e-05    339 3.4692e-03 0.27130 0.017602
## 331   3.2026e-05    340 3.4365e-03 0.27139 0.017603
## 332   3.1997e-05    341 3.4044e-03 0.27126 0.017600
## 333   3.1794e-05    342 3.3724e-03 0.27116 0.017598
## 334   3.1721e-05    343 3.3406e-03 0.27110 0.017598
## 335   3.1683e-05    344 3.3089e-03 0.27110 0.017598
## 336   3.1679e-05    345 3.2772e-03 0.27110 0.017598
## 337   3.1450e-05    346 3.2456e-03 0.27110 0.017598
## 338   3.1290e-05    347 3.2141e-03 0.27110 0.017598
## 339   3.1009e-05    348 3.1828e-03 0.27100 0.017598
## 340   3.0563e-05    349 3.1518e-03 0.27084 0.017595
## 341   3.0494e-05    350 3.1212e-03 0.27104 0.017597
## 342   3.0422e-05    351 3.0907e-03 0.27121 0.017598
## 343   3.0281e-05    352 3.0603e-03 0.27121 0.017598
## 344   2.9948e-05    353 3.0300e-03 0.27123 0.017598
## 345   2.9867e-05    354 3.0001e-03 0.27116 0.017598
## 346   2.9397e-05    355 2.9702e-03 0.27120 0.017598
## 347   2.9208e-05    356 2.9408e-03 0.27141 0.017598
## 348   2.8302e-05    357 2.9116e-03 0.27144 0.017602
## 349   2.7274e-05    358 2.8833e-03 0.27149 0.017601
## 350   2.6894e-05    359 2.8561e-03 0.27158 0.017738
## 351   2.6504e-05    360 2.8292e-03 0.27155 0.017738
## 352   2.6280e-05    361 2.8027e-03 0.27155 0.017738
## 353   2.5799e-05    362 2.7764e-03 0.27157 0.017738
## 354   2.5679e-05    363 2.7506e-03 0.27141 0.017733
## 355   2.5648e-05    364 2.7249e-03 0.27141 0.017733
## 356   2.5592e-05    365 2.6992e-03 0.27141 0.017733
## 357   2.5269e-05    366 2.6737e-03 0.27134 0.017732
## 358   2.4711e-05    367 2.6484e-03 0.27158 0.017737
## 359   2.4439e-05    368 2.6237e-03 0.27189 0.017740
## 360   2.4075e-05    369 2.5992e-03 0.27237 0.017767
## 361   2.3741e-05    370 2.5752e-03 0.27248 0.017772
## 362   2.3482e-05    371 2.5514e-03 0.27245 0.017772
## 363   2.3356e-05    372 2.5279e-03 0.27223 0.017772
## 364   2.3223e-05    373 2.5046e-03 0.27243 0.017774
## 365   2.2713e-05    374 2.4814e-03 0.27233 0.017777
## 366   2.2657e-05    375 2.4586e-03 0.27248 0.017789
## 367   2.2282e-05    376 2.4360e-03 0.27252 0.017788
## 368   2.2231e-05    377 2.4137e-03 0.27254 0.017789
## 369   2.2185e-05    378 2.3915e-03 0.27254 0.017789
## 370   2.1940e-05    379 2.3693e-03 0.27239 0.017761
## 371   2.1898e-05    380 2.3474e-03 0.27251 0.017764
## 372   2.1895e-05    381 2.3255e-03 0.27251 0.017764
## 373   2.1776e-05    382 2.3036e-03 0.27260 0.017764
```

```
## 374   2.1634e-05    383 2.2818e-03 0.27266 0.017764
## 375   2.1522e-05    384 2.2601e-03 0.27266 0.017764
## 376   2.1154e-05    385 2.2386e-03 0.27291 0.017786
## 377   2.1140e-05    386 2.2175e-03 0.27292 0.017785
## 378   2.0967e-05    387 2.1963e-03 0.27291 0.017785
## 379   2.0928e-05    388 2.1754e-03 0.27298 0.017785
## 380   2.0790e-05    389 2.1544e-03 0.27316 0.017788
## 381   2.0684e-05    390 2.1336e-03 0.27320 0.017788
## 382   2.0575e-05    391 2.1130e-03 0.27316 0.017788
## 383   2.0428e-05    392 2.0924e-03 0.27316 0.017788
## 384   2.0102e-05    393 2.0720e-03 0.27314 0.017783
## 385   1.9778e-05    394 2.0519e-03 0.27349 0.017791
## 386   1.9606e-05    395 2.0321e-03 0.27362 0.017796
## 387   1.8490e-05    396 2.0125e-03 0.27345 0.017807
## 388   1.8222e-05    397 1.9940e-03 0.27356 0.017809
## 389   1.8177e-05    398 1.9758e-03 0.27349 0.017802
## 390   1.7974e-05    399 1.9576e-03 0.27331 0.017801
## 391   1.7529e-05    400 1.9396e-03 0.27346 0.017801
## 392   1.7469e-05    401 1.9221e-03 0.27359 0.017804
## 393   1.7452e-05    402 1.9046e-03 0.27359 0.017804
## 394   1.7198e-05    403 1.8872e-03 0.27387 0.017810
## 395   1.6829e-05    404 1.8700e-03 0.27378 0.017809
## 396   1.6675e-05    405 1.8531e-03 0.27380 0.017808
## 397   1.6489e-05    406 1.8365e-03 0.27378 0.017808
## 398   1.6333e-05    407 1.8200e-03 0.27369 0.017805
## 399   1.6145e-05    408 1.8036e-03 0.27371 0.017805
## 400   1.6086e-05    409 1.7875e-03 0.27395 0.017823
## 401   1.6072e-05    410 1.7714e-03 0.27395 0.017823
## 402   1.6031e-05    411 1.7553e-03 0.27395 0.017823
## 403   1.5719e-05    412 1.7393e-03 0.27394 0.017823
## 404   1.5680e-05    413 1.7236e-03 0.27376 0.017823
## 405   1.5456e-05    414 1.7079e-03 0.27347 0.017806
## 406   1.5225e-05    415 1.6924e-03 0.27362 0.017805
## 407   1.4940e-05    416 1.6772e-03 0.27373 0.017807
## 408   1.4910e-05    417 1.6623e-03 0.27376 0.017807
## 409   1.4802e-05    418 1.6474e-03 0.27383 0.017807
## 410   1.4763e-05    419 1.6326e-03 0.27383 0.017807
## 411   1.4653e-05    420 1.6178e-03 0.27389 0.017807
## 412   1.4602e-05    421 1.6032e-03 0.27389 0.017807
## 413   1.4515e-05    422 1.5886e-03 0.27391 0.017807
## 414   1.4464e-05    423 1.5740e-03 0.27392 0.017807
## 415   1.4263e-05    424 1.5596e-03 0.27388 0.017807
## 416   1.3962e-05    425 1.5453e-03 0.27411 0.017810
## 417   1.3783e-05    426 1.5313e-03 0.27407 0.017810
## 418   1.3774e-05    427 1.5176e-03 0.27407 0.017810
## 419   1.3656e-05    428 1.5038e-03 0.27409 0.017810
## 420   1.3625e-05    429 1.4901e-03 0.27409 0.017810
## 421   1.3512e-05    430 1.4765e-03 0.27408 0.017810
## 422   1.3392e-05    431 1.4630e-03 0.27410 0.017810
## 423   1.3363e-05    432 1.4496e-03 0.27410 0.017810
## 424   1.3361e-05    433 1.4362e-03 0.27410 0.017810
## 425   1.3148e-05    434 1.4229e-03 0.27424 0.017813
## 426   1.2946e-05    435 1.4097e-03 0.27431 0.017819
## 427   1.2945e-05    436 1.3968e-03 0.27427 0.017819
```

```
## 428   1.2823e-05    437 1.3838e-03 0.27456 0.017848
## 429   1.2288e-05    438 1.3710e-03 0.27478 0.017846
## 430   1.1986e-05    439 1.3587e-03 0.27471 0.017844
## 431   1.1902e-05    440 1.3467e-03 0.27483 0.017845
## 432   1.1844e-05    441 1.3348e-03 0.27484 0.017845
## 433   1.1071e-05    442 1.3230e-03 0.27486 0.017842
## 434   1.1049e-05    443 1.3119e-03 0.27496 0.017838
## 435   1.0874e-05    444 1.3009e-03 0.27506 0.017838
## 436   1.0781e-05    445 1.2900e-03 0.27499 0.017839
## 437   1.0573e-05    446 1.2792e-03 0.27477 0.017836
## 438   1.0563e-05    447 1.2686e-03 0.27479 0.017836
## 439   1.0418e-05    448 1.2581e-03 0.27485 0.017836
## 440   1.0335e-05    449 1.2477e-03 0.27476 0.017836
## 441   1.0214e-05    450 1.2373e-03 0.27479 0.017836
## 442   9.9032e-06    451 1.2271e-03 0.27495 0.017837
## 443   9.8965e-06    452 1.2172e-03 0.27510 0.017832
## 444   9.8195e-06    453 1.2073e-03 0.27510 0.017832
## 445   9.6599e-06    454 1.1975e-03 0.27504 0.017829
## 446   9.6383e-06    455 1.1878e-03 0.27504 0.017829
## 447   9.6128e-06    456 1.1782e-03 0.27504 0.017829
## 448   9.5928e-06    457 1.1686e-03 0.27504 0.017829
## 449   9.4199e-06    458 1.1590e-03 0.27478 0.017825
## 450   9.3523e-06    459 1.1496e-03 0.27476 0.017825
## 451   9.3145e-06    460 1.1402e-03 0.27488 0.017849
## 452   9.1804e-06    461 1.1309e-03 0.27483 0.017849
## 453   8.9546e-06    462 1.1217e-03 0.27491 0.017818
## 454   8.9157e-06    463 1.1128e-03 0.27496 0.017818
## 455   8.7880e-06    464 1.1039e-03 0.27487 0.017816
## 456   8.7792e-06    465 1.0951e-03 0.27487 0.017816
## 457   8.7502e-06    466 1.0863e-03 0.27487 0.017816
## 458   8.6776e-06    467 1.0775e-03 0.27490 0.017816
## 459   8.6748e-06    468 1.0689e-03 0.27490 0.017816
## 460   8.6656e-06    469 1.0602e-03 0.27490 0.017816
## 461   8.5489e-06    470 1.0515e-03 0.27487 0.017815
## 462   8.4741e-06    471 1.0430e-03 0.27494 0.017816
## 463   8.3887e-06    472 1.0345e-03 0.27498 0.017816
## 464   8.3684e-06    473 1.0261e-03 0.27499 0.017816
## 465   8.1914e-06    474 1.0177e-03 0.27498 0.017816
## 466   8.0522e-06    475 1.0096e-03 0.27521 0.017818
## 467   7.9892e-06    476 1.0015e-03 0.27522 0.017823
## 468   7.8431e-06    477 9.9351e-04 0.27521 0.017822
## 469   7.8296e-06    478 9.8567e-04 0.27535 0.017827
## 470   7.7858e-06    479 9.7784e-04 0.27535 0.017827
## 471   7.7849e-06    480 9.7005e-04 0.27535 0.017827
## 472   7.6234e-06    481 9.6227e-04 0.27535 0.017825
## 473   7.5778e-06    482 9.5464e-04 0.27535 0.017825
## 474   7.5196e-06    483 9.4707e-04 0.27518 0.017818
## 475   7.5193e-06    484 9.3955e-04 0.27523 0.017817
## 476   7.3840e-06    485 9.3203e-04 0.27526 0.017817
## 477   7.2304e-06    486 9.2464e-04 0.27524 0.017817
## 478   7.2222e-06    487 9.1741e-04 0.27527 0.017817
## 479   7.1818e-06    488 9.1019e-04 0.27531 0.017818
## 480   7.1258e-06    489 9.0301e-04 0.27533 0.017819
## 481   7.1220e-06    490 8.9588e-04 0.27537 0.017819
```

```
## 482   7.0731e-06    491 8.8876e-04 0.27537 0.017819
## 483   7.0475e-06    492 8.8169e-04 0.27536 0.017819
## 484   7.0428e-06    493 8.7464e-04 0.27540 0.017819
## 485   7.0327e-06    494 8.6760e-04 0.27546 0.017819
## 486   6.9474e-06    495 8.6056e-04 0.27541 0.017815
## 487   6.9267e-06    496 8.5362e-04 0.27541 0.017816
## 488   6.8270e-06    497 8.4669e-04 0.27542 0.017816
## 489   6.8022e-06    498 8.3986e-04 0.27541 0.017816
## 490   6.7500e-06    499 8.3306e-04 0.27545 0.017816
## 491   6.6965e-06    500 8.2631e-04 0.27544 0.017816
## 492   6.6817e-06    501 8.1962e-04 0.27542 0.017816
## 493   6.6671e-06    502 8.1293e-04 0.27541 0.017816
## 494   6.6504e-06    503 8.0627e-04 0.27541 0.017816
## 495   6.6457e-06    504 7.9962e-04 0.27541 0.017816
## 496   6.6392e-06    505 7.9297e-04 0.27544 0.017816
## 497   6.5928e-06    506 7.8633e-04 0.27544 0.017816
## 498   6.4821e-06    507 7.7974e-04 0.27544 0.017816
## 499   6.4340e-06    508 7.7326e-04 0.27547 0.017816
## 500   6.3821e-06    509 7.6682e-04 0.27547 0.017816
## 501   6.3486e-06    510 7.6044e-04 0.27558 0.017819
## 502   6.3289e-06    511 7.5409e-04 0.27562 0.017819
## 503   6.3267e-06    512 7.4776e-04 0.27562 0.017819
## 504   6.3245e-06    513 7.4144e-04 0.27562 0.017819
## 505   6.3145e-06    514 7.3511e-04 0.27562 0.017819
## 506   6.1150e-06    515 7.2880e-04 0.27574 0.017821
## 507   6.1134e-06    516 7.2268e-04 0.27586 0.017846
## 508   6.1000e-06    517 7.1657e-04 0.27586 0.017846
## 509   6.0345e-06    518 7.1047e-04 0.27583 0.017846
## 510   6.0244e-06    519 7.0443e-04 0.27583 0.017846
## 511   6.0154e-06    520 6.9841e-04 0.27583 0.017846
## 512   5.9750e-06    521 6.9239e-04 0.27581 0.017846
## 513   5.9332e-06    522 6.8642e-04 0.27576 0.017846
## 514   5.8836e-06    523 6.8049e-04 0.27572 0.017846
## 515   5.8262e-06    524 6.7460e-04 0.27590 0.017855
## 516   5.8034e-06    525 6.6878e-04 0.27591 0.017855
## 517   5.7744e-06    526 6.6297e-04 0.27591 0.017854
## 518   5.7590e-06    527 6.5720e-04 0.27591 0.017854
## 519   5.7560e-06    528 6.5144e-04 0.27591 0.017854
## 520   5.7283e-06    529 6.4568e-04 0.27591 0.017854
## 521   5.6853e-06    530 6.3996e-04 0.27591 0.017855
## 522   5.6397e-06    532 6.2858e-04 0.27592 0.017854
## 523   5.5883e-06    533 6.2294e-04 0.27621 0.017868
## 524   5.4521e-06    534 6.1736e-04 0.27619 0.017869
## 525   5.4438e-06    535 6.1190e-04 0.27616 0.017869
## 526   5.4310e-06    536 6.0646e-04 0.27616 0.017869
## 527   5.3040e-06    537 6.0103e-04 0.27626 0.017870
## 528   5.2635e-06    538 5.9573e-04 0.27660 0.017979
## 529   5.2145e-06    539 5.9046e-04 0.27663 0.017979
## 530   5.1969e-06    540 5.8525e-04 0.27656 0.017977
## 531   5.1394e-06    541 5.8005e-04 0.27653 0.017977
## 532   5.1238e-06    542 5.7491e-04 0.27653 0.017977
## 533   5.0230e-06    543 5.6979e-04 0.27650 0.017977
## 534   5.0168e-06    544 5.6476e-04 0.27638 0.017977
## 535   4.9762e-06    545 5.5975e-04 0.27638 0.017977
```

```
## 536   4.8503e-06    546 5.5477e-04 0.27643 0.017976
## 537   4.7767e-06    547 5.4992e-04 0.27646 0.017976
## 538   4.7601e-06    548 5.4514e-04 0.27639 0.017971
## 539   4.6816e-06    549 5.4038e-04 0.27648 0.017973
## 540   4.6477e-06    550 5.3570e-04 0.27641 0.017969
## 541   4.5873e-06    551 5.3106e-04 0.27637 0.017969
## 542   4.5212e-06    552 5.2647e-04 0.27642 0.017969
## 543   4.4763e-06    553 5.2195e-04 0.27639 0.017969
## 544   4.4600e-06    554 5.1747e-04 0.27637 0.017969
## 545   4.4453e-06    555 5.1301e-04 0.27637 0.017969
## 546   4.4062e-06    556 5.0856e-04 0.27639 0.017970
## 547   4.3979e-06    557 5.0416e-04 0.27639 0.017970
## 548   4.3818e-06    558 4.9976e-04 0.27639 0.017970
## 549   4.2829e-06    559 4.9538e-04 0.27637 0.017969
## 550   4.2782e-06    560 4.9110e-04 0.27637 0.017969
## 551   4.2377e-06    561 4.8682e-04 0.27637 0.017969
## 552   4.2296e-06    562 4.8258e-04 0.27637 0.017969
## 553   4.1958e-06    563 4.7835e-04 0.27639 0.017971
## 554   4.1352e-06    564 4.7415e-04 0.27640 0.017971
## 555   4.1217e-06    565 4.7002e-04 0.27640 0.017971
## 556   4.0900e-06    566 4.6590e-04 0.27639 0.017970
## 557   4.0733e-06    567 4.6181e-04 0.27638 0.017970
## 558   3.9945e-06    568 4.5773e-04 0.27650 0.017971
## 559   3.9904e-06    569 4.5374e-04 0.27647 0.017970
## 560   3.9786e-06    570 4.4975e-04 0.27647 0.017970
## 561   3.9676e-06    571 4.4577e-04 0.27647 0.017970
## 562   3.9531e-06    572 4.4180e-04 0.27647 0.017970
## 563   3.9216e-06    573 4.3785e-04 0.27643 0.017968
## 564   3.9011e-06    574 4.3393e-04 0.27648 0.017969
## 565   3.8727e-06    575 4.3003e-04 0.27648 0.017969
## 566   3.8617e-06    576 4.2615e-04 0.27648 0.017969
## 567   3.8538e-06    577 4.2229e-04 0.27648 0.017969
## 568   3.8143e-06    578 4.1844e-04 0.27652 0.017968
## 569   3.8005e-06    579 4.1463e-04 0.27648 0.017968
## 570   3.7993e-06    580 4.1082e-04 0.27649 0.017968
## 571   3.7793e-06    581 4.0703e-04 0.27648 0.017968
## 572   3.7744e-06    582 4.0325e-04 0.27648 0.017968
## 573   3.7667e-06    583 3.9947e-04 0.27648 0.017968
## 574   3.6694e-06    584 3.9571e-04 0.27639 0.017961
## 575   3.6579e-06    585 3.9204e-04 0.27644 0.017961
## 576   3.6270e-06    586 3.8838e-04 0.27626 0.017960
## 577   3.6045e-06    587 3.8475e-04 0.27627 0.017960
## 578   3.5603e-06    588 3.8115e-04 0.27625 0.017960
## 579   3.5187e-06    589 3.7759e-04 0.27623 0.017959
## 580   3.5187e-06    590 3.7407e-04 0.27625 0.017959
## 581   3.5018e-06    591 3.7055e-04 0.27625 0.017959
## 582   3.4779e-06    592 3.6705e-04 0.27622 0.017959
## 583   3.4519e-06    593 3.6357e-04 0.27636 0.017962
## 584   3.4378e-06    594 3.6012e-04 0.27635 0.017962
## 585   3.4367e-06    595 3.5668e-04 0.27633 0.017962
## 586   3.3647e-06    596 3.5324e-04 0.27627 0.017962
## 587   3.3406e-06    597 3.4988e-04 0.27624 0.017962
## 588   3.3044e-06    598 3.4654e-04 0.27623 0.017962
## 589   3.2640e-06    599 3.4323e-04 0.27624 0.017962
```

```
## 590   3.2596e-06    600 3.3997e-04 0.27620 0.017962
## 591   3.2323e-06    601 3.3671e-04 0.27615 0.017962
## 592   3.2313e-06    602 3.3348e-04 0.27615 0.017962
## 593   3.2022e-06    603 3.3025e-04 0.27615 0.017962
## 594   3.1986e-06    604 3.2704e-04 0.27615 0.017962
## 595   3.1961e-06    605 3.2384e-04 0.27615 0.017962
## 596   3.1653e-06    606 3.2065e-04 0.27614 0.017963
## 597   3.1364e-06    607 3.1748e-04 0.27615 0.017963
## 598   3.0745e-06    608 3.1435e-04 0.27612 0.017963
## 599   3.0184e-06    609 3.1127e-04 0.27615 0.017964
## 600   3.0011e-06    610 3.0825e-04 0.27616 0.017963
## 601   2.9868e-06    611 3.0525e-04 0.27618 0.017964
## 602   2.8803e-06    612 3.0227e-04 0.27619 0.017964
## 603   2.8598e-06    613 2.9939e-04 0.27617 0.017953
## 604   2.7543e-06    614 2.9653e-04 0.27622 0.017956
## 605   2.7295e-06    615 2.9377e-04 0.27627 0.017955
## 606   2.7286e-06    616 2.9104e-04 0.27628 0.017955
## 607   2.6972e-06    617 2.8831e-04 0.27628 0.017955
## 608   2.6708e-06    618 2.8562e-04 0.27622 0.017955
## 609   2.6511e-06    619 2.8295e-04 0.27622 0.017955
## 610   2.6383e-06    620 2.8029e-04 0.27623 0.017954
## 611   2.5868e-06    621 2.7766e-04 0.27620 0.017955
## 612   2.5437e-06    622 2.7507e-04 0.27624 0.017955
## 613   2.5245e-06    623 2.7253e-04 0.27624 0.017955
## 614   2.5149e-06    624 2.7000e-04 0.27624 0.017955
## 615   2.4905e-06    625 2.6749e-04 0.27624 0.017955
## 616   2.4877e-06    626 2.6500e-04 0.27620 0.017955
## 617   2.4607e-06    627 2.6251e-04 0.27620 0.017955
## 618   2.4593e-06    628 2.6005e-04 0.27620 0.017955
## 619   2.4589e-06    629 2.5759e-04 0.27620 0.017955
## 620   2.4354e-06    630 2.5513e-04 0.27620 0.017955
## 621   2.4292e-06    631 2.5269e-04 0.27619 0.017954
## 622   2.4130e-06    632 2.5026e-04 0.27619 0.017954
## 623   2.3813e-06    633 2.4785e-04 0.27614 0.017953
## 624   2.3729e-06    634 2.4547e-04 0.27611 0.017953
## 625   2.3663e-06    635 2.4310e-04 0.27610 0.017953
## 626   2.3484e-06    636 2.4073e-04 0.27610 0.017953
## 627   2.3402e-06    637 2.3838e-04 0.27605 0.017953
## 628   2.2398e-06    638 2.3604e-04 0.27606 0.017953
## 629   2.2389e-06    639 2.3380e-04 0.27608 0.017953
## 630   2.2306e-06    640 2.3156e-04 0.27608 0.017953
## 631   2.1643e-06    641 2.2933e-04 0.27604 0.017954
## 632   2.1248e-06    642 2.2717e-04 0.27609 0.017955
## 633   2.1010e-06    643 2.2504e-04 0.27604 0.017954
## 634   2.0939e-06    644 2.2294e-04 0.27603 0.017954
## 635   2.0673e-06    645 2.2085e-04 0.27592 0.017949
## 636   2.0657e-06    646 2.1878e-04 0.27589 0.017949
## 637   2.0541e-06    647 2.1672e-04 0.27589 0.017949
## 638   2.0399e-06    648 2.1466e-04 0.27589 0.017949
## 639   2.0323e-06    649 2.1262e-04 0.27589 0.017949
## 640   2.0322e-06    650 2.1059e-04 0.27589 0.017949
## 641   2.0016e-06    651 2.0856e-04 0.27589 0.017949
## 642   1.9966e-06    652 2.0656e-04 0.27589 0.017949
## 643   1.9830e-06    653 2.0456e-04 0.27588 0.017949
```

```
## 644   1.9619e-06    654 2.0258e-04 0.27589 0.017949
## 645   1.9533e-06    655 2.0061e-04 0.27588 0.017949
## 646   1.9533e-06    656 1.9866e-04 0.27588 0.017949
## 647   1.9482e-06    657 1.9671e-04 0.27588 0.017949
## 648   1.9386e-06    658 1.9476e-04 0.27588 0.017949
## 649   1.9254e-06    659 1.9282e-04 0.27579 0.017947
## 650   1.8728e-06    660 1.9090e-04 0.27587 0.017947
## 651   1.8051e-06    661 1.8902e-04 0.27587 0.017942
## 652   1.7918e-06    662 1.8722e-04 0.27591 0.017943
## 653   1.7861e-06    663 1.8543e-04 0.27591 0.017943
## 654   1.7620e-06    664 1.8364e-04 0.27594 0.017943
## 655   1.7423e-06    665 1.8188e-04 0.27584 0.017940
## 656   1.7328e-06    666 1.8014e-04 0.27583 0.017940
## 657   1.7315e-06    667 1.7840e-04 0.27583 0.017940
## 658   1.7180e-06    668 1.7667e-04 0.27583 0.017940
## 659   1.7138e-06    669 1.7495e-04 0.27585 0.017940
## 660   1.6902e-06    670 1.7324e-04 0.27589 0.017941
## 661   1.6725e-06    671 1.7155e-04 0.27590 0.017941
## 662   1.6702e-06    672 1.6988e-04 0.27605 0.017961
## 663   1.6593e-06    673 1.6821e-04 0.27606 0.017961
## 664   1.6531e-06    674 1.6655e-04 0.27607 0.017960
## 665   1.6395e-06    675 1.6489e-04 0.27605 0.017961
## 666   1.6386e-06    676 1.6325e-04 0.27605 0.017961
## 667   1.6311e-06    677 1.6162e-04 0.27602 0.017961
## 668   1.6268e-06    678 1.5998e-04 0.27603 0.017961
## 669   1.6073e-06    679 1.5836e-04 0.27603 0.017961
## 670   1.5766e-06    680 1.5675e-04 0.27610 0.017961
## 671   1.5657e-06    681 1.5517e-04 0.27600 0.017957
## 672   1.5102e-06    682 1.5361e-04 0.27603 0.017957
## 673   1.5052e-06    683 1.5210e-04 0.27605 0.017957
## 674   1.4892e-06    684 1.5059e-04 0.27605 0.017957
## 675   1.4874e-06    685 1.4910e-04 0.27605 0.017957
## 676   1.4793e-06    686 1.4762e-04 0.27605 0.017957
## 677   1.4714e-06    687 1.4614e-04 0.27606 0.017957
## 678   1.4695e-06    688 1.4467e-04 0.27604 0.017956
## 679   1.4579e-06    689 1.4320e-04 0.27604 0.017956
## 680   1.4532e-06    690 1.4174e-04 0.27604 0.017956
## 681   1.4431e-06    691 1.4028e-04 0.27603 0.017957
## 682   1.4385e-06    692 1.3884e-04 0.27603 0.017957
## 683   1.4370e-06    693 1.3740e-04 0.27603 0.017957
## 684   1.4252e-06    694 1.3597e-04 0.27603 0.017957
## 685   1.4187e-06    695 1.3454e-04 0.27602 0.017957
## 686   1.4032e-06    696 1.3312e-04 0.27597 0.017956
## 687   1.3804e-06    697 1.3172e-04 0.27601 0.017956
## 688   1.3718e-06    698 1.3034e-04 0.27601 0.017956
## 689   1.3644e-06    699 1.2897e-04 0.27599 0.017956
## 690   1.3205e-06    700 1.2760e-04 0.27599 0.017956
## 691   1.2937e-06    701 1.2628e-04 0.27601 0.017957
## 692   1.2937e-06    702 1.2499e-04 0.27608 0.017963
## 693   1.2845e-06    703 1.2369e-04 0.27608 0.017963
## 694   1.2773e-06    704 1.2241e-04 0.27613 0.017964
## 695   1.2697e-06    705 1.2113e-04 0.27608 0.017963
## 696   1.2667e-06    706 1.1986e-04 0.27608 0.017963
## 697   1.2666e-06    707 1.1860e-04 0.27608 0.017963
```

```
## 698   1.2558e-06    708 1.1733e-04 0.27608 0.017963
## 699   1.2557e-06    709 1.1607e-04 0.27611 0.017964
## 700   1.2533e-06    710 1.1482e-04 0.27611 0.017964
## 701   1.2318e-06    711 1.1357e-04 0.27613 0.017964
## 702   1.2272e-06    712 1.1233e-04 0.27609 0.017963
## 703   1.2149e-06    713 1.1111e-04 0.27609 0.017963
## 704   1.2041e-06    714 1.0989e-04 0.27616 0.017969
## 705   1.1818e-06    715 1.0869e-04 0.27616 0.017969
## 706   1.1763e-06    716 1.0751e-04 0.27610 0.017969
## 707   1.1727e-06    717 1.0633e-04 0.27612 0.017969
## 708   1.1696e-06    718 1.0516e-04 0.27609 0.017969
## 709   1.1663e-06    719 1.0399e-04 0.27609 0.017969
## 710   1.1453e-06    720 1.0282e-04 0.27611 0.017971
## 711   1.1314e-06    721 1.0168e-04 0.27613 0.017971
## 712   1.1095e-06    722 1.0054e-04 0.27598 0.017962
## 713   1.0980e-06    723 9.9434e-05 0.27598 0.017955
## 714   1.0775e-06    724 9.8336e-05 0.27598 0.017955
## 715   1.0762e-06    725 9.7259e-05 0.27596 0.017956
## 716   1.0636e-06    726 9.6182e-05 0.27597 0.017955
## 717   1.0568e-06    727 9.5119e-05 0.27593 0.017955
## 718   1.0562e-06    728 9.4062e-05 0.27594 0.017955
## 719   1.0546e-06    729 9.3006e-05 0.27595 0.017955
## 720   1.0536e-06    730 9.1951e-05 0.27595 0.017955
## 721   1.0454e-06    731 9.0897e-05 0.27595 0.017955
## 722   1.0409e-06    732 8.9852e-05 0.27594 0.017955
## 723   1.0078e-06    733 8.8811e-05 0.27596 0.017955
## 724   1.0003e-06    734 8.7803e-05 0.27596 0.017955
## 725   9.9842e-07    735 8.6803e-05 0.27596 0.017955
## 726   9.9839e-07    736 8.5805e-05 0.27596 0.017955
## 727   9.9308e-07    737 8.4806e-05 0.27596 0.017955
## 728   9.7336e-07    738 8.3813e-05 0.27597 0.017955
## 729   9.6209e-07    739 8.2840e-05 0.27600 0.017956
## 730   9.6209e-07    740 8.1878e-05 0.27599 0.017956
## 731   9.4605e-07    741 8.0916e-05 0.27599 0.017956
## 732   9.2932e-07    742 7.9969e-05 0.27598 0.017956
## 733   9.2773e-07    743 7.9040e-05 0.27597 0.017956
## 734   9.2200e-07    744 7.8112e-05 0.27589 0.017952
## 735   9.0751e-07    745 7.7190e-05 0.27591 0.017952
## 736   9.0160e-07    746 7.6283e-05 0.27591 0.017952
## 737   9.0050e-07    747 7.5381e-05 0.27589 0.017952
## 738   8.8798e-07    748 7.4481e-05 0.27589 0.017952
## 739   8.7954e-07    749 7.3593e-05 0.27587 0.017952
## 740   8.6783e-07    750 7.2713e-05 0.27584 0.017951
## 741   8.4462e-07    751 7.1845e-05 0.27578 0.017947
## 742   8.4154e-07    752 7.1001e-05 0.27580 0.017947
## 743   8.4076e-07    753 7.0159e-05 0.27580 0.017947
## 744   8.3625e-07    754 6.9319e-05 0.27580 0.017947
## 745   8.3502e-07    755 6.8482e-05 0.27583 0.017949
## 746   8.2848e-07    756 6.7647e-05 0.27583 0.017949
## 747   8.2546e-07    757 6.6819e-05 0.27583 0.017949
## 748   8.2070e-07    758 6.5993e-05 0.27582 0.017949
## 749   7.9949e-07    759 6.5173e-05 0.27575 0.017948
## 750   7.8072e-07    760 6.4373e-05 0.27574 0.017948
## 751   7.7799e-07    761 6.3592e-05 0.27573 0.017949
```

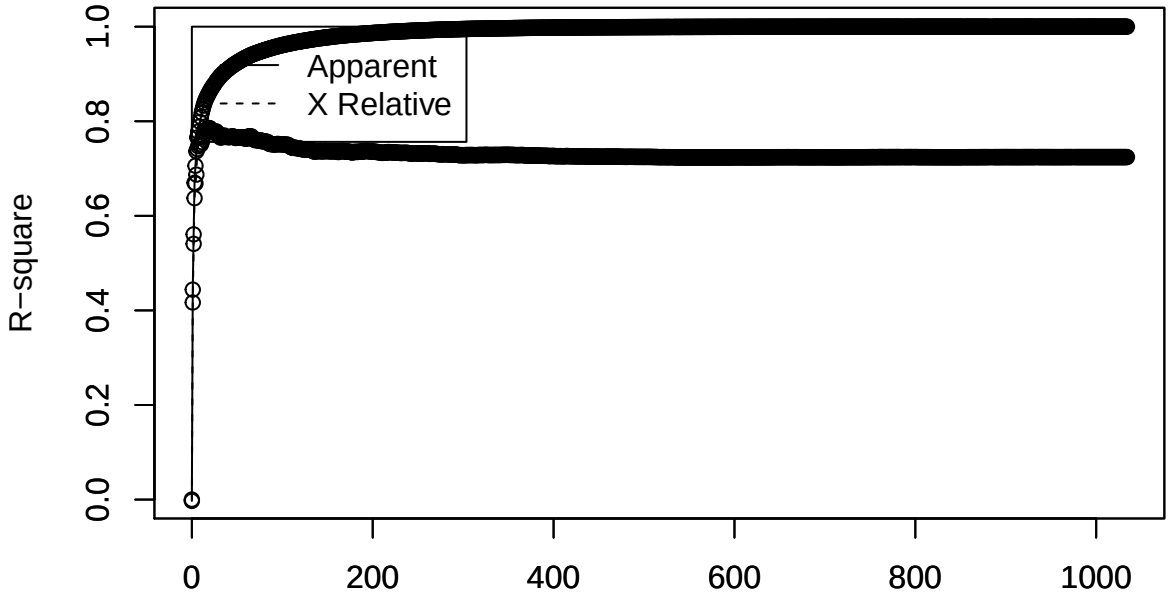
```
## 752   7.7108e-07    762 6.2814e-05 0.27574 0.017948
## 753   7.6765e-07    763 6.2043e-05 0.27575 0.017949
## 754   7.6260e-07    764 6.1276e-05 0.27575 0.017949
## 755   7.6048e-07    765 6.0513e-05 0.27575 0.017949
## 756   7.5478e-07    766 5.9753e-05 0.27576 0.017948
## 757   7.5197e-07    767 5.8998e-05 0.27577 0.017948
## 758   7.4340e-07    768 5.8246e-05 0.27577 0.017948
## 759   7.4304e-07    769 5.7502e-05 0.27577 0.017948
## 760   7.2573e-07    770 5.6759e-05 0.27574 0.017948
## 761   7.1999e-07    771 5.6034e-05 0.27573 0.017948
## 762   7.1431e-07    772 5.5314e-05 0.27573 0.017948
## 763   6.9404e-07    773 5.4599e-05 0.27575 0.017948
## 764   6.8689e-07    774 5.3905e-05 0.27576 0.017949
## 765   6.7907e-07    775 5.3218e-05 0.27574 0.017949
## 766   6.7651e-07    776 5.2539e-05 0.27575 0.017949
## 767   6.6703e-07    777 5.1863e-05 0.27575 0.017949
## 768   6.6607e-07    778 5.1196e-05 0.27576 0.017949
## 769   6.5817e-07    779 5.0530e-05 0.27576 0.017949
## 770   6.5677e-07    780 4.9872e-05 0.27575 0.017949
## 771   6.5358e-07    781 4.9215e-05 0.27575 0.017949
## 772   6.4553e-07    782 4.8561e-05 0.27575 0.017949
## 773   6.3608e-07    783 4.7916e-05 0.27575 0.017948
## 774   6.2921e-07    784 4.7280e-05 0.27574 0.017948
## 775   6.2272e-07    785 4.6650e-05 0.27580 0.017956
## 776   6.1799e-07    787 4.5405e-05 0.27582 0.017956
## 777   5.9973e-07    788 4.4787e-05 0.27576 0.017954
## 778   5.9644e-07    789 4.4187e-05 0.27571 0.017953
## 779   5.9347e-07    790 4.3591e-05 0.27573 0.017953
## 780   5.8318e-07    791 4.2997e-05 0.27577 0.017954
## 781   5.7494e-07    792 4.2414e-05 0.27575 0.017954
## 782   5.7494e-07    793 4.1839e-05 0.27574 0.017954
## 783   5.7421e-07    794 4.1264e-05 0.27574 0.017954
## 784   5.6107e-07    795 4.0690e-05 0.27574 0.017954
## 785   5.5931e-07    796 4.0129e-05 0.27575 0.017954
## 786   5.5123e-07    797 3.9570e-05 0.27573 0.017954
## 787   5.4727e-07    799 3.8467e-05 0.27574 0.017954
## 788   5.4365e-07    800 3.7920e-05 0.27575 0.017954
## 789   5.4288e-07    801 3.7376e-05 0.27575 0.017954
## 790   5.3624e-07    802 3.6833e-05 0.27579 0.017955
## 791   5.3623e-07    803 3.6297e-05 0.27579 0.017955
## 792   5.0802e-07    804 3.5761e-05 0.27584 0.017956
## 793   5.0741e-07    805 3.5253e-05 0.27585 0.017956
## 794   4.9711e-07    806 3.4746e-05 0.27585 0.017956
## 795   4.9698e-07    807 3.4248e-05 0.27585 0.017956
## 796   4.9649e-07    808 3.3751e-05 0.27585 0.017956
## 797   4.9099e-07    809 3.3255e-05 0.27584 0.017956
## 798   4.9085e-07    810 3.2764e-05 0.27584 0.017956
## 799   4.8617e-07    811 3.2273e-05 0.27584 0.017956
## 800   4.8197e-07    812 3.1787e-05 0.27584 0.017956
## 801   4.7090e-07    813 3.1305e-05 0.27583 0.017956
## 802   4.5264e-07    814 3.0834e-05 0.27585 0.017956
## 803   4.4836e-07    815 3.0381e-05 0.27585 0.017956
## 804   4.3767e-07    816 2.9933e-05 0.27587 0.017956
## 805   4.3603e-07    817 2.9495e-05 0.27587 0.017956
```

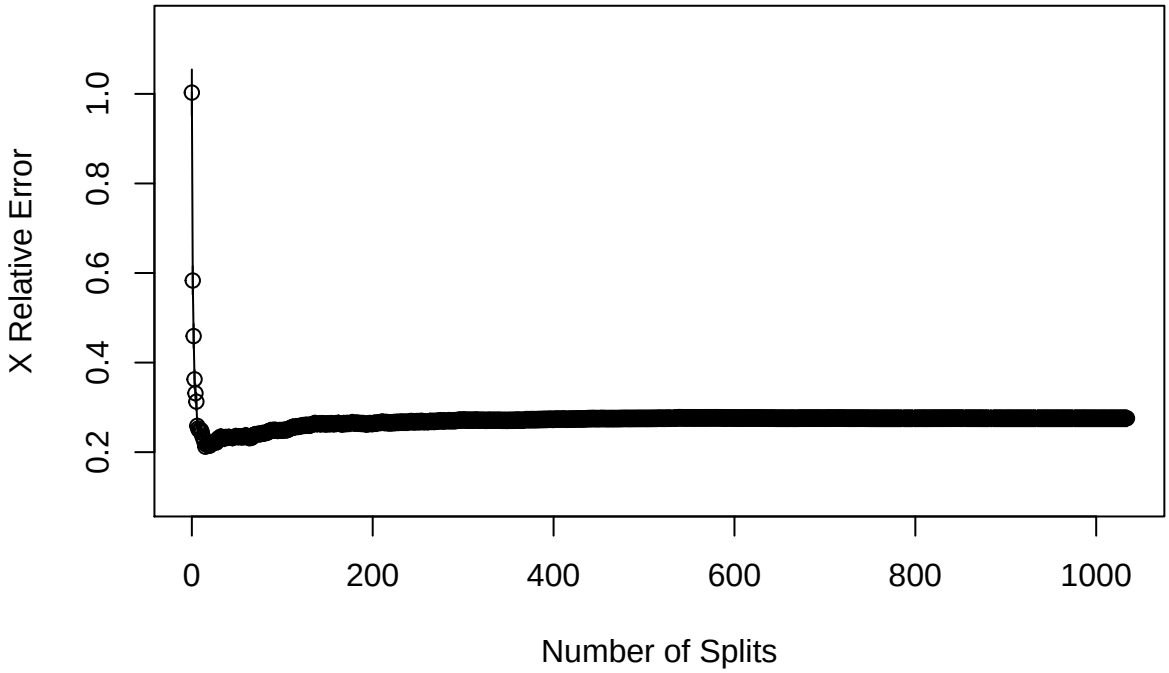
```
## 806   4.3010e-07    818 2.9059e-05 0.27588 0.017956
## 807   4.2856e-07    819 2.8629e-05 0.27587 0.017956
## 808   4.2326e-07    820 2.8201e-05 0.27591 0.017956
## 809   4.2267e-07    821 2.7777e-05 0.27593 0.017957
## 810   4.2112e-07    822 2.7355e-05 0.27593 0.017957
## 811   4.1985e-07    823 2.6934e-05 0.27593 0.017957
## 812   4.1199e-07    824 2.6514e-05 0.27593 0.017957
## 813   4.0362e-07    825 2.6102e-05 0.27593 0.017957
## 814   3.9726e-07    826 2.5698e-05 0.27593 0.017957
## 815   3.9396e-07    827 2.5301e-05 0.27593 0.017957
## 816   3.9166e-07    828 2.4907e-05 0.27594 0.017956
## 817   3.9090e-07    829 2.4515e-05 0.27594 0.017956
## 818   3.8850e-07    830 2.4124e-05 0.27594 0.017956
## 819   3.8191e-07    831 2.3736e-05 0.27594 0.017957
## 820   3.7826e-07    832 2.3354e-05 0.27594 0.017957
## 821   3.6423e-07    833 2.2976e-05 0.27597 0.017957
## 822   3.6382e-07    834 2.2612e-05 0.27595 0.017956
## 823   3.5581e-07    835 2.2248e-05 0.27597 0.017956
## 824   3.5573e-07    836 2.1892e-05 0.27597 0.017956
## 825   3.5507e-07    837 2.1536e-05 0.27597 0.017956
## 826   3.4890e-07    838 2.1181e-05 0.27595 0.017956
## 827   3.2892e-07    839 2.0832e-05 0.27596 0.017956
## 828   3.2835e-07    840 2.0503e-05 0.27599 0.017957
## 829   3.2835e-07    841 2.0175e-05 0.27599 0.017957
## 830   3.2605e-07    842 1.9847e-05 0.27599 0.017957
## 831   3.2131e-07    843 1.9521e-05 0.27598 0.017957
## 832   3.1820e-07    844 1.9199e-05 0.27600 0.017957
## 833   3.1420e-07    845 1.8881e-05 0.27600 0.017957
## 834   3.0296e-07    846 1.8567e-05 0.27599 0.017957
## 835   2.9452e-07    847 1.8264e-05 0.27599 0.017957
## 836   2.8879e-07    848 1.7969e-05 0.27597 0.017957
## 837   2.8366e-07    849 1.7681e-05 0.27595 0.017956
## 838   2.8183e-07    850 1.7397e-05 0.27594 0.017956
## 839   2.8105e-07    851 1.7115e-05 0.27594 0.017956
## 840   2.7556e-07    852 1.6834e-05 0.27596 0.017956
## 841   2.7023e-07    853 1.6558e-05 0.27596 0.017957
## 842   2.6755e-07    855 1.6018e-05 0.27593 0.017956
## 843   2.5674e-07    856 1.5750e-05 0.27592 0.017956
## 844   2.5633e-07    857 1.5494e-05 0.27591 0.017956
## 845   2.4581e-07    858 1.5237e-05 0.27587 0.017956
## 846   2.3976e-07    859 1.4992e-05 0.27587 0.017955
## 847   2.3789e-07    860 1.4752e-05 0.27586 0.017955
## 848   2.3584e-07    861 1.4514e-05 0.27586 0.017955
## 849   2.3403e-07    862 1.4278e-05 0.27586 0.017955
## 850   2.3403e-07    863 1.4044e-05 0.27586 0.017955
## 851   2.3262e-07    864 1.3810e-05 0.27586 0.017955
## 852   2.2893e-07    865 1.3577e-05 0.27586 0.017955
## 853   2.2578e-07    866 1.3348e-05 0.27586 0.017955
## 854   2.2372e-07    867 1.3123e-05 0.27585 0.017955
## 855   2.1774e-07    868 1.2899e-05 0.27585 0.017955
## 856   2.1507e-07    869 1.2681e-05 0.27586 0.017955
## 857   2.1420e-07    870 1.2466e-05 0.27586 0.017955
## 858   2.0698e-07    871 1.2252e-05 0.27586 0.017955
## 859   2.0350e-07    872 1.2045e-05 0.27583 0.017951
```

```
## 860   2.0010e-07    873 1.1841e-05 0.27580 0.017950
## 861   1.9288e-07    874 1.1641e-05 0.27581 0.017950
## 862   1.8634e-07    875 1.1449e-05 0.27579 0.017950
## 863   1.8435e-07    876 1.1262e-05 0.27579 0.017950
## 864   1.7656e-07    877 1.1078e-05 0.27578 0.017950
## 865   1.7578e-07    878 1.0901e-05 0.27578 0.017950
## 866   1.7539e-07    879 1.0725e-05 0.27578 0.017950
## 867   1.7517e-07    880 1.0550e-05 0.27578 0.017950
## 868   1.7305e-07    881 1.0375e-05 0.27578 0.017950
## 869   1.7172e-07    883 1.0029e-05 0.27576 0.017950
## 870   1.6400e-07    884 9.8571e-06 0.27577 0.017949
## 871   1.6276e-07    886 9.5291e-06 0.27577 0.017949
## 872   1.6244e-07    887 9.3663e-06 0.27581 0.017951
## 873   1.6208e-07    888 9.2039e-06 0.27581 0.017951
## 874   1.5996e-07    889 9.0418e-06 0.27579 0.017951
## 875   1.5757e-07    890 8.8818e-06 0.27579 0.017951
## 876   1.5564e-07    891 8.7243e-06 0.27578 0.017951
## 877   1.5564e-07    892 8.5686e-06 0.27579 0.017951
## 878   1.5336e-07    893 8.4130e-06 0.27579 0.017951
## 879   1.5303e-07    894 8.2596e-06 0.27577 0.017951
## 880   1.5088e-07    895 8.1066e-06 0.27576 0.017951
## 881   1.4976e-07    896 7.9557e-06 0.27576 0.017951
## 882   1.4790e-07    897 7.8060e-06 0.27576 0.017951
## 883   1.4729e-07    898 7.6581e-06 0.27576 0.017951
## 884   1.4684e-07    899 7.5108e-06 0.27576 0.017951
## 885   1.4494e-07    900 7.3639e-06 0.27576 0.017951
## 886   1.4476e-07    901 7.2190e-06 0.27575 0.017951
## 887   1.4171e-07    902 7.0742e-06 0.27576 0.017951
## 888   1.3845e-07    903 6.9325e-06 0.27576 0.017951
## 889   1.3812e-07    904 6.7941e-06 0.27576 0.017951
## 890   1.3744e-07    905 6.6559e-06 0.27576 0.017951
## 891   1.3734e-07    906 6.5185e-06 0.27576 0.017951
## 892   1.3685e-07    907 6.3812e-06 0.27576 0.017951
## 893   1.3257e-07    908 6.2443e-06 0.27576 0.017951
## 894   1.3145e-07    909 6.1117e-06 0.27575 0.017951
## 895   1.3059e-07    910 5.9803e-06 0.27576 0.017951
## 896   1.2785e-07    911 5.8497e-06 0.27577 0.017951
## 897   1.2751e-07    912 5.7218e-06 0.27577 0.017951
## 898   1.2718e-07    913 5.5943e-06 0.27577 0.017951
## 899   1.1843e-07    914 5.4672e-06 0.27576 0.017951
## 900   1.1838e-07    915 5.3487e-06 0.27576 0.017951
## 901   1.1830e-07    916 5.2303e-06 0.27576 0.017951
## 902   1.1694e-07    917 5.1120e-06 0.27576 0.017951
## 903   1.1667e-07    918 4.9951e-06 0.27576 0.017951
## 904   1.1336e-07    919 4.8784e-06 0.27576 0.017951
## 905   1.1248e-07    920 4.7651e-06 0.27575 0.017951
## 906   1.1232e-07    921 4.6526e-06 0.27575 0.017951
## 907   1.1141e-07    922 4.5403e-06 0.27575 0.017951
## 908   1.0956e-07    923 4.4289e-06 0.27575 0.017951
## 909   9.7533e-08    924 4.3193e-06 0.27576 0.017951
## 910   9.4979e-08    925 4.2218e-06 0.27574 0.017951
## 911   9.3841e-08    926 4.1268e-06 0.27574 0.017951
## 912   9.2967e-08    927 4.0329e-06 0.27574 0.017951
## 913   9.2776e-08    928 3.9400e-06 0.27574 0.017951
```

```
## 914   9.1039e-08    929 3.8472e-06 0.27574 0.017951
## 915   8.9684e-08    930 3.7562e-06 0.27574 0.017951
## 916   8.8686e-08    931 3.6665e-06 0.27575 0.017951
## 917   8.5795e-08    932 3.5778e-06 0.27576 0.017951
## 918   8.4708e-08    933 3.4920e-06 0.27576 0.017951
## 919   8.3945e-08    935 3.3226e-06 0.27576 0.017951
## 920   8.3586e-08    936 3.2386e-06 0.27576 0.017951
## 921   8.2220e-08    937 3.1550e-06 0.27576 0.017951
## 922   8.1280e-08    938 3.0728e-06 0.27575 0.017951
## 923   8.0421e-08    939 2.9915e-06 0.27575 0.017951
## 924   7.9574e-08    940 2.9111e-06 0.27575 0.017951
## 925   7.8816e-08    941 2.8316e-06 0.27575 0.017951
## 926   7.8741e-08    942 2.7527e-06 0.27575 0.017951
## 927   7.6317e-08    943 2.6740e-06 0.27575 0.017951
## 928   7.3898e-08    944 2.5977e-06 0.27575 0.017951
## 929   6.2689e-08    945 2.5238e-06 0.27575 0.017951
## 930   6.2209e-08    946 2.4611e-06 0.27576 0.017951
## 931   6.2106e-08    947 2.3989e-06 0.27576 0.017951
## 932   6.0847e-08    948 2.3368e-06 0.27576 0.017951
## 933   6.0405e-08    949 2.2759e-06 0.27576 0.017951
## 934   5.8823e-08    950 2.2155e-06 0.27576 0.017951
## 935   5.7948e-08    951 2.1567e-06 0.27576 0.017951
## 936   5.6697e-08    952 2.0988e-06 0.27576 0.017951
## 937   5.6195e-08    953 2.0421e-06 0.27576 0.017951
## 938   5.5373e-08    954 1.9859e-06 0.27576 0.017951
## 939   5.4855e-08    955 1.9305e-06 0.27576 0.017951
## 940   5.4729e-08    956 1.8756e-06 0.27576 0.017951
## 941   5.4253e-08    957 1.8209e-06 0.27576 0.017951
## 942   5.2711e-08    958 1.7667e-06 0.27576 0.017951
## 943   5.2242e-08    959 1.7139e-06 0.27576 0.017951
## 944   5.2070e-08    960 1.6617e-06 0.27576 0.017951
## 945   5.1763e-08    961 1.6096e-06 0.27576 0.017951
## 946   5.1089e-08    962 1.5579e-06 0.27576 0.017951
## 947   4.9817e-08    963 1.5068e-06 0.27576 0.017951
## 948   4.6962e-08    964 1.4570e-06 0.27577 0.017951
## 949   4.5937e-08    965 1.4100e-06 0.27575 0.017949
## 950   4.5903e-08    966 1.3641e-06 0.27575 0.017949
## 951   4.5538e-08    967 1.3182e-06 0.27575 0.017949
## 952   4.5476e-08    968 1.2726e-06 0.27575 0.017949
## 953   4.4229e-08    969 1.2271e-06 0.27575 0.017949
## 954   4.4118e-08    970 1.1829e-06 0.27574 0.017949
## 955   4.3096e-08    971 1.1388e-06 0.27574 0.017949
## 956   4.2799e-08    972 1.0957e-06 0.27575 0.017950
## 957   4.2763e-08    973 1.0529e-06 0.27575 0.017950
## 958   4.0845e-08    974 1.0101e-06 0.27575 0.017950
## 959   3.9053e-08    976 9.2845e-07 0.27575 0.017950
## 960   3.8482e-08    977 8.8940e-07 0.27575 0.017950
## 961   3.7648e-08    978 8.5091e-07 0.27575 0.017950
## 962   3.7376e-08    979 8.1327e-07 0.27575 0.017950
## 963   3.6361e-08    980 7.7589e-07 0.27575 0.017950
## 964   3.6318e-08    981 7.3953e-07 0.27575 0.017950
## 965   3.5484e-08    982 7.0321e-07 0.27575 0.017950
## 966   3.4571e-08    983 6.6773e-07 0.27575 0.017950
## 967   3.4491e-08    984 6.3316e-07 0.27576 0.017950
```

```
## 968   3.1682e-08    985 5.9866e-07 0.27575 0.017950
## 969   3.1435e-08    986 5.6698e-07 0.27575 0.017950
## 970   3.0954e-08    987 5.3555e-07 0.27575 0.017950
## 971   2.8994e-08    988 5.0459e-07 0.27575 0.017950
## 972   2.8890e-08    989 4.7560e-07 0.27575 0.017950
## 973   2.8526e-08    990 4.4671e-07 0.27576 0.017950
## 974   2.6970e-08    991 4.1818e-07 0.27576 0.017950
## 975   2.3800e-08    992 3.9121e-07 0.27575 0.017950
## 976   2.3260e-08    993 3.6741e-07 0.27575 0.017950
## 977   2.2997e-08    994 3.4415e-07 0.27575 0.017950
## 978   2.1158e-08    995 3.2116e-07 0.27575 0.017950
## 979   2.0402e-08    996 3.0000e-07 0.27574 0.017950
## 980   1.9328e-08    997 2.7960e-07 0.27573 0.017947
## 981   1.9228e-08    998 2.6027e-07 0.27573 0.017947
## 982   1.9063e-08    999 2.4104e-07 0.27573 0.017947
## 983   1.8835e-08   1000 2.2198e-07 0.27573 0.017947
## 984   1.8710e-08   1001 2.0314e-07 0.27573 0.017947
## 985   1.5490e-08   1002 1.8443e-07 0.27572 0.017947
## 986   1.5414e-08   1003 1.6894e-07 0.27572 0.017947
## 987   1.4794e-08   1004 1.5353e-07 0.27572 0.017947
## 988   1.4463e-08   1005 1.3873e-07 0.27572 0.017947
## 989   1.2480e-08   1006 1.2427e-07 0.27572 0.017947
## 990   1.2428e-08   1007 1.1179e-07 0.27572 0.017947
## 991   1.0589e-08   1008 9.9362e-08 0.27572 0.017947
## 992   1.0514e-08   1009 8.8773e-08 0.27572 0.017947
## 993   9.4635e-09   1010 7.8260e-08 0.27572 0.017947
## 994   9.4292e-09   1011 6.8796e-08 0.27572 0.017947
## 995   8.6277e-09   1012 5.9367e-08 0.27572 0.017947
## 996   7.6669e-09   1013 5.0739e-08 0.27572 0.017947
## 997   5.9244e-09   1014 4.3072e-08 0.27572 0.017947
## 998   5.8233e-09   1015 3.7148e-08 0.27572 0.017947
## 999   5.8233e-09   1016 3.1325e-08 0.27572 0.017947
## 1000  4.8823e-09   1017 2.5501e-08 0.27572 0.017947
## 1001  4.4189e-09   1018 2.0619e-08 0.27572 0.017947
## 1002  3.1516e-09   1019 1.6200e-08 0.27572 0.017947
## 1003  2.2707e-09   1020 1.3048e-08 0.27572 0.017947
## 1004  1.4831e-09   1021 1.0778e-08 0.27572 0.017947
## 1005  1.3435e-09   1022 9.2947e-09 0.27572 0.017947
## 1006  1.1902e-09   1023 7.9512e-09 0.27572 0.017947
## 1007  1.0923e-09   1024 6.7610e-09 0.27572 0.017947
## 1008  1.0502e-09   1025 5.6687e-09 0.27572 0.017947
## 1009  9.8934e-10   1026 4.6185e-09 0.27572 0.017947
## 1010  9.4436e-10   1027 3.6292e-09 0.27572 0.017947
## 1011  8.8244e-10   1028 2.6848e-09 0.27572 0.017947
## 1012  8.4730e-10   1029 1.8024e-09 0.27572 0.017947
## 1013  8.2462e-10   1030 9.5508e-10 0.27572 0.017947
## 1014  1.0983e-10   1031 1.3047e-10 0.27572 0.017947
## 1015  2.0640e-11   1032 2.0640e-11 0.27572 0.017947
## 1016  5.4101e-32   1033 5.4101e-32 0.27572 0.017947
## 1017 -1.0000e+00   1034 0.0000e+00 0.27572 0.017947
```






```
tree2Preds <- predict(tree2, treeFrame[-indexes,])
rmsle(expm1(targetValue[-indexes]), expm1(tree2Preds))
```

```
## [1] 0.2084194
```

```
#optimal tuning by internal cross validation
tree3 <- rpart(targetValue ~ ., treeFrame[indexes,],control = rpart.control(cp=2.0627e-03)
               , model = T)
tree3Preds <- predict(tree3, treeFrame[-indexes,])
rmsle(expm1(targetValue[-indexes]), expm1(tree3Preds))
```

```
## [1] 0.1876279
```

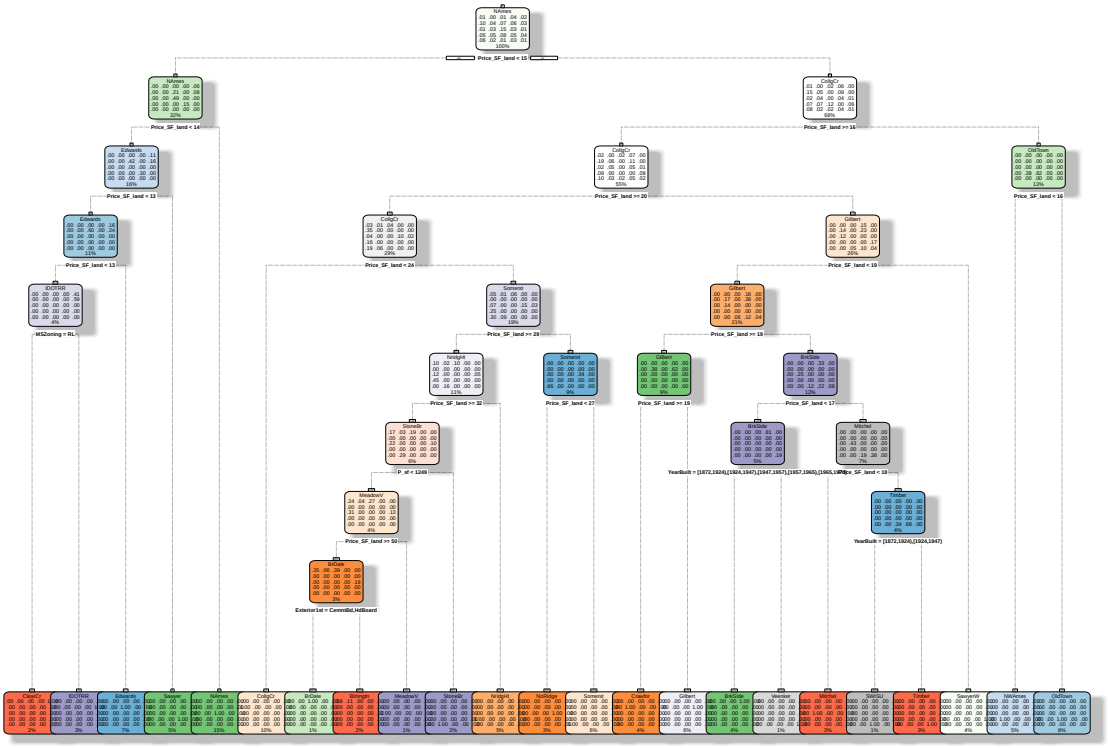
To highlight an issue of data leakage and multicollinearity a decision tree is used to try and classify neighborhood. First the classification rate is around 97%. However, there is perfect multicollinearity between the median price per square foot of land and neighborhood. Once this predictor is removed the classification rate for neighborhood drops to around 50%.

```
#can a tree predict neighborhood (this result is due to data leakage)
tree1class <- rpart(Neighborhood ~ ., treeFrame[indexes,], method = 'class')
tree1Preds <- predict(tree1class, treeFrame[-indexes,], type = 'class')
mean(tree1Preds == treeFrame$Neighborhood[-indexes])
```

```
## [1] 0.9890411
```

```
fancyRpartPlot(tree1class)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```

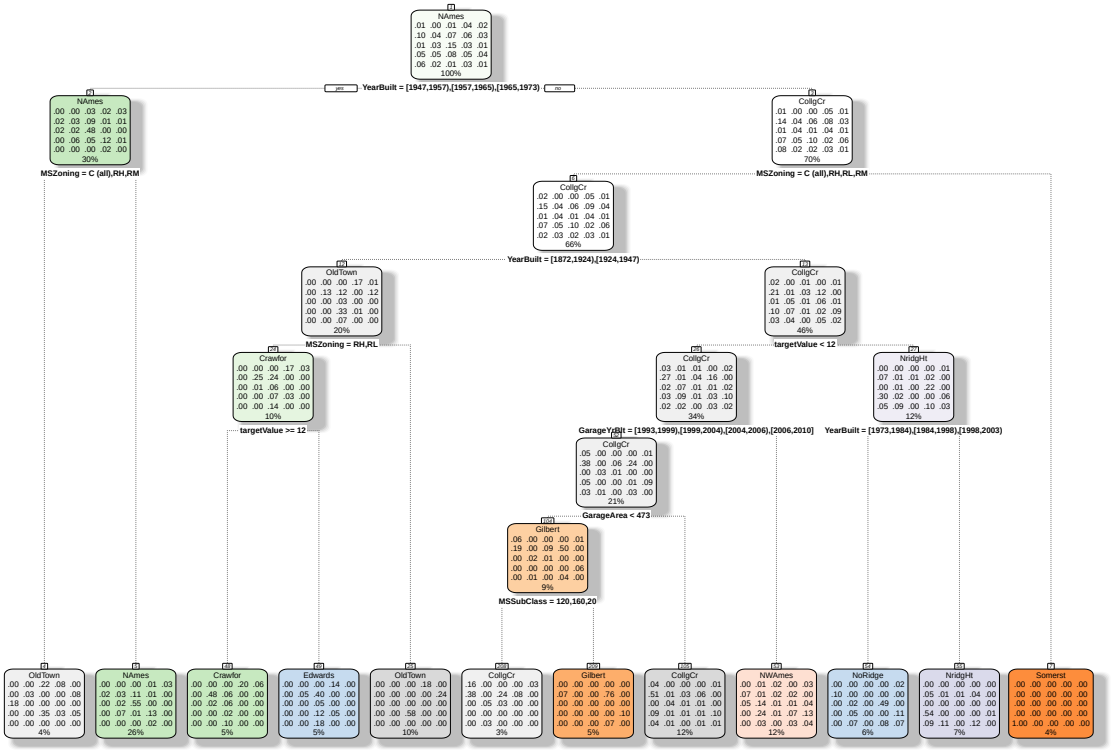


Rattle 2020−Sep−11 11:40:40 ronenreouveni

```
#remove data leaks and try again
tree1classNOLEAK <- rpart(Neighborhood ~ . - P_sf  - Price_SF_land - Price_SF_land_,
                          treeFrame[indexes,], method = 'class')
tree1Preds <- predict(tree1classNOLEAK, treeFrame[-indexes,], type = 'class')
mean(tree1Preds == treeFrame$Neighborhood[-indexes])
```

```
## [1] 0.4712329
```

```
fancyRpartPlot(tree1classNOLEAK)
```



Clustering

Clustering groups like observations. Two main clustering algorithms are used, kmeans and hierarchical.

The data is combined and transformed into a matrix. This matrix is used for the rest of the models. Columns of the matrix with over 99.94% zeros are removed. This is because there is such low variance of that predictor that it will only negatively impact the model. Before clustering, the data is also standardized to improve accuracy.

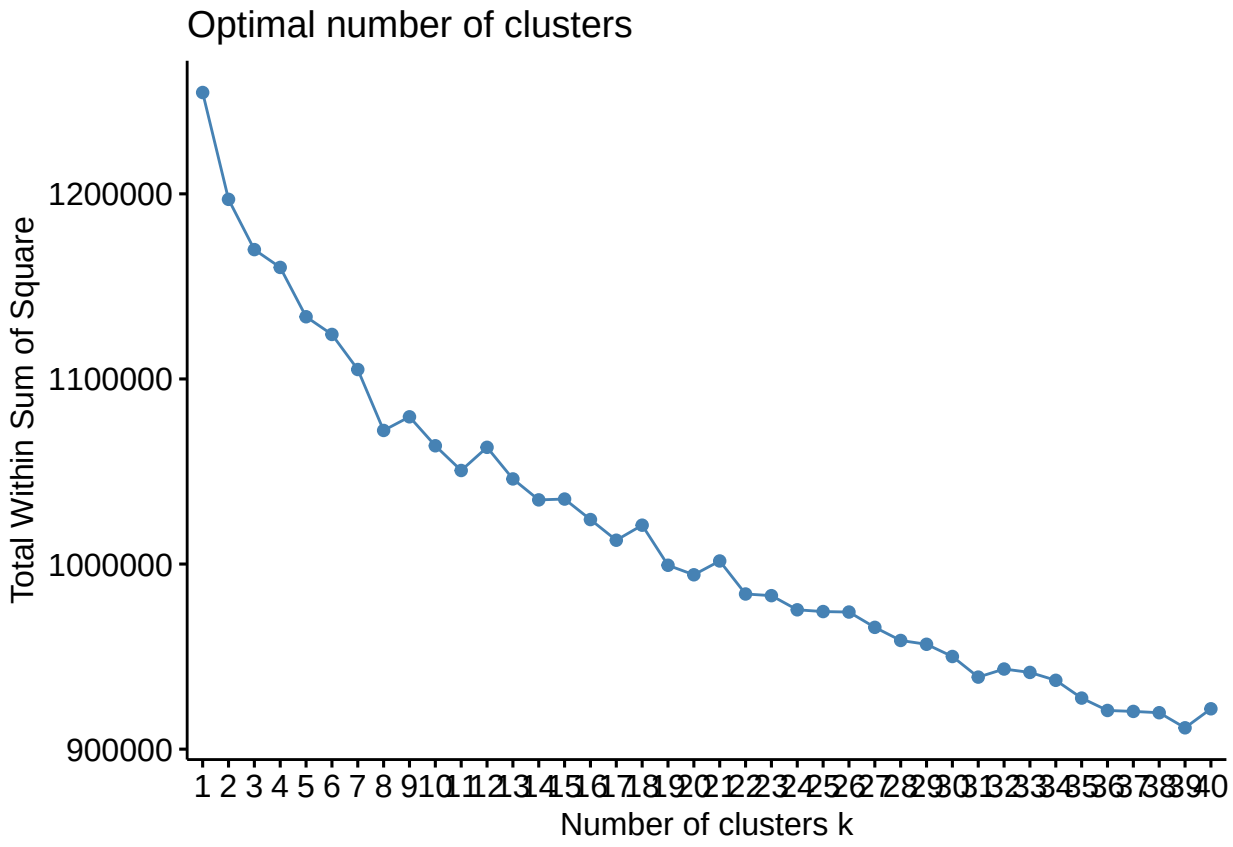
Kmeans clustering and hierarchical clustering are both used with varying distance measures and amount of clusters. The sale price is discretized into 5 bins. These labels are then written as the row names to then easily be able to see if they are grouped together by the clustering algorithm. While 5 labels is an over simplification of the problem compared to the goal of accurately predicting the price, it is beneficial to see how the clustering algorithm decides to group observations.

```
#turn data into matrix
final_matrix <- as.matrix(cbind(acm.disjonctif(fullFrame[,cats]), fullFrame[,ints]))
res <- colSums(final_matrix==0)/nrow(final_matrix)*100
final_matrix <- final_matrix[,-c(which(res>99.94))]
final_move <- final_matrix[(nrow(housingData)+1):(nrow(housingData_test)+nrow(housingData)),]



newRows <- sample(length(targetValue),length(targetValue))
targetValue <- targetValue[newRows]
final_matrix <- final_matrix[newRows,]
```

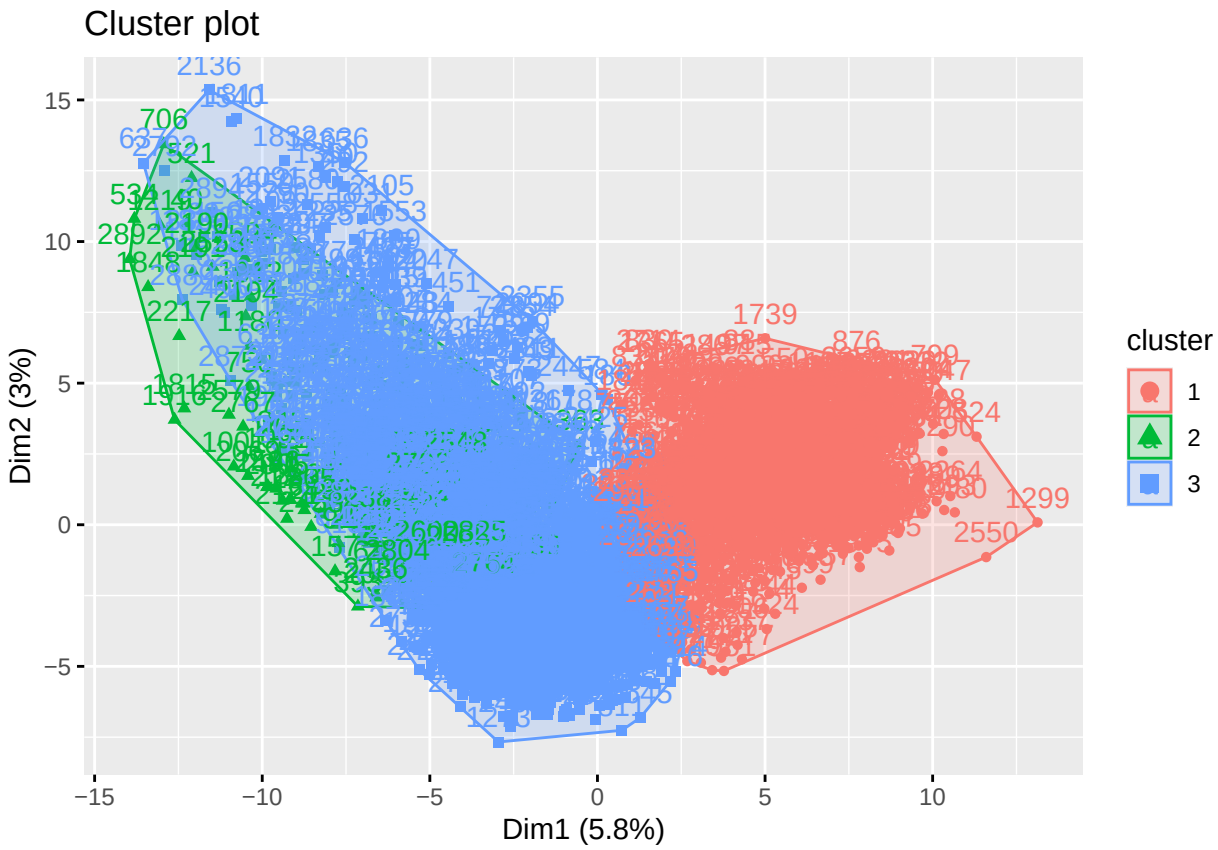
```
rf_matrix <- as.matrix(final_matrix)
rf_final_move <- as.matrix(final_move)


clusterMatrix <- rbind(rf_matrix,rf_final_move)

for (i in 1:ncol(clusterMatrix)){
  clusterMatrix[,i] <- standardize(clusterMatrix[,i])
}
```

For Kmeans the elbow method is used to select potential cluster amounts. K = 3, 8, and 15 are all used. Silhouettes are then used to measure the distance within each cluster. Although kmeans intrinsically uses euclidean distance, by analyzing the clusters it creates with different distance measures it is possible to ascertain which distance measure minimizes the distance between observations in a cluster. Cosine distance seems to be the best in this regard.

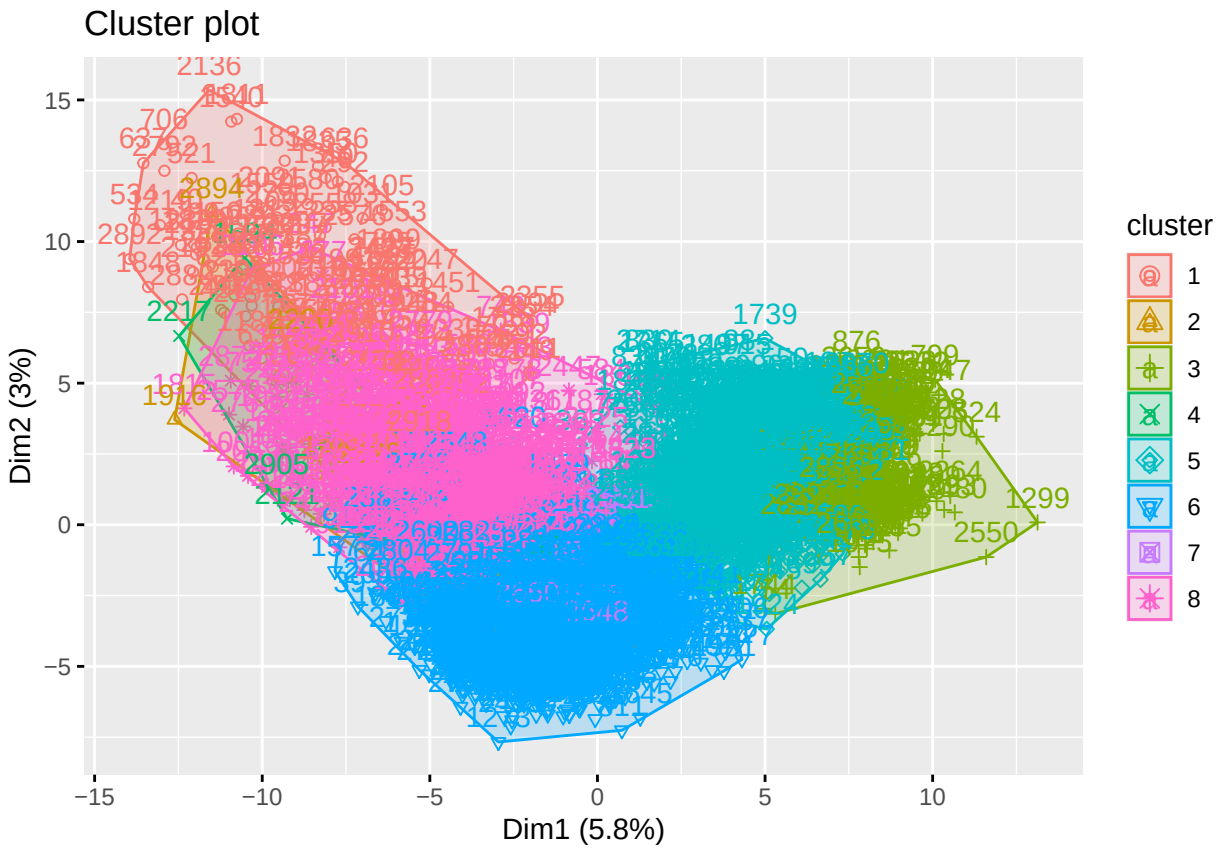
```
#elbow method to choOse clusters
fviz_nbclust(clusterMatrix, kmeans, method = "wss", k.max = 40)
```



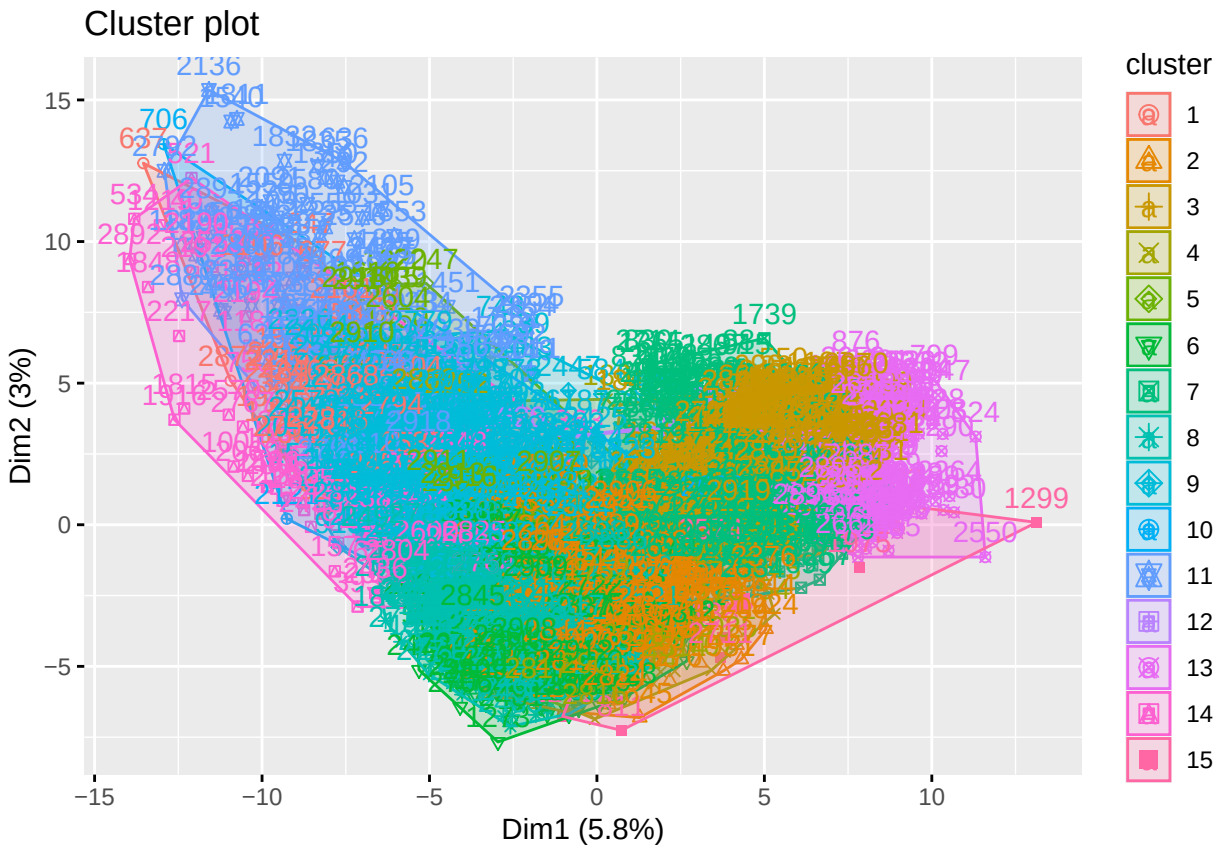

```
X3clust_3 <- kmeans(clusterMatrix,3)
fviz_cluster(X3clust_3, clusterMatrix)
```




```
X3clust_8 <- kmeans(clusterMatrix,8)
fviz_cluster(X3clust_8, clusterMatrix)
```

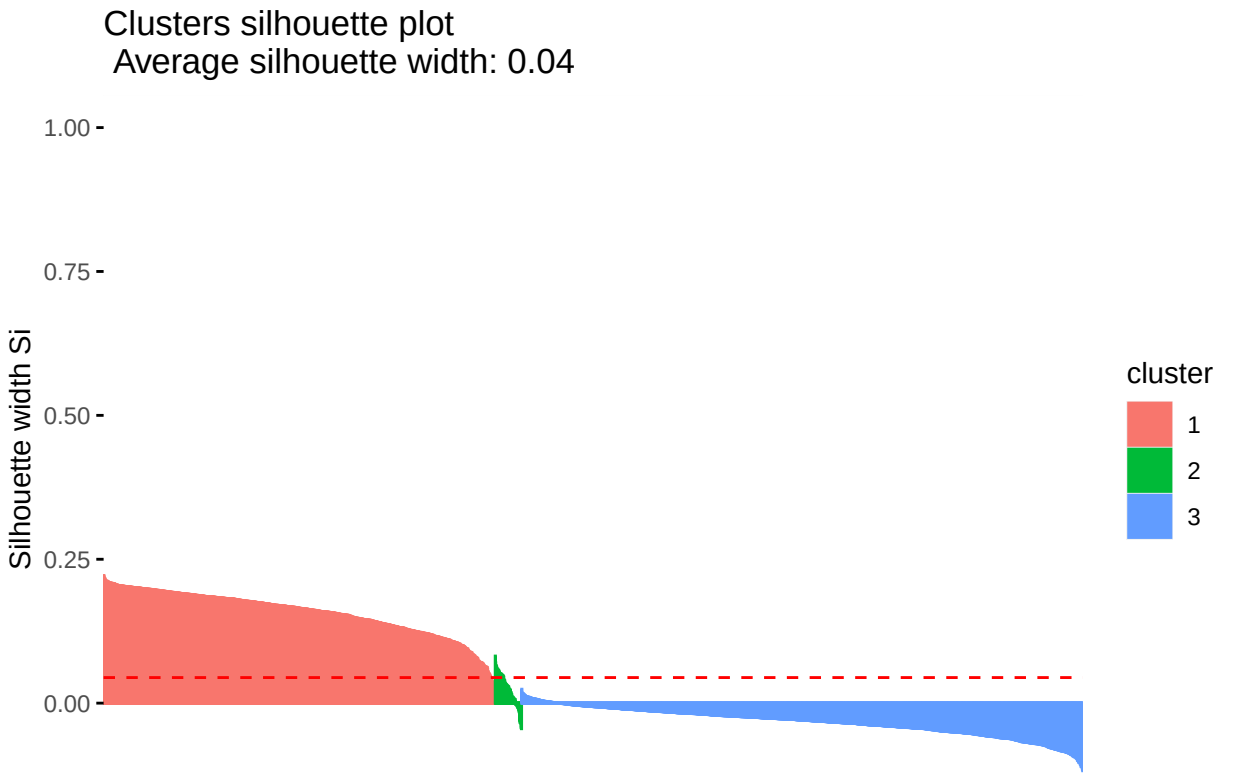



```
X3clust_15 <- kmeans(clusterMatrix,15)
fviz_cluster(X3clust_15, clusterMatrix)
```



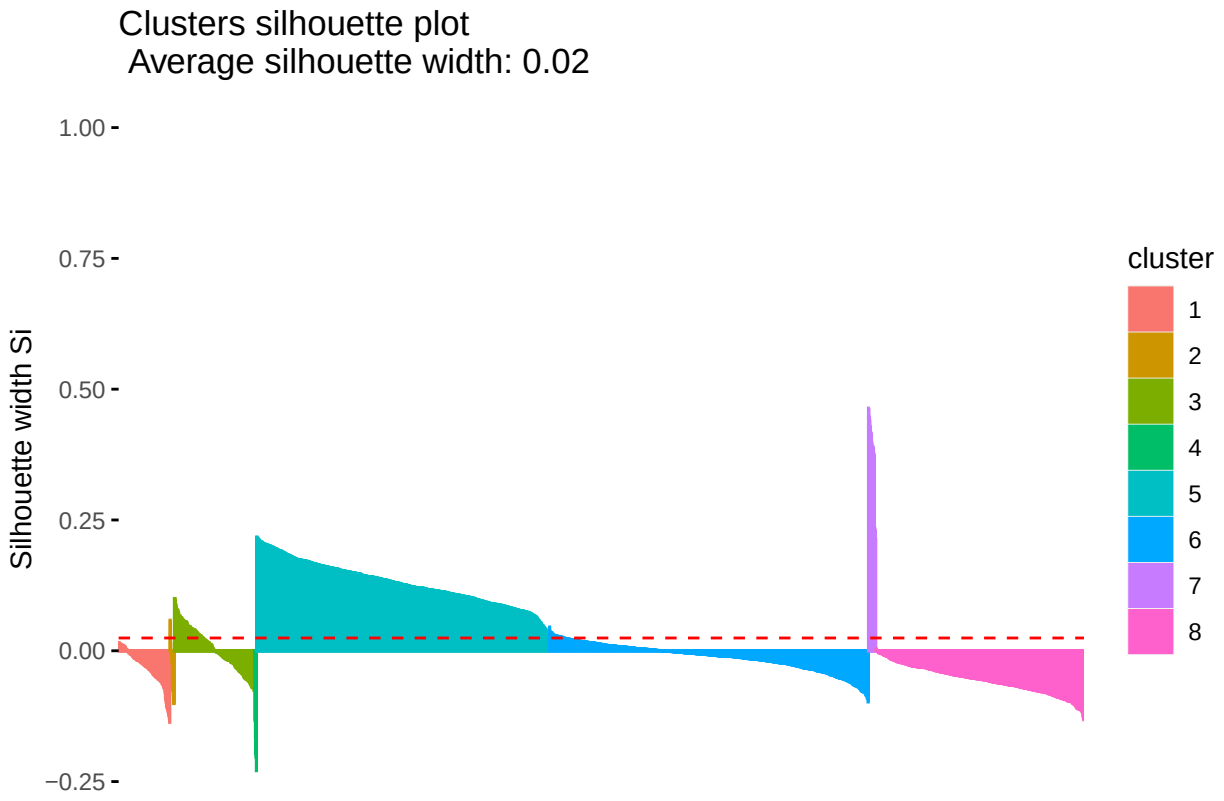

```
sil3 <- silhouette(X3clust_3$cluster, dist(clusterMatrix))
fviz_silhouette(sil3)
```

```
##   cluster size ave.sil.width
## 1       1 1167          0.16
## 2       2   79          0.02
## 3       3 1673         -0.03
```



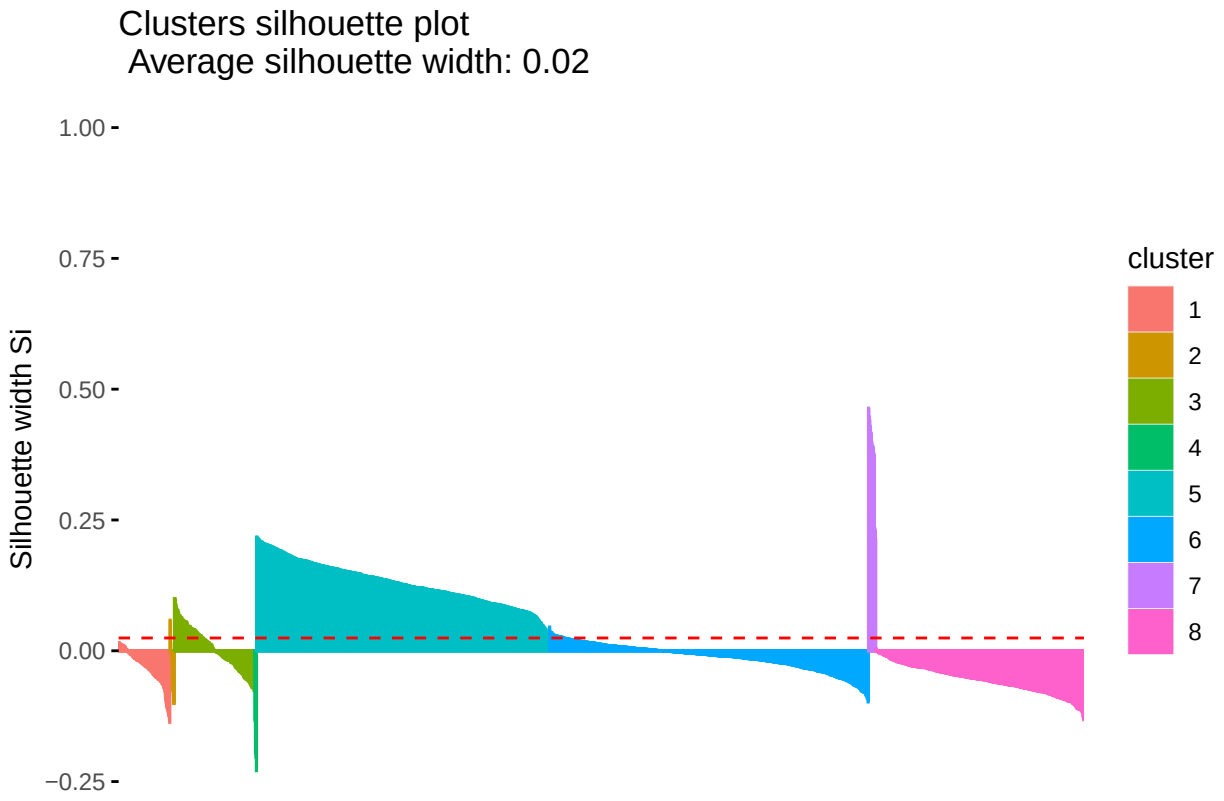

```
sil8 <- silhouette(X3clust_8$cluster, dist(clusterMatrix))
fviz_silhouette(sil8)
```

```
##   cluster size ave.sil.width
## 1       1  155         -0.03
## 2       2   13         -0.03
## 3       3  243          0.00
## 4       4    6         -0.16
## 5       5  886          0.13
## 6       6  964         -0.01
## 7       7   23          0.37
## 8       8  629         -0.06
```



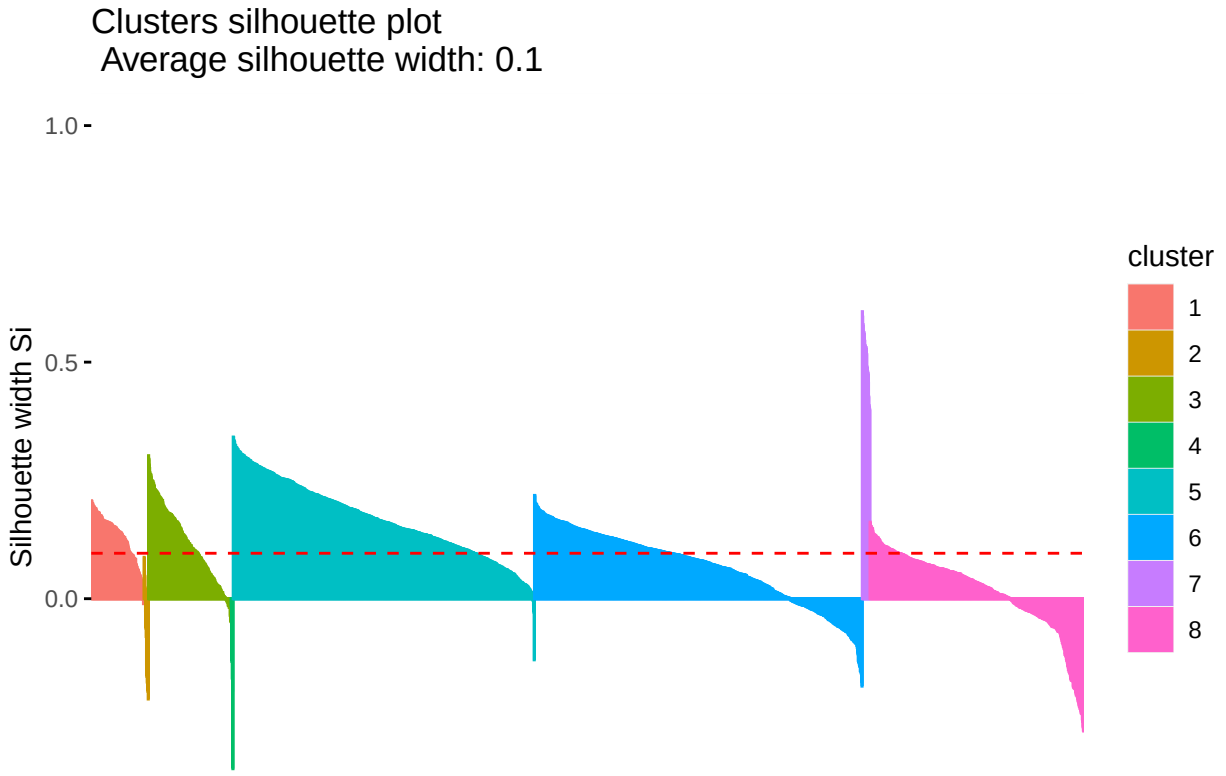

```
sil8_man <- silhouette(X3clust_8$cluster, dist.matrix(clusterMatrix, method = "euclidean"))
fviz_silhouette(sil8)
```

```
##   cluster size ave.sil.width
## 1       1  155         -0.03
## 2       2   13         -0.03
## 3       3  243          0.00
## 4       4    6         -0.16
## 5       5  886          0.13
## 6       6  964         -0.01
## 7       7   23          0.37
## 8       8  629         -0.06
```



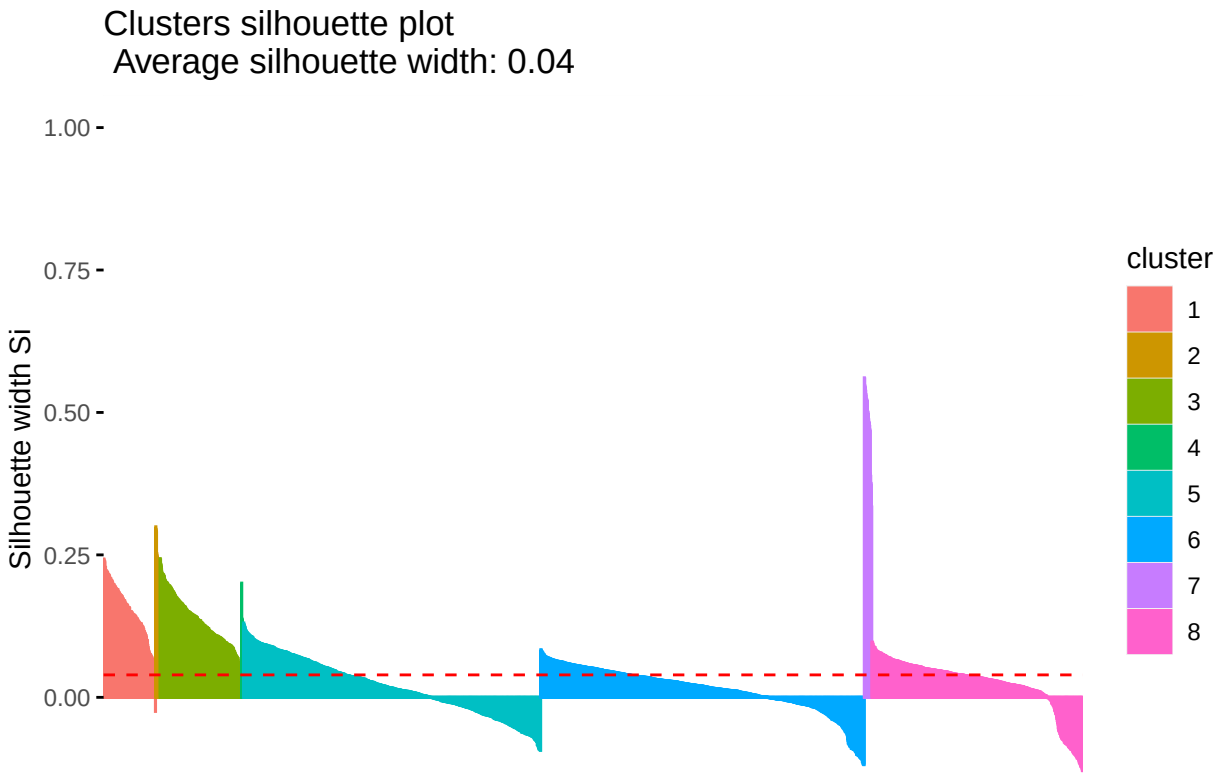

```
sil8_man <- silhouette(X3clust_8$cluster, dist.matrix(clusterMatrix, method = "manhattan"))
fviz_silhouette(sil8_man)
```

```
##   cluster size ave.sil.width
## 1       1  155          0.13
## 2       2   13         -0.08
## 3       3  243          0.12
## 4       4    6         -0.15
## 5       5  886          0.17
## 6       6  964          0.07
## 7       7   23          0.51
## 8       8  629          0.02
```



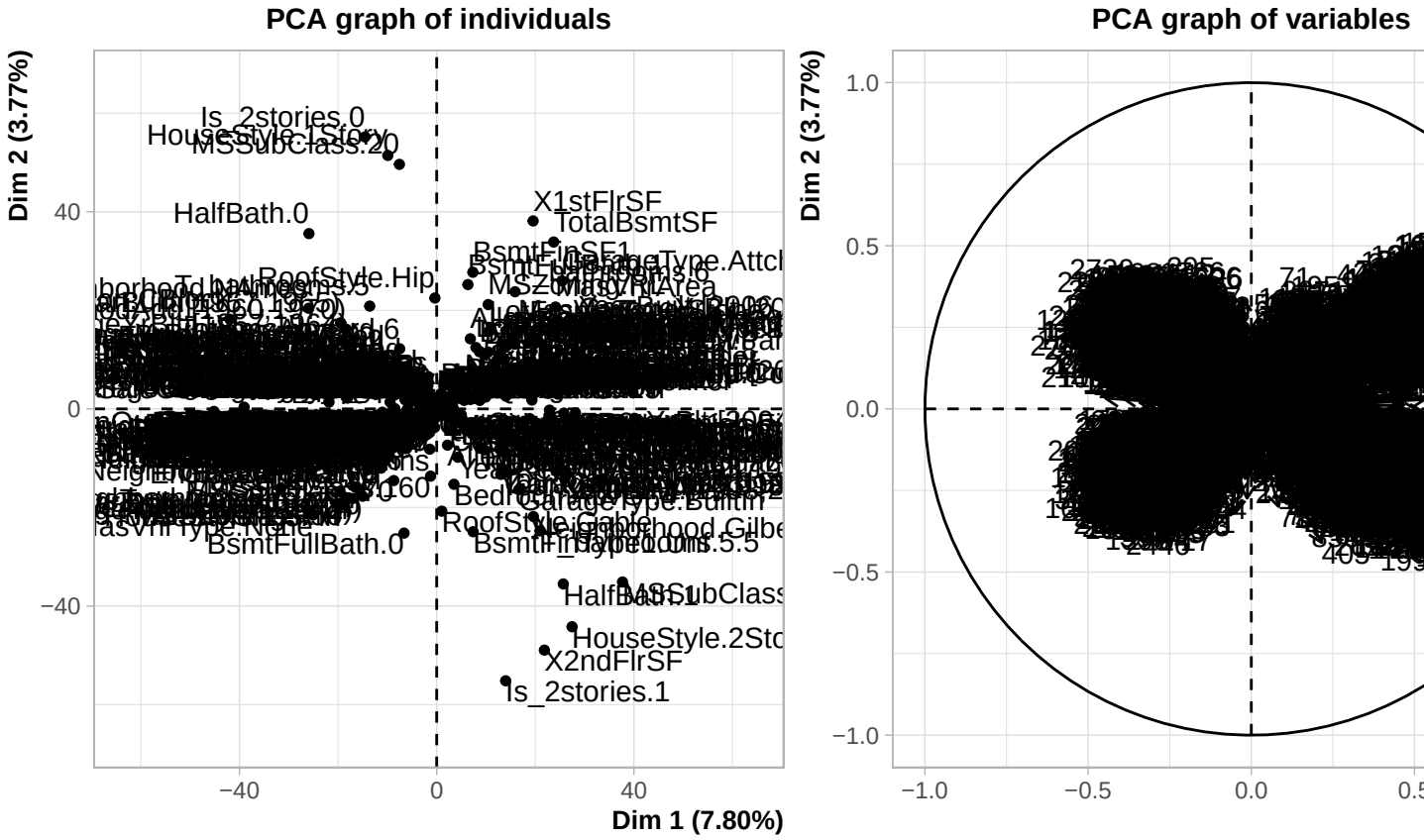

```
#cosine distance seems to minimize the inner cluster distance
sil8_cos <- silhouette(X3clust_8$cluster, dist.matrix(clusterMatrix, method = "cosine"))
fviz_silhouette(sil8_cos)
```

```
##   cluster size ave.sil.width
## 1       1  155          0.15
## 2       2   13          0.21
## 3       3  243          0.14
## 4       4    6          0.11
## 5       5  886          0.02
## 6       6  964          0.01
## 7       7   23          0.48
## 8       8  629          0.02
```

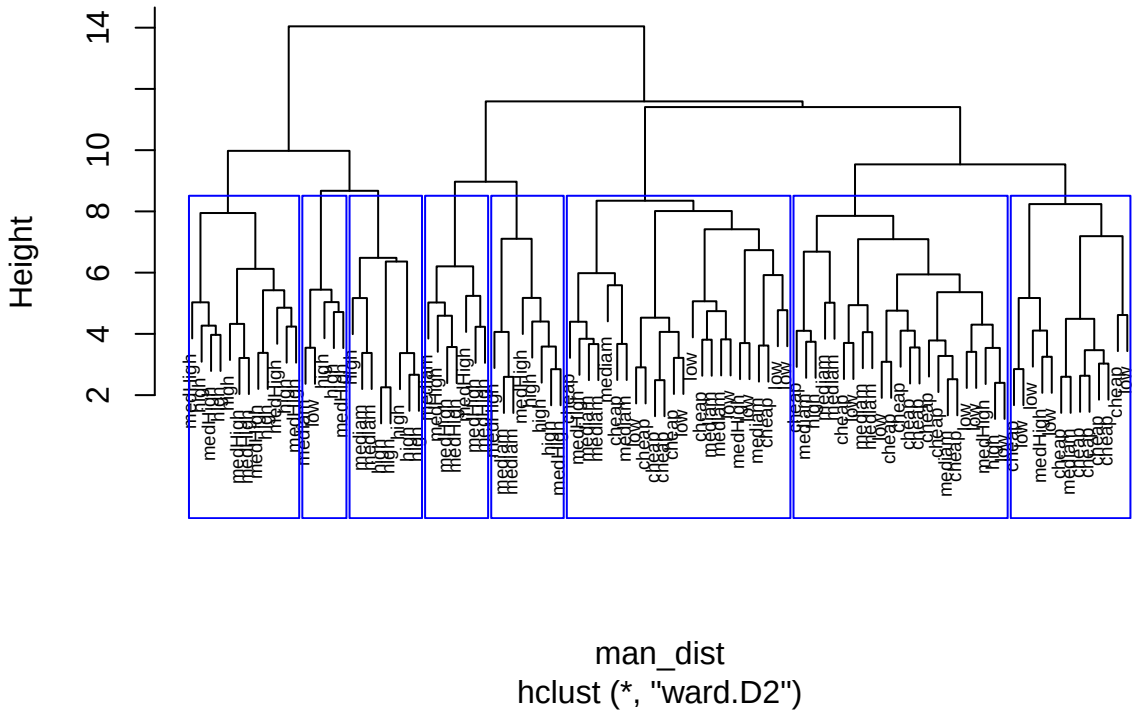


Principal component analysis is used to reduce the dimensionality down to 50. Subsequently, 8 clusters is repeatedly used with different distance measures, for each a dendogram is used to visualize the results. A tanglegram is used to compare the clustering results for cosine distance and euclidean distance.

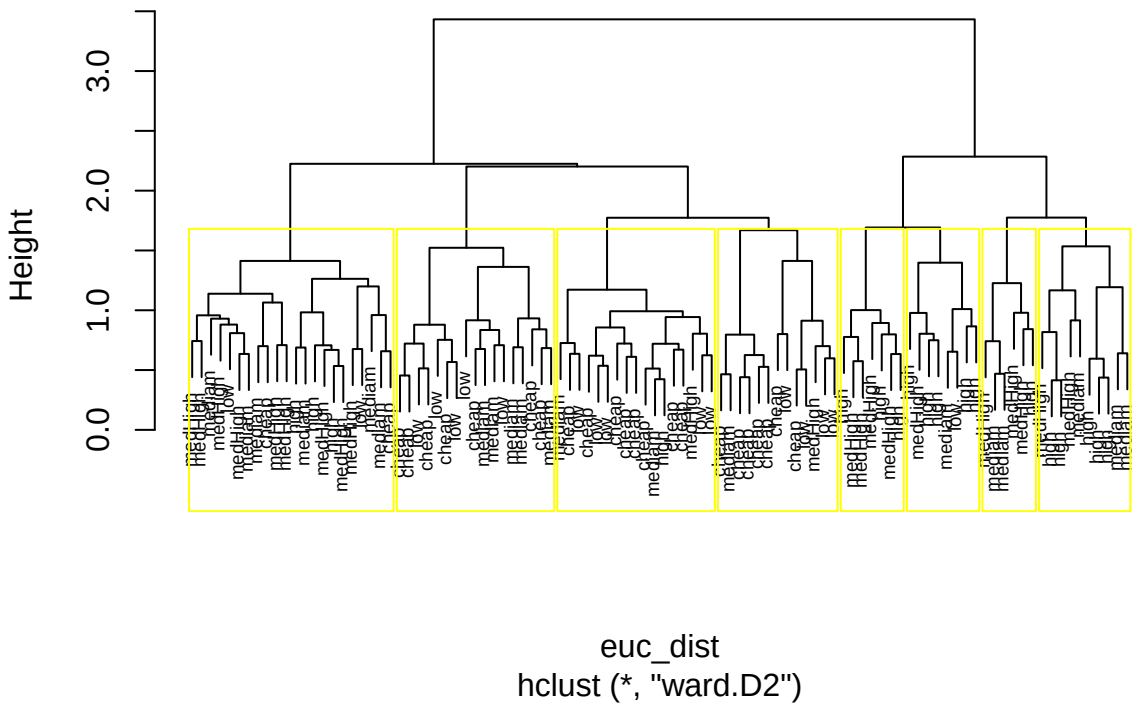
```
HIGHclust <- clusterMatrix
HIGHclust <- PCA(t(HIGHclust), ncp = 50)
```



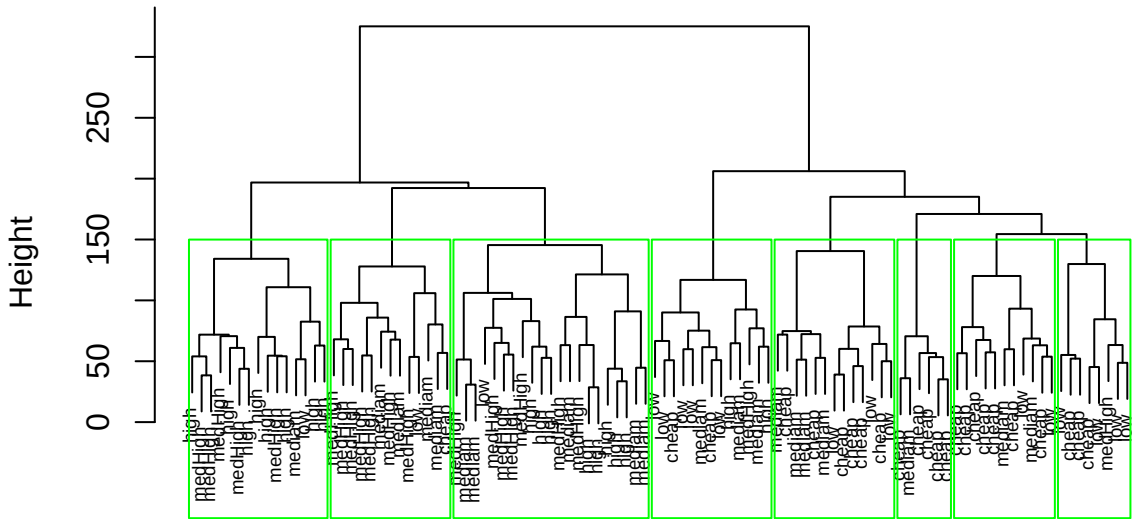

```
HIGHclust <-as.matrix(HIGHclust$var$coord)
clustNames <- discretize((targetValue), breaks = 5)
clustNames <- revalue(clustNames, c("[10.5,11.7)" = 'cheap',
                        '[11.7,11.9)' = 'low', '[11.9,12.1)' = 'mediam' ,
                        '[12.1,12.3)' = 'medHigh',
                        '[12.3,13.5]' = 'high'
                        ))


HIGHclust <- HIGHclust[1:1460,]
samps <- sample(1:1460,100)
row.names(HIGHclust) <- clustNames

man_dist <- as.dist(dist.matrix(as.matrix(HIGHclust[samps,]), method = "manhattan"))
fit1 <- hclust(man_dist, method="ward.D2")
plot(fit1, main = "Manhattan: 5 clusters", cex = .55)
euclid_small <- cutree(fit1, k=8)
rect.hclust(fit1, k=8, border="blue")
```




```
euc_dist <- as.dist(dist.matrix(as.matrix(HIGHclust[samps,]), method = "euclidean"))
fit2 <- hclust(euc_dist, method="ward.D2")
plot(fit2, main = "Euclidean: 5 clusters", cex = .55)
euclid_small <- cutree(fit2, k=8)
rect.hclust(fit2, k=8, border="yellow")
```

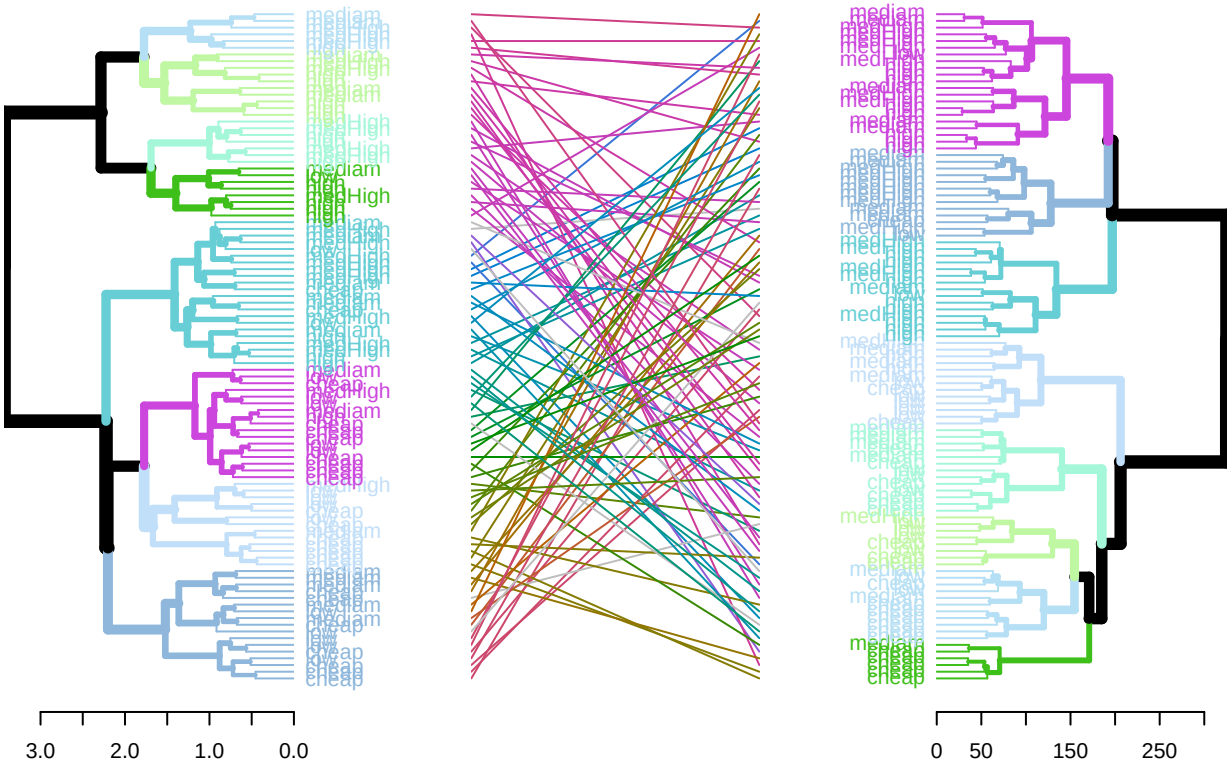



```
euc_co <- as.dist(dist.matrix(as.matrix(HIGHclust[samps,]), method = "cosine"))
fit3 <- hclust(euc_co, method="ward.D2")
plot(fit3, main = "Cosine: 5 clusters", cex = .55)
euclid_small <- cutree(fit3, k=8)
rect.hclust(fit3, k=8, border="green")
```




```
#https://www.r-graph-gallery.com/340-custom-your-dendrogram-with-dendextend.html
fit_3 <- as.dendrogram(fit2)
fit_6 <- as.dendrogram(fit3)
colors <- randomColor(8)
compareCosineData <- dendlist(
  fit_3 %>%
    set("labels_col", value = colors, k=8) %>%
    set("branches_lty", 1) %>%
    set("branches_k_color", value = colors, k = 8),
  fit_6 %>%
    set("labels_col", value = colors, k=8) %>%
    set("branches_lty", 1) %>%
    set("branches_k_color", value = colors, k = 8)
)
tanglegram(compareCosineData, sort = TRUE,
           common_subtrees_color_lines = TRUE, highlight_distinct_edges = TRUE
           , highlight_branches_lwd=TRUE,
           margin_inner=7,
           lwd=1
)
```

```
## Warning in 'order.dendrogram<-'('*tmp*', value = tree_new_leaf_numbers): The
## value you wish to insert into the dendrogram 'objects' has duplicate values.
## Often you are likely to prefer unique values in your vector. So be sure to check
## that you've used the correct vector here...
```

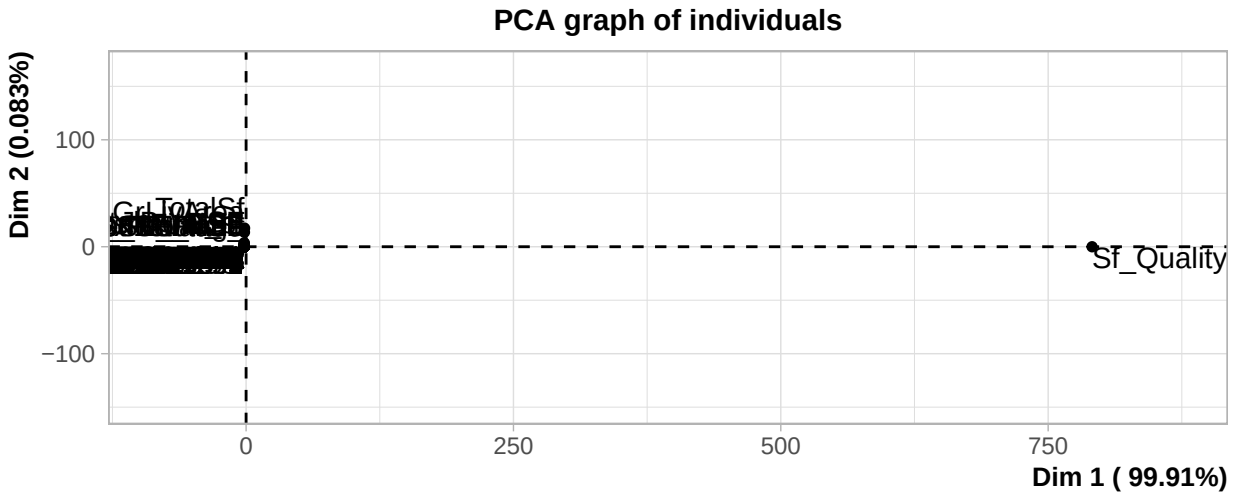


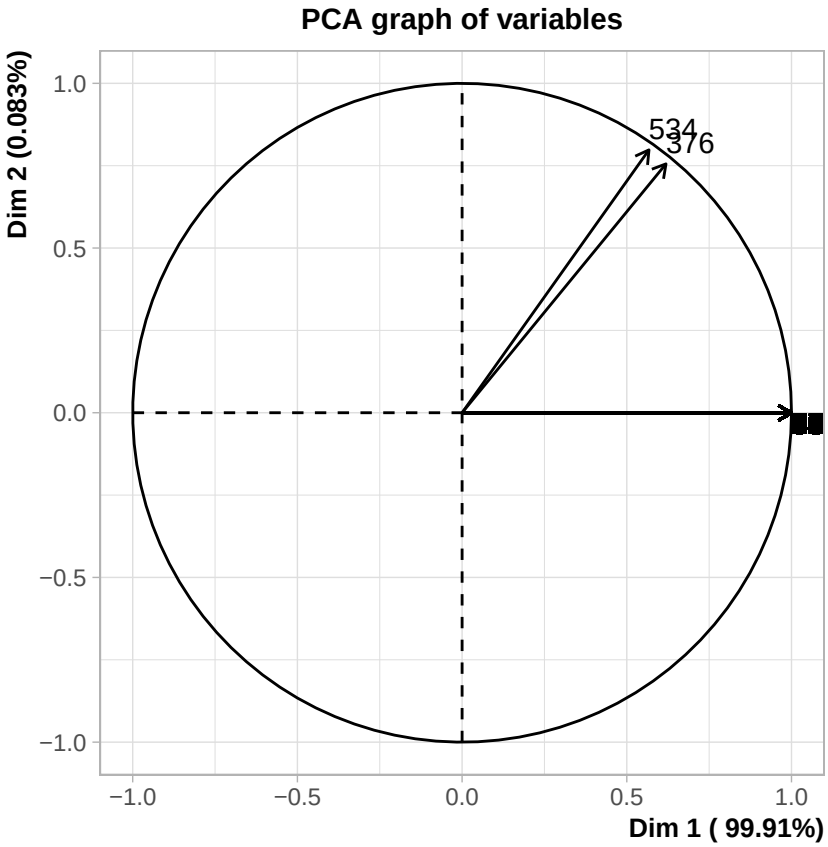
K-Nearest Neighbor

Knn is a simple algorithm that compares an observation to those closest to it. Here, Knn is used with PCA and without PCA to demonstrate that it actually performs better without the PCA.

The algorithm is then retrained with various values of K. These results are then visualized to understand how the error rate changes with K. This model does not do as well as others in predicting sale price and is not used in the final submission on Kaggle.

```
knnMatrix <- PCA(t(rf_matrix), ncp = 75)
```






```
knnMatrix <- data.frame(knnMatrix$var$coord)

knnPreds <- knn.reg(knnMatrix[indexes, ],knnMatrix[-indexes, ], targetValue[indexes], k=10)
scoreKNN <- rmsle(expm1(knnPreds$pred) ,expm1(targetValue[-indexes]))
scoreKNN
```

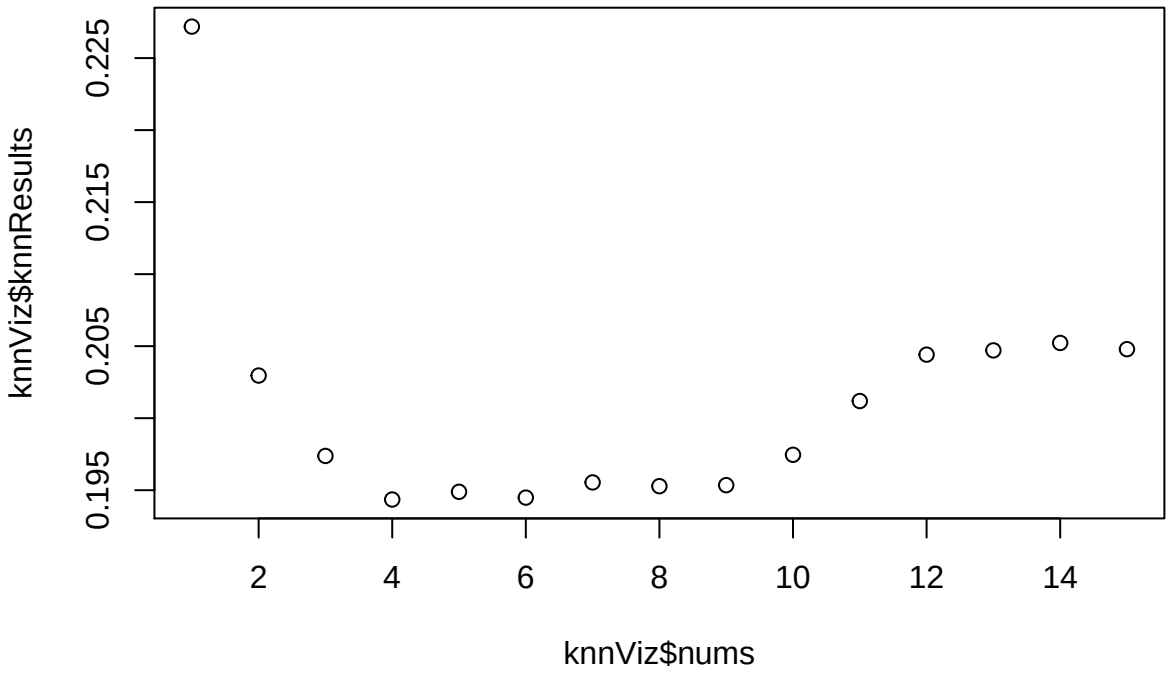
```
## [1] 0.2416432
```

```
knnMatrix <- rf_matrix

nums <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
knnResults <- c()
for (i in nums) {
knnPreds <- knn.reg(knnMatrix[indexes, ],knnMatrix[-indexes, ], targetValue[indexes], k=i)
scoreKNN <- rmsle(expm1(knnPreds$pred) ,expm1(targetValue[-indexes]))
knnResults[i] <- scoreKNN
print(paste(scoreKNN, i))
}
```

```
## [1] "0.227186234176097 1"
## [1] "0.202961268176326 2"
## [1] "0.197379223504742 3"
## [1] "0.194353784086648 4"
## [1] "0.194888788149604 5"
## [1] "0.194486465035189 6"
## [1] "0.195540630988947 7"
```

```
## [1] "0.195278877551828 8"
## [1] "0.195349165162886 9"
## [1] "0.197457779281175 10"
## [1] "0.201190785889734 11"
## [1] "0.204412633912035 12"
## [1] "0.204707259256292 13"
## [1] "0.205222509359923 14"
## [1] "0.204791052678367 15"
```

```
knnViz <- data.frame(nums,knnResults)


plot(knnViz$nums, knnViz$knnResults)
```



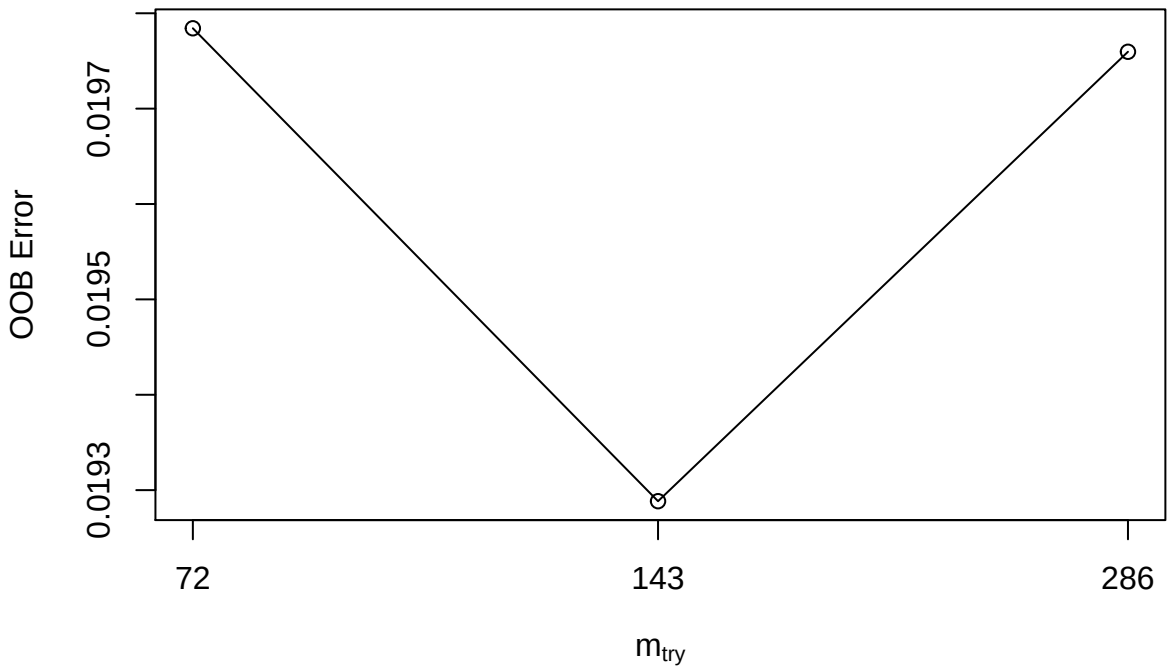
Random Forest

A random forest is an ensemble method. It uses many trees to make a prediction by committee. First the algorithm is tuned to test multiple values of mtry, the number of observations tried at each split. The random forest is then trained using a holdout set and then tested. A random forest is then retrained with all the data and predictions are made on the Kaggle testing set. Random Forest Explainer is then used to analyze the random forest. It shows which variables were selected most and at what position in the tree. Being one of the better models its predictions are used in the Kaggle submission. Although the tuning shows that mtry of 142 is better, an mtry of 72 will be able to generalize better and it is therefore the mtry of choice.

```
tuneRF(rf_matrix, targetValue)
```

```
## mtry = 143  OOB error = 0.01928842
## Searching left ...
## mtry = 72    OOB error = 0.01978424
## -0.02570576 0.05
## Searching right ...
```

```
## mtry = 286   OOB error = 0.01975963
## -0.02442964 0.05
```

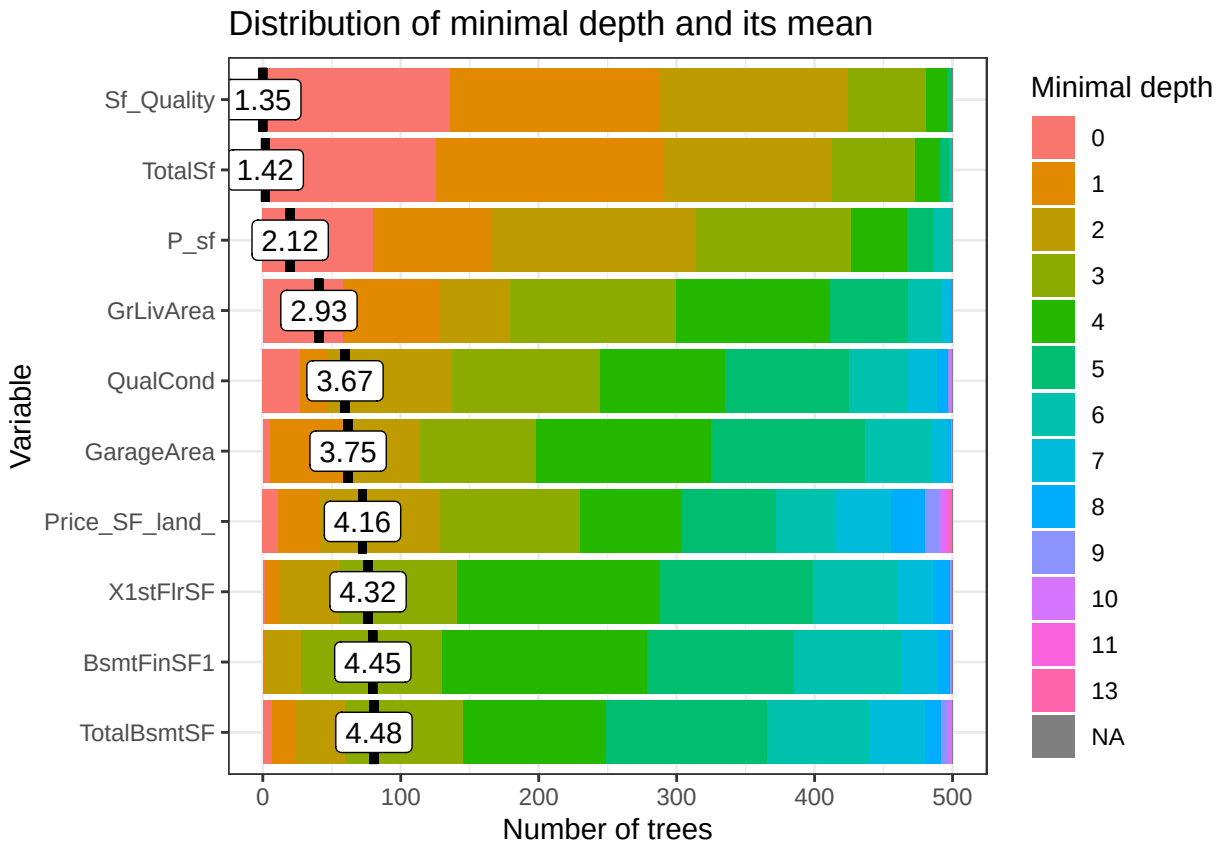



```
##     mtry   OOBError
## 72    72 0.01978424
## 143  143 0.01928842
## 286  286 0.01975963
```

```
rf2 <- randomForest(rf_matrix, targetValue, mtry = 72)
rfPreds <- predict(rf2, rf_final_move)


rf2 <- randomForest(rf_matrix[indexes,], targetValue[indexes])
rfPreds <- predict(rf2, rf_matrix[-indexes,])
rmsle(expm1(rfPreds), expm1(targetValue[-indexes]))
```

```
## [1] 0.1299866
```

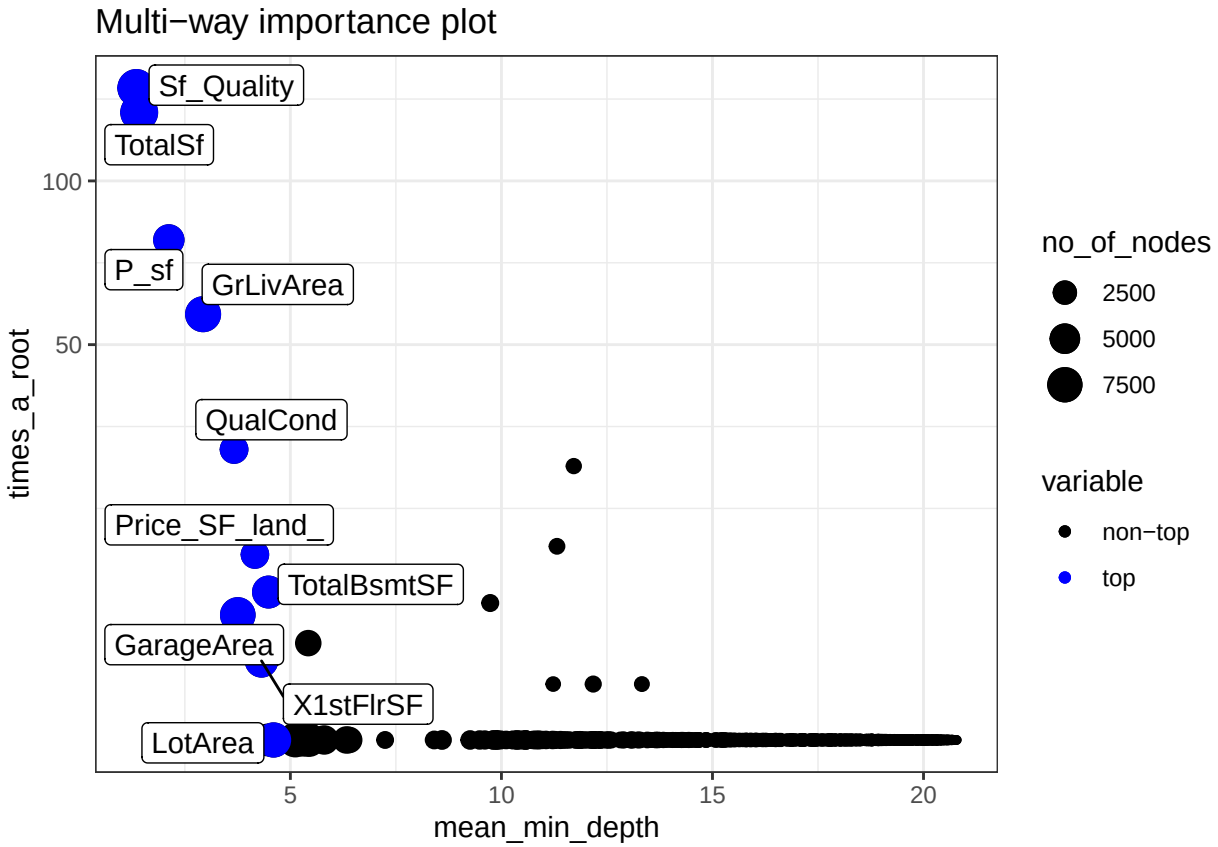
```
rf2 <- randomForest(rf_matrix[,], targetValue)
rfPreds <- predict(rf2, rf_final_move[,])


min_depth_frame <- min_depth_distribution(rf2)
plot_min_depth_distribution(min_depth_frame)
```




```
importance_frame <- measure_importance(rf2)
```

```
## [1] "Warning: your forest does not contain information on local importance so 'mse_increase' measure
```

```
plot_multi_way_importance(importance_frame, size_measure = "no_of_nodes")
```



XGBoost

Extreme Gradient Boosting is also a tree based ensemble algorithm. It was tuned through trial and error. A small learning rate of .0095 combined with many rounds of training, 4,250, yielded the best cross validated results. Alpha and lambda are both set above zero to help the model be more general and prevent over fitting. Col sample by tree and sub sample are set below one so not all data is available for every tree. This also helps reduce over fitting. 5 models are built with a cross validation scheme and all make predictions for Kaggle. The mean prediction is taken among the 5 and stored for later use. This is the strongest model of all attempted.

```
N <- nrow(rf_matrix)
kfolds <- 5
set.seed(5948)
rf_matrixPCA <- rf_matrix
holdout <- split(sample(1:N), 1:kfolds)
Id <- 1:nrow(final_move)
XGaggs <- data.frame(Id)
AllResults <- c()
AllModels <- c()
for (k in 1:kfolds) {
  Test <- rf_matrix[holdout[[k]], ]
  targsTest <- targetValue[holdout[[k]]]
  Train <- rf_matrix[-holdout[[k]], ]
  targsTrain <- targetValue[-holdout[[k]]]
  xg <- xgboost::xgboost(objective = 'reg:squarederror', eval_metric = 'rmsle',
                data = Train, label = targsTrain
                , eta=.0095,nrounds=4250,
                max_depth=3, min_child_weight=1.5,
```

```
                subsample=.4,
                colsample_bytree=.7,
                reg_alpha = .2,
                reg_lambda = 0.2,
                verbose = F)
  logPreds3 <- predict(xg, Test )
  preds <- expm1(logPreds3)
  test3 <- expm1(targsTest)
  score3 <- rmsle(preds ,test3)
  AllResults <- c(AllResults,score3)
  kagPreds <- predict(xg ,rf_final_move)
  newCol <- expm1(kagPreds)
  XGaggs <- cbind(XGaggs , newCol)
}
XGaggs_sub <- XGaggs[,-1]
XGaggs_sub$preFinal <- rowSums(XGaggs_sub)
XGaggs_sub$preFinal <- (XGaggs_sub$preFinal)/kfolds

mean(AllResults)
```

```
## [1] 0.1235897
```

LASSO

Lasso, or least absolute shrinkage and selection operator regression is a great model for this problem because of how to handles multicollinearity. The word shrinkage comes from the idea that certain predictors will be shrunk down to make the model more general. This is the exact problem with this data set so this model is a good choice. To help with the normality assumption of the model logs are taken of each predictor. Although this is brute force transformation it helped the accuracy of the model. The model is built and tested with a holdout data set. It is then retrained with all the training data and predictions are made on the Kaggle testing set. Using trial and error an alpha of .3 and a lambda of .01 were selected for the model.

```
#Take log of all predictors (improved accuracy)
for (i in 1:ncol(rf_matrix)) {
    rf_matrix[,i] <- log1p(rf_matrix[,i])
}

#Take log of all predictors (improved accuracy)
for (i in 1:ncol(rf_final_move)) {
  rf_final_move[,i] <- log1p(rf_final_move[,i])
}


#build model and calculate error rate
lasso_model_ <- glmnet(rf_matrix[indexes,], targetValue[indexes], alpha = .3, lambda = .01, standardize
predsLas_ <- predict(lasso_model_, rf_matrix[-indexes,])
rmsle(expm1(predsLas_), expm1(targetValue[-indexes]))
```

```
## [1] 0.1324491
```

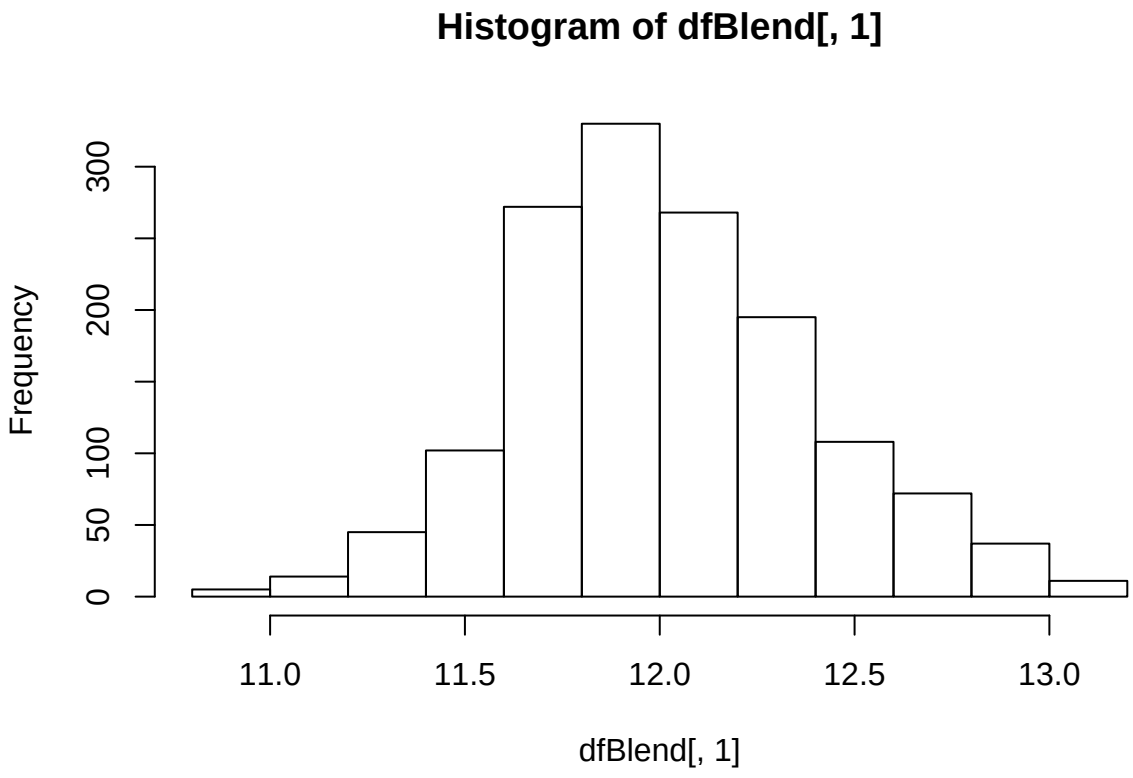
```
#make predictions for the Kaggle testing data
lasso_model <- glmnet(rf_matrix, targetValue, alpha = .3, lambda = .01, standardize = TRUE)
predsLas <- predict(lasso_model, rf_final_move)
predsLas <- expm1(predsLas)
```

Blending

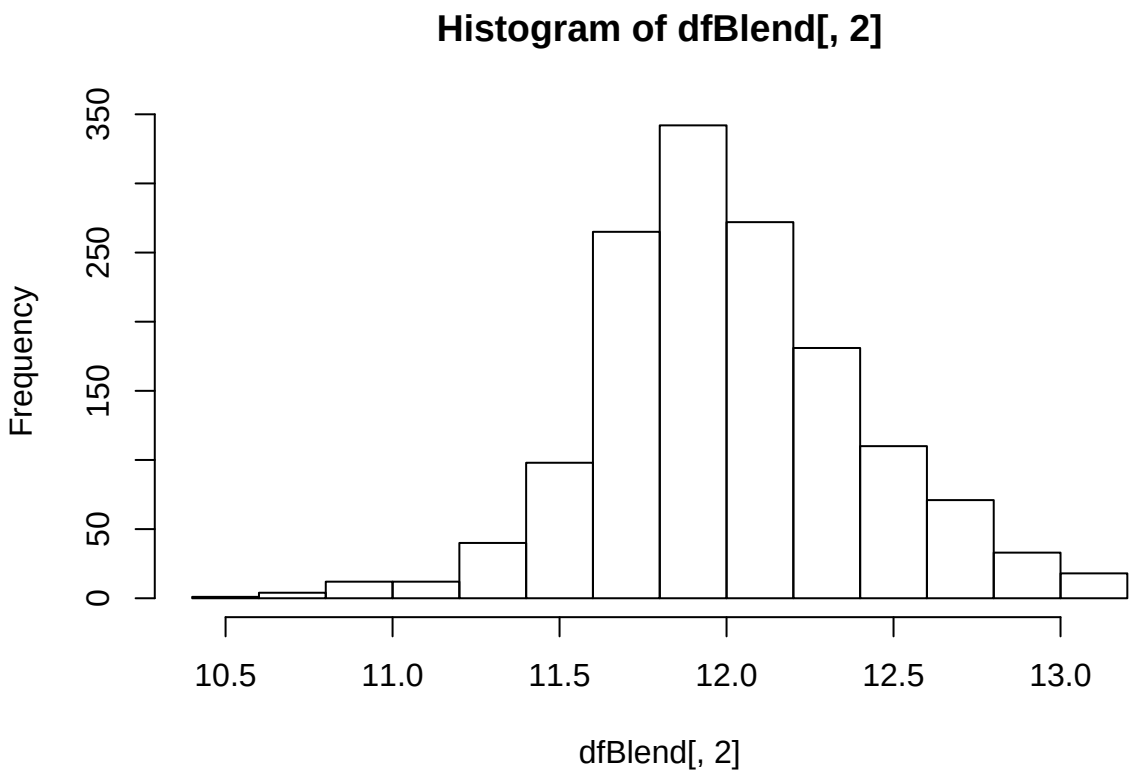
Simply making predictions with the model with the lowest error does not translate to the best possible results. Combining a few of the better models makes for better predictions on Kaggle. The SVM, XGBoost, Random Forest, and Lasso regression models are all used. A weight of .275, .7, -.15, and .175 are used on the models respectively. This drastically improved the score to an rmsle of .1208 and placed these predictions in the top 10% of all submissions on Kaggle for this competition.

However, there was a lot of trial and error used to tune the weights. A better way would potentially be leaving out roughly 15% of the data for the entire script and then use a linear regression to calculate the weights. Although that would improve these results even more this is still a large improvement from any individual model.

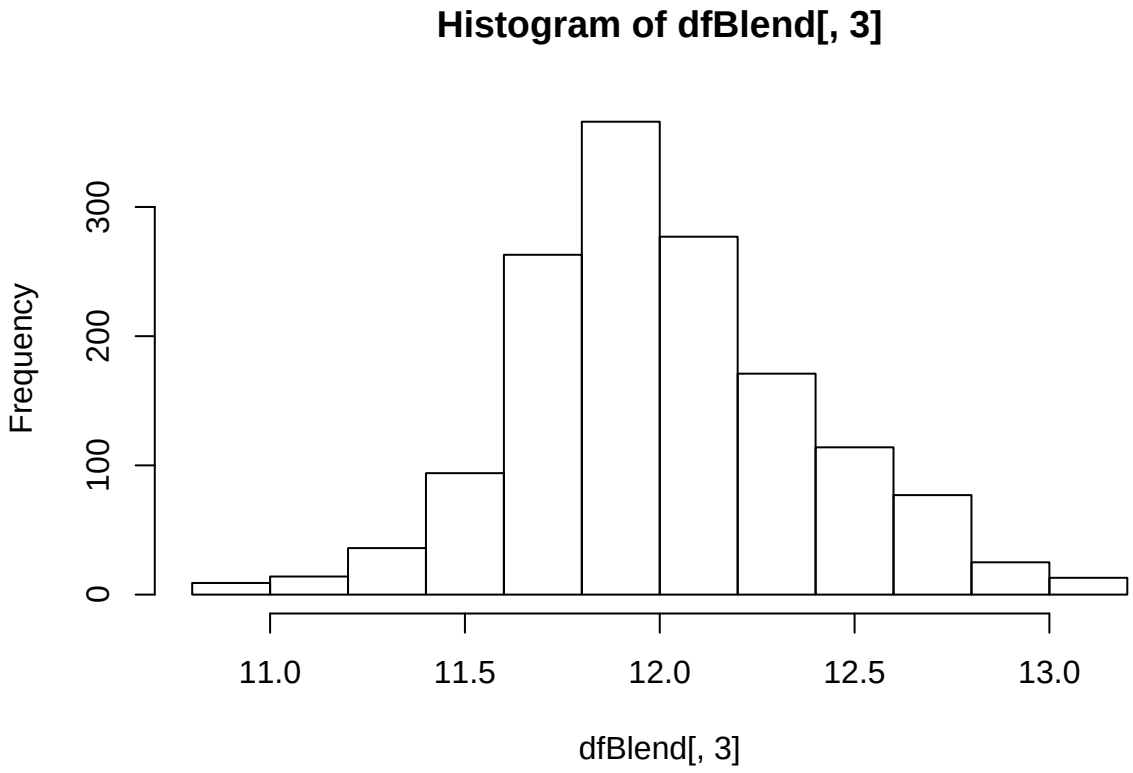
```
#build a data frame of predictions from the 4 best models
dfBlend <- data.frame(log1p(SVMaggs_sub$preFinal),log1p(XGaggs_sub$preFinal),rfPreds, log1p(predsLas))
colnames(dfBlend) <- c("svm", "xgboost", "randomForest", "lass")

#view distribution of each models predictions
hist(dfBlend[,1]) #solid
```



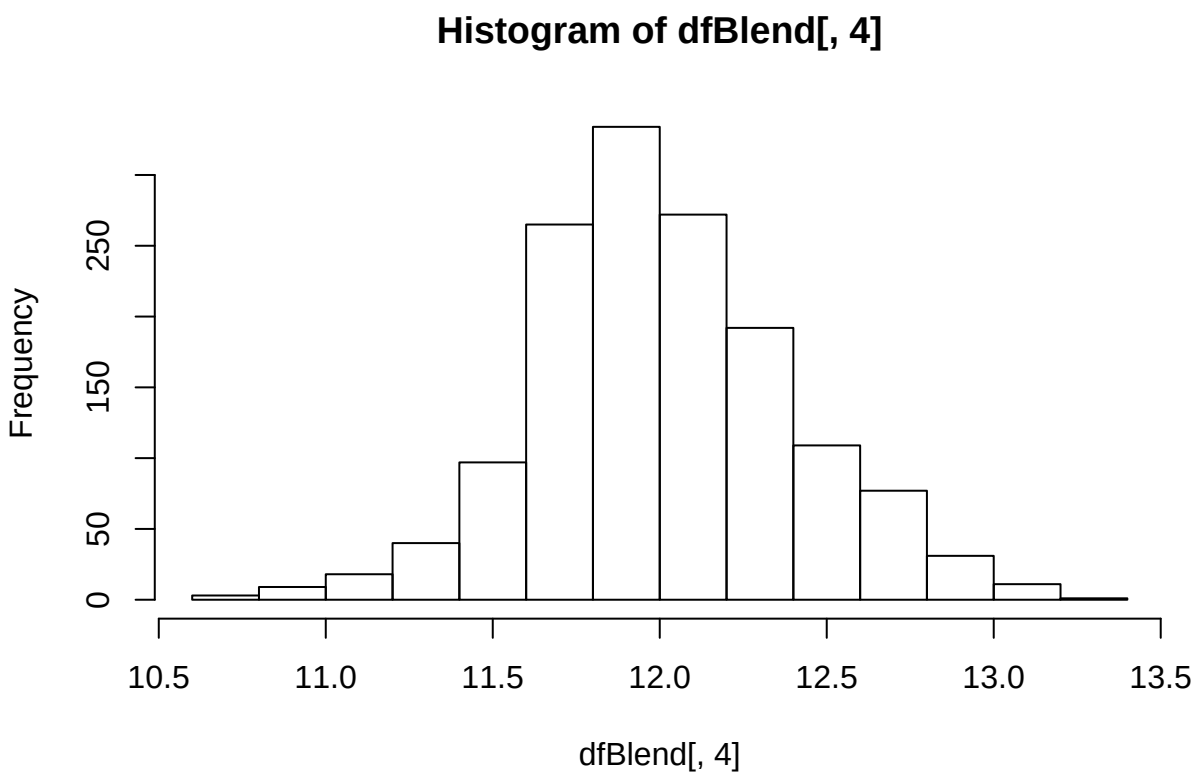

```
hist(dfBlend[,2]) #best model
```




```
hist(dfBlend[,3]) #negative weight seems to be the best
```



```r
hist(dfBlend[,4]) #solid
```




```r
#combine all predictions with the given weights
trythis <- (.275*dfBlend[,1] + .7*dfBlend[,2] +  -.15*dfBlend[,3] + .175*dfBlend[,4])

#write the predictions to a csv file for Kaggle submission
id <- c(1461:2919)
subFrame <- data.frame(id, expm1(trythis))
colnames(subFrame) <- c('Id','SalePrice')

write.csv(subFrame, "/Users/ronenreouveni/Desktop/knittedFile_Finalsub.csv")
```

Results

Naive Bayes

Although Naive Bayes was used in the pre-processing stage it is important to note its impact. The largest missing value in the data set was Lot Frontage. This measure was initially filled by just taking the median value from that neighborhood. However, this was a weak method. As shown in the bubble plot visualizations some neighborhoods had large differences but others do not. Naive Bayes takes much more information into consideration in filling in missing values. One of 10 values were chosen to fill in each missing value. Using Naive Bayes to fill in missing values is an important step in improving the accuracy of the model. Lot Frontage turns out to be a significant predictor and filling in missing values as accurately as possible is what can make the difference between a competitive result and an average one.

Association Rule Mining

ARM was used in a general setting and with setting housing sale price as the right hand side. Although this model was not used to make predictions on housing prices it is very useful. Association Rule Mining allows

for the discovery of interesting patterns in the data set. Understanding these patterns allows for increased insight and therefore a better approach in trying to solve any data issues that my arise.

{BsmtFinSF2=[0,7.3], MiscVal=[0,1.55e+04]} => {PoolArea=[0,738]}

This rule has 100% confidence. It explains that if BsmtFinSF2 and MiscVal are in the first, smallest, bucket then so will Pool Area.

The most interesting rules were found by setting the right hand side of the model to the target value, sale price. This was discretized into three buckets, low, medium, and high. These represent a cost for a home. The following are 4 of the most interesting rules. All of these rules have 100% confidence.

1. {MasVnrType=None, GarageType=none, TotalSf=[334,2.16e+03)} => {targs=low}
2. {OverallQual=4, TotalBsmtSF=[0,861), FullBath=1} => {targs=low}
3. {LotFrontage=[4.38,5.75], HeatingQC=Ex, GrLivArea=[1.66e+03,5.64e+03],Sf_Quality=[7.25e+26,1.11e+37]} => {targs=high}
4. {Neighborhood=NridgHt, MasVnrArea=[4.55,7.38], TotalSf=[2.82e+03,1.18e+04]} => {targs=high}

The first rule explains that with 100% confidence and support of .037, every property between 334 and 2,160 square feet with no mosaic veneer or garage will be a low value home.

The second rule asserts that with 100% confidence and support of .037, every property with an over quality of 4, small to no basement, and only one above ground full bathroom will also have a low value.

The third rule is interesting because with 100% confidence and support of .098, every property with Lot-Frontage log value between 4.38 and 5.75, a relatively large general living area and a high sf quality will mean a high value home.

Finally, the fourth rule brings in mosaic veneer area, NridgHt neighborhood, and large TotalSf. It says that again, with 100% confidence and support of .039, all of these properties will have a high value.

The reason these rules are important is because it helps illuminate that there is a consistent pattern as to what makes a property high, medium, or low valued. There are no exceptions to these rules because they all have a confidence of 100%.

Support Vector Regression

The custom build tuning algorithm yielded very interesting results. The radial kernel performed the worst and the vanilladot, linear kernel, and laplacedot were similar enough to warrant further investigation. The best cost associated with the linear kernel was about .022 and an error rate of .119. The radial kernel could only achieve an error rate of .13 with a cost of about 1.5. The laplacedot kernel had an error rate of just under .123 with a cost of about 18. Multiple regression types were tested with laplacedot and nu-svr was the best. Although at first glance the linear kernel is much better than the laplacedot cross validation is needed to verify this. Due to the multicollinearity issues, the linear kernel will be more susceptible to these issues than the laplacedot kernel. With cross validation, the linear kernel performed with an error rate of .132 and the laplacedot kernel had an error rate of .127. The laplacedot kernel is able to better generalize this specific data. It is the kernel of choice for making predictions on the Kaggle data set. This can also be seen in the box plots of the error rates for each kernels predictions. The linear kernel and laplacedot kernel have some folds with an error rate of under .1. However, the worst model with laplacedot is .18 while the worst linear kernel model was .25. Each model in the 10 fold cross validation scheme with the laplacedot kernel was used to make predictions on the Kaggle testing set. The average is taken of all these 10 predictions and saved for later use as the final SVM prediction.

Decision Trees

For predicting sale price, three different trees were grown. The first used default pruning and it had an error rate of .1966. This is the baseline tree and subsequent trees will be compared to this. An error of .1966 is not strong enough to warrant use in the Kaggle submission. Next, a tree with pruning turned off is grown. As expected, the error rate drops to just above .2. This is still a worthwhile experiment because it

allows for tuning CP. This parameter is the complexity parameter and it is used to stop growing the tree. The cross validated error seems to be minimized at cp of 2.0627e-03. The corresponding error rate is .187. A nice improvement from the default pruning error rate of .1966 but still not good enough to be used for predictions.

Decision Trees were also used to highlight a data leakage and multicollinearity issue. A tree was grown in order to predict what neighborhood an observation falls into. The error rate is initially nearly 99%. However, the feature engineering created a feature that has perfect multicollinearity with neighborhood. It was price_sf_land. It came from taking median value for each neighborhood that could then be scaled and used in other interactions. Once this is removed the accuracy for predicting neighborhood falls to under 50%. This makes sense because while some neighborhoods are very different, most are very similar. This was also evident from the bubble plot of sale price by neighborhood.

Clustering

Kmeans:

Clustering was used to try and understand if similar priced homes could be grouped together. Kmeans is initially used without sale price. This is to see if observations could be separated and grouped based on only the predictors. The elbow method is used to select values for K. K = 3, 8 and 15 were all tested. Based on the silhouette results this does not seem to be possible. Kmeans intrinsically uses euclidean distance for its calculations. However, once it creates its clusters different distance measures can be used to measure the average silhouette width. Whichever distance measure minimizes this value will probably be the best distance measure to use. Cosine distance seemed to do the best, this makes sense because it is the best at handling high dimensionality.

Hierarchical:

Sale price was discretized into 5 bins, cheap, low, medium, medHigh, and high. An observations row name is then replaced with the appropriate price category. This allows for a very easy analysis of the dendrogram. Principal component analysis is used to lower the dimensionality down to 50 observations. A dendrogram is made using 5 clusters and euclidean, Manhattan, and cosine distance measures. The goal is to see if the clustering algorithm can group observations with the same price bucket together. While there were some observations grouped correctly together clustering did not do a good job of this. Furthermore, splitting homes into 5 bins is an oversimplification and even with this clustering could not consistently cluster them together. Finally, a tanglegram is used to compare the clusters of two different distance measures, euclidean and cosine. The fact that the web is so messy indicates that there are not consistent clusterings between these two models.

K-Nearest Neighbor

Knn can be used in regression problems as well as classification ones. PCA was also used here but it did not improve the model. With PCA reducing the dimensionality down to 75 predictors and k = 10 the error rate is .24. This is the worst error rate so far. The corresponding error rate with k = 10 and no PCA is .197, a large improvement. A for loop is then run to test each K between 1 and 15. The values for K and their respective error rates are graphed to visualize how the error rate changes with K. The error rate is minimized with a K of 4 and an error rate of .1943. This is not accurate enough to use in the final submission to Kaggle.

Random Forest

The random forest is first tuned to find the optimal value of mtry. Although the tuning algorithm says that the best mtry is 143 this makes the model less generalized. When making submissions to Kaggle 72 is actually better because it is more general. The random forest has an error rate of about .128. This is a relatively strong error rate compared to other models. The random forest visualizations show which predictors were set as the root more often. It shows that the top 3 most important predictors were TotalSf, Sf_Quality, and P_sf. All three of these values were engineered. This is very encouraging because it highlights that the engineered features are adding a tremendous amount of value. These are the predictors that on average reduce the most noise in the data set. The model is retrained on all the data and predictions on the Kaggle test set are saved for later use.

XGBoost

The XGBoost model has the strongest predictions with an average error rate of .123. Cross validation is used to test the model. The XGBoost model is trained and tested with 5 folds, each model's predictions are averaged to then be submitted to Kaggle. This was found to be the strongest model and the most significant in predicting sale price.

Lasso

A Lasso model was selected because it can handle multicollinearity and high dimensional data well. This was confirmed because the error rate is .132. Although this is well above the XGBoost it is strong enough to potentially be a significant predictor of sale price. This model also generalizes well and will be a stabilizing factor in the final predictions. The model is trained and tested using a simple train and test set. Finally the model is trained on all the available data and predictions are made on the Kaggle test set.

Blending

Although XGBoost has the best error rate, combining all relevant models performs much better than the XGBoost model, or any model, individually. The four best models were all chosen to participate in the blending process. The SVM, XGBoost, Random Forest, and Lasso regression were all selected. They were given weights of .275, .7, -.15, and .175 respectively. Weights were chose by putting more emphasis on models that had better results. However, there was also a lot of trial and error. Once the weight of the random forest was set to zero, more improvements were made by reducing it further. This process can be continued, but the point of this section is show that a blended model can outperform the best model within the blending group. This blending technique places the predictions in the top 10% of all Kaggle submissions for this competition.

Conclusion

Advancements in technology and prediction accuracy often have the power to make human predictions obsolete. This does not mean that there is no human that can make better predictions than a machine. However, majority of the time the machine will do better. Real estate markets are very often controlled by real estate agents making decisions about how much a property is worth. Allowing technology to replace this aspect can have a huge impact on this market. Not only in terms of everyday buyers and sellers benefiting but also investment firms can gain a large competitive advantage here. Having access to information that competitors do not have is an exploit that can lead to a massive advantage.

This report shows that technology can be used to achieve this competitive advantage. If a firm has the ability to check housing prices with a model then they can compare those predictions with what properties are listed at. It would be simple to purchase properties that the model predicts are worth more than what the listed price is. If this process was repeated then an investment firm could build an entire portfolio of properties they purchased for less than what they are worth. This is a prime example of how innovative technologies and utilizing analytics could create a large competitive advantage. On a smaller scale this could be very beneficial for individual or family investors.

Peoples' homes are often their highest valued asset. Building wealth is usually based on the value of one's home. If an average person severely overpays for their home then this will have a negative impact on their entire financial lives. A real estate agent may just be motivated to make a deal and not always care about getting the absolute best price possible. This technology could help save many people from falling into the trap of spending too much on a home or selling their home for too little. A system that can accurately predict the sale price of home has the potential to drastically alter the way real estate markets function. Who has access to this technology will decide who benefits and who does not.