My Comprehensive Evaluation

A Comprehensive Evaluation Report
Presented to
The Statistics Faculty
Amherst College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts
in
Statistics

Thyne Boonmark

February 2019

Acknowledgements

I would like to thank professor Nick Horton for all of his advice and help throughout my statistical career at Amherst. Thank you for guidance over the past few years. I would also like to thank professor Amy Wagaman for being a wonderful professor and whose classes have fostered my love for statistics. Lastly, none of my work at Amherst would have been possible without the endless support from my parents. Thank you so much for believing in me and helping me achieve all that I have done.

Table of Contents

List of Figures

Abstract

The use of automated machine learning models has become more and more ubiquitous in many industries today. However, while these methods may be great for prediction, the complexity nature of their training make it difficult to understand the models and interpretation of their results even more so. In an effort to improve model interpretability for individual observations, this report will discuss the calculation of shapley values. Shapley values are a measure of what an observation's feature value contributes to a model's prediction and can be incredibly useful due to the relative ease of their interpretation. To fully understand shapley values, this report will walkthrough how to calculate and approximate them using the IML R package. Included is also a study of the King's County housing dataset using shapley values, which demonstrates how these measurements of feature contribution can confirm human intuition about data.

Introduction

After using various statistical models in research and throughout my classes, I began being concerned with not only the predictive power of these models but also the interpretability of their results. Today many business make use of blackbox models in order to automate their large distributed systems; but with the ubiquitous use of these systems, the need to understand how they make predictions becomes even more necessary.

One recent problem that got me interested in the subject of model interpretability was related to Youtube and their monetization of people's videos. Youtube uses machine learning systems to automatically handle copyright claims made on videos and to decide whether or not a person's channel can run advertisements to make money. However due to the complexity of the models, there have been many instances where Youtube creators have had their videos prohibited from running advertisements without any clear reasoning. Due to the blackbox models being used, these creators are unable to understand the advertisement system and unknowingly make videos that do not meet Youtube's standards. Methods such as shapley values can help improve transparency of similar automated systems and, in this case, would help identify what kinds of videos would meet proper advertising standards.

In this report I aim to do the following: explain how shapley values can be calculated for single observations, explain how shapley values can be calculated more efficiently for datasets with many predictors using the IML package in R, apply these shapley values to better understand how a randomforest model predicts house prices.

Chapter 1

The Shapley Value

1.1 What is the Shapley Value

The shapley value was originally a game theory concept used to answer the following question: in a group of players with varying skills who are working together for a collective payout, what is the fairest way to divide the payoff among the players? For shapley values' statistical application, we can think of the following terms in relation to our model and its features.

- Players: feature values for an observation
- Game: making a prediction for that observation
- Payout: Actual prediction for that observation minus the average prediction for all instances
- Coalition: combination of feature values for an observation not including the feature you are testing for

Our overall goal is to explain the difference between an observation's prediction and the average prediction for all other observations by calculating each feature value's contribution to the prediction. Going back to its game theory roots, it is easy to understand the process of calculating marginal contribution. Let us say that all features are players and are working together in a room to make a prediction. Then the marginal contribution of one feature i is the difference between the prediction made by the group of features already in the room and the prediction made when the feature i joins. (Molnar, 2019)

To demonstrate how Shapley Values are calculated, we will make use of the diamond dataset (found here: `https://www.kaggle.com/shivam2503/diamonds/version/1`) which includes the following variables:

- price: price of the diamond in US dollars
- carat: weight of the diamond in carats
- cut: quality of the diamond's cut with values fair, good, very good, premium, or ideal
- color: diamond color which can be J, worst color; I; H; G; F; E; and D, best color

- clarity: measure of how clear a diamond is; I1 worst clarity, SI1, SI2, VS1, VS2, VVS1, VVS2, IF best clarity
- x: diamond length in millimeters
- y: diamond width in millimeters
- z: diamond depth in millimeters
- depth: total depth percentage, which is calculated from x, y, and z
- table: width of top of the diamond relative to widest point

For this example we will predict the price of a diamond using its carat weight, clarity, length, width, and depth. We will only make use of these predictors to keep things simple, and we will also remove a few outliers in the dataset whose sizes to price relationship differ greatly from the rest of the dataset and those that have diamond dimensions of 0.

When looking at the variable length's relationships with price in figure 1.1, we see a distinctly curved relationship. In order to better fit a linear model to the data we will transform the price variable by taking its square root.

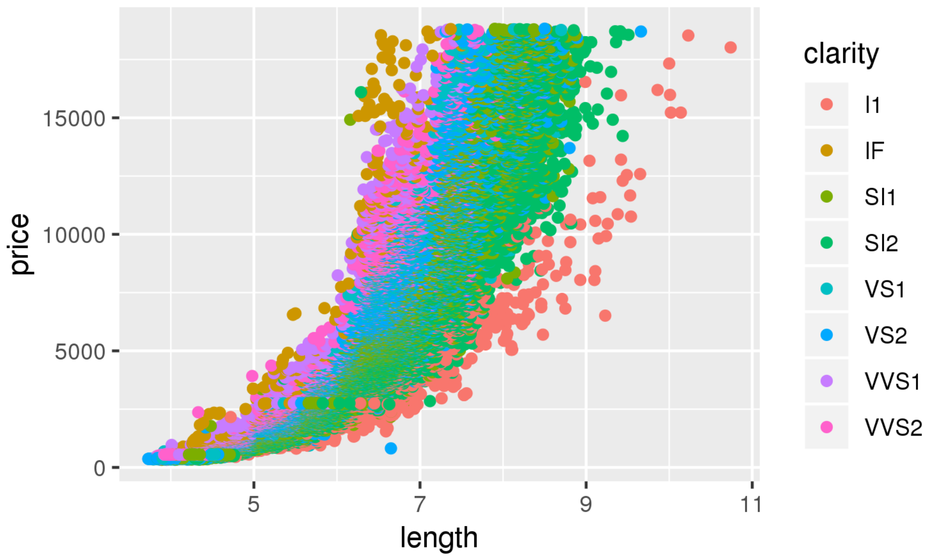


Figure 1.1: Diamond Weight in Carats vs Price of the Diamond

In figure 1.2 we see the relationship between diamond length and square root price. Across all diamond clarities, the price of a diamond tends to increases as length of the diamond increases.

In figure 1.3 we see the relationship between diamond width and square root price. Across all diamond clarities, the price of a diamond tends to increases as width of the diamond increases.

In figure 1.4 we see the relationship between diamond depth and square root price. Across all diamond clarities, the price of a diamond tends to increases as depth of the diamond increases.

In figure 1.5 we see the relationship between diamond weight in carats and square root price. Across all diamond clarities, the price of a diamond tends to increases as

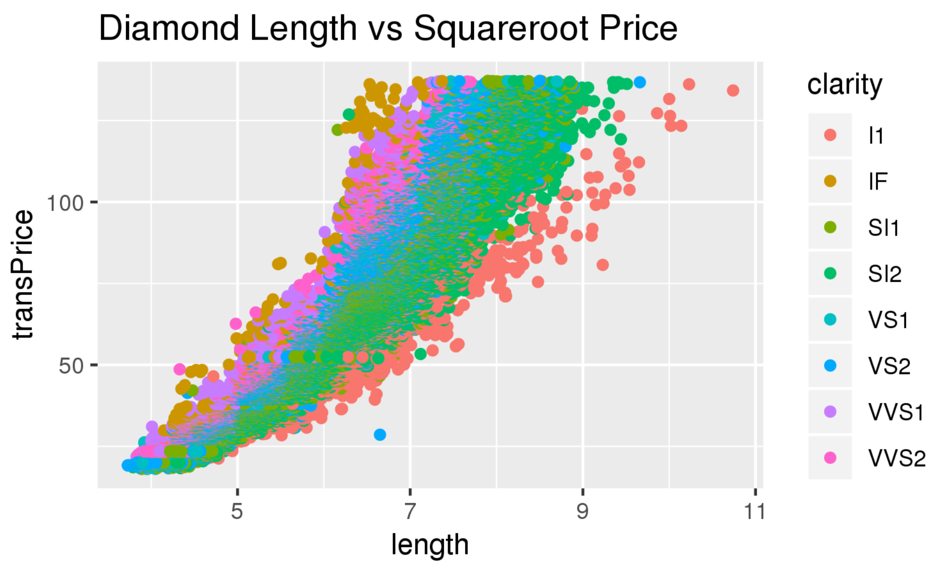


Figure 1.2: Diamond Length in millimeters vs Price of the Diamond

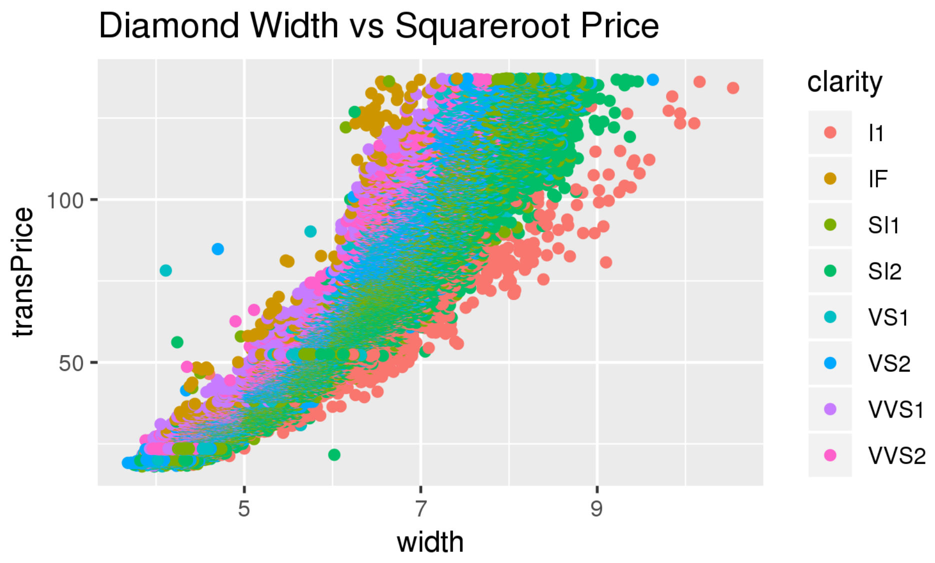


Figure 1.3: Diamond Width in millimeters vs Price of the Diamond

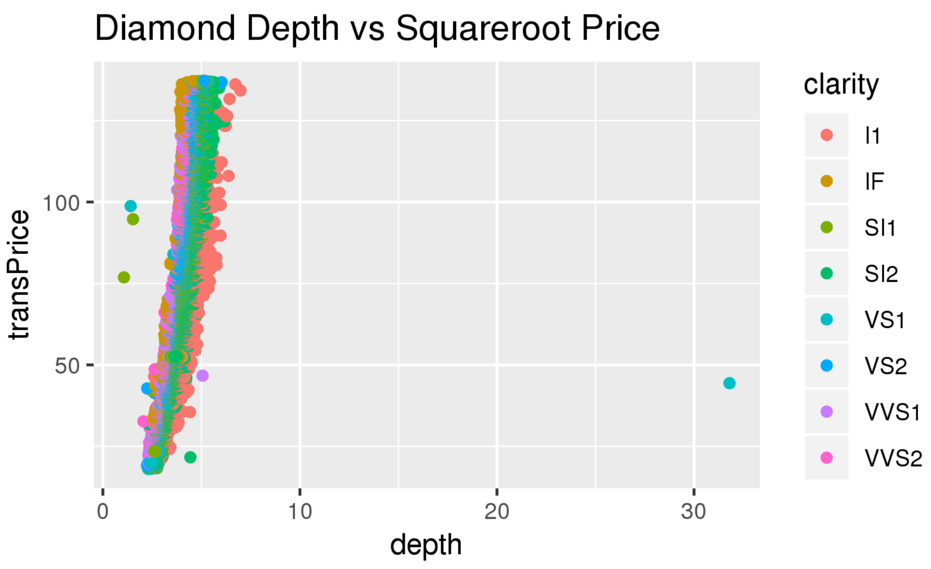


Figure 1.4: Diamond Depth in millimeters vs Price of the Diamond

the weight of the diamond increases.

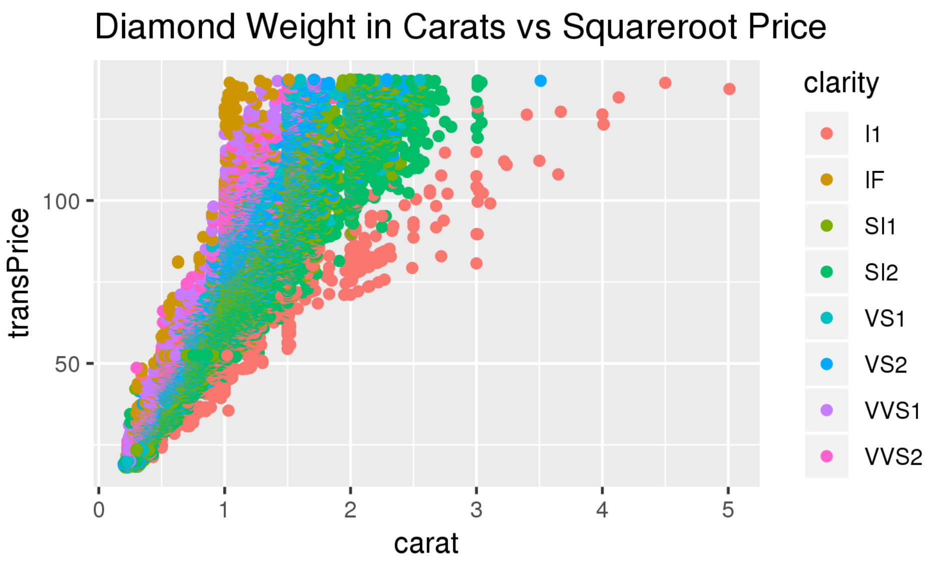


Figure 1.5: Diamond Weight in Carats vs Price of the Diamond

In each of the figures 1.1 through 1.5 we can also see how diamond prices tend to be distributed across the different clarity levels. According to the Gemological Institute of America, the different levels of clarity are as follows, in order from highest to lowest clarity:

- IF
- VVS1
- VVS2

- VS1
- VS2
- SI1
- SI2
- I1

Diamonds of higher clarity are regarded as more valuable, and we can see this in the data. When we look specifically at the distribution of diamond prices among diamonds of similar size in figure 1.6, we can verify that diamonds of with greater clarity tend to have higher prices. Here we compare the prices of similar sized diamonds, ones below the 25% quartile of length, width, and depth, because most large diamonds typically have worse clarity but their greater size raises their price to be more than that of smaller, higher clarity diamonds.

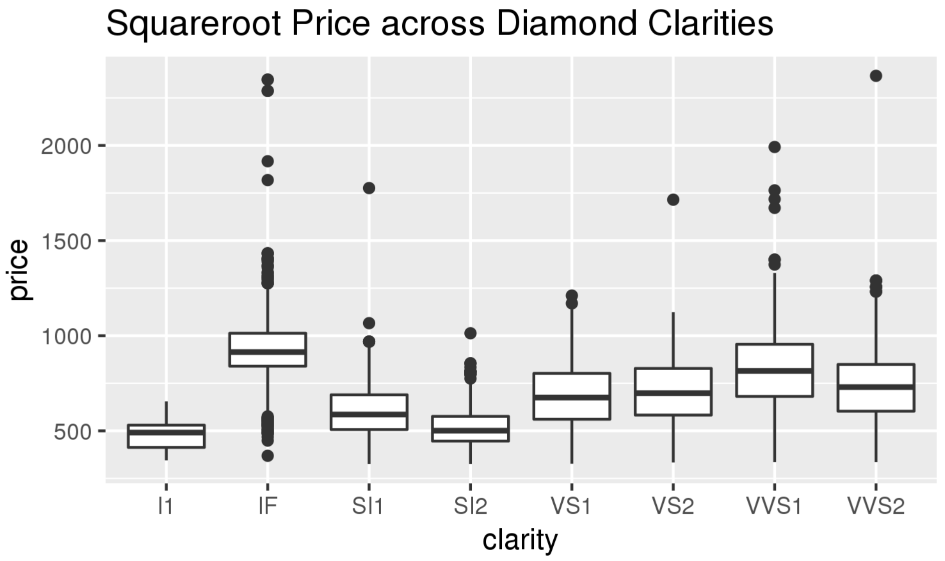


Figure 1.6: Diamond Clarity vs Price of the Diamond

Now that we better understand how the different features of diamonds relate to their price, we can fit our model. We fit a multiple linear regression model to predict price using diamond weight in carats, width, length, depth and clarity. Looking at the model, we have an adjusted R-squared value of 0.945 and every predictor appears to be significant. We also see that the quantitative predictors carat, length, width, and depth all have positive coefficients. This matches our initial intuition that the larger the diamond the greater the price. For clarity, every clarity level has a positive coefficient and the magnitude of this coefficient is greater for the higher quality clarity levels. This also matches our intuition that higher clarity diamonds tend to demand a higher price.

```
Call:
lm(formula = transPrice ~ carat + clarity + length + width +
    depth, data = diamondData)

Residuals:
       Min          1Q     Median          3Q         Max
-80.878955  -3.527671  -0.296196   3.127912  51.997008

Coefficients:
               Estimate  Std. Error    t value   Pr(>|t|)
(Intercept) -79.8602698   0.5523859 -144.57334 < 2.22e-16 ***
carat        25.3170002   0.2973088   85.15387 < 2.22e-16 ***
clarityIF    34.1893790   0.3000461  113.94710 < 2.22e-16 ***
claritySI1   22.1134716   0.2572427   85.96346 < 2.22e-16 ***
claritySI2   16.8190111   0.2585630   65.04801 < 2.22e-16 ***
clarityVS1   27.8135305   0.2623073  106.03414 < 2.22e-16 ***
clarityVS2   26.4198882   0.2582278  102.31232 < 2.22e-16 ***
clarityVVS1  32.1314248   0.2774316  115.81748 < 2.22e-16 ***
clarityVVS2  31.7552377   0.2700964  117.57003 < 2.22e-16 ***
length        2.6969738   0.5168605    5.21799 1.8154e-07 ***
width        12.0272563   0.5085828   23.64857 < 2.22e-16 ***
depth         1.7764208   0.1911359    9.29402 < 2.22e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.711446 on 53906 degrees of freedom
Multiple R-squared:  0.9450804, Adjusted R-squared:  0.9450691
F-statistic: 84330.64 on 11 and 53906 DF,  p-value: < 2.2204e-16
```

After creating out model, we will pick an observation from the dataset and calculate shapley values for each of its feature values in order to explain the difference between that observation's prediction and the average prediction across the data. The average prediction for square root price on the diamond dataset is 55.775 and the first observation has a predicted square root price of 5.6199 The observation that we will examine has the following feature values:

- carat = 0.23 carats
- clarity = SI2
- length = 3.95 millimeters
- width = 3.98 millimeters
- depth = 2.43 millimeters

Comparing this diamond's feature values to the features' distribution across the dataset in figure 1.7, we see that the carat weight of 0.23 is below the 1st quartile of diamond weights, the length of 3.95 is below the 1st quartile of diamond lengths, the width of 3.98 is below the 1st quartile of diamond widths, the depth of 2.43 is below the 1st quartile of 2.91, and the clarity of SI2 is the second worst clarity level.

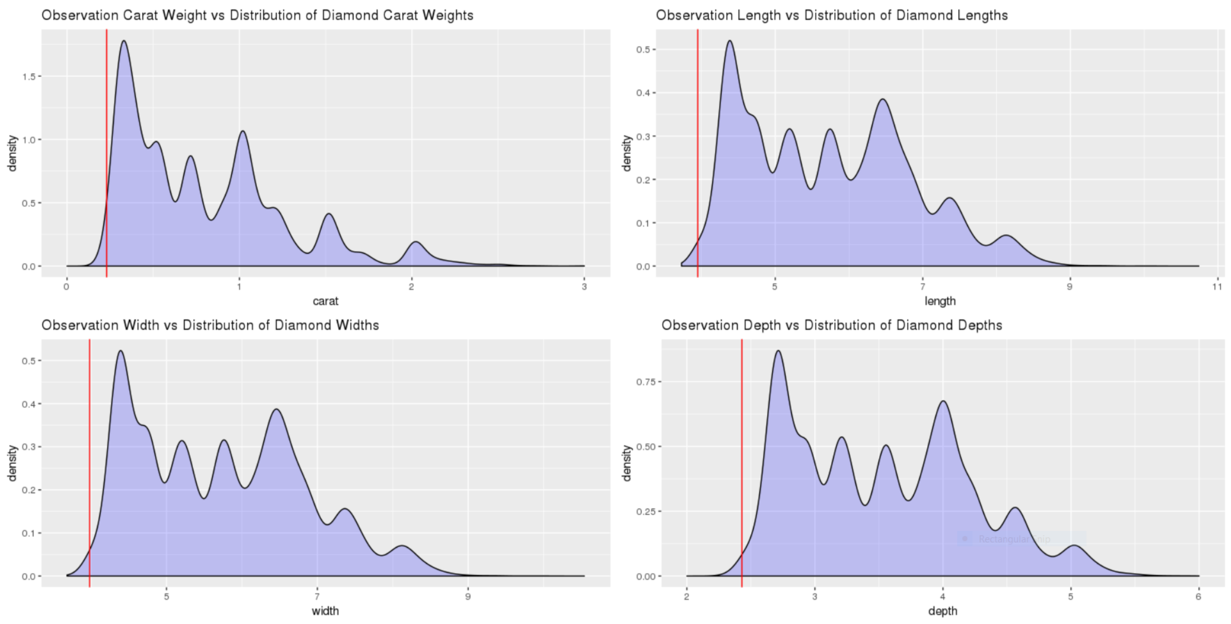


Figure 1.7: Single Observation's feature values Vs feature values across Diamond Dataset

Because the dimensions and weights of the diamond is quite low compared to the other diamonds and the clarity is of low quality, we would expect these features to lower to predicted price of the diamond. This is reflected by the fact that the predicted square root price 5.6199 is much lower than the average prediction across the dataset of 55.775. We can better quantify each feature's contribution to the prediction using shapley values.

1.2 How to calculate Shapley Values for a single observation

A shapley value explains a single feature value's contribution to the difference in prediction and average prediction. We calculate this contribution by taking the weighted average of the marginal contributions across all possible coalitions of features. Let us say we are interested in how the first observation's carat value of 0.23 contributes to the predicted square root price of 5.6199. To do this, we need to calculate a linear model trained on each coalition of predictor variables that includes carat, and a linear model trained on the same coalition but not including carat. We then take the difference between the predictions made by the carat including model and the model not including carat.(Molnar, 2019) We then repeat this process for all other coalitions of variables. Just to clarify, a coalition is any combination of features. The coalitions including carat would then include the following:

- carat
- carat, clarity

- carat, length
- carat, width
- carat, depth
- carat, clarity, length
- carat, clarity, width
- ...

The corresponding coalitions without carat would then be:

- intercept only
- clarity
- length
- width
- depth
- clarity, length
- clarity, width
- ...

We will refer to the set of all coalitions including carat as C, and any subset of these including carat as s and the subsets without carat as $s\backslash carat$. For each of these coalitions, we calculate a linear model with and without variable interaction terms referred to as $f_s(x)$ and $f_{s\backslash carat}(x)$.

For each coalition in S, we take the weighted sum of $f_s(x) - f_{s\backslash carat}(x)$. We will weight by the number of features in each coalition and the number of total features. The contribution $\phi_{carat}(f, x)$ is then the sum of all these weighted differences:

$$\phi_{carat}(f,x) = \sum_{s \subseteq C} \frac{|s|!(M-|s|-1)!}{M!}[f_s(x) - f_{s\backslash carat}(x)]$$

(Lundberg & Lee, 2017)

To calculate shapley values of my own, I created different models using all possible coalitions as predictor variables (code can be found in the appendix). Using this code, we can calculate the shapley values for each feature value of our diamond data observation.

For the observation mentioned at the end of section 1.1, the feature values are:

- carat = 0.23 carats
- clarity = SI2
- length = 3.95 millimeters
- width = 3.98 millimeters
- depth = 2.43 millimeters

We get the following shapley values for each feature.

- ϕ_{carat}: -0.685
- $\phi_{clarity}$: -0.672

- ϕ_{length}: -1.207
- ϕ_{width}: -1.162
- ϕ_{depth}: -1.108

The negative shapley values suggest that each feature value for this observation lead to a lower predicted price compared to the average predicted price of the dataset. As mentioned at the end of section 1.1 we see that the feature values for this single observation, carat = 0.23, length = 3.95, width = 3.98, depth = 2.43, are quite low relative to the rest of the dataset. We would then intuitively expect this diamond to have a lower price due to its smaller weight, size, and poor clarity. The shapley values calculated here confirm this intuition.

1.3 Calculating Shapley Values with IML Package

As seen in the above example, calculating shapley values requires the creation of and use of many models. To predict diamond price, I used 5 predictor variables. Even with this low number of predictors, since we have to create models with and without interaction terms for all possible coalitions, $2 * 2^5 = 64$ models had to be created. The number of models necessary to calculate shapley values increases exponentially the more predictor variables you have.

It is quite clear how calculating shapley values this way becomes impractical for datasets using even a moderate number of predictors. However, the IML package in R can help approximate shapley values without having to create multiple models.(Molnar, Bischl, & Casalicchio, 2018) Instead of training a new model for each coalition of predictor variables, IML uses a single model to make predictions and simply replaces the values of features not in the coalition with a randomly chosen value from the dataset.(Molnar, 2019)

For example, take the same observation from the diamond package mentioned earlier and let us say we are trying to calculate the marginal contribution of its carat value with the coalition length, width, and clarity:

- Actual Observation: carat = 0.23, clarity = SI2, length = 3.95, width = 3.98, depth = 2.43

When we calculate the prediction for the coalition including carat, we would feed the following “observation” to the full model.

- carat = 0.23, clarity = SI2, length = 3.95, width = 3.98

Since the depth predictor is not in our coalition, we use a randomly sampled value from the dataset:

- depth = 4.16

The full “observation” we feed the model is then:

- carat = 0.23, clarity = SI2, length = 3.95, width = 3.98 depth = 4.16

We do the same process for the coalition without the variable of interest, using a carat value randomly sampled from the dataset of 0.58:

- carat = 0.58, clarity = SI2, length = 3.95, width = 3.98 depth = 4.16

We then use the MLR model created earlier to predict the square root prices of these observations, which are 4.5591 and 17.554. The weighted marginal contribution for carat in this coalition is then the difference of $4.5591 - 17.554 = -12.995$. Since the contribution depends on randomly chosen values for features not in the coalition, this process is repeated and the average of these weighted marginal contributions becomes the final marginal contribution for this coalition. If the number of total coalitions is still too large to feasibly perform this calculation, this process can be made more efficient by randomly sampling from all possible coalitions and only calculating the marginal contribution of the feature of interest for these sampled coalitions.(Molnar, 2019)

1.4 Properties of the Shapley Value

The shapley value is the only method that can calculate a "fair payout" or appropriate marginal contributions. This is because the shapley value has the following properties

- **Efficiency**: the feature contributions must add up to the difference in prediction and average prediction

$\sum_{j=1}^{p} \phi_j = \hat{f}(x) - E_X(\hat{f}(x))$

- **Symmetry**: the contribution of two features should be the same if they contribute equally to all coalitions

let $f_s(x) - f_{s\setminus j}(x)$ and $f_s(x) - f_{s\setminus i}(x)$ be the marginal contributions of feature values j and i
if $f_s(x) - f_{s\setminus j}(x) = f_s(x) - f_{s\setminus i}(x)$ for all $s \subseteq coalitions$
then $\phi_j = \phi_i$

- **Dummy**: a feature value that does change the predicted value has a shapley value of 0

if $f_s(x) = f_{s\setminus j}(x)$ for all $s \subseteq coalitions$
then $\phi_j = 0$

- **Additivity**: if we had two different attribution problems involving the same features but different marginal contribution functions as follows:

ϕ_1 and ϕ_2

We can then model the two problems into one problem such that the marginal contribution function is in the form:

$\phi_1 + \phi_2$

The additivity property of shapley values is very helpful for ensemble models. For example, if we are working with a randomforest model, we could calculate the shapley values of each feature for each tree in the forest and then take the average to get the shapley value for the entire randomforest. (Lundberg & Lee, 2017) (Molnar, 2019)

These describe properties make shapley values the only explanation method that has a solid theoretical background. There are other methods such as local interpretable model-agnostic explanations (LIME) which do not guarantee a fair "payout" to each feature value and also does not have any theory as to why the method works. This makes shapley values the best tool to improve model prediction interpretability. (Molnar, 2019)

Chapter 2

Applying Shapley Values

2.1 King's County Housing Data

To demonstrate the IML package and how shapley values work for larger datasets, I will use shapley values to better understand the multiples models I have trained to predict King's County Washington housing prices. The data can be found here `https://www.kaggle.com/shivachandel/kc-house-data`, and the variables in the dataset are as follows:

- Date: Date of the home sale
- Price: price of the home sold
- Bedrooms: number of bedrooms
- Bathrooms: number of bathrooms (0.5 is a bathroom with a toilet and no shower)
- Sqft_living: square footage of the apartment's interior living space
- Sqft_lot: square footage of the land space
- Floors: number of floors
- Waterfront: whether an apartment is overlooking the waterfront or not
- View: an index from 0 to 4 indicating how good the view from the property was
- Condition: an index from 1 to 5 indicating how good condition the property was in
- Grade: index from 1 to 13 where 1-3 falls short of building construction and design, 7 has an average level of construction and design, 11-13 has a high level of construction and design
- Sqft_above: the square footage of the interior space that is above ground level
- Sqft_basement: the square footage of the interior space that is below ground level
- Yr_built: the year the house was build
- Yr_renovated: the year of the house's last renovation
- Zipcode: zipcode of the house
- Lat: latitude of the house
- Long: longitude of the house
- Sqft_living15: the square footage of interior living space for the nearest 15

neighbors
- Sqft_lot15: the square footage of land lots for the nearest 15 neighbors

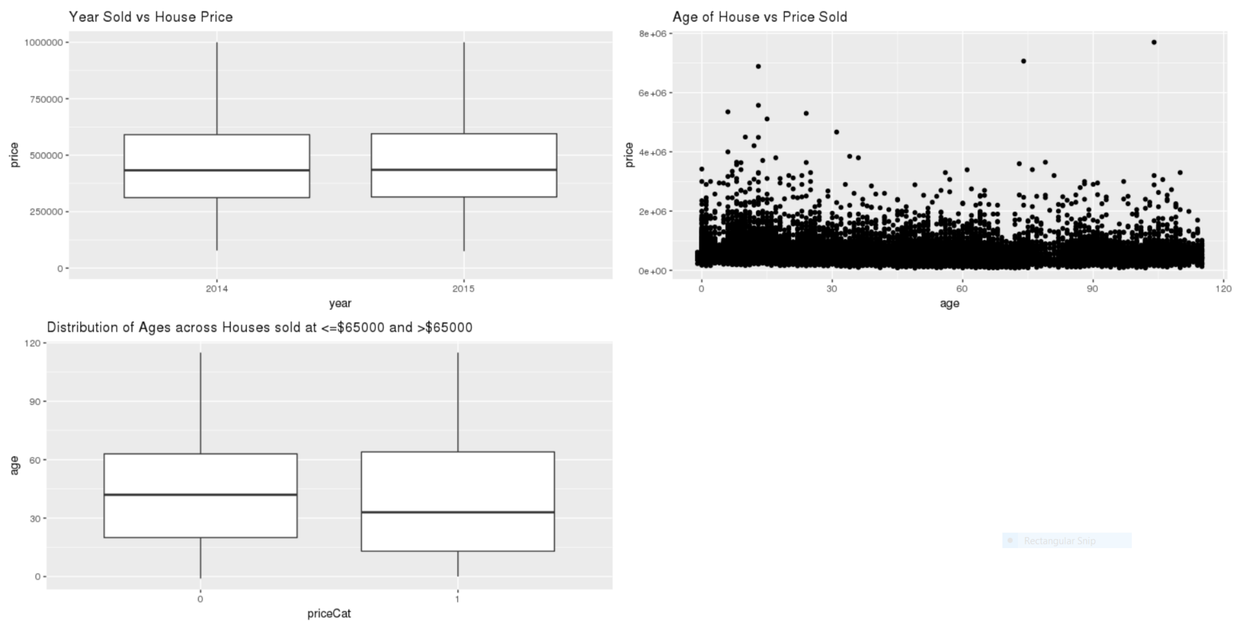


Figure 2.1: Relationships between age of home and year sold with price the home was sold at

In figure 2.1 we can look at the the distribution of house price across year sold (all homes were sold in either 2014 or 2015) and the relationship between age of a home and its house price. We can see that the distribution of house prices across 2014 and 2015 is quite similar, and when we take a look at the how age relates to house price we do not see any significant linear relationship. Furthermore, when we look at the distribution of house ages for homes sold for more that \$650000 and homes sold for less than or equal to \$650000, the distributions are also appear quite similar. This would lead us to believe that age of a home may not be a strong indicator of the home's price.

In figure 2.2 we can look at the distribution of house price across homes with and without waterfront views, different levels of view quality, different conditions and grades, and homes that have been renovated or not. We can see homes with a waterfront view, on average, are sold at higher prices than homes without a waterfront view. Furthermore, although we see that homes with view qualities of 1, 2, and 3 have a similar distribution of house price, homes with the highest quality of views (4) appear to be sold for significantly higher prices. Across different levels of condition, homes with the best condition (5) have the highest average house price compared to the other condition levels. However due to the large amount of spread in each boxplot, there is a lot of overlap between these distributions across the condition levels. This leads us to believe that house condition may not be a strong predictor for house price. We also see a lot of overlap between the distribution of house price for unrenovated and renovated homes. While renovated homes on average fetch a higher price, there

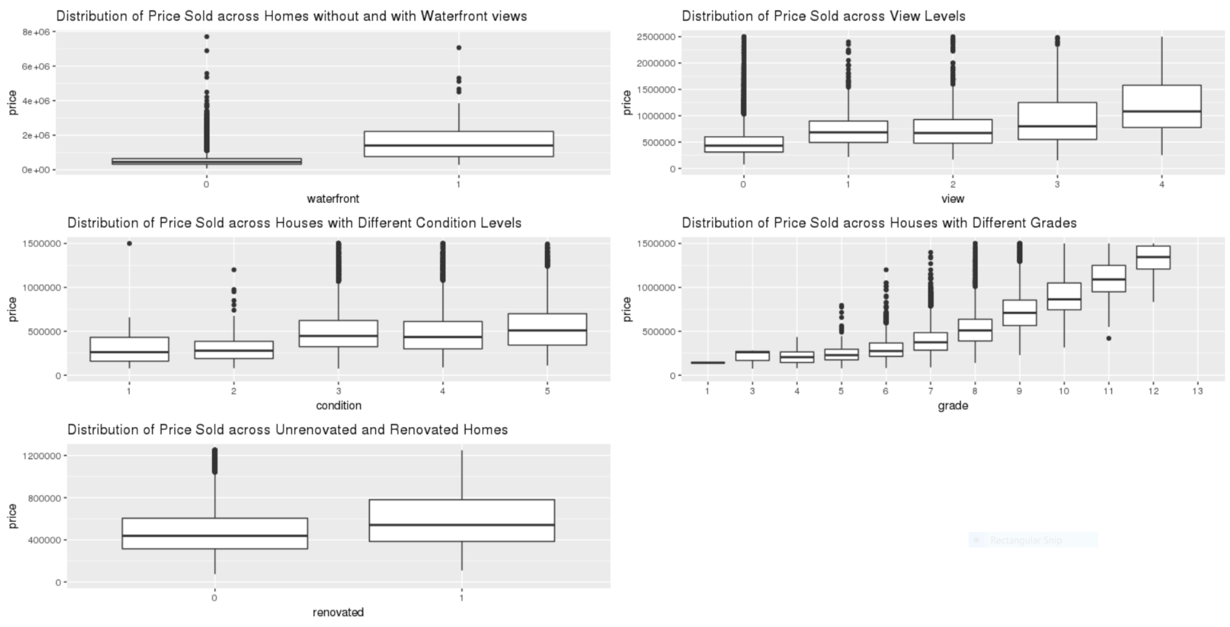


Figure 2.2: Distribution of house price across different levels of waterfront, view, condition, grade, and renovated

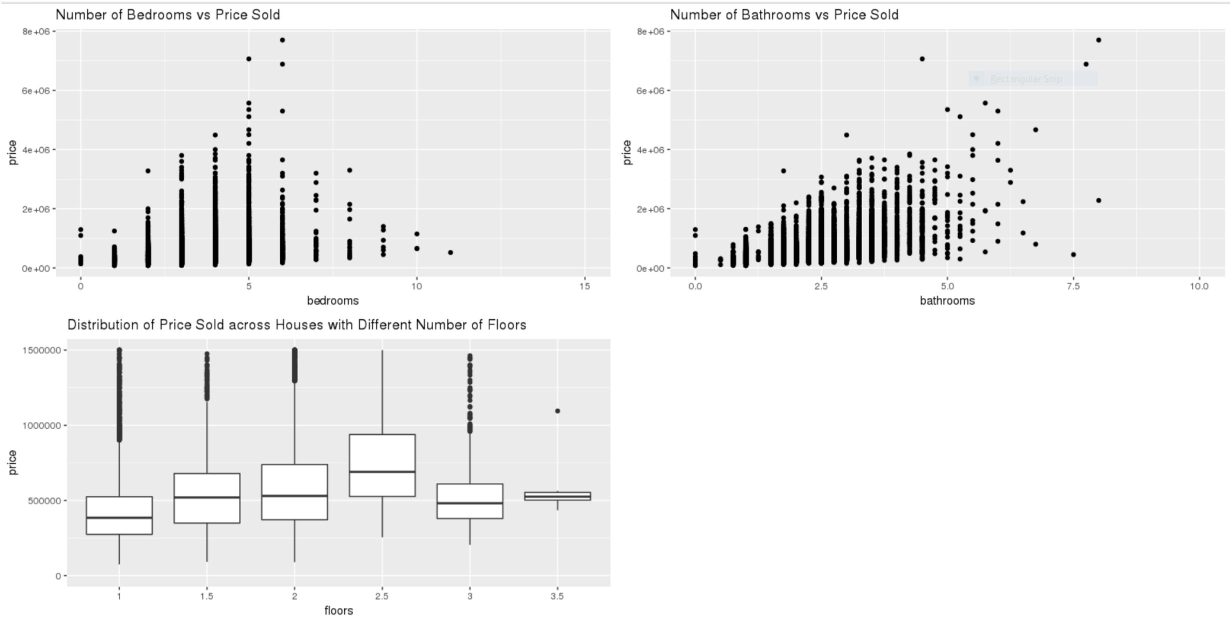


Figure 2.3: Distribution of house price across homes with different number of bedrooms, bathrooms, and floors

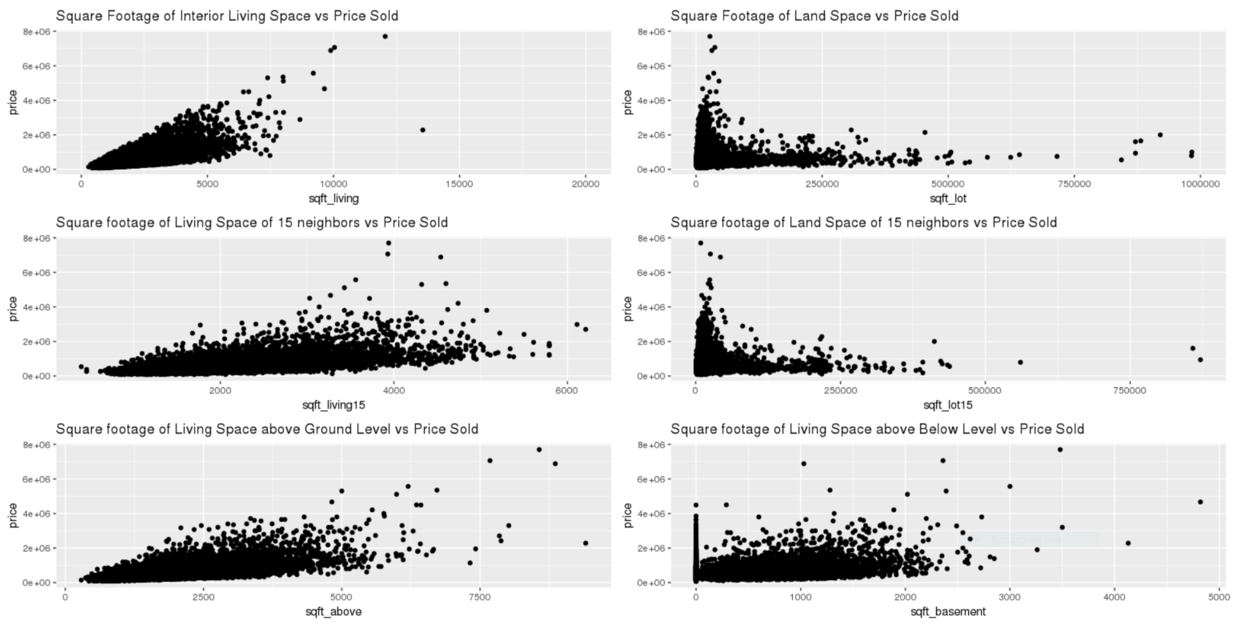


Figure 2.4: Distribution of house price across different levels of waterfront, view, condition, grade, and renovated

is not as big of a difference between the distributions as initially expected. Grade appears to be a stronger predictor for house price. Homes with higher grades have a much higher average price sold than homes with lower grades.

In figure 2.3 we can see how the number of bedrooms, bathrooms, and floors of a home relate to its price. In all three graphs, there does not appear to be a strong relationship. There may be a very slight positive relationship between the number of bedrooms or bathrooms with house price, but these variables do not appear to be strong predictors of price. Looking at the distributions of house prices across homes with different numbers of floors, we find something unexpected. Homes with 2.5 floors appear to cost the most on average and cost significantly more than homes with 1, 3, and 3.5 floors. We would expect homes with more floors to be bigger and thus tend to cost more, but this does not appear to always be the case.

In figure 2.4 we look at how the square footage of different parts of a home relates to the home's price. Across all different measures of square footage, sqft_living, sqft_living15, and sqft_above, we see a slight positive linear relationship. For sqft_lot, sqft_lot15, and sqft_basement, we see a peak followed by a downward curve. Besides these strange relationships, we generally see that homes with larger living spaces fetch higher prices.

From the 2-dimentional density plot in figure 2.5, we can see that the more expensive houses (with price $>$ \$650000) tend to be above latitude 47.5. Less expensive houses (price $\leq$ \$650000) a more widely distributed. We can use this information to create a new predictor variable latThresh.

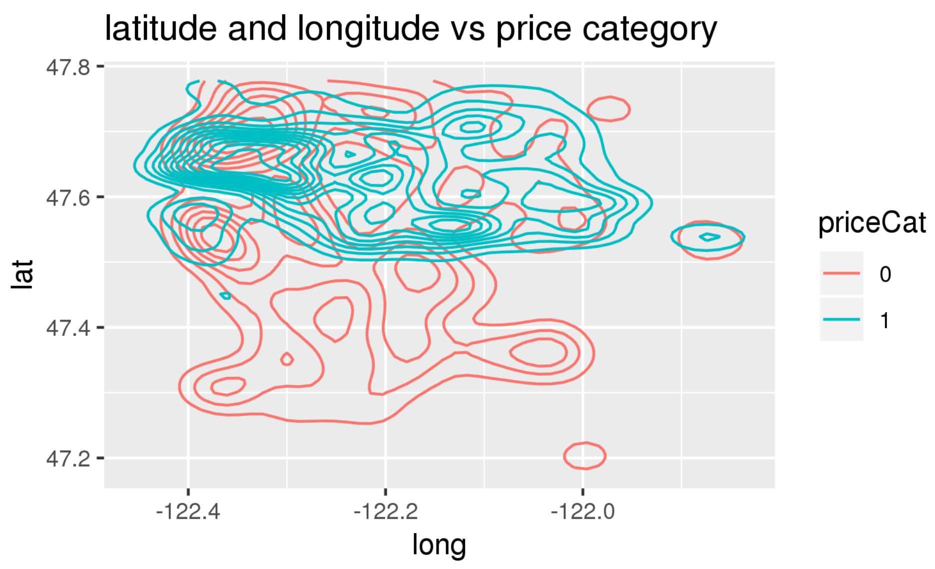


Figure 2.5: Density of expensive and less expensive houses by lat and long

2.2 Creating our Models

The first thing to notice when creating models to predict house price is the fact that there are much fewer expensive houses (price > $650000).

```
summary(data$priceCat)
```

```
    0     1
16406  5207
```

Because of this, when creating the training data we will sample $\frac{2}{3}$ of the observations from each price category. After creating out training and test sets, we will use the caret package to tune our randomforest models.

For the housing data I am primarily interested in predicting two things, price of the house sold and whether or not the house was sold for $650000 or more. To do this I created two randomforest models, one that predicted the continuous variable price and one to predict the categorical variable priceCat (1 if house price > $650000, else 0).

```
set.seed(1337)
dataCleanedCat = data[, -match(c("id", "date", "zipcode", "yr_built",
    "yr_renovated", "lat", "long"), colnames(data))]
dataCleanedCat$year = as.factor(dataCleanedCat$year)
lowerPriceData = dataCleanedCat %>% dplyr::filter(priceCat ==
    0)
higherPriceData = dataCleanedCat %>% dplyr::filter(priceCat ==
    1)

# dealing with very unbalanced dataset, have to make sure to
# get enough of the less represented category in the
# trainingset
trainLowerIndices = sample(nrow(lowerPriceData), 0.66 * nrow(lowerPriceData))
trainHigherIndices = sample(nrow(higherPriceData), 0.66 * nrow(higherPriceData))

trainLower = lowerPriceData[trainLowerIndices, ]
trainHigher = higherPriceData[trainHigherIndices, ]
testLower = lowerPriceData[-trainLowerIndices, ]
testHigher = higherPriceData[trainHigherIndices, ]

train = rbind(trainLower, trainHigher)
trainCat = train[, -match("price", colnames(train))]
trainCont = train[, -match("priceCat", colnames(train))]
test = rbind(testLower, testHigher)
testCat = test[, -match("price", colnames(test))]
testCont = test[, -match("priceCat", colnames(test))]
```

```
# create own RF function so we can optimize mtry and ntree
customRF <- list(type = "Classification", library = "randomForest",
    loop = NULL)
customRF$parameters <- data.frame(parameter = c("mtry", "ntree"),
    class = rep("numeric", 2), label = c("mtry", "ntree"))
customRF$grid <- function(x, y, len = NULL, search = "grid") {
}
customRF$fit <- function(x, y, wts, param, lev, last, weights,
    classProbs, ...) {
    randomForest(x, y, mtry = param$mtry, ntree = param$ntree,
        ...)
}
customRF$predict <- function(modelFit, newdata, preProc = NULL,
    submodels = NULL) predict(modelFit, newdata)
customRF$prob <- function(modelFit, newdata, preProc = NULL,
    submodels = NULL) predict(modelFit, newdata, type = "prob")
customRF$sort <- function(x) x[order(x[, 1]), ]
customRF$levels <- function(x) x$classes

control <- trainControl(method = "repeatedcv", number = 10, repeats = 3,
    search = "grid")
tunegrid = expand.grid(.mtry = c(1:10), .ntree = c(450, 500,
    550, 600))
```

```
rfCat = train(priceCat ~ ., data = trainCat, method = customRF,
    tuneGrid = tunegrid, trControl = control)
rf = train(price ~ ., data = trainCat, method = customRF, tuneGrid = tunegrid,
    trControl = control)
```

For both of the models, I used the caret package to determine optimal hyperparameters for the model. The optimal mtry parameters found for each model was 5 for the predicting price, and 4 for predicting priceCat. Both models have an nTree value of 500. The MSE for the continuous predictor model is 11672358367 and the accuracy for the categorical model is 97.58%.

```
#somewhere waterfront was converted into a numeric had to fix this
testCont$waterfront = as.factor(testCont$waterfront)
testCat$waterfront = as.factor(testCat$waterfront)
testCont$floors = as.factor(testCont$floors)
testCat$floors = as.factor(testCat$floors)

rfCont$mtry
```

```
[1] 5
```

```
rfCat$mtry
```

```
[1] 4
```

```
mean((predict(rfCont, testCont)-testCont$price)^2)
```

```
[1] 11672358367
```

```
table(predict(rfCat, testCat), testCat$priceCat)
```

```

        0    1
  0 5360    0
  1  219 3436
```

```
mean(predict(rfCat, testCat)==testCat$priceCat)
```

```
[1] 0.975818081
```

2.3 Using IML Shapley Values

It is important to reiterate the interpretation of shapley values once again. For some feature value j, ϕ_j represents their shapley value. The feature value j contributed ϕ_j to the prediction of its observation compared to the average prediction across the dataset.

The figure 2.6 shows the shapley values (labelled phi on the x-axis) for each feature of an observation in the housing dataset. At the top of the plot we can see the actual prediction our randomforest model made for this observation, as well as the average of all predictions made on the entire housing dataset. On the y-axis we can see each predictor variable as well as their corresponding value for this observation. This observation is a home that has a latThresh value of 1, age of 59 years, 1 floor, sqftlot15 of 5650 square feet, sqftlot of 5650, no waterfront view, no renovations, lowest quality of view, 3 bedrooms, sold in the year 2014, condition of 3, sqftbasement of 0 square feet (no basement), 1 bathroom, sqftabove of 1180 square feet, sqftliving15 of 1340 square feet, grade of 7, and sqftliving of 1180 square feet.

What the phi value tells us is that, all feature values with phi value less than 0 had a negative contribution and lowered the predicted price;conversely, all phi values greater than 0 had a positive contribution and increased the predicted price.(Molnar, 2019) For example, the home's sqftliving value of 1180 square feet decreased the predicted price by \$85550 compared to the average predicted price of \$540968.59. Looking at the distribution of sqftliving values across the housing dataset, this home's sqftliving of 1180 is below the 1st quartile of all home's sqftliving, which is 1427 square feet. Thus this home would be considered a smaller home and this intuitively would have us believe that this would cause the home to cost less. The shapley value here captures this intuition. We see that the sum of all the shapley values describes the total difference between the prediction and average prediction, $\sum_{j\subseteq features}\phi_j = 248172.43 - 540968.59$.

In figure 2.7 we can see how shapley values work for models predicting categorical variables. We now have 2 graphs showing the probability that an observation is either in category 0, $price \leq \$650000$, or category 1, $price > \$650000$.Instead of contributions to a numerical prediction, feature values contribute to a probability of being within a certain category. We see that most of the example observation's feature values contribute a positive phi value for category 0 and negative phi value for category 1. Each of these feature values then increase the probability of the observation being in category 0 and decrease the probability of the observation being in category 1. Again, the sqftliving for this home is 1180 square feet which is relatively small. We then see that this feature value decreases the probability that the home costs more than \$650000 and increases the probability that the home costs less than or equal to \$650000.

After calculating shapley values for a large sample of observations in the housing dataset, we can get a better idea of the contributions of each feature to the prediction of housing price. In the tables 2.1. and 2.2 we can see that the predictor variables with the greatest contributions (average of the absolute value of their shapley values) to house price are latThresh, grade, sqftliving, and age. The predictor variables with the greatest contributions to predicted price category ($> \$650000 or \leq \$650000$) are

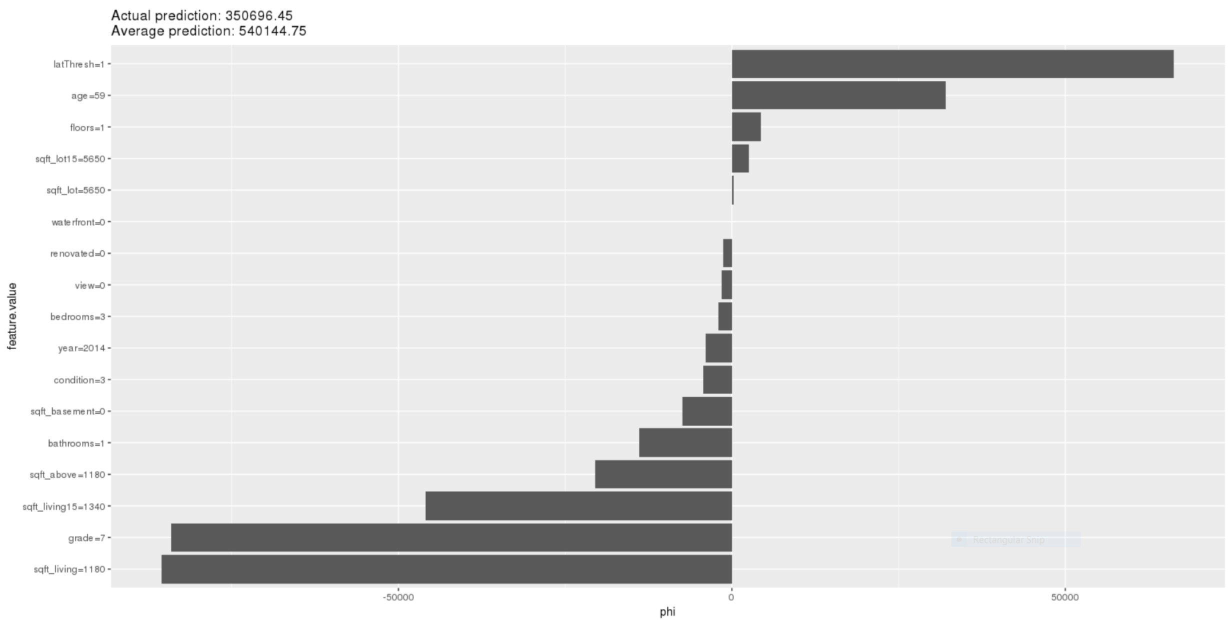


Figure 2.6: Feature Value Contributions to Predicted House Price for A Single Observation

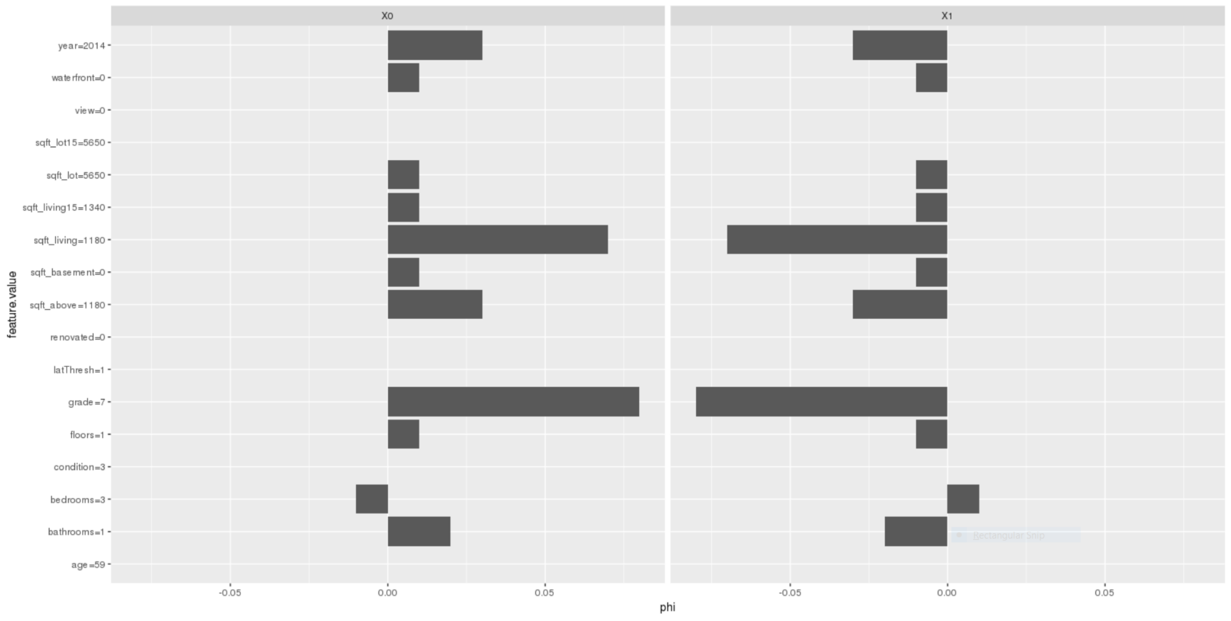


Figure 2.7: Feature Value Contributions to Predicted House Price Category (>650000*or* <=650000) for A Single Observation

grade, latThresh, sqftliving, and age. Although it is somewhat surprising to see age as one of the most contributing predictor variables, since there did not appear to be a strong relationship between age and house price in our data exploration, the other top 3 predictor variables match our initial findings as well as our intuition about the dataset. Home size, location, and grade (construction quality) are all expected to greatly influence the price of a home.

	feature	aveAbsShap
1	latThresh	84006.10
2	grade	73075.63
3	sqft_living	61451.27
4	age	38550.82
5	sqft_living15	33025.58
6	sqft_above	24302.07
7	condition	11439.19
8	sqft_lot15	11139.27
9	view	9860.34
10	bathrooms	8621.50
11	sqft_basement	7057.55
12	sqft_lot	6476.98
13	year	3623.03
14	floors	2923.12
15	renovated	2292.26
16	waterfront	2068.81
17	bedrooms	1741.82

Table 2.1: Mean Absolute Contribution of Predictors for Model Predicting Price

When we take a look at the variable importance of the randomforest model in figure 2.8, we can see that sqft_living, latThresh, and grade are also among the top most important variables. The similarity between the most important variables according to Gini index and Shapley Values shows us that feature contribution can be a good measurement of variable importance if calculated for the entire dataset. There are in fact methods that perform feature selection using shapley values, but such methods go beyond the scope of this report.

Shapley values also let us explore what kind of feature values contribute to higher or lower house prices. For instance, when we separate sqft_living values by whether they have a positive or negative shapley value, we can get an idea of what sized home costs more.

In figure 2.10 we see that the distribution for sqft_living values with negative shapley values is centered around 1500 square feet while the distribution for sqft_living values with positive shapley values achieves its peak around 2750 square feet. The two distributions have a bit of overlap, but in general we see that homes with sqftliving larger than 2000 square feet will fetch a higher price than homes with sqftliving less

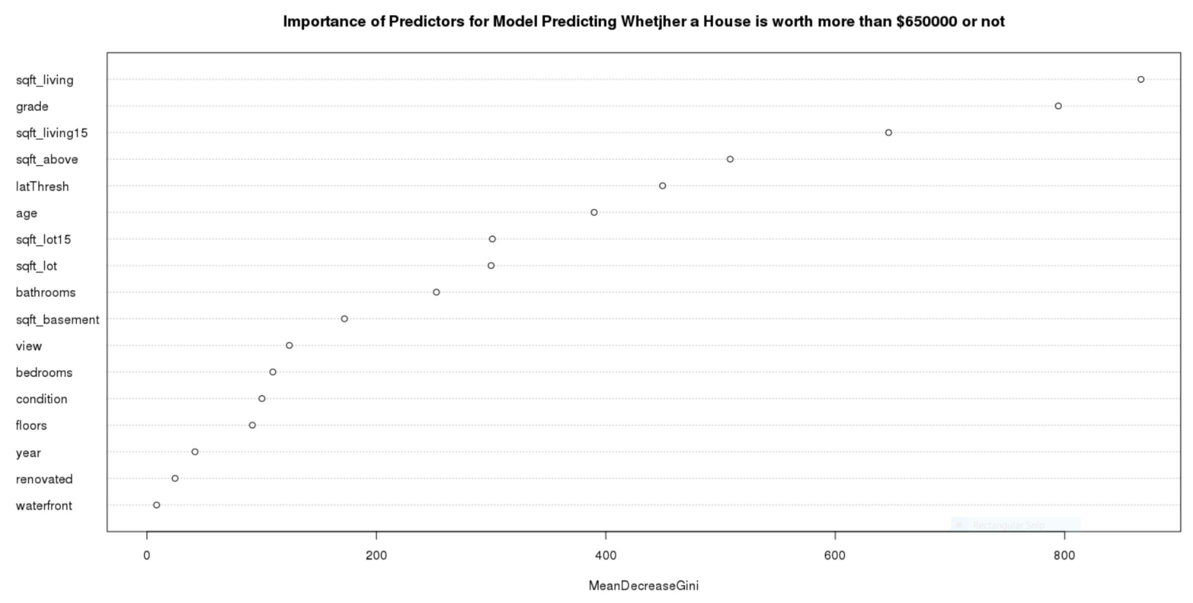


Figure 2.8: Categorical Prediction Model Variable Importance

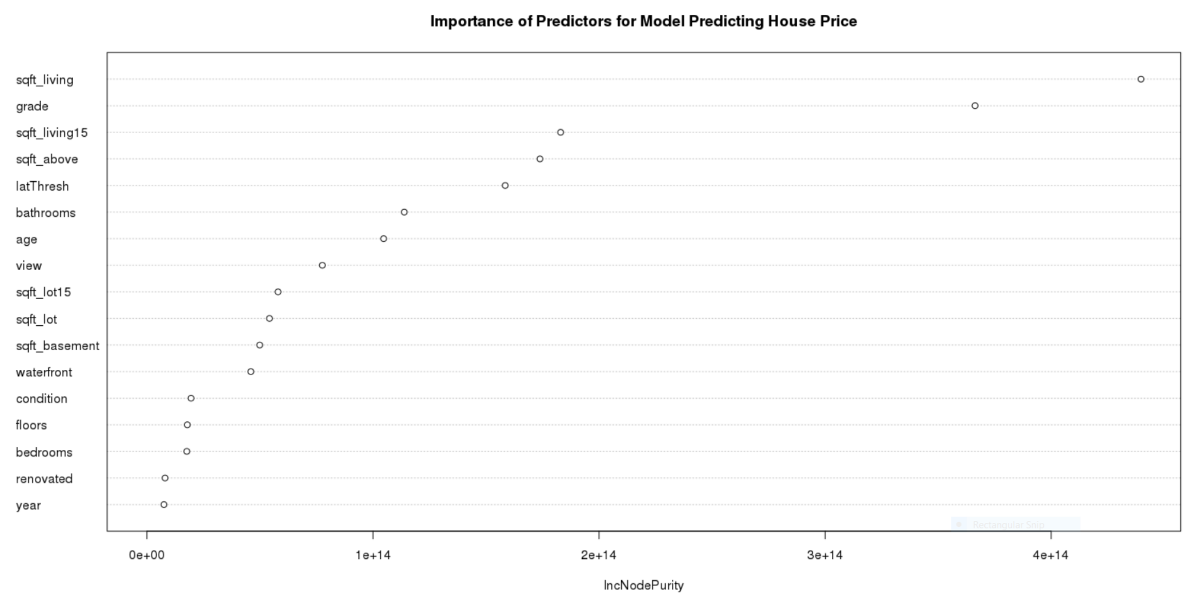


Figure 2.9: Quantitative Prediction Model Variable Importance

	feature	aveAbsShap
1	grade	0.05
2	latThresh	0.04
3	sqft_living	0.03
4	age	0.03
5	sqft_living15	0.02
6	sqft_above	0.02
7	sqft_basement	0.01
8	sqft_lot15	0.01
9	condition	0.01
10	sqft_lot	0.01
11	bathrooms	0.01
12	view	0.01
13	renovated	0.01
14	floors	0.01
15	bedrooms	0.01
16	year	0.00
17	waterfront	0.00

Table 2.2: Mean Absolute Contribution of Predictors for Model Predicting Whether or not a House is worth more than 650000 Dollars

than 2000 square feet. Again, this observation from shapley values matches the basic intuition that larger homes are typically more expensive than smaller homes.

We can also affirm our intuition about the prices of homes with better views. In figure 2.11 we see that the only view level that decreased an observation's prediction compared to the average prediction were views of level 0. However when we look at all the view values with a positive contribution, we find that view levels 1 through 4 increased the predicted home price compared to the average predicted price. This would suggest that having a view increases the price of a home, again, supporting basic intuition.

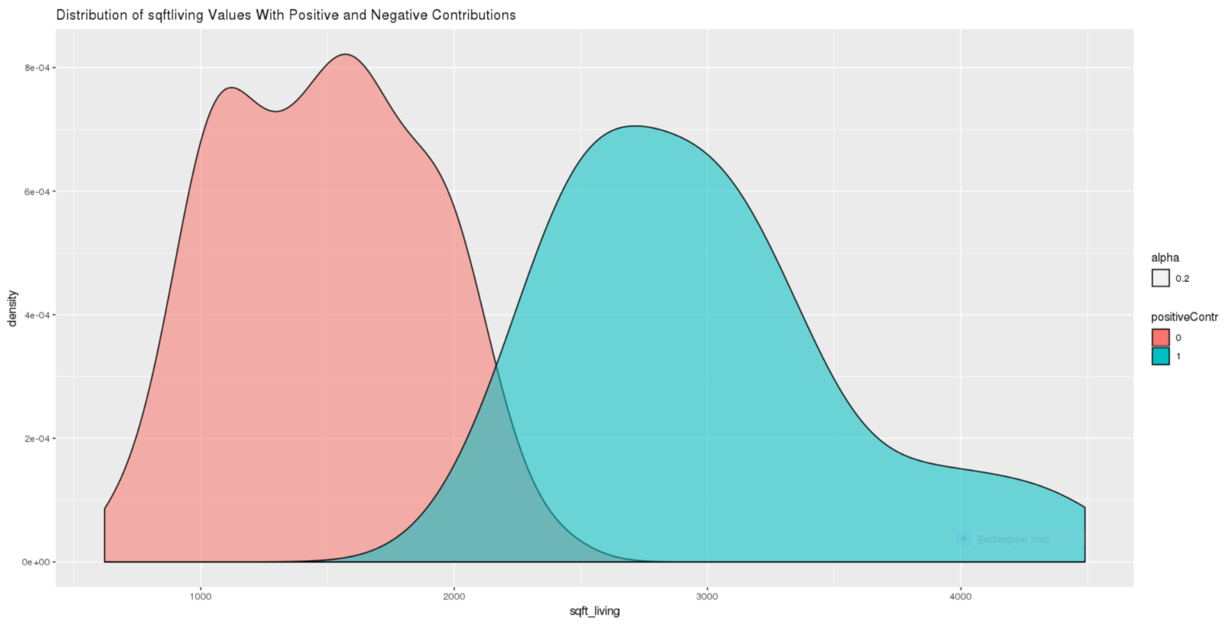


Figure 2.10: comparison of sqftliving values with negative vs positive shapley values (blue mean positive contribution, red means negative contribution)

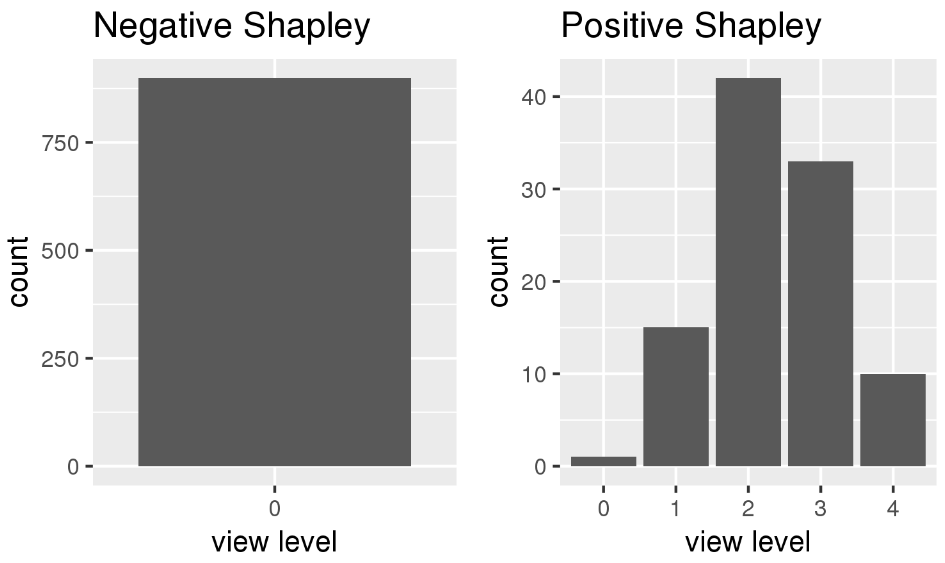


Figure 2.11: comparison of view levels with negative vs positive shapley values

Conclusion

In this project I have covered what shapley values are, how they can be calculated as well as approximated, and have shown how they can be used to understand a model's predictions. To do this, I created my own code to generate linear models trained on all possible groupings of predictor variables and used these models to calculate marginal contributions for the diamond dataset's predictors. I then took the weighted average of these marginal contributions to get an observation's shapley values. I also did a study making use of the IML package to understand what factors lead to higher and lower house prices in King's County, Washington. This study demonstrated the shapley values' ability to understand the factors that lead to an individual prediction, but also what values in general may be associated with certain factors. For example, the shapley values confirmed our intuition that factors, such as house size and whether or not the home has a view, correspond with a home having a greater selling price.

Because of their strong theoretical backing, ease of interpretation, and ability to match human intuition; shapley values are a great tool for improving readability of complex model predictions. Their application can help make machine learning systems more transparent and improve people's understandings of such systems.

Appendix A

Appendix

In Chapter 1:

In order to calculate my own shapley values I had to create code to automatically create all of the models I needed, as well as perform all of the calculations for marginal contribution. The code I created is replicated here.

```r
diamondData = diamondData[-c(1, 3, 4, 6, 7)]
diamondData$clarity = as.factor(diamondData$clarity)
colnames(diamondData) = c("carat", "clarity", "price", "length",
    "width", "depth")
diamondData = filter(diamondData, width != 0 & length != 0 &
    depth != 0 & width <= 20)

# x, y, and z variables appear to have an exponential
# relationship with price, so we transformed the price
# variable
diamondData$transPrice = (diamondData$price)^(1/4)
```

```r
predictors = c("carat", "clarity", "length", "width", "depth")

# creates linear regression models using all combinations of
# variables in predictors with and without a specified
# variable of interest @param interestedVariable: your chosen
# variable of interest, this variable will not be included in
# half of the models output: list of lists each list within
# the overall output contains 4 elements: model without
# variable of interest, number of predictors in that model,
# model with variable of interest, and number of predictors
# in that model
getModels <- function(interestedVariable) {
    interestVar = interestedVariable
    models = list()
```

```r
preds = predictors[!grepl(interestVar, predictors)]
noVarComb1 = combn(preds, 1)
noVarComb2 = combn(preds, 2)
noVarComb3 = combn(preds, 3)
noVarComb4 = combn(preds, 4)

# create formulas with and without variable of interest
formulaWithVar = paste("transPrice~", interestVar, sep = "")
model = lm(transPrice ~ 1, data = diamondData)
modelWithVar = lm(formulaWithVar, data = diamondData)
models[[length(models) + 1]] = list(model, 0, modelWithVar,
    1)
for (h in 1:ncol(noVarComb1)) {
    formula = paste(noVarComb1[, h], collapse = "+")
    formulaWithVar = paste(interestVar, formula, sep = "+")

    formula = as.formula(paste("transPrice~", formula, sep = ""))
    formulaWithVar = as.formula(paste("transPrice~", formulaWithVar,
        sep = ""))

    model = lm(formula, data = diamondData)
    modelWithVar = lm(formulaWithVar, data = diamondData)
    models[[length(models) + 1]] = list(model, 1, modelWithVar,
        2)
}
for (i in 1:ncol(noVarComb2)) {
    formula = paste(noVarComb2[, i], collapse = "+")
    formulaWithVar = paste(interestVar, formula, sep = "+")

    formula = as.formula(paste("transPrice~", formula, sep = ""))
    formulaWithVar = as.formula(paste("transPrice~", formulaWithVar,
        sep = ""))

    model = lm(formula, data = diamondData)
    modelWithVar = lm(formulaWithVar, data = diamondData)
    models[[length(models) + 1]] = list(model, 2, modelWithVar,
        3)
}
for (j in 1:ncol(noVarComb3)) {
    formula = paste(noVarComb3[, j], collapse = "+")
    formulaWithVar = paste(interestVar, formula, sep = "+")

    formula = as.formula(paste("transPrice~", formula, sep = ""))
    formulaWithVar = as.formula(paste("transPrice~", formulaWithVar,
            sep = ""))

        model = lm(formula, data = diamondData)
        modelWithVar = lm(formulaWithVar, data = diamondData)
        models[[length(models) + 1]] = list(model, 3, modelWithVar,
            4)
    }

    formula = paste(preds, collapse = "+")
    formulaWithVar = paste(interestVar, formula, sep = "+")

    formula = as.formula(paste("transPrice~", formula, sep = ""))
    formulaWithVar = as.formula(paste("transPrice~", formulaWithVar,
        sep = ""))

    model = lm(formula, data = diamondData)
    modelWithVar = lm(formulaWithVar, data = diamondData)
    models[[length(models) + 1]] = list(model, 4, modelWithVar,
        5)
    return(models)
}

# creates all possible first order interaction formulas given
# a list of variable names @param: list of predictor
# variables you want to create formulas with output: a string
# that can be used within a formula that contains all first
# order interactions example input: width, length, depth
# example output: width*length + width*depth + length*depth
createInteractionTerms <- function(predictorVars) {
    endLoop = length(predictorVars)
    interactions = list()

    if (length(predictorVars) <= 1) {
        return(predictorVars[[1]])
    }
    for (i in 1:(endLoop - 1)) {
        for (j in i:endLoop) {
            if (!grepl(predictorVars[[i]], predictorVars[[j]])) {
                interactions[[length(interactions) + 1]] = paste(predictorVars[[i]
                  predictorVars[[j]], sep = "*")
            }
        }
    }
    return(paste(interactions, collapse = "+"))
```

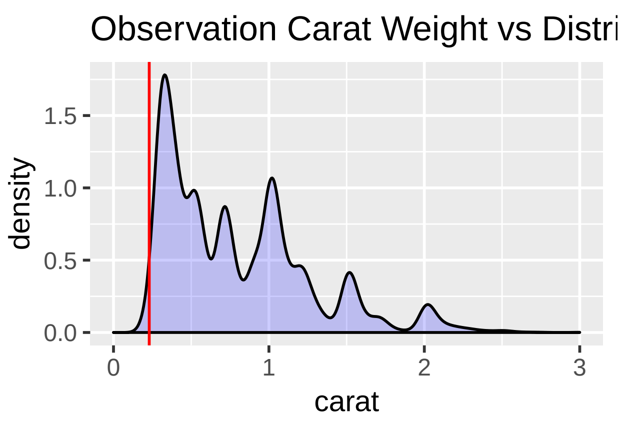
```
}

# creates interaction terms between a given variable and a
# list of other variables @param interested: predictor
# variable you want to calculate shapley value for @param
# otherVars: list of other predictor variables in the
# coalition output: a string that can be used within a
# formula that contains all first order interactions between
# interested variable and other predictor variables
createInteractionInterested <- function(interested, otherVars) {
    endLoop = length(otherVars)
    otherInteractions = createInteractionTerms(otherVars)

    interactions = list()
    for (i in 1:endLoop) {
        interactions[[length(interactions) + 1]] = paste(interested,
            otherVars[[i]], sep = "*")
    }
    return(paste(paste(interactions, collapse = "+"), otherInteractions,
        sep = "+"))
}

# create all models containing all possible combinations of
# predictors' interaction terms only uses first order
# interactions since higher order interactions are unlikely
# @param interestedVariable: your chosen variable of
# interest, this variable will not be included in half of the
# models output: list of lists each list within the overall
# output contains 4 elements: model without variable of
# interest, number of predictors in that model, model with
# variable of interest, and number of predictors in that
# model
getInteractionModels <- function(interestedVariable) {
    interestVar = interestedVariable
    models = list()

    preds = predictors[!grepl(interestVar, predictors)]
    noVarComb1 = combn(preds, 1)
    noVarComb2 = combn(preds, 2)
    noVarComb3 = combn(preds, 3)
    noVarComb4 = combn(preds, 4)

    # create formulas with and without variable of interest
    for (i in 1:ncol(noVarComb1)) {
        formula = noVarComb1[, i]
        formulaWithVar = createInteractionInterested(interestVar,
            noVarComb1[, i])

        formula = as.formula(paste("transPrice~", formula, sep = ""))
        formulaWithVar = as.formula(paste("transPrice~", formulaWithVar,
            sep = ""))

        model = lm(formula, data = diamondData)
        modelWithVar = lm(formulaWithVar, data = diamondData)
        models[[length(models) + 1]] = list(model, 1, modelWithVar,
            2)
    }
    for (i in 1:ncol(noVarComb2)) {
        formula = createInteractionTerms(noVarComb2[, i])
        formulaWithVar = createInteractionInterested(interestVar,
            noVarComb2[, i])

        formula = as.formula(paste("transPrice~", formula, sep = ""))
        formulaWithVar = as.formula(paste("transPrice~", formulaWithVar,
            sep = ""))

        model = lm(formula, data = diamondData)
        modelWithVar = lm(formulaWithVar, data = diamondData)
        models[[length(models) + 1]] = list(model, 2, modelWithVar,
            3)
    }
    for (j in 1:ncol(noVarComb3)) {
        formula = createInteractionTerms(noVarComb2[, i])
        formulaWithVar = createInteractionInterested(interestVar,
            noVarComb2[, i])

        formula = as.formula(paste("transPrice~", formula, sep = ""))
        formulaWithVar = as.formula(paste("transPrice~", formulaWithVar,
            sep = ""))

        model = lm(formula, data = diamondData)
        modelWithVar = lm(formulaWithVar, data = diamondData)
        models[[length(models) + 1]] = list(model, 3, modelWithVar,
            4)
    }

    formula = createInteractionTerms(noVarComb4)
    formulaWithVar = createInteractionInterested(interestVar,
        noVarComb4)

    formula = as.formula(paste("transPrice~", formula, sep = ""))
    formulaWithVar = as.formula(paste("transPrice~", formulaWithVar,
        sep = ""))

    model = lm(formula, data = diamondData)
    modelWithVar = lm(formulaWithVar, data = diamondData)
    models[[length(models) + 1]] = list(model, 4, modelWithVar,
        5)
    return(models)
}

# create linear models from all possible coalitions with and
# without first order interaction effects @param
# interestedVariable: variable you want to calculate shapley
# value for output: list of lists each list within the
# overall output contains 4 elements: model without variable
# of interest, number of predictors in that model, model with
# variable of interest, and number of predictors in that
# model
getAllShapModels <- function(interestedVariable) {
    noInteractionModels = getModels(interestedVariable)
    models = getInteractionModels(interestedVariable)

    for (i in 1:length(models)) {
        models[[length(models) + 1]] = noInteractionModels[[i]]
    }
    return(models)
}
```

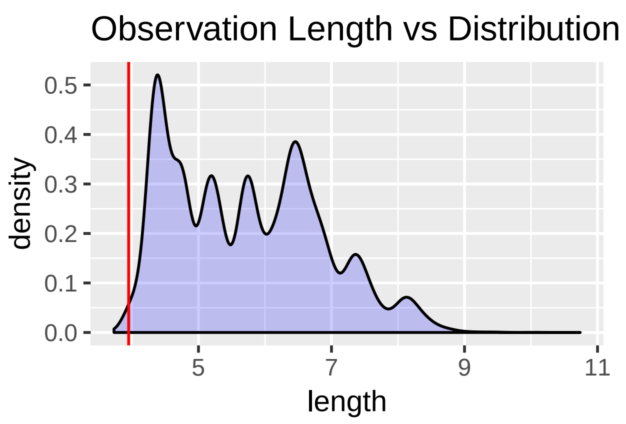
```
# code used to generate appropriate models
caratModels = getAllModels("carat")
clarityModels = getAllModels("clarity")
lengthModels = getAllModels("length")
widthModels = getAllModels("width")
depthModels = getAllModels("depth")

# average transformed price of the dataset
meanTransPrice = mean(diamondData$transPrice)

# calculates shapley values for a given observation feature
# value (which feature value is specified in the models)
# @param observation: observation you would like to calculate
# shapley values for @param models: list of all models that
# will be used to calculate shapley values (use output of
# getAllModels() function) output: shapley value for the
# given observation's feature value
calculateShapley <- function(observation, models) {
    output = 0
    for (i in 1:length(models)) {
        modelWithIntVar = models[[i]][[3]]
        model = models[[i]][[1]]

        z = models[[i]][[2]]
        M = 5
        multiplier = factorial(z) * factorial(M - z - 1)/factorial(M)
        predWithIntVar = predict(modelWithIntVar, observation)
        pred = predict(model, observation)
        marginalContribution = multiplier * (predWithIntVar -
            pred)
        output = output + marginalContribution
    }
    return(output)
}
```

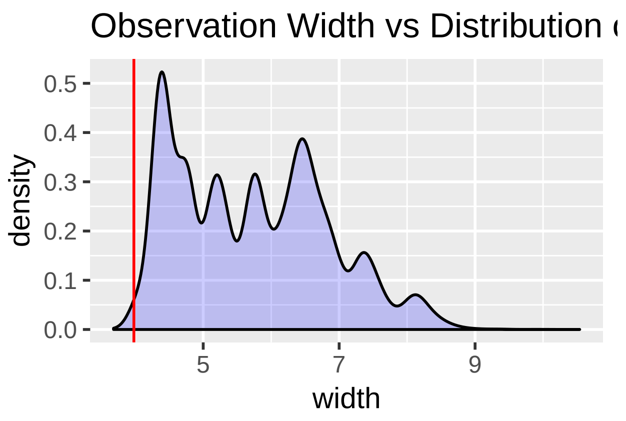
```
# code to calculate first observation's shapley values
caratShapley = calculateShapley(diamondData[1, ], caratModels)
clarityShapley = calculateShapley(diamondData[1, ], clarityModels)
lengthShapley = calculateShapley(diamondData[1, ], lengthModels)
widthShapley = calculateShapley(diamondData[1, ], widthModels)
depthShapley = calculateShapley(diamondData[1, ], depthModels)

caratShapley
clarityShapley
lengthShapley
widthShapley
depthShapley
```

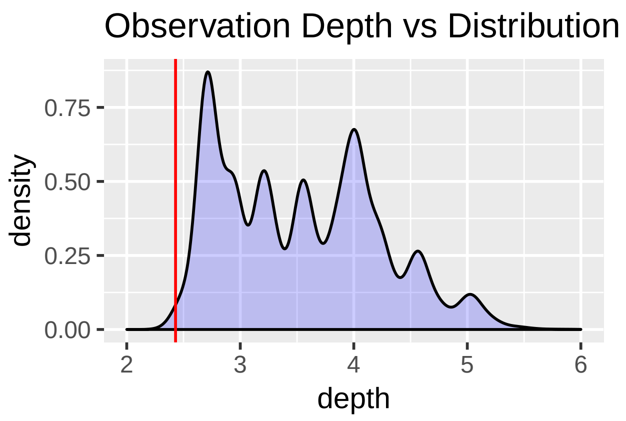
```
Warning: Removed 32 rows containing non-finite values (stat_density).
```

```
Warning: Removed 16 rows containing non-finite values (stat_density).
```









In Chapter 2:

```
# read in housing data
data = read.csv("data/kc_house_data.csv")
data$date = ymd_hms(data$date)
data = data %>% mutate(year = year(date))
data = data %>% mutate(age = year - yr_built)

# convert appropriate variables into factors
data$year = as.factor(data$year)
data$yr_built = as.factor(data$yr_built)
data$floors = as.factor(data$floors)
data$waterfront = as.factor(data$waterfront)
data$view = as.factor(data$view)
data$condition = as.factor(data$condition)
data$grade = as.factor(data$grade)

# create categorical variable to tell if a house's price is
# greater than $650000 and if the house has been renovated
data = data %>% mutate(priceCat = ifelse(price > 650000, 1, 0),
    renovated = ifelse(yr_renovated == 0, 0, 1))
```

```r
data$renovated = as.factor(data$renovated)
data$priceCat = as.factor(data$priceCat)
```

```r
bedroomVPrice = ggplot(data,
    aes(x = bedrooms,
        y = price)) +
    geom_point() + scale_x_continuous(limits = c(0,
    15)) + ggtitle(label = "Number of Bedrooms vs Price Sold")

bathroomVPrice = ggplot(data,
    aes(x = bathrooms,
        y = price)) +
    geom_point() + scale_x_continuous(limits = c(0,
    10)) + ggtitle(label = "Number of Bathrooms vs Price Sold")

# had to remove
# outliers, but for
# the majority of
# homes the higher
# the
sqftlivingVPrice = ggplot(data,
    aes(x = sqft_living,
        y = price)) +
    geom_point() + scale_x_continuous(limits = c(0,
    20000)) + ggtitle(label = "Square Footage of Interior Living Space vs Price So

sqftLotVPrice = ggplot(data,
    aes(x = sqft_lot,
        y = price)) +
    geom_point() + scale_x_continuous(limits = c(0,
    1e+06)) + ggtitle(label = "Square Footage of Land Space vs Price Sold")

# in order to get a
# better look at the
# distribution of
# prices among houses
# with different
# floor counts,
# treated floors as a
# categorical
# variable
floorsVPrice = ggplot(data,
    aes(x = floors, y = price)) +
    geom_boxplot() +
    scale_y_continuous(limits = c(0,
        1500000)) + ggtitle(label = "Distribution of Price Sold across Houses with Diffe

conditionVPrice = ggplot(data,
    aes(x = condition,
        y = price)) +
    geom_boxplot() +
    scale_y_continuous(limits = c(0,
        1500000)) + ggtitle(label = "Distribution of Price Sold across Houses with Diffe

gradeVPrice = ggplot(data,
    aes(x = grade, y = price)) +
    geom_boxplot() +
    scale_y_continuous(limits = c(0,
        1500000)) + ggtitle(label = "Distribution of Price Sold across Houses with Diffe

sqftaboveVPrice = ggplot(data,
    aes(x = sqft_above,
        y = price)) +
    geom_point() + ggtitle(label = "Square footage of Living Space above Ground Level vs

sqftbasementVPrice = ggplot(data,
    aes(x = sqft_basement,
        y = price)) +
    geom_point() + ggtitle(label = "Square footage of Living Space above Below Level vs

sqftliving15VPrice = ggplot(data,
    aes(x = sqft_living15,
        y = price)) +
    geom_point() + ggtitle(label = "Square footage of Living Space of 15 neighbors vs Pr

sqftlot15VPrice = ggplot(data,
    aes(x = sqft_lot15,
        y = price)) +
    geom_point() + ggtitle(label = "Square footage of Land Space of 15 neighbors vs Pric

yearSoldVPrice = ggplot(data,
    aes(x = year, y = price)) +
    geom_boxplot() +
    scale_y_continuous(limits = c(0,
        1e+06)) + ggtitle(label = "Year Sold vs House Price")

ageVPrice = ggplot(data,
    aes(x = age, y = price)) +
    geom_point() + ggtitle(label = "Age of House vs Price Sold")

ageVPriceCat = ggplot(data,
    aes(x = priceCat,
        y = age)) + geom_boxplot() +
    ggtitle(label = "Distribution of Ages across Houses sold at <=$65000 and >$650

renovatedVPrice = ggplot(data,
    aes(x = renovated,
        y = price)) +
    geom_boxplot() +
    scale_y_continuous(limits = c(0,
        1250000)) + ggtitle(label = "Distribution of Price Sold across Unrenovated

waterfrontVPrice = ggplot(data,
    aes(x = waterfront,
        y = price)) +
    geom_boxplot() +
    ggtitle(label = "Distribution of Price Sold across Homes without and with Wate

viewVPrice = ggplot(data,
    aes(x = view, y = price)) +
    geom_boxplot() +
    ggtitle(label = "Distribution of Price Sold across View Levels") +
    scale_y_continuous(limits = c(0,
        2500000))
```

```
roomsAndFloorsPlots = grid.arrange(bedroomVPrice, bathroomVPrice, floorsVPrice, nr

sqftVPrice = grid.arrange(sqftlivingVPrice, sqftLotVPrice,
                          sqftliving15VPrice, sqftlot15VPrice,
                          sqftaboveVPrice, sqftbasementVPrice,
                          nrow = 3)

catVPrice = grid.arrange(waterfrontVPrice, viewVPrice,
                         conditionVPrice, gradeVPrice,
                         renovatedVPrice,
                         nrow = 3)

timeVPrice = grid.arrange(yearSoldVPrice, ageVPrice, ageVPriceCat, nrow = 2)
```

```
# create plot to view distribution of expensive and less
# expensive homes by location
ggplot(data, aes(x = long, y = lat, col = priceCat)) + geom_density_2d() +
    ggtitle("latitude and longitude vs price category")
```

```
# create new variable out of latitude bound observed in the
# data
data = data %>% mutate(latThresh = ifelse(lat >= 47.5, 1, 0))
data$latThresh = as.factor(data$latThresh)
ggplot(data, aes(x = latThresh, y = price)) + geom_boxplot() +
    scale_y_continuous(limits = c(0, 1e+06)) + ggtitle("latThresh vs house price")
```

```
# code to create randomforest model predicting whether or not
# a house's price is greater than $650000
set.seed(1337)
dataCleanedCat = data[, -match(c("id", "date", "zipcode", "yr_built",
    "yr_renovated", "price"), colnames(data))]
dataCleanedCat$year = as.factor(dataCleanedCat$year)
lowerPriceData = dataCleanedCat %>% dplyr::filter(priceCat ==
    0)
higherPriceData = dataCleanedCat %>% dplyr::filter(priceCat ==
    1)

# dealing with very unbalanced dataset, have to make sure to
# get enough of the less represented category in the
# trainingset
trainLowerIndices = sample(nrow(lowerPriceData), 0.66 * nrow(lowerPriceData))
trainHigherIndices = sample(nrow(higherPriceData), 0.66 * nrow(higherPriceData))

trainLower = lowerPriceData[trainLowerIndices, ]
trainHigher = higherPriceData[trainHigherIndices, ]
testLower = lowerPriceData[-trainLowerIndices, ]
testHigher = higherPriceData[trainHigherIndices, ]

trainCat = rbind(trainLower, trainHigher)
testCat = rbind(testLower, testHigher)

control <- trainControl(method = "repeatedcv", number = 10, repeats = 3,
    search = "grid")
tunegrid = expand.grid(.mtry = c(1:17))
rfCat = randomForest(priceCat ~ ., data = trainCat, method = "rf",
    tuneGrid = tunegrid, trControl = control)

table(predict(rfCat, testCat), testCat$priceCat)
mean(predict(rfCat, testCat) == testCat$priceCat)
```

```
# read in shapley value data and explore what feature values
# for sqft_living contribute to higher and lower costs
priceFeatContr = readRDS("data/priceShaps.rds")
priceFeatContr$featureVal = as.numeric(as.character(priceFeatContr$featureVal))

priceFeatContr = priceFeatContr %>% mutate(positiveContr = ifelse(phi >=
    0, 1, 0))

sqft_livingNegContr = priceFeatContr %>% filter(feature == "sqft_living",
    positiveContr == 0)
sqft_livingPosContr = priceFeatContr %>% filter(feature == "sqft_living",
    positiveContr == 1)

# create plot of sqft living values for negative and positive
# shapley values
contrp1 <- ggplot(sqft_livingNegContr, aes(x = featureVal)) +
    geom_histogram() + ggtitle(label = "negative shapley value") +
    xlab(label = "sqft_living")
contrp2 <- ggplot(sqft_livingPosContr, aes(x = featureVal)) +
    geom_histogram() + ggtitle(label = "positive shapley value") +
    xlab(label = "sqft_living")
ggsave("figure/sqft_livingContr.png", grid.arrange(contrp1, contrp2,
    nrow = 1), width = 5, height = 3)
```

```
# read in shapley value data and explore what feature values
# for view contribute to higher and lower costs
priceFeatContr = readRDS("data/priceShaps.rds")
priceFeatContr$featureVal = as.numeric(as.character(priceFeatContr$featureVal))

priceFeatContr = priceFeatContr %>% mutate(positiveContr = ifelse(phi >=
    0, 1, 0))

viewNegContr = priceFeatContr %>% filter(feature == "view", positiveContr ==
    0)
viewNegContr$featureVal = as.factor(viewNegContr$featureVal)
viewPosContr = priceFeatContr %>% filter(feature == "view", positiveContr ==
    1)
viewPosContr$featureVal = as.factor(viewPosContr$featureVal)

# create plot of view levels for negative and positive
# shapley values
contrp1 <- ggplot(viewNegContr, aes(x = featureVal)) + geom_bar() +
    ggtitle(label = "Negative Shapley ") + xlab(label = "view level")
contrp2 <- ggplot(viewPosContr, aes(x = featureVal)) + geom_bar() +
    ggtitle(label = "Positive Shapley ") + xlab(label = "view level")
ggsave("figure/viewContr.png", grid.arrange(contrp1, contrp2,
    nrow = 1), width = 5, height = 3)
```

References

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in neural information processing systems* (pp. 4765–4774).

MISHRA, S. K. (n.d.). Shapley value regression and the resolution of multicollinearity. *Journal of Economics and Political Economy.* Retrieved from `http://kspjournals.org/index.php/JEB/article/view/850/1048`

Molnar, C. (2019, February). Interpretable machine learning. *Christoph Molnar.* Retrieved from `https://christophm.github.io/interpretable-ml-book/shapley.html#the-shapley-value-in-detail`

Molnar, C., Bischl, B., & Casalicchio, G. (2018). Iml: An r package for interpretable machine learning. *JOSS*, *3*(26), 786. `http://doi.org/10.21105/joss.00786`