Assessing the Impact of Property Features on Price of a Home

Abhishek Dendukuri

I. Introduction

Housing prices have long been dictated and settled by real estate agents as their background knowledge about properties reliably led to sales. But as technology evolved and as new data was found, it was only time before companies like Zillow attacked the online housing market to track property prices for prospective homeowners.

The Zestimate model that Zillow developed computes the estimated market value of a home using a proprietary formula. Using certain factors and environmental variables, it generates a comparative market analysis - or CMA - that can determine a valuation similar to one given by a real estate agent.

Currently we have no way of assessing whether or not Zestimate is the most reliable way to predict home prices. But there are not a lot of companies that have had the resources or time to tackle this as long as Zillow has, so it is perhaps the only way we can get a good idea of a property's value.

The Problem

What I'd like to investigate is if a subset of Zillow's provided features can lay a foundation for estimating the price of a home. I believe that there are certain features that hold much higher value than the rest, and therefore a better estimate can be made with only a subset of the features. We can make an approximation using the random forest machine learning method. Though I do understand that this method will provide a significantly less accurate estimate with respect to Zestimate, I would like to understand if using the entire breadth of factors provided by Zillow or only a subset of said factors will provide a better estimate of the home prices.

The point of this investigation is to try to correctly predict housing prices using machine learning models. This will help comprehend what exactly holds other companies back from creating an effective solution and revealing what can be added or removed to provide a more effective analysis.

II. Cleaning the Dataset

One of the benefits of having a dataset that Zillow provides is the incredible range of factors as well as the vast number of properties that the Zestimate analyzes. It is important to have a number of data points in order to create a much more comprehensive formula that can utilize all this information to create a better model. Further analysis reveals that the entire dataset provided only supplies housing data for properties in the Los Angeles area. This is really helpful because we can visualize how the real estate market is for only one small area of the map - anything larger would cause the number of properties to explode, making it harder to organize the data and thus making it more challenging to devise a reliable model.

Setup

First we set the working directory and read in the information from our properties file into a dataframe. The properties file contains over two million data points, each populated with its own unique set of feature data. The train dataset is much smaller in size since it only includes a small subset of the properties found in the properties file, but it does contain logerror information for these selected properties. This spreadsheet is also read into a dataframe.

```
setwd("/Users/JARVIS/RStudio/Capstone")

df <- read.csv("properties_2017.csv")
properties <- tbl_df(df)


df2 <- read.csv("train_2017.csv")
train <- tbl_df(df2)
```

Here we create a new dataframe that merges the previous two. As discussed above, the train dataset only contains a subset of the factors contained in the much larger properties dataset. In order to streamline the information available, we can call the merge function to perform an inner join, and maintain all the properties that contain the additional data found in the train dataset.

```
prop.train.join <- merge(properties, train, by="parcel_id")
nrow(properties)

## [1] 2985217

nrow(prop.train.join)

## [1] 77613
```

The reason for removing over one million data points is because we have logerror information for the parcels specified in the train file; it indicates how well the Zestimate predicted the price of the home. Considering that we have this information for about 77000 parcels, these become infinitely more valuable than the rest of the data because not only do we know that these values are relatively close to the actual price of the property, but also that the values that are not included may be skewed. Without logerror, we do not have sufficient evidence stating otherwise, so it is safer to remove these values.

Cleaning

If we take a look at the structure of the merged dataframe we can see a couple of interesting developments pop up:

```
str(prop.train.join)
```

```
## 'data.frame':    77613 obs. of  60 variables:
##  $ parcel_id                   : int  10711855 10711877 10711888 10711910
10711923 10711945 10711956 10711995 10712005 10712007 ...
##  $ ac_type_id                  : int  NA 1 1 NA NA 1 1 1 1 1 ...
##  $ arch_style_type_id          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ basement_sqft               : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ bath_count                  : num  2 2 2 2 2 2 3 2 2 2 ...
##  $ bed_count                   : num  3 4 4 3 4 3 3 4 3 3 ...
##  $ build_class_type_id         : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ build_quality_type_id       : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ calc_bath_and_bed           : num  2 2 2 2 2 2 3 2 2 2 ...
##  $ deck_type_id                : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ finished_floor_1_sqft       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ calc_finished_sqft          : num  2107 1882 1882 1477 1918 ...
##  $ finished_living             : int  2107 1882 1882 1477 1918 2031 1678
1882 1709 1639 ...
##  $ finished_perimeter_living   : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ finished_total_area         : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ finished_living_size        : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ base_area                   : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ fips                        : int  6037 6037 6037 6037 6037 6037 6037
6037 6037 6037 ...
##  $ fireplace_count             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ full_bath_count             : int  2 2 2 2 2 2 3 2 2 2 ...
##  $ garage_car_count            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ garage_sqft                 : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ has_hot_tub                 : Factor w/ 2 levels "","true": 1 1 1 1 1
1 1 1 1 1 ...
##  $ heat_system_type_id         : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ latitude                    : int  34222559 34220261 34222491 34221864
34220619 34220664 34224186 34223458 34224173 34224530 ...
##  $ longitude                   : int  -118617387 -118616409 -118616854 -1
```

```
18615739 -118615253 -118614105 -118614125 -118617964 -118618635 -118618636 ..
.
##  $ lot_size                     : num  9158 9035 9800 11285 11239 ...
##  $ pool_count                   : int  1 1 NA 1 1 1 NA NA 1 1 ...
##  $ pool_size                    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ spa_or_tub                   : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ pool_no_tub                  : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ pool_and_tub                 : int  1 1 NA 1 1 1 NA NA 1 1 ...
##  $ county_land_use_code         : Factor w/ 235 levels "","0","010","0100"
,..: 5 5 4 5 5 5 4 4 5 5 ...
##  $ land_type_id                 : int  261 261 261 261 261 261 261 261 261
261 ...
##  $ land_zone_desc               : Factor w/ 5652 levels "","**AHRP","#12",
..: 1925 1925 1925 1916 1916 1916 1916 1925 1925 1925 ...
##  $ raw_census_tract_and_block   : num  60371132 60371132 60371132 60371132
60371132 ...
##  $ region_id_city               : int  12447 12447 12447 12447 12447 12447
12447 12447 12447 12447 ...
##  $ region_id_county             : int  3101 3101 3101 3101 3101 3101 3101
3101 3101 3101 ...
##  $ region_id_neighborhood       : int  268588 268588 268588 268588 268588
268588 268588 268588 268588 268588 ...
##  $ region_id_zip                : int  96339 96339 96339 96339 96339 96339
96339 96339 96339 96339 ...
##  $ room_count                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ floor_type_id                : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ three_quarter_bath           : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ construction_material_type_id: int  NA NA NA NA NA NA NA NA NA NA ...
##  $ unit_count                   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ patio_in_yard                : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ shed_in_yard                 : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ year_built                   : num  1972 1972 1972 1960 1960 ...
##  $ num_stories                  : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ fireplace_exists             : Factor w/ 2 levels "","true": 1 1 1 1 1
1 1 1 1 1 ...
##  $ struct_tax_value_dollars     : num  249655 253000 257591 57968 167869 .
..
##  $ total_tax_value_dollars      : num  624139 660000 542923 78031 415459 .
..
##  $ assessment_year              : int  2016 2016 2016 2016 2016 2016 2016
2016 2016 2016 ...
##  $ land_tax_value_dollars       : num  374484 407000 285332 20063 247590 .
..
##  $ tax_amount                   : num  7659 8124 6673 1116 5240 ...
##  $ tax_delinquency_flag         : Factor w/ 2 levels "","Y": 1 1 1 1 1 1 1
1 1 1 ...
##  $ tax_delinquency_year         : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ census_tract_and_block       : num  6.04e+13 6.04e+13 6.04e+13 6.04e+13
6.04e+13 ...
##  $ logerror                     : num  -0.00736 0.02107 0.07717 -0.04124 -
```

```
0.0095 ...
##  $ date                          : Factor w/ 264 levels "2017-01-01","2017-
01-02",..: 188 241 94 76 83 30 184 68 213 53 ...
```

The first is that each feature is given a data type, whether it is *int*, *num*, or *Factor*. In order to create a working machine learning model, it is vital to focus on *int* and *num* – in other words, continuous variables. Any feature labeled *Factor* could have a wide range of levels – these are categorical variables. As shown above, there are three features – `county_land_use_code`, `land_zone_desc`, and `date` – that contain well over 200 levels, whereas the remaining three features have two levels. Since we are planning on creating regression trees, having *Factor* variables can stop the process from happening because there is no range of values that these types of variables are bound to. Therefore, for the purpose of the analysis we will remove categorical features in order to ensure that the data can be used to create regression trees.

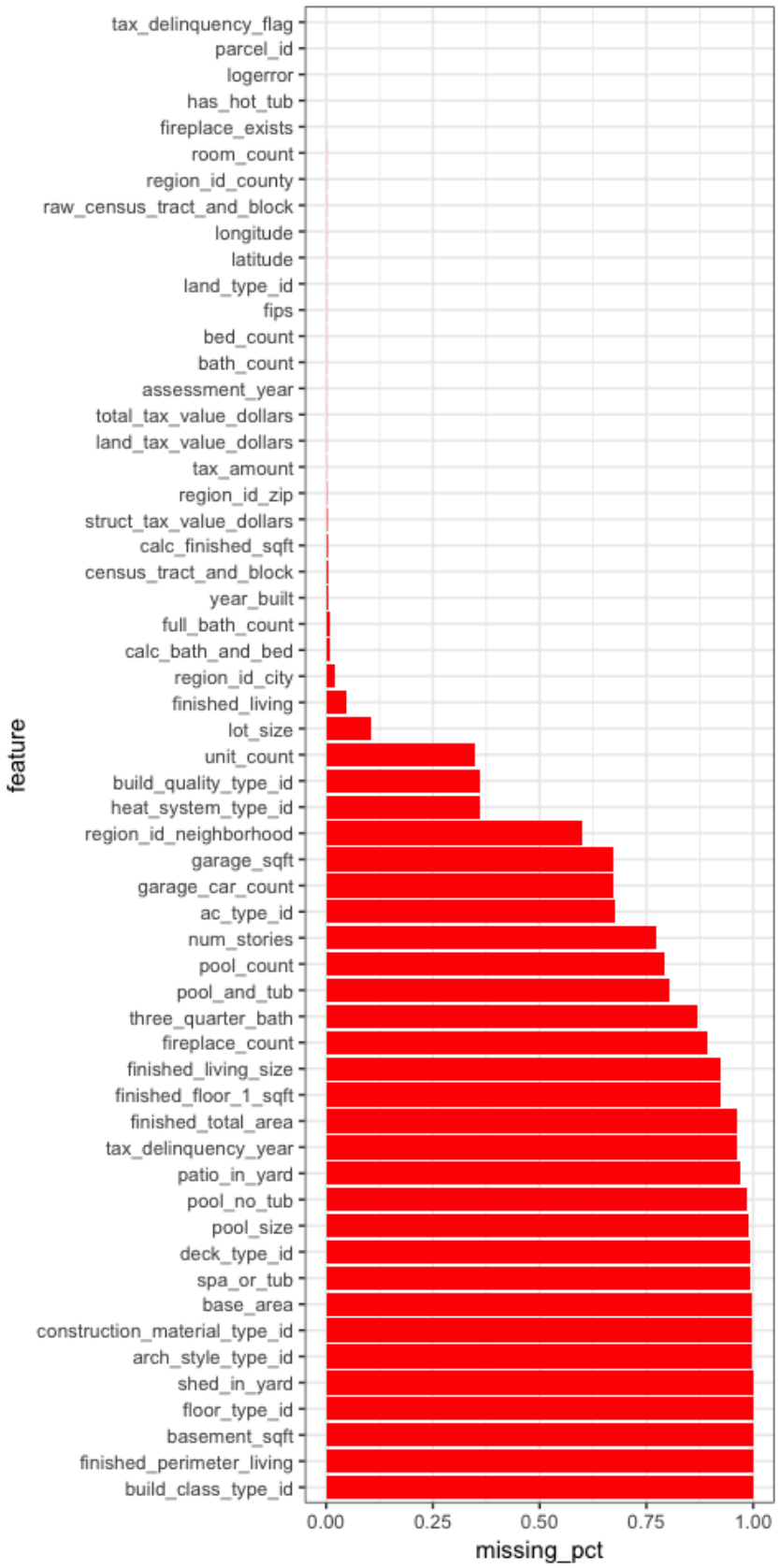
```
prop.train.join <- prop.train.join %>%
  select(-county_land_use_code, -land_zone_desc, -date)
```

The second interesting development is that there are still several features populated with 'NA' values. Since the analysis is going to dive into the functionality of each feature when it comes to predicting a housing price, it is imperative to have as many data points as possible in order to contribute to a better machine learning model. We can either handle this by imputing data (replacing NA with an estimate) into rows that are missing values or we can negate the features that are missing a certain threshold of information.

Missing Percentage Plot

Due to the vast nature of the dataset, using an imputation algorithm is not very efficient because even with a small number of iterations through the data, the time it takes to finish this algorithm is very lengthy. If the rows that contain 'NA' values were omitted, about 13000 observations would be deleted from a dataset that contained over 77000, but we'd still have a significant enough sample size.

But to understand which information can be removed, it is best to display what factors are missing the highest amount of values. From here we can create a cutoff that would represent our data in a tidier fashion.


tax_delinquency_flag
parcel_id
logerror
has_hot_tub
fireplace_exists
room_count
region_id_county
raw_census_tract_and_block
longitude
latitude
land_type_id
fips
bed_count
bath_count
assessment_year
total_tax_value_dollars
land_tax_value_dollars
tax_amount
region_id_zip
struct_tax_value_dollars
calc_finished_sqft
census_tract_and_block
year_built
full_bath_count
calc_bath_and_bed
region_id_city
finished_living
lot_size
unit_count
build_quality_type_id
heat_system_type_id
region_id_neighborhood
garage_sqft
garage_car_count
ac_type_id
num_stories
pool_count
pool_and_tub
three_quarter_bath
fireplace_count
finished_living_size
finished_floor_1_sqft
finished_total_area
tax_delinquency_year
patio_in_yard
pool_no_tub
pool_size
deck_type_id
spa_or_tub
base_area
construction_material_type_id
arch_style_type_id
shed_in_yard
floor_type_id
basement_sqft
finished_perimeter_living
build_class_type_id
feature
0.00
0.25
0.50
0.75
1.00
missing_pct

Since we are not using an imputation algorithm, features that maintain at most of their values will be highly important for the analysis. The chart reveals a natural cutoff around the .125 mark and from there we can utilize the remaining variables. Using this information, we can now limit our data to the most consistently available factors using the following chunk of code:

```
# Create array that stores the missing percentage of each feature
percentMiss <- sapply(prop.train.join, function(x) {sum(is.na(x)) / length(x)
})

# Remove values that have a percent miss rate of greater than 12.5%
removeMiss <- which(percentMiss > 0.125)
prop.train.join <- subset(prop.train.join, select=-removeMiss)


# Next omit the remainder of the NA values that persisted in the remaining
# columns
prop.train.join <- na.omit(prop.train.join)
```

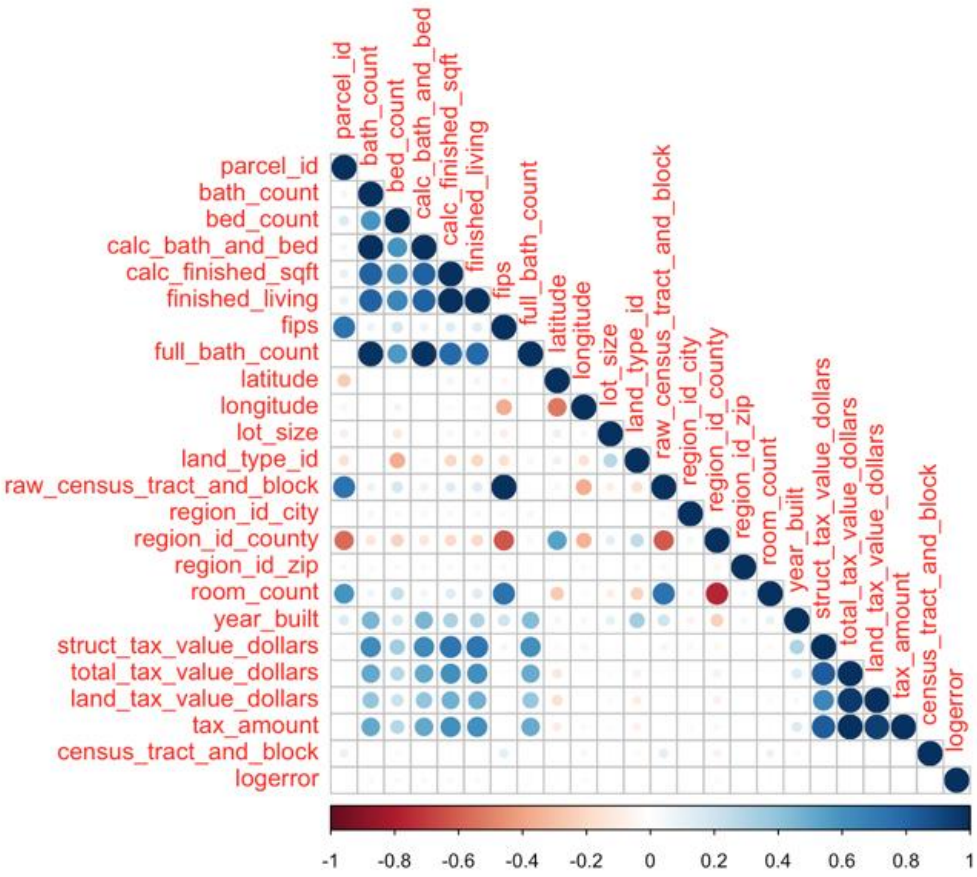
III. Exploring the Data

After perusing the generated data sheet created from prop.train.join, I've deduced that `total_tax_value_dollars` is the total price of the property and is comprised of two parts: `struct_tax_value_dollars` and `land_tax_value_dollars`.

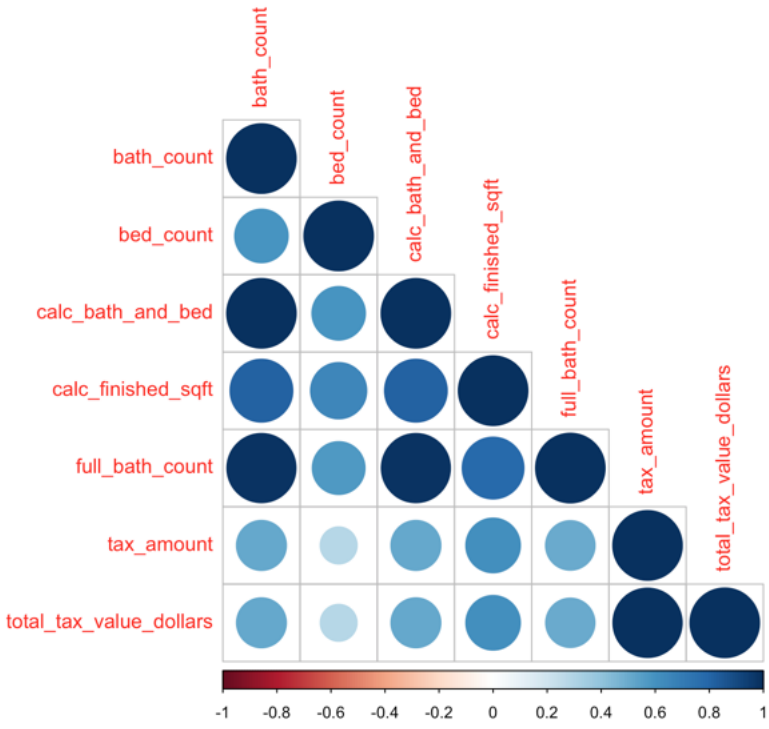
```
prop.train.join %>%
  select(struct_tax_value_dollars, land_tax_value_dollars, total_tax_value_do
llars) %>%
  head()
```

```
##   struct_tax_value_dollars land_tax_value_dollars total_tax_value_dollars
## 1                   249655                  374484                   624139
## 2                   253000                  407000                   660000
## 3                   257591                  285332                   542923
## 4                    57968                   20063                    78031
## 5                   167869                  247590                   415459
## 6                    85298                  339116                   424414
```

Knowing this information, we can create a correlation plot that directly shows how each variable correlates to each other.



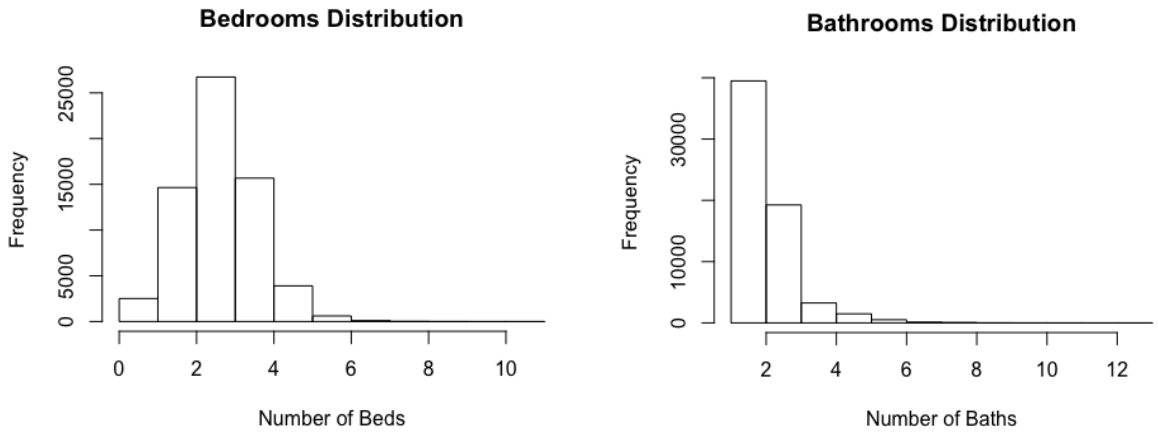
Though this initial correlation plot is not very elegant, it has however revealed that there are certain sets of variables that reveal a relatively strong correlation with `total_tax_value_dollars`. We can narrow the features down to a much smaller correlation plot.

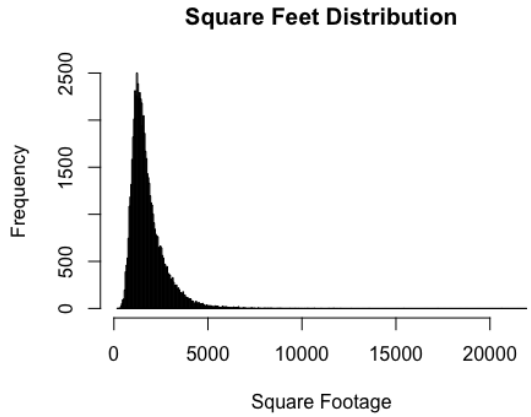


There are relatively strong correlations with `total_tax_value_dollars` among the following variables:

1. `bed_count`
2. `bath_count`
3. `calc_bath_and_bed`
4. `calc_finished_sqft`
5. `full_bath_count`
6. `tax_amount`

If we set aside `tax_amount` for the fact that it is not a visible feature like the others, we are essentially left with `bed_count`, `bath_count`, and `calc_finished_sqft`. We can understand the importance of these variables by looking into their distribution. `calc_bath_and_bed` and `full_bath_count` are important variables in their own right, but since they share a near 1:1 correlation with `bath_count`, their distribution plots will look relatively similar.





Though histograms reveal the mode of the data, they can help decipher patterns. What's interesting about this data is that the highest frequency ranges for each of the features - 1-3 bedrooms, 1-2 bathrooms, 1000-3000 square feet – seemingly constitute the base of an average home. In Further Exploration we will revisit these three features and reveal what we can learn from this.

We can investigate the impact of the rest of the features by applying machine learning models. Using the method random forest, we can create subsets of desired parameters and understand the mean squared error that results from these models. This will then help analyze which subset of features leads to the best predictor of price.

IV. Machine Learning

First we need to set a seed that tells the system where to start for random number generation using the `set.seed()` function. This will be the basis for random number generation for the random forest models.

Next we create the dataset `pt.MLSet` as a subset of `prop.train.join`. After multiple random forest generations using a different number of values from the `prop.train.join` dataset, yI've concluded that 10,000 randomly selected property values can serve as a good sample size for the random forest to generate an analysis from - it provided a similar result to the same analysis done to all 75,180 values, but with much greater efficiency. We then remove the categorical variables in order to maintain a random forest made up of only continuous variables.

Finally, we create two separate datasets - train and test. 70% of the values in the initial dataset - named `pt.MLSet` here - will go to the train set which will be used for the random forest model and the remaining 30% will go to the test dataset which will be used to predict the final values.

```
# Set and random number as a seed for reproducability
set.seed(123)

## Prepare the training set ##
# after several trials, 10000 rows provided a relatively quick output time
# relative to the volume
pt.MLSet <- prop.train.join %>%
  sample_n(10000) %>%

  # Remove the categorical factors
  select(-assessment_year)

# Create the training and testing sets
ptSplit <- initial_split(pt.MLSet, prop = .7)
pt_trainSet <- training(ptSplit)
pt_testSet <- testing(ptSplit)
```

Baseline Plot

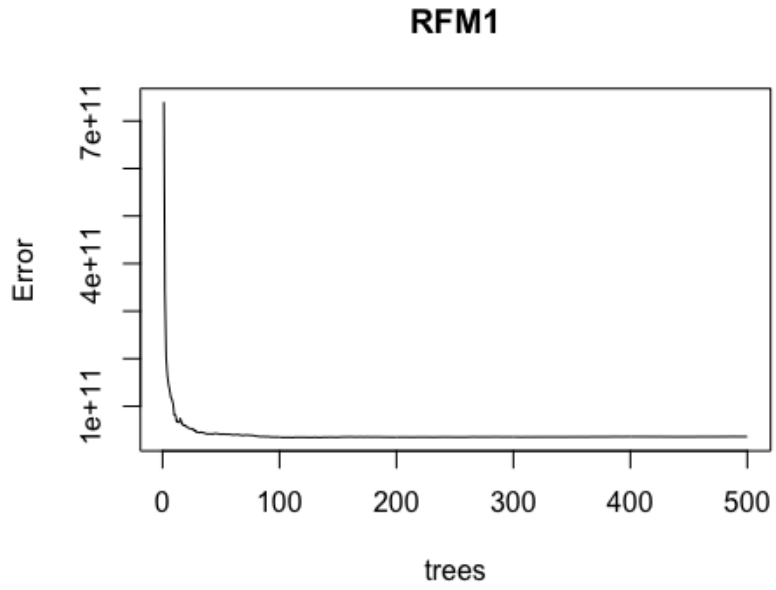
As a baseline, let's create a random forest using the entire set of features. From this information we can find the root mean squared error which indicates the predictive power of the algorithm on this data. For each of the cases listed below, root mean squared error represents the average error in the price in dollar amounts.

```r
# Start the random forest generation using ALL the features
RFM1 <- randomForest(
  formula = total_tax_value_dollars ~ .,
  data = pt_trainSet
)

# Plot the random forest generation
RFM1
```

```
##
## Call:
##  randomForest(formula = total_tax_value_dollars ~ ., data = pt_trainSet)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 35982857394
##                     % Var explained: 90.72
```

```r
plot(RFM1)
```




```r
# Find the number of trees with the lowest mse
minTree <- which.min(RFM1$mse)
minTree
```

```
## [1] 110
```

```r
# Using this info, derive the rmse
sqrt(RFM1$mse[minTree])
```

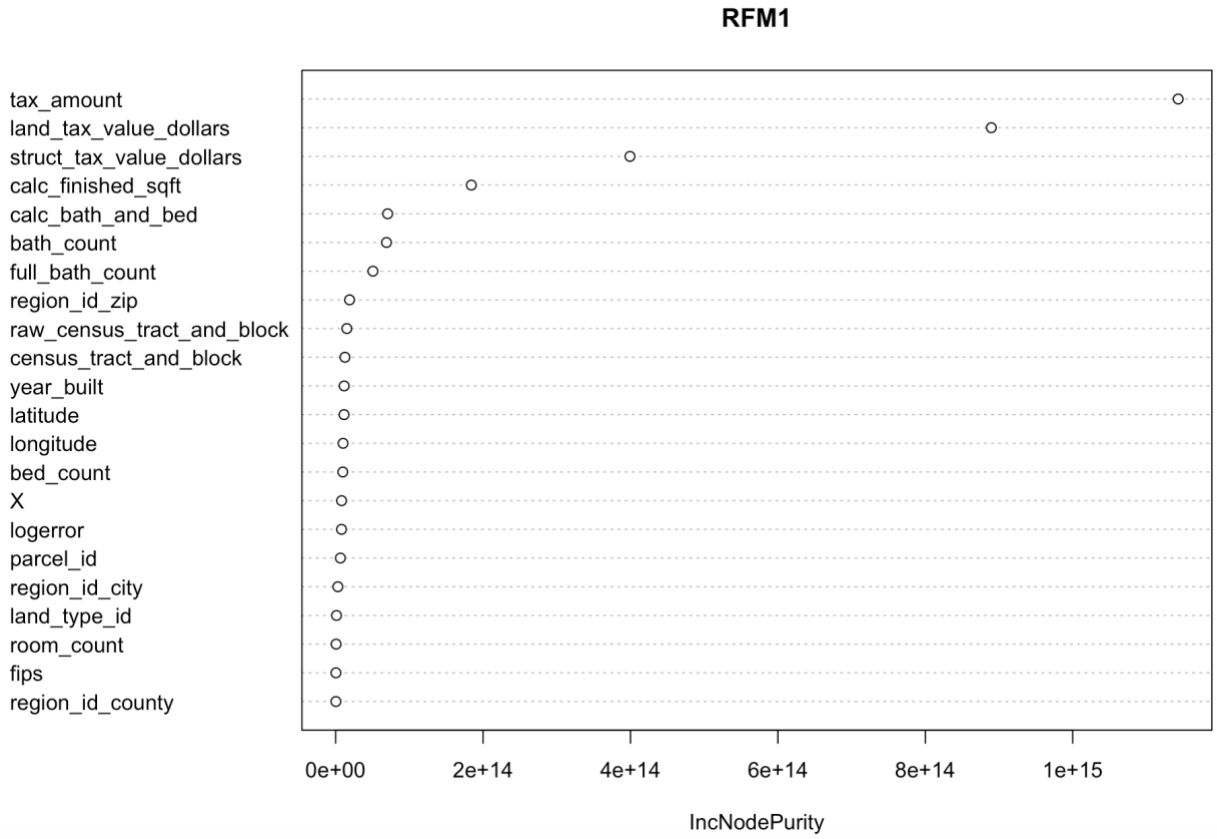
```
## [1] 185123.7
```

As seen here, the curve of the error line experiences a sharp decline as the number of trees initially and plateauing around the $200,000 mark. The root mean square error at the minTree value specified above reveals a value of $185,123.7 which will be used as the benchmark for the next plot. The closer the values is to zero, the better.

Testing Plot

The test random forest model we are going to try will be based off of a variable importance plot derived from `RMF1`. This variable importance plot will generate the significance of the remaining variables with respect to `total_tax_value_dollars`. We can use then use the highest values to derive which values to use for the final random forest model.

```
varImpPlot(RFM1)
```

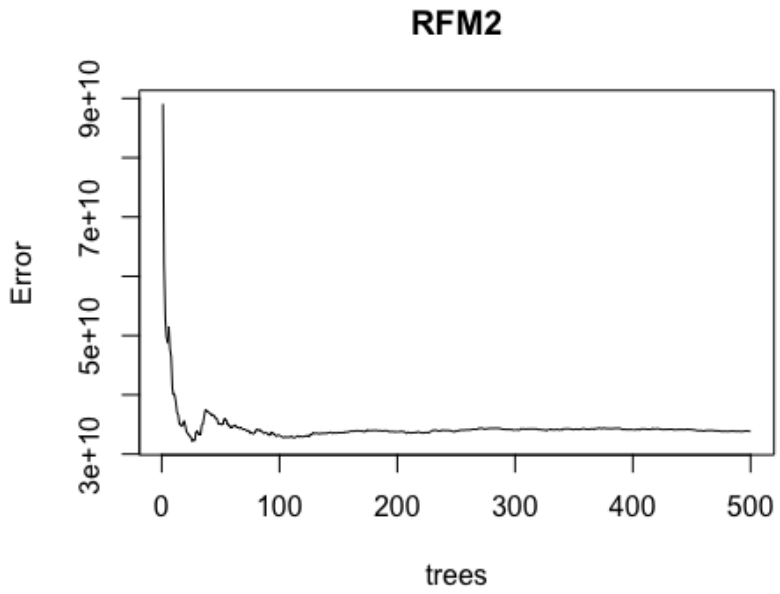



```
# Start the random forest generation using the following features
RFM2 <- randomForest(
  formula = total_tax_value_dollars ~ tax_amount + land_tax_value_dollars +
struct_tax_value_dollars + calc_finished_sqft + calc_bath_and_bed +
bath_count + full_bath_count,
  data = pt_trainSet
)
```

```
# Plot the random forest generation
RFM2

##
## Call:
##  randomForest(formula = total_tax_value_dollars ~ tax_amount + land_tax_va
lue_dollars + struct_tax_value_dollars + calc_finished_sqft + calc_bath_and_b
ed + bath_count + full_bath_count, data = pt_trainSet)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 33846882864
##                     % Var explained: 91.27

plot(RFM2)
```




```
# Using this info, derive the rmse
sqrt(RFM2$mse[which.min(RFM2$mse)])

## [1] 179299
```

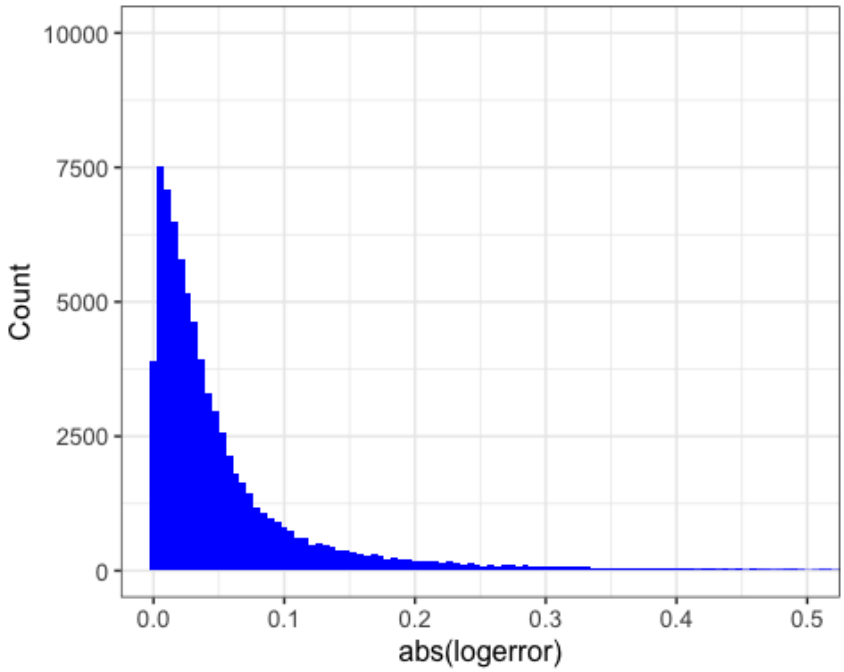
As we can see here, the curve is not as fluid as the benchmark plot, showing a slight increase in error at around fifty trees and declining again from there with slight blips along the way. The lowest point in the plot is revealed to be at 26 trees for an error of $179,299 - slightly lower than the benchmark. As this indicates, limiting the factors to those that the variable importance plot favor led to a decrease in the root mean squared error. This is important because this means that the variables that were used in `RFM2` held a much higher significance in determining an accurate price estimate than the variables that were not included.

V. Conclusion

After examining the machine learning models, we can see that `RFM2` had a lower root mean square error than `RFM1` using only a subset of the features provided in `RFM1`. This is significant because it shows that not every variable considered in the Zestimate carries highly significant value when determining a price estimate. Logically, this makes sense because there features like square feet, bathrooms, and bedrooms are more readily available to consumers and as proven above, these are some of the variables with a stronger correlation to the price of the property. Therefore, utilizing only a subset of the features provided was enough to create a simple predictive model for determining price.

Limitations

Though this study provided a good basis of how well random forest fared via mean squared error, the limitations largely stemmed off of the fact that random forest generated too basic of a model for serious use. Obviously, Zillow's pricing algorithm is extremely refined and looking through the train datasets `logerror` values, we can visualize the remarkable accuracy of the algorithm.

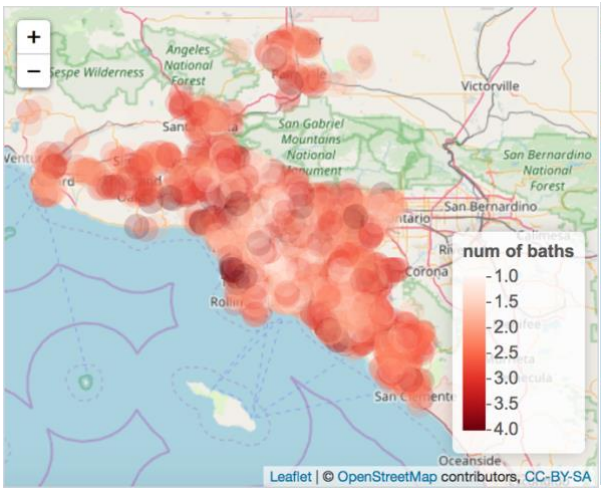


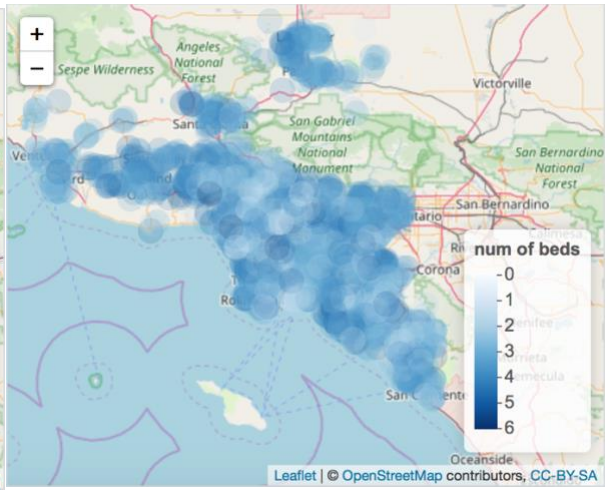
Another limitation was the volume of missing data in the properties dataset. Even though we narrowed down the features to the ones that had a significant amount of data, it is apparent that there are so many other features that go into the price. This information may not have been available, accounted for, or even simply, documented which severely limited the number of factors that mattered in the cleaned data and may have led to more significant results.

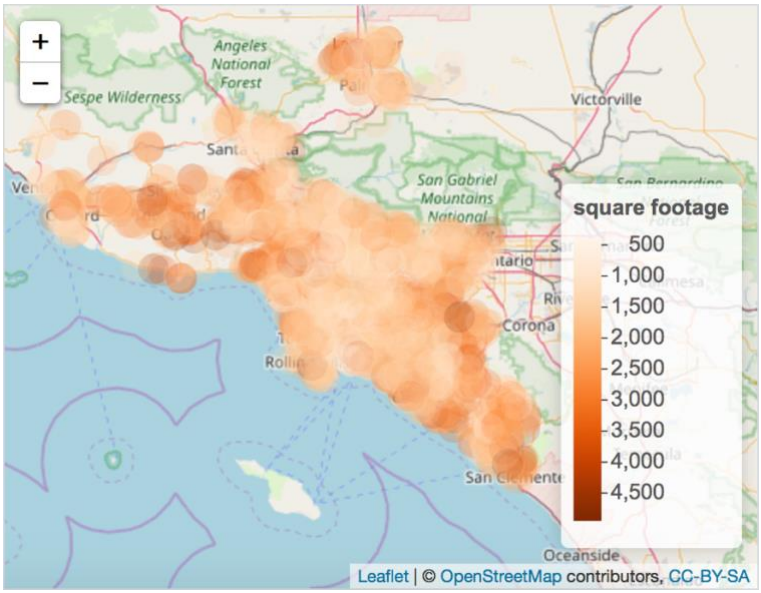
The final limitation was that we explored the data using only machine learning model. Despite individual efforts to utilize different models such as linear regression and support vector machine, I was not able to receive results that were either as consistent or as significant as those that random forest provided.

Further Exploration

As we saw earlier, `bed_count`, `bath_count`, and `calc_finished_sqft` are significantly correlated features with respect to `total_tax_value_dollars`. This is interesting because information about square feet, number of beds, and number of baths are highly transparent when it comes to looking at a home or an apartment. Using a map, we can visualize properties with these features to show their distribution over the Los Angeles area.







Fascinatingly enough, all three of these graphs indicate that the properties with the lowest number of beds and baths, and with the lowest square footage can be found in the center of Los Angeles. Using this information, we can further visualize other features and recognize patterns that can be applied to other cities.

Though the initial properties dataset came with over two million properties, it only scratched the surface of what this information can visualize. As noted we were limited in looking at properties in Los Angeles, but what if we had the information to other major metropolitan cities or even the information to sparsely populated parts of the US. What it can be used to find and how it can affect the real estate industry is key to unlocking even more data that has not been explored. For example, the Zestimate does not take into account a neighborhood safety index or school district ranking. For some people, the house can be everything they want but if the house is in the middle of a shady neighborhood or if the surrounding schools for their children do not provide the best education, it will be for naught.

Beyond these two examples, there is still so much data left untapped or unaccounted for. The potential for technology to grow within the real estate industry remains promising and with so much more being discovered every day, it will only be time before the online real estate market surpasses the everyday realtor. Whether it is for better or for worse, Zillow seems to have primed a revolution.