Cade Westlake Final Project 2 Real Estate Dataset

Cade Westlake

2024-12-08

Introduction

Hello!

Thank you for checking in with my final project. This is for the HarvardX PH125.9x Data Science: Capstone - Choose Your Own Final Project. This has been an exciting past 8 months for me going through this course. I've been a data geek for a few years now with Excel and have been able to build robust spreadsheet automation systems for my company. They encouraged me to take this class to learn a new data science language to see if that could help the company further. I must admit, during several of the 9 courses in this course series, I did not know how I could apply it to my company. I was having a difficult time seeing how casino-based betting games could apply to analyzing real estate transactions. However, as I progressed in the course series, I saw how regression and machine learning models could help us predict sale prices of homes, and I was eager to work a final project on thistopic. After I completed the first final project using the course-provided dataset of movie ratings, which was a useful and informative exercise, I looked on the course-suggested website Kaggle.com to look for a cleaned dataset that contained real estate transaction data. I eventually found this dataset from the URL https://www.kaggle.com/datasets/derrekdevon/real-estate-sales-2001-2020?resource=download. The user who scraped this data is username: derrekdevon. The data set was well over 900,000 rows. It contains real estate transactions all across the country from 2001-2020 in 170 towns with several property types. The file was roughly 95MB. Even zipped, it was over 25MB. In order to upload to Github, I had to reduce the compressed file size to under 25MB, so I deleted roughly 373,000 rows of data that had the Property Type of Nan. After zipping the "reduced" dataset, it was 17.7MB, which was able to be uploaded to Github. This did alter some of the graphs because it deleted most of the real estate transactions that occurred before 2007. You'll see below that a few graphs have really small bars before 2007 and the bars rise after 2007. Before I had to delete a large chunk of the dataset, the bars were high in 2001, rose slightly, then fell between 2005 and 2007 drastically, then they rose again after 2007. In addition, we frequently check the top ten rows of our datasets as we modify them simply to check our work before continuing.

I hope you can learn a little about the nation-wide real estate market in the United States of America from 2001-2020.

Thank you for taking the time to look at my project. This first section explores the data, performs a couple cleaning steps to ensure the data will perform in different functions, and makes some graphs that help us better understand the data through visualization.

The goals of this project are to learn more about the housing market across the country from 2001-2020, to figure out what factors had strong links to accurate predictions of sale amounts of individual properties, to then make predictions based off those # predicting factors using machine learning algorithms that R has built-in, then compare those to the actual test set sale amounts. We'll then calculate and graph how accurate the predictions model were.

Let's take a look at the structure of this dataset by looking at the first ten rows.

```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town    Address          Assessed.Value
##            <dbl>     <dbl> <chr>         <chr>   <chr>                     <dbl>
##  1       2020348      2020 9/13/21       Ansonia 230 WAKELEE AVE          150500
```

```
##  2         20002      2020 10/2/20       Ashford 390 TURNPIKE RD          253000
##  3        200212      2020 3/9/21        Avon    5 CHESTNUT DRIVE         130400
##  4        200243      2020 4/13/21       Avon    111 NORTHINGTON~         619290
##  5        200377      2020 7/2/21        Avon    70 FAR HILLS DR~         862330
##  6        200109      2020 12/9/20       Avon    57 FAR HILLS DR~         847520
##  7       2020180      2020 3/1/21        Berlin  1539 FARMINGTON~         234200
##  8       2020313      2020 7/1/21        Berlin  216 WATCH HILL ~         412000
##  9        200097      2020 6/4/21        Bethany 23 AMITY RD              511000
## 10         20139      2020 12/16/20      Bethel  16 DEEPWOOD DRI~         171360
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

From this, we know there are five more columns that are not being shown. we can see that there is a random serial number generated and assigned to each real estate transaction. There is the year the property was listed for sale, and the month, day, year date the property was sold. I would have loved to see the month and date the property was listed for sale. That would have given us the ability to analyze average time on market in different towns, which is very relevant for what I do at my work. When my company, a fix and flip lender, is looking for new cities to lend in, we look at the average time a house will sit on the market before selling. If properties, on average, take many months to sell, our borrowers are more likely to default, which is bad for our business. If properties sell quickly on average, our borrowers make more money, which means they'll be good borrowers for us. Moving on, we see there is a town column and street address column. There is an appraised value and a actual sale amount value. This is also relevant to what I do. My company has an in-house licensed appraiser who runs basic underwriting and value assessments. We then order third party appraisals and compare the two values. Then, when the property sells, we compare the three values: our predicted sale amount, an outside appraiser's predicted sale amount, and the actual sale amount. We run ratios on these to see if we need to adjust our in-house value-assessment practices or find better outside appraisers. The Sales Ratio column compares the assessed value to the actual sale amount. The property type column tells us if it's commercial or residential, the residential type column tells us if a residential property is a single family, multi-family, condo, etc. Each property type will have significantly different averages, distributions, etc. I would have loved to see square footage, year built, number of bedrooms and baths, etc. That would be incredibly useful data for us to analyze and then predict what a property would sell for per square foot in how many years. All these different data points are so useful for making accurate predictive models.

Thankfully, this data appears to be well cleaned. This gives us a little luxury to simply begin analysis.

Methods / Analysis

In our methods / analysis section, we first split the dataset into a train set and a test set. We explore the train set and make data visualizations of average sale amount based off different towns, years, property types, etc. We use a 75% split to the train set and 25% for the test set. I used a 75% split to the train set because I then split that train set further into a train-train and a train-test set, so we can train the machine learning algorithms on the train-train set, test their predictions on the train-test set, and if they're terrible, we can adjust the algorithm on the train-train set, further evaluate predictions on the train-test set, and keep iterating over and over again, until we finally decide to evaluate on the test set. We cannot use the test set during evaluation, training, and tuning. We can only test it at the very end. For the machine learning algorithms, I chose to use algorithms that work well with large datasets. I had chosen to use, I believe, K-Nearest Neighbors and it crashed my R Console and I had to force shut down my computer. After that experience, I researched which machine learning algorithms work efficiently and quickly on massive datasets. One of them, randomForest, we learned in the course. I tried that, and R told me it can only use categorical variables with 53 or less categories. I was using the Town column and there's 169 towns, clearly more than 53. Further research told me that the ranger package is a refined version of randomForest that is optimized to work on extremely massive datasets, but it works almost the same way. See references below. The other highly-recommended algorithm is XGBoost. This stands for extreme gradient boosting. This essentially

works by building decision trees one after the other, and each residual decision tree corrects the errors of the previous tree by fitting to the negative gradient of the loss function of the previous decision tree. This, performed repeatedly, building off the previous tree, performs a massive ensemble. You can decide how many trees you want to perform. Choosing too many trees can lead to overfitting. It also uses regularization.

Let's first split the dataset into train and test sets.

Look at the first ten rows of each data set to confirm their data is different. First, let's look at the train data set.

```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town       Address       Assessed.Value
##            <dbl>     <dbl> <chr>         <chr>      <chr>                  <dbl>
##  1       2020348      2020 9/13/21       Ansonia    230 WAKELEE ~         150500
##  2         20002      2020 10/2/20       Ashford    390 TURNPIKE~         253000
##  3        200212      2020 3/9/21        Avon       5 CHESTNUT D~         130400
##  4        200243      2020 4/13/21       Avon       111 NORTHING~         619290
##  5        200377      2020 7/2/21        Avon       70 FAR HILLS~         862330
##  6       2020180      2020 3/1/21        Berlin     1539 FARMING~         234200
##  7        200097      2020 6/4/21        Bethany    23 AMITY RD           511000
##  8         20139      2020 12/16/20      Bethel     16 DEEPWOOD ~         171360
##  9        200086      2020 8/10/21       Bethlehem  39 WOODLAND ~         168900
## 10       2000381      2020 9/13/21       Bloomfield 9 SADDLE RID~         163730
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

Then, let's look at the test data set.

```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town       Address       Assessed.Value
##            <dbl>     <dbl> <chr>         <chr>      <chr>                  <dbl>
##  1        200109      2020 12/9/20       Avon       57 FAR HILLS~         847520
##  2       2020313      2020 7/1/21        Berlin     216 WATCH HI~         412000
##  3        201295      2020 9/9/21        Bristol    609 CAMP ST           144340
##  4        200354      2020 12/29/20      Bristol    391 TIFFANY ~         173740
##  5      20000179      2020 2/22/21       Brookfield 72 HOMESTEAD~          94770
##  6        200042      2020 5/19/21       Canaan     93 MAIN ST            355800
##  7        200117      2020 4/12/21       Canton     52 COUNTRY L~         160850
##  8        200078      2020 9/20/21       Chester    6 SUNSET AVE          250100
##  9        200268      2020 7/9/21        Colchester 347 CABIN RD          104000
## 10        200035      2020 10/29/20      Colchester 160 SHADBUSH~         223900
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

Let's take a look at how many rows each dataset has. First, let's look at the number of rows the train data set has.

```
## [1] 470389
```

Then, let's look at how many rows the test data set has.

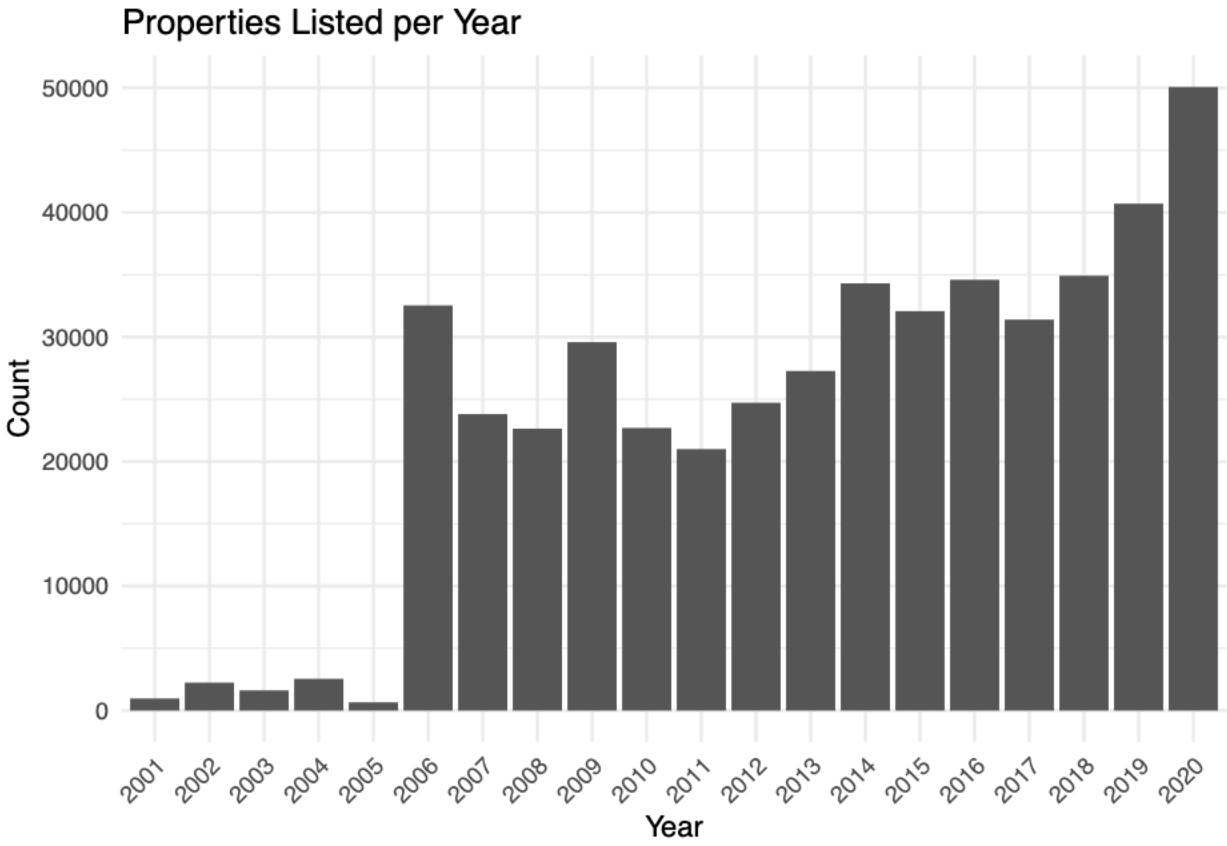
```
## [1] 156793
```

Let's take a look at the volume of properties listed in each year. We will be exploring the train set because it has over 470,000 rows, which is more than enough of a sample to explore for a homework project.

```
##    Year Count
## 1  2001   973
## 2  2002  2247
```

```
## 3  2003  1621
## 4  2004  2550
## 5  2005   663
## 6  2006 32533
## 7  2007 23806
## 8  2008 22638
## 9  2009 29586
## 10 2010 22698
```

Let's plot the number of properties listed per year.



Please note that since I had to delete 370,000 rows of data in order to upload the dataset to Github, this graph does not show the whole story. I deleted over 95% of rows that had a property type of NaN. Before I had to delete those rows, this bar graph showed 2001-2005 as being very high bars, in the 40,000 to 50,000 range. It downturned in 2006. It is somewhat surprising to me there was a huge downturn of properties between 2006 and 2014. I know everybody was affected by the real estate crash. However, I thought there was an increase of inventory listed for sale by banks due to foreclosures, causing the supply to be larger than demand, causing a downturn of prices. Of course, there are a million other variables that played into this. I thought there would be a huge spike of inventory. I suppose it actually makes sense though that the number of properties listed for sale decreased. If you bought your home before '07 for $600,000.00, and did not default, you would not want to list your home for $300,000.00. You'd lose money. That would explain why there was a huge downturn of properties listed for sale during these years while values were down.

Let's take a look at the volume of properties sold (recorded) each year.

First, we will need to extract the year from the month/day/year format in the Date.Recorded column. Before we do that, let's check the format of the Date.Recorded column.

```
##  chr [1:470389] "9/13/21" "10/2/20" "3/9/21" "4/13/21" "7/2/21" "3/1/21" ...
```

This states it is a "chr" format, meaning it's a character string. We need to convert this into a date.

Let's check the structure now to check that our code worked.

```
##  Date[1:470389], format: "2021-09-13" "2020-10-02" "2021-03-09" "2021-04-13" "2021-07-02" ...
```

Great! This column is now a date.

Let's do this to the test data set as well for uniformity and to help later test predictions algorithms run smoothly.

Let's check the structure of the test date recorded column to double check our work.

```
##  Date[1:156793], format: "2020-12-09" "2021-07-01" "2021-09-09" "2020-12-29" "2021-02-22" ...
```

We successfully changed the format to a date. Let's take the year out of the date for easier analysis.

Let's check to see how that worked.

```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town       Address       Assessed.Value
##            <dbl>     <dbl>         <dbl> <chr>      <chr>                 <dbl>
##  1       2020348      2020          2021 Ansonia    230 WAKELEE ~         150500
##  2         20002      2020          2020 Ashford    390 TURNPIKE~         253000
##  3        200212      2020          2021 Avon       5 CHESTNUT D~         130400
##  4        200243      2020          2021 Avon       111 NORTHING~         619290
##  5        200377      2020          2021 Avon       70 FAR HILLS~         862330
##  6       2020180      2020          2021 Berlin     1539 FARMING~         234200
##  7        200097      2020          2021 Bethany    23 AMITY RD           511000
##  8         20139      2020          2020 Bethel     16 DEEPWOOD ~         171360
##  9        200086      2020          2021 Bethlehem  39 WOODLAND ~         168900
## 10       2000381      2020          2021 Bloomfield 9 SADDLE RID~         163730
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

Let's convert the test set as well to just a year for the date recorded. We'll use this later in the machine learning prediction models. Let's take the year out of the date in the test data set to match our train data set.

Let's check to see how that worked.

```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town       Address       Assessed.Value
##            <dbl>     <dbl>         <dbl> <chr>      <chr>                 <dbl>
##  1        200109      2020          2020 Avon       57 FAR HILLS~         847520
##  2       2020313      2020          2021 Berlin     216 WATCH HI~         412000
##  3        201295      2020          2021 Bristol    609 CAMP ST           144340
##  4        200354      2020          2020 Bristol    391 TIFFANY ~         173740
##  5      20000179      2020          2021 Brookfield 72 HOMESTEAD~          94770
##  6        200042      2020          2021 Canaan     93 MAIN ST            355800
##  7        200117      2020          2021 Canton     52 COUNTRY L~         160850
##  8        200078      2020          2021 Chester    6 SUNSET AVE          250100
##  9        200268      2020          2021 Colchester 347 CABIN RD          104000
## 10        200035      2020          2020 Colchester 160 SHADBUSH~         223900
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

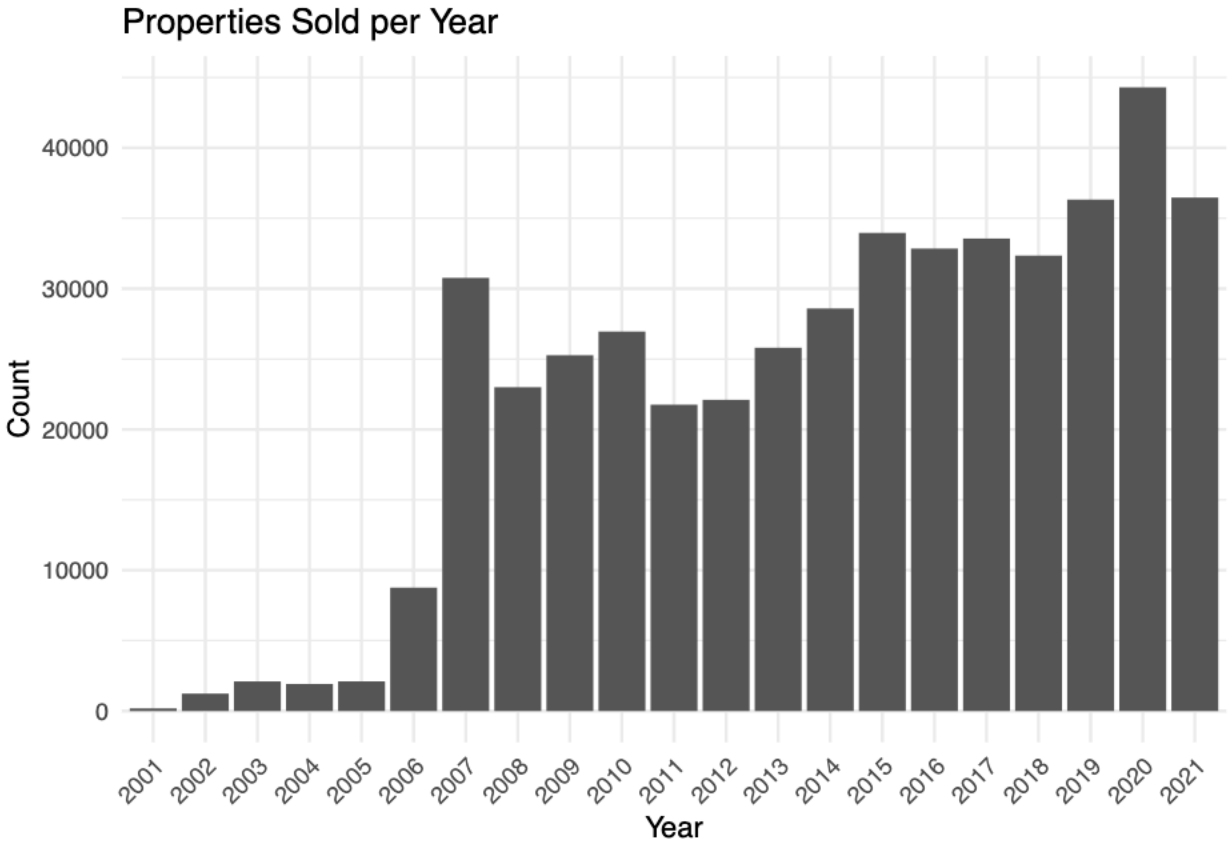
This deleted the month and day, and that's okay for our analysis project. It would be so incredibly useful if the list.Year column had a month and a day along with its year. Then, we could have counted how many days on average it took properties to sell after listing in each town. It could very well be that in Avon, properties

are selling on average two weeks after listing. That's a hot town for a short-term fix and flip lender to lend in. It's borrowers would likely do well and not default on their loans, so that would help the mortgage lender to succeed. If a town, say, Ashford, takes 180 days on average to sell after listing, it is not a town the mortgage lender, nor a fix and flipper wants to do properties in. Unfortunately, the dataset only contains the year for the list.Year column. Due to this, even if we kept the month and day in the sold year (Date.Recorded) column, it would not be clean data for us. If a property sold on January 1st, 2020, but it was listed in a random month in 2019, it could have taken one day to sell or it could have taken a whole year to sell. This makes it useless to try to analyze the average time on the market, at least from this dataset.

Now, let's graph the amount of properties sold per year. First, we need to count the number per year.

```
##    Year Count
## 1  2001   188
## 2  2002  1238
## 3  2003  2104
## 4  2004  1921
## 5  2005  2106
## 6  2006  8763
## 7  2007 30768
## 8  2008 23001
## 9  2009 25271
## 10 2010 26952
```

Let's plot.



This graph again shows the negative effects of having to delete the 373,000 rows in order to fit the file into Github. Originally, the bars for 2001-2006 were in the 30,000 to 40,000 range, higher than 2006 through 2014.The properties sold follow a similar pattern as the properties listed per year with a slump between 2005 and 2014. It could be interesting to plot the difference between properties listed and properties sold per year

on a bar chart. If more properties were listed than sold, the prices would likely drop that year, and vice versa. That's the law of supply and demand.

Let's count how many towns are listed to see if we can look at volume of properties listed/sold per town.

```
## # A tibble: 1 x 1
##   count
##   <int>
## 1   169
```

Wow! We see there are 169 unique towns. This would be difficult to visualize on a graph, whether it's a barplot of median sale price per town, a line graph of average sale price per town per year, etc. A graph of 169 lines or 169 bars would be difficult to see. We'll probably try to keep it to about ten bars, or five lines, of the biggest or smallest numbers per variable per town.

Let's pair down the data we want to graph to the five towns with the largest average sale amounts. After that, let's graph the five towns with the smallest average sale amounts.

Let's look at the average sale amount per town for all towns, over all time.

```
## # A tibble: 10 x 2
##    Town         AverageSaleAmount
##    <chr>                    <dbl>
##  1 Andover                243401.
##  2 Ansonia                188145.
##  3 Ashford                207910.
##  4 Avon                   443772.
##  5 Barkhamsted            239559.
##  6 Beacon Falls           275121.
##  7 Berlin                 264500.
##  8 Bethany                356327.
##  9 Bethel                 337441.
## 10 Bethlehem              302736.
```

Let's look at the 5 towns with the most expensive sale amounts over all time.

```
## # A tibble: 5 x 2
##   Town       AverageSaleAmount
##   <chr>                  <dbl>
## 1 Greenwich           2043216.
## 2 Darien              1534842.
## 3 New Canaan          1506610.
## 4 Westport            1396587.
## 5 Weston               937056.
```

Wow. The most expensive town (Greenwich) has an average sale amount of $2,043,216 over all time. I'm sort of hoping that the vast majority of the properties in that town are industrial or commercial properties. Further analysis would be beneficial to determine the percentages of property types sold in that town. If this ends up being all residential properties, that would be shocking. Of course, I know nothing about where Greenwich is. If it's in Beverly Hills or on the beach in California, that would make more sense.

Let's pull these top five towns from the train data set to be by themselves so we can graph their average sale amount per year.

Let's check the first ten rows to ensure we only have data from these towns.

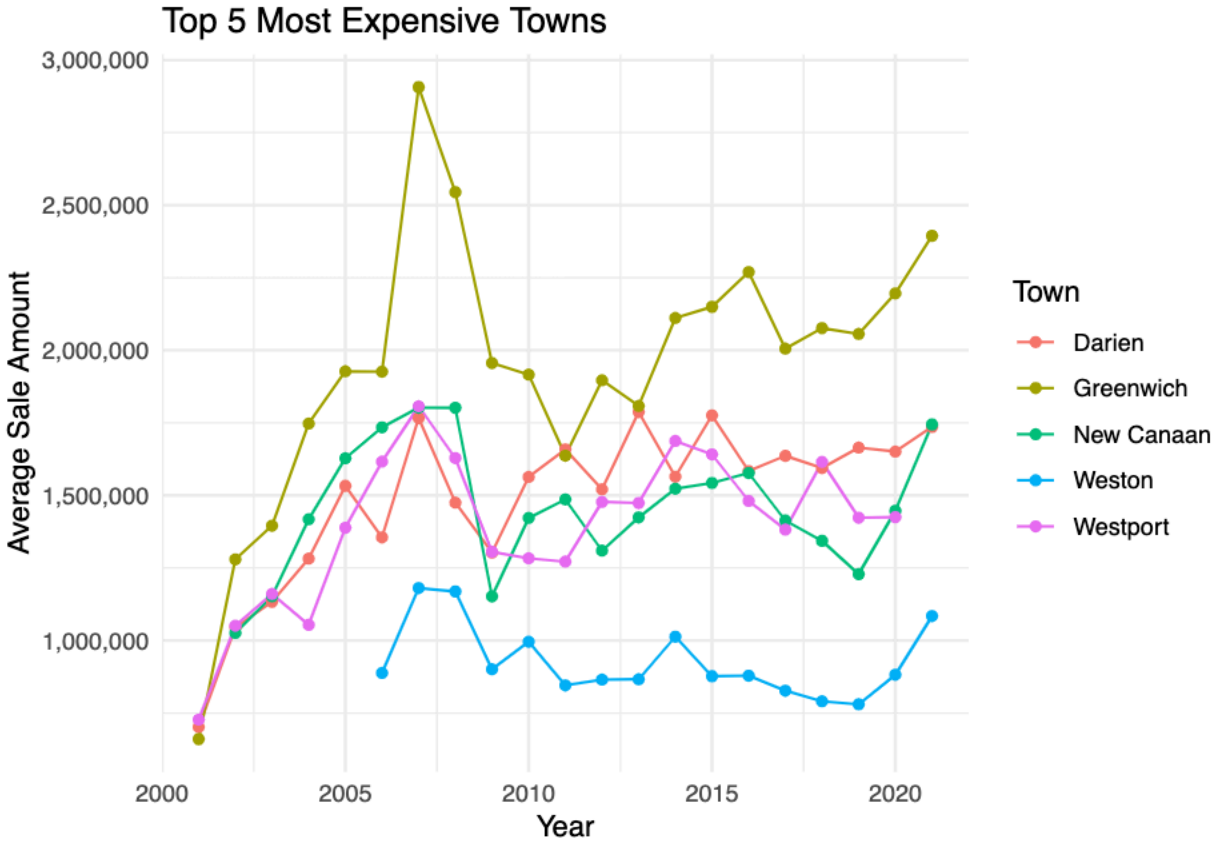
```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town       Address       Assessed.Value
##            <dbl>     <dbl>         <dbl> <chr>      <chr>                 <dbl>
##  1        200579      2020          2021 Greenwich  2195000             1061900
```

```
##  2        201494      2020          2021 Greenwich  20 EAST LYON~         632170
##  3        200421      2020          2020 Greenwich  130 NORTH WA~         288190
##  4        200338      2020          2021 New Canaan 116 ROCKY BR~         620760
##  5        200661      2020          2021 New Canaan 322 MAIN ST           1279460
##  6         20964      2002          2003 Greenwich  48 BENJAMIN ~         513730
##  7         21310      2002          2003 Greenwich  58 MARY LN            329490
##  8        201446      2020          2021 Greenwich  14 RIVERVIEW~         620480
##  9         20166      2020          2021 Darien     159 MIDDLESE~         693980
## 10        200245      2020          2020 Greenwich  15 PALMGER S~         385000
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

Let's group the towns' average sale amount by year.

```
## # A tibble: 10 x 3
##    Date.Recorded Town       AverageSaleAmount
##            <dbl> <chr>                  <dbl>
##  1          2001 Darien                701411.
##  2          2001 Greenwich             660000
##  3          2001 Westport              727078.
##  4          2002 Darien               1045018.
##  5          2002 Greenwich            1279410.
##  6          2002 New Canaan           1025725.
##  7          2002 Westport             1050749.
##  8          2003 Darien               1133265.
##  9          2003 Greenwich            1395278.
## 10          2003 New Canaan           1152445.
```

Let's graph these on a line chart.



This is a very useful graph. We can see that Greenwich has always had higher average sale values than any other city, making it the most expensive town at least since 2000. Westport, New Canaan, and Darien have all swapped positions for which town has a higher average sale price. Of course, as mentioned before, it would be worthwile to analyze and graph which property types have which average sale amounts in which towns. Even further, it would be worthwile to analyze which streets in which towns have higher average sale amounts. For example, in Las Vegas, Nevada, in the United States, there are several million people who live in that metropolitan area. There are suburbs with single family housing, apartments, condos, etc. However, despite the metro having hundreds, even thousands or tens of thousands of square miles, the Las Vegas Strip is about a one to two mile road that has about 50 or so massive hotels that attract millions of tourists a year. The value of real estate on this tiny section in Las Vegas is absolutely astronomical. To put it this way, a Taco Bell value box in my hometown in Arizona costs 5 dollars. On the Las Vegas Strip, that same 5 dollar box cost me 27 dollars when I went there on business last month. That difference in price for a couple burritos is purely to cover the rent for that business to pay their extremely local real estate value. A condo on the strip would likely cost much more than a condo on the outskirts of the metro. As we can see from our graph, all the towns tended to trend upwards over twenty years, despite the 2007 crash. In a prediction model, we could use the year to train the prediction model to adjust a sale amount prediction to be less if it's an older year, or higher if it's a newer year. The machine learning models included in R could even adjust for if the year is in between 2007 and 2014 to adjust to have a lower property sale amount.

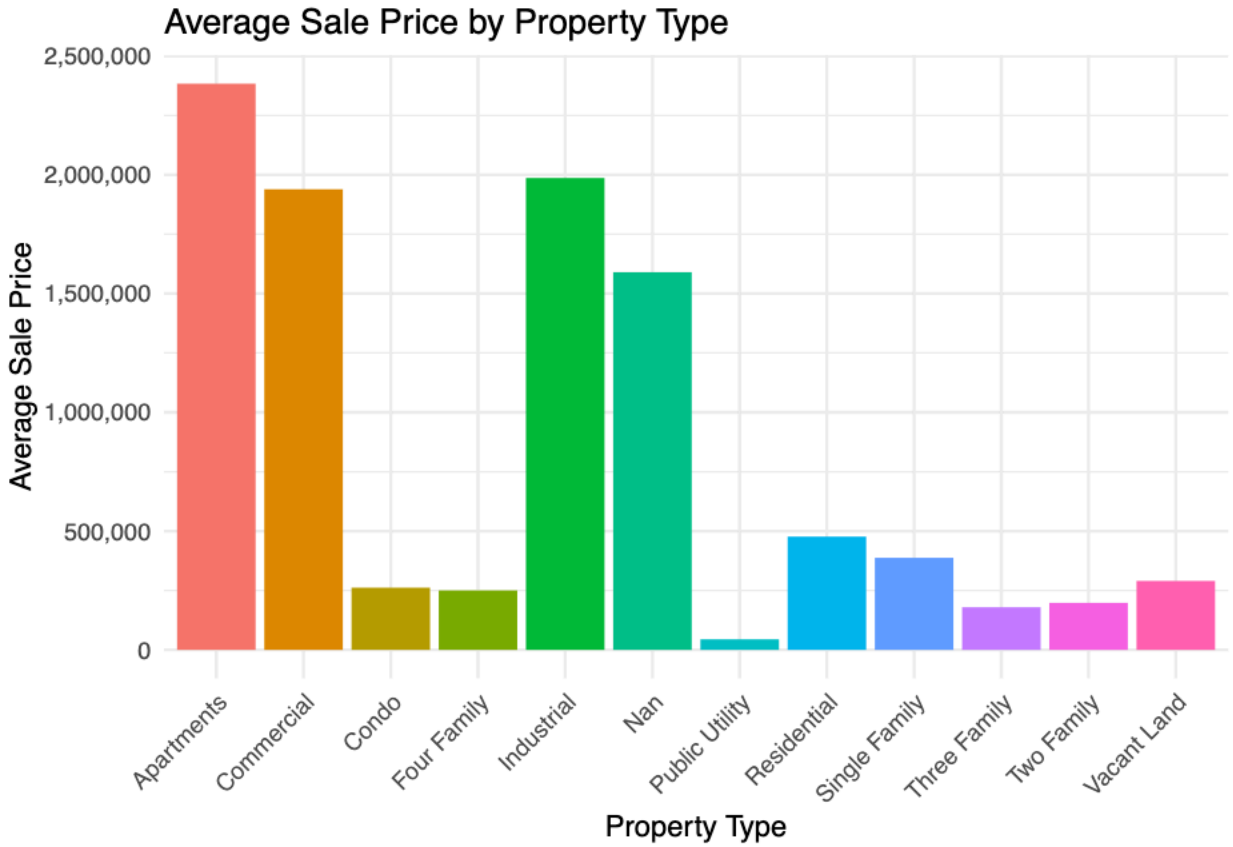
If this data analysis project was for my work, I'd compare the assessed value against the recorded sale value for each property individually. I'd compare the differences and look at the standard deviation between assessed and recorded sale values. I'd look at the distributions between the two and compare to see what their differences in smoothed distributions graphs per year were to gauge our accuracy with our different in-house appraisers we've had over the years. However, for the scope of this project, simply trying to train a machine learning model to predict a property's sale price, we're really going to be adjusting for property type, town, and year. These three factors will be able to help a machine learning model more accurately predict a property's recorded sale price. So far, we've taken a look at average sale price for five towns across twenty years, and this demonstrated to us very well that the year has a significant effect on property's recorded sale prices. In addition, it showed that the town the property is located in also has a significant effect.

Let's take a look at the average sale price by property type across all time. If this is significantly different, which I imagine it will be, we can reasonably assume the year and town the property type is located in will also have an effect as it did for the top 5 most expensive towns.

Let's take a look at the average sale price by property type across all time.

```
## # A tibble: 10 x 2
##    Property.Type  avg_sale_amount
##    <chr>                    <dbl>
##  1 Apartments            2383531.
##  2 Commercial            1939155.
##  3 Condo                  261903.
##  4 Four Family            250127.
##  5 Industrial            1986531.
##  6 Nan                   1589987.
##  7 Public Utility          44515.
##  8 Residential            476833.
##  9 Single Family          387524.
## 10 Three Family           179603.
```

We can see that each property type as a whole over twenty years has wildly different average sale amounts. Let's graph.



Wow, that's powerful. We see that commercial and industrial properties on average sell for about $2 million across all towns and the past twenty years. The rest of the property types average between $150,000.00 and $500,000.00. I wonder if we could take a further look into three of these property types by town by year. I'd like to look at single family properties, commercial properties, industrial properties, and apartments, across the top five most expensive towns, across the past twenty years.

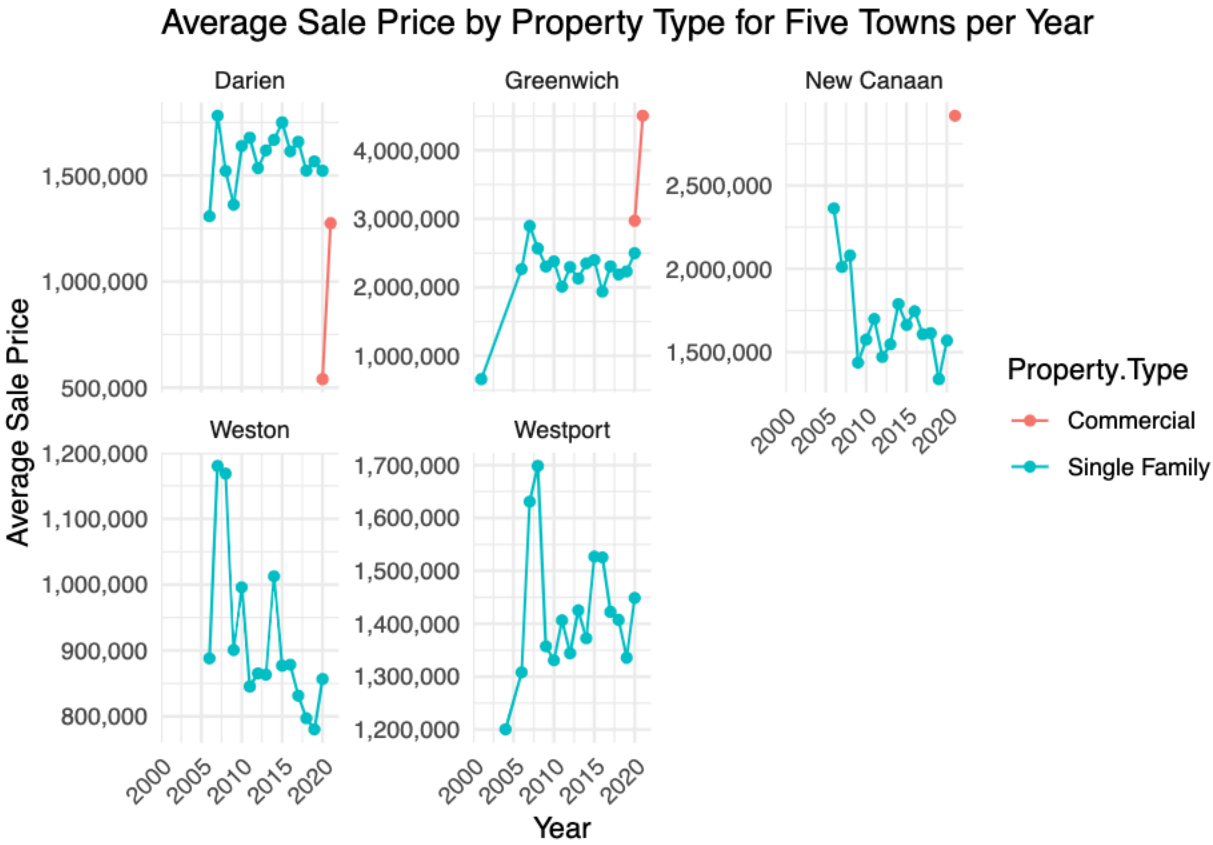
Let's start by filtering out the property types and five most expensive towns. Since we already know the five towns names listed in the five most expensive towns dataset, we'll make it simpler on ourself and just list the town names in the filter function.

Let's check out this filtered dataset to see what it ended up looking like.

```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town      Address        Assessed.Value
##            <dbl>     <dbl>         <dbl> <chr>     <chr>                   <dbl>
##  1        201635      2020          2021 Greenwich 41 WEST PUTNA~       10225390
##  2        200356      2020          2020 Greenwich 99 RIVER ROAD         4608520
##  3        201751      2020          2021 Greenwich 30 SOUTH WATE~        2162510
##  4         20614      2020          2021 Darien    7 SEDGWICK AVE        1194900
##  5        200066      2020          2020 Greenwich 40 WEST ELM S~         464100
##  6        200024      2020          2020 Greenwich 406 EAST PUTN~        1661730
##  7        200948      2020          2021 Greenwich UN E8 RIVER C~        5406590
##  8        200424      2020          2020 Greenwich 195 FIELD POI~        1820000
##  9        200843      2020          2021 Greenwich UND3 RIVER RO~        5406590
## 10        201138      2020          2021 Greenwich 7 RIVER ROAD ~        5406590
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

We see we got our desired property types in our desired towns. What you can't see in the pdf is where it lists

property types. However, when running in R Script, you can see the correctly filtered property types. Let's calculate the average sale price by year for each property type in each town and graph.



This is extremely interesting. We can clearly see here that Darien, Greenwich, and New Canaan did not have any commercial properties sold until about 2020. Of course, it is far more likely that this dataset simply did not scrape for commercial properties, and instead it scraped for primarily residential properties and happened to pick up a few commercial properties in the noise. We could verify this by checking out this property type in all towns. It is interesting that we filtered to see industrial and apartments as well and they do not show up in these five towns. Again, it is more likely that this dataset is simply missing that data than it is that these property types never had a single transaction in these towns in twenty years. I can say that before I had to remove the 370,000 rows to fit the dataset into github, these property types still did not appear until 2020 or so. In Weston and Westport on our current graph, they only have single family data.

I'd like to compare commercial, industrial, and residential property types across all the towns and years in one graph. This will tell us if the data scraping process did in fact collect more data on these property types and these five towns didn't sell many of these property types, or it will tell us the dataset simply does not have the data. It is extremely unlikely that across 170 towns or so that little to no commercial, industrial, or apartment properties were sold in twenty years.

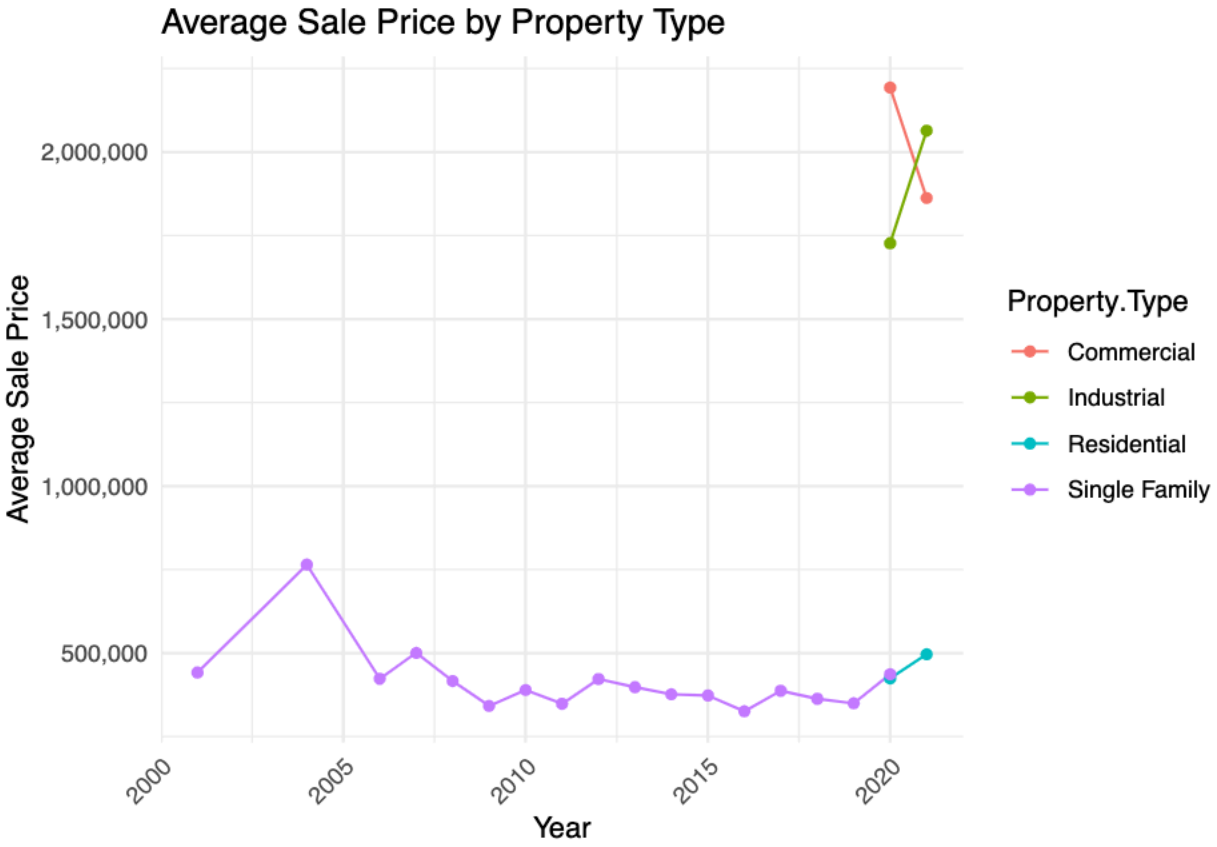
To begin comparing these different property types, let's first filter our dataset for these property types. Let's check the first ten rows in our filtered dataset to verify it worked.

```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town      Address       Assessed.Value
##            <dbl>     <dbl>         <dbl> <chr>     <chr>                 <dbl>
##  1       2020348      2020          2021 Ansonia   230 WAKELEE ~         150500
##  2         20002      2020          2020 Ashford   390 TURNPIKE~         253000
##  3        200212      2020          2021 Avon      5 CHESTNUT D~         130400
##  4        200243      2020          2021 Avon      111 NORTHING~         619290
```

```
##  5        200377      2020          2021 Avon       70 FAR HILLS~         862330
##  6       2020180      2020          2021 Berlin     1539 FARMING~         234200
##  7        200097      2020          2021 Bethany    23 AMITY RD           511000
##  8         20139      2020          2020 Bethel     16 DEEPWOOD ~         171360
##  9        200086      2020          2021 Bethlehem  39 WOODLAND ~         168900
## 10       2000381      2020          2021 Bloomfield 9 SADDLE RID~         163730
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

It looks like it worked. What you can't see in this pdf is the column where Property.Type lists Condo for the first ten rows. You can see that in the R Script.

Let's calculate the average sale amount per property type and graph.



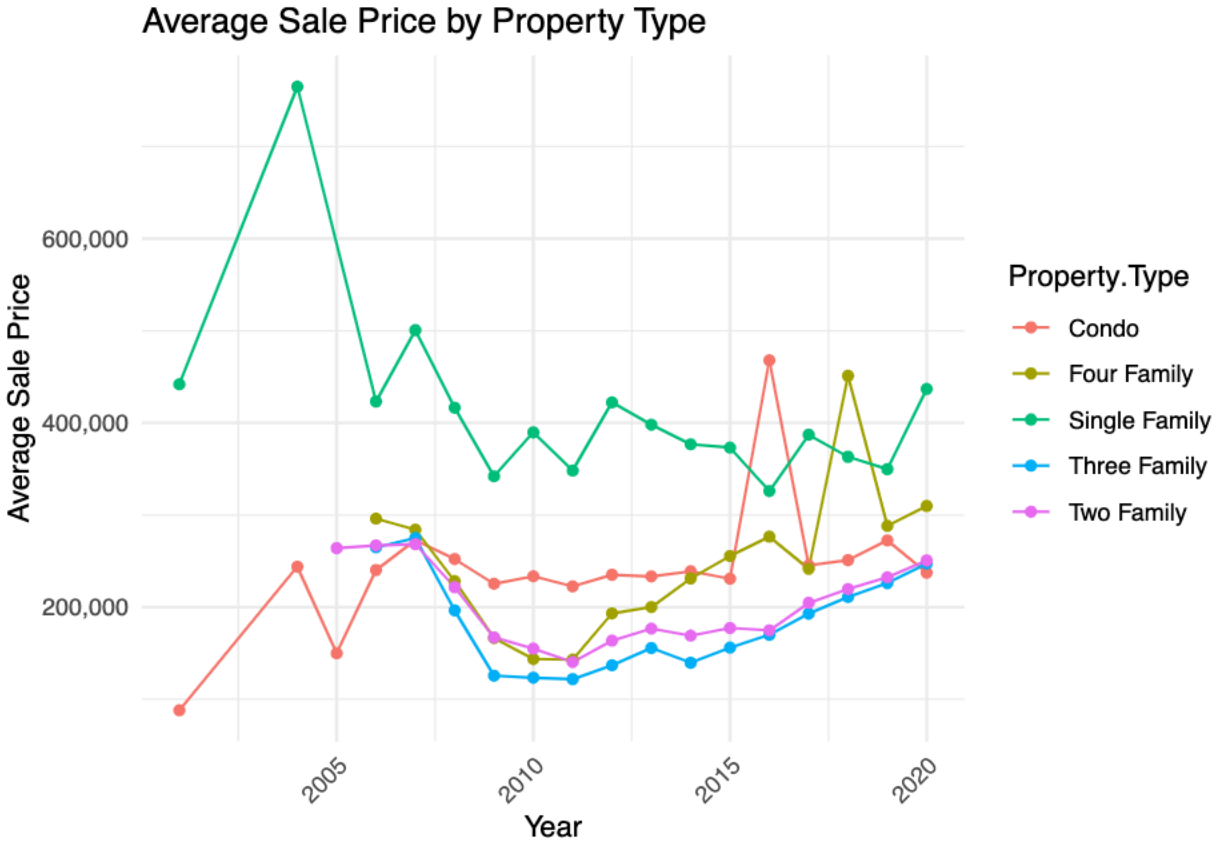
That is so curious. I thought there would be more sales of industrial and commercial properties since we were incorporating all 170 towns instead of just 5. In addition, I thought there would be more residential properties sold. It looks like they were all classified as single-family properties instead of simply residential. For some reason, properties were classified as residential in 2020-2021, and commercial and industrial properties were also only sold in 2020-2021 in this data set. I'd say with certainty that this dataset did not scrape for commercial or industrial properties very thoroughly.

Now let's compare single-family, condos, and multi-family. First, we'll filter our dataset for these property types and check the first ten rows to verify we did so correctly. Again, you won't see that in the pdf, but you can see that in the R Script.

```
## # A tibble: 10 x 11
##    Serial.Number List.Year Date.Recorded Town        Address      Assessed.Value
##            <dbl>     <dbl>         <dbl> <chr>       <chr>                 <dbl>
##  1         60228      2006          2007 Bethel      10 HUNTINGT~         120960
##  2         60416      2006          2007 Newington   29 STERLING~         221970
```

```
##  3         60537      2006          2007 Branford    91 JEFFERSO~         118800
##  4         60028      2006          2006 Ledyard     120 GALLUP ~          54950
##  5         60455      2006          2007 Danbury     163 SOUTH S~          79200
##  6         60082      2006          2007 Marlborough 11 SACHEM DR         158900
##  7         60169      2006          2007 Cromwell    134 SALEM DR          99830
##  8         60327      2006          2007 Cromwell    8 WATCH HL ~          97050
##  9         60354      2006          2007 Newington   41 WEBSTER ~          85750
## 10         60022      2006          2006 Bristol     279 REDSTON~          40000
## # i 5 more variables: Sale.Amount <dbl>, Sales.Ratio <dbl>,
## #   Property.Type <chr>, Residential.Type <chr>, Years.until.sold <dbl>
```

Let's calculate the average sale amount and graph.



This is a useful graph. It clearly shows the average sale amount for single family, two family, three family, and four family homes, as well as condos. I personally am curious about this data here. It appears that single family properties have always sold for higher prices than the other multi-family properties. This comes as a shock, because if you look online at the sale prices for a whole multi-family property, it is always in the high hundreds of thousands, even in the 1-2 million dollar range. This data would suggest that most of these multi family properties that are being sold are in fact individual units within the multi family properties. It would make a lot of sense that a single 600sqft unit in a three family unit would average around 120,000 and a single family property would average around 375,000 in that same year (2010). It would not make sense if an entire three family unit that's generating 5,000 a month in rent would average an entire sale price of 120,000.

If this was for my company, we would want to do a more in depth analysis. We'd analyze different residential property types per town per year compared to assessed values, average time to sell a property after listing, as well as if a town is currently trending up or down for actual recorded sale amounts. I could list a lot more things we'd analyze. However, what we have here is enough to prove I've learned how to explore and visualize data in R for the scope of this project. This also shows that town, year, and property type certainly have an effect on final sale amount of a property. Now that this is proven, we can begin to work on our machine

learning models to predict a property's final sale price based off these variables.

Let's build our machine learning models.

The goal is to predict sale amount using models that take into account the year, property type, and town.

We will use the train data set to train the models, then we'll apply those models to the test set to test the accuracy.

We will start with the random Forest model because it works well with large datasets. I initially tried to train a K-Nearest_Neighbors model to this dataset and it exceeded the memory limits of my computer. We need to use machine learning models that are optimized for large datasets and run efficiently with less RAM. First, we need to remove rows from our dataset that are NA's because machine learning models and all functions won't work with NA's being in the dataset.

I chose a seventy percent split of the train set into the train-train and train-test set because a 70% split of 470,000 rows would give us roughly a train-train set of 350,000 rows (mental math), which is a great amount to train a dataset. We can then test it on the train-test 120,000 rows, then go back and forth and fine tune. The train-test set is large enough and similar in size enough to the train-train set that it would protect us against overfitting. This also is similar in size to the final test set which has between 150,000 and 250,000 rows, which again is similar in size and would be good to prove we have not overfitted the algorithms during training and tuning.

I tried to fit the Random Forest Model here, but received an error. Please see the code:

rf_model <- randomForest(Sale.Amount ~ Date.Recorded + Town + Property.Type, data = train_train_data, importance = TRUE, # Calculate feature importance ntree = 500) # Number of trees

This gives an error stating that Random Forest models can only handle categorical variables with less than 53 categories. Well, we know we have about 170 towns, which means 170 categories. I researched how to fix this issue and CRAN, Cornell University, and a couple other resources (see References section below) state that the ranger package is similar to Random Forests but does not have the 53 category limit. It's faster, more efficient, and works off similar architecture to Random Forest. It is better for large datasets.

Let's fit the ranger model to the train_train dataset. This took a few minutes to run. I found it interesting that the R-squared and the predictions graph changed hardly at all when I changed the number of trees from 500 to 300, however the processing time of the model was much faster. I bet that if you started at ten trees, ran the model, then moved to fifty trees, the accuracy would skyrocket, and anything above 100 trees will only have marginal increases in accuracy, much longer processing times, and lead to overfitting. I then tried 700 trees to see what the result would be to compare. I found that the processing time went from thirty seconds to several minutes, the R-squared did not change more than a couple thousandths, and the predictions graph did not appear to change. I also imagine I'd be at a higher risk for overfitting. For this reason, I left the number of trees at 300 for ease of processing.

Let's fit the model.

```
## Growing trees.. Progress: 72%. Estimated remaining time: 11 seconds.
```

```
## Ranger result
##
## Call:
##  ranger(Sale.Amount ~ Date.Recorded + Town + Property.Type, data = train_train_data,      importance
##
## Type:                             Regression
## Number of trees:                  300
## Sample size:                      324917
## Number of independent variables:  3
## Mtry:                             1
```

```
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       69453574664
## R squared (OOB):                  0.3369461
```

We can see that the R squared was about 33.7 percent. That means the predicting variables of date.recorded, town, and property type, combined, have a 33.7 percent affect on the final property price.

Let's make predictions on the train_test_data and view the first ten predicted sale amounts.

```
##  [1] 376262.6 368855.5 368076.2 352357.6 374011.3 226554.3 380742.5 380742.5
##  [9] 294195.6 243549.6
```

Now, let's compare the predictions to the actual values.

```
##     Actual Predicted
## 1  1447500  376262.6
## 2   352000  368855.5
## 3   700000  368076.2
## 4    91000  352357.6
## 5   365000  374011.3
## 6    25000  226554.3
## 7   421000  380742.5
## 8   390500  380742.5
## 9     5000  294195.6
## 10  225000  243549.6
```

So far, that looks pretty inaccurate. Let's view the mean squared error.

```
## [1] "Mean Squared Error: 68757254494.794"
```

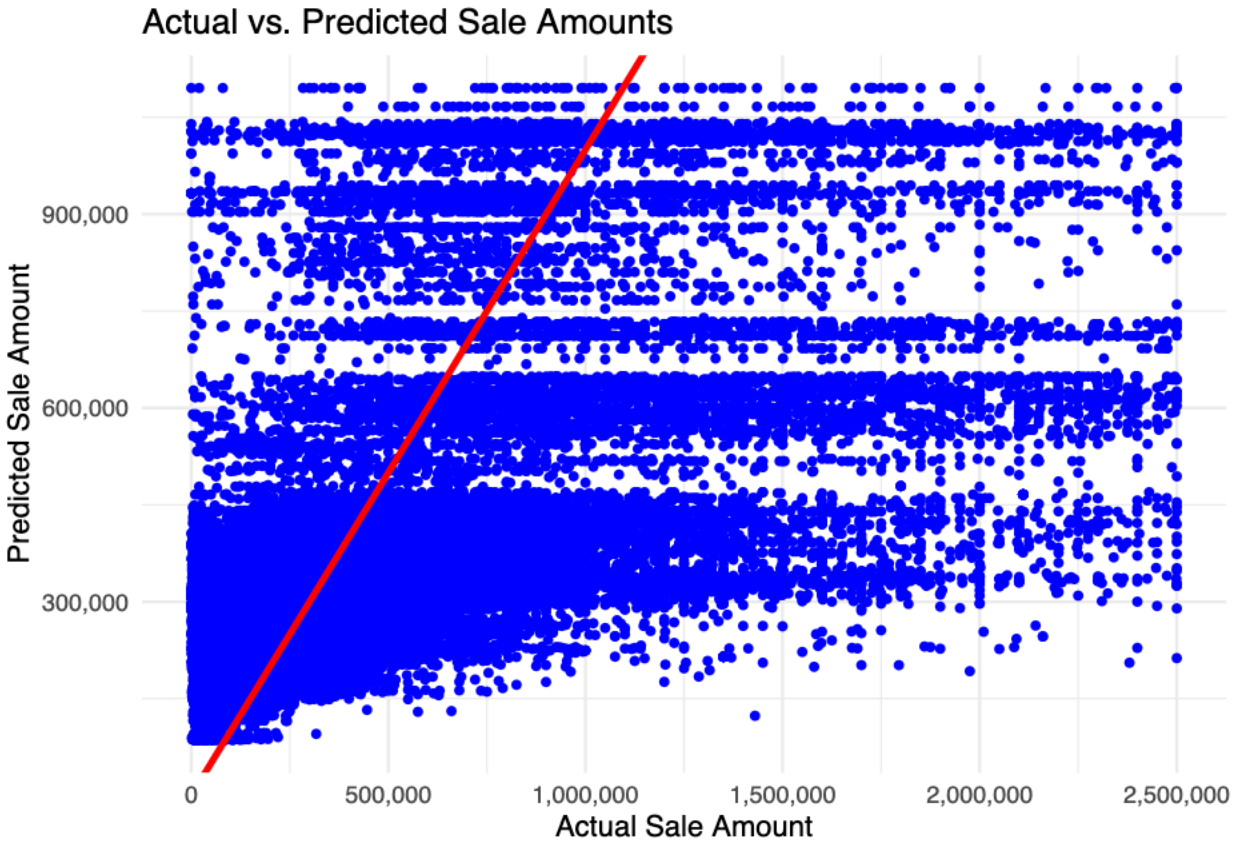
Holy cow. The mean squared error is in the tens of billions. That is incredibly high. To give some context on what mean squared error is, for each nth observation, we subtract the actual and predicted values, square that, then take the average of all the observations squares. The smaller that final number is, the more accurate it is. That being said, a mean squared error in the tens of billions, is pretty dismal.

Let's calculate the R-squared.

```
## [1] "R-squared: 0.337361818180839"
```

The R-squared returns roughly 33-34%. This means the ranger model is only explaining about 34% of the variability within the dataset. In other words, it's not a very accurate prediction model. We'd like to see closer to 90%. Basically, the R squared formula looks at the difference between all individual predicted values and actual values, then squares them all individually. Then, it divides that by the sum of all those squares. Then, it takes one and subtracts all of that. If it is close to zero, that means the prediction model is not accurate. If it is closer to 1, it means it is more accurate. Of course, it is possible to have a prediction model close to 1 that then performs very poorly on a final test set. This would be due to overfitting. Therefore, the R-Squared is not the end-all-be-all, however, it is a solid metric to use and to compare with other solid metrics of measuring accuracy of prediction models. Again, our ranger random forest model gave us an R-Squared comparable to the predictions in the 30% range. We definitely have some linkage between year, town, and property type to give us relatively more accurate predictions than throwing a dart at a dart board.

Let's plot the actual values versus the predicted values to help visualize our predictions vs actual sale amounts.



The red line in this graph is intentionally placed there to show a perfectly accurate prediction. The slope is one, meaning that if the predicted sale amount is $500,000.00 and the actual sale amount is $500,000.00, the dot would fall on the red line. Clearly, most of the dots are not on the red line. The ranger (random Forest) model was not given data that has a stronger link to the variability in sale prices. There are several incredibly important data points this dataset is missing. These are square footage, beds and baths count, year built, lot size, garage spaces, community features, if the property was recently renovated, what cosmetic materials were used for the property, etc. These are incredibly important metrics for predicting sale price of a residential property. My company relies very heavily on these factors, as do our borrowers. We spend a lot of time and a lot of money employing in-house appraisers to make their estimates of current value of a property in its run-down state, as well as making their estimates of current value of the same property after renovations. Our borrowers do the same; they want to make sure they will make money if they spend time and money renovating a property. They want to make sure they can sell the property for a profit. They rely heavily on square footage, beds and bath count, as well as the other factors I mentioned. I must admit I am disappointed this dataset does not have these data points. This would help us develop a much stronger machine learning prediction model. People will pay more money for a bigger house with more square footage, more bedrooms, and more bathrooms. It's simple, but it is probably the strongest link to sale price of the home. This data set does not give us the variables with stronger connections to sale price. It will be impossible to develop a highly accurate predictions model based off this dataset. We can clearly see that there are many very straight lines for the predicted sale amount. This is due to the machine learning model only being able to use year, town, and property type. We can see much more than 170 horizontal lines on this graph. If we multiply 170 towns by 7 property types by 20 years, that is how many horizontal lines are likely on this graph. Of course, most are clustered undder a million of actual sale amount and under $500,000 of predicted sale amount, which is why that is a big blue blob with no white space because those hundreds of thousands of dots overlap each other. If we had more variables, especially stronger variables, with more variation within the variables, we'd have a much more sporadic looking graph, and it would very likely all cluster much closer to the red line. Think about it, square footage has thousands of possible numbers because it is a continuous variable. This graph would not look so clean cut and we would not see so many clearly defined horizontal prediction lines.

Despite this data set not giving us stronger variables that tie to sale price of a property, we need to attempt to fit another machine learning model to this dataset to make predictions of sale amount to meet the grading criteria. I researched online to see what machine learning models work best with large datasets. My research found a few good articles on using the XGBoost package, which is efficient with working on large datasets. I reference those sources down below.

Let's train the XGBoost model.

I played around with the number of rounds. I found that the root mean squared error from iteration 1 of 434,466 dropped down to 218,228 at iteration 100, and to 217,483 at iteration 200. Clearly, there are marginal returns past iteration 100, it increases processing time, and can lead to overfitting. This is why I adjusted the number of rounds back to 100. This was at a maximum depth of ten for the trees. I then played around with the maximum depth of the trees at 100 iterations. I went from ten for the depth to 6 for the depth. The RMSE at iteration 100 was 222,843, which compared to 218,228 is roughly a 2% swing, decreased our processing time, and protects against overfitting. I left it at 6 for the maximum depth. It is extremely impressive that this machine learning model performs so efficiently. It only takes about 15 seconds to run through 100 iterations. Here, you will see the XGBoost model run through 100 iterations and you'll see the RMSE lower with each iteration.

```
## [12:11:48] WARNING: src/learner.cc:767:
## Parameters: { "silent" } are not used.
##
## [1]  train-rmse:434441.800117
## [2]  train-rmse:413263.157918
## [3]  train-rmse:392414.903651
## [4]  train-rmse:373553.009840
## [5]  train-rmse:360338.795586
## [6]  train-rmse:347085.665775
## [7]  train-rmse:337753.632193
## [8]  train-rmse:323536.966568
## [9]  train-rmse:316941.383216
## [10] train-rmse:307450.498774
## [11] train-rmse:296784.085968
## [12] train-rmse:289436.301238
## [13] train-rmse:286172.608727
## [14] train-rmse:279266.070676
## [15] train-rmse:275434.708828
## [16] train-rmse:269108.189741
## [17] train-rmse:265405.154335
## [18] train-rmse:264083.609058
## [19] train-rmse:259383.420396
## [20] train-rmse:256970.271361
## [21] train-rmse:254734.794056
## [22] train-rmse:251646.877657
## [23] train-rmse:249702.759200
## [24] train-rmse:248488.050260
## [25] train-rmse:247064.544839
## [26] train-rmse:246695.777776
## [27] train-rmse:246389.472115
## [28] train-rmse:246148.558242
## [29] train-rmse:244098.880110
## [30] train-rmse:243919.106812
## [31] train-rmse:241769.502865
## [32] train-rmse:240956.354086
## [33] train-rmse:240092.539157
```

```
## [34] train-rmse:239987.624042
## [35] train-rmse:239901.086789
## [36] train-rmse:239832.385420
## [37] train-rmse:239775.595359
## [38] train-rmse:238250.832099
## [39] train-rmse:237447.090465
## [40] train-rmse:236288.625970
## [41] train-rmse:236241.406472
## [42] train-rmse:235748.738045
## [43] train-rmse:234884.898380
## [44] train-rmse:234842.407315
## [45] train-rmse:233613.022459
## [46] train-rmse:233581.788015
## [47] train-rmse:232757.463673
## [48] train-rmse:232730.186869
## [49] train-rmse:232378.133903
## [50] train-rmse:231744.152914
## [51] train-rmse:231718.406061
## [52] train-rmse:231372.891585
## [53] train-rmse:230694.419258
## [54] train-rmse:230269.955357
## [55] train-rmse:230052.681444
## [56] train-rmse:229491.615453
## [57] train-rmse:229097.508528
## [58] train-rmse:228500.581466
## [59] train-rmse:228141.908572
## [60] train-rmse:227856.799279
## [61] train-rmse:227683.289559
## [62] train-rmse:227233.380317
## [63] train-rmse:226984.648235
## [64] train-rmse:226606.934131
## [65] train-rmse:226389.878806
## [66] train-rmse:226181.133928
## [67] train-rmse:226028.185099
## [68] train-rmse:225846.597346
## [69] train-rmse:225625.539761
## [70] train-rmse:225534.932791
## [71] train-rmse:225339.086905
## [72] train-rmse:225244.652565
## [73] train-rmse:225070.422878
## [74] train-rmse:224928.331918
## [75] train-rmse:224734.136654
## [76] train-rmse:224511.612316
## [77] train-rmse:224392.452649
## [78] train-rmse:224241.652047
## [79] train-rmse:224127.189917
## [80] train-rmse:224098.279764
## [81] train-rmse:224074.309727
## [82] train-rmse:223889.006584
## [83] train-rmse:223750.826325
## [84] train-rmse:223666.713535
## [85] train-rmse:223627.850926
## [86] train-rmse:223498.406124
## [87] train-rmse:223461.166914
```

```
## [88] train-rmse:223440.460038
## [89] train-rmse:223422.354797
## [90] train-rmse:223406.070944
## [91] train-rmse:223392.345198
## [92] train-rmse:223109.211083
## [93] train-rmse:223105.802858
## [94] train-rmse:223094.932827
## [95] train-rmse:223061.382636
## [96] train-rmse:223054.562394
## [97] train-rmse:222947.568463
## [98] train-rmse:222916.756819
## [99] train-rmse:222843.640387
## [100]    train-rmse:222695.879909
```

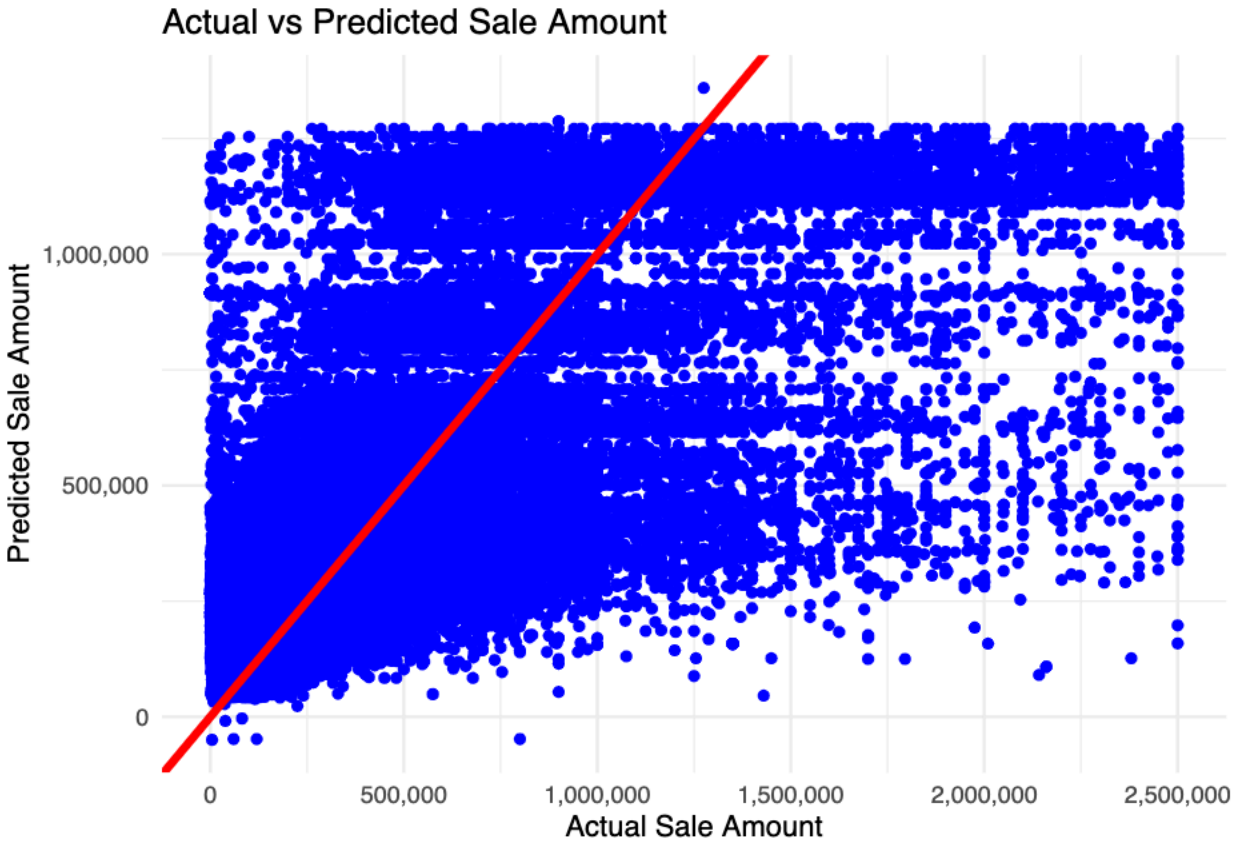
Now that the model is trained, let's make predictions on the test data and view the first ten rows of predictions.

```
##  [1] 440681.1 349150.9 362532.4 246700.3 358078.0 119127.6 362497.6 362497.6
##  [9] 339031.3 213233.5
```

Now, let's evaluate the model's performance. Let's calculate the RMSE.

```
## RMSE: 222118.7
```

Let's plot the actual sale amounts versus the predicted sale amounts.



As we can see, this plot looks very similar to the plot we got from using the Random Forest (ranger) model. We again see that we simply do not have strong enough predictors. The predictors we use do have some weight and importance to them as we can see by the clustering under a million of actual sale and under 700,000 of predicted sale. This cluster is close to the perfect prediction line in red. However, having stronger predictors such as square footage, bed/bath count, and lot size in addition to town, year sold, and property

type would strengthen this prediction model greatly. The blue dots would cluster significantly closer to the red line.

Results

Now it's time to see if we've overfit our machine learning models. Let's use our trained models to predict on the final test set and see what the results are.

Let's use the ranger package to predict on the final test set and view the first ten rows of predictions for sale amounts.

```
##  [1] 352462.2 367270.5 352357.6 328265.9 375728.9 373705.9 374030.6 373991.3
##  [9] 217651.5 351267.3
```

Now, let's compare the predictions to the actual values.

```
##     Actual Predicted
## 1  1250000  352462.2
## 2   677500  367270.5
## 3   249900  352357.6
## 4   299000  328265.9
## 5   170000  375728.9
## 6   599000  373705.9
## 7   280000  374030.6
## 8   418800  373991.3
## 9   350000  217651.5
## 10  392000  351267.3
```

So far, that looks pretty inaccurate. Let's view the mean squared error.

```
## [1] "Mean Squared Error: 69779978648.381"
```

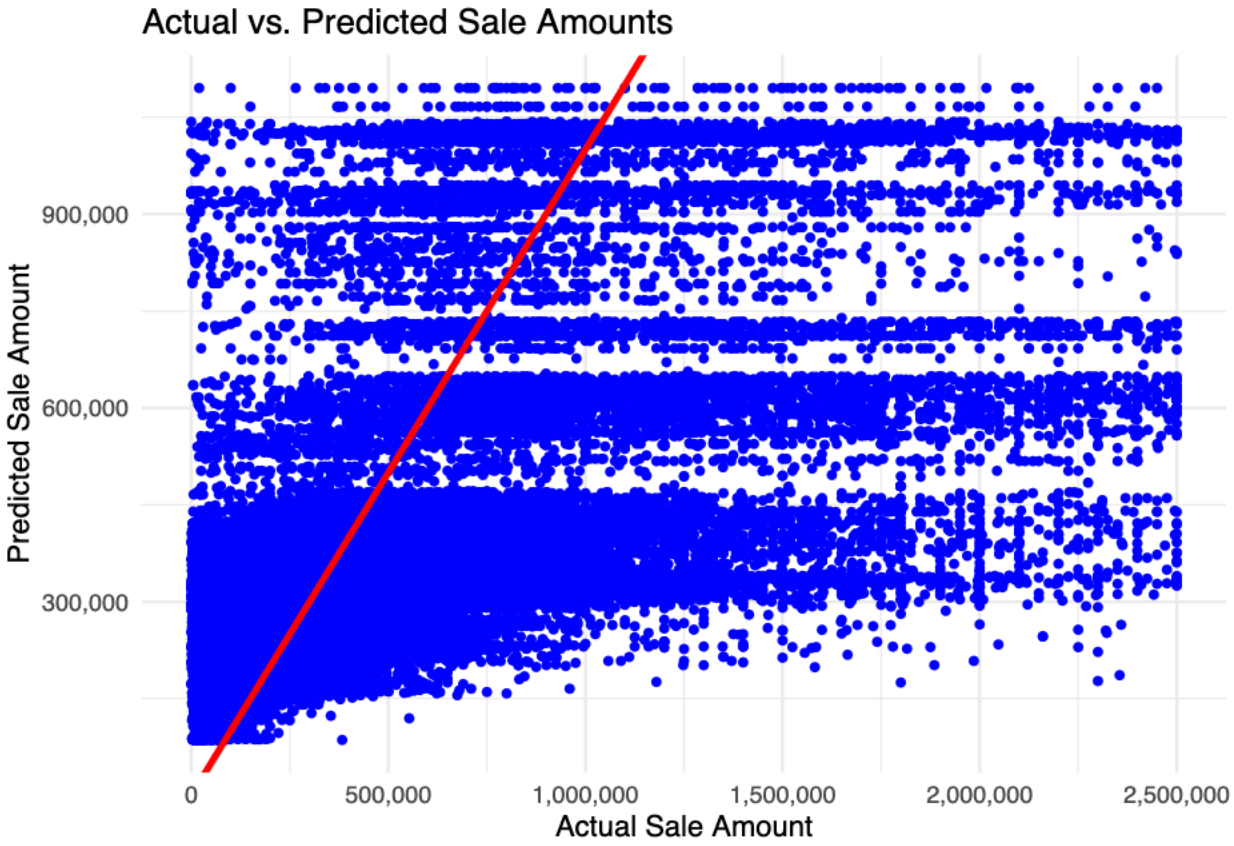
The mean squared error is very high. This is unsurprising, as it was very high on the train_test data set. We went from 68 billion on the train-test set to 69 billion on the final test set. Of course, this is not ideal. This is a pretty inaccurate prediction model due to the lack of stronger predicting variables.

Let's calculate the R-squared.

```
## [1] "R-squared: 0.338406627910605"
```

The R-squared returns roughly 33-34%. This means the ranger model is only explaining about 33-34% of the variability within the dataset. This is very comparable to the 33-34% the train_test set's result. It looks like we did not overtrain nor overfit. That is a very good thing. Of course, we'd like this number to be much higher and closer to 90%, but, we at least have a win by not overtraining nor undertraining the ranger random forest model.

Let's plot the actual values versus the predicted values.



We have more good news. The final test set has a similar-looking graph to the train test set. This is further evidence we had a good model that wasn't overtrained. Of course, our predictors still aren't iron-clad, but at least we have consistency in our relative inaccuracy due to lack of strength in our predictors. We again see the clustering beneath 1 million of actual sale amount and beneath 500,000 of predicted sale amount.

Let's predict the XLGBoost model on the test set and view the first ten rows of predictions.

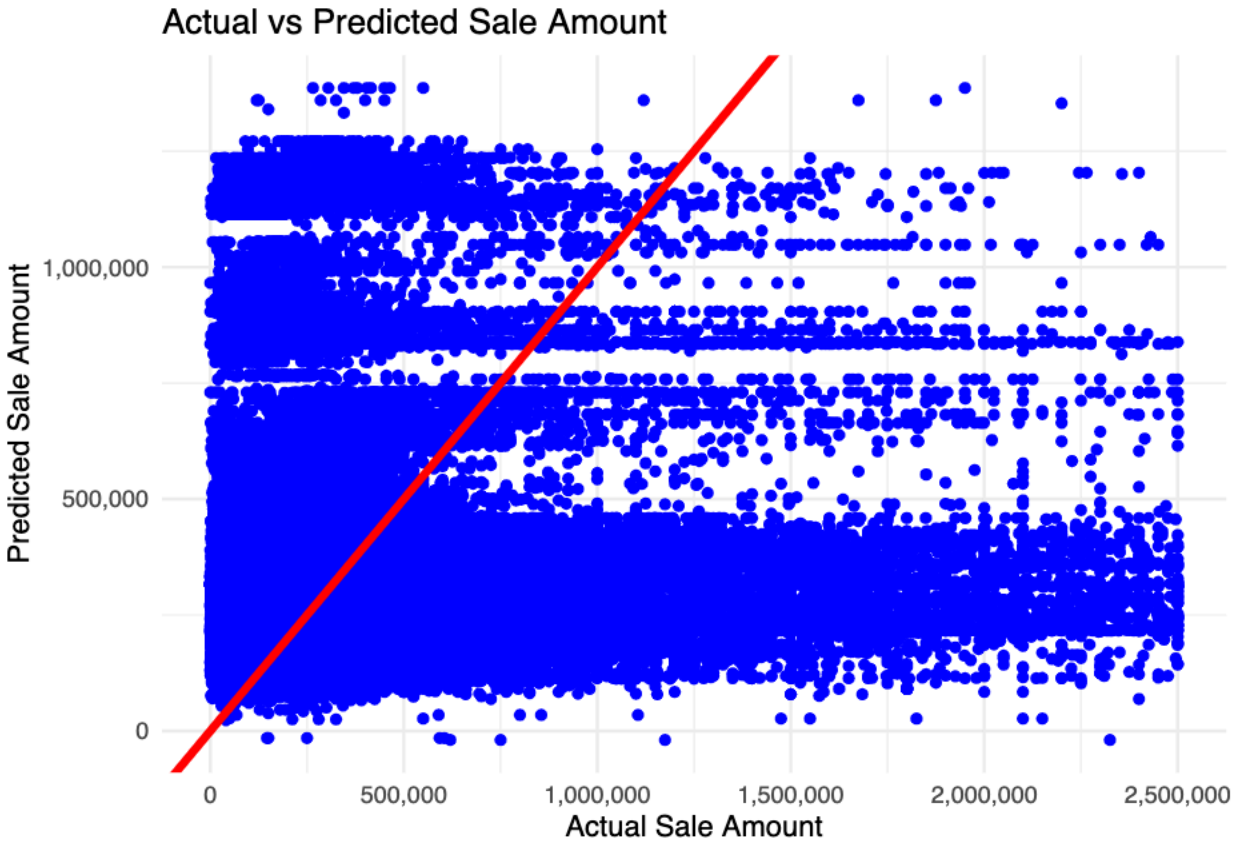
```
##  [1] 293706.7 376034.5 402986.0 410424.4 290946.7 336601.3 358078.0 343958.4
##  [9] 188635.3 312023.0
```

Now, let's evaluate the model's performance. Let's calculate the RMSE.

```
## RMSE: 366309.7
```

The RMSE is roughly 366,000. This is relatively poorly comparable to the roughly 222,000 we had before. It's over a 50% swing, which is pretty significant. We do expect the RMSE to rise in a final test set. I don't believe the XGBoost model was overtrained.

Let's plot the actual sale amounts versus the predicted sale amounts to see if the graph looks largely different since there was a greater than 50% increase in the RMSE.



This graph is super interesting. As we can see, the graph is similar to the one we obtained in the XGBoost model in the train_test data set. However, we do see a lot of clusteringin the left side of the graph with actual sale amounts less than 500,000 and predicted sale amounts higher than 750,000. That wasn't as pronounced before. The prediction model worked on the test very similarly as it did on the train_test set. It has a slightly more pronounced cluster towards higher predicted sale amounts for lower actual sale amounts. This again shows that the predictors in the original data set are not strongly tied to the target variable of sale amount; however, they are somewhat tied to it. If we had square footage, beds/baths count, and a few more variables, then tied them to the town, Date.Recorded, and property type, we'd have a rock solid prediction model.

It all comes down to having good data.

Conclusion

This final project was a wonderful exercise to stretch the brains of data analysts wanting to learn R. This specific data set has given us a wonderful lesson in the importance of having good quality data. While important, the volume of the data isn't everything. It sometimes is quality over quantity. This dataset is pretty limited in helping us accurately predict sale prices. Perhaps we could have grouped streets and towns together to help improve the prediction model. To be frank, I don't believe that it would help that much because in my home metropolitan area of Arizona, one street name will go on for 30 miles and cover a wide array of homes with a wide array of values. Further north, McClintock Drive has 3 million dollar homes, further south, it has 100 year old dilapidated shacks that are 100 square feet with a property value of 50 grand. Perhaps we'd have to group by the street numbers. Perhaps the 3000's of McClintock Drive have a higher property value because that neighborhood is grouped together in the 3000's of that street number, and the 10,000's of that street number are in lower-income neighborhoods. We could have attempted to separate out the street addresses and grouped by street number and street name in the towns. In reality, however, the easiest way to determine a property's value is by square footage and bed bath count. Right after that is location, so street number and street name, town, year, and property type. Of course, if you talk to a

licensed appraiser, they could give you a hundred other things that would determine a property's value. I know because I read their 50 page reports many times a week on many different properties. This has been a great exercise that has taught us that certain variables are more strongly linked to affecting certain outcomes than other variables. If we're going to analyze data, we need to know what our strong predictors are. If we don't know what they are, then trial and error in the machine learning models (and visualizing through graphs) is the best way to figure out which predictors are more strongly linked. Usually, if you're an analyst in an industry, you likely know at least a few of the important variables that are linked to certain outcomes. I'm excited to take what I've learned and then apply it to my company's data to see what useful information I can find and show my managers. I believe I will find some useful things that will help our company head into new markets and perhaps pair down a few current markets. I do know we keep record of square footage, so I'd be highly interested in throwing that into a machine learning model to see what its affect is.

All in all, this final project has really solidified my confidence in taking on real-world analysis projects in R. I must admit, before taking this class, I wondered why my managers were so interested in me learning a different data science software language besides Excel and the formulas and graphs I already know in Excel. Now, at the end of this class, specifically at the end of this final project analyzing real estate transactional data, I see why. I can help us find new markets where property values are trending up year by year, month by month, where average time on the market is trending down, where inventory levels are lowering, where price per square foot is rising, and more. I can help us take a look at our current markets we lend in and see if property values are stagnating or lowering, or even are rising slower than other markets we aren't lending in. I can help us analyze if average time on market in a city is increasing, where inventory levels are rising, where price per square foot is lowering, and more. This is all because of this final project. I'm excited to see where this newfound knowledge and experience can take me and my company.

Thank you Rafael Izarry for putting together this course. Thank you for EdX and HarvardX for hosting this great platform.

-Cade Westlake

References

Bobbitt, Z. (2020, November 30). XGBoost in R: A Step-by-Step Example. Statology. https://www.statology.org/xgboost-in-r/

Devon, D. (2023, December 7). Real estate sales 2001-2020. Kaggle. https://www.kaggle.com/datasets/derrekdevon/real-estate-sales-2001-2020?resource=download

GeeksforGeeks. (2024, April 22). Ranger function in R. GeeksforGeeks. https://www.geeksforgeeks.org/ranger-function-in-r/

Irizarry, R. (n.d.). HarvardX PH125.8x Data Science: Machine Learning. edX. https://learning.edx.org/course/course-v1:HarvardX+PH125.8x+1T2024/home

Rossiter, D. G. (2023, April 28). Random forests two ways. CSS Cornell EDU. https://www.css.cornell.edu/faculty/dgr2/_static/files/R_html/CompareRandomForestPackages.html

Srivastava, T. (2024, November 11). How to Use XGBoost Algorithm in R?. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/

Wright, M. N. (n.d.). Ranger: Ranger. RDocumentation. https://www.rdocumentation.org/packages/ranger/versions/0.16.0/topics/ranger

Wright, M. N., Wager, S., & Probst, P. (2024, November 8). Ranger: A fast implementation of random forests. CRAN.R.Project. https://cran.r-project.org/web/packages/ranger/ranger.pdf

XGBoost R Tutorial. Read the Docs. (2022). https://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html

Yuan, J. (2024, July 22). Xgboost: Extreme Gradient Boosting. Cran.R.Project. https://cran.r-project.org/web/packages/xgboost/xgboost.pdf