Real Estate Linear Regression Program

Mission Statement of The Project

The purpose of the Real Estate Linear Regression Program is to:

1. Gather data house from real estate websites (like Realator.com).
2. Use a Linear Regression algorithm to find patterns in the data.
3. Generate weights for the features.
4. Finally, be able to generate a plausible cost from a set of given features.

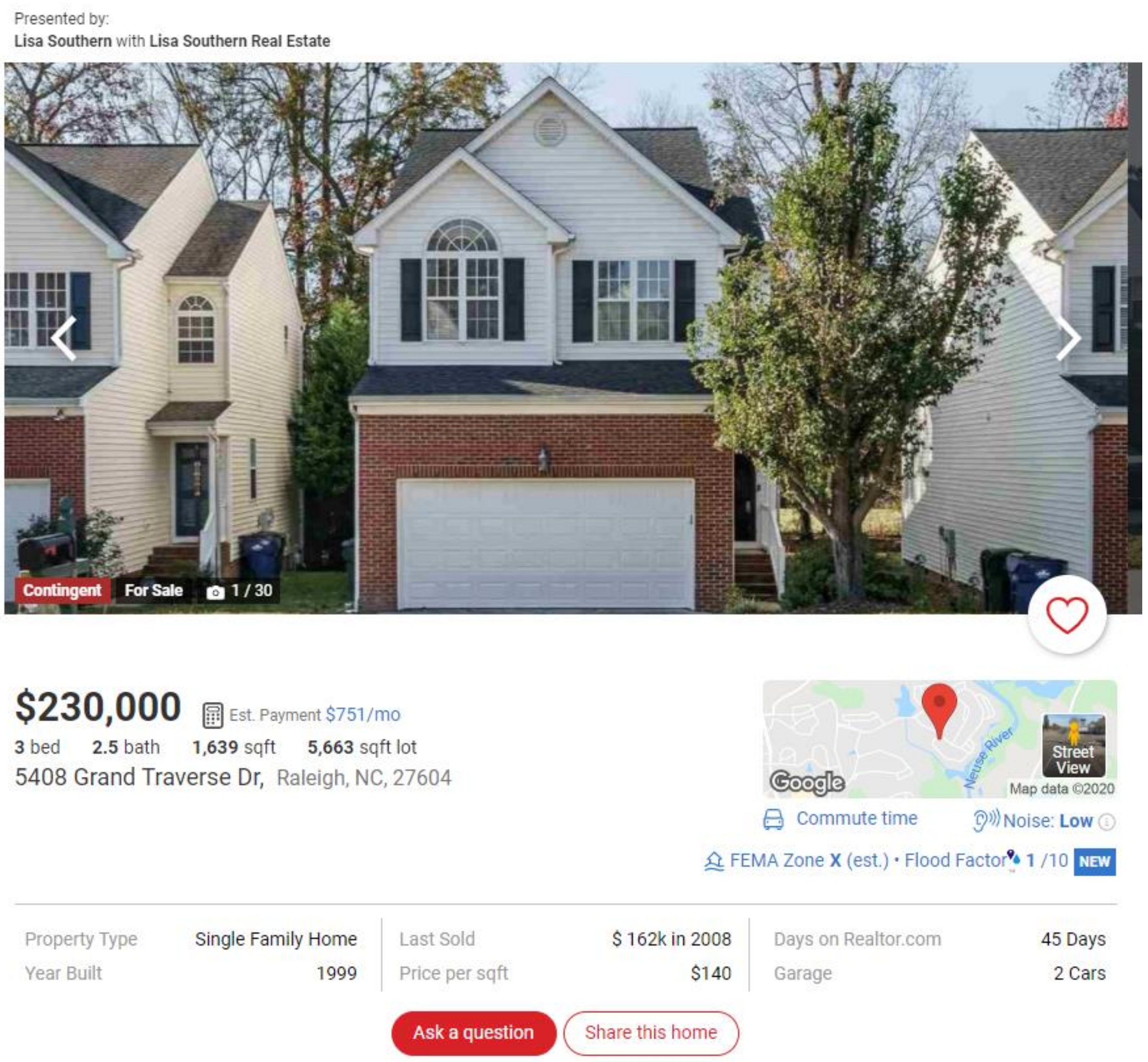
Part 1: Gathering House Data

Considering that I have never worked in real estate before, I really had to do some research in what things determined the worth of a home and where I could potentially procure such information about currently listed homes. One idea that I had initially was find some sort of real estate registry that kept track of the prices that homes sold for. Hopefully, I could also gather some more information about the home itself. However, because I have taken courses in Economics, I know that a house with all the exact same features and design will most likely be more expensive in 2021 than it would be in 1972. This is squarely upon the shoulders of inflation. This means that if I incorporated past house data into my algorithm, the houses sold in earlier time periods will negatively affect the accuracy of current housing (because it would skew so much lower). A good analogy might be in predicting the price of a basic cola from a gas station. If I only take account the prices of this cola around today, I will certainly be able to predict a reasonable price for the soda today. However, if I incorporate prices from the 1930s as well (when Coca-Cola was only 5 cents), my prediction would be erroneously skewed lower.

Another flaw with this approach would be finding real estate registries for each state. They would all most likely have vey different interfaces, as such it would be quite difficult to gather data on from each individual state. A more standardized approach is in order.

With all of this in mind, I decided to find a website that listed current homes on the market and allowed you to specify what zip code that you were interested in. Apparently Realator.com is one of the most popular real estate websites there is, and therefore I decided to use it. Because of how popular it is, there is a good chance that I can extract a lot of different house data from it.

Every house in my data set has looks like this:



Presented by:
Lisa Southern with Lisa Southern Real Estate

$230,000 Est. Payment $751/mo

3 bed 2.5 bath 1,639 sqft 5,663 sqft lot

5408 Grand Traverse Dr, Raleigh, NC, 27604

Commute time Noise: Low

FEMA Zone X (est.) • Flood Factor 1 /10 NEW

Property Type	Single Family Home	Last Sold	$ 162k in 2008	Days on Realtor.com	45 Days
Year Built	1999	Price per sqft	$140	Garage	2 Cars

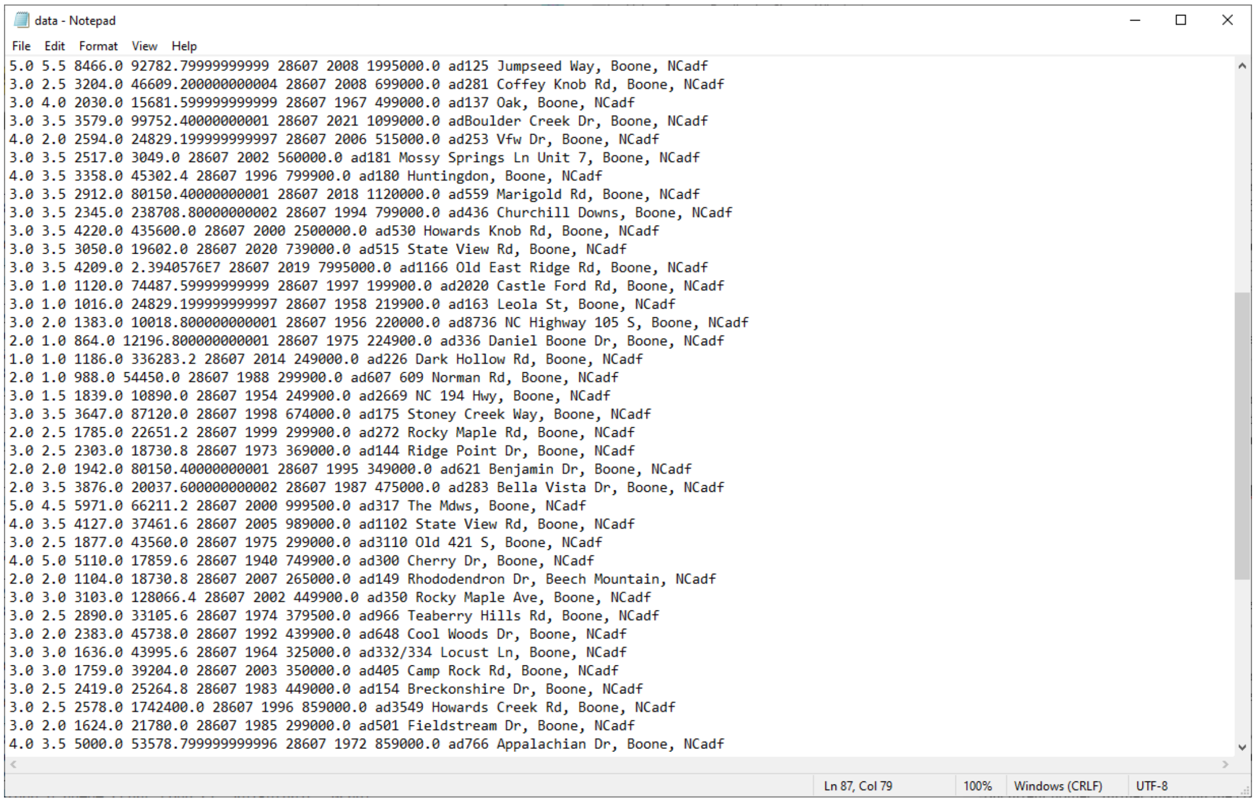
The things I believe are important to a home price is the number of bedrooms, bathrooms, square footage of the interior, the square footage of the exterior, and finally its age. Because just about every house listed on Realator.com has these features, I will make them the features of each house in the dataset.

Gathering the data from the website required utilizing the Jsoup library in java. Jsoup is a HTML web parser which allows me to rip information from a website. I had utilized this library previously in other projects so utilizing it again was not an issue. However, an issue I had never seen before is bot detection. Realator.com is rife with bot detection, and as such most of my usual simple methods for sifting through websites no longer work. As such, some sacrifices had to be made in efficiencies. While my Java class works (as in if given a zip code, it will find all of the homes registered within that zip code on Realator.com and generate a data file that can be used in conjunction with the Linear Regression Python Program), it takes an average of 13 seconds to generate a digital home from each page. This can make large zip codes take some extra time, but not an absolutely abysmal amount though. Another unfortunate consequence of the bot detection is the loss of user access.

Allow me to explain, the java class uses the Robot Class. The Robot Class is special because it allows java itself to access your keyboard and mouse controls. Now this means that if one were to use this class, they would not be able to effectively use their computer until the process was completed. I made this program specifically for an older computer in my home that no one uses personally. That way it can be given as much time as its required and we don't have to really worry about things users doing things while the program is running. While I am aware how unfortunate this can be, I don't think there can be another way unless one were to find a way around captcha checks.

Gathering the house data is really just an exercise in string manipulation. In fact, the text that links to each of the houses actually contains 5 aspects of data we could use. However, the text never contains the price or the year the house was built. For that my program actually downloads the page from the browser itself. This downloaded page is then sent to the program for more string manipulation. If we didn't download it, the bot detection would catch the program and then we wouldn't be able to continue. The program does perform some weeding out as certain houses don't list their lot acreage.

When house data is gathered it is placed in a .dat file labeled data. Each data file looks like this:



```
5.0 5.5 8466.0 92782.79999999999 28607 2008 1995000.0 ad125 Jumpseed Way, Boone, NCadf
3.0 2.5 3204.0 46609.200000000004 28607 2008 699000.0 ad281 Coffey Knob Rd, Boone, NCadf
3.0 4.0 2030.0 15681.599999999999 28607 1967 499000.0 ad137 Oak, Boone, NCadf
3.0 3.5 3579.0 99752.40000000001 28607 2021 1099000.0 adBoulder Creek Dr, Boone, NCadf
4.0 2.0 2594.0 24829.199999999997 28607 2006 515000.0 ad253 Vfw Dr, Boone, NCadf
3.0 3.5 2517.0 3049.0 28607 2002 560000.0 ad181 Mossy Springs Ln Unit 7, Boone, NCadf
4.0 3.5 3358.0 45302.4 28607 1996 799900.0 ad180 Huntingdon, Boone, NCadf
3.0 3.5 2912.0 80150.40000000001 28607 2018 1120000.0 ad559 Marigold Rd, Boone, NCadf
3.0 3.5 2345.0 238708.80000000002 28607 1994 799000.0 ad436 Churchill Downs, Boone, NCadf
3.0 3.5 4220.0 435600.0 28607 2000 2500000.0 ad530 Howards Knob Rd, Boone, NCadf
3.0 3.5 3050.0 19602.0 28607 2020 739000.0 ad515 State View Rd, Boone, NCadf
3.0 3.5 4209.0 2.3940576E7 28607 2019 7995000.0 ad1166 Old East Ridge Rd, Boone, NCadf
3.0 1.0 1120.0 74487.59999999999 28607 1997 199900.0 ad2020 Castle Ford Rd, Boone, NCadf
3.0 1.0 1016.0 24829.199999999997 28607 1958 219900.0 ad163 Leola St, Boone, NCadf
3.0 2.0 1383.0 10018.800000000001 28607 1956 220000.0 ad8736 NC Highway 105 S, Boone, NCadf
2.0 1.0 864.0 12196.800000000001 28607 1975 224900.0 ad336 Daniel Boone Dr, Boone, NCadf
1.0 1.0 1186.0 336283.2 28607 2014 249000.0 ad226 Dark Hollow Rd, Boone, NCadf
2.0 1.0 988.0 54450.0 28607 1988 299900.0 ad607 609 Norman Rd, Boone, NCadf
3.0 1.5 1839.0 10890.0 28607 1954 249900.0 ad2669 NC 194 Hwy, Boone, NCadf
3.0 3.5 3647.0 87120.0 28607 1998 674000.0 ad175 Stoney Creek Way, Boone, NCadf
2.0 2.5 1785.0 22651.2 28607 1999 299900.0 ad272 Rocky Maple Rd, Boone, NCadf
3.0 2.5 2303.0 18730.8 28607 1973 369000.0 ad144 Ridge Point Dr, Boone, NCadf
2.0 2.0 1942.0 80150.40000000001 28607 1995 349000.0 ad621 Benjamin Dr, Boone, NCadf
2.0 3.5 3876.0 20037.600000000002 28607 1987 475000.0 ad283 Bella Vista Dr, Boone, NCadf
5.0 4.5 5971.0 66211.2 28607 2000 999500.0 ad317 The Mdws, Boone, NCadf
4.0 3.5 4127.0 37461.6 28607 2005 989000.0 ad1102 State View Rd, Boone, NCadf
3.0 2.5 1877.0 43560.0 28607 1975 299000.0 ad3110 Old 421 S, Boone, NCadf
4.0 5.0 5110.0 17859.6 28607 1940 749900.0 ad300 Cherry Dr, Boone, NCadf
2.0 2.0 1104.0 18730.8 28607 2007 265000.0 ad149 Rhododendron Dr, Beech Mountain, NCadf
3.0 3.0 3103.0 128066.4 28607 2002 449900.0 ad350 Rocky Maple Ave, Boone, NCadf
3.0 2.5 2890.0 33105.6 28607 1974 379500.0 ad966 Teaberry Hills Rd, Boone, NCadf
3.0 2.0 2383.0 45738.0 28607 1992 439900.0 ad648 Cool Woods Dr, Boone, NCadf
3.0 3.0 1636.0 43995.6 28607 1964 325000.0 ad332/334 Locust Ln, Boone, NCadf
3.0 3.0 1759.0 39204.0 28607 2003 350000.0 ad405 Camp Rock Rd, Boone, NCadf
3.0 2.5 2419.0 25264.8 28607 1983 449000.0 ad154 Breckonshire Dr, Boone, NCadf
3.0 2.5 2578.0 1742400.0 28607 1996 859000.0 ad3549 Howards Creek Rd, Boone, NCadf
3.0 2.0 1624.0 21780.0 28607 1985 299000.0 ad501 Fieldstream Dr, Boone, NCadf
4.0 3.5 5000.0 53578.799999999996 28607 1972 859000.0 ad766 Appalachian Dr, Boone, NCadf
```

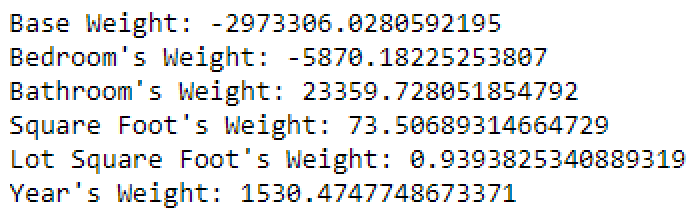
As an example I used the data from the 28607 zip code. The first column is the number of bedrooms, the second is the number of bathrooms, the third is the square footage of the interior of the home, the fourth column is the square footage of the lot, the fifth is the zip code (which should generally remain constant. Although you could tweak this if you wanted to), the seventh is the listed price of the home, and finally the address is the last column. The address is encased between the text ad and adf. This makes extracting the address easier, as you don't have to worry about spaces.

Part 2: Linear Regression Algorithm

Luckily, I already have a linear regression program built so I really didn't have to add many changes. This program was built in python and utilizes the NumPy library heavily. It uses simple linear algebraic properties to find the optimal weights for the linear regression algorithm. All that really needed changing was the inclusion of a method to take in the data from .dat file and another method to show which weights lined up to which feature.

Part 3: Generating Weights

Using the standard Linear Regression Algorithim, the program can generate weights for the different features. I initially assumed that every weight would be positive and add up to generate an appropriate cost, but it actually turned out that some these weights actually have a negative cost to counter act other weights. This is fine as long as it generates the appropriate prices, but I found it a most interesting occurrence. This is what weight generation looks like:



```
Base Weight: -2973306.0280592195
Bedroom's Weight: -5870.18225253807
Bathroom's Weight: 23359.728051854792
Square Foot's Weight: 73.50689314664729
Lot Square Foot's Weight: 0.9393825340889319
Year's Weight: 1530.4747748673371
```

Part 4: Generate Plausible Cost

In order to test whether the costs generated are plausible, I tried testing it on the elements of the dataset. The difference between the predicted value and the actual value I have dubbed "the delta." I then created an array that found the delta for each home. With this I can find the average delta and the total delta. What I found was surprising.

The algorithm actually turned out to only be a moderately successful predicter of costs. For example, in my current zip code, 27545, the average delta discrepancy turned out to be 30,000$. This isn't awful, but it certainly isn't optimal. It looks like there is some tweaking to be done.

Potential Additions and Tweaking

Seeing such striking discrepancies between the values predicted and the actual values certainly implies a great deal more work is required to make this program achieve its full potential.

One of the first things that comes to mind when working with machine learning algorithms is giving the computer more data. The criterion to be included in the dataset is stricter than one might believe, making some zip codes rather lacking in the information department. One way this might be done is by further extending the web crawling program to not only work with Realator.com, but also with other popular web based real estate websites like Trulia and Zillow. In fact, while building the web scrapping program, I found that Zillow and Realator.com had several concurrent homes, further implying the connection between these two websites.

Another thing that has also come to mind is incorporating clustering. There are other “style” variables on the Relator.com pages, like Ranch style homes, Traditional homes, Contemporary homes, etc. It has certainly come to my attention that despite two homes having the exact same age, square footage, acre size, and number of rooms, they can have two starkly different prices because of their style. Placing homes with similar types together, I believe, will create better predictors for that style. In a lot of ways my current algorithm is comparing every car on a lot with each other instead of comparing sports cars with sports cars and trucks with trucks. Of course, a worry with this approach will be a potential dearth of usable data for each type, but perhaps this could be remediated by taking the actions mentioned earlier. Perhaps weeding things out further by removing outliers is also in order as well.

This next potential tweak has to do with data gathering. Obviously, the method I am currently using isn’t as user friendly as I would like it to be, nor is it as fast. Most of the potential work in this area almost fully hinge on whether I can somehow write a program that can avoid bot detection. Otherwise, the current long method is the only way to get the data needed. There is no simple fix to this issue, it will simply require a great deal more work to determine which routes provide the best path without detection. One solution might hinge on using a different class from Jsoup (which is only a HTML parser.) Finding a parser that can utilize JavaScript might seem more like a real user.

With all of these solutions in mind, I certainly believe that I can improve upon this program!