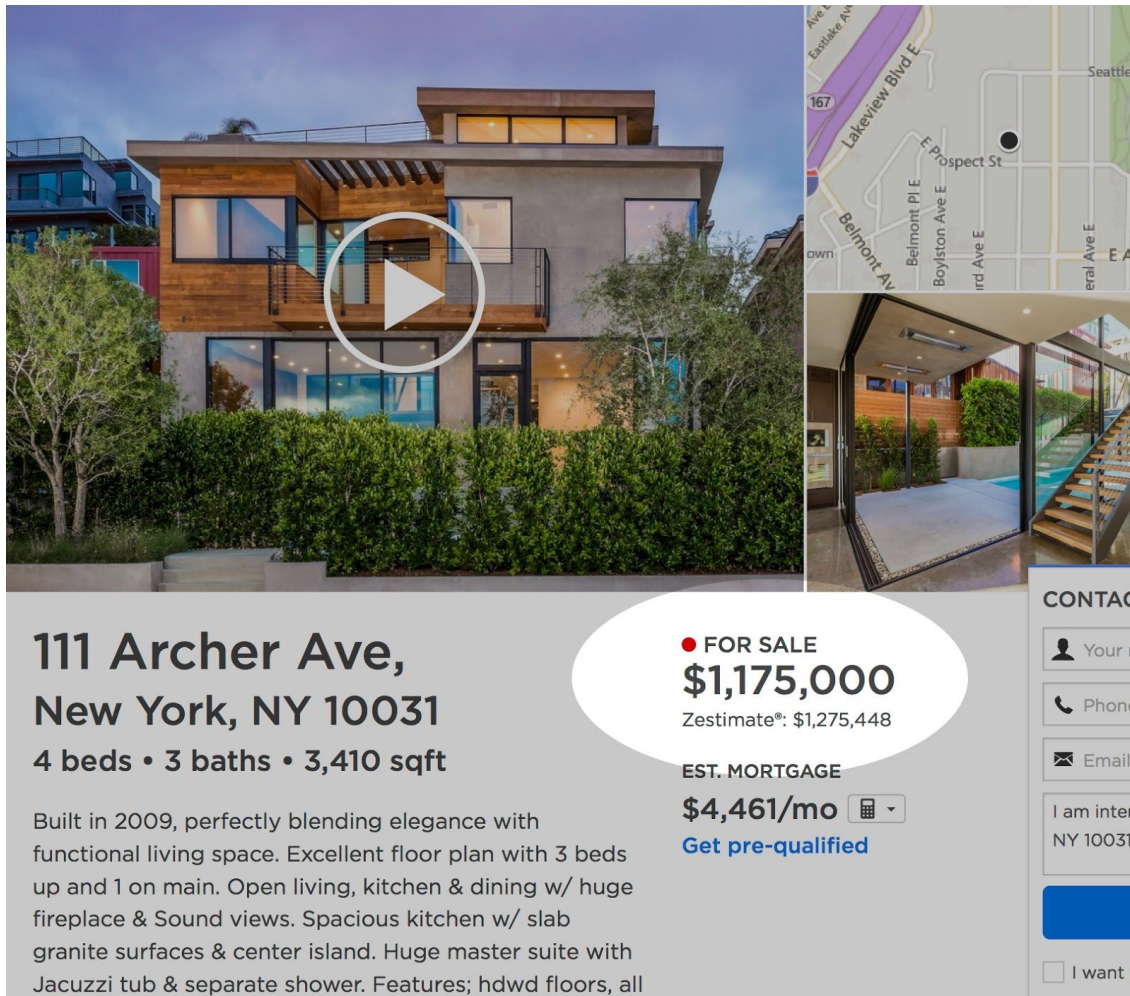



Zillow's Home Value Prediction







**111 Archer Ave,
New York, NY 10031**
4 beds • 3 baths • 3,410 sqft

Built in 2009, perfectly blending elegance with functional living space. Excellent floor plan with 3 beds up and 1 on main. Open living, kitchen & dining w/ huge fireplace & Sound views. Spacious kitchen w/ slab granite surfaces & center island. Huge master suite with Jacuzzi tub & separate shower. Features; hdwd floors, all

FOR SALE
\$1,175,000
Zestimate®: \$1,275,448

EST. MORTGAGE
\$4,461/mo 
[Get pre-qualified](#)

CONTACT
 Your name
 Phone
 Email
I am interested in NY 10031.

☐ I want to see this property

By
Pasindu Siriwardena
Ivan Yu
Chan In Kou
Carlos Larios-Solis
Zachary Portillo

Table of Contents:

Introduction

Project description (purpose, what is Zillow?)	pg. 3
Data Analysis	pg. 3

Possible Methods/Algorithms:

K-nearest neighbor	pg. 9
Decision Tree	pg. 9
Random Forest	pg. 9
K-means Clustering Algorithm	pg. 10
Linear Regression	pg. 10

Applying Algorithms:

2017 Data	pg. 11
2017 Data Visualized	pg. 12
2016 Data	pg. 16
2016 Data Visualized	pg. 17
2016 and 2017 Data	pg. 19
2016 and 2017 Data with Outliers Removed	pg. 22
2016 and 2017 Data with Outliers Removed and Reduced Fields	pg. 23
2016 and 2017 Data with Reduced Fields	pg. 24
2016 and 2017 Data with Outliers Removed and Reduced Fields Visualized	pg. 25

Conclusion:

Conclusion	pg. 26
------------	--------

Responsibilities:

Responsibilities	pg. 27
------------------	--------

Introduction

What is Zillow?

Zillow is the leading real estate marketplace dedicated to matching consumers with property, offering them the data and knowledge of home-related information.

What is a Zestimate?

Zillow's Zestimates are estimated home values based on over 7.5 million homes around the country. Zestimates analyze hundreds of home variables to predict the value of the home or property.

Project Overview

Zillow wants us to predict the log-error between the 'Zestimate' and the actual sale price of the home.

$$\text{logerror} = \log(\text{Zestimate}) - \log(\text{SalePrice})$$

Objective(s)

Using any of the machine learning algorithms learned in the Data Science course, predict the log-error using the dataset Zillow provides

There are two main approaches to supervised learning: Classification and Regression. In this project, we will be using regression models to predict the log-error because it is a continuous valued output. In other words, our labels are continuous, and not discrete.

Data

Zillow has provided us with the following data files:

1. **Properties_2016.csv** - All properties with their home features for 2016
2. **Properties_2017.csv** - All properties with their home features for 2017
3. **Sample_submission.csv** - Sample submissions in the correct format
4. **Train_2016.csv** - The training set with the transactions from January-December 2016
5. **Train_2017.csv** - The training set with the transactions from January-December 2017
6. **Zillow_data_dictionary.csv** - A dictionary explaining what exactly each feature means

Both **properties_2016.csv** and **properties_2017.csv** contained nearly 3 million entries each!

Properties are all located in Los Angeles, Orange County and Ventura, California.

Let's look inside the data to get a better understanding what we are dealing with

First, let's put the **properties_2017.csv** and **train_2017.csv** into **Pandas Data Frames**

```
df_properties = pd.read_csv('./properties_2017.csv', low_memory=False)
df_properties.shape

(2985217, 58)
```

```
df_train = pd.read_csv('./train_2017.csv')
df_train.shape

(77613, 3)
```

As you can see, our properties file contains a much more substantial amount of data than our training set. This means, there will be labels missing from a lot of our entries.

Let's Look inside our data:

properties_2017.csv

```
In [4]: df_properties.head()

Out[4]:
```

	parcelid	airconditioningtypeid	architecturalstyletypeid	basementsqft	bathroomcnt	bedroomcnt	buildingclasstypeid	buildingqualitytypeid	calculatedbathnt
0	10754147	NaN	NaN	NaN	0.0	0.0	NaN	NaN	Na
1	10759547	NaN	NaN	NaN	0.0	0.0	NaN	NaN	Na
2	10843547	NaN	NaN	NaN	0.0	0.0	5.0	NaN	Na
3	10859147	NaN	NaN	NaN	0.0	0.0	3.0	6.0	Na
4	10879947	NaN	NaN	NaN	0.0	0.0	4.0	NaN	Na

5 rows × 58 columns

train_2017.csv

```
In [6]: df_train.head()

Out[6]:
```

	parcelid	logerror	transactiondate
0	14297519	0.025595	2017-01-01
1	17052889	0.055619	2017-01-01
2	14186244	0.005383	2017-01-01
3	12177905	-0.103410	2017-01-01
4	10887214	0.006940	2017-01-01

Between both files, you might notice a similar feature: *parcelid*

We will use this feature to merge the properties file and the training set, in order to train/test the data. Once we successfully merge the two datasets, we will come up with our solution to predict the log-error.

Joined_data_set

```
# Since theres a large amount of differences in entries
# We merge it, to reduce the amount of entries we don't need
joined_data_set = pd.merge(df_properties, df_train)
joined_data_set.shape

(77613, 60)
```

Now let's look at the data types of our joined data set

```
data_type = joined_data_set.dtypes.reset_index()
data_type.columns = ["Feature Column", "Data Type"]
data_type
```

Feature Column Data Type

0	parcelid	int64
1	airconditioningtypeid	float64
2	architecturalstyletypeid	float64
3	basementsqft	float64
4	bathroomcnt	float64
5	bedroomcnt	float64
6	buildingclasstypid	float64
7	buildingqualitytypeid	float64
8	calculatedbathnbr	float64
9	decktypeid	float64
10	finishedfloor1squarefeet	float64
11	calculatedfinishedsquarefeet	float64
12	finishedsquarefeet12	float64
13	finishedsquarefeet13	float64
14	finishedsquarefeet15	float64
15	finishedsquarefeet50	float64
16	finishedsquarefeet6	float64
17	fips	float64
18	fireplacecnt	float64
19	fullbathcnt	float64
20	garagecarcnt	float64
21	garagetotalsqft	float64

22	hashottuborspa	object
23	heatingorsystemtypeid	float64
24	latitude	float64
25	longitude	float64
26	lotsizesquarefeet	float64
27	poolcnt	float64
28	poolsizeum	float64
29	pooltypeid10	float64
30	pooltypeid2	float64
31	pooltypeid7	float64
32	propertycountylandusecode	object
33	propertylandusetypeid	float64
34	propertyzoningdesc	object
35	rawcensustractandblock	float64
36	regionidcity	float64
37	regionidcounty	float64
38	regionidneighborhood	float64
39	regionidzip	float64
40	roomcnt	float64
41	storytypeid	float64
42	threequarterbathnbr	float64
43	typeconstructiontypeid	float64
44	unitcnt	float64

45	yardbuildingsqft17	float64
46	yardbuildingsqft26	float64
47	yearbuilt	float64
48	numberofstories	float64
49	fireplaceflag	object
50	structuretaxvaluedollarcnt	float64
51	taxvaluedollarcnt	float64
52	assessmentyear	float64
53	landtaxvaluedollarcnt	float64
54	taxamount	float64
55	taxdelinquencyflag	object
56	taxdelinquencyyear	float64
57	censustractandblock	float64
58	logerror	float64
59	transactiondate	object

As you can see, most of the features have the data type of *float64*. Later we find that the features that did not have this data type were of least significance and contained the most *NaN* values.

```
# Now we take the data apart so we have usable data
X = joined_data_set[df_properties.keys()]
print(X.shape)
```

```
y = joined_data_set[df_train.keys()]
print(y.shape)
```

```
(77613, 58)
(77613, 3)
```

Now before splitting the data into testing and training sets, we take apart the data so it can be usable.

Now to visualize, we are going to see how many NaN values are in our data set

```
df_null_values = df_properties.isnull().sum(axis=0).reset_index()
df_null_values.columns = ['column_name', 'missing_count']
df_null_values = df_null_values.loc[df_null_values['missing_count']>0]

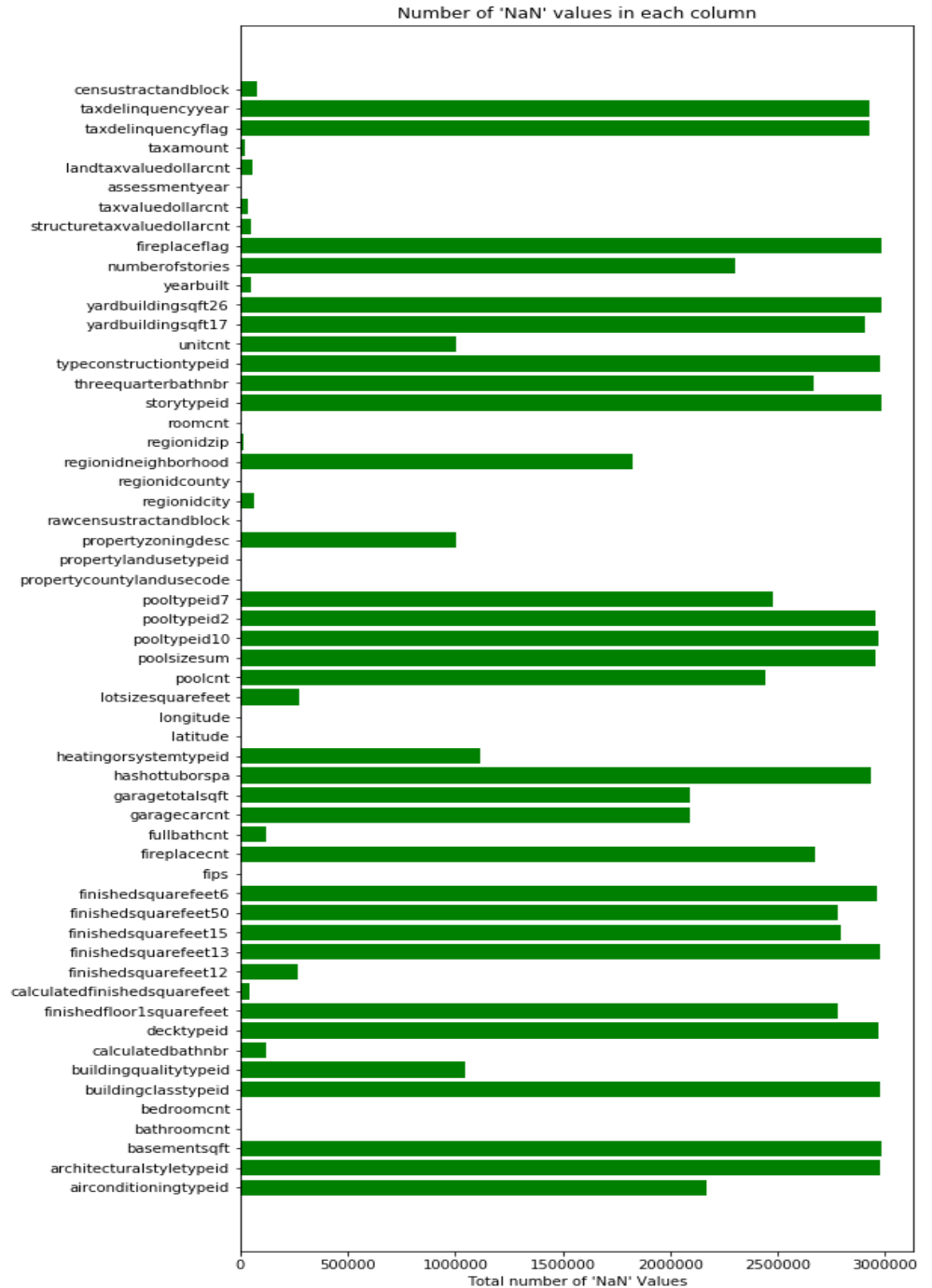
ind = np.arange(df_null_values.shape[0])
width = 1
fig, ax = plt.subplots(figsize=(8,19))
rects = ax.barh(ind, df_null_values.missing_count.values, color='green')
ax.set_yticks(ind)
ax.set_yticklabels(df_null_values.column_name.values, rotation='horizontal')
ax.set_xlabel("Total number of 'NaN' Values")
ax.set_title("Number of 'NaN' values in each column")
plt.show()
```

Looking at this chart, you can see columns that contained a substantial amount of “NaN” values. We wanted to visualize this because, we needed to know what columns were unnecessary to train. For instance, the column “*basementsqft*” has practically all values that contain “NaN”.

Which makes sense because not a lot of homes have basements, hence the “NaN” value.

What we had in mind, was to remove the columns that contained mostly “NaN” values.

For columns that had few “NaN” values in them, we replaced the value with a “0”



Wrapping up the data analyzation, we find that removing the features that contained the most “NaN” values would be fine. These features serve as least significance in our dataset.

Once finishing the removal process, we normalize our dataset and prepare for our algorithm analysis.

```
# Now we remove data thats hard to work with
y = y.drop('transactiondate', 1)
y = y.drop('parcelid', 1)

# Replacing NaN for all
X = X.fillna(0)

X = X.drop('hashottuborspa', 1)
X = X.drop('propertycountylandusecode', 1)
X = X.drop('propertyzoningdesc', 1)
X = X.drop('fireplaceflag', 1)
X = X.drop('taxdelinquencyflag', 1)
X = X.drop('parcelid', 1)

print(X.shape)
print(y.shape)

# And scale
X = preprocessing.scale(X)

(77613, 52)
(77613, 1)
```

Later, we will go more in depth with our data and how we used them in our algorithms

Possible Algorithms

K-Nearest Neighbors Classifiers

K-Nearest Neighbors classifiers is a supervised learning. It basically learning from labeled observations. Labels are the target outputs. For example, labels will be the rainy/sunny in lecture 2. This algorithm needs features (training data) and their known labels in training state, in order to train a model that provides future inputs with unknown labels. With our dataset, we can't really use classification because our labels are not categorical, which means we can't predict a discrete valued output. We are supposed to use regression since we are trying to predict a continuous valued output with number like housing price. Nearest Neighbor classification is no good because it makes decisions only based on one sample in the training set. Nearest Neighbor Classification would not work if we pick one bad sample with a group of same samples surrounding it. K-Nearest Neighbors classifiers is better because it is based on a group of K closest training samples in the feature space instead of only one. We can't use K-Nearest Neighbors because the training data is just numbers like logerror. For example, we can't be like out of the 5 NN: 4 have two bathrooms and 1 have one bathroom. And we can't say our prediction for the house price is \$10000.

Decision Tree Classifier

Decision Tree classifier is also a supervised learning. It is a classification. It makes a decision based on only one feature, two features, three features or more. It goes down like a tree, it predicts from one feature at the top of the tree, and then split the data samples and so on. The first feature at the top of the tree should be the one that provides the majority amount of information about the label trying to predict. For example, in lecture 5 and lecture 6, we used a topic on predicting survival on the Titanic. It could predict if someone will be more likely to survive based on their gender, age and so on. Decision Tree needs Entropy to measure the uncertainty. With more information, we could reduce uncertainty before splitting any data. Decision Tree could handle both numerical and categorical data. But it is still not an algorithm that we could use for our dataset because our training data is just log error.

Random Forest

Random Forest is part of ensemble learning. Ensemble learning uses base learners and combine the results of them using Voting to get better accuracy. Base learners are basically grouping many weak classifiers and combine it into a strong classifier to get better accuracy. An important step that Random Forest uses is Bootstrapping. Bootstrapping generate a few training datasets and train a decision tree for each one. Another method for Random Forest is using random selection of features to split on each node of each decision tree. Random Forest can handle missing given value and big features. Even though in lecture 16, it says that Random Forest can

handle hundreds of features and missing value. We did not know how to use Random Forest for our dataset.

K-Means Clustering Algorithm

K-Means Clustering Algorithm is an unsupervised learning. Which means that there are no labeled samples showing how the data should be grouped. Unsupervised learning is different from Supervised learning because one is to train a model that get new inputs with the known labels and Unsupervised learning only have features and nothing else. For example, in lecture 17, hunting for alien life is perfect example for Unsupervised learning, no one knows how aliens speak or move. Unsupervised learning goal is to discover hidden patterns from features itself. K-Means partition n data samples based on common properties into k clusters, so each sample will cluster to the nearest mean. First set the K position random in feature space, then update centroid locations of k to the mean location of the cluster and keep reassign till the k is almost to the center. We did not use K-Means Clustering Algorithm because we do know what we are looking for, which is the log error and dataset did provide more than just features.

Linear Regression Algorithm

The linear regression algorithm is a supervised learning algorithm that predicts a continuous valued output. It can be used in predicting stock prices, housing prices and value where the labeled samples are continuous. For example we can use linear regression to predict the annual rainfall of certain city or state for the upcoming year using the weather information that has already been collected in the past. By collecting multiple values from past years, we can calculate the predicted value of rain by plotting all the values and then creating our predictive linear model. This regression model is built based on the training set of values and then will be used to predict or estimate any new values. When using linear regression an error value is always present, but the error percentage should become smaller the better the training set becomes. Linear regression also has a special case called the “Univariate Linear Regression” where a regression model is created using a single input variable, a single feature. This was not used in our case because we had multiple features and no single feature was able to encompass all the features present in our dataset.

Applying the Algorithms

To explore the possibilities of variation within the models, we decided to test on certain scenarios. We split the tasks into 6 scenarios, each with a unique twist to show how root mean square error (RMSE) would be affected, with the end goal of reducing RMSE. Developing into scenarios was easy due to the fact that the data is already split for us to begin with. We started off with the 2017 data, 2016 data, 2016 and 2017 data combined, then ran 3 separate scenarios on the combined data set. We saw what would happen if we removed entries, and if any significant difference was made by doing so.

2017 Data

To explore if there was a difference in possible models, we decided to see what would happen on just the 2017 data. To do so we must first load the data, a step that had to be done each time.

```
df_properties = pd.read_csv('./properties_2017.csv', low_memory=False)
df_properties.shape

(2985217, 58)
```

```
df_train = pd.read_csv('./train_2017.csv')
df_train.shape

(77613, 3)
```

Upon loading the data we can see that there is a large discrepancy in the amount of provided data, and training data. This was a mind-bending scenario since in class we had yet to encounter such a thing. Our solution was to merge the datasets, thereby reducing the data to process. After reducing we split the data. Our new set was just over 2.5% the size of the original, a huge reduction, but the only way we could figure with our knowledge.

After reducing the data set we had

```
# Replacing NaN for all
X = X.fillna(0)
```

```
# Since theres a large amount of differences in entries
# We merge it, to reduce the amount of entries we don't need
joined_data_set = pd.merge(df_properties, df_train)
joined_data_set.shape

(77613, 60)
```

```
# Now we take the data apart so we have usable data
X = joined_data_set[df_properties.keys()]
print(X.shape)

y = joined_data_set[df_train.keys()]
print(y.shape)

(77613, 58)
(77613, 3)
```

to deal with all of the NaN that the data contained. Running ML algorithms on datasets that contain NaN is simply not possible! There were two routes that we could've gone with, the first being replacing the NaN values with the average of the column or replacing with 0.

We went with the latter, mainly because we saw most columns with missing data, contained a significant amount of missing data. Providing an average for the values would yield a similar result as zero.

Once replacing the NaN's we had to scale the data, we did this to be sure that the algorithms would work nicer. Apart from scaling we dropped a couple of columns that were composed of strings and knew were not useful. As we learned in class the next step was to break the data into test and training sets. After that we then defined and trained our models.

```
# Splitting into Testing and Training Data
testSize = 0.25
randomState = 100
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testSize, r
```

```
# Defining Regression Methods
my_linreg = linear_model.LinearRegression()
my_ridge = linear_model.Ridge(alpha=.5)
my_ridge_cv = linear_model.RidgeCV(alphas=np.logspace(-6, 6, 13))
my_lassolars = linear_model.LassoLars(alpha=.1)
my_lassolars_cv = linear_model.LassoLarsCV(cv=10)
```

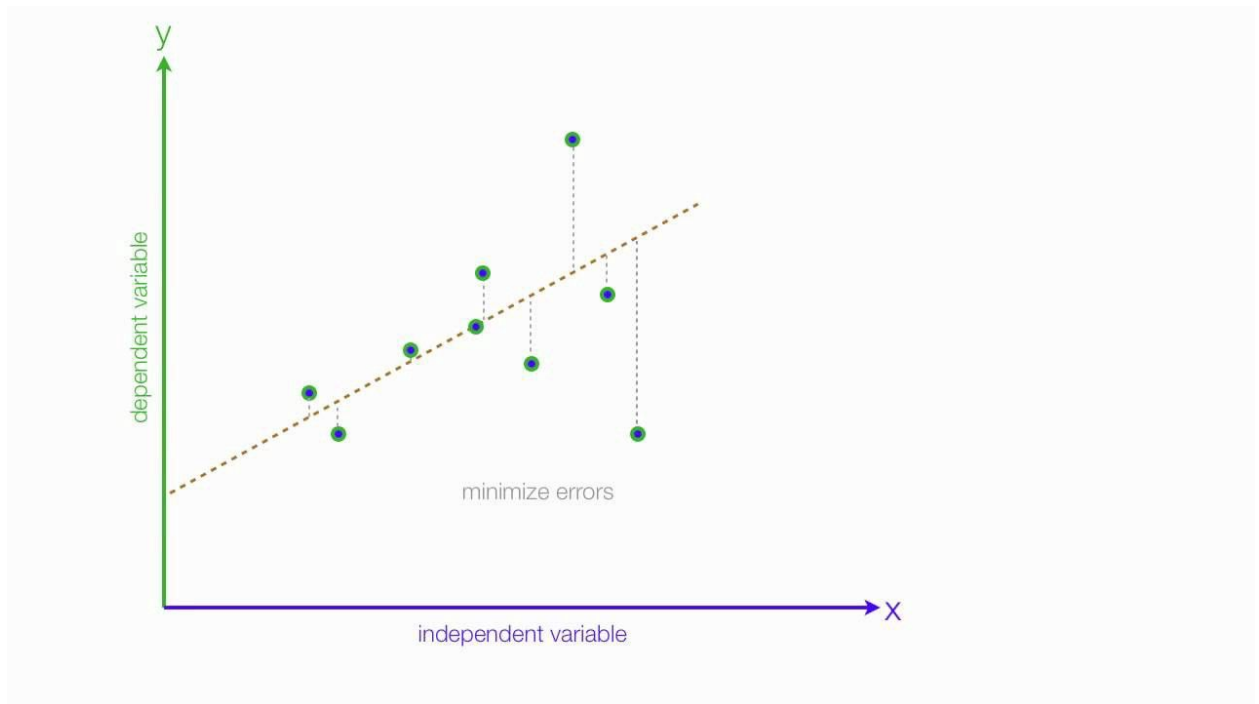
```
# Training Models
my_linreg.fit(X_train, y_train)
my_ridge.fit(X_train, y_train)
my_ridge_cv.fit(X_train, y_train)
my_lassolars.fit(X_train, y_train)
my_lassolars_cv.fit(X_train, y_train)
```

We chose five methods of modeling to gain a richer source from which to compare results. Once our models were finished training, we then fed our testing “y” data to see what predictions they would yield. Once the predictions were made we calculated Mean Square Error (MSE) using the Metrics package, then with NumPy found the RMSE. We found that Linear Regression held the highest value, and that Cross Validated Lasso Lars held the smallest.

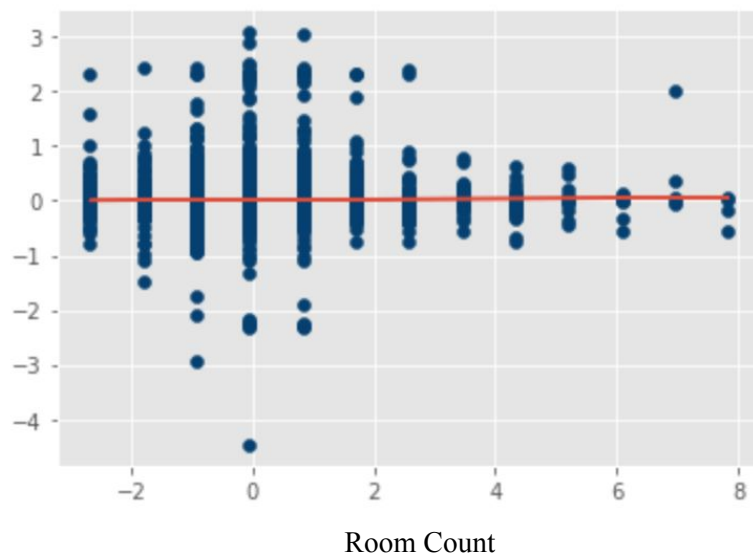
```
RSME Value Using Liar Regression:  0.17662309017944727
RSME Value Using Ridge Regression:  0.17661671553509561
RSME Value Using Ridge Regression Cross Validation:  0.17623084579531348
RSME Value Using Lasso Lars:  0.17642723889508136
RSME Value Using Lasso Lars Cross Validation:  0.17620702919334602
```

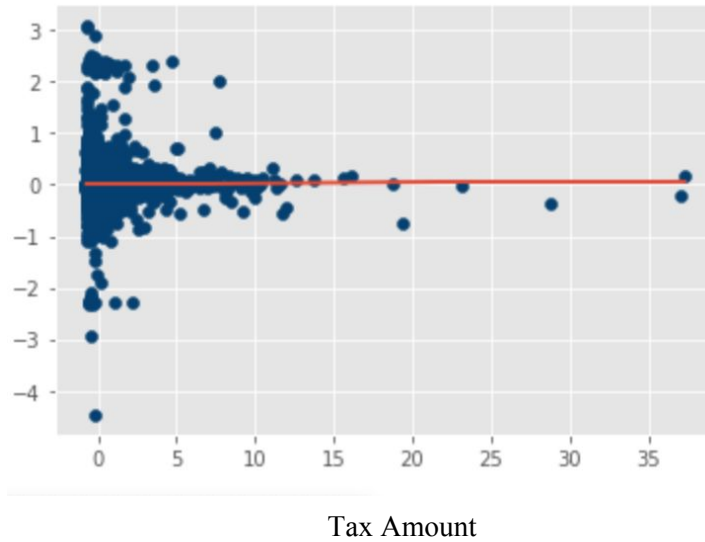
Visual Data Analysis

After running the algorithm we decided to plot each feature and compare the predicted logerror with the actual logerror. Linear regression works by comparing the independent variable to the dependent variable. Then, it would try to create a linear line as close to all the points as possible and use that to make predictions. The following is an example of how linear regression is supposed to work

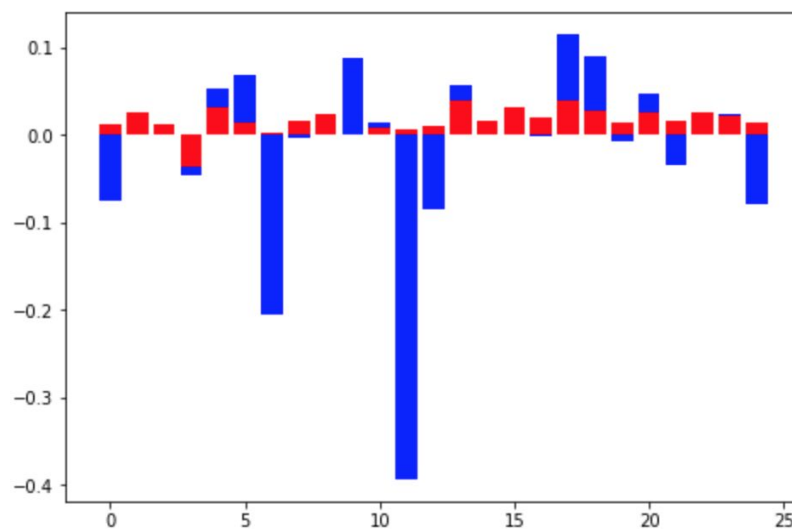


Here is an example of how most of the linear lines in our model looked like



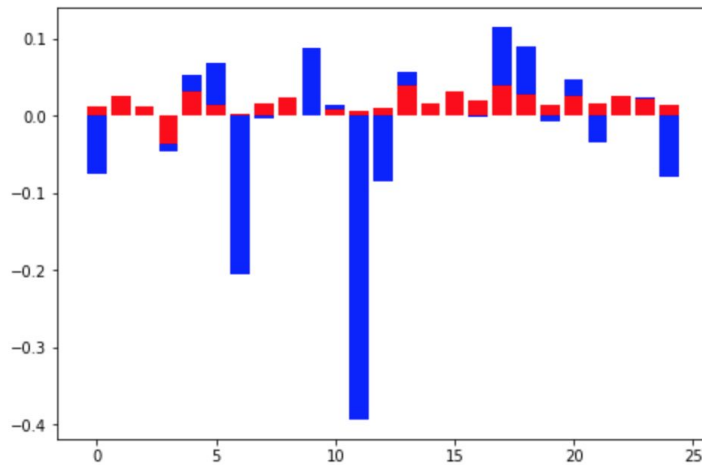


While our data did not all end up like this, this is a generalization of how most of them turned out. Unfortunately, due to our data set not having a true linear relationship, it did not generate the best lines for our prediction model. The following is 25 random predictions compared to its actual value.

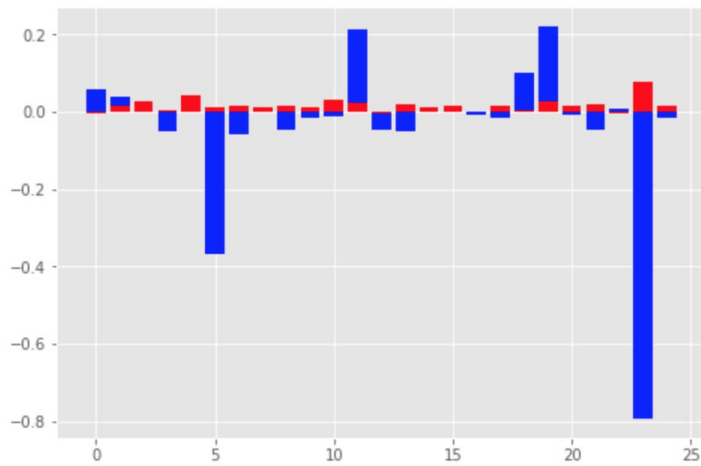


As you can see, this model is not very accurate. The blue is the actual value and the red is our predicted value.

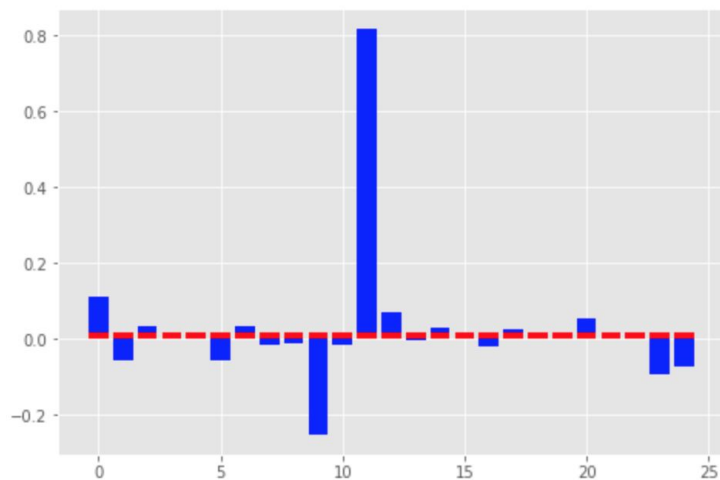
We attempted ridge regression which reduces model complexity. This resulted in a slight accuracy increase but it is still far from being accurate.



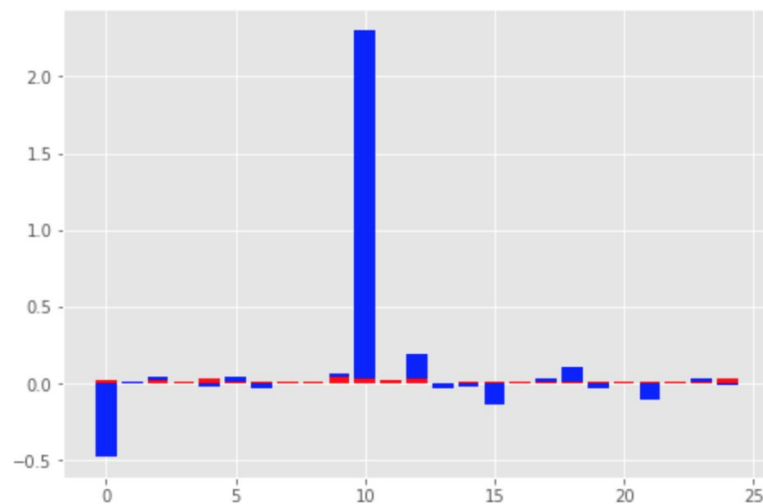
We then attempted to use cross validation to train the model. We were hoping that reducing some of the over fitting would increase our accuracy. However, it didn't see any significant increase in accuracy.



The following is from using lasso lars.



Using cross validation



Both yielded similar results in terms of accuracy.

2016 Data

The 2016 data contains much of the same results, however there are some things we should take note of. The first thing we see is the same discrepancy in the provided data, except this time we have more data to train with.

```
# What happens if we use the 2016 data?
df_properties = pd.read_csv('./properties_2016.csv', low_memory=False)
print(df_properties.shape)

df_train = pd.read_csv('./train_2016_v2.csv')
print(df_train.shape)
```

(2985217, 58)
(90275, 3)

After going through the same procedure as the 2017 data to merge and split the data, we ultimately reduced our data set to slightly over 3% of its original size. Again, due to us not knowing any other approach, this

was what we concluded as a correct step to move in. So, what improvements did we see? RMSE dropped by about 0.01 or so across the board. What could be the driving force behind such improvement?

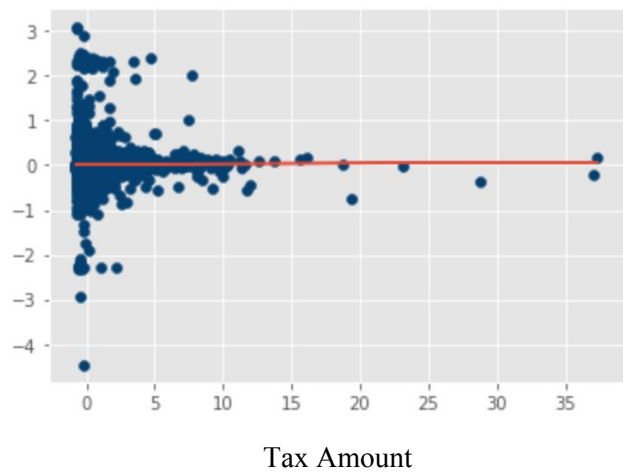
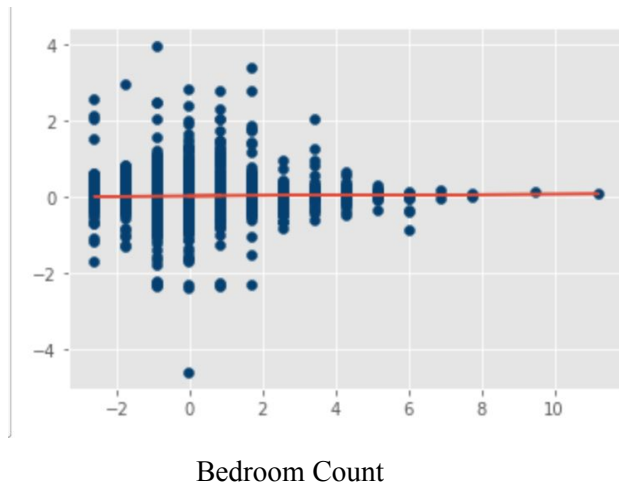
Some of you might have noticed that there is about a thirteen thousand row increase from the 2016 data to the 2017

data when it comes to testing. This led to an increase in accuracy simply because we had more data to test against, a surprise to be sure but a welcome one at that. Originally, we were only going to work on one data set going forward, but seeing these results led us to conclude that more data would produce a more accurate model. Our next step would be to combine both the 2016 and 2017 data to hopefully yield better results, or at least present a more diverse outcome.

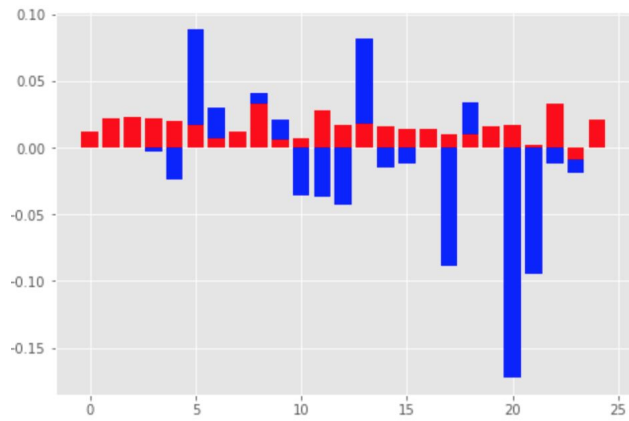
```
RSME Value Using Linear Regression: 0.16545118543063186
RSME Value Using Ridge Regression: 0.1654522451965398
RSME Value Using Ridge Regression Cross Validation: 0.16537956889615604
RSME Value Using Lasso Lars: 0.16574364014185616
RSME Value Using Lasso Lars Cross Validation: 0.1653608307445429
```

Visual Data Analysis

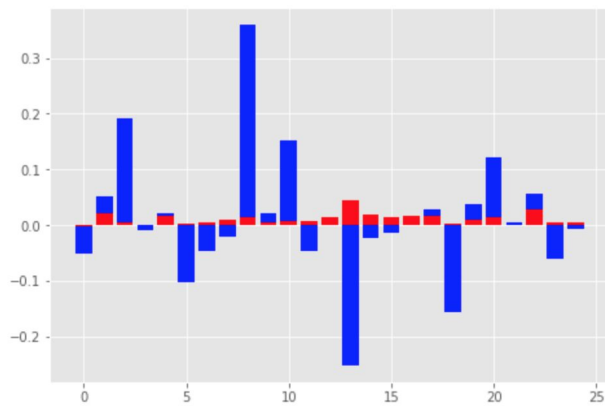
Similar to analyzing 2017 data we look at the slope again,



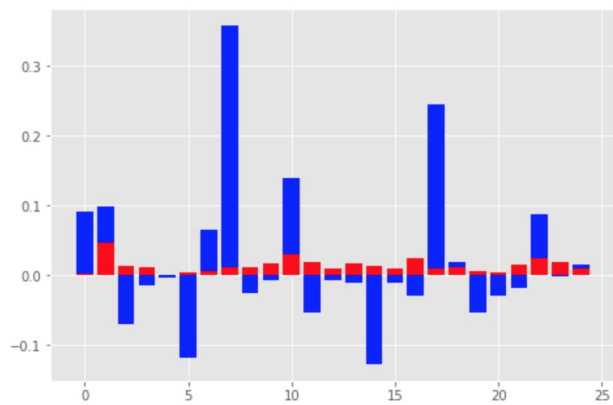
The following is 25 random predictions compared to its actual value using linear regression.



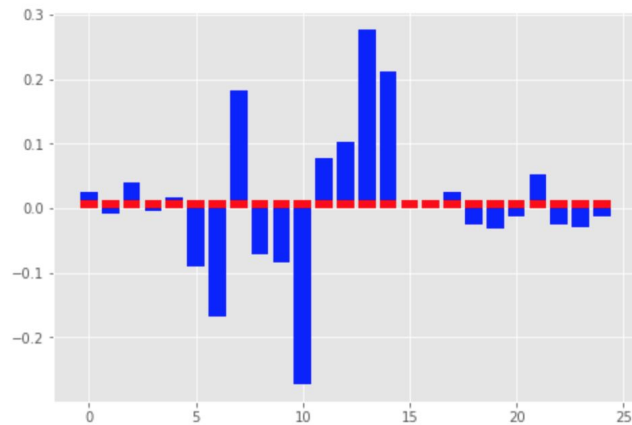
Using Ridge



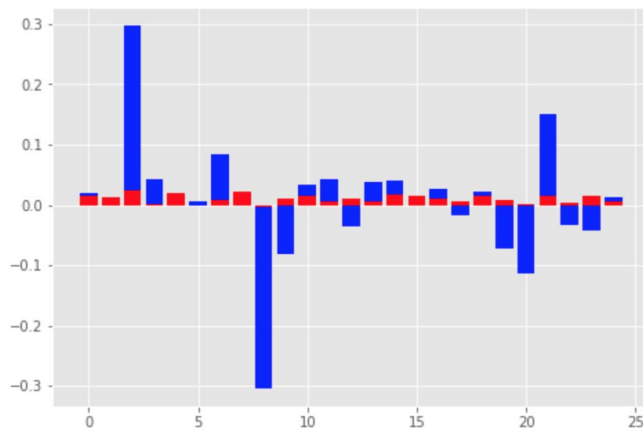
Using Ridge with cross validation



Using Lasso Lar



Using Lasso Lar with cross validation



As you can see, the 2016 dataset did not yield better results.

2016 and 2017 Data

Combining the data was a little bit of a mind-bending thing to do. We could reload the data that we had used previously or find a way to reuse it. We decided to reuse it. Going this route was an easy

```
# Now lets merge the data and see what happens
X_test_c = np.concatenate((X_test, X_test_2))
X_train_c = np.concatenate((X_train, X_train_2))
y_test_c = np.concatenate((y_test, y_test_2))
y_train_c = np.concatenate((y_train, y_train_2))
```

decision for us, we had our files separate, but when combining our findings, we saw we could simply

reuse our previous X and Y sets to create our new training X and Y's. Doing so cut down on memory usage.

We did run into some issues however, we didn't realize that the data had been converted into a NumPy array and was no longer a Pandas DataFrame as we had previously assumed. Some head scratching and google searching eventually led us to overcome this small hurdle.

```
my_linreg.fit(X_train_c, y_train_c)
my_ridge.fit(X_train_c, y_train_c)
my_ridge_cv.fit(X_train_c, y_train_c)
my_lassolars.fit(X_train_c, y_train_c)
my_lassolars_cv.fit(X_train_c, y_train_c)
```

We then trained the data like normal, using the same procedures that we have previously highlighted. Since we were training with more data, we expect to have more overall accurate results. Since our lone 2016 data led to more accurate results than the 2017, our initial guess had been an RMSE that's slightly lower than both separate sets.

So our results were not what we expected! The model ended up being roughly somewhere in the

```
RSME Value Using Linear Regression: 0.17056420411775808
RSME Value Using Ridge Regression: 0.1705647371245366
RSME Value Using Ridge Regression Cross Validation: 0.17053042956242923
RSME Value Using Lasso Lars: 0.17079688665256818
RSME Value Using Lasso Lars Cross Validation: 0.17052868584471745
```

middle when it comes to accuracy! So, what could have caused this to happen? Too much data being fed, was a possible conclusion. We also know that there could be outliers in the data that could skew the results. That then led us to our next scenario, finding out what happens when we remove outliers.

2016 and 2017 Data with Outliers Removed

Having combined our data previously and finding that our RMSE in general had landed somewhere in the middle of the 2016 and 2017 predictions led us to take a new approach. So we decided to remove outliers. One of the first things we had to do was reload the data. Our previous way of concatenating the data was admittedly a bit hackish and led to some issues when we tried finding outliers. With the data reloaded in a more useful manner now we can go on to continue our project!

Our method of finding outliers was based on what we had previously learned in our curriculum, statistics. The method we used was to find the z-score of the data. A

z-score is basically how many standard deviations from the mean a data point is, in our case, any row with a data point that had a z-score above 3, meaning it was 3 standard deviations from the mean, we decided to remove from our data set. In general ± 3 is seen as the roof and floor of normal data, so it is justifiable to remove data that contains an abnormal z-score. We were lucky to find a built-in method that could do

```
from scipy import stats
X_keys = X_z.keys()
y_keys = y_z.keys()

print(X_z.shape)
z = np.abs(stats.zscore(X_z))
print(z)
X_z = X_z[(z < 3).all(axis=1)]
print(X_z.shape)

(167888, 53)
[[ 1.34600081  0.30880747  0.04954193 ...  0.51085951  0.1667827
   0.20751942]
 [ 1.34625345  0.30880747  0.04954193 ...  0.06225182  0.1667827
   0.20751943]
 [ 1.3503535  0.30880747  0.04954193 ...  0.41591069  0.1667827
   0.2075201 ]
 ...
 [ 0.3266214  0.22244643  0.04954193 ...  0.12717605  0.1667827
   0.04376241]
 [ 0.43618012  0.30880747  0.04954193 ...  0.90380087  0.1667827
   13.35045316]
 [ 0.54416992  0.30880747  0.04954193 ...  0.07177166  0.1667827
   13.35045316]]
(127426, 53)

RMSE Value Using Linear Regression: 29460451433.668846
RMSE Value Using Ridge Regression: 0.1615783544013458
RMSE Value Using Ridge Regression Cross Validation: 0.16159120033299065
RMSE Value Using Lasso Lars: 0.16202534780364442
RMSE Value Using Lasso Lars Cross Validation: 0.16158560181522003
```

were better suited towards. We would however be open to finding out in what situations Linear Regression is ill-suited for, since up until now it has had predictions that fall in line with other methods.

```
X_z = X_z.append(X_z_temp)
X_z = X_z.fillna(0)
y_z = y_z.append(y_z_temp)
print(X_z.shape)
print(y_z.shape)

(167888, 58)
(167888, 3)
```

this for us. You can see that close to 40,000 entries have been removed!

From there we merge and split based on parcel ID's, to ensure that we have matching data sets for our X and y. And then split our data for actual training and testing, the methods we have previously highlighted in other sections. Once we had trained our models we then predicted, and finally found the RMSE for the various methods used.

Upon looking at the results we can see that they are a lot more accurate than any previous models we had created. But what's wrong with Linear Regression? We honestly could not come up with a solid answer Our educated guess was the smaller amount of data lead to a situation where the other algorithms

2016 and 2017 Data, No Outliers, Reduced Fields

One of the things that stood out to us in this data set was the amount of NaN that the fields contained. Printing the head of the raw data provided us with an almost equal amount of acceptable entries. To get rid of these NaN values we were filling them in with zeroes, thereby allowing the ML algorithms to run and yield results. But now we wanted to see what would happen if we got rid of fields that contained less than 30% valid entries, as we would find out, many fields were going to be gone.

NaN	NaN	9.0	2016.0	9.0	NaN	NaN	NaN	NaN
NaN	NaN	27516.0	2015.0	27516.0	NaN	NaN	NaN	NaN
NaN	660080.0	1434941.0	2016.0	774261.0	20800.37	NaN	NaN	NaN
NaN	580059.0	1174475.0	2016.0	594416.0	14557.57	NaN	NaN	NaN
NaN	196751.0	440101.0	2016.0	243350.0	5725.17	NaN	NaN	NaN

```
# Replacing NaN for all  
X = X.fillna(0)
```

```
X_z = X_z.drop('airconditioningtypeid', 1)  
X_z = X_z.drop('architecturalstyletypeid', 1)  
X_z = X_z.drop('basementsqft', 1)  
X_z = X_z.drop('buildingclasstypid', 1)  
X_z = X_z.drop('decktypeid', 1)  
X_z = X_z.drop('finishedfloorissquarefeet', 1)  
X_z = X_z.drop('finishedsquarefeet13', 1)  
X_z = X_z.drop('finishedsquarefeet15', 1)  
X_z = X_z.drop('finishedsquarefeet50', 1)  
X_z = X_z.drop('finishedsquarefeet6', 1)  
X_z = X_z.drop('fireplacecnt', 1)  
X_z = X_z.drop('poolcnt', 1)  
X_z = X_z.drop('poolsizeum', 1)  
X_z = X_z.drop('pooltypeid10', 1)  
X_z = X_z.drop('pooltypeid2', 1)  
X_z = X_z.drop('pooltypeid7', 1)  
X_z = X_z.drop('storytypeid', 1)  
X_z = X_z.drop('threequarterbathnbr', 1)  
X_z = X_z.drop('typeconstructiontypeid', 1)  
X_z = X_z.drop('yardbuildingsqft17', 1)  
X_z = X_z.drop('yardbuildingsqft26', 1)  
X_z = X_z.drop('numberofstories', 1)  
X_z = X_z.drop('taxdelinquencyyear', 1)
```

```
X_keys = X_z.keys()  
y_keys = y_z.keys()  
  
print(X_z.shape)  
z = np.abs(stats.zscore(X_z))  
print(z)  
X_z = X_z[(z < 3).all(axis=1)]  
print(X_z.shape)  
  
(167888, 30)  
[[1.34600081e+00 2.70878297e+00 8.33929950e-01 ... 1.84251290e-01  
 5.10859510e-01 2.07519425e-01]  
 [1.34625345e+00 2.87439603e-01 3.57641279e-02 ... 3.70166726e-02  
 6.22518246e-02 2.07519430e-01]  
 [1.35035350e+00 7.86810032e-01 3.57641279e-02 ... 4.38997043e-01  
 4.15910694e-01 2.07520101e-01]  
 ...  
 [3.26621404e-01 2.87439603e-01 9.05458205e-01 ... 4.89928423e-03  
 1.27176055e-01 4.37624114e-02]  
 [4.36180116e-01 2.11930825e-01 3.57641279e-02 ... 1.20625388e+00  
 9.03800872e-01 1.33504532e+01]  
 [5.44169923e-01 2.11930825e-01 3.57641279e-02 ... 1.22479642e-01  
 7.17716609e-02 1.33504532e+01]]  
  
RSME Value Using Linear Regression: 0.15722339192212734  
RSME Value Using Ridge Regression: 0.15722185741150327  
RSME Value Using Ridge Regression Cross Validation: 0.1572053480606298  
RSME Value Using Lasso Lars: 0.15752374665604352  
RSME Value Using Lasso Lars Cross Validation: 0.15719744554080614
```

You can see we dropped 23 columns! Our next step was to re-calculate the z-scores for the columns. Since we took out many columns, we are now calculating on what could be perceived as “new” data. Of course we still had to fill any remaining NaN entries with 0.

You can see that our new dataset has been reduced to about 149k entries from 167k. This is about an 18k reduction in rows, close to 22k more entries than our previous method! Why are there more entries this time around though? The answer is quite simple, since there are fewer columns, those columns are no longer taken into consideration when calculating z-score, leading to less outliers. Since there are fewer outlier properties, then we have fewer entries that we must get rid of.

Once proceeding like in previous steps we got our results: A more accurate set of predictions than any previous method! About a .004 drop in RSME from our previous testing. Removing our extra columns however made us a bit worried in how well our models were trained however, so we decided what would happen if we

just trained without removing z-score outliers, and only on a model with columns removed.

2016 and 2017 Data with Reduced Fields

So now at the last scenario we are looking at has to do with the combined data sets, but with reduced fields. As in our previous test, we removed columns that had less than a 30% valid amount of entries according to the Kaggle website. This again led to us to the same procedures as before: merging

the data, splitting the data, and then fitting the data into the models.

```
X_z = X_z.drop('airconditioningtypeid', 1)
X_z = X_z.drop('architecturalstyletypeid', 1)
X_z = X_z.drop('basementsqft', 1)
X_z = X_z.drop('buildingclasstypeid', 1)
X_z = X_z.drop('decktypeid', 1)
X_z = X_z.drop('finishedfloorlsquarefeet', 1)
X_z = X_z.drop('finishedsquarefeet13', 1)
X_z = X_z.drop('finishedsquarefeet15', 1)
X_z = X_z.drop('finishedsquarefeet50', 1)
X_z = X_z.drop('finishedsquarefeet6', 1)
X_z = X_z.drop('fireplacecnt', 1)
X_z = X_z.drop('poolcnt', 1)
X_z = X_z.drop('poolsizeum', 1)
X_z = X_z.drop('pooltypeid10', 1)
X_z = X_z.drop('pooltypeid2', 1)
X_z = X_z.drop('pooltypeid7', 1)
X_z = X_z.drop('storytypeid', 1)
X_z = X_z.drop('threequarterbathnbr', 1)
X_z = X_z.drop('typeconstructiontypeid', 1)
X_z = X_z.drop('yardbuildingsqft17', 1)
X_z = X_z.drop('yardbuildingsqft26', 1)
X_z = X_z.drop('numberofstories', 1)
X_z = X_z.drop('taxdelinquencyyear', 1)
```

For fitting the data we did not run into any issues since it was straight forward. This time however we were running the algorithm on about 167k entries, the same amount as the first 2016 and 2017 combined raw scenario. This increases our confidence since we have the same amount of entries.

Once merging and splitting the data in the same way that we have done previously, we can then fit the data into the models to get our predictions. Once again we used the same five methods we have been using previously, Linear Regression, Ridge Regression, Ridge Regression with Cross Validation, Lasso Lars, and Lasso Lars with Cross Validation, each method hopefully providing us with more accurate results.

```
RSME Value Using Linear Regression: 0.16516719500418073
RSME Value Using Ridge Regression: 0.16516484596171407
RSME Value Using Ridge Regression Cross Validation: 0.16517446983538486
RSME Value Using Lasso Lars: 0.1654748556924704
RSME Value Using Lasso Lars Cross Validation: 0.16517835468886924
```

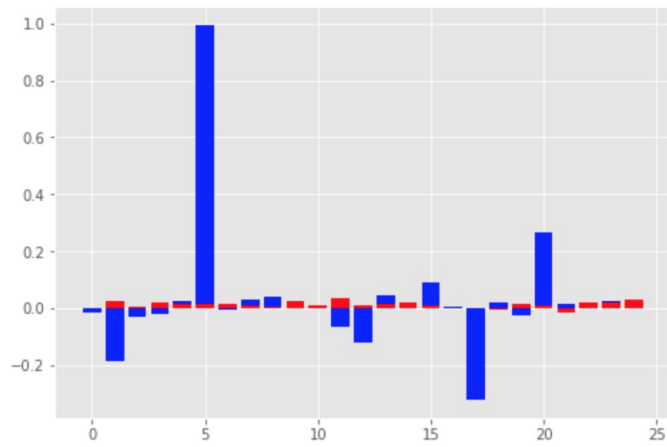
So once we find our MSE, calculate our RMSE, we can finally see what happens. Our results are more accurate than just doing the raw 2016 and 2017

data set but are about as accurate as our 2016 only trained models.

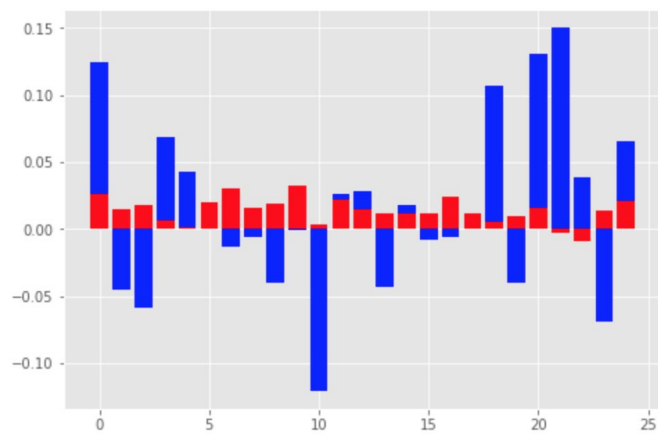
Visual Data Analysis for Reduced Field and Removed Outliers

After combining the dataset, reducing the number of fields, and removing the outliers. We were hoping that this time our accuracy would go up. Here are 25 random values testing data versus our prediction.

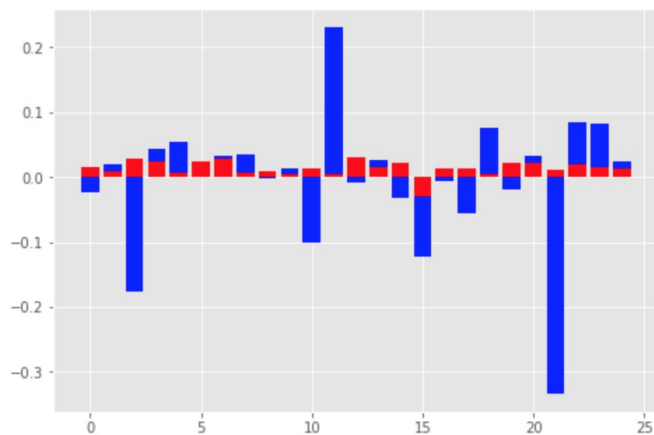
Using Linear Regression



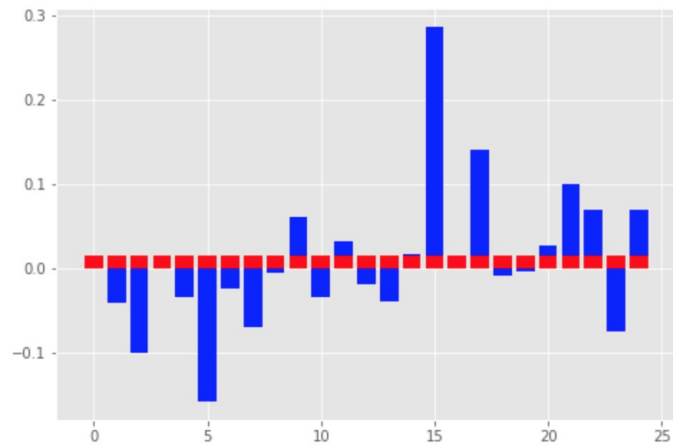
Using Ridge Regression



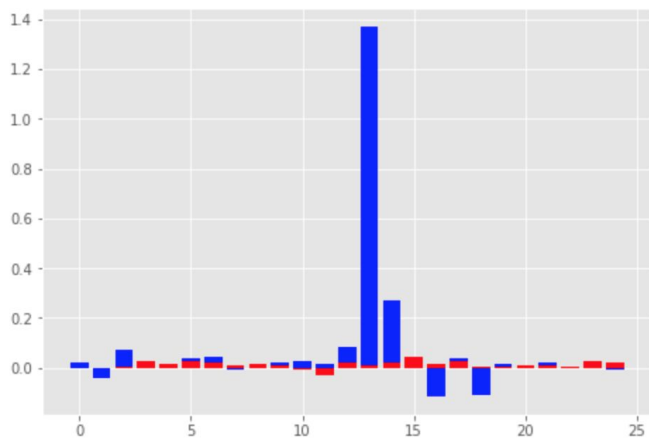
Using Ridge Regression with cross validation



Using Lasso Lar



Using Lasso Lar with cross validation



Unfortunately, our prediction still ended with really low accuracy numbers. However, we were able to see a slight increase in accuracy after eliminating . The increase in accuracy could definitely be debatable.

Conclusion

From our models we can conclude on one thing for sure: removing the extra columns might have trained our data incorrectly. In general, we all agree that removing the outliers based on the z-score was a step in the right direction. However, removing columns based on how much valid data they had was a situation that we could debate on. We all agreed that combining the data yielded results that we were more confident in. More data means that our results had a wider range of information to be able to form a conclusion on. The results that we are the most confident in is the set of combined 2016 and 2017 data with outliers removed.

If we were to repeat the project one thing we would for sure look into would be methods of using all the entries within the CSV files. We greatly reduced the amount of data that our models relied, simply because we could not train on all of it. One way we can think of is if Zillow provided data for only the properties that were sold through its site. We can safely assume that not all properties were sold because of Zillow, and that not all the properties given were properties that were for sale. Zillow is a website that also lists properties that are available for lease or rent. Properties listed on Zillow are also listed on other places, the same way a seller might list the same items on eBay and Craigslist, it's the same item, just listed in different places.

Responsibilities

Names	Responsibilities	
Zachary Portillo	Introduction and Data analysis	
Pasindu Siriwardena	Algorithms + Analysis	
Chan In Kou	Project Algorithm Discussion + Presentation	
Ivan Yu	Charts and Visualizations	
Carlos Larios-Solis	Algorithms + Analysis	