Project 1

The first portion of this project was exploratory data analysis, and data cleanup. This was, by far, the vast majority of the time spent. There was a lot of missing data requiring imputation. I attempted several different imputation techniques based on what the underlying data looked like, as well as some assumptions I made about the data, it’s meaning and how it was collected. First, we can work through the outlined questions.

1. EDA
2. **Determine the number of missing values on each feature. Determine the parcelid that has the most number of missing features.**

The parcelid with the most missing features was found to be 17188934. This was found by summing all of the NaNs by parcelid, and taking the maximum

1. **Determine the number of cities in each county.**

This is easily done with a groupby + distinct count:

A screenshot of a cell phone

Description automatically generated

1. **Determine which county has the highest logerror.**

There are two ways to look at this. We can look for which county has the highest logerror on average, or the highest maximum logerror. I’ll report both (using median as the average). Interestingly, region 3101 has the highest error, but region 2061 tends to be more erroneous overall.

A screenshot of a cell phone

Description automatically generated

1. **There are total of 17 columns containing column names with ‘id’. Referring to the data dictionary, which id’s can be eliminated? Justify your reasons.**

Here, I plot the amount of missing data by id, by percentage (higher is worse) for each of the columns.

buildingclasstypeid 99.982276

storytypeid 99.952368

architecturalstyletypeid 99.710883

typeconstructiontypeid 99.668790

decktypeid 99.271116

pooltypeid10 98.713930

pooltypeid2 98.666297

pooltypeid7 81.504292

airconditioningtypeid 68.118527

regionidneighborhood 60.108557

heatingorsystemtypeid 37.878704

buildingqualitytypeid 36.456383

regionidzip 0.038770

propertylandusetypeid 0.000000

regionidcity 0.000000

regionidcounty 0.000000

parcelid 0.000000

Going through these one by one, I can talk about the id columns I will keep, and those I will discard.

1) **buildingclasstypeid** will be dropped. There are not enough values.

2) **storytypeid** will be dropped. There are not enough values.

3) **architecturalstyletypeid** will be dropped. There are not enough values.

4) **typecontructiontypeid** will be dropped. There are not enough values.

5) **decktypeid** will be dropped. There are not enough values.

6-8) **pooltype** id cols are mostly useless, but perhaps we can scrape something from these columns along with poolcnt and poolsizesum. I will talk about this later.

9) **airconditioningtypeid** has enough values we can work with it. It will require a bit of imputation, and one-hot-enncoding.

10) **regionidneighborhood** I think will have to go. There are ~500 unique values, and they are not an 'ordered' column. If we don't mind 500 features from this column alone, we can try keeping it, but it would blow up our feature space.

11 **buildingqualitytypeid** we can attempt to impute this, this is actually a 'continuous' variable.

12) **heatingorsystemtypeid** will definitely be kept. Heat has significant value to the price of a house (from prior knowledge) and is not missing many values. This can also be one-hot-encoded.

13) **regionidzip** we can get rid of, we can just use city/county instead.

14) **propertylandusetypeid** I will try to keep, as it can greatly affect home value. This will need to be one-hot-encoded.

15) **regionidcity** we can keep, it could contain good information

16) **regionidcounty** we can keep, it could contain good information

17) **parcelid** is just our index, we will not use it as a feature

1. **Determine the correlations between the size and the room numbers. Are there any ways you can fill in NaN using the correlations? After you fill-in, make histrograms and observe if such method is meaningful. Explain how the new correlation changed after you fill-in NaN from the correlation. Did it improve?**

By size, I think we're referring to finishedsquarefeet6, finishedsquarefeet12, and calculatedfinishedsquarefeet; our total area, finished living area, and calculated finished area, respectively.

A screenshot of a cell phone

Description automatically generated

Interestingly, there is almost zero correlation between roomcnt and finishedsquarefeet12, while there is a slight correlation between roomcnt and finishedsquarefeet6. We can impute the finishedsquarefeet columns as a function of the two columns each here is correlated with. I'll use a simple linear regression for each. Before we impute, we can take a look at the distributions of each. Calculated square footage is redundant with finishedsqft12, so we will drop one of them. Here are our prior distributions:

A screenshot of a social media post

Description automatically generated

Here are our distributions prior to imputation:

A screenshot of a cell phone

Description automatically generated

Silly me, I didn't realize that finishedsquarefeet6 was 99% missing. Notice how the distribution for this histogram changed so wildly. I will drop this column in its entirety. The finishedsquarefeet12 column appears to have been imputed well, the histogram did not change much.

1. **Determine if the roomcnt is the sum of all room features, bedroomcnt and bathroomcnt. etc.**

First, I find all the columns with ‘room’ in the name.  
A screenshot of a cell phone

Description automatically generatedIt looks like there are only three? There are a couple of other columns that are bathrooms, but it looks like those are subsets of bathroomcnt. I calculated a ‘roof difference’ array.  


If room count was a sum of these two columns, this array should be close to zero most of the time.

A screenshot of a cell phone

Description automatically generated

Interestingly, there are almost always more bedrooms+bathrooms than there are rooms in 'roomcnt'. I wonder what percentage of houses have strictly more bedroomcnt and bathroomcnt than roomcnt. In fact, checking the quantiles, the percent of parcels with bedroom+bathroom > roomcnt: 77.58%. Clearly, there are a lot of rooms unaccounted for here.

1. **After you investigate more, delete features and rows that do not carry enough information and report the final dimension of the dataframe.**

I spent a lot of time imputing different columns, and will briefly talk about each of them here.

1. Threequarterbathnbr missing values are imputed with as a function of the other bathroom count columns (bathroomcnt and fullbathroomcnt)
2. Pool count is just a 1/0, but all the zeros are nans. We can fill those with zero. We will impute pool size with the median pool size of the non-zero sizes.
3. Additionally, I impute the spa\_hottub column using the other pooltypeid columns.
4. Airconditioningid is imputed with ‘5’, the id for no airconditioning (the most common value). This column is then one-hot-encoded.
5. Heatingorsystemid is imputed using weighted sampling of the existing distribution. Heating systems I think was too important to impute with none. After filling missing values, I one-hot-encode this as well.
6. I created a new column from airconditioning and heating, whether or not a house has central air (often a desirable feature in a house)
7. Buildingqualityid appears to be a normally distributed continuous variable (discrete integers, however). For this, I imputed with the mean. This will not change the distribution.
8. Yearbuilt is also imputed with the mean. There were no strong correlations between this column and other features.
9. Unitcnt I impute with 1. This is the mean, median, and mode. Additionally, I think most records missing this feature are more likely single houses anyway (units are much more likely to report unitcount).
10. Numberofstores I imputed using a simple linear model, using the most highly correlated features.
11. propoertylandusetypeid is one-hot-encoded.
12. And so on, I imputed many other columns as well. See the code for reference.

The columns I deleted are as follows. Columns were deleted due to lack of data, difficult to one-hot-encode/non-ordered, redundant, as well as columns that were one-hot-encoded are no longer needed.

finishedsquarefeet50, finishedsquarefeet15, fireplaceflag, pooltypeid2, heatingorsystemtypeid, censustractandblock, propertyzoningdesc, airconditioningtypeid, regionidneighborhood, hashottuborspa, pooltypeid7, buildingclasstypeid, taxdelinquencyyear, pooltypeid10, transactiondate, finishedfloor1squarefeet, finishedsquarefeet12, calculatedbathnbr, propertycountylandusecode, decktypeid, taxdelinquencyflag, finishedsquarefeet13, basementsqft, fips, yardbuildingsqft17, regionidzip, architecturalstyletypeid, propertylandusetypeid, typeconstructiontypeid, fireplacecnt, storytypeid, yardbuildingsqft26

In order to filter out rows I am not interested in, I worked backwards. That is, after I determined what columns I was interested in keeping as features, I dropped any rows from my initial dataset that were missing 10 or more features. These rows are going to largely just be imputed values, and I do not want to model them.

**The final shape of my dataset is: (89136, 65)**

The dataset is wider, due to one-hot-encoded columns.

1. **5-fold Modeling**

I broke the data into five folds based on the transaction date, in order to try and keep from biasing the model

* 1. **One the 1st sample - Do a normal linear regression. Report MSE value.**

Using sklearn, I train a simple linear regression model, default paramters.

Simple Linear Regression (all features) MSE: 0.03362251697974833

* 1. **2nd sample - make the lasso penalty, l1 and predict logerror. Report MSE value.**

Once again, we use sklearn. For this model, I tried an alpha of 0.75. I also increased the number of iterations from the default of 1000 up to 5000, due to errors that the model was not converging.

Lasso (all features) MSE: 0.02513385975833594

* 1. **3rd sample - Use the result of 2nd validation and reduce the number of features. Report the features you included and excluded. Report MSE value. You can try with many 𝜆λ values you would like to. However, due to the time limits, let's not try more than 7 values. This applies to step d and e as well.**

To do this, I grab the coefficients from the previous model, and the features associated. Here are the 11 most important features to the Lasso model:

calculatedfinishedsquarefeet, latitude, longitude, lotsizesquarefeet, rawcensustractandblock, regionidcity, structuretaxvaluedollarcnt, taxvaluedollarcnt, landtaxvaluedollarcnt, taxamount, date\_numeric

It looks like this model is content with only 11 features. Looking at them, this seems to make sense. We have four basic variables included in these 11 features: house size, house value (based on taxes), location information, and time.

Next, I retrained the model with a range of different alphas. The best result is reported below (see the code for the other results).  
Lasso (reduced features, alpha=0.2) MSE: 0.023261566224503823

* 1. **4th sample - Using the ridge regularization, predict logerror and report MSE value. You can try with many 𝜆λ values.**

For ridge regression, I tried five lambdas, one at each order of magnitude between 0.001 and 10. The best result was at lambda=10, shown below:

Ridge (all features, lambda=10) MSE: 0.022747009337094348

* 1. **5th sample - Predict logerror using elasticnet regularization. You can try with many 𝜆λ values. \ f. Report the best model from 5-folds.**

The same process for d was repeated in step e, using elasticnet. I again use five lambdas, one at each order of magnitude between 0.001 and 10. This time, a lambda of 0.001 reported the best MSE value.

ElasticNet (all features, lambda=0.001) MSE: 0.02143333770301487

1. **Among the tasks you made in step 3, choose the best model you have and predict logerror on the test set, that made transactions after 10/15/2016.**

As we saw, the elasticnet model was the best of the models we tested. We can now run this on our test set to see how well it does:  
ElasticNet Post 10/15/2016 MSE: 0.027363494743906182

1. **Using the logerror given values, make the dataset a binary problem - overestimated with “+1” and underestimated with “0”. Using a linear classifier, determine at least three properties that over estimates the price of house.**

Over-estimated houses are houses in which the Z-estimate is greater than the actual sale price (the house was worth less than the Zillow estimate). This corresponds to positive values of logerror. I will use a logistic regression model to make our estimates. Below are some metrics on how well the model is working. Additionally, I create a cross-tabulation.

Logistic Regression Overpriced Classifier Accuracy: 0.6041259500542888

Logistic Regression Overpriced Classifier Class 0 Accuracy: 0.13308189655172414

Logistic Regression Overpriced Classifier Class 1 Accuracy: 0.9221535103674063

Logistic Regression Overpriced Classifier Precision: 0.6117277992277992

Logistic Regression Overpriced Classifier Recall: 0.9221535103674063

A screenshot of a cell phone

Description automatically generated

This model is very good at predicting overpriced houses, but not good at predicting an underpriced house. To get three houses where we overestimated the price, we can grab the three highest predicted probabilities.

Below are ten parcels the model predicted to be over estimated, with their respective, actual logerrors.:

parcelid

11654953 -0.3711

11989106 0.0411

14317947 0.0742

12545569 -0.3397

14296390 0.0742

11006773 0.0411

12107635 0.0469

17241917 0.0344

10736758 0.0257

17053788 0.5271

Note, we get two of these estimations incorrect! The first record looks like quite the outlier though, with 15 bathrooms and 20,000 sq ft.

1. **Using the classifier of overestimation prediction identification you made in step 5, use the information of January of 2017 information and predict if the prices were overestimated or not. Compare your result with the actual logerror value.**

To do this, we need to load the data, and process it in the same way we processed the 2016 data (imputation, etc). Once trained, we can make our final predictions:

Logistic Regression Overpriced Classifier January 2017 MSE: 0.4001452432824982