Project 2 – Predicting Housing Prices using Multiple Regression

## Introduction

This project uses housing data from King County. The goal of this project is to develop best performing regression models using linear, ridge (L2 regularization), lasso (L1 regularization), and K-nearest neighbor regressions, with and without polynomial transformations. The dataset holds the following target features (independent variables): # of bedrooms and bathrooms, square feet of living and lot space, # of floors, waterfront, view, condition, grade, square feet above and basement, year built and renovated (if at all), latitude, longitude, square feet of living and lot (15), and zip code. The dataset has approximately 21k observations. The target (dependent variable) is housing price and we separated this into its own data-frame early in the data cleaning process. We also dropped ID and date from the dataset, as they do not have predictive value.

To begin the modeling process, we first developed a few functions that would make calculating the residual sum of squared error, the absolute mean percent error, and data cleaning steps simpler. The first function calculates the residual sum of squared errors by taking the difference between actual housing prices and model predictions, squaring the difference and adding the resulting squares. The absolute mean error function takes a ratio of the absolute value of the difference between actual housing prices and model predictions and actual housing prices. This is then multiplied by a 100 to get a percentage and the mean of all percentages are found as an overall average percent error value. Lastly, the age function subtracts the present year from a past year to return an age (for this dataset, it applies to the year the house was built).

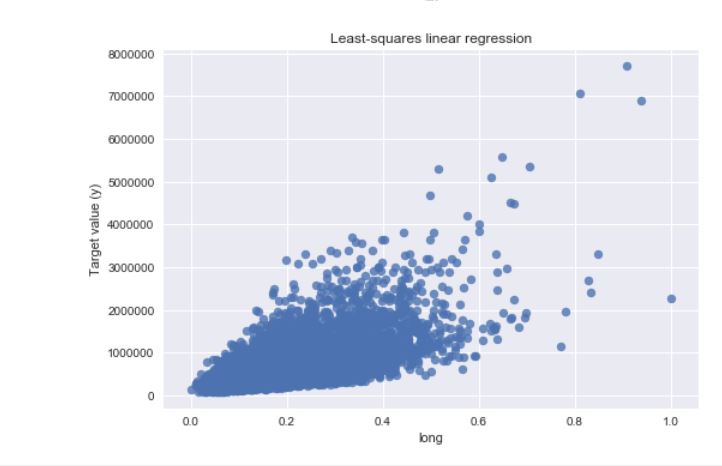
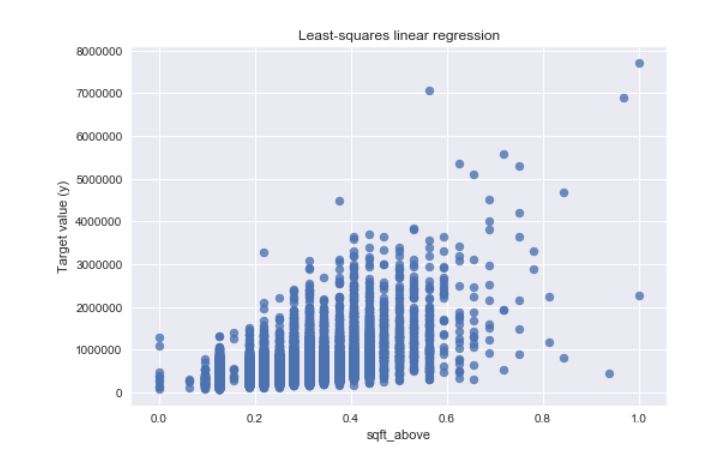
Next, we began cleaning the data. We left the target data fame alone, however we did have to perform some transformations to the target feature dataset. Zip code is a nominal feature, so we created dummy variables for each zip code in the column and merged the dummies back onto our model training dataset. This created 70 additional columns. Then, we converted year built into a continuous age value (using the age function) and mapped that value onto itself in the dataset and dummied year renovated into either a renovation occurred or it did not. Following that step, we then checked for null or missing values, and there was none. The final cleaned training dataset did have 87 features that could be used in the price models. Because of this number of columns, we also created an alternative dataset dropping the zip code dummy variables that could be used in polynomial transformations without causing a memory failure. Finally, we applied the Min/Max scaler to all values in both of our training datasets to ensure all values were to scale.

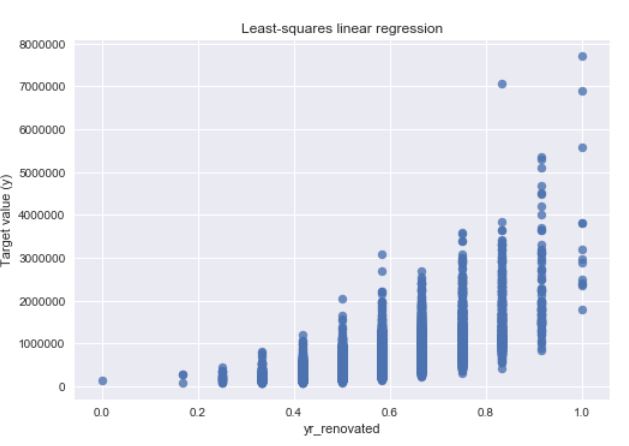
*Below are plots of paired target features. We can see some have linear relationships, and this will be addressed using feature selection methods.*

Once our training datasets were cleaned, we began by first visualizing the data using scatter plots. We plotted features against each other using pair plots, and here are a few:

We can see that some features do have slightly linear relationships with each other, and these overlapping features may be dropped later in the modeling process.

After exploring the independent variables, we then explored the relationships between the independent variables and the target. Here are some relationship illuminating plots of the independent variable against the dependent variable:





We can see that independent variables that have a linear relationship with the dependent variable, housing prices, will be most predictive in comprehensive multiple regression models. Here we see that longitude, loft size and whether a renovation happened are predictive of housing price, where higher longitudes, loft sizes and a renovation occurring lead to higher house prices. Once the data had been cleaned and explored, we were ready to build predictive linear regression models. Before each model build, we did split the data into training and test sets using the train-test split method in python’s sklearn library typically with an 80/20% split (unless using validation set runs).

## Part I

Interpretation for each variable:

**Assumption**: - Description of variables is not provided. Interpretation of variable is based to common definition.

|  |  |
| --- | --- |
| **Features** | **Parameter Estimate** |
| bedrooms | 2.62428021955e+17 |
| bathrooms | -842803.057331 |
| sqft\_living | 93014.6525863 |
| sqft\_lot | 6.07989635673e+18 |
| floors | 463308.576689 |
| waterfront | -143827.316411 |
| view | 680890.651623 |
| condition | 226132.039951 |
| grade | 127669.004473 |
| sqft\_above | 608443.476532 |
| sqft\_basement | -4.18480413384e+18 |
| yr\_built | -2.21170569354e+18 |
| yr\_renovated | -7680.0 |
| lat | 56448.0 |
| long | 125312.0 |
| sqft\_living15 | -172928 |
| sqft\_lot15 | 66048.0 |
| Zipcode | 2.62428021955e+17 |

**Parameter Estimate:**

**Bedrooms:** The coefficient is 2.624. Every unit increase in no. of bedroom, 2.624 unit increase in house price is expected, keeping all other variable constant.

**Bathrooms**: The coefficient is -842803.057331. Every unit increase in no. of bathrooms, 842803.057331 unit decrease in house price is expected, keeping all other variable constant.

**sqft\_living:** The coefficient is 93014.6525863. Every unit increase in square feet of living area, 93014.6525863 unit increase in house price is expected, keeping all other variable constant.

**sqft\_lot:** The coefficient is 6.07989635673e+18. Every unit increase in square feet of lot, 6.07989635673e+18 unit increase in house price is expected, keeping all other variable constant.

**Floors:** The coefficient is 463308.576689. Every unit increase in floor, 463308.576689 unit increase in house price is expected, keeping all other variable constant.

**Waterfront=1 vs Waterfront= 0:** The coefficient is -143827.316411. If the house has no waterfront than 143827.316411 unit decrease in house price is expected compare to the house has waterfront.

**View:** The coefficient is 680890.651623. Every unit increase in view, 680890.651623 unit increase in house price is expected, keeping all other variable constant.

**Condition:** The coefficient is 226132.039951. Every unit increase in condition, 226132.039951 unit increase in house price is expected, keeping all other variable constant.

**Grade:-** The coefficient is 127669.004473. Every unit increase in grade, 127669.004473 unit increase in house price is expected, keeping all other variable constant.

**sqft\_above:** The coefficient is 608443.476532. Every unit increase in sqft\_above, 608443.476532 unit increase in house price is expected, keeping all other variable constant.

**sqft\_basement:** The coefficient is -4.18480413384e+18. Every unit increase in sqft\_basement, 4.18480413384e+18 unit decrease in house price is expected, keeping all other variable constant.

**yr\_built(Age):** The coefficient is -2.21170569354e+18. Every unit increase in Age of house, 2.21170569354e+18 unit decrease in house price is expected, keeping all other variable constant.

**yr\_renovated =1 vs yr\_renovated =0 :-** The coefficient is -7680.0. If the house is not renovated than 143827.316411 unit decrease in house price is expected compare to the renovated house .

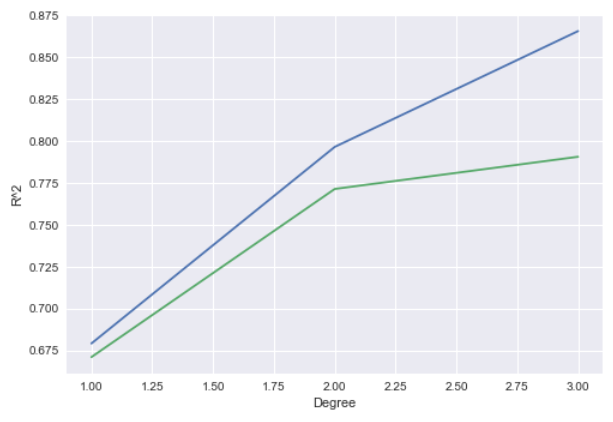
**lat**: The coefficient is 56448.0. Every unit increase in lat, 56448.0 unit increase in house price is expected, keeping all other variable constant.

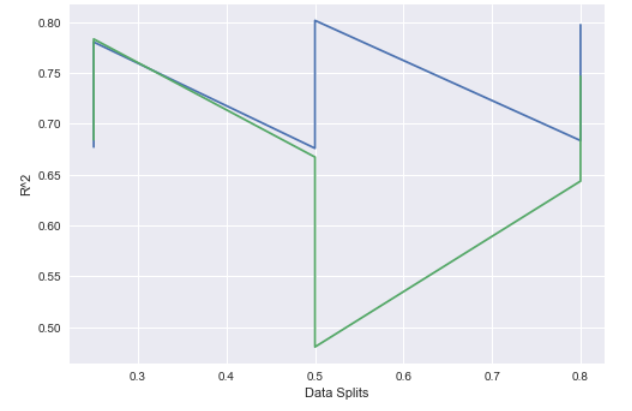
**Long:** The coefficient is 125312.0. Every unit increase in long, 125312.0 unit increase in house price is expected, keeping all other variable constant.

**sqft\_living15:** The coefficient is -172928. Every unit increase in sqft\_living15, 172928 unit decrease in house price is expected, keeping all other variable constant.

**Different Zipcode Vs Base Zipcode :** The coefficient is 2.62428021955e+17. Every change in zipcode, 2.62428021955e+17 unit increase in house price is expected, compared to base zip code keeping all other variable constant.

## Part II

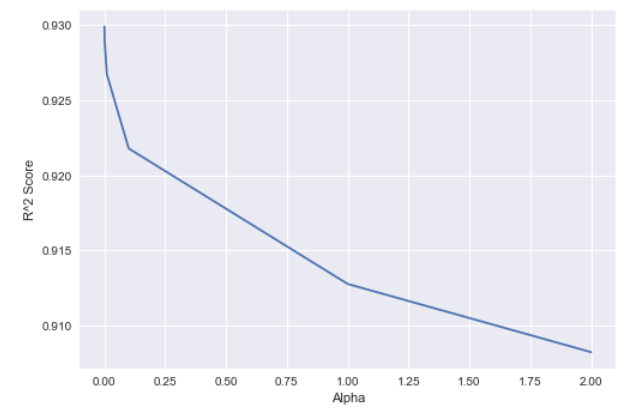
In order to improve the linear regression model, we then employed polynomial data transformations to see if this would improve the overall fit of the model and reduce prediction errors. To transform the dataset, we used sklearn’s polynomial feature transformation method. Because of the number of features in our dataset (88), our first runs to transform the dataset past the second degree failed due to memory overflow. To correct this we reduced the number of features by dropping zip code dummy variables. We transformed the dataset using up to three degrees and trained a linear regression model using the transformed data. Based on best model fit (highest R-squared values) we determined that the best polynomial transformation was a third degree transformation, with an R-squared values of .87 and .79 for the training and test sets, respectively, and shown to the left:

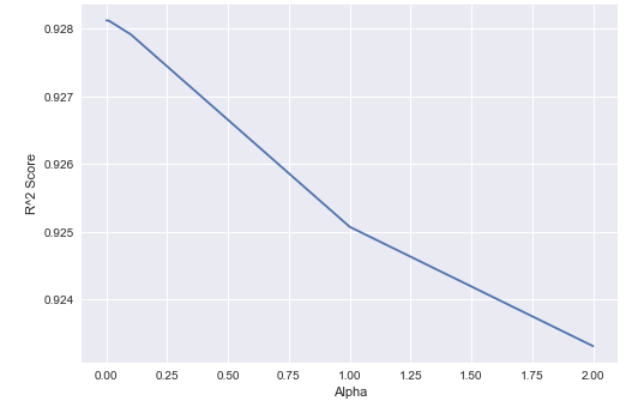
To ensure that we did select the best degree for the polynomial transformation, we did then run the linear regression model using various training/validation dataset splits from 25-80% on the training data while also looping through various polynomial transformation degrees from 1-3. We did find that the best model fit resulted from using the most training data (80% split) with a second degree dataset transformation, and this can be seen in the subsequent plot:

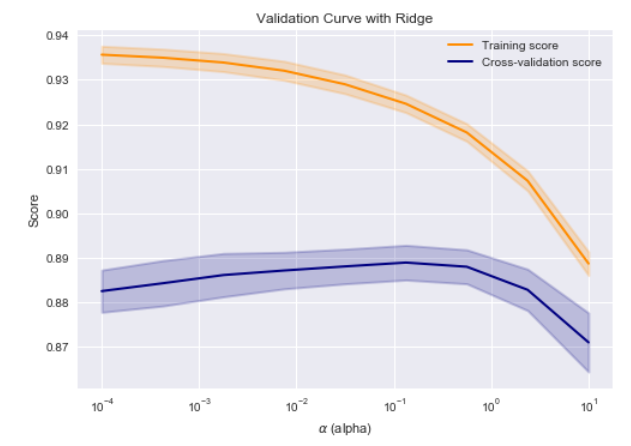
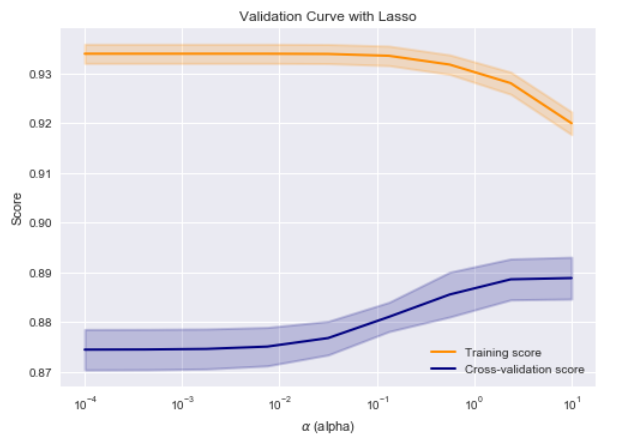
Due to the size of the full dataset and based on the validation model runs, we chose a polynomial transformation of degree 2 for models moving forward. This is because using a polynomial transformation with a higher degree does result in a better fit model, and a second degree transformation will not cause a memory failure when using all possible features, which also results in a better model fit.

*This point indicates a training/validation split of 80/20%, with a second degree polynomial transformation, and R-squared values of .79 & .74.*

## Part III

 For this part, we employed both ridge (L2 regularization) and lasso (L1 regularization) models using degree 2 polynomial transformation. In our first model runs, we used a standard 80/20% training and test set split on the data with several alpha values, (α = .0001,.001,.01,.1,1,2). For the ridge regression, the best regularization value for alpha was closest to zero (.0001) with a polynomial transf. degree of 2, producing an R-square value of .929, and this can be seen by the following R-square vs. Alpha plot:

 For the lasso regression, the best regularization value for alpha was close to .01 and a polynomial transf. degree of 2, and an R-square value of .928, which can be seen in the following R-square vs. Alpha plot:

After these preliminary model runs, we then used validation curves to get closer to the best alpha values for the ridge and lasso regularization models. Both ridge and lasso regressions were run using a training dataset with a polynomial transformation degree of 2. The validation curves revealed that the best alpha for the ridge regression model was approximately .13, and the best alpha for the lasso regression model was approximately 2.317, illustrated in the validation curve plots:

From the ridge regression validation curve, we can see that alpha values closest to .1 produce the best fit linear models, and from the lasso regression curve, we can see that alpha values closer to 1 produce the best fit models without overfitting (which we can see for lower alpha values on this curve).

For the final task, we predicted the housing prices using the lasso regression model with a dataset that has a polynomial transformation of 2 and an alpha value of 2.317. The model’s training model R-square value was .924 and testing R-square value was .899. The residual sum of squared errors value was 5.19E13. The average absolute percent error of the predictions was 12.9% and there were 871of non-zero features. The model zeroed out 3,044 features, because they did not significantly contribute to the accuracy of predicting housing prices. The significant features in the model were the following:

* # of Bedrooms
* # of Bathrooms
* Square feet of living
* Floors
* Waterfront (yes/no)
* View
* Condition
* Grade
* Square feet above
* Square feet basement
* Year built
* Year Renovated (yes/no)
* Latitude
* Longitude
* Square feet of living and lot – 15
* Various zip codes

## Part IV

For the final model, we used K-nearest neighbor regression to predict housing price. To train this model, we used all features of the training dataset (80/20% train-test split) and looped K values from 1-5 through 5 training/validation folds (training-validation split: 80/20% as well). Using the average R-square value from the validation folds, we were able to determine a best K value of 1 with an average R-square value of .99 for the training data across all training folds and an average R-square of .64 for the model fit on the validation data across all folds. All values of K we tried did result in overfitting, so this may not be the best machine learning model to use on this dataset.

To test the final KNN regression model, we then trained it on an 80/20% train-test split using K = 1, determined from the cross-validation model runs. The final model did have an R-square value of .696 on the test data, a residual sum of squared errors of 2.18E14, and an average absolute percent mean error of 19.34%.

## Conclusion

Based on all of the models using various types of data transformation, the best performing model was the L1 regularization or Lasso model. This is because it produced the highest R-square value on test data (89% variance explained), while avoiding over-fitting. It also created the simplest model, using data transformed with a second-degree polynomial, lasso forced insignificant parameters to zero, therefore dropping thousands of unnecessary input features. The best lasso model had an average prediction accuracy of 87.9% for housing prices, performing much better than the linear and KNN regression models and being simpler than the ridge regression model, because of the built-in feature selection.