Regression on Housing Prices

By: Ved Rajesh and Aditya Vikrant

Background:

As college students moving into our senior year, we are looking for off-campus housing. The average rent in Boston is $2887, which is 13% higher than a year ago. This is extremely high, especially for college students, so we would want to find cheaper housing with fewer amenities. This is why we are looking at housing prices for our project, and specifically, trying to figure out which houses would be cheaper based on their various 'features'. We want to build a model that is very accurate so that we can use that model to find apartments that fit our needs for the right price.

This is exactly why we chose to do the Kaggle competition "House Prices - Advanced Regression Techniques". The dataset we are given has over 30 different features which makes it very in-depth. Additionally, the dataset is also based on the Boston Housing dataset which is again very relevant to us, as we are college students in Boston currently looking for affordable housing.

Technical Statement:

As the problem we are currently trying to solve is based on Machine Learning, we do not have any way of giving a formal problem formulation for this project. However, we can give a very brief and informal overview of what this project will entail.

The problem we are trying to solve, as previously mentioned, is trying to predict the prices for various houses given to us in a dataset based on their various features. In order to do so, we need to train our data using the training dataset, which includes data points (houses) and their various features. Examples of these features are further explained in the next section.

After training our data, we will test our model's accuracy against the test dataset, which again contains the same things as the training dataset. The output of the model here will therefore be the predicted house prices for each test data point in the test dataset.

Regarding the choosing of the actual models themselves, the challenge here lies in figuring out what model would work best for this particular task. As a result, we are looking to leverage various techniques to do so, such as Lasso (L1) regression, Deep Neural Networks, and other models if required.

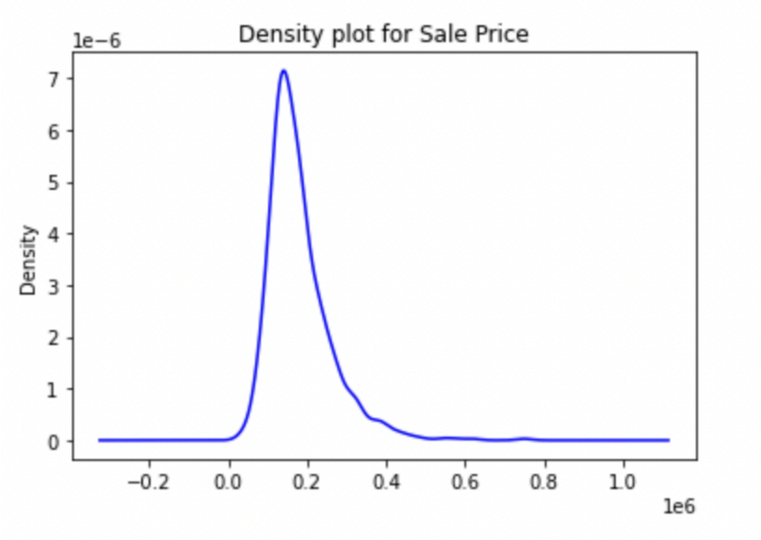
Dataset:

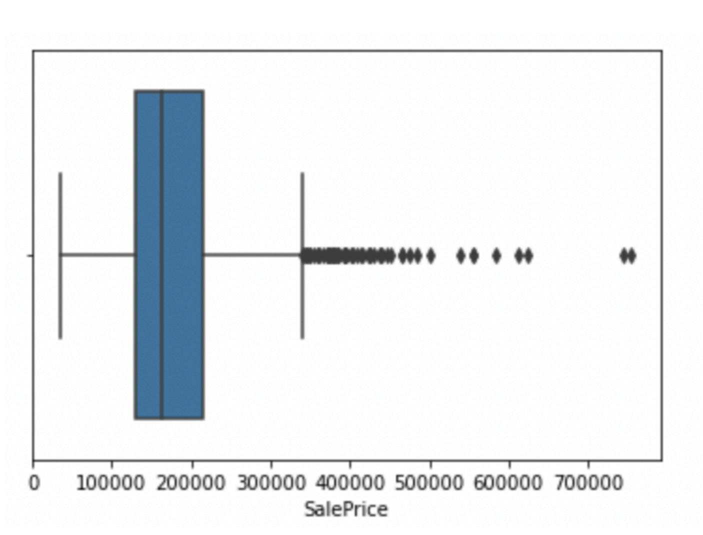
Since the problem we are trying to address involves finding final house prices for inputs, the training dataset (which is in the form of a CSV file) would naturally contain many features regarding each of these houses and their final prices as well, and this is shown through one of the files given to us, "data_description.txt".

To be more specific, the dataset contains 30+ features, with some examples being LotFrontage (which is the linear feet of street connected to the property), LotArea (lot size in square feet), Utilities (type of utilities available), and LandSlope (type of slope at the property). Most of these variables are categorical, with a couple of them being numeric, such as LotArea.

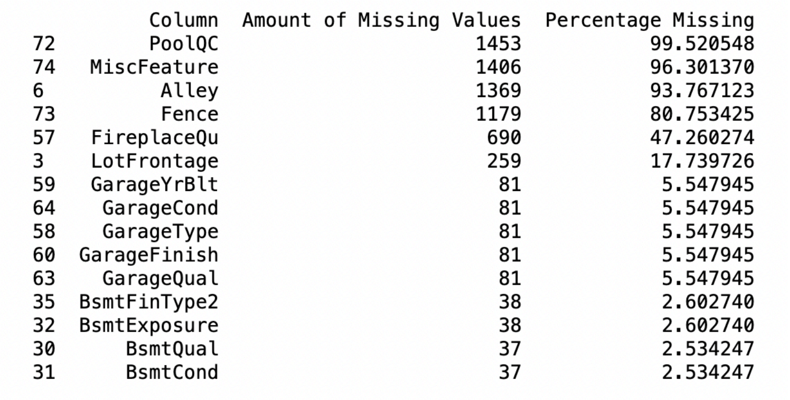
Cleaning the Dataset:

Before we start modeling our dataset, we have to clean it. So we first check if our main feature, the Sale Price has any anomalies that we need to consider. We did that by first creating a density plot, and then a box plot of all the data points.





As you can see there are a few anomalies, especially the two main points after 700000 in the box plot. We can account for this by scaling the data. But before we do that, we need to further clean the data by changing the categorical data to numerical data. The thing is, not all the features given are needed, so we can remove a lot of them by checking how many NaN results exist for the numerical features. If the result is over 70% we remove those features entirely, as they're not useful. Below is the table we created with the percentage missing. We removed the top 4 columns as most of their data was missing.



```
          Column  Amount of Missing Values  Percentage Missing
72        PoolQC                      1453           99.520548
74   MiscFeature                      1406           96.301370
6          Alley                      1369           93.767123
73         Fence                      1179           80.753425
57   FireplaceQu                       690           47.260274
3    LotFrontage                       259           17.739726
59   GarageYrBlt                        81            5.547945
64    GarageCond                        81            5.547945
58    GarageType                        81            5.547945
60  GarageFinish                        81            5.547945
63    GarageQual                        81            5.547945
35  BsmtFinType2                        38            2.602740
32  BsmtExposure                        38            2.602740
30      BsmtQual                        37            2.534247
31      BsmtCond                        37            2.534247
```

After doing that we changed the categorical data. We first looked at categorical data which only had two values, such as CentralAir, which is Yes and No, and we just replaced Yes with 1 and No with 0. With the other categorical features, we just replaced them with numbers starting with 1 and ending with however many values there are. An example of this is KitchenQual, where the values are Ex, Gd, TA, Fa, and Po. These values will be replaced with 1, 2, 3, 4 and 5. For all the other Nan values, we replaced them with 0, as that is what they represent, a lack of that specific feature. After that, we finally scaled the data.

Another cleaning method that we did was to log the data. This is done to further scale the data. The reason we did that was because when we began modeling, our root mean square would be over 100,000 which shouldn't be the case. After logging our dataset our overall root mean square was less than 0.2, which shows that the models are somewhat accurate.

Models and Analysis:

After cleaning our dataset, we are now ready to move onto actually deciding on and creating the required models. In order to decide which models would be the best to use, we first looked at the structure of the problem itself. Even though we had a lot of features at our disposal, we would start with the simplest model, which is a linear model. After doing so, we would also want to see whether the data had non-linear relationships in them, which we would try to capture using polynomial models. We therefore decided on using polynomial models with degrees 2 and 3 in order to do so.

However, with so many available features to use, it was apparent that not all of them would be important for predicting the final price of every house, which made us come to the conclusion that we needed to find a way to systematically eliminate certain features from our model. In order to fix this problem, we researched the various available models that would help eliminate features.

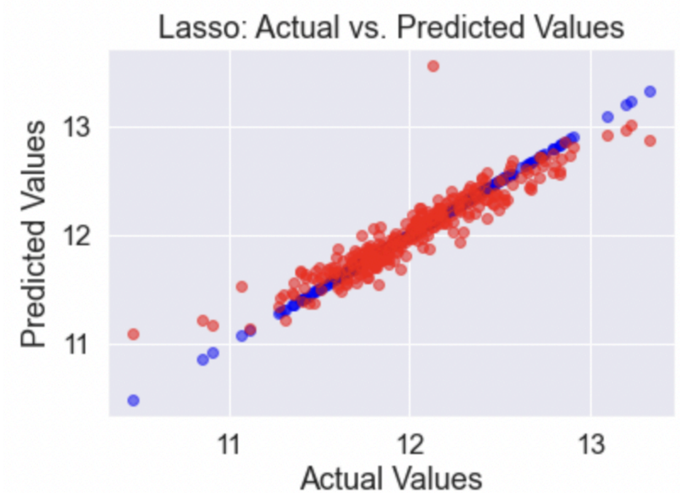
After hours of research, we landed on the Lasso (L1) regression model. This is very similar to the Ridge (L2) regression model we learned in class, where it includes a hyperparameter in order to determine the strength of the penalty for each parameter in the model. In Lasso, however, a higher strength would make the parameters for more features *equal* to zero (as compared to *close* to zero in Ridge), and a lower strength conversely makes fewer features equal to zero. This therefore ‘eliminates’ some features from the model, therefore effectively performing feature selection. Another benefit of this model (which is also a regularization technique) that we realized upon doing our research is that it will prevent overfitting, which will lead to better results on unseen data.

We came across another interesting algorithm during our research called Decision Trees. Since we had not learned about this through the class material, we decided to delve deeper into the topic in order to try to understand and use it for our task. Decision trees essentially provide a tree-like structure of the given data, and this contains various nodes, all of which contain data points split up based on various conditions. For example, if we start with 20 data points at the root node, these points will be split by a condition for a specific feature. If the condition is, for example, data points whose ‘LotArea’ is less than 1000 square feet, then those points which satisfy this condition will be put in the left node, and those that don’t will be put in the right. Now a problem arises here - how do we choose which feature and its corresponding condition to split the data by? This is chosen on the basis of the *purity* of the child nodes after the split. Purity is essentially a measure of variance for the data - if the total variance of both child node’s data is low, the purity is also low, and we would prefer this. However, if the variance is high, the purity would be high, and we would therefore prefer a feature and condition that would give us a lower purity (variance). This process therefore goes on until we satisfy certain pre-specified termination conditions, such as a minimum reduction in purity from one level of nodes to another or a maximum depth.

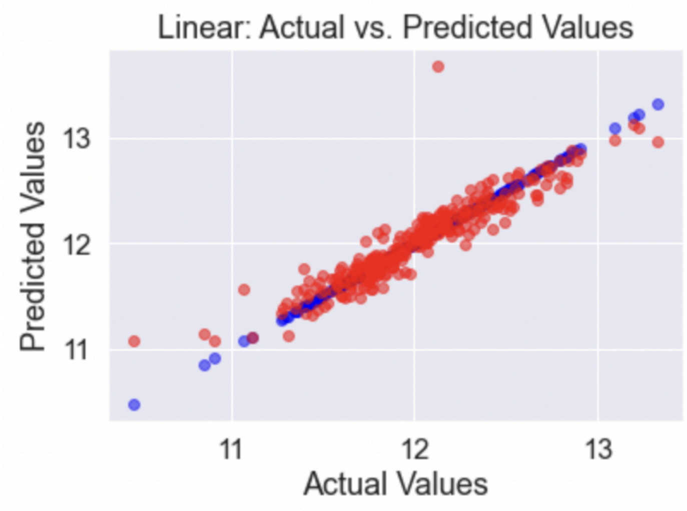
Upon forming the above tree, we can now use it for regression purposes (to predict the final house prices). We will pass a data point in, and this will reach a specific leaf node (also called terminal node) based on the predefined features and conditions in the tree. Upon reaching the leaf node, there will be certain data points associated with it, and to predict the passed in data point’s value, we simply take the mean of the data points associated with the leaf node. This technique, however, can lead to overfitting, especially if the tree is large. This can therefore lead to bad predictions on unseen data, which is why we found another method that solves this issue - Ensemble methods.

Ensemble methods essentially incorporate an element of diversity and “randomness” into the final model. This is done in two ways - bagging and boosting. In bagging, instead of training a huge decision tree on all the data, we split the data up into smaller datasets. This is done by choosing data points for each subset and replacing them in the original dataset, which is called bootstrapping. Each subset is then used to train individual decision trees, and after doing so, we can predict data. Here, when a data point is passed in, each individual tree has its own prediction (which is done as explained in the previous paragraph), and for regression purposes, we simply average the results of all of the individual trees to get the final result. Averaging over all these different trees will therefore remove any errors/biases from individual models, therefore making it less prone to overfitting and giving better predictions. In boosting, we first train a decision tree on the entire dataset. Then we get the erroneous data points and weigh them higher than the data points that were correctly predicted. We then do the same thing in each subsequent iteration, but from now, our aim while training our decision trees is to reduce the overall error but *with an emphasis on the data points that are weighted higher than the others (i.e. previously incorrectly predicted)*. This therefore continually improves upon the base model, thus improving prediction accuracies. In the end, we decided to use three ensemble methods, Random Forest (bagging), Gradient Boosting (boosting) and AdaBoost (boosting).

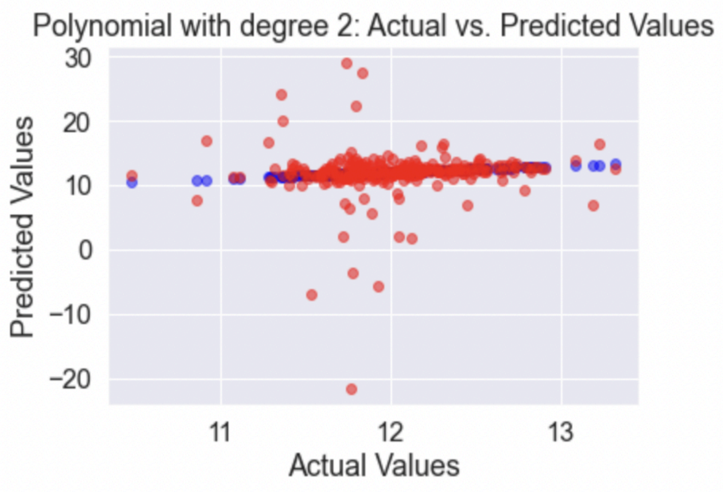
```
Root Mean Squared Error: 0.15121350929964564
```



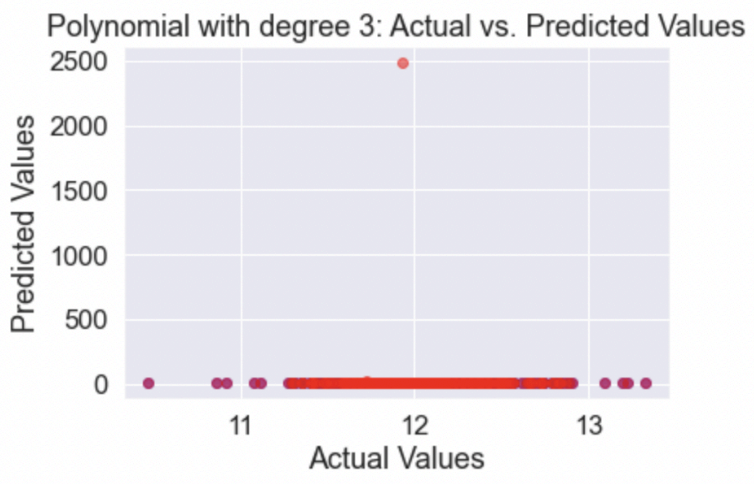
```
Root Mean Squared Error: 0.15209152938590406
```



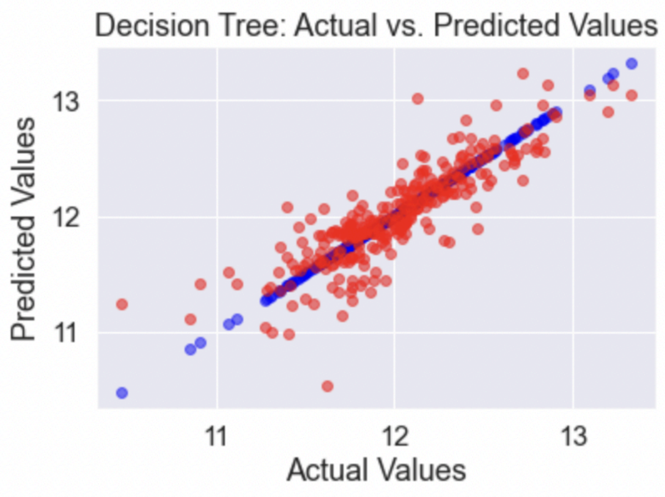
```
Root MSE with degree 2 is 3.587280232841871
```



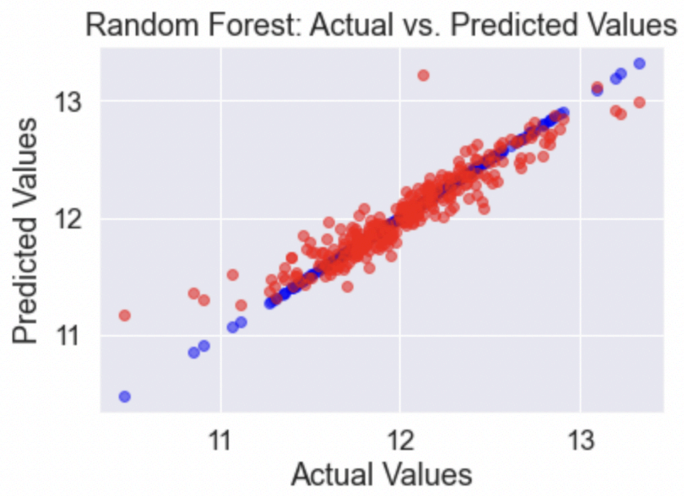
```
Root MSE with degree 3 is 144.89651682261226
```



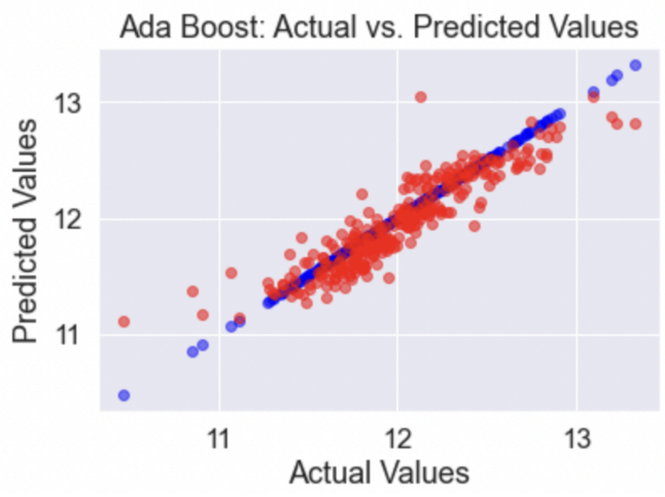
```
Root Mean Squared Error: 0.2186922505768193
```



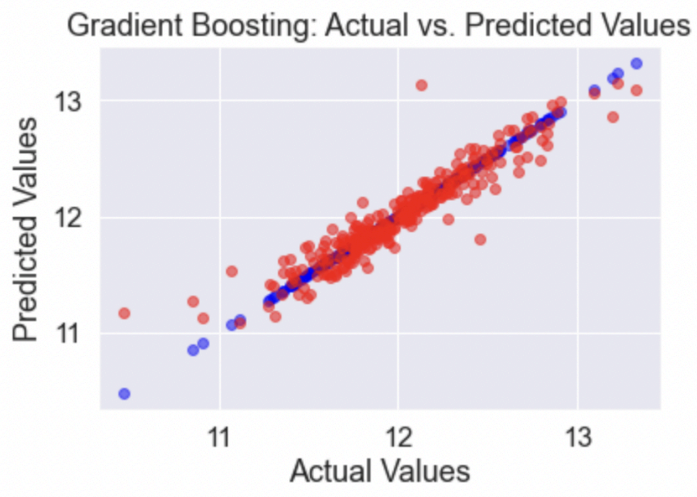
```
Root Mean Squared Error: 0.1468005327980576
```



```
Root Mean Squared Error: 0.17366903775000977
```



```
Root Mean Squared Error: 0.1388055106652956
```



In the graphs above, the red dots represent the predicted values that were created using that certain mode, while the blue dots represent the actual data. The two initial methods that we modeled first were the Lasso and Linear Regression. As seen, Lasso performed marginally better than Linear regression, with an RMSE (root mean squared error) of 0.151, as compared to 0.152 for Linear regression. This is something we assumed would happen, as Lasso performs feature selection and therefore 'regularizes' the model, all of which prevents overfitting to the training data. Therefore, there was clearly a benefit in modeling Lasso Regression.

The next 2 plots seen are the polynomial regression models with different degrees. You can see from the graphs alone that the prediction values do not fit well with the actual values. There are a lot of data points (red dots) that seem like anomalies as the difference between the values is that great. Also, the root mean square error shows that when the degree is 2, the root mean squared error is 3.587, and when you increase the degree to 3, the root mean squared error dramatically increases to 144.897. Since the values are so far away from 0, which represents a perfect fit, it is not worth trying to further tune certain values to get a better value.

After creating these three models, we moved on to (as discussed earlier) Decision Trees. We first fit a simple decision tree model, which we expected to perform the worst out of all the decision tree algorithms as there was a high chance of overfitting. Upon doing so, we realized that we were correct, as this model gave us an RMSE of 0.218, which was much higher than the other decision tree models. Therefore, in order to prevent overfitting, we then moved on to Ensemble methods, which were (as previously discussed) Random Forest, AdaBoost, and Gradient Boosting. Upon fitting the models to our data, we saw that as expected, all of them performed better than the regular Decision Tree model, with an average RMSE of approximately 0.1531 (rounded to four decimal places) compared to 0.218 for the regular decision tree. Individually, the best performing model was the Gradient Boosting Regressor, with a RMSE of 0.138. As previously mentioned, this was due to the models including 'diversity'/averaging of multiple trees upon different subsets of the data, which therefore led to less overfitting and better predictions on unseen/test data.

Firstly, we wanted to see why the Gradient Boosting Regressor performed the best out of all Decision tree models. After more research, we came to the conclusion that it must be for a couple of reasons. Firstly, Gradient Boosting is more robust to noisy data outliers than both AdaBoost and Random Forest, and as seen in the graph above, the original data does have quite a few outliers. Secondly, Gradient Boosting (and Boosting methods in general) reduce bias of models more than Random Forest, as they optimize models in a sequential manner by constantly improving upon previous errors, therefore making it possible to capture complex relationships and making better predictions. Random forests do not reduce the bias as much, as they build multiple trees *independently* and do not improve upon previous errors, thus possibly leading to higher bias and worse predictions overall.

The best performing model overall was also the Gradient Boosting Regressor, with a RMSE of 0.138. We therefore wanted to analyze why this model (a Decision Tree based model) performed better than all of our linear models as well. This was probably due to the fact that decision trees can capture relationships between features better, as they can split nodes based on *combinations* of different features, while in linear models, explicit feature engineering would have to be performed in order to do the same. Another reason would be decision trees' ability to handle outliers better, as data points that are significantly different from the others would generally be put into their own leaf nodes, therefore not affecting most predictions.

One thing that was quite interesting to see in our analysis was how close all the Ensemble method's RMSEs were to each other. This led us to conclude that these methods are all overall very effective, with small nuances that might affect their performances depending on the dataset we are using. Another thing that was interesting to look at was the plotted data, which compared the actual (shown in blue) and predicted (shown in red) values for each model. These visualizations allowed us to clearly see how all the models' performances differed from each other, as the better performing models had predictive values 'closer' to the actual data on the graph, while the worse performing ones had predictive values further from the actual data on the graph.

Reflection:

A lot of the difficulties we had were initially in the cleaning process. The way we initially cleaned our data was wrong as we did not implement any regularization methods after scaling the data. Also because of that, our feature selection was wrong, making us have to redo all the models. Since our features were wrong, our model's root mean squared was very high, as they were almost all above 100,000. However, since we logged the data, the root mean squared error reduced significantly, and most of the RMSE values were below 0.3. There were multiple times during the cleaning process where we had to undo all the cleaning done so far, so that was quite difficult to deal with.

If we had more time we would have looked more deeply into different models such as neural networks, as they would have given more plentiful and interesting results which we could have analyzed and changed further to make our project more interesting. Sadly, the library, Tensorflow, which is the library widely used for neural networks, did not work on any of our machines, so we were unable to see the benefits of using these neural networks.

Even though the cleaning that we did on the dataset given was good and effective, further cleaning could have been done to avoid small outliers. This could be done by inspecting each column intuitively. This would further decrease the root mean squared error and would increase the accuracy of the models that we used.