

**CE/CZ4041 Machine Learning Project Report**

**AY22/23 S1**

**Group 6**

**Team Members:**

| **No.** | **Name** | **Matriculation Number** |
| --- | --- | --- |
| 1 | ERICA ER MING CHEE | U1922739B |
| 2 | VEDULA KARTIKEYA | U1923891G |
| 3 | GAVIN NEO JUN HUI | U1921265L |
| 4 | CHOCKALINGAM KASI | U1920428E |

### Table of Content

[**Table of Content**](#_dc9u6gikr1i9) **2**

[**Zillow’s Home Value Prediction (Zestimate)**](#_lbg8utabpv4w) **3**

[**1. Problem statement**](#_b00et2898f0s) **3**

[**2. Challenge Statement**](#_tdn8yoest5gf) **3**

[**3. Workflow**](#_725dn99okbcx) **4**

[**4. Exploratory Data Analysis**](#_3cingk7vecel) **5**

[**5. Feature Engineering**](#_9ls8lda41q7f) **9**

[5.1 Data Preparation](#_phw4idlwceyq) 9

[5.2 Combining and processing of features](#_idb4grsq0z90) 9

[5.3 Removing features filled with missing data](#_un7783qwcm07) 10

[5.4 Replacing null values](#_usgy1gux4v61) 11

[5.5 One-hot encoding](#_krz8mtdl3xj3) 11

[5.6 Recursive Feature Elimination](#_smihwod4g5g) 12

[5.7 Splitting of the dataset to train and test datasets](#_6m8jrfmsqct4) 12

[**6. Models explored**](#_5627b459mah1) **12**

[6.1 Linear Regression](#_r6yfdaacqt0j) 13

[6.2 Random Forest and Extra Trees Regressors](#_br78r3tf2e2c) 13

[6.3 Gradient boosting models](#_y4kbsrt0ggvr) 13

[6.3 Results from each model](#_78w0vgs6yfu6) 14

[**7. Hyperparameter tuning**](#_axbs3vgrov8v) **16**

[**8. Ensemble Learning**](#_t8v2x3plmbhp) **17**

[**9. Final Results**](#_2r67xmb18j53) **18**

[**10. Solution Novelty**](#_pdnzem5a4kph) **19**

[**11. Future Improvements**](#_1m0rdap6699z) **19**

[**12. Conclusion**](#_q5kdv3ldkrnk) **20**

[**13. Team contribution**](#_o4zspruzx1nx) **21**

### Zillow’s Home Value Prediction (Zestimate)

### 1. Problem statement

In the United States, there are multiple real estate classified websites where properties are listed for buy/rent/sell purposes. Some examples would be Zillow, Realtor.com, and Trulia. However, in each of these websites, we can see a lot of inconsistencies in terms of the pricing of a property and there are some cases when similar properties are priced differently which may lead to the consumers feeling that the prices may not be justified for a particular listed property.

Prices of real estate properties are sophisticatedly linked with our economy. All the real estate transactions in the U.S. are publicly available. Despite this, we do not have accurate measures of housing prices based on the vast amount of data available. Therefore, the goal of this project is to use machine learning to predict the selling prices of houses based on many economic factors. The dataset we will be using is available on the Kaggle website under the competition “Zillow Prize: Zillow’s Home Value Prediction (Zestimate)”. We are provided with a data set containing a full list of real estate properties during the years 2016 and 2017. The data has about 58 features that are represented by the respective columns.

### 2. Challenge Statement

In this project we are challenged to build a model to analyze and accurately predict the log error for each parcelID, for 6 different timestamps (October 2016, November 2016, December 2016, October 2017, November 2017, and December 2017).

After this, we need to submit our predictions on Kaggle and achieve a high ranking in both public and private leaderboards. The private leaderboard is calculated with approximately 49% of the test data and the public leaderboard is calculated with approximately 51% of the train data.

The Zestimate was created to give consumers as much information as possible about homes and the housing market. Zillow is asking you to predict the log error between their Zestimate and the actual sale price, given all the features of a home.

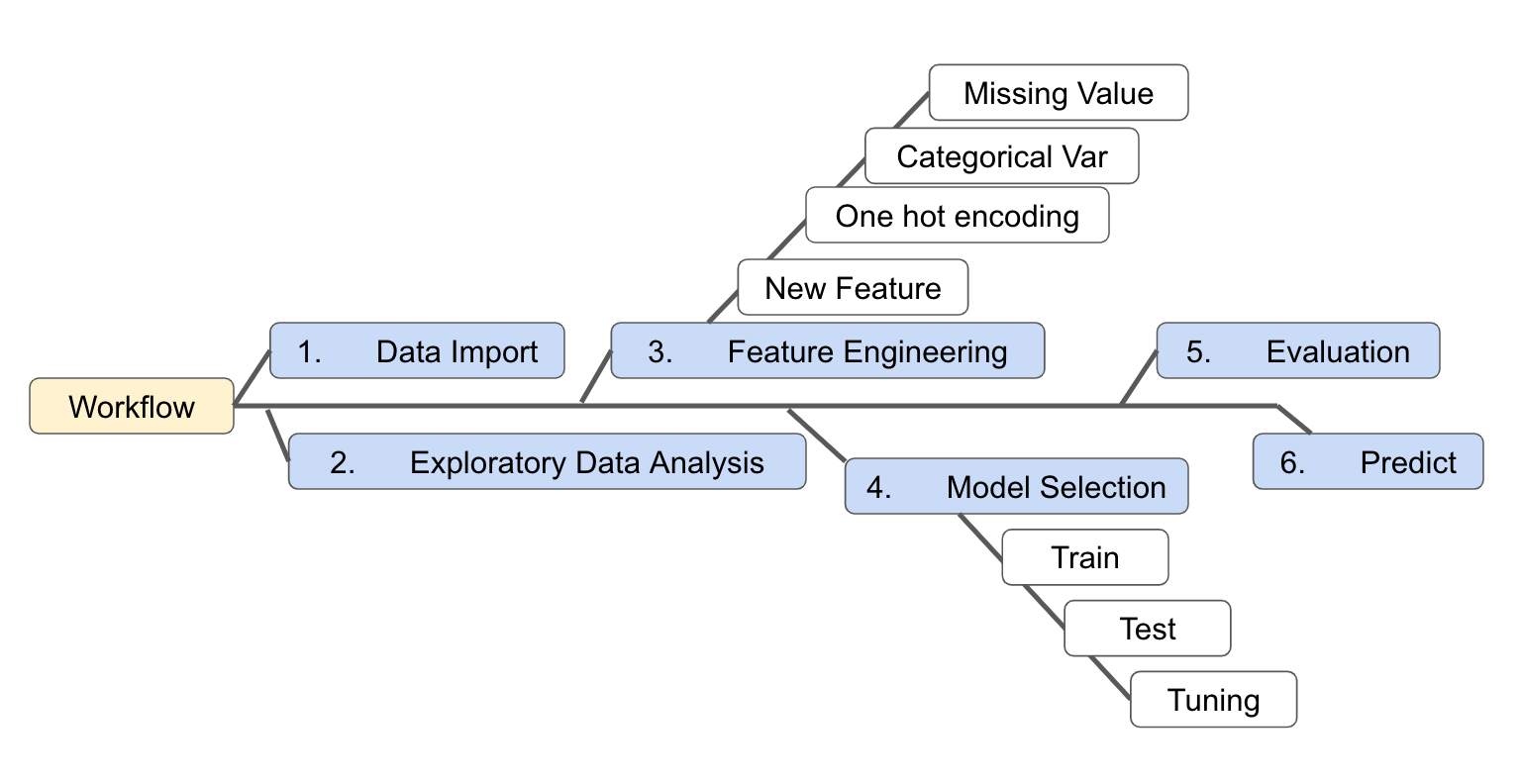
The objective of the Kaggle competition is to predict the log error between the predicted log error and the actual log error. Since the true price of the house is hidden by Zillow, we trained our dataset on a logerror.

The log error is defined in the mathematical formula as shown below:

Therefore, the output of the model is also a logerror between the Zestimate model and the actual price of the house. We then submit the output file to Zillow Prize: Zillow’s Home Value Prediction (Zestimate) to get the final score.

The goal of the competition is to predict the difference between the Zestimate and the actual sales price of homes. This may help Zillow identify where their algorithm falls short.

### 3. Workflow



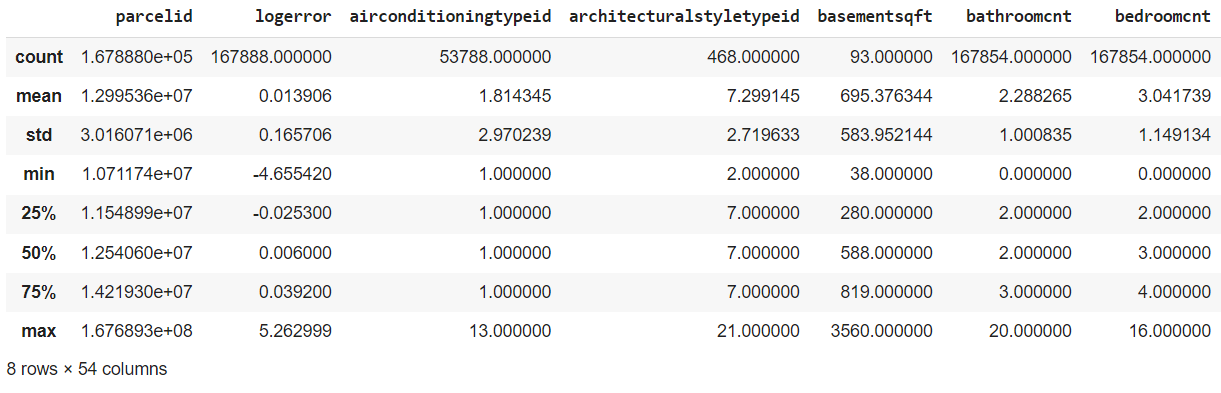
**Figure 3.1 Workflow chart**

The first step is to import data from Kaggle. After retrieving the data, we performed exploratory data analysis on the dataset to gain visual understanding of the data. We then perform feature engineering on the dataset, such as adding new features, one hot encoding, label encoding on categorical variables, and removing missing values to increase the performance of the machine learning models. After that, we built linear regression models, random forest regressor, extra trees regressor, gradient boosting regressor, XGB Regressor, LightGBM, and catboost models, and fed the preprocessed data into the models as input. We also tune the model to increase its accuracy. Finally, we compare the performance of all models on the dataset.

### 4. Exploratory Data Analysis

Exploratory Data Analysis refers to the initial phase of performing initial Investigations on data to discover patterns, anomalies, outliers and to identify any possible trends that may be found in the Zillow Housing Dataset.

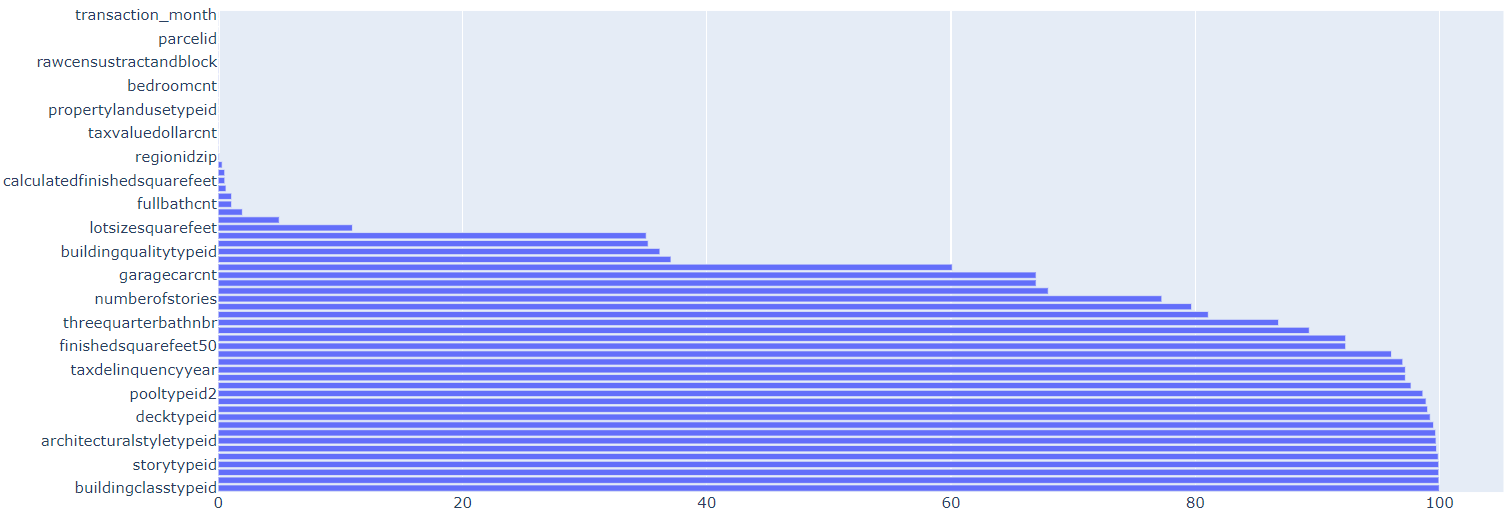
We began by exploring the training and test data to gain more insight into the overall task. As we were predicting log error and training our models on a subset of data, we first wanted to ensure that the data was normally distributed. This is an important step for maintaining consistent distribution and performance. If we randomly split our training data into a training and test set, the model can perform differently on our test data if we do not have the same type of distribution.



**Figure 4.1 Calculating the Statistical Information of the Data using the describe function**

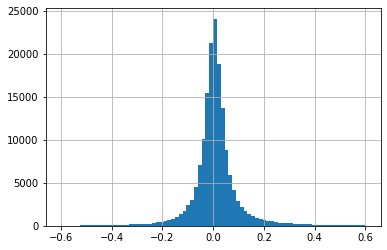
Handling missing values was probably one of the most important processes during feature engineering. we can see in Figure 4.2 about half of the features missed 60% or more. In fact, some features are missing nearly completely. So, we probably have to focus on other features.

Both the training and test data had a large number of missing values that required our attention. In the training data alone, 17 variables had over 90% missingness.

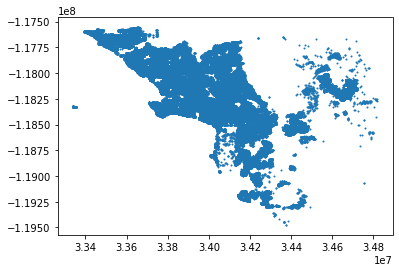
****

**Figure 4.2 Plot the missing value dataframe**

In Figure 4.3, there is a Gaussian (normal) distribution to the log error; this means that the random sample we use to test our model will have the same distribution as our overall data, and we can guard against overfitting.

****

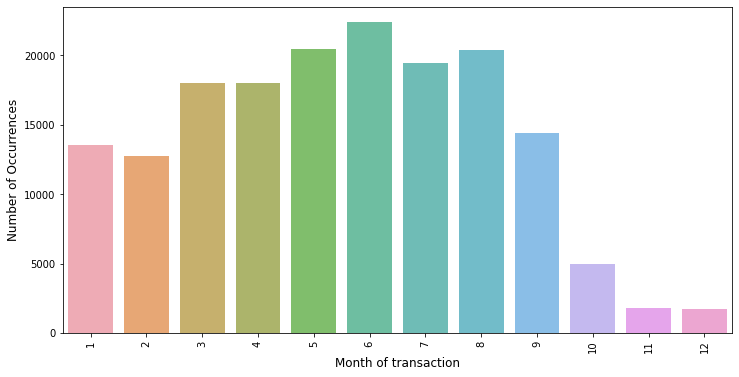
**Figure 4.3 Plot the logerror**

****

**Figure 4.4 Plotting Location all Zillow Housing based on latitude and longitude**

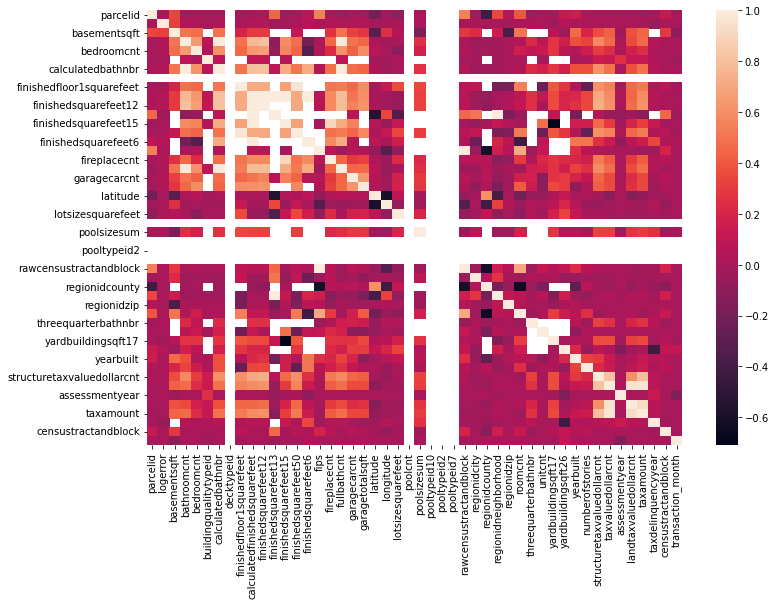
|  |  |
| --- | --- |
| **regionid\_neighborhod == 27080** | **regionid\_neighborhod == 37739** |

**Table 4.5 Plotting Zilow Houses based on Neighbourhood Regions**

****

**Figure 4.6 Zillow Transactions based on Month**

Given the total number of features, we looked at the correlation of numerical features with both the absolute log error and each other to identify variables that might be particularly helpful in prediction.

****

**Figure 4.7 Heatmap to determine the correlation for the continuous variables**

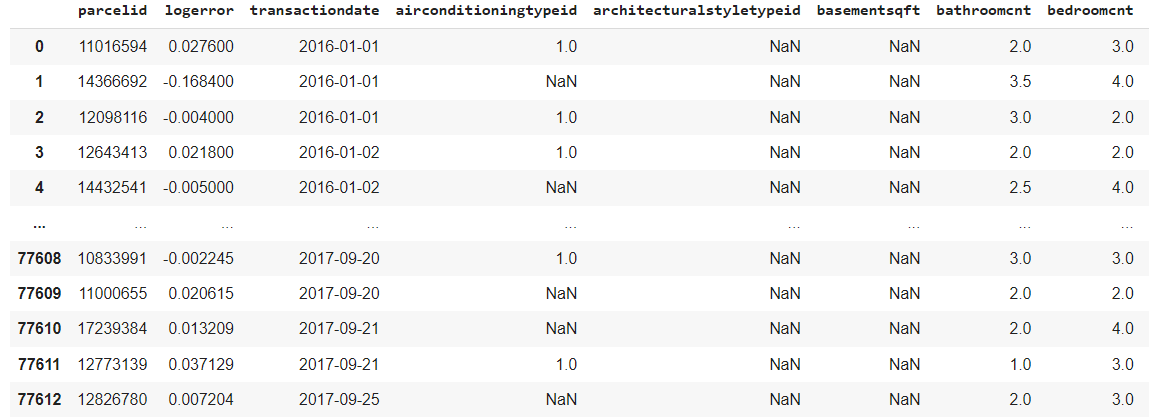
****

**Figure 4.8 Heatmap to determine the correlation for the Categorical variables**

### 5. Feature Engineering

#### 5.1 Data Preparation

We first need to combine the 4 csv files, mainly properties\_2016.csv, properties\_2017.csv, train\_2016\_v2.csv, and train\_2017.csv into a single dataframe.



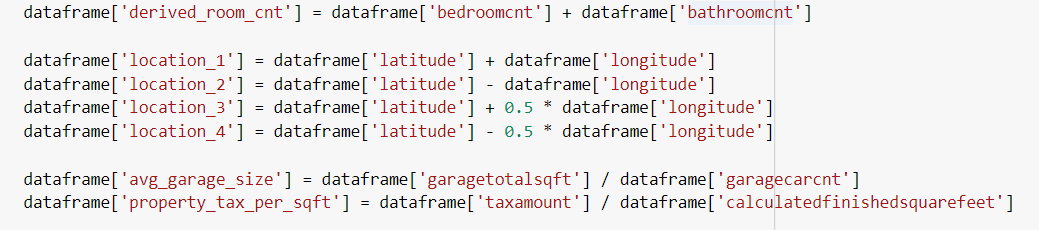
We then started off with exploratory data analysis to take a brief look at the dataset. This involved hypothesis generation, hypothesis testing, exploring the dataset, and univariate and bivariate analysis. Followed, we implemented a series of steps to prepare a proper dataset for both training and testing.

#### 5.2 Combining and processing of features

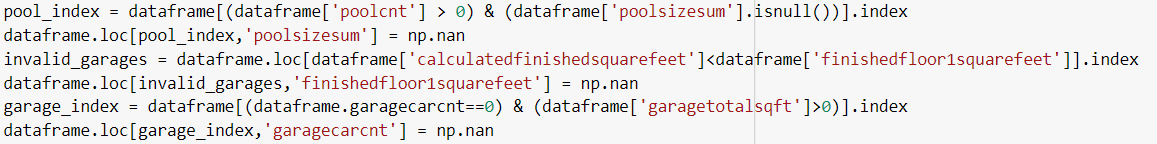
There are some features that can be combined into one single feature. This is done so to reduce the overall dimensions of the dataset, for better accuracy and faster training. One such example is as such:

***Conversion and Combining Features:*** In the picture below, figure 5.2.1 & 5.2.2

* latitude and longitude column values
* Determine Average garage size using garage size
* Calculate property tax based on per square feet data



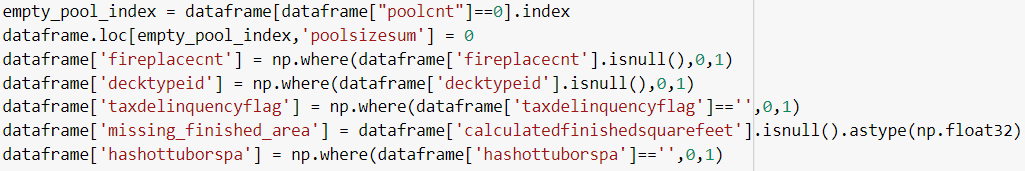
**Figure 5.2.1 Converting Column Features to suitable formats**



**Figure 5.2.2 Combine Multiple Features into 1 Feature**

***Representing Features as Binary: Figure 5.2.3***

1 if feature is available and 0 if feature is unavailable

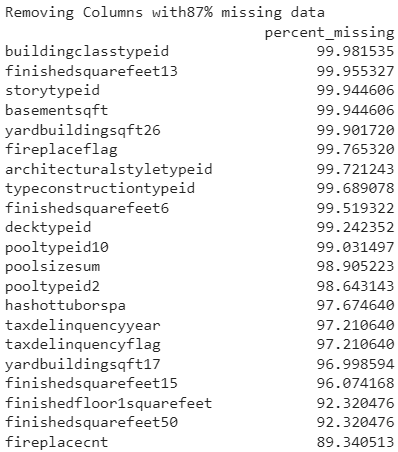


**Figure 5.2.3 Representing Features as Binary**

#### 5.3 Removing features filled with missing data

We have observed that there are some features that contain mostly empty values. This can greatly affect the accuracy of our model prediction.

If the proportion of the missing value exceeds 90%, it is better to remove the feature altogether, since there is not enough information to determine the overall influences of this feature on the output. Figure 4.3.1 shows features, with 87% and above of its data missing in the dataset. Missing data can affect our prediction accuracy, so we have decided to remove these selected features from model training and testing.



**Figure 5.3.1: Features with % missing data**

#### 5.4 Replacing null values

For each data entry, some feature values may be missing. One way to resolve this is to remove the missing value and replace each value with 0. During the model training process, a value of 0 may have little influence on the output.

For the remaining features, for simplicity, we replaced the nan values with 0, using the fillna() function. This is to fit the data into the training model without facing any errors.

#### 5.5 One-hot encoding

One hot encoding is a process to convert categorical features to a form that is better understandable for machine learning algorithms. One-hot encoding is used for categorical features, such as 'airconditioningtypeid' and 'heatingorsystemtypeid'. This is to ensure that the machine learning algorithms do not treat the order of numbers as an attribute of significance.

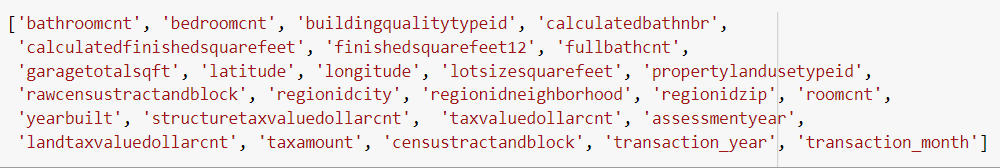
For example, the “'heatingorsystemtypeid'” is represented by a type id between 1 and 25. One Hot Encoding technique will convert the type id into 25 columns, with each column referring to a type. The column of the type of a particular heating or system class will have a value of 1, and others in the same row will have a value of 0.

Otherwise, this can lead to issues with predictions and poor performance. One hot encoding removes the relation between a low integer value with a high integer value in the categorical features.

#### 5.6 Recursive Feature Elimination

In a dataset, it is important to identify the most important features required to predict the price of the house. The tool is available with the sklearn python library.

The recursive feature elimination is popular as it is easy to configure and use because it is effective in selecting the columns in a training dataset that is very relevant in predicting the target variable. You can choose the number of features to select at the end of the algorithm. Different machine learning algorithms were used in the recursive feature elimination function available in the scikit-learn python library. This gave about 26 features.



#### 5.7 Splitting of the dataset to train and test datasets

After finishing the cleaning and the preprocessing portion of the dataset. We split the dataset into train and test datasets, to be used for the training of the machine learning model, followed by predicting on the test dataset. We used to train\_test\_split() function, to randomly split the dataset based on the parameters set.



We chose a test size of 0.2, and the remaining 0.8 as our training dataset. Once the splitting has been completed, we thus fit the training dataset into the model.

### 6. Models explored

As our problem statement requires a regressor model for prediction, here are the following models that we have explored and tested with our training dataset:

1. Linear Regression
2. Random Forest Regressor
3. Extra Trees Regressor
4. Gradient Boosting Regressor
5. XGB Regressor
6. LightGBM
7. Catboost

We used default parameters to train for each model.

#### 6.1 Linear Regression

Linear Regression is one of the simplest models and easiest models to implement. The model predicteds the target value based on the individual values of each feature. During training, it creates a best fit line from the input features. Once training is done, the model will then find the best prediction using the best fit line.

**Advantage**: The biggest advantage is linearity. Hence it makes the estimation procedure simple and easy to understand

**Disadvantage**: It works better with datasets that have linearity between dependent and independent variables.

#### 6.2 Random Forest and Extra Trees Regressors

Random Forest and Extra Trees Regressor models belong to a class of algorithms known as ensemble learning. Both the models construct many decision trees, and combine the results to achieve better results as compared to a single decision tree. The difference between the two is that for Random Forest, it constructs multiple decision trees with different sample subsets. The process of sampling subsets is also known as bootstrapping. On the other hand, the Extra Trees construct trees based on the samples of the entire dataset, and not through bootstrapping.

Random forest is a benchmark model and is popular in solving regression and classification problems. Random forest sums up all the decision trees to reduce overfitting. A tree is traversed until a decision is reached and simple tests are performed on each tree.

**Advantage**: The model can utilize both numerical and categorical data and can perform complex decision boundaries. Requires less data preparation steps.

**Disadvantage**: The decision tree algorithm is highly time consuming in the training phase and has limited performance in regression.

#### 6.3 Gradient boosting models

The remaining four models, Gradient Boosting Regressor, XGB Regressor, Catboost and LightGBM are all variants of gradient boosting. These models also use ensemble learning of decision trees and boosting. The principle of boosting is to boost a set of weak learners into strong learners. It makes instances currently misclassified more important. Each model comes from different open sources, as such there are differences between each model. We decided to test and compare the results of these models.

Gradient-boosted trees are ensembles of decision trees. This model uses boosting technique to iteratively train the decision trees. Each tree is dependent on the previous tree and reduces the discrepancy by learning from the previous tree.

The XGBoost algorithm was used for solving the problem. XGBoost is an implementation of gradient boosted decision tree and it stands as eXtreme Gradient Boosting. XgBoost is known for its flexibility, performance and speed. Compared to other models, XGBoost is fast. This model is best used for tabular dataset and structured and it works best for classification and regression models. XGBoost is enabled with parallel processing making it at least ten times faster than any other tree-based models. It avoids overfitting by Regularization.

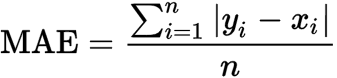
CatBoost builds upon the theory of decision trees and gradient boosting. The main idea of boosting is to sequentially combine many weak models (a model performing slightly better than random chance) and thus through greedy search create a strong competitive predictive model. Because gradient boosting fits the decision trees sequentially, the fitted trees will learn from the mistakes of former trees and hence reduce the errors. This process of adding a new function to existing ones is continued until the selected loss function is no longer minimized.

LightGBM is a fast, distributed, high-performance gradient boosting framework based on a decision tree algorithm, used for ranking, classification and many other machine learning tasks. Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word ‘Light’.

#### 6.3 Results from each model

After training, we predict the test data with each model, and obtain the results to compare performance using the Mean Absolute Error () and the Root Mean Squared Error ().

The measures the average magnitude of the errors in a set of forecasts, without considering their direction. It measures accuracy for continuous variables. The equation is given in the library references. Expressed in words, the is the average over the verification sample of the absolute values of the differences between forecast and the corresponding observation. The is a linear score which means that all the individual differences are weighted equally in the average. is defined by the equation:

****

**Figure 6.3.1 MAE equation**

= mean absolute error

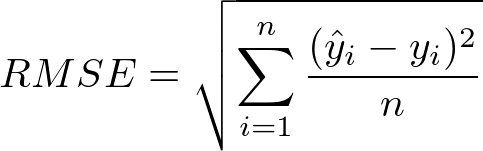
i = predicted house value

i = true value of the house

= total number of training data entry

= a Greek letter called sigma which represents ‘sum’

is a quadratic scoring rule which measures the average magnitude of the error. The equation for is given in both of the references. Expressing the formula in words, the difference between forecast and corresponding observed values are each squared and then averaged over the sample. Finally, the square root of the average is taken. Since the errors are squared before they are averaged, the gives a relatively high weight to large errors. This means the is most useful when large errors are particularly undesirable. is defined by the equation:



**Figure 6.3.2 RMSE equation**

= root-mean-square deviation

i = predicted house value

i = true value of the house

= sample size (the number of observations)

= a Greek letter called sigma which represents ‘sum’

The and the can be used together to diagnose the variation in the errors in a set of forecasts. The will always be larger or equal to the ; the greater the difference between them, the greater the variance in the individual errors in the sample. If the , then all the errors are of the same magnitude

Both the and can range from to . They are negatively-oriented scores, therefore, the lower values are better.

| **Models** | **MAE** | **RMSE** |
| --- | --- | --- |
| Linear Regression | 0.070321 | 0.17012 |
| Random Forest Regressor | 0.070084 | 0.16971 |
| Extra Trees Regressor | 0.077523 | 0.17404 |
| Gradient Boosting Regressor | 0.070173 | 0.16973 |
| XGB Regressor | 0.070133 | 0.16949 |
| LightGBM | 0.069832 | 0.16960 |
| Catboost | 0.068812 | 0.16951 |

Based on the results as shown in the table above, Catboost shows the best result with the lowest and score. We then select the catboost as our main model, and conduct hyperparameter tuning.

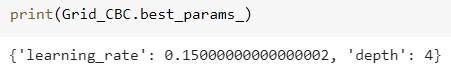
### 7. Hyperparameter tuning

The purpose of hyperparameter tuning is to search for the optimal hyperparameter values for Catboost training. We used the RandomizedSearchCV() function, to find the optimal values from the following parameters:

1. Learning Rate: Values between 0 to 5
2. Depth: Values from 4 to 10

### 

Here's the final optimal learning rate and depth values:



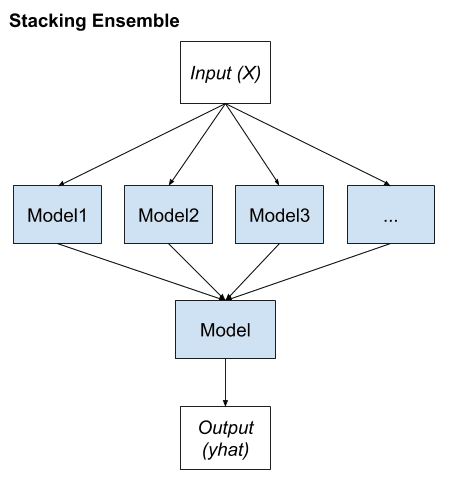
After which, we retrain the model with the optimal hyperparameter values instead of default values, and compare the results between the two.

| **Models** | **MAE** | **RMSE** |
| --- | --- | --- |
| Catboost (Without optimal hyperparameters) | 0.068812 | 0.16951 |
| Catboost (With optimal hyperparameters) | 0.068807 | 0.16933 |

The result shows a slight improvement after hyperparameter tuning.

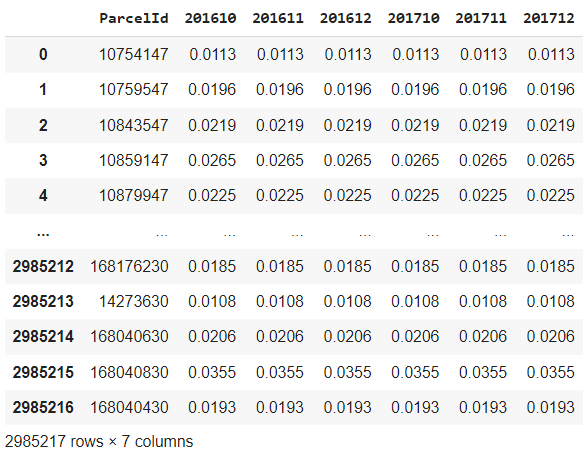
### 8. Ensemble Learning

To achieve higher accuracy, instead of using just one individual model for prediction, we trained for 5 CatBoost models, each using a different seed. The 5 models will be used to predict the log errors for the 6 different timestamps as required by the Kaggle competition. Then we take the average value across the results from the 5 models. Ensemble learning helps to improve machine learning results by combining several models together. This approach allows the production of better predictive performance.



**Figure 8.1 Ensemble Model**

### 9. Final Results



**Figure 8.2 Submission.csv File**

|  | **Leaderboard Scores** | **Leaderboard Ranking** | **Percentile (Top)** |
| --- | --- | --- | --- |
| Public | 0.06429 | 537/3772 | 14.24% |
| Private | 0.07507 | 142/3772 | 3.76% |

### 

**Figure 9.1 Evaluation score and ranked position of our prediction results for the Zillow competition in Kaggle**

### 10. Solution Novelty

*Feature Engineering*

Feature engineering is considered as a smart technique which uses the domain knowledge of data to create additional features that makes the machine learning algorithm work and makes the model look simpler. When executed correctly, feature engineering helps in increasing the predictive power of the algorithm by creating features from the raw data which will help facilitate the process as well as the model.

*Ensemble Learning*

Ensemble learning is used in machine learning because the generalization ability (predictive power) of an ensemble is greater than a single “learner” (model). For the best results, correlation between learners should be minimized - where one model is deficient, another might excel. Ensemble learning did improve our results slightly.

*Model Stacking*

Stacking involves using a machine learning model to learn how to combine predictions from models. This ensures that your model does not rely on a single model and instead takes a weighted approach. After retrieving the model, we would use the predictions as features for a higher level model.

*Hyperparameter Tuning*

Hyperparameters are the “higher-level” model properties that have an effect on the model training process (e.g., learning/shrinkage rate, tree depth). Our team used a random search for optimal hyperparameters.

*Challenges faced and solved*

Given the amount of missing data and relatively large number of features in this project, feature engineering and hyperparameter tuning were very important with respect to reducing prediction error. Using a simple Catboost model was the most significant factor to improving our results.

### 11. Future Improvements

With more time, we would continue to iterate on our workflow and introduce new strategies to improve our model results. There were several modeling techniques that the team did not have a chance to explore due to time constraints, including Timer Series Prediction - ARIMA. The ability to make predictions based upon historical observations creates a competitive advantage. In the domain of machine learning, there’s a specific collection of methods and techniques particularly well suited for predicting the value of a dependent variable according to time.

ARIMA stands for Auto Regressive Integrated Moving Average. Autoregressive models (AR) operate under the premise that past values have an effect on current values. AR models are commonly used in analyzing nature, economics, and other time-varying processes. As long as the assumption holds, we can build a linear regression model that attempts to predict the value of a dependent variable today, given the values it had on previous days. Moving Average Model (MA) assumes the value of the dependent variable on the current day depends on the previous day's error terms. The ARMA model is simply the combination of the AR and MA models. The ARIMA (aka Box-Jenkins) model adds differencing to an ARMA model. Differencing subtracts the current value from the previous and can be used to transform a time series into one that’s stationary.

In addition, with the large number of missing data we can maybe try to implement an imputation strategy so that we can preserve all cases by replacing missing data with an estimated value based on other available information. Once all missing values have been imputed, the data set can then be analyzed using standard techniques for complete data.

### 12. Conclusion

Purchasing, renting or selling a real estate property is often the highest and most important purchase that a person can make. People are willing to spend upwards of thousands and/or millions to buy/rent/sell a property. However, when it comes to paying for said property, one needs to ensure that a fair sale price is placed on the property, and also ensure that the price of the property is justified. Accurate estimation of home prices is an extremely challenging and serious problem because there are numerous factors that influence the value of a property. These factors include region, size of property, tax amount, interest rate, and the overall economy. In the past, people have mostly relied on real estate agents to value their property which leads to huge inconsistencies between the actual value of a property and the proposed sale prices.

Therefore, in this report, we have presented various features to consider while predicting the real estate properties using machine learning. We have performed the preliminary assessment of the dataset like merging data files and properties files, cleaning the data by dropping columns with more than 87% NaN values. We also performed the correlation analysis to find the highly correlated continuous features. Using MAE and RMSE, we measured the performance of all models (linear regression, random forest regressor, extra trees regressor, gradient boosting regressor, XGB Regressor, LightGBM and catboost). Overall, we found that for this particular problem, CatBoost with its default hyperparameters performed the best with lowest MAE score of 0.068807 and RMSE score of 0.16933 and to further improve our performance on the leaderboard, we did hyperparameter tuning and used ensembling and stacking.

### 13. Team contribution

| **NAME** | **Roles and contributions** |
| --- | --- |
| ERICA ER MING CHEE | * Researched feature engineering methods such as one-hot encoding. * Researched and learned about linear regression and random forest. * Explored hyperparameter tuning and ensemble learning. |
| GAVIN NEO JUN HUI | * Researching some feature engineering methods such as one-hot encoding * Exploring models such as linear regression and some gradient boosting methods * Exploring ensemble learning |
| VEDULA KARTIKEYA | * Helped out with Feature Engineering of Data. * Attempted to explore different methods to find outliers in the data and perform feature engineering to ensure that the data is more normally distributed. * Learnt and Implemented Random Forest and LightGBM model to carry out predictions. * Learnt and adapted Optuna Framework to finetune LightGBM parameters (Fine tuning with Optuna gave similar results to default value of parameters and hence it was omitted). * Stacking of LightGBM and CatBoost to achieve better results. |
| CHOCKALINGAM KASI | * Setup the Model Pipeline for Training and Testing. * Carried out Exploratory Data Analysis. * Wrote Custom functions for Feature Engineering such as identifying best features to be used in the model and feature processing techniques * Learnt and implement models such as decision Trees, RandomForest and CatBoost to carry out predictions. * Hyperparameter Tuning of CatBoost Model to improve accuracy of the model on the Kaggle Platform. |