

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**  
**SINGAPORE**

**SC4000: Machine Learning  
GROUP 4 Report**

<b>No.</b>	<b>Name</b>	<b>Matriculation Number</b>
<b>1</b>	<b>Darren Choo Jian Hao</b>	<b>U2121223J</b>
<b>2</b>	<b>Aloysius Ang</b>	<b>U2120520B</b>
<b>3</b>	<b>Wang Shiqiang</b>	<b>U2121266G</b>
<b>4</b>	<b>Gan Hao Yi</b>	<b>U2120769F</b>
<b>5</b>	<b>Foo Jin Rui</b>	<b>U2120527E</b>

## **Table of Contents**

<b>1. Introduction</b>	<b>3</b>
1.1 Project Outline and Aim:	3
1.2 Dataset Description:	3
<b>2. Exploratory Data Analysis</b>	<b>3</b>
2.1 Plot Log Error Against Time:	3
2.2 Variation of Log Error with Respect to Latitude and Longitude:	4
2.3 Null-Value Counts by Columns:	5
2.4 Correlation Matrix for Ordinal Features:	6
2.5 Correlation Matrix for Numerical Features:	7
<b>3. Proposed Methodologies</b>	<b>8</b>
3.1 Model Selection:	8
3.2 Ensemble Learning:	9
<b>4. Data Preprocessing</b>	<b>10</b>
4.1 Data Concatenation:	10
4.2. Data Splitting:	10
4.3. Preprocessing Techniques:	10
<b>5. Hyperparameter Tuning</b>	<b>12</b>
<b>6. Experiment</b>	<b>14</b>
<b>7. Final Test Score</b>	<b>17</b>
<b>8. Conclusion</b>	<b>18</b>
<b>9. References</b>	<b>19</b>

## **1. Introduction**

### **1.1 Project Outline and Aim:**

The aim of the Zillow Prize competition is to develop algorithms that will help predict the future sale prices of homes. This purpose is achieved through improving the Zestimate residual error. As houses are usually one of the most expensive and important purchases a person would make in their lives, the Zillow's Zestimate hopes to provide people a platform to monitor this asset by providing as much information about homes and the housing market free of charge.

### **1.2 Dataset Description:**

We are tasked to predict the log-error between their Zestimate and the actual sale price, given the attributes of a property. The log-error is calculated as follows:

$\text{logerror} = \log(\text{Zestimate}) - \log(\text{SalePrice})$ . The attributes of a property includes its *Parcel ID* (the property's unique identifier), the *airconditioning type*, *architectural type* etc. There are 58 features in total, including the *Parcel ID*. The datasets given to us are as follows:

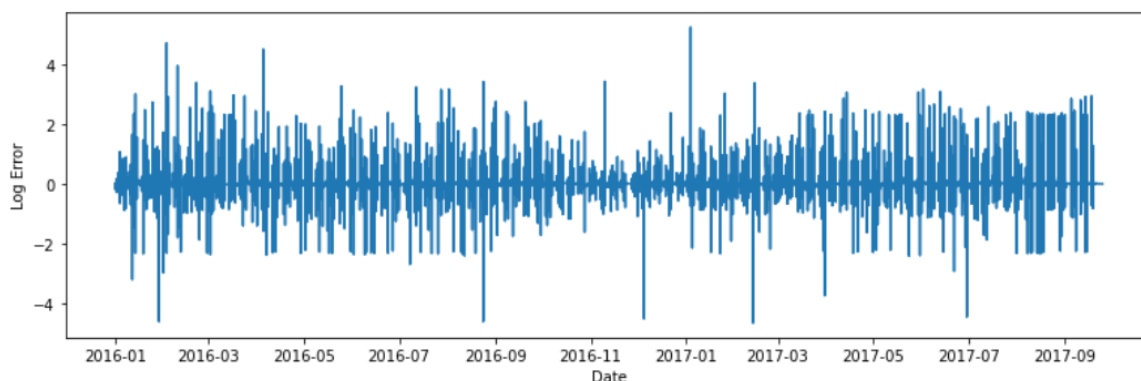
- *Properties\_2016 / Properties\_2017*: Contains the 58 features of each property in the respective years
- *Train\_2016 / Train\_2017*: Contains the *Parcel ID*, *log error* and *transaction dates* of the properties that we sold in the respective years.

## **2. Exploratory Data Analysis**

In this section, we conduct univariate and multivariate analyses to enhance our comprehension of the provided data.

### **2.1 Plot Log Error Against Time:**

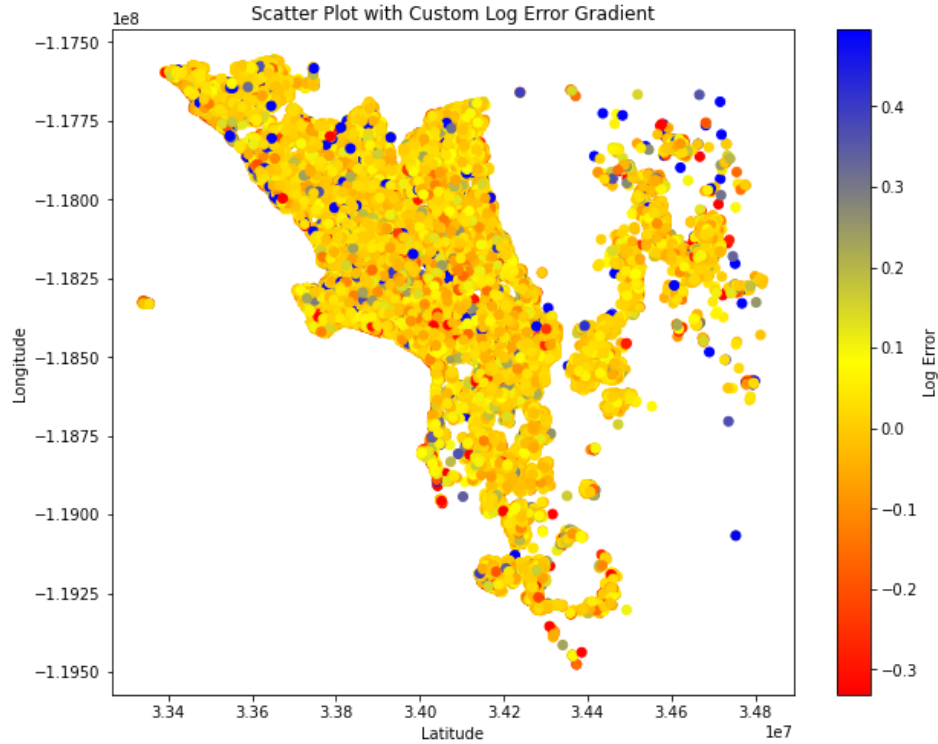
In *Figure 1*, the plot of *log error* against *time* exhibits a stationary pattern, which suggests that the statistical properties of this time series remain constant over time. This stationary behavior indicates that the 2016 and 2017 housing data can potentially be trained using a unified model, as the underlying data characteristics do not exhibit significant variations over the observed time period.



**Figure 1.** Plot of Log Error Against Date

## 2.2 Variation of Log Error with Respect to Latitude and Longitude:

Figure 2 illustrates the variation in *log error* when plotted on a *latitude* and *longitude* chart. In this representation, the *log error* values are intentionally clipped, ranging from the 1<sup>st</sup> percentile to the 99<sup>th</sup> percentile, in order to enhance the visibility of the color gradient. From the plot, it is evident that a majority of the *log error* predictions cluster around 0, and the *log error* values seem to appear randomly distributed across the geographical region. This supports the absence of any discernible geographical impact on log errors.



**Figure 2.** Variation of *Log Error* with *Latitude* and *Longitude*

## 2.3 Null-Value Counts by Columns:

In Figure 3, it is evident that features such as “*storytypeid*” and “*basementsqft*” exhibit an exceptionally high count of null values, exceeding 99.7%. This observation suggests that these particular features may not be suitable for further analysis within our dataset.

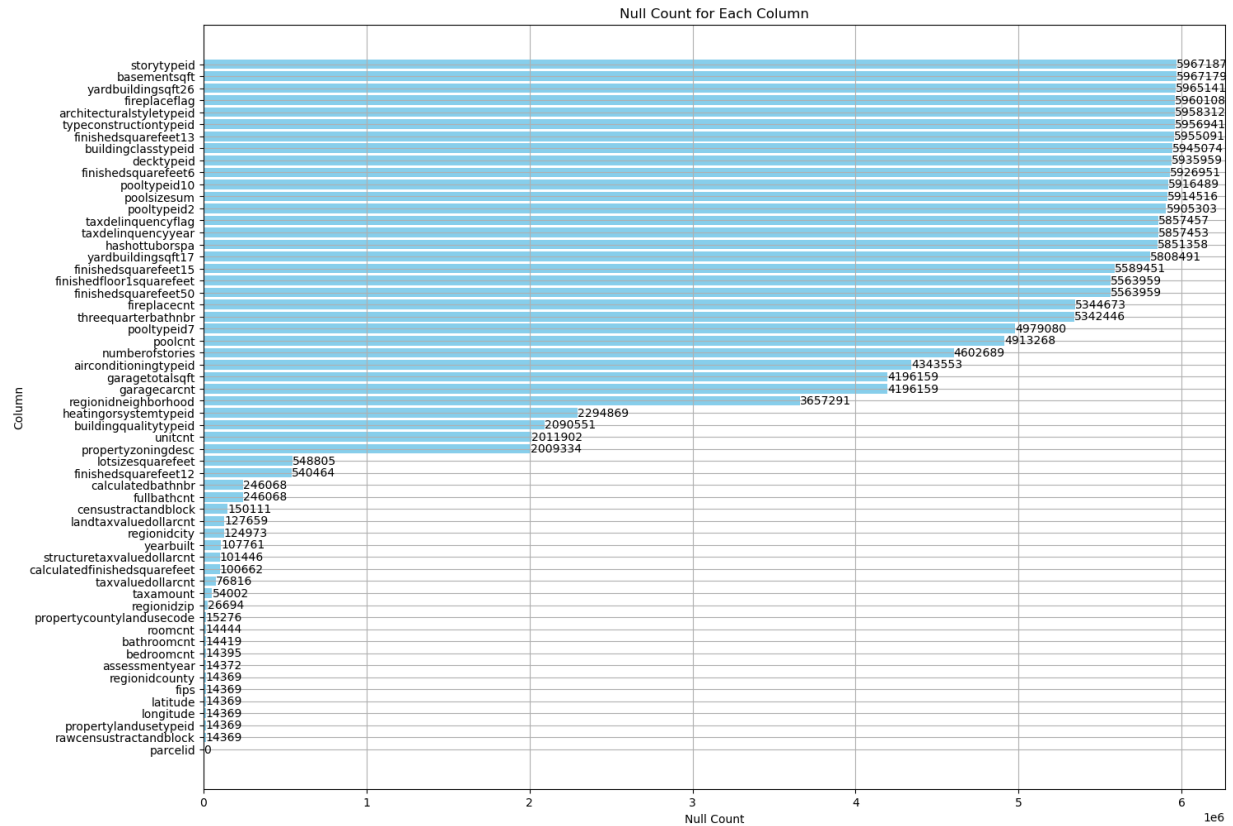
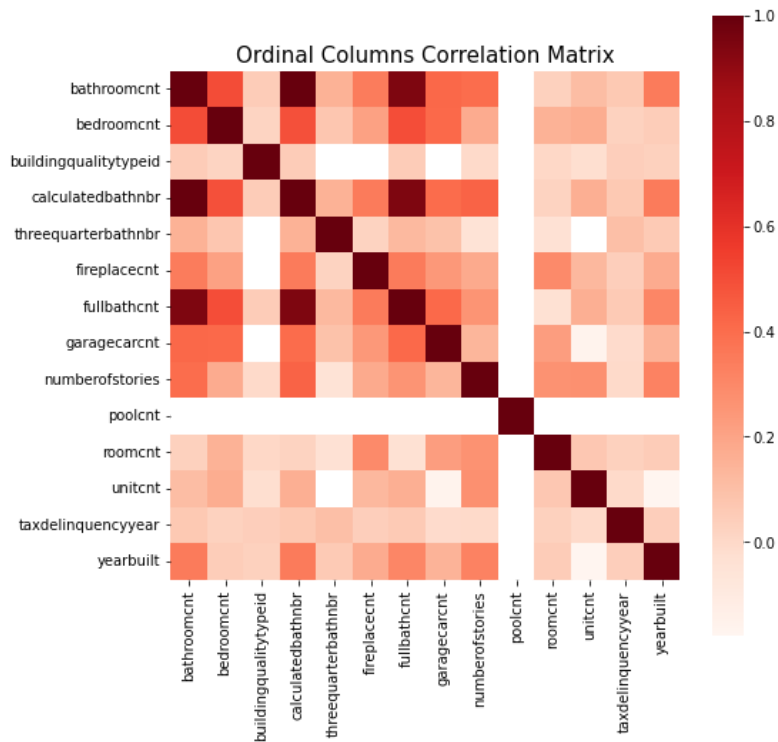


Figure 3. Null-Value Counts by Columns

## 2.4 Correlation Matrix for Ordinal Features:

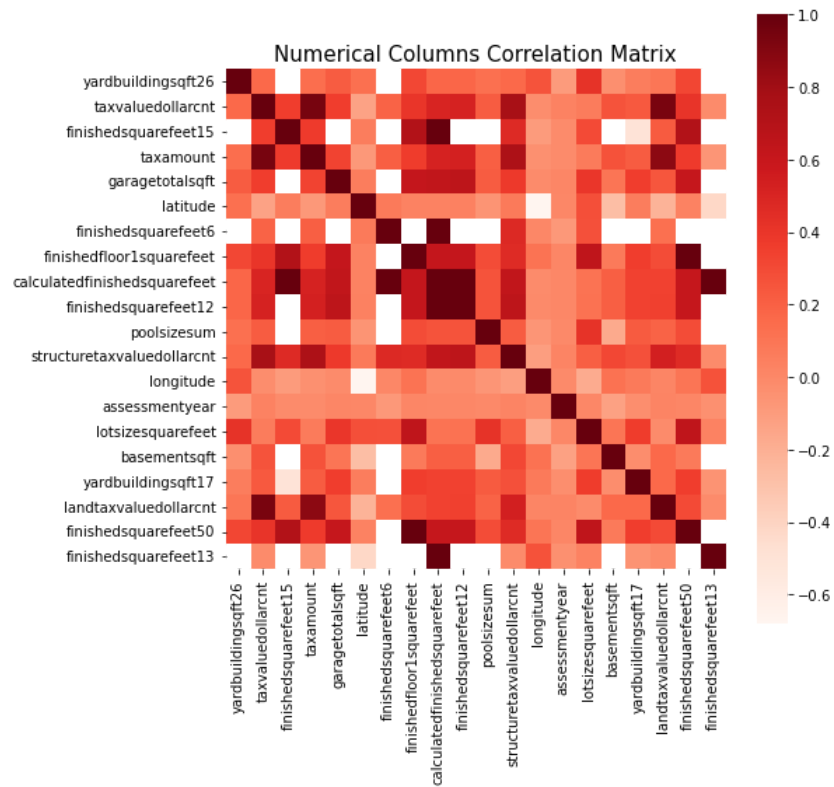
In *Figure 4*, we present the correlation matrix of all ordinal features, utilizing the *Kendall Tau* Correlation Coefficient. The figure utilizes color gradient scaling to emphasize correlation strength, revealing significant associations among specific features, particularly a strong correlation between "*bathroomcnt*" and "*calculatedbathnbr*".



**Figure 4.** Correlation Matrix for Ordinal Features

## 2.5 Correlation Matrix for Numerical Features:

In Figure 5, the correlation matrix of all numerical features utilizing the Spearman Correlation Coefficient is displayed. The color gradient scaling in the figure highlights the strength of correlations, making it evident that certain features, such as "*taxamount*" and "*taxvaluedollarcnt*," are strongly correlated with each other.



**Figure 5.** Correlation Matrix for Numerical Features

### **3. Proposed Methodologies**

#### **3.1 Model Selection:**

We have experimented on 5 different models as follows.

##### **3.1.1 Histogram-based Gradient Boosting Regressor:**

Histogram-based Gradient Boosting Regressor, implemented in Scikit-learn as HistGradientBoostingRegressor, is a faster form of the gradient boosted tree algorithm.

Gradient boosting is essentially an ensemble of decision trees algorithms, where decision trees are typically constructed by exhaustively searching for the best split points for each feature, which can be computationally expensive and time consuming. HistGradientBoostingRegressor addresses this issue by using histograms to speed up the training process. The model is also able to handle null values in the data.

##### **3.1.2 Feed-Forward Network:**

Our group has expanded our investigation by delving into deep learning to predict the target variable by using a fully connected neural network.

To ensure the comprehensiveness of our approach, we conducted a systematic approach of exploring different architectures of our Feed-Forward Network. This involved fine-tuning the number of hidden layers and diversifying our selection of non-linear activation functions beyond the conventional ReLU function. In addition, we have also employed early stopping to safeguard against any potential overfitting of the model.

To implement this, we chose to create a custom fully connected neural network using the PyTorch library in Python, which provides us with the flexibility to easily tailor the model's architecture to align with our objectives.

##### **3.1.3 XGBoost:**

We selected the XGBRegressor model from XGBoost, which implements Gradient Boosting. It is an ensemble learning algorithm that works by sequentially building a series of decision trees to create a strong predictive model.

XGBoost minimizes a regularized L1 and L2 objective function that combines a convex loss function based on the difference between the predicted and target output and a penalty term for model complexity. XGBoost adds L1 and L2 regularization to the objective function as penalty terms during training to control the complexity of the model. This is to prevent overfitting on the model.

The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It uses a gradient descent algorithm to minimize the loss when adding new models.



#### 3.1.4 LightGBM:

LightGBM is a powerful gradient boosting framework for machine learning. It excels at handling large datasets and is known for its speed and efficiency.

LightGBM uses a histogram-based approach for finding the best splits in decision trees, reducing memory usage and boosting training speed. LightGBM can handle categorical data without the need to manually encode or preprocess it. It also does not require us to explicitly impute missing data as it has built-in support for missing values during training.

For our LightGBM boosting algorithm, we tested out both the conventional gradient boosting decision tree as well as Dropouts meet Multiple Additive Regression Trees (DART) which introduces a dropout-like regularization technique inspired by neural networks. In DART, during the training process, a random subset of the trees is dropped (removed) at each iteration, preventing the model from overfitting the training data.

#### 3.1.5 CatBoost:

CatBoost is a machine learning algorithm and open-source Python library specifically designed for gradient boosting on decision trees. It is particularly well-suited for tabular data and is known for its robustness, high performance and ease of use. CatBoost stands for "Categorical Boosting", highlighting its strength in handling categorical features in a dataset.

CatBoost allows us to handle categorical data without the need to manually encode or preprocess it. Additionally, CatBoost does not require us to explicitly impute missing data as it has built-in support for missing values during training.

#### 3.2 Ensemble Learning:

For ensemble learning, we used multiple models to obtain better predictive performance than a singular base model. We made use of stacking ensemble learning, which takes the average of the outputs of the models in the ensemble to return one final prediction.

The final ensemble learning framework would be more robust than the individual models that constitute the ensemble as ensembling reduces the variance in the prediction errors.

## 4. Data Preprocessing

Prior to training, we have to preprocess our data in order to fit our dataset into our models.

### 4.1 Data Concatenation:

We performed inner joins on "parcelid" to merge the datasets for "properties\_2016" and "train\_2016" and on "parcelid" for "properties\_2017" and "train\_2017".

After the joins, we will concatenate the two resulting datasets to create a unified training dataset, incorporating all 58 features and the log error ground truth.

### 4.2. Data Splitting:

Next, we split the unified training dataset into a training set and a validation set using a 70%:30% holdout split. This separation will allow us to evaluate the performance of our models.

### 4.3. Preprocessing Techniques:

Regarding the preprocessing techniques described below, we will initially apply them to the training set and then apply the same transformations to the validation and test sets. This approach ensures that our data preparation remains consistent.

It is worth mentioning that not all preprocessing techniques will be used in conjunction. Various models will employ different combinations of these preprocessing methods, and we have provided a comprehensive summary of the results obtained for each model, along with their corresponding preprocessing steps, in *section 6: Experiment*.

Preprocessing Technique:	Description:	Purpose:
1. Data Normalization	Scale and center data to a common range or distribution.	Ensure all features are on a common scale, making them suitable for modeling and analysis.
2. Outlier Removal	Removing outliers from each feature.	Outliers are extreme values found outside the range of what is expected and is unlike the other data. These may cause results to be skewed and inaccurate.
3. Data Imputation	Filling null or missing values.	Some models may not work with null values. Null values can also shrink the sample size, distort statistics and affect performance.

4. Highly Correlated Features Removal	Removing features that are highly correlated to other features.	Having multiple highly correlated features will not only not bring additional information but will also increase the complexity of the models, increasing the risk of errors.
5. Lowly Correlated Features Removal	Removing features that have a low correlation with the target variable.	Features that have a low correlation with the target variable are unlikely to provide useful information during model training.
6. Label Encoding	Replacing categorical labels (including null values) with our defined labels.	This removes null values from categorical columns and is able to produce a more interpretable model. It could increase performance in some models as well.
7. Nominal Category Removal through ANOVA	Removing nominal features with a p-value less than 0.05 in a one-way ANOVA test with the target variable.	Features that do not pass the one-way ANOVA test often have closely related levels, suggesting that they may provide similar or redundant information when predicting the target variable, rendering them potentially irrelevant.
8. Removal of columns with 1 or > X no. of categories	Removing categorical columns which contain less than / more than a certain number of different categories (1 or > X different unique categories).	Columns with 1 unique value are not useful as it is pointless, and columns with too many unique categories will have too many splits.
9. Removing columns with > X% of missing data	Removing columns which contain over a certain percentage of null values.	These columns may not be useful as they contain too many null or missing values.
10. Prune columns with inadequate / Excessive Uniqueness	Removing columns that have too little / too many unique values.	Features with too little unique values do not value add while features with too many unique values can introduce noise to the model.
11. Using features with highest SHAP importance	Using only features that have good importance for the model.	Removing features that are not used by the model or did not provide much information is able to make the model less complex and focus more on the relationships between the other more important features.

12. Date pre-processing	Instead of splitting the data into training and test datasets randomly, we split according to date	The data in the project is a time series data as house prices are likely to be fluctuating as time goes on. Splitting the data according to date would mean that a better training dataset would be passed into the model.
-------------------------	--	--

## 5. Hyperparameter Tuning

For our 5 models, we tuned their most important hyperparameters and obtained the best set through cross validation.

General Hyperparameter	
<b>Learning rate</b>	To determine how quickly or slowly the model learns, converges, and adapts to the data.

Histogram-based Gradient Boosting Regressor	
<b>Max Iterations</b>	The maximum number of iterations of the boosting process, which is also the maximum number of trees. - Best value: 100
<b>Max leaf nodes</b>	The maximum number of leaves for each tree. - Best value: 15
<b>Min samples leaf</b>	The minimum number of samples per leaf. - Best value: 10

XGBoost	
<b>N_estimators</b>	Number of gradient boosted trees. - Best value: 30
<b>Max_depth</b>	Maximum tree depth for base learners. - Best value: 6
<b>Learning rate</b>	Boosting learning rate. - Best value: 0.06

LightGBM	
<b>Max_bin</b>	Max number of bins that feature values will be bucketed in. - Best value: 1028

<b>Boosting</b>	Boosting algorithm. - Best value: DART
<b>Num_iterations</b>	Number of boosting iterations. - Best value: 30

<b>CatBoost</b>	
<b>Iterations</b>	The number of boosting iterations. - Best value: 1000
<b>Depth</b>	The maximum depth of individual decision trees. - Best value: 6
<b>L2_leaf_reg</b>	L2 regularization term applied to the leaf values of decision trees. - Best value: 3

<b>Feed-Forward Network</b>	
<b>Hidden Layers</b>	The number of hidden layers in the feedforward neural network. - Best value: 5
<b>Optimiser</b>	An algorithm that adjusts the model's parameters during training to minimize the loss function. - Best value: Adam
<b>Activation Function</b>	A function to decide the activation level of a neuron. - Best value: ReLU

## 6. Experiment

We will be providing the top 3 Kaggle Scores that our models have managed to obtain, alongside the relevant details of the models. This is done with the best set of hyperparameters that we have already tuned for our respective models.

It is important to note that the validation score for each model instance may not serve as an accurate indicator of the overall Kaggle score. This discrepancy arises because the validation score is heavily influenced by the specific combination of preprocessing techniques employed by each model instance. For instance, it's natural for the validation score to decrease significantly when outliers are removed, yet this change may not be reflected in the Kaggle score.

For our data preprocessing techniques, please refer to section 4.3 for the description of it.

Histogram-based Gradient Boosting Regressor			
Data Processing Techniques:	Hyperparameter:	Validation Score:	Kaggle Score:
6, 12	learning_rate: 0.01 max_iter: 100 max_leaf_nodes: 15 min_samples_leaf: 10	0.06929	0.06511
6, 11, 12	learning_rate: 0.01 max_iter: 100 max_leaf_nodes: 15 min_samples_leaf: 10	0.06929	0.06512
6, 9, 10, 11	learning_rate: 0.01 max_iter: 100 max_leaf_nodes: 15 min_samples_leaf: 30	0.06992	0.06522

XGBoost			
Data Processing Techniques:	Hyperparameter:	Validation Score:	Kaggle Score:
2, 6, 8	N_estimators: 30 Max_depth: 6 Learning rate: 0.06	0.03288	0.0648

6, 11	N_estimators: 30 Max_depth: 6 Learning rate: 0.06	0.06886	0.06522
6, 8	N_estimators: 30 Max_depth: 6 Learning rate: 0.06	0.06898	0.06507

LightGBM			
Data Processing Techniques:	Hyperparameter:	Validation Score:	Kaggle Score:
6, 8, 9	max_bin: 1028 boosting: gbd num_iterations: 30 learning_rate: 0.1	0.06882	0.06502
1, 6, 8, 9, 10	max_bin: 1028 boosting: dart num_iterations: 30 learning_rate: 0.1	0.06880	0.06501
1, 6, 8, 9, 10	max_bin: 1028 boosting: dart num_iterations: 30 learning_rate: 0.1 Ensemble: 10 models	0.06883	0.06512

CatBoost (all results are obtained from Ensemble Learning using 5 models)			
Data Processing Techniques:	Hyperparameter:	Validation Score:	Kaggle Score:
3, 6, 8, 10	iterations=1000, learning_rate=0.03, depth=6, l2_leaf_reg=3	0.06803	0.06425
3, 6, 8, 9, 10	iterations=1000, learning_rate=0.03, depth=6, l2_leaf_reg=3	0.06804	0.06427
3, 4, 6,	iterations=1000,	0.06813	0.06430




8, 9, 10	learning_rate=0.03, depth=6, l2_leaf_reg=3		
----------	--	--	--

Feed-Forward Network			
Data Processing Techniques:	Hyperparameter:	Validation Score:	Kaggle Score:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10	Learning_rate: 0.001 Hidden_layers: 5 Optimizer: Adam Activation_function: ReLU	0.0603	0.0647
1, 3, 4, 5, 6, 7, 8, 9, 10	Learning_rate: 0.001 Hidden_layers: 6 Optimiser: Adam Activation_function: PreLU	0.0678	0.06478
1, 3, 6, 8, 9, 10	Learning_rate: 0.001 Hidden_layers: 5 Optimiser: SGD Activation_function: SiLU	0.0681	0.06481











## 7. Final Test Score

The model that provided us with the best test score was the ensemble CatBoost with 1000 iterations for each model.

Submission and Description		Private Score 	Public Score 
	<b>submission_ensemble_1000iterations.csv</b> Complete (after deadline) · 17s ago	<b>0.07512</b>	<b>0.06425</b>

**Figure 6.** Top Kaggle Score

Overview	Data	Code	Models	Discussion	Leaderboard	Rules	Team	Submissions	Late Submission	
455	Sed Bennett							0.06424	64	6y
456	Me							0.06424	32	6y
457	terenceflow							0.06424	20	6y
458	Issam28							0.06424	10	6y
459	namakemono							0.06424	5	6y
460	graymant							0.06425	53	6y
461	jonsey							0.06425	60	6y
462	CeShine Lee							0.06425	11	6y

**Figure 7.** Kaggle Score on Leaderboard

According to the leaderboard, we ranked **460th** out of 3772 submissions. Thus we have achieved the **Top 12.2%** out of all submissions in the competition.

## **8. Conclusion**

Our group has embarked on an enriching journey, exploring a diverse array of machine learning models that extend beyond the scope of this module. Along the way, we've gained valuable insights into different preprocessing techniques and hyperparameters fine-tuning for models to achieve improved results.

We intended to explore advanced imputation methods, such as KNN Imputation and imputation using regression, to enhance the robustness of our models. Unfortunately, limitations in computational resources hindered us from pursuing these avenues.

To further elevate our analysis, future work could involve the exploration of various regularization techniques, like L1 (Lasso) or L2 (Ridge), aimed at enhancing the generalization of our models by preventing overfitting. This addition would contribute to the overall completeness of our approach and improve the reliability of our predictions.

## **9. References**

- (n.d.). XGBoost Documentation — xgboost 2.0.1 documentation. Retrieved November 4, 2023, from <https://xgboost.readthedocs.io/en/stable/>
- (n.d.). CatBoost - open-source gradient boosting library. Retrieved November 4, 2023, from <https://catboost.ai/>
- (n.d.). Welcome to LightGBM's documentation! — LightGBM 4.0.0 documentation. Retrieved November 7, 2023, from <https://lightgbm.readthedocs.io/en/stable/>
- (2019, March 9). kaggle. Retrieved November 7, 2023, from <https://www.kaggle.com/competitions/zillow-prize-1/discussion/47283>
- Brownlee, J. (2021, April 27). *Histogram-Based Gradient Boosting Ensembles in Python - MachineLearningMastery.com*. Machine Learning Mastery. Retrieved November 7, 2023, from <https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/>
- Guide, S. (2023, March 30). *What is XGBoost? | Introduction to XGBoost Algorithm in ML*. Analytics Vidhya. Retrieved November 4, 2023, from <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>
- sklearn.ensemble.HistGradientBoostingRegressor* — *scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved November 7, 2023, from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html>
- XGBoost explainability with SHAP*. (n.d.). Kaggle. Retrieved November 4, 2023, from <https://www.kaggle.com/code/bryanb/xgboost-explainability-with-shap>
- XGBoost explainability with SHAP*. (n.d.). Kaggle. Retrieved November 7, 2023, from <https://www.kaggle.com/code/bryanb/xgboost-explainability-with-shap>