NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

SC4000 / CE4041 / CZ4041 Assignment

AY 2023-2024

Kaggle Competition: ***Zillow Prize: Zillow's Home Value Prediction (Zestimate)***

Group members:

Name	Matric No.
Au Yew Rong Roydon	U2021424J
Chen Kang Ming	U2021023H
Dion Toh Siyong	U2021674D
Lim Ziyi Janesse	U2022165D
Tan Yue Jun	U2021253G

Table of Contents

1. Executive Summary

1.1. Problem Statement

From the problems stipulated for selection, we (our team) have decided to take on the Zillow Home Value Prediction model challenge. In this challenge, we were tasked to improve Zillow's Zestimate, a model used to predict the value of properties located in the United States. This model uses a diverse range of metrics, ranging from the presence or absence of facilities and structures in a house (fireplace, swimming pool), to the location (latitude, longitude) and even the status of the property (tax delinquent, etc.).

In this scenario, we were provided training data from real estate properties in 3 counties in the United States, comprising over 2 million rows. Using this data, we were instructed to predict the logarithmic error between the estimated sales price (the one generated by our model) and the actual sales price of a property. The metric used to benchmark our results was **Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).**

There were a total of **3772 participants** for this competition when the challenge was ongoing between 2017 and 2019.

1.2. Division of Work

Throughout this project, we worked as a 5-member team. All of us contributed equally in all aspects of the project, including the report, video, codebase and ideation.

1.3. Workflow of Project

We began the project by understanding and preprocessing the data. After that was done, we proceeded to train the model, with iterative fine-tuning and adjustments made to improve the performance. Finally, our results and conclusions were tabulated and drawn up.

2. Preprocessing Of Data

2.1. Exploratory Analysis

We first began preprocessing by loading the relevant datasets, 'properties_2016', 'properties_2017', 'train_2016' and 'train_2017'. These datasets contain all the metrics and prices of properties tracked in years 2016 and 2017 in the United States. For convenient analysis and training, firstly, we combined the 'properties' table with the 'train' tables by joining on the common column 'parcelid' (primary key). We then merged the 2016 and 2017 tables together to obtain a unified dataset containing **the output variable to be predicted** across both years (Appendix 1A).

Initially, we observed that the dataset size was huge (more than 2 million rows). To better reduce its computational requirements for training locally and on cloud-hosted services (Google Colab), we hypothesized that we should **(1) remove/update columns with a large number of missing values and (2) perform manipulation of data types** for efficient storage space utilization.

2.1.1. Removal and Updating Of Missing Values

We realised there were many columns that had more than 99% of empty rows. These columns might not be useful for training and hence, we theorized to either impute values to fill the empty rows, or drop these columns (Appendix 1C) to see improvement. We first analyzed the different columns and compiled an interpretation of what each column represented (Appendix 1B) to decide if imputation or dropping was the method we should use. Finally, columns that fulfill our criteria were operated on below:

2.1.1.1. Imputation of 0 for 'poolcnt', 'poolsizesum'

We found that for 'poolcnt', the only values were 1. This would mean the null values were most likely 0 representing that there are no pools within the property. To verify this further, we checked for the rows where 'poolcnt' were null which corresponded to a null value for the other pool related columns, such as 'poolsizesum' (Appendix 1D), which ascertained our initial claim. Therefore, '0' was imputed to the null values in these columns.

2.1.1.2. Imputation of 0 for 'id'-related Columns:

Since 'id'-related columns should be unique for each row, imputing the mean values of these columns is not the right approach. We opted not to remove rows with null values because despite those specific cells being empty, the remaining columns contained valuable data. Eliminating these rows might lead to a significant loss of information (Appendix 1E). Checking for the minimum values for these columns, we found that all of them were greater than 0. Hence, during the preprocessing, we imputed '0' to fill the 'id'-related columns. Subsequently, if these columns did not value-add to the training processes, we will drop them.

2.1.1.3. Imputation with Integer Values for 'cnt' and Numerical Columns

For columns containing the substring 'cnt,' except for 'bathroomcnt,' we observed that the values in these columns, despite being float-typed, are whole numbers (refer to Appendix 1G). Moreover, given the presence of numerous outliers in these features, we chose to impute the data using the median. It's worth noting that in our dataset, the 'cnt' columns shared identical mode and median values.

For the rest of the numerical columns, we proceeded with exploratory analysis to decide on whether to impute these missing values with the mean or median of the column values. It turns out that these columns had many outliers, and therefore were also imputed with the median of their column values (Appendix 1H).

2.1.1.4. Imputation/Dropping of Values/Rows for Object/Categorical Columns

From the categorical columns with missing data (Appendix 1I), we identified 4 columns of interest:

a) **'hashottuborspa'** - the only value in the dataset was 'true'. Hence, the null values were most likely representing 'false', which we imputed subsequently.
b) **'propertycountylandusecode'** - only 1 null value, therefore row was dropped.
c) **'propertyzoningdesc'** - null values were imputed with our own category 'unknown'.
d) **'taxdelinquencyflag'** - the only value was 'Y'. Hence, the null values were most likely to be the value of **'N'**, which we imputed subsequently.

2.1.1.5. Dropping of Rows with Null Values for Columns 'propertylandusetypeid', 'regionidcounty', 'rawcensustractandblock', 'censustractandblock'

For these columns, we checked that for the rows with null values, many of its other columns are also null-valued. Furthermore, the null-valued rows were only a small percentage of the whole dataset. Hence, we decided to drop these rows (Appendix 1F).

2.1.2. Changing of Data Types

Analysis of the relevant numerical columns was also done and we checked for possibilities in changing data types. This step was only done after ensuring that there were no null/empty values remaining in the dataset. We checked for the impact of converting it to an integer via the magnitude of loss in precision. If the alteration resulted in a loss of less than 0.01, we proceeded with the conversion. Consequently, in suitable cases, we converted the relevant data to integer types, significantly decreasing memory usage from around 1 GB to 34.9 MB.

2.2. Exploration Of Prediction Variable

Basic exploration of the prediction variables was then performed. Firstly, we began our exploration with the 'logerror' variable (the Y variable). In this case, a positive 'logerror' represents an overestimation of sales price, and a negative would mean an underestimation. The values also seemed to follow a normal distribution, with small outliers at the extreme values as shown in Figure 1. With this observation, we decided to drop the outliers that were 2.5 standard deviations away from the mean, to help the model(s) better generalize to the dataset and not be influenced by the noise from these outliers. This accounted for almost 2% of the training data.

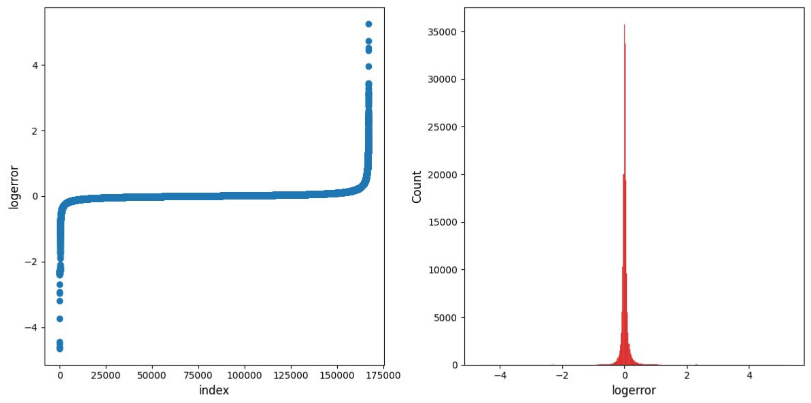


Figure 1: plotting of 'logerror'

We also plotted the mean absolute 'logerror' against time and found that 'logerror' seemed to have decreased for the first 6 months in 2016 (Appendix 1J). However, for the month of July, it started increasing again (poor predictions). This trend is repeated for the year 2017 as well. Upon further exploration, we found that this trend is due to the lack of data - post-June, in both years there was a notable decrease in the percentage of available data, directly impacting the 'logerror'. We went on to explore the potential of feature engineering as a means to address the scarcity of available data in our dataset.

2.3. Feature Engineering And Dropping Of Columns

2.3.1. Adding Of New Features (Feature Engineering)

Feature engineering facilitates increased interpretability and amount of information in our dataset to be used in training. We based the process on the assumption that sales price is correlated to 'logerror' since 'logerror' is calculated from sales price. Hence, we concurred that features that affect sales price would in turn affect the 'logerror'. Table 2 shows the added features, along with elaboration and reasoning behind them.

Added Feature	Explanation Of Feature	Reason For Adding
'percentage_error_of_living_area_12'	Error made between **estimated** living area and **actual** living area after building has completed	A percentage > 100% would mean calculated living area > actual living area. Hence this would imply that the house is smaller than expected and sales price could be affected negatively.
'percentage_error_of_living_area_15'		
'calculated_living_area_prop'	Proportion of living area to lot size (Calculated)	This generates a ratio called building footprint or building coverage ratio. A higher proportion may be perceived as greater value since buyers would desire a property with a larger proportion of living space, which would impact sales price positively.
'living_area_prop'	Actual proportion of living area to lot size	
'extra_space'	Amount of extra living space due to error between calculated and actual	A larger than expected living area would affect sales price positively
'extra_lot_space'	Amount of extra lot space due to error between calculated and actual	A larger than expected lot space would affect sales price positively.
'total_rooms'	Total number of bathrooms and bedrooms.	Having more core/primary rooms in a property would usually lead to a higher sales price.
'value_prop'	Ratio of built structure value to land area	The property-to-land value ratio reflects how efficiently the land is utilized, which in turn could affect sales price. A higher ratio suggests a significant portion of the property value is attributed to improvements, possibly indicating undeveloped land. Land development also leads to increased sales prices.

‘location’ (l)	Combing latitude (lat) and longitude ($long$) through addition $l = lat + long$	To aim to capture specific patterns or relationships between latitude and longitude.
‘location_2’ (l_2)	Combing lat and long through subtraction $l_2 = lat - long$	
‘location_3’ (l_3)	$l_3 = lat + 0.5 \times long$	
‘location_4’ (l_4)	$l_4 = lat - 0.5 \times long$	
‘location_5” (l_5)	$l_5 = lat \times long$	
‘tax_value_ratio’	Ratio of assessed tax over actual tax of property	A higher ratio suggests assessed tax > actual tax which would impact sales price. (Possibility of an increase of decrease in taxation)
‘tax_value_score’	Total tax score	A multiplication of assessed tax and actual tax is performed to get the total tax value of property. A higher tax value would mean a possible higher sales price.
‘zip_num’	Number of properties in the zip code area	More properties in the area could drive sales prices up or down depending on customer expectations and the size of the area. Over-congestion of an area could affect sales price negatively.
‘city_num’	Number of properties in the city	
‘county_num’	Number of properties in the county	
‘neighbourhood_num’	Number of properties in the neighbourhood	
‘avg_garage_size’	Average size of garage	A larger garage size could lead to a higher sales price
‘property_tax_per_sqft’	Gives the tax per square feet of the property	This could be useful since the property size and tax value would affect this feature, providing a different perspective on taxation of property.
‘avg_area_per_room’	Average room size	A larger room size could lead to a higher sales price.
‘derived_avg_area_per_room’	Average size of bathroom and bedroom	A larger bathroom/bedroom size could lead to a higher sales price.
‘house_age’	The age of the property from the year it was built till today	Instead of the original feature ‘year_built’ provided by the kaggle dataset, more useful information could be obtained by finding how old the property is.

Table 2: Features added via feature engineering

2.3.2. Dropping Of Features

We dropped existing columns that were highly correlated with each other to remove redundancy and reduce overfitting of our data. The corresponding correlation plots of the groups of similar features can be found in Appendix 1K. Note that the correlation analysis was done before any dropping or imputation of features. Table 3 shows the features dropped and the reasons for doing so.

Dropped Feature	**Reason For Dropping**
‘bathroomcnt’	Since ‘bathroomcnt’, ‘calculatedbathnbr’ and ‘fullbathcnt’ are highly correlated, and ‘bathroomcnt’ has more null values, we decided to keep ‘fullbathcnt’ and drop the rest.
‘calculatedbathnbr’	***See ‘bathroomcnt’ reasoning.***
‘taxvaluedollarcnt’	Since ‘structuretaxvaluedollarcnt’, ‘taxvaluedollarcnt’, ‘landtaxvaluedollarcnt’ and ‘taxamount’ are closely correlated with each other, we decided to keep ‘taxamount’ and drop the others. This decision is based on the high multicollinearity the dropped features exhibited with other features in the dataset, as seen in the Multicollinearity Analysis using VIF (Appendix 1L), which is undesirable.
‘structuretaxvaluedollarcnt’	***See ‘taxvaluedollarcnt’ reasoning.***
‘landtaxvaluedollarcnt’	***See ‘taxvaluedollarcnt’ reasoning.***
‘finishedsquarefeet12’	Since ‘calculatedfinishedsquarefeet’ has perfect correlation with ‘finishedsquarefeet6’, ‘finishedsquarefeet12’, ‘finishedsquarefeet13’, and ‘finishedsquarefeet15’, we decided to keep ‘calculatedfinishedsquarefeet’ and drop the rest. This decision is reinforced by its minimal null values compared to the other features with much more null values.
‘finishedsquarefeet15’	***See ‘finishedsquarefeet12’ reasoning.***
‘finishedfloor1squarefeet’	Although ‘finishedfloor1squarefeet’ and ‘finishedsquarefeet50’ exhibit a correlation of approximately 0.7 with ‘calculatedfinishedsquarefeet’, we opted to retain one of the features, considering their near-perfect correlation with each other, which may reveal some additional information. We chose to keep ‘finishedsquarefeet50’ due to its slightly lower correlation with ‘calculatedfinishedsquarefeet’, while deciding to drop ‘finishedfloor1squarefeet’.

Table 3: Features removed by dropping columns

2.4. Ablation Study on Features

An ablation study was performed to analyse the effectiveness of each strategy done above.

We conducted the study on missing value imputation, column removal, and the addition of specific engineered features, we found that the imputation and removal of certain columns, as performed previously, did not yield any improvements benchmarking against the CatBoost, LGBM, and XGBoost models. In fact, these models fared better than our attempt at imputation as they are inherently able to automatically handle missing data. That being said, the addition of features (in Section 2.3.1) and dropping of ineffective columns (in Section 2.3.2) helped improve the accuracy of our models tremendously. Therefore moving forward, we have decided to proceed with the dataset containing **engineered features and removed columns**, **excluding user imputation**. Table 4 and 5 illustrates some of our experiments:

Examples Of Study	RMSE		MAE	
	No Imputation	Our Imputation	No Imputation	Our Imputation
CatBoost	0.082977	0.09567	0.0052141	0.0062145
LGBM	0.082652	0.09432	0.0052371	0.006321
XGBoost	0.155032	0.18258	0.067484	0.08452

Table 4: Ablation study on preprocessing done

Examples Of Study	RMSE		MAE	
	Original Features	Added Features	Original Features	Added Features
CatBoost	0.083068	0.082977	0.052240	0.0052141
LGBM	0.082935	0.082652	0.052531	0.0052371
XGBoost	0.155394	0.155032	0.067823	0.067484

Table 5: Ablation study on preprocessing done (pt. 2)

2.4.1. Feature Importance

As shown in Figure 2 below, most of our features rank within the top 10 in terms of feature importance. Some of these features with high importance include ‘tax_value_ratio’, ‘property_tax_per_sqft’, ‘house_age’, ‘value_prop’ and ‘derived_avg_area_per_room’.

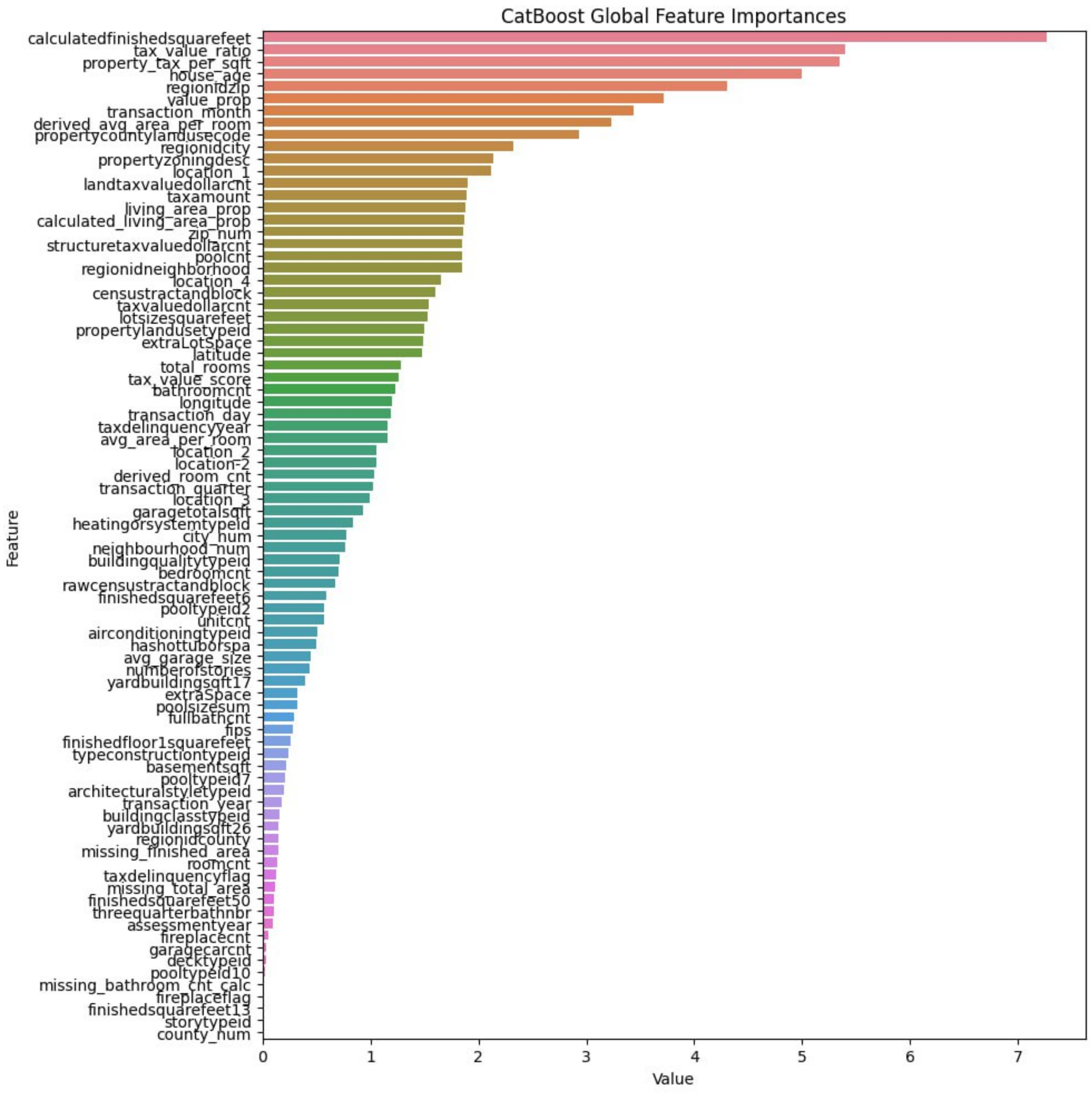


Figure 2: Feature ranking for variables, including feature-engineered variables

3. Machine Learning Models Used

Gradient boosting has been known to be one of the most popular and effective machine learning algorithms for **tabular datasets.** Gradient boosting models are a type of ensemble machine learning method that builds multiple decision trees sequentially, with each tree correcting the errors of its predecessors. It is highly useful in finding any nonlinear relationship between data points and has great versatility, dealing with missing values, outliers and even high cardinality categorical values on features without **any special treatment.** An example would be their built-in capability to automatically handle missing data, as outlined in Section 2.4.

We employed **CatBoost**, **LightGBM (LGBM)**, and **XGBoost** for the remainder of our project. These are popular models used in multiple industries and have been proved effective on tabular data. In particular, we opted for CatBoost due to its excellent handling of categorical data without the need for extensive preprocessing. LGBM is quick and efficient, which could prove well in model stacking to boost performance of other models in tandem. Lastly, XGBoost gives us the option to sensitively fine-tune the data, with a wide range of hyperparameters to experiment with.

3.1. Individual Model Analysis

We conducted a comparative analysis of the aforementioned machine learning models, and noted that each of the models exhibited distinct strengths and weaknesses, which is highlighted by the difference in their performances on the same dataset.

The performance of each of the trained models was evaluated by their **Public Score** on the Kaggle submission portal along with the **RMSE** and **MAE scores**. Throughout this process, we attempted to fine-tune the hyperparameters for each model to optimize their performance in hopes of getting a better public score. The final optimum results for each of the models are shown in Table 4.

Model	**RMSE**	**MAE**	**Public Score**	**Top Ranking (%)**
CatBoost	0.082977	0.052141	**0.06426**	**12.911**
LGBM	0.082652	0.052371	0.06435	17.736
XGBoost	0.155033	0.067484	0.06630	90.376

Table 4: Results of individual models

Based on Table 4, **CatBoost** emerged as the most promising model based on its superior performance in terms of the Public Score metric. Moving forward, we decided to **drop XGBoost** due to its underwhelming performance as compared to the other two models.

3.2. Model Stacking

Following the individual model evaluation, we ventured into the utilization of model stacking, a technique aimed at leveraging the diverse strengths of each model to achieve superior predictive performance.

Our stacking approach included exploring different combinations of CatBoost and LGBM. We assigned unique weights to the outputs produced by each model, with the goal of optimizing their combined predictive abilities. The results of the various model stacking are presented in Table 5 below.

Weightage (%)		**Public Score**	**Top Ranking (%)**
CatBoost	**LGBM**		
10	90	0.06430	14.422
20	80	0.06426	12.911
30	70	0.06420	10.764
40	60	0.06420	10.764
50	50	0.06419	8.192
60	40	**0.06418**	**7.635**
70	30	0.06419	8.192
80	20	0.06420	10.764
90	10	0.06423	11.691

Table 5: Results of model stacking

From the results seen above, on average, the public score of a stacked model is better in contrast to just having individual models of Catboost and LGBM. Model stacking did yield considerable improvements, the best combination putting us **within the 92nd percentile in relation to the public score**. This can be due to the synergistic effect of the models being able to capture different aspects of the data, allowing the stacked model to learn and generalize better by combining the strengths of CatBoost and LGBM - at the same time mitigating their individual weaknesses.

3.3. Data Drifting

The possibility of model degradation was also explored. Model degradation is a common issue faced when deploying machine learning models in the real world. New data points could exhibit a different pattern from older data points due to external factors such as changes in government policy or market sentiments. In such situations, the distribution of the new data points could differ from the original data distribution which the models were trained on. We decided to investigate the extent of model degradation and explore possibilities for further improvements (if needed). The impact of degradation is important since the majority of our 2017 data, especially those of the later months were missing. This would mean the majority of 'logerror' predictions for 2017 rows would have to rely on 2016 data.

Using the Alibi Detect library, we applied the TabularDrift function to detect which features have drifted in the 2017 dataset. We first split the data into two data frames containing 2016 and 2017 data respectively. We then proceeded to take a sample of 1000 data points from each data frame to analyze possible drift in certain features.

The following features have drifted between 2016 and 2017:

```
transactiondate -- Drift? Yes! -- Chi2 2000.000 -- p-value 0.000
buildingqualitytypeid -- Drift? Yes! -- Chi2 700.341 -- p-value 0.000
assessmentyear -- Drift? Yes! -- K-S 1.000 -- p-value 0.000
year_month -- Drift? Yes! -- K-S 1.000 -- p-value 0.000
regionidzip -- Drift? Yes! -- K-S 0.064 -- p-value 0.032
tax_value_ratio -- Drift? Yes! -- K-S 0.094 -- p-value 0.000
```

P-values lower than **0.05** suggest that there have been significant changes in the distribution of these features in 2017 compared to 2016 hence impacting the model since the patterns learned have become less useful.

The above highlights that a covariate shift or concept drift has occurred. Covariate shift refers to changes in the distribution of the input features (X) without changes in the conditional distribution of the target variable (Y|X). Concept drift happens when the underlying concept or the relationship between the features and the target variable changes over time. In the context of housing, the model's assumptions about the relationship between the features and price may no longer hold due to new changes in the housing markets such as housing market dynamics, economic conditions etc. This would cause the model to perform poorly since it has become outdated.

A possible improvement could be to remove some of these columns, as shown in Table 6.

Model	RMSE	MAE	Public Score	Top Ranking (%)
LGBM	0.08281	0.05241	0.06434	15.986
CatBoost	0.08300	0.05222	0.06426	12.911
CatBoost + LGBM	-	-	**0.06420**	**10.764**

Table 6: Results after dropping drifted features

While there was only marginal improvement observed for the LGBM model improving from a score of 0.06435 to 0.06434, both the CatBoost model and the stacked model experienced a decline in the public score.

The mixed impact on model performance suggests that our approach to feature selection and handling data drift may have influenced individual models differently. Further investigations and adjustments may be warranted to refine our feature selection strategy and maintain the stability and generalizability of our models.

Due to limited time, we only proposed the dropping of columns as a solution. However, more solutions specific to targeting covariate shift and concept drift could be explored in the future to possibly improve the accuracy of the model.

3.4. Meta Learner

To further optimize predictive performance, we also decided to streamline our modeling approach by exclusively focusing on the **stacked LGBM and CatBoost** for prediction. By narrowing our focus, we sought to intensify our efforts in fine-tuning the parameters of LGBM and CatBoost for enhanced predictive accuracy.

We utilised another LGBM to train on the predictions generated by both the feature-trained LGBM and CatBoost models. However, our results indicated that this particular configuration did not give us satisfactory improvements in predictive performance.

RMSE	**MAE**	**Public Score**	**Top Ranking (%)**
0.082721	0.05222	0.06429	14.157

Table 7: Results for Meta Learner

This outcome underscores the intricate nature of model interactions and the nuanced dependencies within the dataset. This experience highlights the importance of continuous experimentation and adjustment in the pursuit of refining model architectures and training strategies. However, owing to time constraints, we did not sufficiently explore other methods that could potentially boost the performance further. Future iterations could involve further exploration of model combinations, parameter tuning, or alternative training methodologies to unlock the full potential of LGBM and CatBoost for this specific predictive modeling task.

4. Results

Table 8 illustrates our final results from the analysis above, and Figure 3 shows receipts of our result.

Category	**Model(s)**	**Public Score**	**Top Ranking (%)**
Individual	**CatBoost**	0.06426	12.911
Stacked	**CatBoost + LGBM**	**0.06418**	**7.635**
MetaLearner	**LGBM (using CatBoost and LGBM predictions)**	0.06429	14.157
Stacked w MetaLearner	**CatBoost + LGBM + MetaLearner**	**0.06418**	**7.635**

Table 8: Comparisons between all the top performers in each category

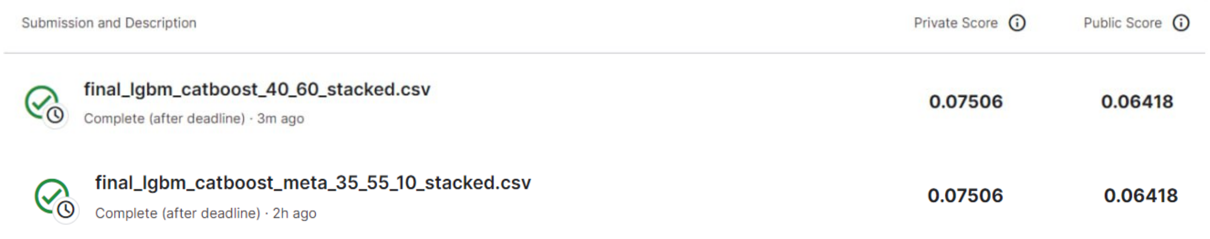

Submission and Description	Private Score	Public Score
final_lgbm_catboost_40_60_stacked.csv Complete (after deadline) · 3m ago	0.07506	0.06418
final_lgbm_catboost_meta_35_55_10_stacked.csv Complete (after deadline) · 2h ago	0.07506	0.06418

Figure 3: Best Kaggle competition scores achieved

5. Challenges

5.1. Data Limitations

The data provided, while plenty, was difficult to work with due to the presence of many missing values, coupled with multiple outliers and wrong data types. The majority of data from the later months in 2017 were also missing, which further exacerbated the prediction accuracy of data points of a later timestamp. To overcome these limitations, we had to employ multiple strategies, like feature engineering and data drift techniques, to better understand how the information provided by these features changes with time.

It was also difficult to extend our dataset by collecting data online, since each property had an ID tied to it, which was only provided by Kaggle.

5.2. Model Exploration

It was also a challenge finding the right type of model to work with. Prior to the decision of using gradient boosting models, various experimentations with other models were also done. This includes more complex models such as neural network models, where we initially attempted to build a simple feed-forward network. However, even after training the model on our preprocessed dataset, the accuracy was poor and failed to obtain a satisfactory result. Even after fine-tuning hyperparameters such as dropout, learning rate, threshold, and patience, and adding more layers, this remained the case. In conjunction with the difficulty in the set-up and workflow of the neural networks, we eventually decided to forego these models due to time constraints. We overcame this by redirecting our resources to other popular models (like gradient boosting).

5.3. Unpredictability Of Stacking

Multiple manual experiments had to be held for different combinations of percentages of stacking, as well as combinations of features used for training of different models. The impact of features on different models are unique and multiple experiments had to be conducted to find the best set of features. To do so, an exhaustive search had to be done trying out different experiments manually, which was time-consuming.

6. Conclusion

To summarise, we have undertaken the Zillow Zestimate challenge for this project. Preprocessing of data was done, eventually settling on feature engineering and column dropping as our desired approach. Different gradient boosting methods were implemented in our training phase, in which eventually 2 models **(stacked and stacked w/ MetaLearner)** stood out. Both gave us a public score of **0.06418**, which ranks firmly within the **92nd percentile** of all submissions in Kaggle.

In undertaking this machine learning project, we have gained insightful knowledge on how concepts that we learnt in lectures and our modules, as well as advancements in machine learning techniques help us get accurate predictions on real-world applications, such as housing prices (the one we worked on), and also extending to other domains like mass trend predictions, natural language generation, and even intelligent agents.

Appendix:

1A. Explanation of Tables/'.csv'

Table	Description
properties_2016	All the properties with home features for the year of 2016
properties_2017	All the properties with home features for the year of 2017
train_2016	The training set with transactions from 1/1/2016 to 31/12/2016. It contains the output variable logerror that we are supposed to predict.
train_2017	The training set with transactions from 1/1/2017 to 15/09/2017. It contains the output variable logerror that we are supposed to predict.

1B. Explanation of Features

Feature	Description
parcelid	Unique Identifier
logerror	Output variable to be predicted
transactiondate	Date of purchase
airconditioningtypeid	Air Condition Id code indicating the type of air conditioning system present in the property
bathroomcnt	Number Of Bathrooms
bedroomcnt	Number of Bedrooms
buildingqualitytypeid	An Id representing the overall quality of the building
calculatedbathnbr	The calculated number of bathrooms in the property
finishedfloor1squarefeet	The square feet of the finished first floor
calculatedfinishedsquarefeet	The square feet of the finished property
finishedsquarefeet12	The finished living area of the property
finishedsquarefeet15	The total finished living area of the property
finishedsquarefeet50	Unknown area of the property
fips	Federal Information Processing Standards Code that identifies the county where the property is located
fireplacecnt	The number of fireplaces.
fullbathcnt	The number of bathrooms.
garagecarcnt	The number of garages.
garagetotalsqft	The total square foot of the garage
hashottuborspa	Indicates whether the property has a hot tub or a spa
heatingorsystemtypeid	An id indicating the type of heating system in the property
latitude	The latitude of the property's location
longitude	The longitude of the property's location
lotsizesquarefeet	The size of the lot in square feet
poolcnt	The number of pools on the property
poolsizesum	The total size of the pools on the property
pooltypeid2	An id indicating the type of the pool on the property
pooltypeid7	An id indicating the type of the pool on the property
propertycountylandusecode	An id indicated the zone of the property (County)
propertylandusetypeid	An id representing the type of land use for the property
propertyzoningdesc	A description of the zoning for the property
rawcensustractandblock	Raw census tract and block code that identifies the area where the property is located
regionidcity	An id that identifies the city where the property is located
regionidcounty	An id that identifies the county where the property is located
regionidneighborhood	An id that identifies the neighbourhood where the property is located

regionidzip	An id that identifies the postal zone where the property is located
roomcnt	The number of rooms in the property
threequarterbathnbr	The number of three-quarter bathrooms in the property
unitcnt	The number of units that the property has
yardbuildingsqft17	The square foot of patio space
yearbuilt	The year when the property was built
numberofstories	The number of stories in the property
structuretaxvaluedollarcnt	The assessed tax value of the property's structure
taxvaluedollarcnt	The total assessed tax value of the property
assessmentyear	The year the property was tax / assessed
landtaxvaluedollarcnt	The assessed tax value of the property's land
taxamount	The total property tax
taxdelinquencyflag	Indicates if there is tax delinquency for the property
taxdelinquencyyear	The year when the property's tax became delinquent
censustractandblock	The census tract and block code that identifies the area where the property is located

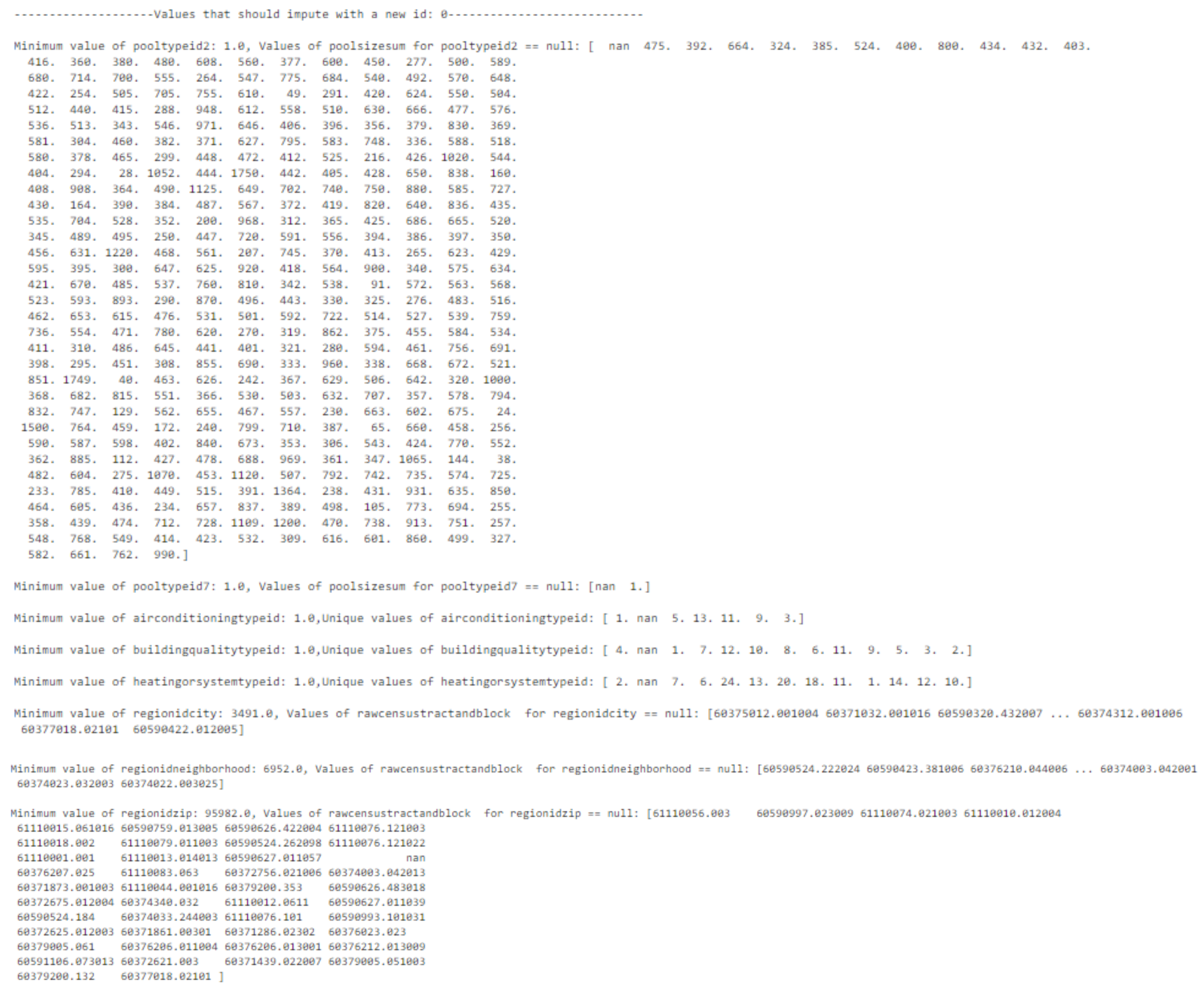
1C. Features with 99% of Missing Data

```
Columns with missing values > 166209.12: Index(['architecturalstyletypeid', 'basementsqft', 'buil
dingclasstypeid',
       'decktypeid', 'finishedsquarefeet13', 'finishedsquarefeet6',
       'pooltypeid10', 'storytypeid', 'typeconstructiontypeid',
       'yardbuildingsqft26', 'fireplaceflag'],
      dtype='object')
```

1D. Exploring 'poolcnt', 'poolsizesum' for Imputation

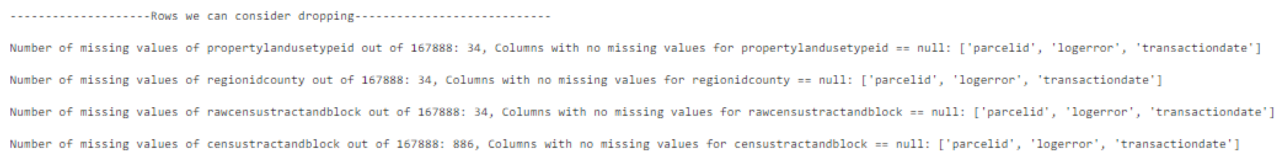
```
--------------------Values that should impute with 0----------------------------

Minimum value of poolsizesum: 24.0, Values of poolsizesum, pooltypeid2, pooltypeid7, for poolcnt == null: [nan], [nan], [nan]

Number of missing values of poolcnt out of 167888: 133813, Columns with no missing values for poolcnt == null: ['parcelid', 'logerror', 'transactiondate']
```

1E. Exploring Imputation for IDs



```
 -------------------Values that should impute with a new id: 0---------------------------

 Minimum value of pooltypeid2: 1.0, Values of poolsizesum for pooltypeid2 == null: [  nan  475.  392.  664.  324.  385.  524.  400.  800.  434.  432.  403.
   416.  360.  380.  480.  608.  560.  377.  600.  450.  277.  500.  589.
   680.  714.  700.  555.  264.  547.  775.  684.  540.  492.  570.  648.
   422.  254.  505.  705.  755.  610.   49.  291.  420.  624.  550.  504.
   512.  440.  415.  288.  948.  612.  558.  510.  630.  666.  477.  576.
   536.  513.  343.  546.  971.  646.  406.  396.  356.  379.  830.  369.
   581.  304.  460.  382.  371.  627.  795.  583.  748.  336.  588.  518.
   580.  378.  465.  299.  448.  472.  412.  525.  216.  426. 1020.  544.
   404.  294.   28. 1052.  444. 1750.  442.  405.  428.  650.  838.  160.
   408.  908.  364.  490. 1125.  649.  702.  740.  750.  880.  585.  727.
   430.  164.  390.  384.  487.  567.  372.  419.  820.  640.  836.  435.
   535.  704.  528.  352.  200.  968.  312.  365.  425.  686.  665.  520.
   345.  489.  495.  250.  447.  720.  591.  556.  394.  386.  397.  350.
   456.  631. 1220.  468.  561.  207.  745.  370.  413.  265.  623.  429.
   595.  395.  300.  647.  625.  920.  418.  564.  900.  340.  575.  634.
   421.  670.  485.  537.  760.  810.  342.  538.   91.  572.  563.  568.
   523.  593.  893.  290.  870.  496.  443.  330.  325.  276.  483.  516.
   462.  653.  615.  476.  531.  501.  592.  722.  514.  527.  539.  759.
   736.  554.  471.  780.  620.  270.  319.  862.  375.  455.  584.  534.
   411.  310.  486.  645.  441.  401.  321.  280.  594.  461.  756.  691.
   398.  295.  451.  308.  855.  690.  333.  960.  338.  668.  672.  521.
   851. 1749.   40.  463.  626.  242.  367.  629.  506.  642.  320. 1000.
   368.  682.  815.  551.  366.  530.  503.  632.  707.  357.  578.  794.
   832.  747.  129.  562.  655.  467.  557.  230.  663.  602.  675.   24.
  1500.  764.  459.  172.  240.  799.  710.  387.   65.  660.  458.  256.
   590.  587.  598.  402.  840.  673.  353.  306.  543.  424.  770.  552.
   362.  885.  112.  427.  478.  688.  969.  361.  347. 1065.  144.   38.
   482.  604.  275. 1070.  453. 1120.  507.  792.  742.  735.  574.  725.
   233.  785.  410.  449.  515.  391. 1364.  238.  431.  931.  635.  850.
   464.  605.  436.  234.  657.  837.  389.  498.  105.  773.  694.  255.
   358.  439.  474.  712.  728. 1109. 1200.  470.  738.  913.  751.  257.
   548.  768.  549.  414.  423.  532.  309.  616.  601.  860.  499.  327.
   582.  661.  762.  990.]

 Minimum value of pooltypeid7: 1.0, Values of poolsizesum for pooltypeid7 == null: [nan  1.]

 Minimum value of airconditioningtypeid: 1.0,Unique values of airconditioningtypeid: [ 1. nan  5. 13. 11.  9.  3.]

 Minimum value of buildingqualitytypeid: 1.0,Unique values of buildingqualitytypeid: [ 4. nan  1.  7. 12. 10.  8.  6. 11.  9.  5.  3.  2.]

 Minimum value of heatingorsystemtypeid: 1.0,Unique values of heatingorsystemtypeid: [ 2. nan  7.  6. 24. 13. 20. 18. 11.  1. 14. 12. 10.]

 Minimum value of regionidcity: 3491.0, Values of rawcensustractandblock  for regionidcity == null: [60375012.001004 60371032.001016 60590320.432007 ... 60374312.001006
  60377018.02101  60590422.012005]


Minimum value of regionidneighborhood: 6952.0, Values of rawcensustractandblock  for regionidneighborhood == null: [60590524.222024 60590423.381006 60376210.044006 ... 60374003.042001
 60374023.032003 60374022.003025]

Minimum value of regionidzip: 95982.0, Values of rawcensustractandblock  for regionidzip == null: [61110056.003    60590997.023009 61110074.021003 61110010.012004
 61110015.061016 60590759.013005 60590626.422004 61110076.121003
 61110018.002    61110079.011003 60590524.262098 61110076.121022
 61110001.001    61110013.014013 60590627.011057             nan
 60376207.025    61110083.063    60372756.021006 60374003.042013
 60371873.001003 61110044.001016 60379200.353    60590626.483018
 60372675.012004 60374340.032    61110012.0611   60590627.011039
 60590524.184    60374033.244003 61110076.101    60590993.101031
 60372625.012003 60371861.00301  60371286.02302  60376023.023
 60379005.061    60376206.011004 60376206.013001 60376212.013009
 60591106.073013 60372621.003    60371439.022007 60379005.051003
 60379200.132    60377018.02101 ]
```

1F. Dropping of Rows with Null Values

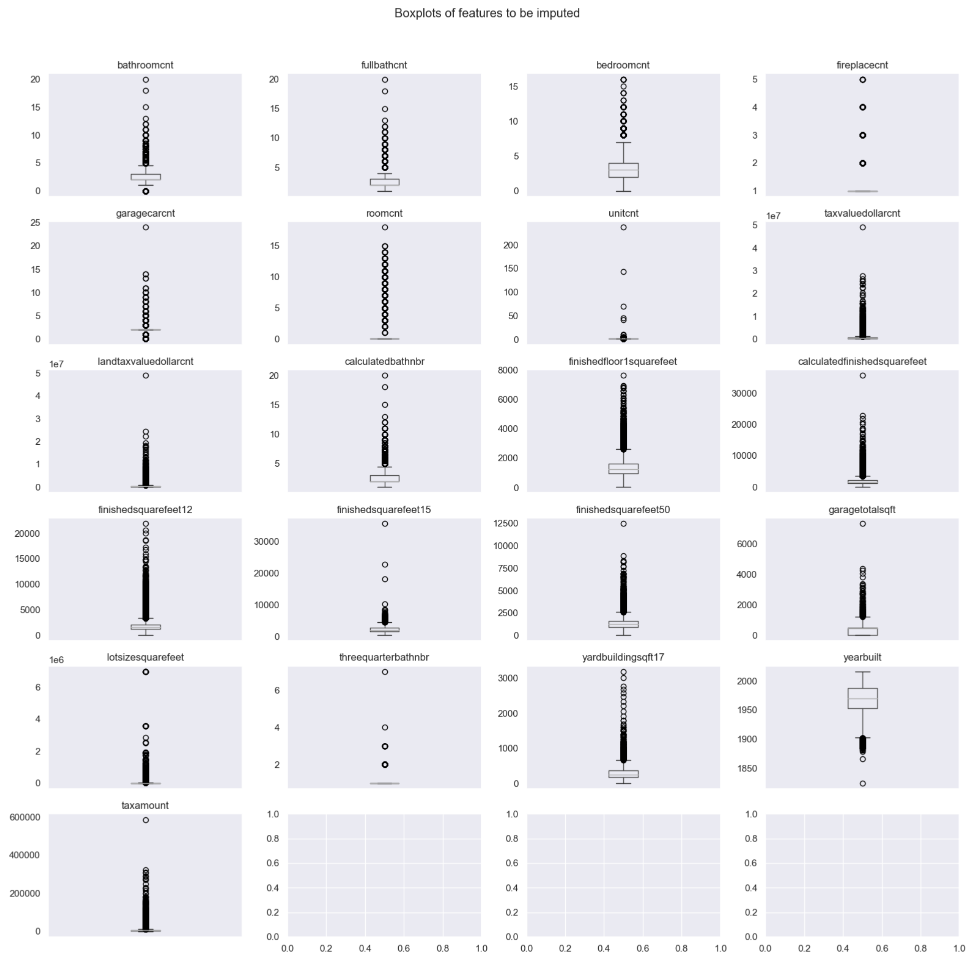


```
-------------------Rows we can consider dropping---------------------------

Number of missing values of propertylandusetypeid out of 167888: 34, Columns with no missing values for propertylandusetypeid == null: ['parcelid', 'logerror', 'transactiondate']

Number of missing values of regionidcounty out of 167888: 34, Columns with no missing values for regionidcounty == null: ['parcelid', 'logerror', 'transactiondate']

Number of missing values of rawcensustractandblock out of 167888: 34, Columns with no missing values for rawcensustractandblock == null: ['parcelid', 'logerror', 'transactiondate']

Number of missing values of censustractandblock out of 167888: 886, Columns with no missing values for censustractandblock == null: ['parcelid', 'logerror', 'transactiondate']
```

1G. Imputation of 'cnt' Columns

```
bathroomcnt unique values: [ 2.   3.5  3.   2.5  4.   1.   5.   5.5  1.5  8.   0.   4.5  9.   7.
  6.  10.   6.5  7.5 12.  11.  20.   8.5 15.   nan 18.  13. ]

bedroomcnt unique values: [ 3.  4.  2.  5.  1.  6.  7.  0. 12. 11.  8.  9. 10. 16. 14. 13. 15. nan]

fireplacecnt unique values: [nan  1.  2.  3.  4.  5.]

fullbathcnt unique values: [ 2.  3.  4.  1.  5.  8. nan  9.  7.  6. 10. 12. 11. 20. 15. 18. 13.]

garagecarcnt unique values: [nan  2.  1.  3.  0.  4.  6.  8.  5.  7. 11. 10. 24.  9. 13. 14.]

roomcnt unique values: [ 0.  8.  6.  5.  7.  4.  3.  9. 12. 11. 10.  2.  1. 13. 15. 14. 18. nan]

unitcnt unique values: [  1.  nan   2.   4.   3.   6. 143.  11.   9.   5.  70.  45.  42. 237.]

taxvaluedollarcnt unique values: [360170. 585529. 119906. ... 354621.  67205.  49546.]

landtaxvaluedollarcnt unique values: [237416. 239071.  57912. ... 214889. 221068. 283704.]
```

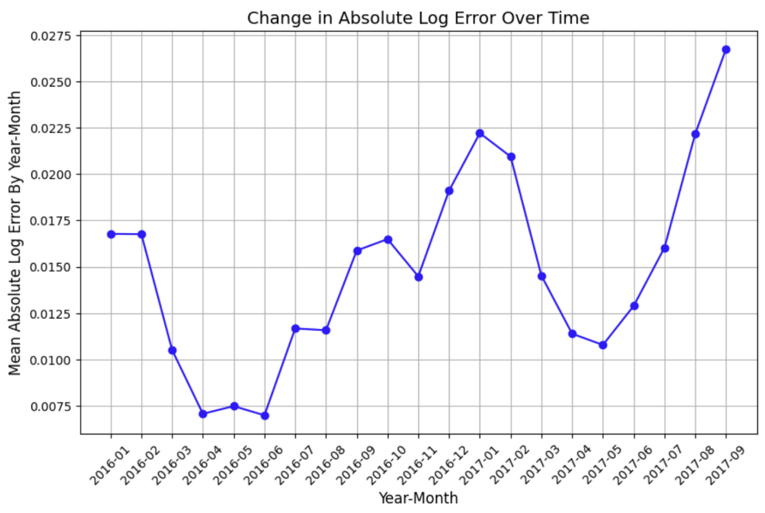
1H. Data Exploration To Decide Impute With Median Or Mean

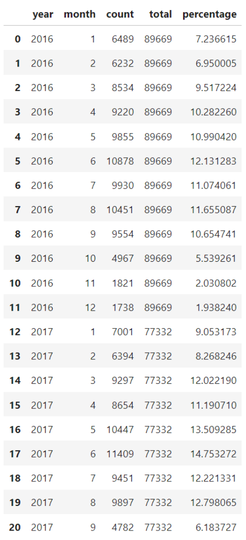


1I. Exploration of Null Values of Object Types

```
-------------------Na values of object type-----------------------
transactiondate                   0
hashottuborspa               163098
propertycountylandusecode         1
propertyzoningdesc            58698
taxdelinquencyflag           162333
dtype: int64
-------------------Na analysis for imputation------------------------
Unique values of hashottuborspa: [nan True]
Unique values of propertyzoningdesc: ['LARS' 'PSR6' 'LAR3' ... 'LCRA 7500*' 'LCRA7000-R' 'BFA15000*']
```

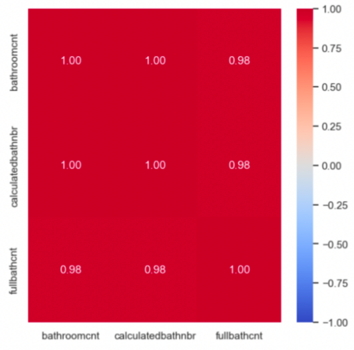
1J. Mean Absolute Log Error Against Time And Percentage Of Data Per Month Per Year

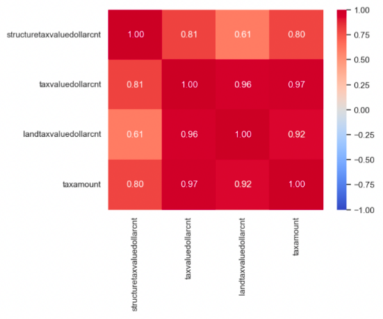


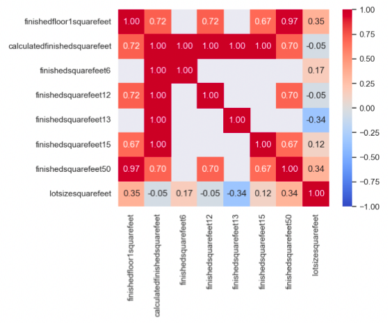


	year	month	count	total	percentage
0	2016	1	6489	89669	7.236615
1	2016	2	6232	89669	6.950005
2	2016	3	8534	89669	9.517224
3	2016	4	9220	89669	10.282260
4	2016	5	9855	89669	10.990420
5	2016	6	10878	89669	12.131283
6	2016	7	9930	89669	11.074061
7	2016	8	10451	89669	11.655087
8	2016	9	9554	89669	10.654741
9	2016	10	4967	89669	5.539261
10	2016	11	1821	89669	2.030802
11	2016	12	1738	89669	1.938240
12	2017	1	7001	77332	9.053173
13	2017	2	6394	77332	8.268246
14	2017	3	9297	77332	12.022190
15	2017	4	8654	77332	11.190710
16	2017	5	10447	77332	13.509285
17	2017	6	11409	77332	14.753272
18	2017	7	9451	77332	12.221331
19	2017	8	9897	77332	12.798065
20	2017	9	4782	77332	6.183727

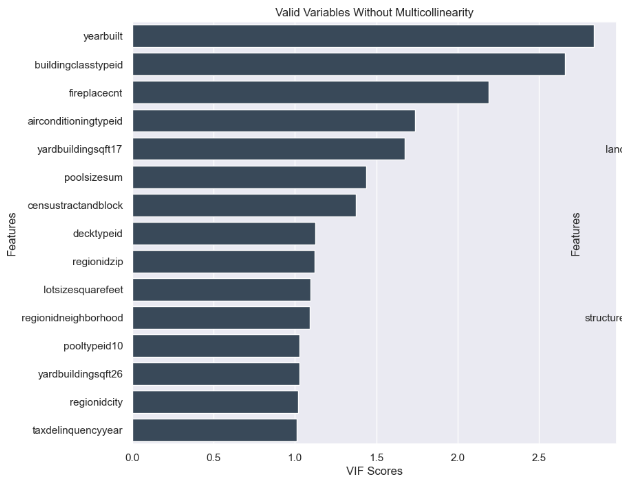
1K. Correlation Heatmap Plots of the Features to be Dropped

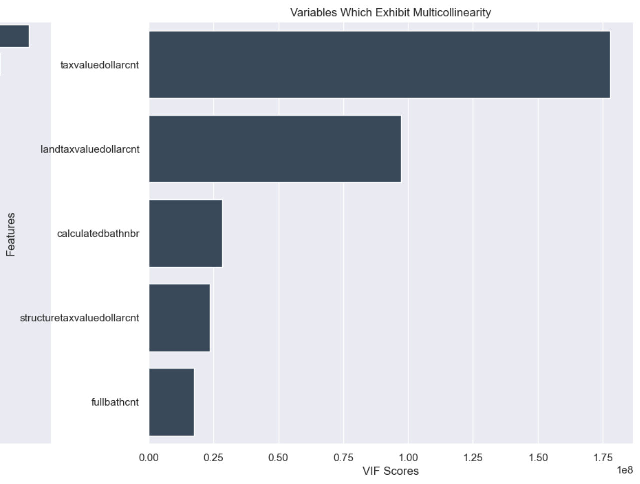






1L. Multicollinearity Analysis using VIF


Valid Variables Without Multicollinearity
Features
VIF Scores
yearbuilt
buildingclasstypeid
fireplacecnt
airconditioningtypeid
yardbuildingsqft17
poolsizesum
censustractandblock
decktypeid
regionidzip
lotsizesquarefeet
regionidneighborhood
pooltypeid10
yardbuildingsqft26
regionidcity
taxdelinquencyyear


Variables Which Exhibit Multicollinearity
Features
VIF Scores
taxvaluedollarcnt
landtaxvaluedollarcnt
calculatedbathnbr
structuretaxvaluedollarcnt
fullbathcnt
1e8