COEN 281 Term Project

Comparison of Regression Models on House Value Prediction

Team 1

Lixiang Zhu

Louise Li

Instructor

Prof. Ming-Hwa Wang

Santa Clara University

March 17, 2020

Preface

The objective of this project is to compare linear regression model and gradient boosting model on the use case of predicting house prices. The ability to predict real time market value of a house is extremely beneficial to home buyers, investors, and realtors. This project utilizes knowledge from data mining and machine learning. It is the term project of the subject Pattern Recognition & Data Mining in winter 2020 under the supervision of Dr. Ming-Hwa Wang.

Acknowledgement

We would like to take this opportunity to express our gratitude to Professor Ming-Hwa Wang. His teaching on the data mining topics has been extremely helpful toward the completion of this project. We would also like to thank him for being encouraging and supportive whenever we encounter a problem during the class. In addition, we would like to thank all the contributors mentioned in the bibliography. Their works were vital for the success of this project. Lastly, we thank all the team members for dedicatedly pushing this project into completion.

Table of Contents

List of Tables & Figures

Abstract

Buying a home is usually one of the largest and most expensive purchases a person can ever make during his or her lifetime. Therefore, being able to monitor the value of these assets becomes extremely important. Current existing solutions on predicting house prices include using domain knowledge as independent features and feeding them into machine learning algorithms as input to get prediction of the house value. However, these models did not evaluate the effects of feature engineering on the performance of the model. In this project, we performed various feature engineering techniques on the dataset before feeding it into our model as input. We used Zillow open source datasets from Kaggle as the training data, and compared the performance of the linear regression models with our tuned gradient boosting model using mean absolute error metric.

Introduction

Objective

The purpose of this project is to compare linear regression models and gradient boosting models on predicting the price of houses. New implementation of the gradient boosting model is developed to increase the accuracy of the prediction. The models are trained with Zillow open source dataset, and the performances of the models are then analyzed and discussed.

Why:

Buying or Owning a house is probably one of the most crucial investment decisions money-wise for a person has to make in his lifetime (especially here in the Bay area). Therefore, being aware of the real time market-value would be very valuable for a homeowner.

What:

Using housing transaction data including housing property such as number of bedrooms, total area, location to forecast the property price in the coming future becomes a classical **regression question.** We will make a comprehensive **compassion** of the performance on machine learning model by using housing property data to train not only traditional **LR** (linear regression) model, but also develop some of the new coming implementation framework based on (**GBDT**) gradient boosting model such as (**XGBOOST**, **LightGBM**).

Why other approach is no good:

Linear regression assumes all the features in a data entry are independent. However, in the house price prediction problem, there exists collinearity, which means some of the independent variables are highly correlated. This will affect the accuracy of the final prediction. Moreover, it is more complicated to deal with categorical variables in linear regression. We can not directly feed them into the model, instead, we need to perform some preprocessing before the training.

Generally, a linear regression model is limited to linear relationships. It only looks at linear relationships between dependent and independent variables and assumes there is a straight-line relationship between them. However, real-life scenarios are more complicated than that. A linear regression model is also sensitive to outliers. Surprising data points may affect the overall performance of the model.

Why our approach is better:

The XGBoost model solves the problems imposed by linear regression models. Gradient boosting model is able to handle both numerical and categorical variables very well. It can optimize on different loss functions and provides several hyperparameter tuning options for machine learning engineers to adjust the model. As an implementation of the gradient boosting decision tree, XGBoost supports in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization, which prevents the model from overfitting. It is much faster than GBM because it utilizes parallel processing. Furthermore, it allows machine learning engineers to use cross-validation at each iteration.

Area of Investigation:

We focused on the full list of real estate properties in three counties (Los Angeles, Orange and Ventura, California) data in 2016. (released by **Zillow**), and compared the performance between traditional linear model and a series of variation of Gradient Boosting Decision tree model (XGBOOST) with metric in MAE (mean absolute error), which is defined by the equation:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\boldsymbol{x}_i - \boldsymbol{x}|$$

Theoretical Bases and Literature Review

Definition of the Problem

In this project, we are trying to compare the performance between linear regression models and ensemble boosting models (GBDT, XGBOOST) on predicting real time house prices.

Theoretical background of the problem

Our project is closely related to regression modeling. Regression analysis is a statistical process for estimating the relationships between a dependent variable, which is the outcome, and one or more independent variables, which is the features in the training set. The most common form of regression analysis is linear regression, which maps a function to the data according to a specific mathematical criterion. It is widely used for prediction and forecasting.

Related research to solve the problem

Cinar's work on making predictions of housing values [1] combines the use of domain knowledge and machine learning techniques. He used a neural network model and a XGBoost model to train the input data and compared their performance using MSE.

Varma's work on house price prediction using machine learning and neural networks [4] proposed various regression techniques to gain a weighted mean for the most accurate results. He also proposed to use real-time neighborhood details using Google maps to get exact real-world valuations.

Chen introduced a document of using xgboost in R [3]. It is an efficient implementation of the gradient boosting framework. The package supports various objective functions such as regression and classification, and is also made to be extensible to the users.

Advantage/disadvantage of those research

These researches mainly focus on the implementation of the machine learning models as well as their performances, but do not analyze much on feature engineering. In our project, we spent more effort on the comparison of different feature engineering techniques on the dataset before feeding them into machine learning algorithms as inputs.

Our solution to solve the problem

Our solution is to implement the linear regression model and gradient boosting model with Zillow open source data as input. We will then evaluate the performance of these models using MAE (mean absolute error). In the equation below, n is the total number of training data entry, x is the predicted house value, and xi is the real value of the house.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} | x_i - x |$$

Fig 1. MAE equation

Where our solution is different from others

Cinar's work on making predictions using boosting techniques on house value [1] combines the use of domain knowledge and machine learning techniques. He used XGBoost model to train the input data. In our project, we developed some new implementation based on the gradient boosting model (GBDT) and compared it with linear regression models.

Current models for predicting housing value did not analyze much on feature engineering. In our solution, we performed evaluation on how various feature engineering techniques, especially dealing with missing values, affects the performance of linear regression models and gradient boosting models.

Why our solution is better

While the traditional linear model (LR,ridge) is simple and very effective for linear data , it has the potential inefficiency in predicting non-linear data and easily gets underfitted. On the other hand, the ensemble boosting model (GBDT, XGBOOST) not only performs very well on all types of data (linear or nonlinear) and gets a very generalized model with very less training time but also very interpretable due to the nature of the tree model.

Hypothesis

a) Null hypothesis(Ho) :

There is **no significant difference** in real estate market value prediction performance (RMSE) between linear model and ensemble boosting learning model. (XGBOOST)

b) Alternative hypothesis (Ha):

There is **a significant difference** in real estate market value prediction performance (RMSE) between linear model and ensemble boosting learning model.(XGBOOST)

Methodology

Input Data

We used the open source data from Zillow Prize: Zillow's Home Value Prediction (Zestimate) provided by Kaggle as the input and fed them into the machine learning models. To download the dataset, we first had to register an Kaggle account and create an API key and install the kaggle command through the Python Package Installer. The dataset was then downloaded with the command line:

```
kaggle competitions download -c zillow-prize-1
```

The data includes 58 features related to the construction of a house, including the number of bedrooms, total living area, zip code of the house, etc. Table 1 shows some of the features and their description of the input data. While these features are numerical values, the data also contains categorical features that require further differentiation. For example, the type of air conditioner is represented by a type id, and its type can be central, chilled water, partial, etc.

Table 1: Input Feature Description

Feature	Description
'airconditioningtypeid'	Type of cooling system present in the home (if any)
'architecturalstyletypeid'	Architectural style of the home (i.e. ranch, colonial, split-level, etc…)
'basementsqft'	Finished living area below or partially below ground level
'bathroomcnt'	Number of bathrooms in home including fractional bathrooms
'bedroomcnt'	Number of bedrooms in home
'buildingqualitytypeid'	Overall assessment of condition of the building from best (lowest) to worst (highest)
'buildingclasstypeid'	The building framing type (steel frame, wood frame, concrete/brick)
'calculatedbathnbr'	Number of bathrooms in home including fractional bathroom
'decktypeid'	Type of deck (if any) present on parcel
'threequarterbathnbr'	Number of 3/4 bathrooms in house (shower + sink + toilet)
'finishedfloor1squarefeet'	Size of the finished living area on the first (entry) floor of the home
'calculatedfinishedsquarefeet'	Calculated total finished living area of the home
'finishedsquarefeet6'	Base unfinished and finished area
'finishedsquarefeet12'	Finished living area
'finishedsquarefeet13'	Perimeter living area
'finishedsquarefeet15'	Total area

How to Solve the Problem

Algorithm Design

1. Data Preprocessing: Data preprocessing, also called feature engineering, is required in order to build a better model with good prediction.

   One Hot Encoding:

   One hot encoding is a process to convert categorical features to a form that is better understandable for machine learning algorithms. For example, the air-conditioner type is represented by a type id between 1 and 4. One Hot Encoding technique will convert the type id into 4 columns. Each column refers to a type. The column of the type of a particular air-conditioner will have a value of 1, and others in the same row will have a value of 0.

   Fill NaN with mean value:

   For each data entry, some feature values may be missing. One way to resolve this is to replace the NaN with the mean of the feature. This may be a good estimation of the missing value.

   Fill NaN with 0:

   Another way to remove the missing value is to replace each value with 0. During the model training process, a value of 0 may have little influence on the output.

Remove Missing Values:

If the proportion of the missing value exceeds 97%, it may be better to just remove the feature altogether, since there is not enough information to determine the overall influences of this feature on the output.

Ordinal Encoding:

Ordinal Encoding transforms non-numerical labels into numerical labels. The numbers are always between 1 and the number of categories that need to be encoded.

Remove Outliers:

We also need to remove abnormal data points such as extreme value of the house due to special reasons.

2. Building Models: For the machine learning part, we will build linear regression models and a gradient boosting model with some of the new coming implementation framework.

Linear Regression Models:

Linear regression models are commonly used on predictive analysis. This type of model examines whether a set of variables perform well on predicting the outcome and explores the relationships between one or more independent variables. In this project, in addition to the regular linear regression model, we also built the linear regression model with L1 regularization, L2 regularization, and a combination of both L1 and L2 regularization.

The linear regression model fits the data using a polynomial function, as shown in the equation below. M is the order of the polynomial, it usually equals to the number of features in the dataset. w = (w1, w2, …, wm) is the coefficient of the polynomial function.

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Fig2. Linear Regression Objective Function

The objective of the linear regression model is to minimize the sum of squared estimates of errors between the data entries in the dataset, and the targets predicted by the linear approximation, which is shown in the equation below. y(x, w) is the predicted value and tn is the true value. N is the total number of training data.

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

Fig3. Linear Regression Loss Function

The image below shows an example of the linear regression. The blue dots represent the data points in the dataset and the green curve represents the function learned by the linear model to map the input to the corresponding output.

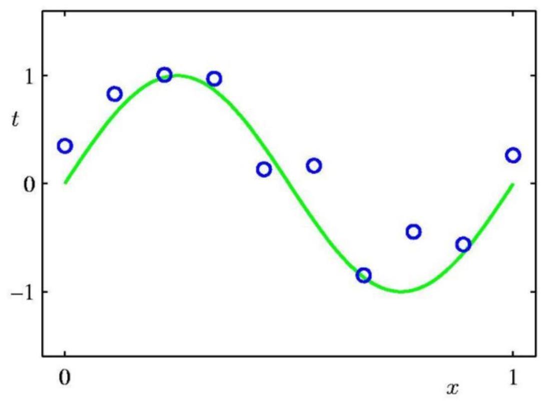


Fig4. Linear Regression Example

However, sometimes the model may not generalize well and causes an overfitting problem. Regularization is a process of introducing additional information in order to prevent overfitting. L1 regularization in linear regression is called Lasso regression. It implements the L1 norm for regularization by adding the L1 regularization term in the loss function. Lambda controls the degree of the generalization. If lambda is too big, it may cause underfitting problems.

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

Fig5. Lasso Regression Loss Function

Similarly, L2 regularization adds L2 term into the loss function, called Rigid Regression, as shown in the equation below.

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

Fig6. Lasso Regression Loss Function

The key difference between these techniques is that Lasso shrinks the less important feature's coefficient to zero, and therefore, removes some features altogether. So, this works well for feature selection in case we have a huge number of features. In the last, we also combined both L1 regularization and L2 regularization in our linear regression model.

Gradient Boosting Models:

Gradient boosting is a technique for classification and regression problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It is an ensemble learner that creates a final model based on a collection of individual models.

It begins by training a decision tree in which each observation is assigned an equal weight. After evaluating the first tree, it identifies the weakness of the tree by using gradients in the loss function. The loss function is a measure of how good the model's coefficients are at fitting the training data. Then, the second tree is grown on this weighted data to improve upon the predictions of the first tree. The new model is formed by *Tree 1 + Tree 2*. The classification error is computed from this new 2-tree ensemble model and a third tree is grown to predict the revised residuals. In other words, it converts weak learners into strong learners by making each new tree a fit on a modified version of the original dataset.

```
1: $F_0(x) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2: For $m = 1$ to $M$ do:
3:     $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, i = 1, \ldots, M.$
4:     $a_m = \arg\min_{a,\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(x_i : a)]^2$
5:     $\rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i : a_m))$
6:     $F_m(x) = F_{m-1}(x) + \rho_m h(x : a_m)$
7: end For
8: end Algorithm
```

Fig 7. Pseudo code for gradient boosting model

This process is repeated for a specified number of iterations. Subsequent trees help to classify observations that are not well classified by the previous trees. Predictions of the final ensemble model is therefore the weighted sum of the predictions made by the previous tree models.

3. Performance Evaluation: The performance of both the linear regression model and the gradient boosting model is evaluated using the root-mean-square error metric. The performance of these models with different feature engineering techniques are also compared and evaluated.

Language Used

We will use Python as the primary programming language for our project. It is the most popular language suitable for a variety of tasks in machine learning.

Tools Used

For data preprocessing and visualization, we used pandas, a Python data analysis library, to manipulate the dataset, and we used matplotlib to plot the data into a visually viewable format. In the model implementation step, We used scikit-learn, which is a library for machine learning in Python and contains many efficient methods for statistical modeling such as regression, classification, and dimensionality reduction. We will also use Github for teamwork communication and collaboration.

How to generate output

Since the true price of the house is hidden by Zillow, we trained our dataset on a logerror, which is defined by the following equation:

$$logerror = log(Zestimate) - log(SalePrice)$$

Fig 8. Logerror Equation

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 Price: Zillow's Home Value Prediction to get the final score.

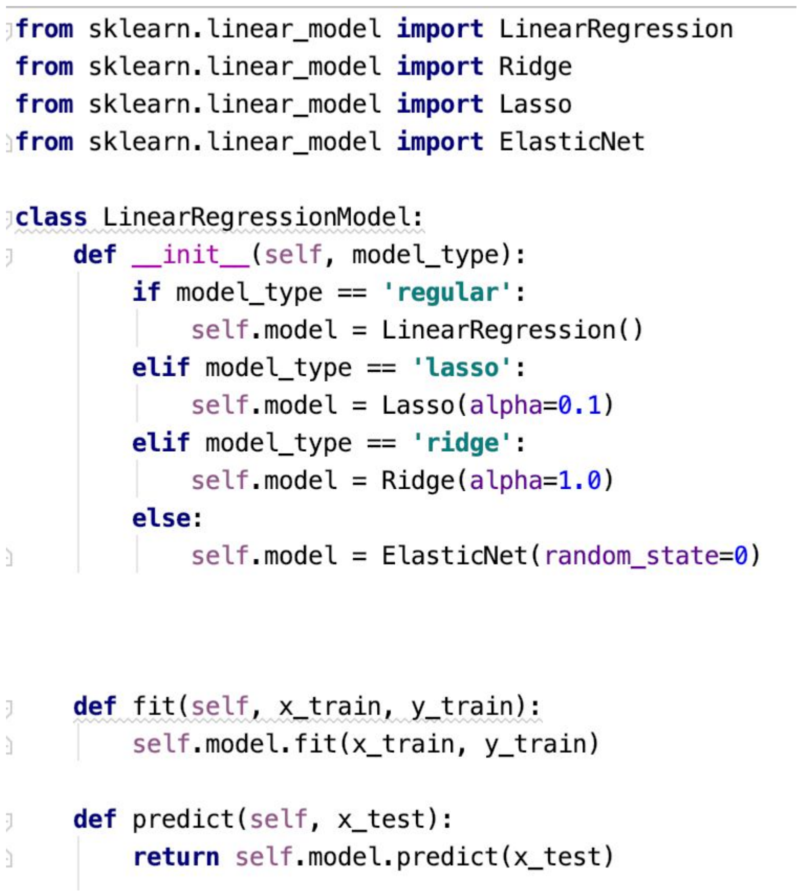
How to test against hypotheses

We compared the performance of both linear regression models and gradient boosting models on the same dataset and test cases. The result would show whether these two types of models have no difference in their performance or significant difference in their performance in predicting the house prices.

Implementation

Code

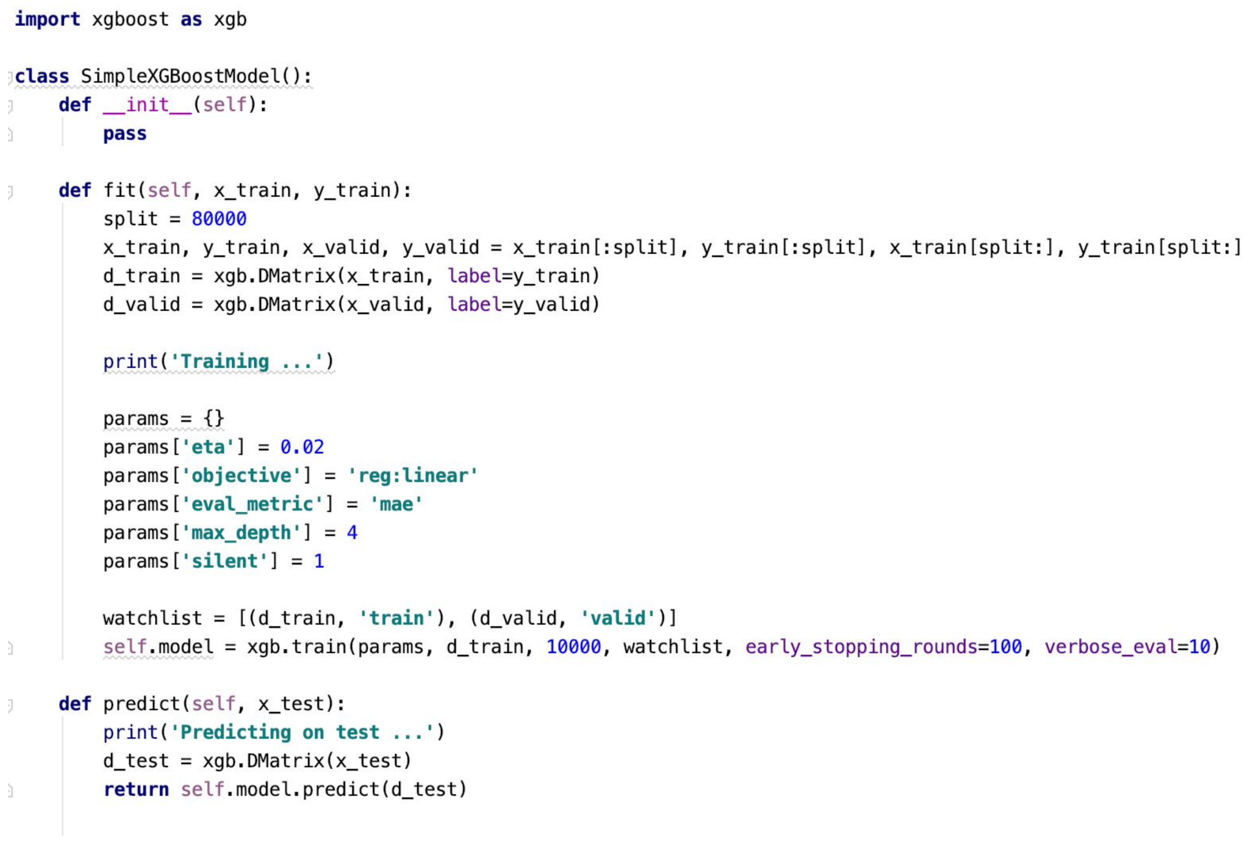
Linear Regression Model



```python
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

class LinearRegressionModel:
    def __init__(self, model_type):
        if model_type == 'regular':
            self.model = LinearRegression()
        elif model_type == 'lasso':
            self.model = Lasso(alpha=0.1)
        elif model_type == 'ridge':
            self.model = Ridge(alpha=1.0)
        else:
            self.model = ElasticNet(random_state=0)



    def fit(self, x_train, y_train):
        self.model.fit(x_train, y_train)

    def predict(self, x_test):
        return self.model.predict(x_test)
```

Fig 9. Linear Regression Code

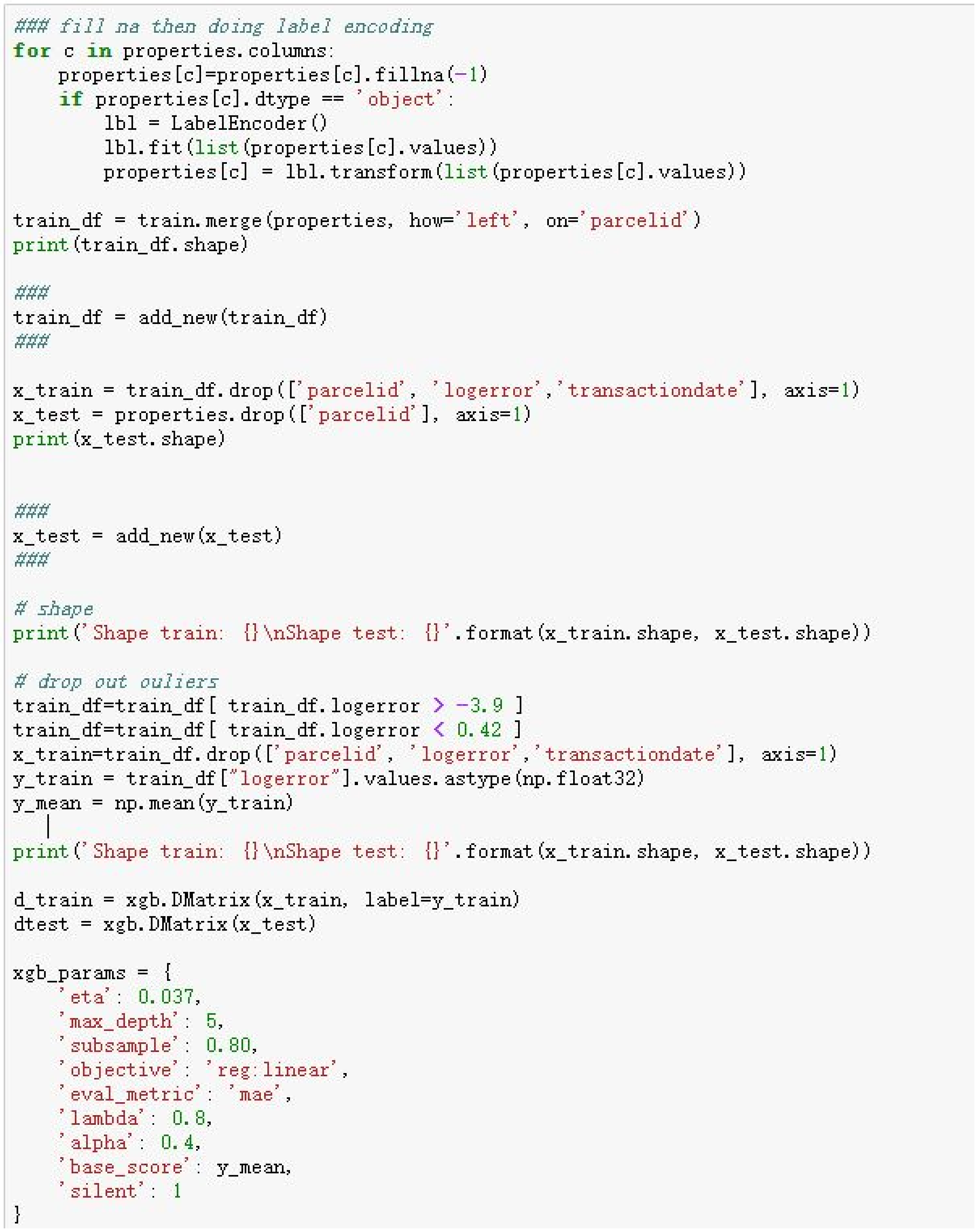
Simple XGBoost



```python
import xgboost as xgb

class SimpleXGBoostModel():
    def __init__(self):
        pass

    def fit(self, x_train, y_train):
        split = 80000
        x_train, y_train, x_valid, y_valid = x_train[:split], y_train[:split], x_train[split:], y_train[split:]
        d_train = xgb.DMatrix(x_train, label=y_train)
        d_valid = xgb.DMatrix(x_valid, label=y_valid)

        print('Training ...')

        params = {}
        params['eta'] = 0.02
        params['objective'] = 'reg:linear'
        params['eval_metric'] = 'mae'
        params['max_depth'] = 4
        params['silent'] = 1

        watchlist = [(d_train, 'train'), (d_valid, 'valid')]
        self.model = xgb.train(params, d_train, 10000, watchlist, early_stopping_rounds=100, verbose_eval=10)

    def predict(self, x_test):
        print('Predicting on test ...')
        d_test = xgb.DMatrix(x_test)
        return self.model.predict(d_test)
```

Fig 10. Simple XGBoost Code

Tuned XGBoost



```
### fill na then doing label encoding
for c in properties.columns:
    properties[c]=properties[c].fillna(-1)
    if properties[c].dtype == 'object':
        lbl = LabelEncoder()
        lbl.fit(list(properties[c].values))
        properties[c] = lbl.transform(list(properties[c].values))

train_df = train.merge(properties, how='left', on='parcelid')
print(train_df.shape)

###
train_df = add_new(train_df)
###

x_train = train_df.drop(['parcelid', 'logerror','transactiondate'], axis=1)
x_test = properties.drop(['parcelid'], axis=1)
print(x_test.shape)


###
x_test = add_new(x_test)
###

# shape
print('Shape train: {}\nShape test: {}'.format(x_train.shape, x_test.shape))

# drop out ouliers
train_df=train_df[ train_df.logerror > -3.9 ]
train_df=train_df[ train_df.logerror < 0.42 ]
x_train=train_df.drop(['parcelid', 'logerror','transactiondate'], axis=1)
y_train = train_df["logerror"].values.astype(np.float32)
y_mean = np.mean(y_train)

print('Shape train: {}\nShape test: {}'.format(x_train.shape, x_test.shape))

d_train = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test)

xgb_params = {
    'eta': 0.037,
    'max_depth': 5,
    'subsample': 0.80,
    'objective': 'reg:linear',
    'eval_metric': 'mae',
    'lambda': 0.8,
    'alpha': 0.4,
    'base_score': y_mean,
    'silent': 1
}
```

Fig 11. Tuned XGBoost Code

Design Document

Each class represents a model type and mainly contains two functions.

Table 2. Functions Description

Function Name	Input	Output	Description
constructor()	model_param (optional)	None	Construct the machine learning model.
fit()	x_train: training data y_train: training label	None	Feed the model with training data and labels.
predict()	x_test: testing data	predicted labels	Predict the testing data using the trained model.

Workflow Chart

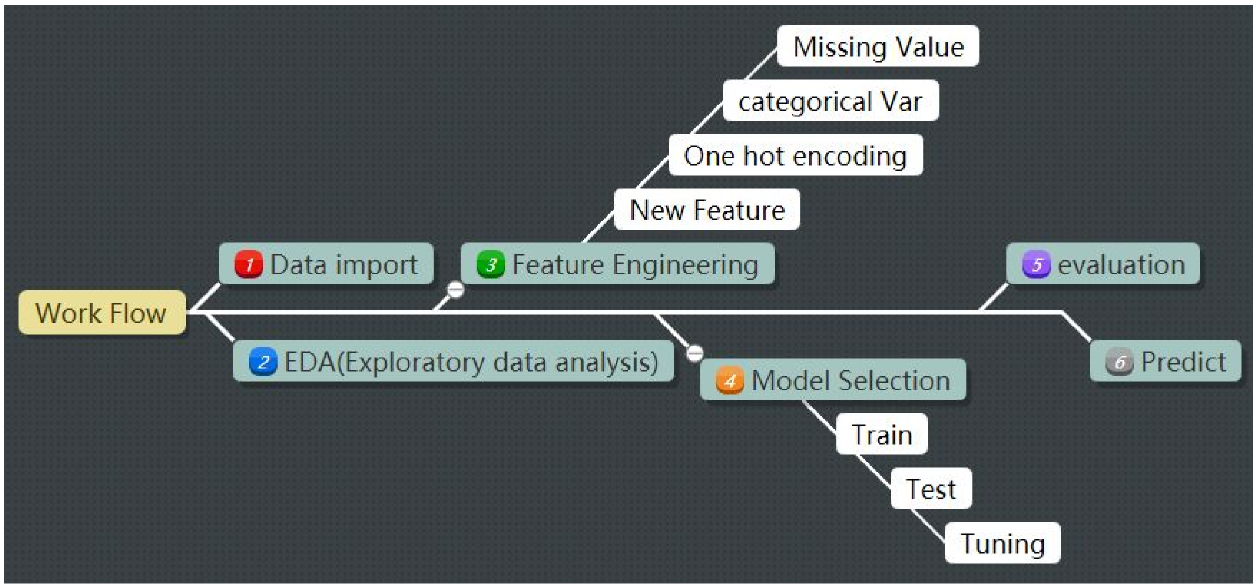


Fig 12. Workflow Chart

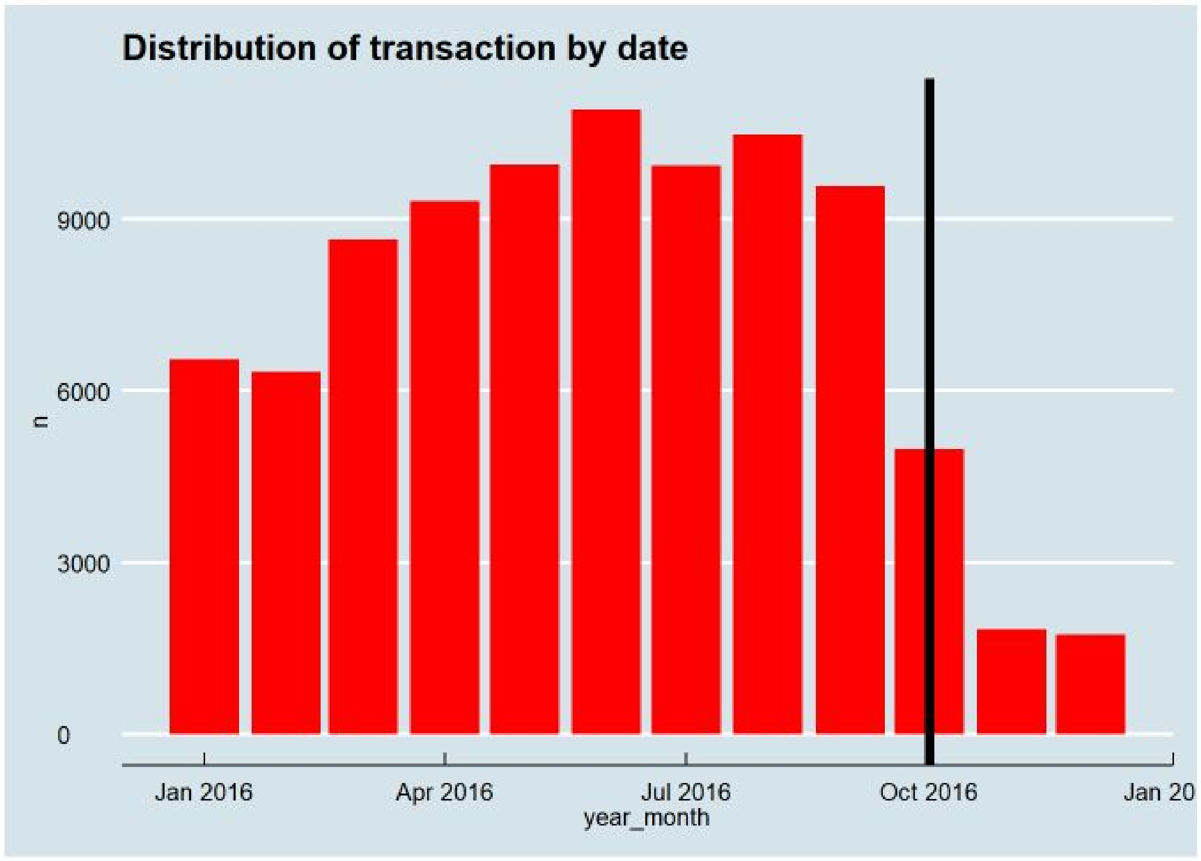
The first step is to import data from Kaggle. Once we retrieved the data, we performed exploratory data analysis on the dataset to gain a visual understanding of the data. We then performed feature engineering on the dataset, such as adding new features, one hot encoding, label encoding on categorical variables, and removing missing values to increase the performance of the machine learning models. After that, we built linear regression models and XGBoost models, and fed the preprocessed data into the models as input. We also tuned the XGBoost model to increase its accuracy. Finally, we compared the performance of all models on this dataset.

Data analysis and Discussion

Output generation

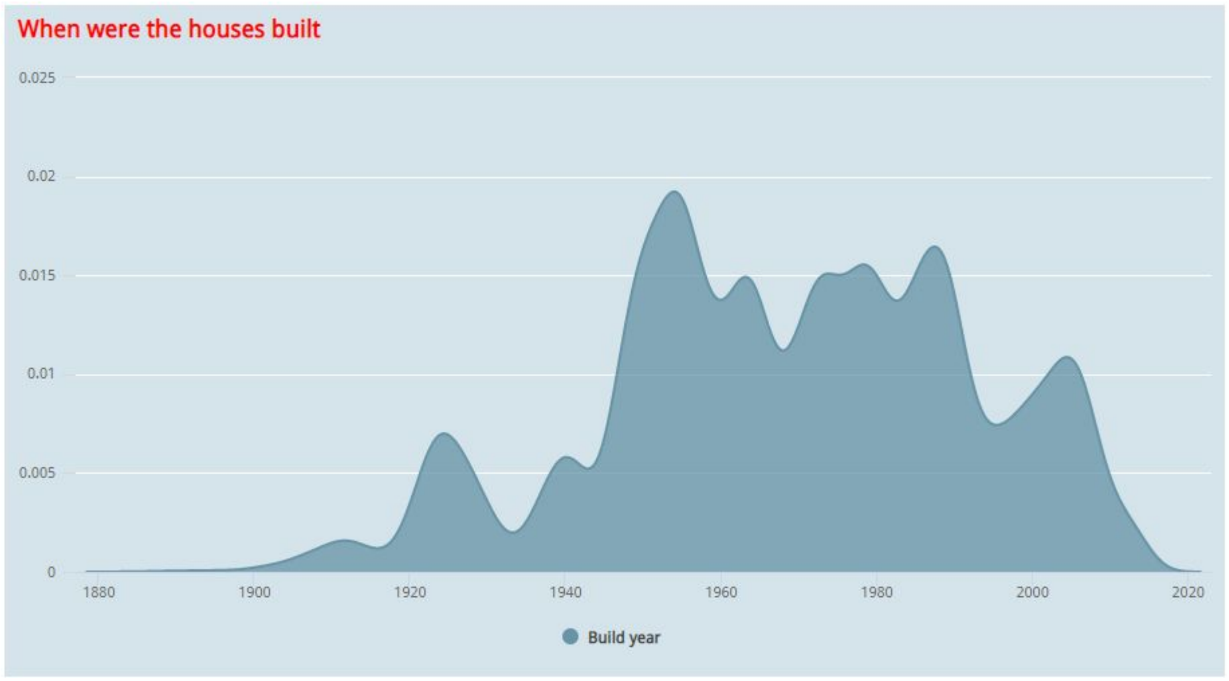
For exploratory data analysis , we used various analysis and visualization packages in **R** to implement interactive visualization to help us better understand the characteristic for our data before further feature engineering and modeling.

This is the transaction distribution over time. There were only some of the transactions after 2016/10 in the train set, because the rest is in the test set.



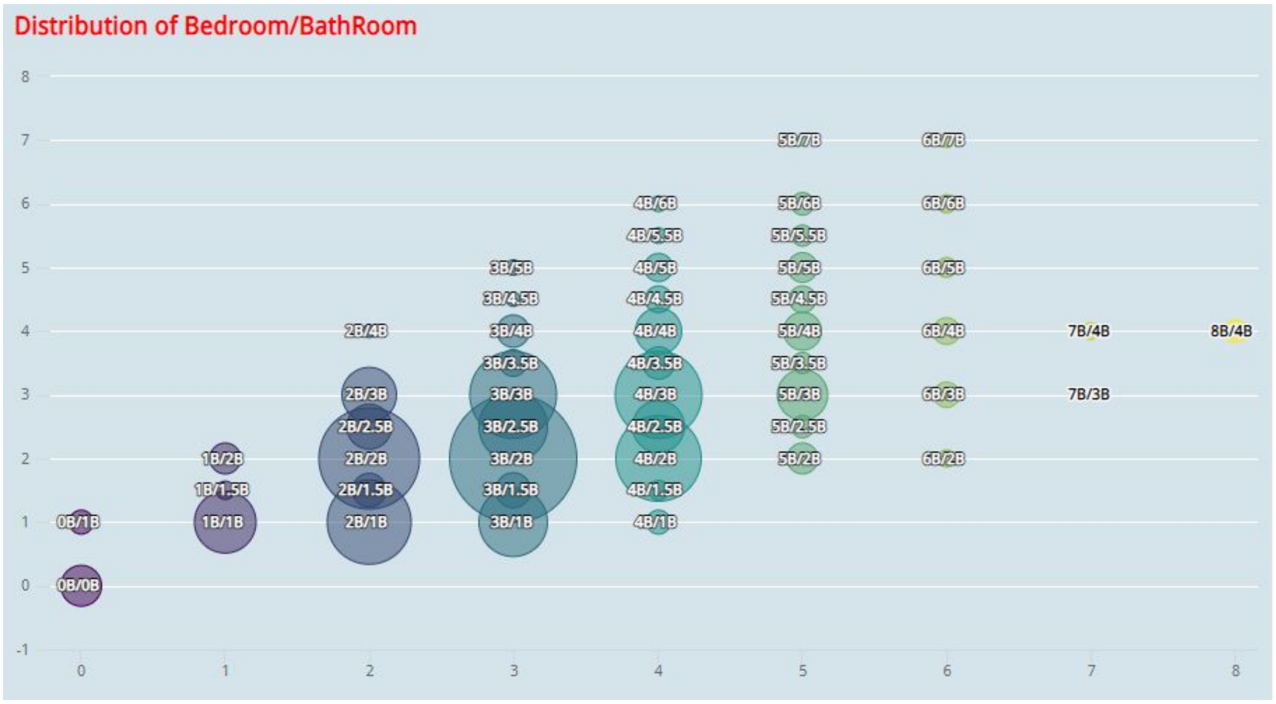
When house were built

The majority of houses were built between 1955~1995. Surprisingly, after 2000 there were no newly built houses.



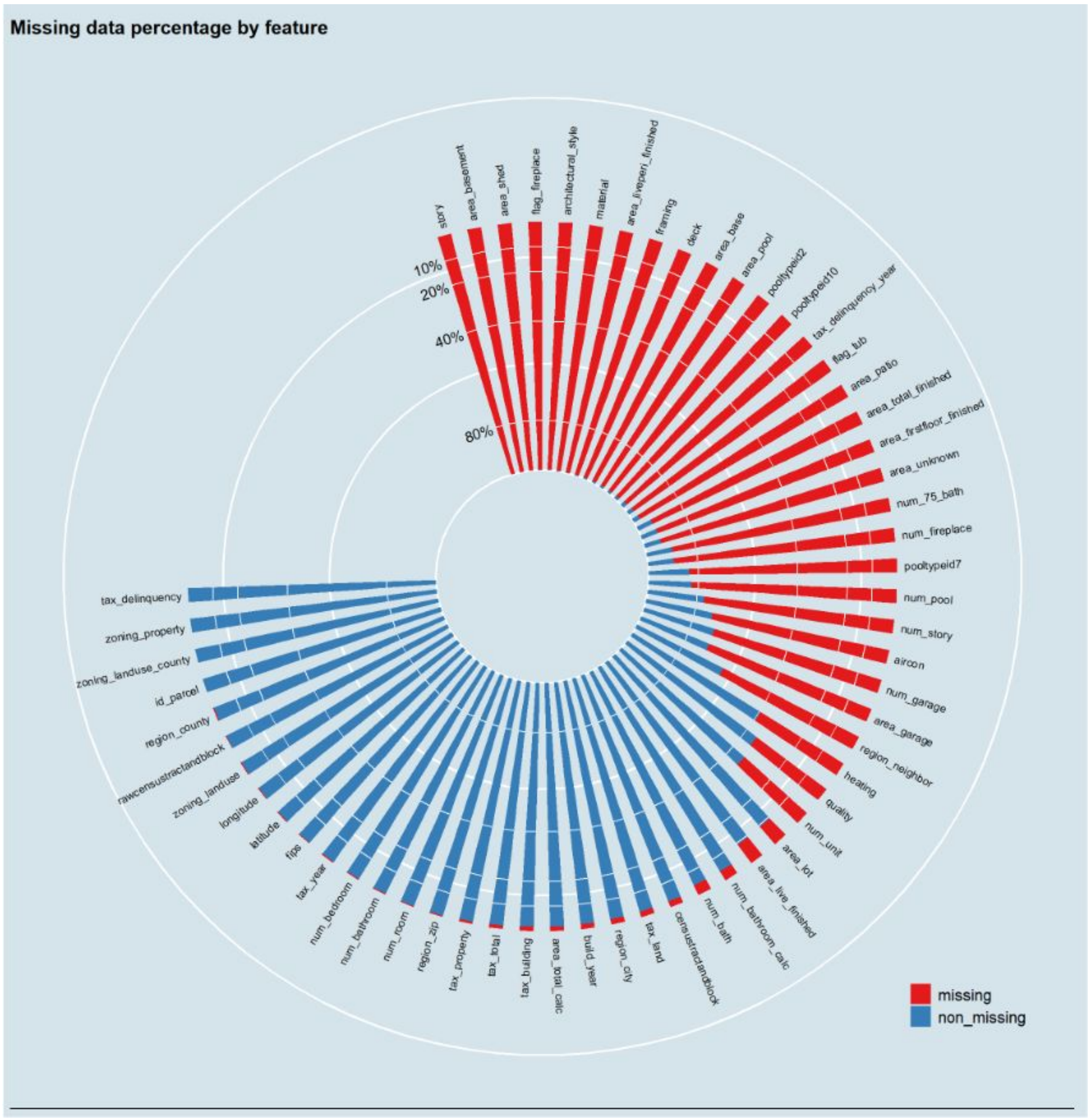
BB(Bed/Bath) combination distribution

Through the distribution of room type, we can see that 2b2b and 3b2b was most popular in the LA real estate Market.



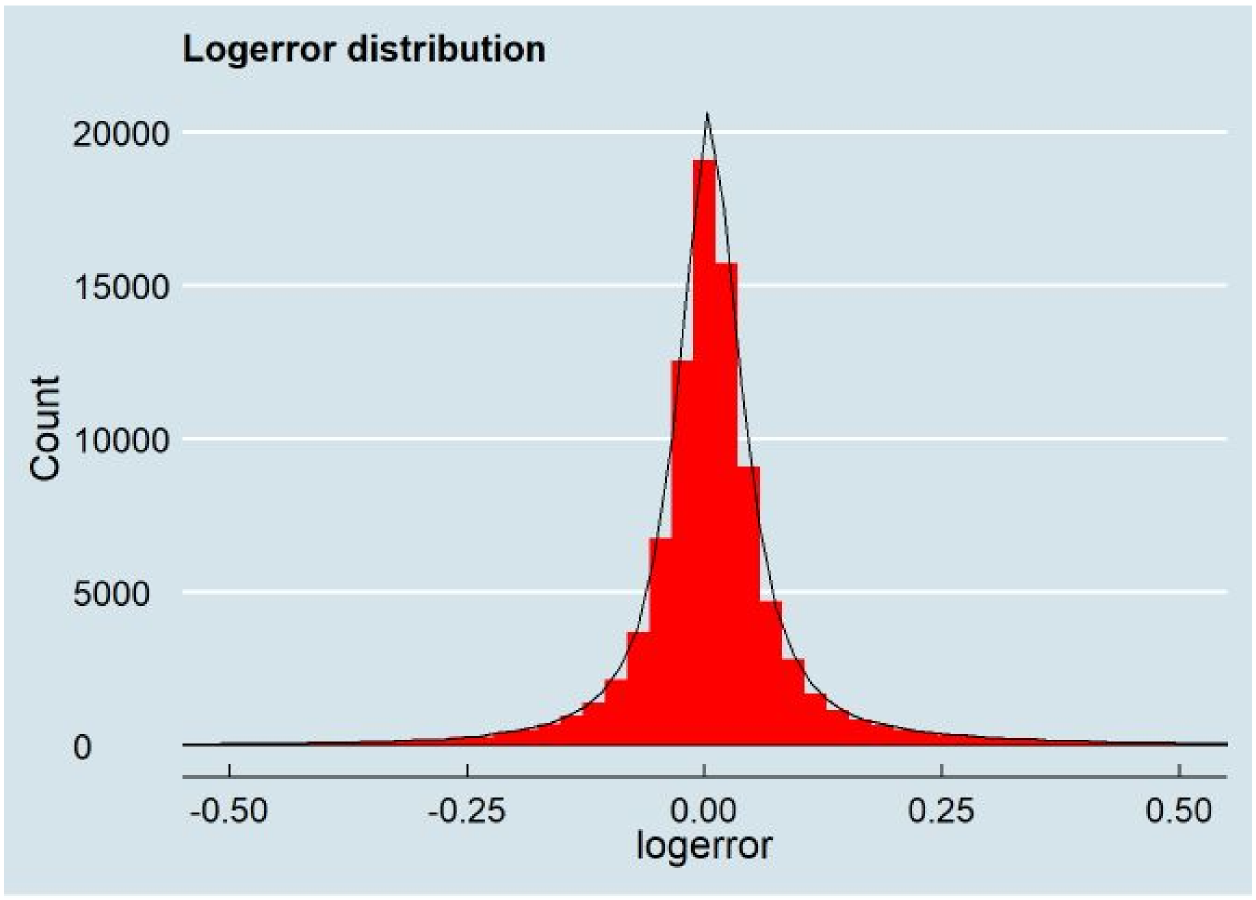
Missing value

Handling missing values was probably one of the most important processes during feature engineering. we can see 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.



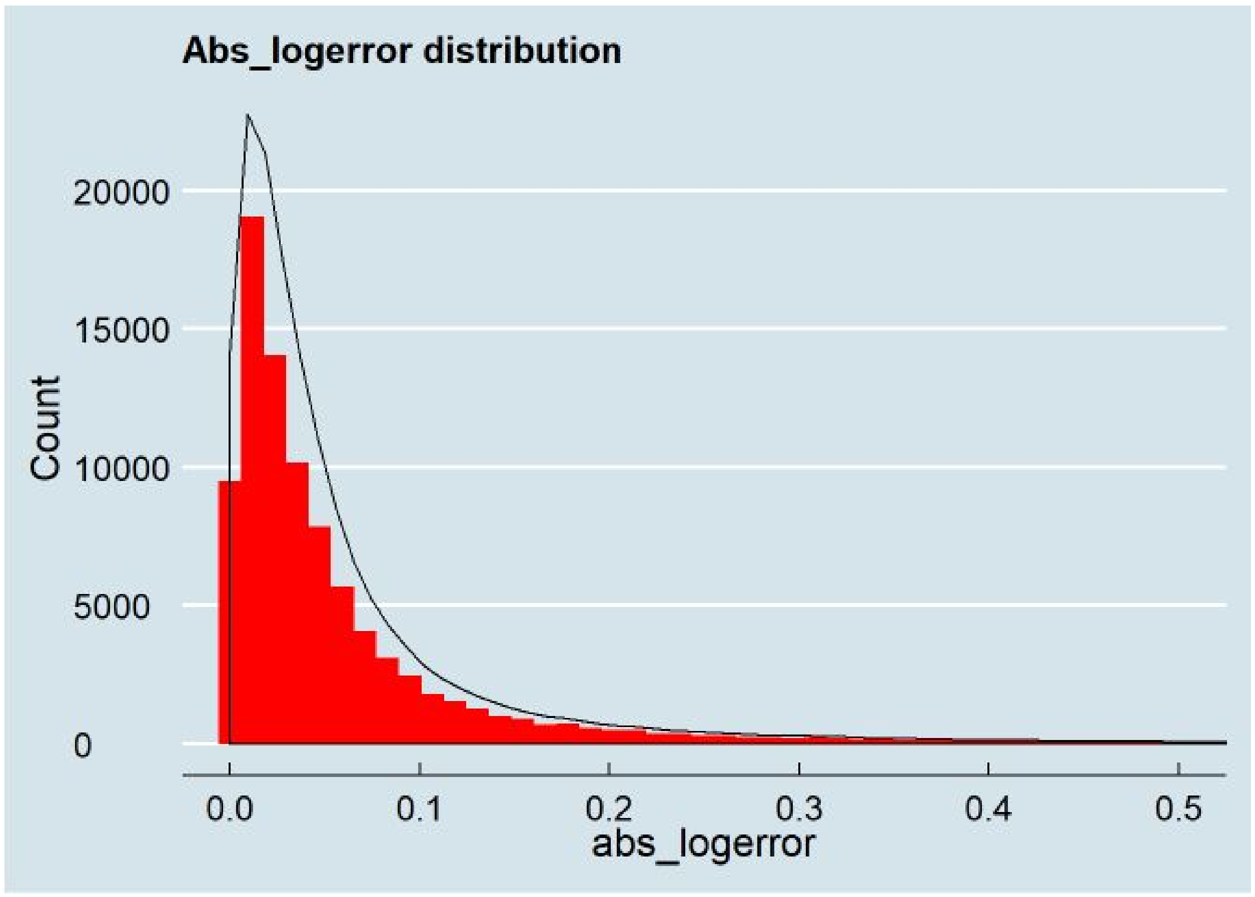
Distribution of Logerror

To get the feel of the label(y) , we can check the distribution of our outcome (logerror). Notice the difference in log(Zestimate) - log(Realprice).



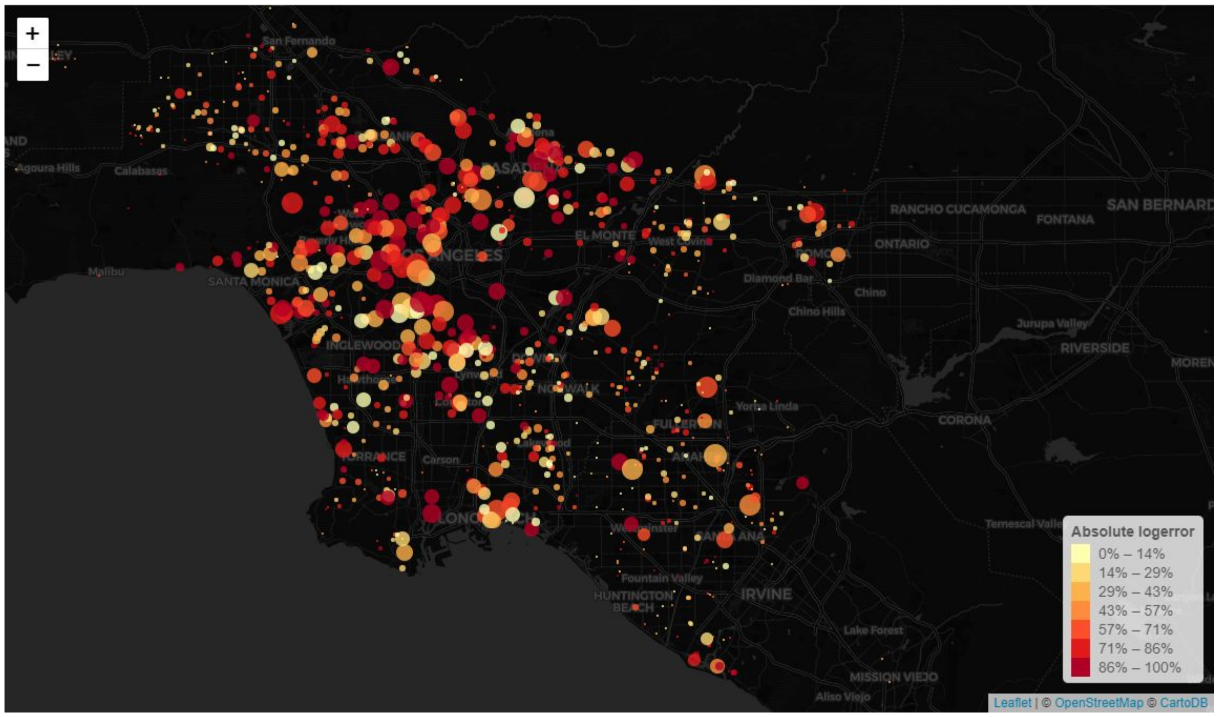
Absolute logerror

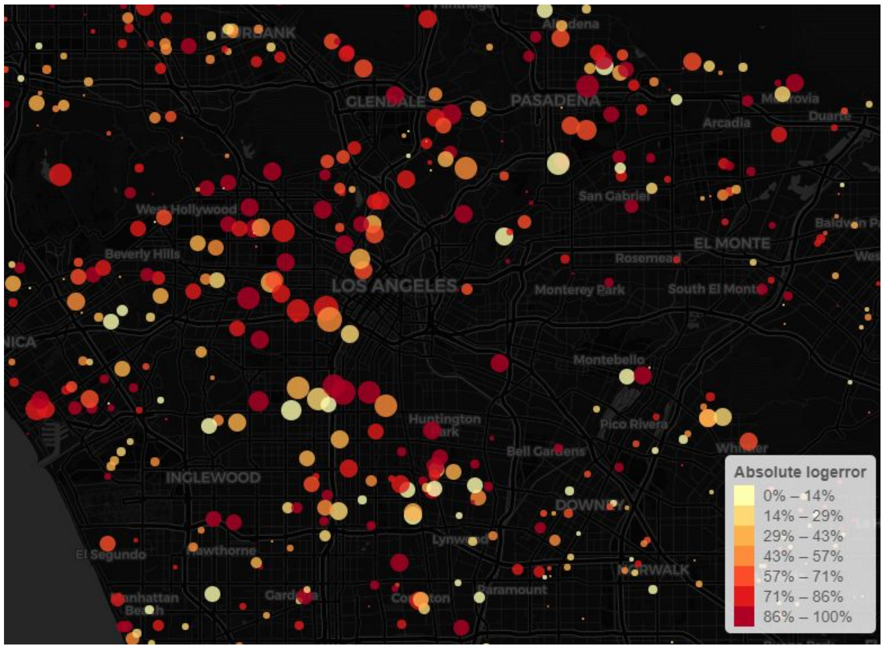
A smaller value of abs_error that Zestimate is close to real value. So, Zestimate predictions are close to the Sale Price. So the greater abs error will indicate that it's over- or under-estimating the real price.



Abs_error on the map

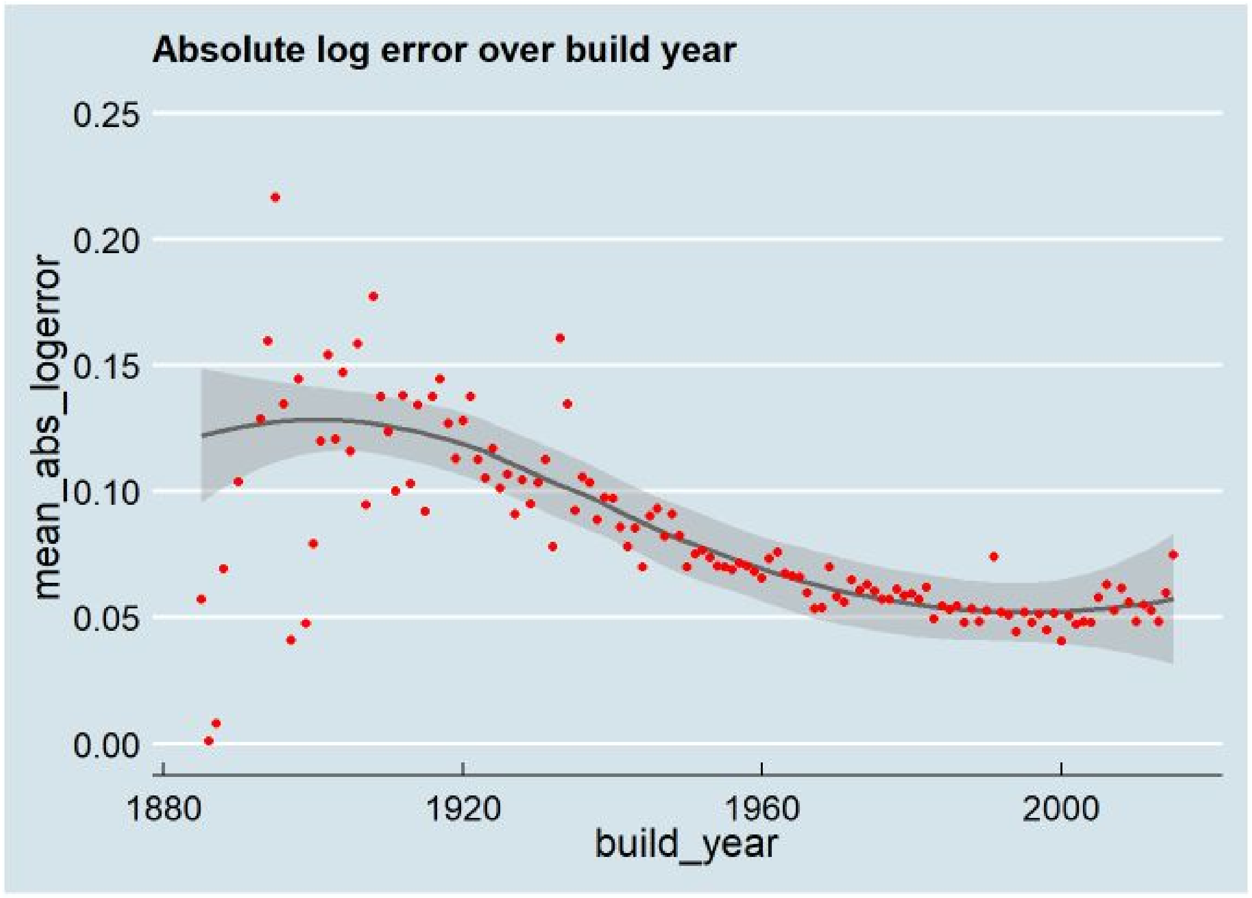
Here we sampled 2000 housing data on the map. The larger the size of bubble, the larger of this housing age is, and higher colour means larger abs error. It gives us a feel that, older house tends to easily get miss-predicted (overestimated or underestimated)





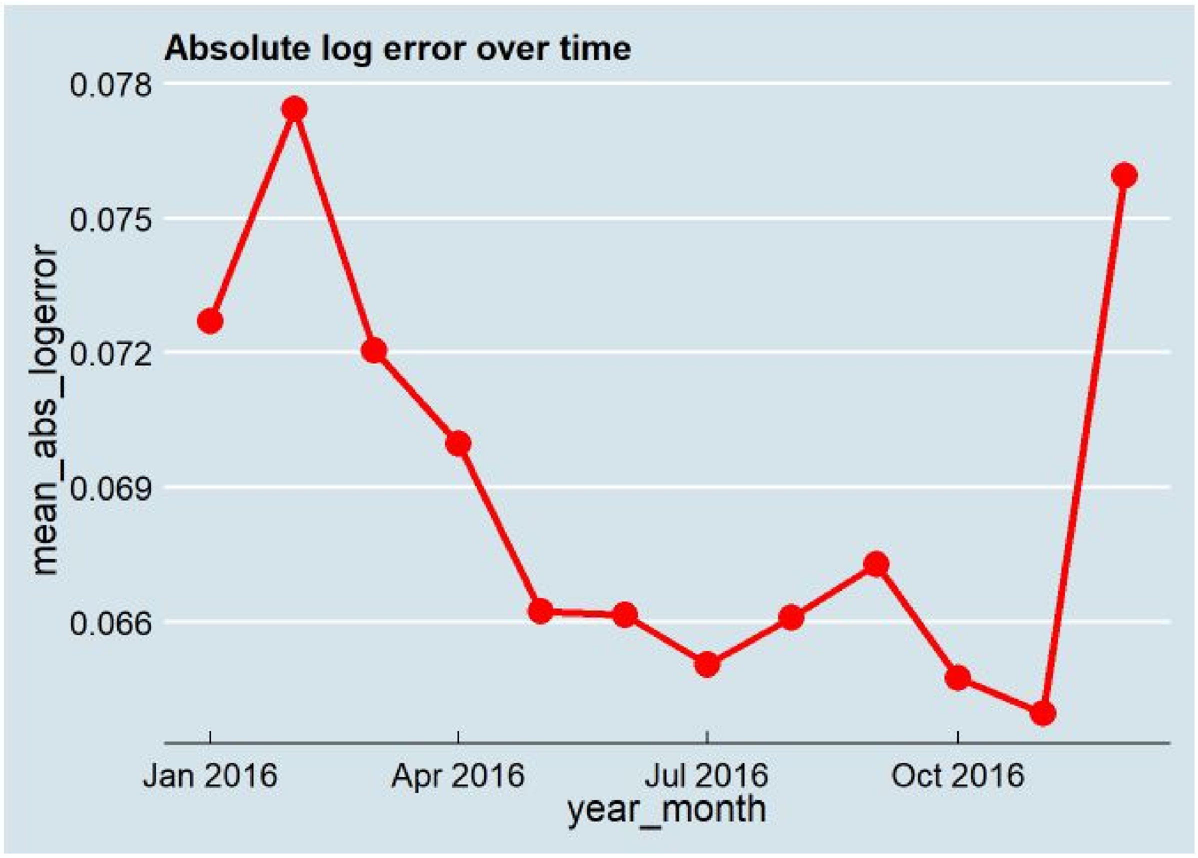
Abs_error change with build_year

Here the plot seems corresponding with what we saw previously. And also we can see that houses built around 1950 seem to also get a very good estimation.



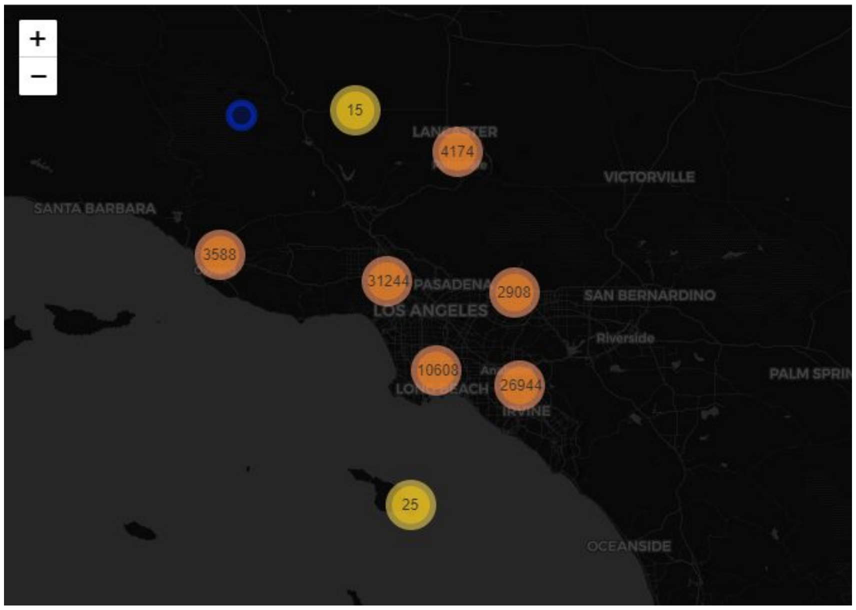
Absolute log error over time

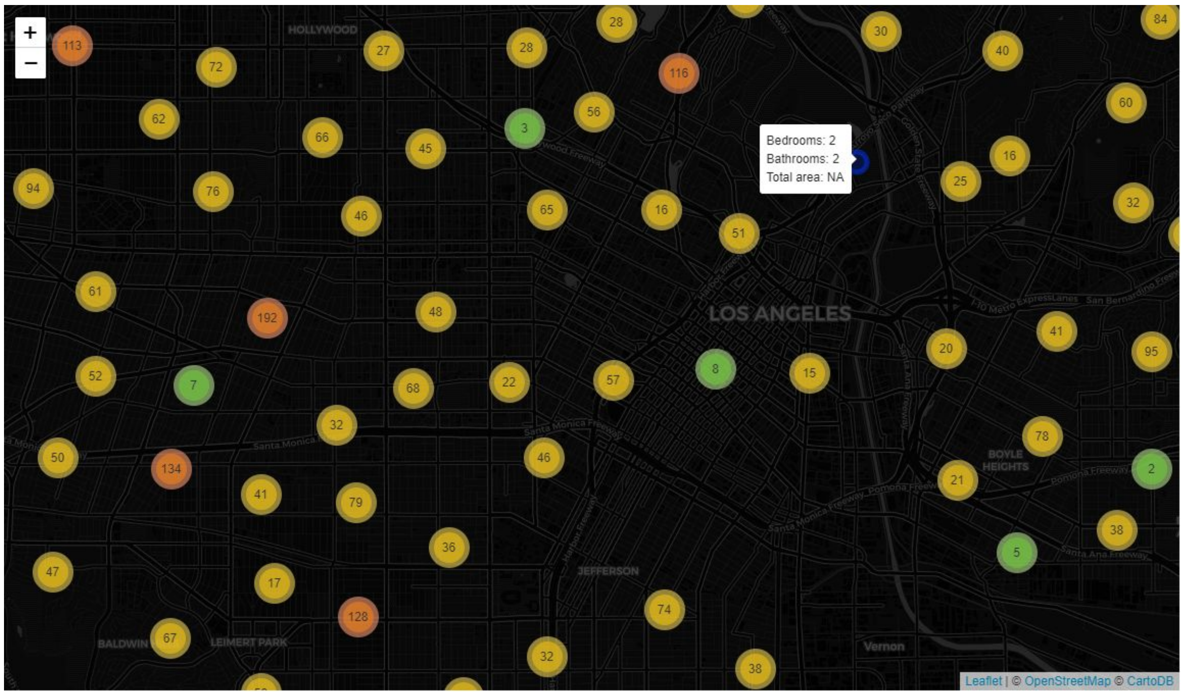
It looks like the abs log error in 2016 for zillow zestimate is getting better prediction.



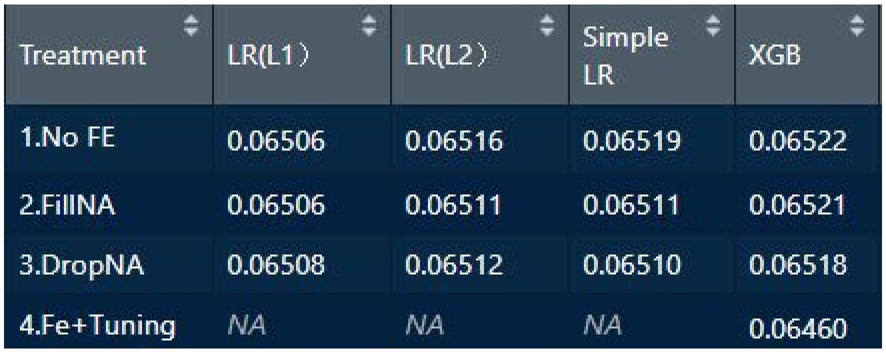
Cluster of housing point on map

Check how those 80,000 housing transaction data are distributed on map with automatic clustering.



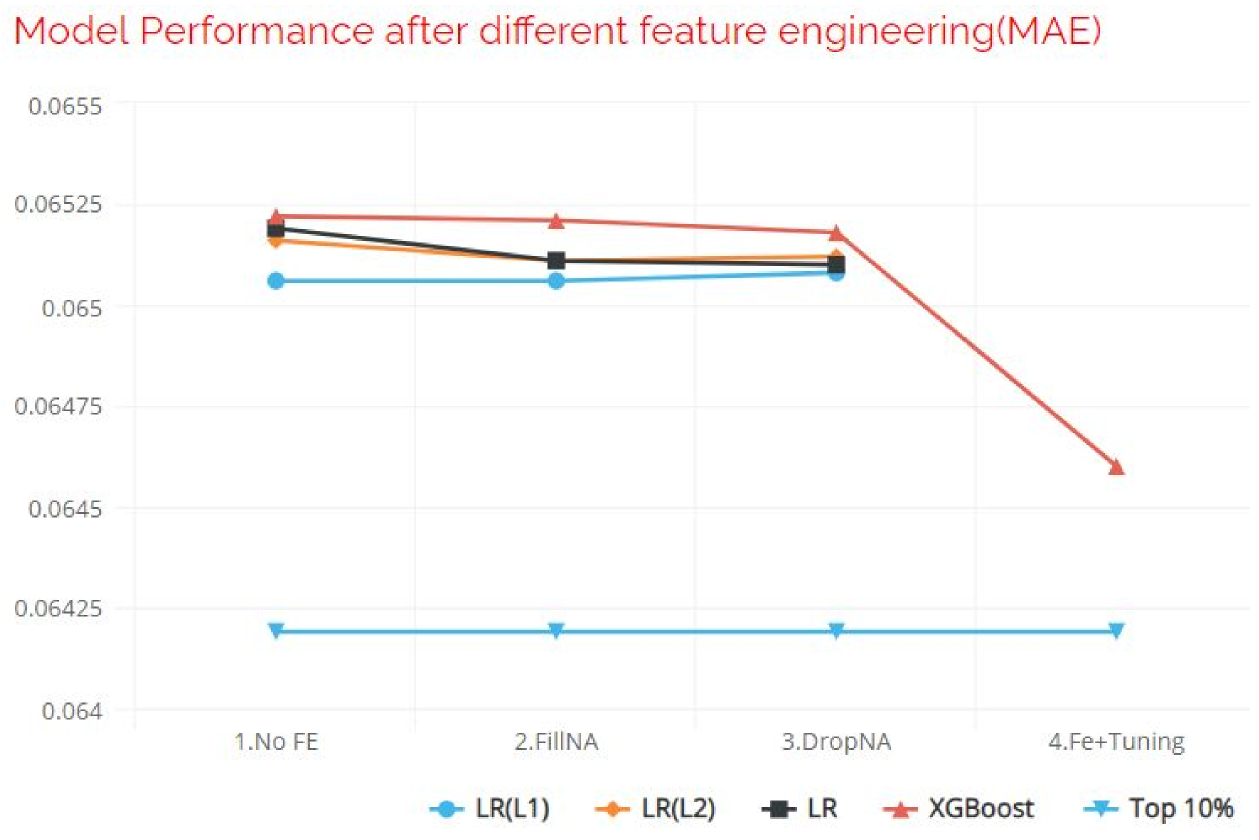


Output analysis

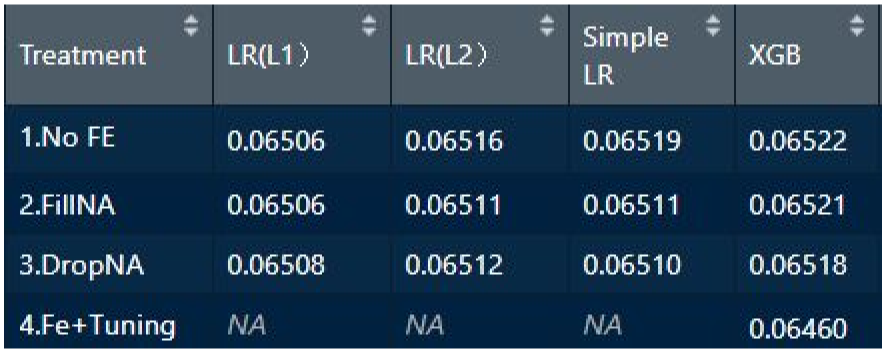


Treatment	LR(L1）	LR(L2）	Simple LR	XGB
1.No FE	0.06506	0.06516	0.06519	0.06522
2.FillNA	0.06506	0.06511	0.06511	0.06521
3.DropNA	0.06508	0.06512	0.06510	0.06518
4.Fe+Tuning	*NA*	*NA*	*NA*	0.06460

After each stage of experiment on different models, we had the test result above. XGboost has the best performance in terms of MAE as 0.06460 after several feature engineering and hyperparameter tuning.As a reference, we got the Kaggle leaderboard for top 10% model performance at 0.06419.



Compare output against hypothesis

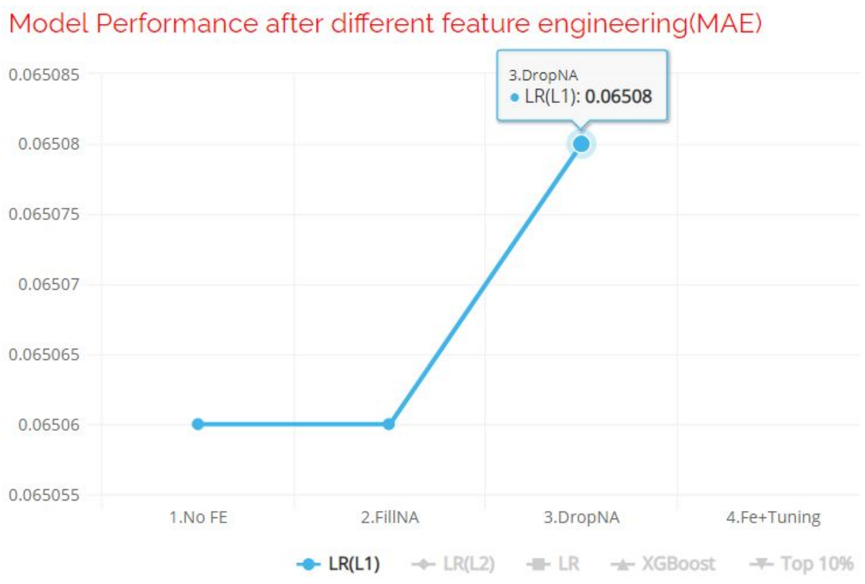


Treatment	LR(L1）	LR(L2）	Simple LR	XGB
1.No FE	0.06506	0.06516	0.06519	0.06522
2.FillNA	0.06506	0.06511	0.06511	0.06521
3.DropNA	0.06508	0.06512	0.06510	0.06518
4.Fe+Tuning	*NA*	*NA*	*NA*	0.06460

Based on the result here , we can reject the null hypothesis which hereby said that the XGBOOST actually performs better performance for linear regression model after feature engineering and hyperparameter tuning.

Abnormal case explanation

One of the abnormal cases we have here is after dropping the na column for L1 model, the MAE actually goes up which means the performance goes down. Possible reason could be underfitting since Drop missing value will lose information for linear model at L1 normalization.



Conclusions and Recommendations

Summary and conclusions

Throughout our project, we could conclude that XGBOOST as an GBDT could gives us Slightly better model performance especially in the housing price prediction area but only as a starter. What is more important, it's how you interpret your data and do feature engineering. There is an "old saying" in the ML area: Data and features determine the upper limit of machine learning, and models and algorithms just approach this upper limit. In machine learning, a prediction model (linear regression, logistic regression, SVD, etc.) and a bunch of raw data are often used to get some predicted results. What people need to do is to extract better results from this bunch of raw data. And then make the best prediction. This includes two aspects, the first is how to select and use various models, and the second is how to use these raw data to achieve the best results. So how can we get the best results? : In fact, the success of all machine learning algorithms depends on how you present your data.

Recommendations for future studies

For better model performance, there were also several options that could be done for future study.

* feature engineering:

During our research project, we only tried adding new features such as ratio or sum of other features.However, for those missing values , we could implement K-NN for restoring missing values instead of filling it with -1 or median.

* new model:

While XGBoost performs really great, there were a lot of models that came out recently with better results. For example , LightGBM (by Microsoft) and CatBoost.

* model fusion of ensemble model:

There were a lot of top results（ie.champion for this competition）,and used model ensembles for better results. We could consider stacking several different boosting models with giving weight to get a better result.

Bibliography

1. Cinar, Utku. (2019). Combining Domain Knowledge & Machine Learning: Making Predictions using Boosting Techniques. 9-13. 10.1145/3369114.3369125
2. “Installation Guide¶.” *Installation Guide - Xgboost 1.1.0-SNAPSHOT Documentation*, xgboost.readthedocs.io/en/latest/build.html#python-package-installation.
3. Chen, Tianqi, and Carlos Guestrin. “XGBoost.” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016, doi:10.1145/2939672.2939785.
4. Varma, Ayush, et al. “House Price Prediction Using Machine Learning and Neural Networks.” *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, doi:10.1109/icicct.2018.8473231.