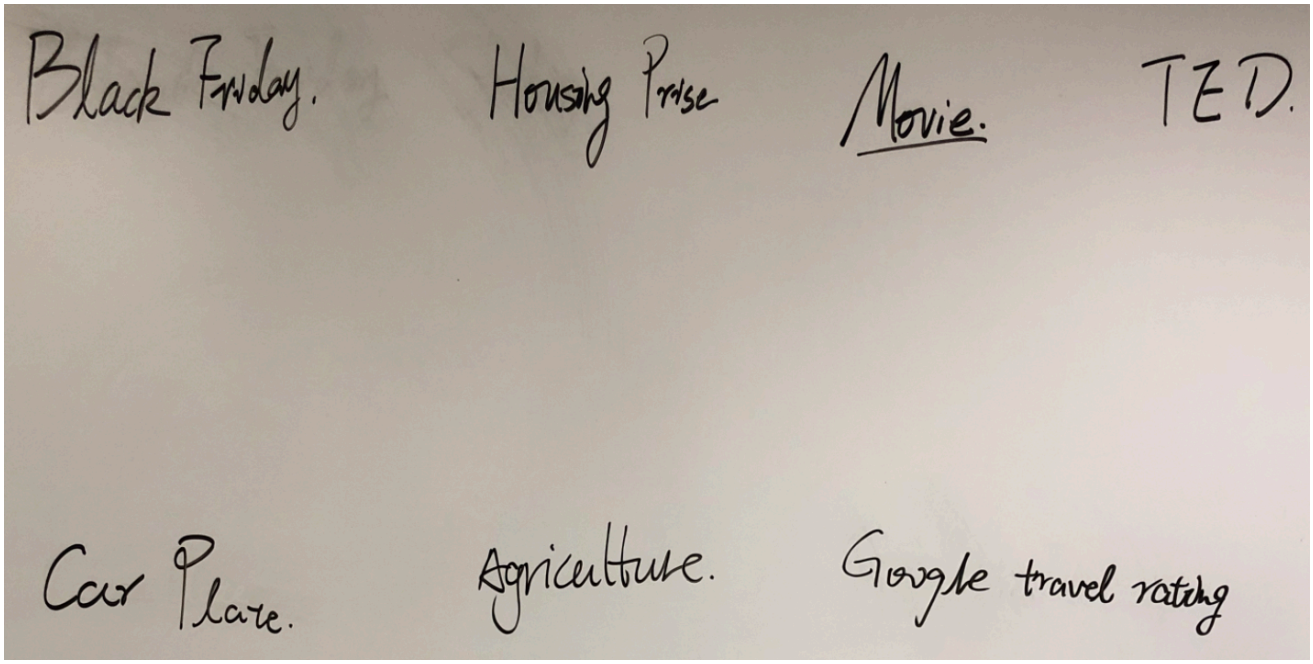
# House Price Prediction

## Machine Learning Final Project

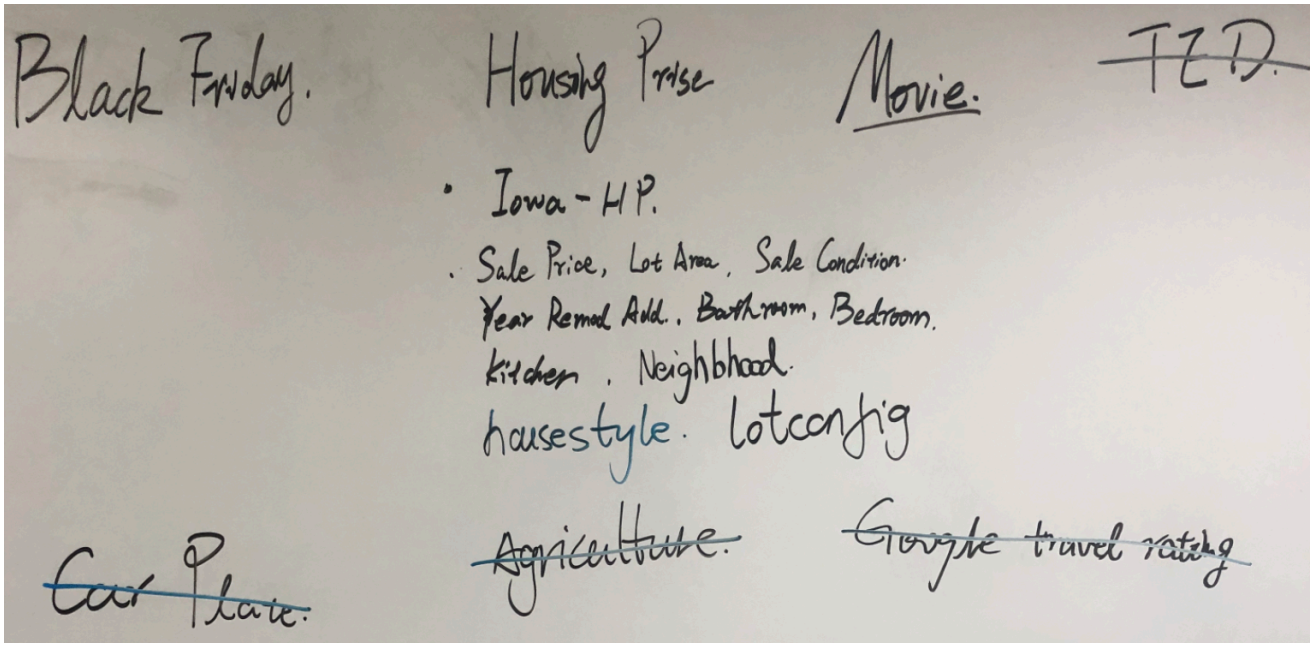
Zikun Qian, N15544629
Zhong Wang, N12413140
Chenxi Cheng, N17666649
Sanyang Yang, N18678343
Github: https://github.com/ChenxiiCheng/ML-Final

## Topic Selection

At first, we find and come up with some interesting topics, shown below:




After discussion, we selected the Housing Price Prediction as the final topic. As a popular and familar topic, it has abundant resources we can refer to online. Then we filtered some relevant features according to the data set.



## General Procedure

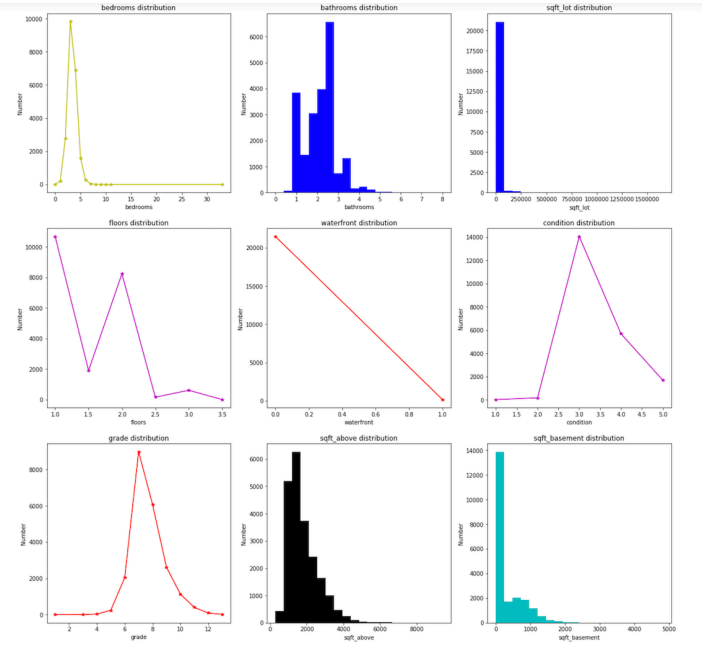
- Data visualization. Select some features manually from the data and visulize the data to get an intuitive recogtion of the data set.
- Data preprocessing. Do preprocessing like normalization and filling in the blank on the data to improve the validation accuracy.
- Model selection. Try to use some models we studied or from external resources to train the data, and use K-fold validation methods to evaulate the error on the test data to select the best model.
- Evaluate the model. Select the best model and retrain the model with some extreme value removed to check whether it imporves the performance.

## Data Visualization

We selected 9 features that are most relevant to house price from on data set, including `bedrooms`, `bathrooms`, `sqft_lot`, `floors`, `waterfront`, `condition`, `grade`, `sqft_above`, `sqft_basement`.

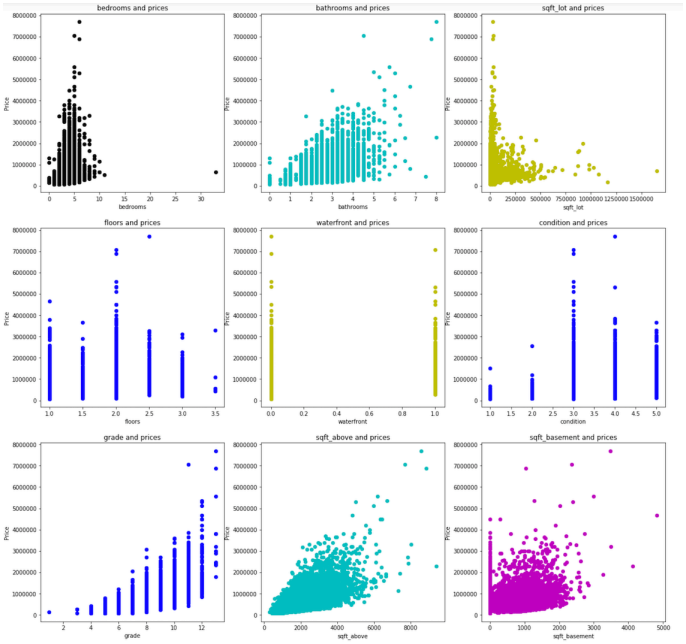
1.**Distribution**.

For every feature, we first compute each distinct value and the number of samples equal to the value in the training data.If there are no more than 20 distinct values, we plot a line graph, otherwise we plot a histogram with 20 blocks. This gives us an intuitive distribution of the feature.

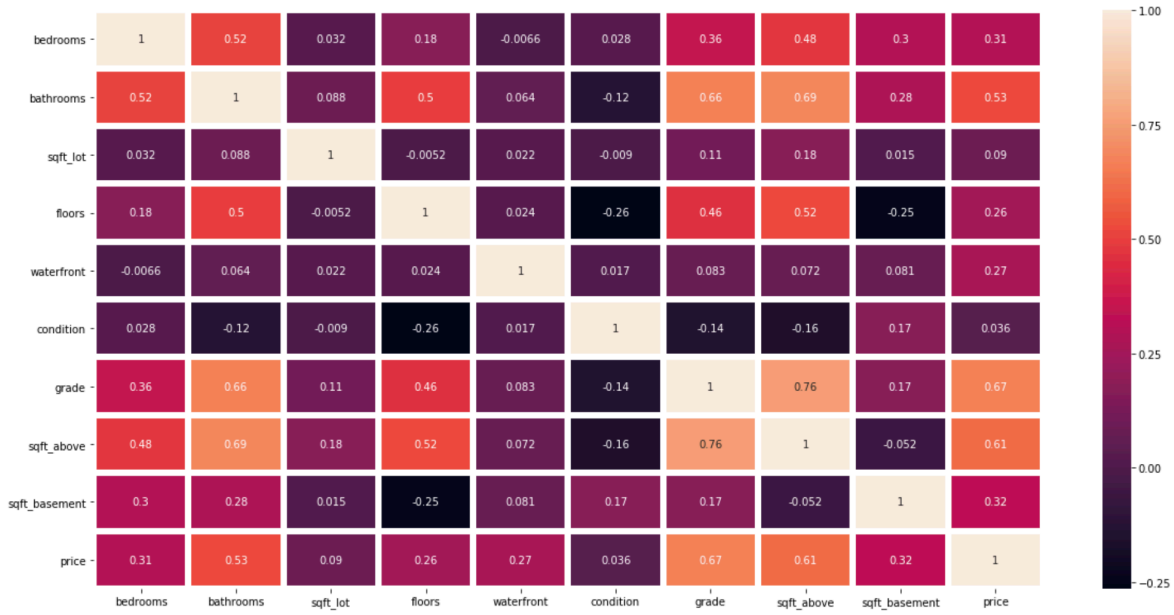


2.**Correlation with the price**.

For every feature, we plot a scatter graph showing the feature value and the price. Usually all features are in the positive correlation with the price attribute. We can briefly draw a conclusion from the graph which feature has more weight in deciding the price.




3.**Correlation of each feature**. Calculate and plot the heap map of each pair of attributes. And may exclude the features with the least correlation with the price in future training.




## Data process

As we know, the data acquired online usually have many null values, so it's necessary to preprocess the data so that the final prediction will have a better performance. Measures are as listed:

1. Eliminate the whole data row if there is any null value exist,
2. Replace the null value with the mean value of a certain feature.

```
def detect_outliers(df,n,features):

    outlier_indices = []

    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col], 75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step)].index

        # multiple_outliers = list( k for k, v in outlier_indices.items() if v > n)
        outlier_indices.extend(outlier_list_col)
        x = np.array(outlier_list_col)
        #print(len(x))
    return outlier_list_col

Outliers_to_drop = detect_outliers(df, 2, ['price'])

print("The outliers detected before drop: ", len(Outliers_to_drop))

df = df.drop(Outliers_to_drop, axis=0).reset_index(drop=True)

Outliers_to_drop = detect_outliers(df, 2, ['price'])

print("The outliers detected after drop:  ", len(Outliers_to_drop))
```

```
The outliers detected before drop:  1146
The outliers detected after drop:   264
```

**Analyse**

Since there are more than 10k samples in the original data, it would not hurt the whole integrality if the incomplete data is less than 5%, however, if there are too many samples to be eliminated, then the second measure will be better for it can suit the major part of data except for some outliers and it won't cause problems for reducing the sample size.

We selected the first one because of its perfect data integrality.

However, after the preliminary data process, a problem still exists. Some of the samples with extreme feature values, are too rare in the graph to consider as the training sample, but still affect the whole prediction, so it is necessary to eliminate these outliers and then to train these sample. The specific removal method is shown above.

As the price is most significant, we detected some outliers in 'Price', remove them, and then train the data again and again until we find the best outlier_step, which decides the range of removal. After removing these outliers, we obtained a better prediction, with 4% improvement, compared with the first beginning.

Besides, there are so many features we need to consider, which may not be that relevant, so we thought it should have a better performance if we remove certain less relevant features, and pay more attention to those main influential features. Then we tried to remove the last 2 features,'sqft_lot' and 'condition' according to the heatmap. To our surprise, even though it has low relation to the price, after removing these 2 features, we didn't get what we want, the final R2_score is a little bit lower than before. So we kept our old way.

```
# Reconstruct the model with selected features,since we know the relation between features and price

feature_list1 = df[['bedrooms','bathrooms','sqft_living','floors',
             'view', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'lat', 'long']]
X_new = np.array(feature_list)
y_new = np.array(price)
X_train, X_test, y_train, y_test = train_test_split(X_new, y_new, test_size=0.30, train_size=0.70, shuffle=True)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
X_train, X_test, y_train, y_test = train_test_split(X_new, y_new, test_size=0.30, train_size=0.70, shuffle=True)
GradientBoostingRegressor_model = GradientBoostingRegressor(n_estimators = 400, min_samples_split = 2,
         learning_rate = 0.1, loss = 'ls')
GradientBoostingRegressor_model.fit(X_train, y_train)

yhat = GradientBoostingRegressor_model.predict(X_test)
R2 = 1 - np.mean((y_test - yhat)**2) / (np.std(y_test)**2)

kf = KFold(n_splits=5, shuffle=True)
score_ndarray = cross_val_score(GradientBoostingRegressor_model, X_test, y_test.ravel(), cv=kf)
print('r2_score = ', score_ndarray)
print('r2_score_average = ', score_ndarray.mean())
print(X_train.shape[0])
# we can see that the final r2_score is not higher than before, so we can learn that the more related fetures we use, t
# accurate the final r2_score is, though it may cost more time to calculate
```

```
(15129, 13) (6484, 13) (15129, 1) (6484, 1)
```

```
/Applications/Anaconda/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:578: DataConversionWarning:
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example
using ravel().
  y = column_or_1d(y, warn=True)
```

```
r2_score =  [0.83997417 0.86227008 0.79795894 0.87565575 0.85403129]
r2_score_average =  0.8459780460332107
```

### Model selection, training, and prediction

In this project, we use each of following method to instantiate the model object. After that, use the model to predict yhat and KFold to instantiate cross validation object. Finally, we choose cross_val_score method for cross validation. We judge the fitness of the model by calculating the value of R2. Usually, the closer the value is to 1, the better the effect.

```
# Get the value of X and y by the original data and related features
X = np.array(feature_list)
y = np.array(price)

# Split X, y to X_train, X_test, y_train, y_test according to cross-validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, train_size=0.70, shuffle=True)
```

1. **Linear Regression Model**

   The first model is linear regression model. Linear regression can describe the relationship between data accurately with a straight line. In this way, when new data emerges, a simple value can be predicted. Linear regression is often used in house price forecasting. Linear Regression have a fast Modeling speed, it does not need very complicated calculation. Also, Each variables can be understand and explained according to the coefficients. Besides these, It is sensitive to outliers.

```
# Module 1: LinearRegression

LinearRegression_model = LinearRegression()
LinearRegression_model.fit(X_train, y_train)

yhat = LinearRegression_model.predict(X_test)
R2 = 1 - np.mean((y_test-yhat)**2) / (np.std(y_test)**2)

kf = KFold(n_splits=5, shuffle=True)
score_ndarray = cross_val_score(LinearRegression_model, X_test, y_test.ravel(), cv=kf)

print('r2_score = ', score_ndarray)
print('r2_score_average = ', score_ndarray.mean())
```

```
r2_score =  [0.66416555 0.68054637 0.63917549 0.68854991 0.65898906]
r2_score_average =  0.6662852767895473
```

2. **Decision Tree Regression Model**

   The second model is decision tree regression model. Decision tree is a common non-parametric supervised learning method for classification and regression. The goal is to create a model to predict the value of target variables by deducing simple decision rules from data characteristics. It is a module that simple and easy to understand, and the number structure can be visualized. Also, It can deal with multi-output problem.

```
# Module 2: DecisionTreeRegressor

DecisionTree_model = DecisionTreeRegressor()
DecisionTree_model.fit(X_train, y_train)

yhat = DecisionTree_model.predict(X_test)
R2 = 1 - np.mean((y_test-yhat)**2) / (np.std(y_test)**2)

kf = KFold(n_splits=5, shuffle=True)
score_ndarray = cross_val_score(DecisionTree_model, X_test, y_test.ravel(), cv=kf)
print('r2_score = ', score_ndarray)
print('r2_score_average = ', score_ndarray.mean())
```

```
r2_score =  [0.66757911 0.68832401 0.55077919 0.76002975 0.71067302]
r2_score_average =  0.6754770160867867
```

3. **Random Forest Regression Model**

   For Random Forest Regressor, it is a random forest regressor, and fits a number of classifying decision trees on various sub-subsample of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. This regressor will randomly choose features to predict, which will probably be a potential problem. It have a fast training spped of random forest algorithm, and random forest algorithms acan be applied to many type of model tasks.

```
# Module 3: RandomForestRegressor

randomForest_model = RandomForestRegressor()
randomForest_model.fit(X_train, y_train.ravel())

yhat = randomForest_model.predict(X_test)
R2 = 1 - np.mean((y_test-yhat)**2) / (np.std(y_test)**2)

kf = KFold(n_splits=5, shuffle=True)
score_ndarray = cross_val_score(randomForest_model, X_test, y_test.ravel(), cv=kf)
print('r2_score = ', score_ndarray)
print('r2_score_average = ', score_ndarray.mean())
```

```
r2_score =  [0.8588983  0.83115627 0.81851871 0.7828116  0.79474527]
r2_score_average =  0.8172260301876589
```

4. **Extra Tree Regression Model**

   ExtraTreeRegressor module is an extra-trees regressor, and this class implements a meta estimator that fits a number of randomized decision trees on various sub-samples of dataset. This estimator wouldn't pick features randomly, it will randonly collect part of feature and then use entropy information to find the best/most important sum-sample features.

```
# Module 4: ExtraTreesRegressor

ExtraTrees_model = ExtraTreesRegressor()
ExtraTrees_model.fit(X_train, y_train.ravel())

yhat = ExtraTrees_model.predict(X_test)
R2 = 1 - np.mean((y_test-yhat)**2)/(np.std(y_test)**2)
#print("R2: ", R2)

kf = KFold(n_splits=5, shuffle=True)
score_ndarray = cross_val_score(ExtraTrees_model, X_test, y_test.ravel(), cv=kf)
print('r2_score = ', score_ndarray)
print('r2_score_average = ', score_ndarray.mean())
```

```
r2_score =  [0.82908551 0.80194699 0.84114173 0.82166216 0.81319646]
r2_score_average =  0.8214065711450493
```

5. **Gradient Boosting Regression**

   We also use the GradientBoostingRegressor method to instantiate the model object. Then, use the model to predict yhat and use KFold method to instantiate cross validation object. Finally, use the cross_val_score method for cross validation. We judge the fitness of the model by calculating the value of R2. Usually, the closer the value is to 1, the better the effect.

```
# Module 5: GradientBoostingRegression

GradientBoostingRegressor_model = GradientBoostingRegressor(n_estimators = 400, min_samples_split = 2,
         learning_rate = 0.1, loss = 'ls')
GradientBoostingRegressor_model.fit(X_train, y_train.ravel())

yhat = GradientBoostingRegressor_model.predict(X_test)
R2 = 1 - np.mean((y_test-yhat)**2)/(np.std(y_test)**2)

kf = KFold(n_splits=5, shuffle=True)
score_ndarray = cross_val_score(GradientBoostingRegressor_model, X_test, y_test.ravel(), cv=kf)
print('r2_score = ', score_ndarray)
print('r2_score_average = ', score_ndarray.mean())
```

```
r2_score =  [0.86707119 0.86607006 0.83826359 0.85544069 0.7878716 ]
r2_score_average =  0.8429434267587963
```

## Cost performance evaluation

One of the meaning to do this project, since people all would like to buy a cost-effective house, it 's necessary to evaluate the house price according to the model, then we built a CP function, as long as input the basic features, then compare the predicted price with the actual price, it will tell you the cost performance with a specific value. In the future, with more data acquired and modify the model, we would get a more accurate evaluation.

```
def CP(x,yts):
    yhat = GradientBoostingRegressor_model.predict(x)
    m = (yts - yhat) / yts
    if(((yts - yhat) / yts) >= 0.05):
        print(f'it has a good cost performance,which is  {np.round(m*100,2)} %' )
    if(((yts - yhat) / yts) < 0.05 and (yts - yhat) > 0 ):
        print(f'it has a normal cost performance,which is {np.round(m*100,2)}% cheaper than expect' )
    if(((yts - yhat) / yts) < -0.05 and ((yts - yhat) / yts) > -0.1 ):
         print(f'it is cost performance is less than satisfactory, which is
               {np.round(m*100,2)}%  expensive than expect' )
    if(((yts - yhat) / yts) < -0.1 ):
        print(f'it is hard bargain, which is {np.round(m*100,2)}%  expensive than expect' )
CP( [[4,  2,  1.78000e+03,  7.35000e+03,  1.00000e+00,
  0.00000e+00,  5.00000e+00,  7.00000e+00,  9.00000e+02,  8.80000e+02,
   1.97400e+03,  4.74562e+01, -1.22158e+02]],280000)
```

```
it is cost performance is less than satisfactory, which is  [-6.4]%  expensive than expect
```

## Summary

According to the above experiment data, we first visualize the data set, and get the distribution map, correlation graph and thermodynamic diagram, then use function `detect_outliers` to detect and eliminate the extreme value.

After that, we reasearch and decide to use five different kinds of models to train and predict: `Linear Regression` model, `Decision Tree Regression` model, `Random Forest Regression` model, `Extra Tree Regression` model, and `Gradient Boosting Regression` model. After we use cross-validation and K-fold method, we find out that Gradient Boosting Regression model have the highest accuacry, which is close to 85%.

Finally, we creat a Cost Performance Evaluation function, it can predict and compare with the cost performance and house price, which is helpful for people who want to buy a cost-effective house. We hope our project can help more people when they are looking for a perfect home.