Utilizing Linear Regression for Residential Price Forecasting

Le Minh Hai
20225491
hai.lm225491@sis.hust.edu.vn

Trieu Kinh Quoc
20225524
quoc.tk225524@sis.hust.edu.vn

Vu Duc Minh
20225514
minh.vd225514@sis.hust.edu.vn

Tran Khoi Nguyen
20225453
nguyen.tk225453@sis.hust.edu.vn

Vu Quoc Dung
20225488
dung.vq225488@sis.hust.edu.vn

***Abstract*— In the developed world of real estate, the needs for prediction of house prices has become a crucial aspect for homeowners, investors, and industry professionals. The dynamic nature of housing markets, influenced by different aspects of economic, social, and environmental factors, necessitates the application of robust predictive models. As technology continues to advance, machine learning and data analytics have emerged as powerful tools to point out patterns, trends, and correlations within vast dataset. House price prediction, a subset of real estate analytics, plays an integral role in aiding decision-making processes related to property transactions, investment strategies, and urban planning.**

I. Introduction

As mentioned in the abstract, the prediction of house prices now plays a vital role in the real estate industry, as well as in financial planning and policy adjustments. Reliable house price forecasting provides valuable insights into market trends, which assists both buyers and sellers in formulating effective economical strategies.

In this project, we will explore the field of house price prediction using a widely-acclaimed approach - constructing a Linear Regression model - and present a comprehensive analysis of factors influencing house prices through various statistical methods, such as MinMaxScaler for data standardization, RFE (Recursive Feature Elimination) for feature selection, OLS (Ordinary Least Square) for calculating the cost function needed for estimating the parameters, etc.

By employing advanced machine learning techniques, we aim to propose a solution suitable for making informed decisions regarding house prices of the current market.

II. Material and methods

A. Sample

In the dataset we are using to train and evaluate the model, there are 1460 samples (n=1460), each contains 12 different features as followed:

- MSSubClass: The type of dwelling involved in the sale. It is a categorical feature that provides information about the building class. The values in this category represent various types of properties. The standardization of MSSubClass is listed below:

Value	Standardization
20	1-story 1946 & newer all styles
30	1-story 1945 & older
40	1-story w/finished attic all ages
45	1-1/2 story - unfinished all ages
50	1-1/2 story finished all ages
60	2-story 1946 & newer
70	2-story 1945 & older
75	2-1/2 story all ages
80	split or multi-level
85	split foyer
90	duplex - all styles and ages
120	1-story PUD (planned unit development) - 1946 & newer
150	1-1/2 story PUD - all ages
160	2-story PUD - 1946 & newer
180	PUD - multilevel - including split level/foyer
190	2 family conversion - all styles and ages

- MSZoning: The zoning classifications assigned to the properties in the dataset. Zoning classifications are regulations that define how land and properties can be used within a specific area or district, which can have implications for its usage, building restrictions, and potential value. The standardization of MSZoning is listed below:

Value	Standardization
RL	Residential Low Density
RM	Residential Medium Density
C (all)	Commercial
FV	Floating Village Residential
RH	Residential High Density

- LotArea: The area of the lot (square feet)
- LotConfig: Different configurations or positions of the lots where the properties are located. The values in this category can have implications for factors such as access, privacy, and overall desirability of the location. The standardization of LotConfig is listed below:

Value	Standardization
Inside	Lot is located inside a subdivision (typically surrounded by other properties within the same development.)
Corner	Lot is located at the corner of two streets
CulDSac	Lot is located at the end of a dead-end street
FR2	Lot is located adjacent to a frontage road
FR3	Lot is located adjacent to a third avenue

- BldgType: Different types or categories of buildings in the dataset. Each value corresponds to a specific type of building or dwelling, indicating whether it is a single-family house, a converted two-family dwelling, a duplex, or a townhouse. The standardization of BldgType is listed below:

Value	Standardization
1Fam	Single-family detached
2FmCon	Two-family conversion; originally built as one-family dwelling
Duplx	Duplex
TwnhsE	Townhouse end unit
TwnhsI	Townhouse inside unit

- OverallCond: Overall condition of the house (out of 10)
- YearBuilt: The construction year
- YearRemodAdd: The year of remodeling or additions (same as YearBuild if there is no remodeling or additions for the house)
- Exterior1st: The primary exterior covering material of the property
- BsmtFinSF2: The finished square footage in the basement area of a property (if the property has no basement area, this value is equivalent to 0)
- TotalBsmtSF: Total square feet of the basement area
- SalePrice: The sale price of the house

In our experimental methodology, we partitioned the dataset into two distinct subsets for the purpose of model development and evaluation. The initial subset, encompassing 70% of the total dataset, was exclusively employed for the construction and training of the predictive model. Following this, the remaining 30% constituted the secondary subset, reserved specifically for the assessment of the model's performance and generalization capabilities.

B. Algorithm

In this model, we implemented an algorithm called Linear regression. Linear regression is a statistical technique employed to model the relationship between a dependent variable (in this context, the sale price of a house, denoted as "y") and multiple independent variables (features), represented as $x_1, x_2, ..., x_n$. The model is expressed as:

$$y = w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n + b$$

In this equation, x_j denotes the value of the j-th feature, a_j is the corresponding coefficient, and b is the intercept term. The objective is to determine optimal coefficients (a_j) that best fits the training data.

To quantify the goodness of fit, a cost function, denoted as $J(w, b)$, is introduced. The cost function is defined as:

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^{m} y^{(\mathrm{i})}_{\mathrm{pred}} - y^{(\mathrm{i})}$$

In this equation, m stands for the number of training example $y^{(\mathrm{i})}_{\mathrm{pred}}$ is the predicted sale price for the i^{th} example, and $y^{(\mathrm{i})}$ is the actual sale price. The goal is to minimize $J(w, b)$ with respect to the coefficients a_j and the intercept b.

To achieve this minimization, the gradient descent algorithm is employed. Gradient descent is an optimization technique that iteratively adjusts the coefficients and intercept based on the gradients of the cost function with respect to these parameters. The process involves partial differentiation of the cost function, and updates are made in the direction that reduces the cost, ultimately converging to a local minimum. We start gradient descent by defining the initial coefficients a_j and the intercept b.

With the aim of having more visualized and general understanding, we sketch a graph:

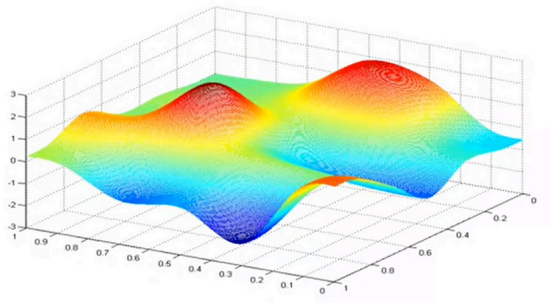

Figure 1: Cost Function J of an algorithm

Gradient descent is an iterative process. It originates from the initial parameters assign, and in each iteration, it systematically assesses the alterations in the cost function concerning all feasible directions. The critical principle of gradient descent is choosing the direction such that the sloop is steepest and move to new point towards this direction. From this point, the process recommences. This sequence of iterations happens until a point is attained where the change of the cost function approaches zero. It indicates the convergence of the algorithm to a point where the cost function attains a local minimum.

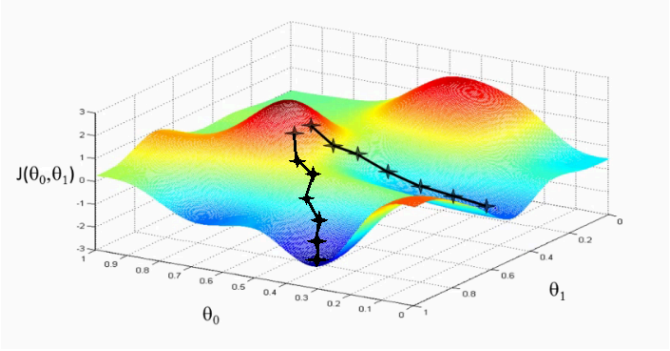

Figure 2: Usage of Gradient Descent

Gradient descent has a huge limitation as it may converge to local rather than global minima. However, in linear regression, the cost function is typically convex (bow-shaped). Therefore, the local minimum coincides with the global minimum. This feature makes gradient descent well-suited for linear regression optimization, despite its vulnerability to local minima in some different algorithms.

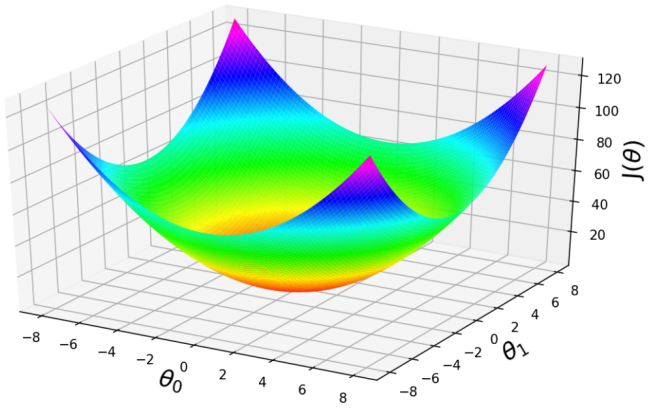

Figure 3: Graph of cost function J of Linear Regression

In terms of mathematics, we have the formula of gradient descent in each iteration:

$$\theta^{(\text{next step})} = \theta - \eta \bigtriangledown_\theta J(w,b)$$

In this equation, θ stands for the current position, $\theta^{(\text{next step})}$ stands for the next position, η stands for the learning rate and $\bigtriangledown_\theta f(\theta)$ is a gradient term that defines the direction of gradient descent. Applying this equation to each feature, we have:[1]

$$w_i = w_i - \eta \frac{\partial J(w,b)}{\partial x_i}$$

$$b = b - \eta \frac{\partial J(w,b)}{\partial b}$$

In this equation, the learning rate η determines how big the gradient descent steps. Therefore, it determines the speed in which the algorithm moves towards the optimum values of the cost function.

After fitting a linear regression model, in order evaluate the efficiency of our linear regression model, we use a term which is called R^2. R^2 is a measurement for the fitness of linear regression models according to our dataset. R-squared measures the strength of the relationship between your model and the dependent variable on a convenient 0 – 100% scale. The larger the R^2, the better the linear regression fits our dataset. We define:

$$\text{RSS (Residual Sum of Squares)} = \sum_{i=1}^{n} \left(y_{\text{pred}}^{(\text{i})} - y^{(\text{i})}\right)^2$$

$$\text{SST (Total Sum of Squares)} = \sum_{i=1}^{n} \left(y^{(\text{i})} - \overline{y}\right)^2$$[2]

The formula of R^2:

$$R^2 = 1 - \frac{\text{RSS}}{\text{SST}}$$

The linear regression model is said to fit our dataset if R^2 is more than 0.5.

Beside the use of R^2, we also use a term called *VIF*. *Variance Inflation Factor (VIF)* is a measure used to detect multicollinearity in regression analysis. Multicollinearity refers to a situation in which strong linear relationships between

[1] We simultaneously update the value of $x_1, x_2, ..., x_n, b$ after each iteration

[2] $\overline{y}$: the mean value of $y^{(\text{i})}$

independent variables are present. Multicollinearity occurs when predictor variables in a regression model are highly correlated with each other.

VIF measures how much the variance of an estimated regression coefficient is increased due to multicollinearity in the model. In the context of predicting house prices using a linear regression model, *VIF* can help identify if certain predictor variables are highly correlated with each other.

The formula of *VIF*:

$$\text{VIF}_i = \frac{1}{1-R_i^2}$$

VIF_i is the *Variance Inflation Factor* for the i^{th} predictor variable

R_i^2 is the R^2 value obtained from the regression of the i^{th} predictor variable against all the other predictor variables in the model.

There are general guidelines that are often used to assess the level of multicollinearity based on *VIF* values:

VIF values in range 1 to 5 are considered acceptable and suggest low levels of multicollinearity. Some analyses might even consider *VIF* values below 4 or 2 to indicate minimal multicollinearity.

VIF values between 5 and 10 may indicate moderate multicollinearity. While it's not extremely alarming, it could suggest a potential issue with multicollinearity that might need to be addressed, especially if it's on the higher end of this range.

VIF above 10: *VIF* values above 10 are often considered problematic and indicate a high degree of multicollinearity. In such cases, it's crucial to address this issue before interpreting the regression coefficients or making predictions from the model.

C. *Model building procedures:*

Below is the process we use to build our model:

1) *Import the necessary libraries:*

Numpy: Numerical computations, scaling numerical variables and array operations.

Pandas: Data manipulation & analysis. This library provides the DataFrame data structure, a two-dimensional table-like object that stores data in rows and columns and reads the dataset in .xlsx format into a pandas DataFrame.

Scikit_learn: Split the data into training and testing datasets. Except from that; this library is also used for data preprocessing, feature selection, model training, and evaluation.

Matplotlib, Seaborn: Data visualization (used to examine visual implementation of data distributions and relationships between different features in the dataset)

2) *Data preprocessing:*

Data Cleaning

After the housing dataset is loaded from a *.xlsx* file and stored in the *housing* dataframe, the first thing we have to do is check for the *Null* value. *Null* values are those that are absent and undefined in the dataset and make our model unstable. Afterwards, we examine and analyse the distribution patterns inherent in the dataset's features.

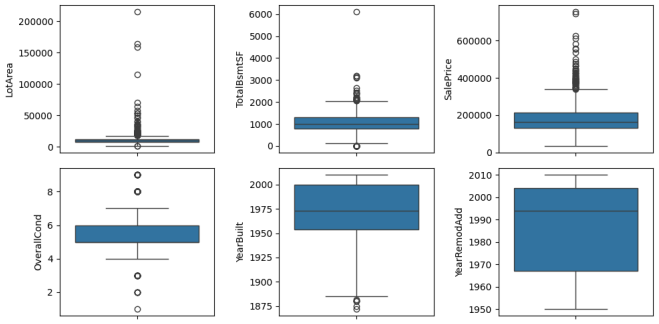

Figure 4: Distribution of features before removing outliers.

Outliers values in the *"LotArea"*, *"TotalBsmtSF"*, and *"SalePrice"* columns of the housing dataset are removed using the interquartile range *IQR* method.

IQR method is a statistical technique used to identify outliers in a dataset. It involves calculating the range between the first quartile (25%) and the third quartile (75%) of the data. The *IQR* is calculated using the **quantile()** method in Pandas. This method is used to specify the desired quantile as an argument. To calculate the first quartile (*Q1*), we would use **quantile(0.25)**, and for the third quartile (*Q3*), we would use **quantile(0.75)**. Once the first quartile (*Q1*) and third quartile (*Q3*) are computed, the IQR is obtained by subtracting Q1 from Q3:

$$\text{IQR} = \text{Q3} - \text{Q1}$$

From this, we erase all the rows that have this feature with the value *a* of this feature such that:

$$a < \text{Q1} - 1.5\ \text{IQR} \tag{1}$$

Or

$$a > \text{Q3} + 1.5\ \text{IQR} \tag{2}$$

We apply *IQR* method with each feature *"LotArea"*, *"TotalBsmtSF"* and *"SalePrice"*.

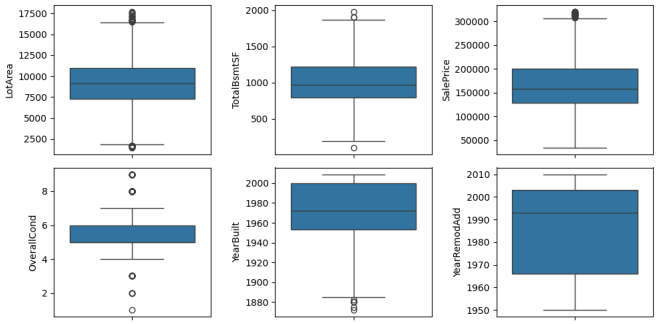

Figure 5: Distribution of features after removing outliers.

Data Preparation

There is also some *object* variables in the dataset, namely "MSZoning", "LotConfig", "BldgType", and "Exterior1st". They are converted into numerical representations using ***get_dummies*** function from Pandas, a convenient way to perform one-hot encoding on categorical variables which creates binary columns for each category and assigns a value of 1 if the category is present for a particular data point, and 0 otherwise. This helps change all the values of *object* type into *int* type; facilitate the use of *Linear Regression*.

Data scaling

The numerical values, namely "MSSubClass", "LotArea", "OverallCond", "YearBuilt", "YearRemodAdd", "TotalBsmtSF", and "BsmtFinSF2", are scaled using the MinMaxScaler from sklearn.preprocessing module.

The MinMaxScaler method is a commonly used scaling technique in data preprocessing. It works by transforming the numerical variables so that they are scaled to a specified range, typically between 0 and 1.

Operations of MinMaxScaler begins by calculating the minimum and maximum values of each numerical variables in the dataset. It then scales the data using the following formula:

$$x_{\text{scaled}} = \frac{x_{\text{original}} - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

The scaled values are then obtained for each variable, ensuring that the minimum value becomes 0 and the maximum value becomes 1.

When all variables are scaled similarly, models that are sensitive to the scale of the input features perform better. Because variables with higher values have the potential to dominate the model and produce biased results, it makes sure that every feature contributes equally to the model training process. Also, data scaling prevents the huge step sizes brought on by unscaled features, which aids in the faster convergence of gradient descent algorithm.

Splitting the dataset

The dataset is split into training and testing datasets using the ***train_test_split*** function from *sklearn.model_selection*. Afterwards, the variable *"SalePrice"* is separated from the training dataset and stored in the *y_train* variable. This variable will be used as the dependent variable in the regression analysis.

3) *Feature Elimination (RFE):*

We select the most important features for predicting the target variable. RFE uses a specified machine learning algorithm (in this case, linear regression) and iteratively eliminates the least important features based on their ranking. In the code, the number of features to select is set to 12, and the selected features are stored. This method is used in machine learning to improve model performance, reduce overfitting, and enhance interpretability. The operation of RFE, based on a recursive system, is explained below:

- Starts by training a model on the entire set of features. The importance or relevance of each feature is then evaluated based on a specific criterion, in this case, the coefficients in a linear model.
- The model will randomly choose 1 feature to eliminate, then retrain the data using the newly-acquired set of features. It repeats the process with all features in the dataset, then compute the significance of how each feature individually affects the prediction.
- The least important features, according to the criterion, are eliminated from the feature set. This reduction step can be performed in different ways, either removing a fixed number of features or removing a percentage of the least important features.
- The model is retrained on the reduced feature set, and the process repeats. The importance of features is re-evaluated, and the least important ones are eliminated in each iteration.
- Throughout the iterations, RFE ranks the features based on their importance or relevance. The final ranking reflects the relative importance of the features in improving the model's performance.

4) *Apply Linear Regression:*

The selected features are used to train an ordinary least squares (OLS) regression model using the *Statsmodels* library. OLS is a statistical method for estimating the parameters of a linear regression model. Afterwards, we apply the ***LinearRegression()*** method. This method is used to build a predictive model for the house sale prices. The operations of this method is listed below:

- Model Initialization: Initializes a linear regression model by creating an instance of the LinearRegression class and assigning it to a variable. This model will be used to fit the data and make predictions.
- Fitting the Model: The model is fitted to the training data using the ***fit()*** method. It is trained using Linear Regression algorithm with regard to *x_train* and *y_train* dataset
- Prediction: Once the model is trained, predictions are made. In the code, the model predicts the sale prices for the training data x_train by using the ***predict()*** method.
- Get parameters w_j and b: To get the Linear Regression equation of our model, we utilize the method ***params()***. By this way, we obtain the equation that best fits with our *x_train* and *y_train* dataset.

D. Statistical analysis:

When we completed building our model, it raises the need for examining the linear parameters and evaluating the efficiency of our model.

First, we calculate the parameters for the linear regression model. As stated in the *Algorithms* subpart, our model relies primarily on a linear equation in the form of $y = \sum_{i=1}^{n} a_n x_n + b$. The parameters are computed through ***lm.params*** of the LinearRegression() module. The calculated values are as followed:

Area	**Parameters**
const	−121794.150146
MSSubClass	129266.836252
LotArea	60163.212623
OverallCond	38341.117570
YearBuilt	78073.823394
YearRemodAdd	31507.900763
BsmtFinSF2	−21676.654081
TotalBsmtSF	158323.020366
C (all)	−28920.690942
1Fam	72980.321591
2fmCon	−36168.913616
BrkComm	−82826.390141
Stone	63494.312478

Next, as we have mentioned before, we need to check the *Variance Inflation Factor* (*VIF*) of each feature choosen by *RFE* to avoid multicollinearity. The model is said to have no multicollinearity if for each feature *j*:

$$\text{VIF}_j < 5$$

In terms of our model, the *VIF* for 12 features choosen by *RFE* is:

Area	**VIF**
const	109.40
MSSubClass	4.63
LotArea	1.61
OverallCond	1.35
YearBuilt	2.42
YearRemodAdd	1.84
BsmtFinSF2	1.05
TotalBsmtSF	1.79
C (all)	1.04
1Fam	4.45
2fmCon	1.53
BrkComm	1.01
Stone	1.03

Table 1: VIF values for each selected features

Then, we apply the *r2_score* evaluation metric to measure the goodness of fit of a regression model. It calculates the coefficient of determination, also known as R-squared (R^2), which quantifies the proportion of the variance in the dependent variable that can be explained by the independent variables in the model.

The R^2 value ranges from 0 to 1, where 0 indicates no explanatory power of the independent variables and a poor fit to the data, while 1 indicates full explanatory power and a perfect fit to the data.

Because of that, we want this value to be as close to 1 as possible, which indicates that the model built by the algorithm expresses a good fit for the dataset. The value is calculated by importing *r2_score* module from sklearn.metrics library. The *r2_score* of our model is the value below:

$$r2\_score = 0.7044846667824867$$

The value is aprroximately 70.04 %, which means 70,04 % of the dataset fit in well with our regression model, thus, the model can be utilized for predicting unknown data.

To evaluate the performance of the linear regression model, the code calculates the residuals, which represent the differences between the predicted sale prices (*y_train_price*) and the actual sale prices (*y_train*).

III. Discussion

A. Link to the code

For further information about our works and submission, please visit our Github repository: Github

B. Issues and difficulties

During the project preparation, we have learnt how to use the model and apply it to our study; however, we have to deal with two main and biggest problems which are handling the outliers of the data and choosing the number of features needed in the model.

In the data processing part, *IQR* method is used to remove outliers of the datasets. In our model, this method is apply with each feature "*LotArea*", "*TotalBsmtSF*" and "*SalePrice*". However, the feature "*BsmtFinSF2*" also has a lot of outliers:

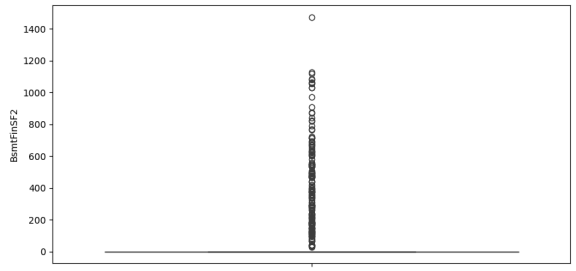

Figure 6: Distribution of BsmtFinSF2 before removing outliers.

In the first place, we tried to remove the extreme values of this feature, but the distribution of "*BsmtFinSF2*" after cutting was a straight line.

Figure 7: Distribution of BsmtFinSF2 after removing outliers.

In this case, the outliers contain critical information, they represent rare occurrences or unexpected events that are essential for modeling. Therefore, if we remove them, it leads to the loss of important information and causes errors in our model.

Next, in linear regression modeling, the choice of the number of variables can significantly impact the model's performance and characteristics. Having too few variables can lead to an oversimplified model (underfitting), which might not capture the complex relationships between the input and output. Conversely, having too many variables might lead to overfitting, where the model performs well on the training data but fails to generalize effectively to new data.

The decision to opt for 12 variables is rooted in their influence on the Variance Inflation Factor (VIF), a measure indicating multicollinearity among predictors in a regression model. By selecting this specific number of variables, we aim to mitigate issues related to high multicollinearity, therefore improving the stability and reliability of the regression coefficients. The consideration of 12 variables is strategic as it strikes a balance between incorporating adequate information into the model while avoiding excessive redundancy that might inflate VIF values, ultimately ensuring a more robust and interpretable linear regression model.

IV. Conclusion

In conclusion, by analysing data features, using linear regression model, we have successfully forecasted house price. Through this scientific report, we delved into the methodology of utilizing linear regression to analyze various features and their impact on housing prices. The model's performance in predicting prices was assessed, demonstrating its ability to approximate house values based on factors such as house types, area, zoning type, etc. However, certain limitations within the model make it unreliable for accurately predicting real-life house prices, so, we are continuing research and learning in order to complete our study and make it applicable for real estate prediction.

Acknowledgements:

This work is supported and supervised by professor Than Quang Khoat under the course of *Introduction to Artificial Intelligence*. Also, our research and project are partially supported by other students at School of Information and Communication Technology, Hanoi University of Science and Technology.

Data availability:

Publicity available datasets were analysed in this study. This data can be found here: HousePriceDataset

References:

[1] Stanford CS229: Machine Learning - Linear Regression and Gradient Descent | Lecture 2 (Autumn 2018)

[2] Course: Supervised Machine Learning: Regression and Classification *by* DeepLearning.AI & Stanford University

[3]https://ndquy.github.io/posts/gradient-descent-2/

[4]https://www.investopedia.com/terms/r/residual-sum-of-squares.asp

[5]https://en.wikipedia.org/wiki/Coefficient_of_determination

[6]https://builtin.com/data-science/gradient-descent

[7]https://www.freecodecamp.org/news/gradient-descent-machine-learning-algorithm-example/

[8]https://www.kaggle.com/code/ashydv/housing-price-prediction-linear-regression (Code reference)