In the Final Project of IE343, the task was to predict housing prices in Seoul and Busan in 2023 using data from 2018 to 2022, as well as additional data on daycare centers in each gu and various parks in each dong. The submissions were evaluated using mean average error, and the baseline to pass was 107117.26662.

To pass the baseline and then to lower the error, I implemented some data preprocessing, feature engineering and model selection techniques. These techniques are listed in chronological order of their implementation, and the detailed description along with the score are listed in this report.

1. Splitting task into 2: predicting for Seoul and Busan separately; increasing number of predictors. (Score: **107107.60558**)

   The first technique I implemented was to build additional models for Seoul and Busan. This is a common technique in machine learning, where the data can be split into few groups, and train a new model on each single group. This allows for the model to learn patterns in data which are unique to each group, as well as fine-tune the model itself on each group, which lowers the error.

   At this stage, I used the baseline code directly with data splitting and increase in number of predictors, no additional hyperparameter tuning was done.

2. Model selection: CatBoost (Score: **66632.16996**)

   For further improvement, I decided to use gradient boosted trees, as they have a proven record of superior performance on tabular data [1]. I chose CatBoost as it can natively handle categorical variables, missing values and has good initial hyperparameters. By increasing iteration count, I was able to get my score down to 66632.16996 with some basic hyperparameter tuning using CatBoost's built-in grid search capabilities.

3. Combining datasets with feature selection (Score: **63232.12303**)

   After selecting the model, I combined all 3 datasets and created new features. From park dataset, I created total number of parks, average area and average opening year for each type of park for each dong. From the daycare dataset, I created average number of babies, teachers, nursing rooms, playgrounds, CCTVs and commuting vehicles for each type of daycare for each gu. Gu information is linked to dong information using the park data set. After combining datasets, handling missing values and inconsistent entries, I trained the model to see feature importances to iteratively select top features, to introduce sparsity.

   Also, at this stage I tried various feature engineering. Initially, I used pair plots to see if features are correlated with each other.

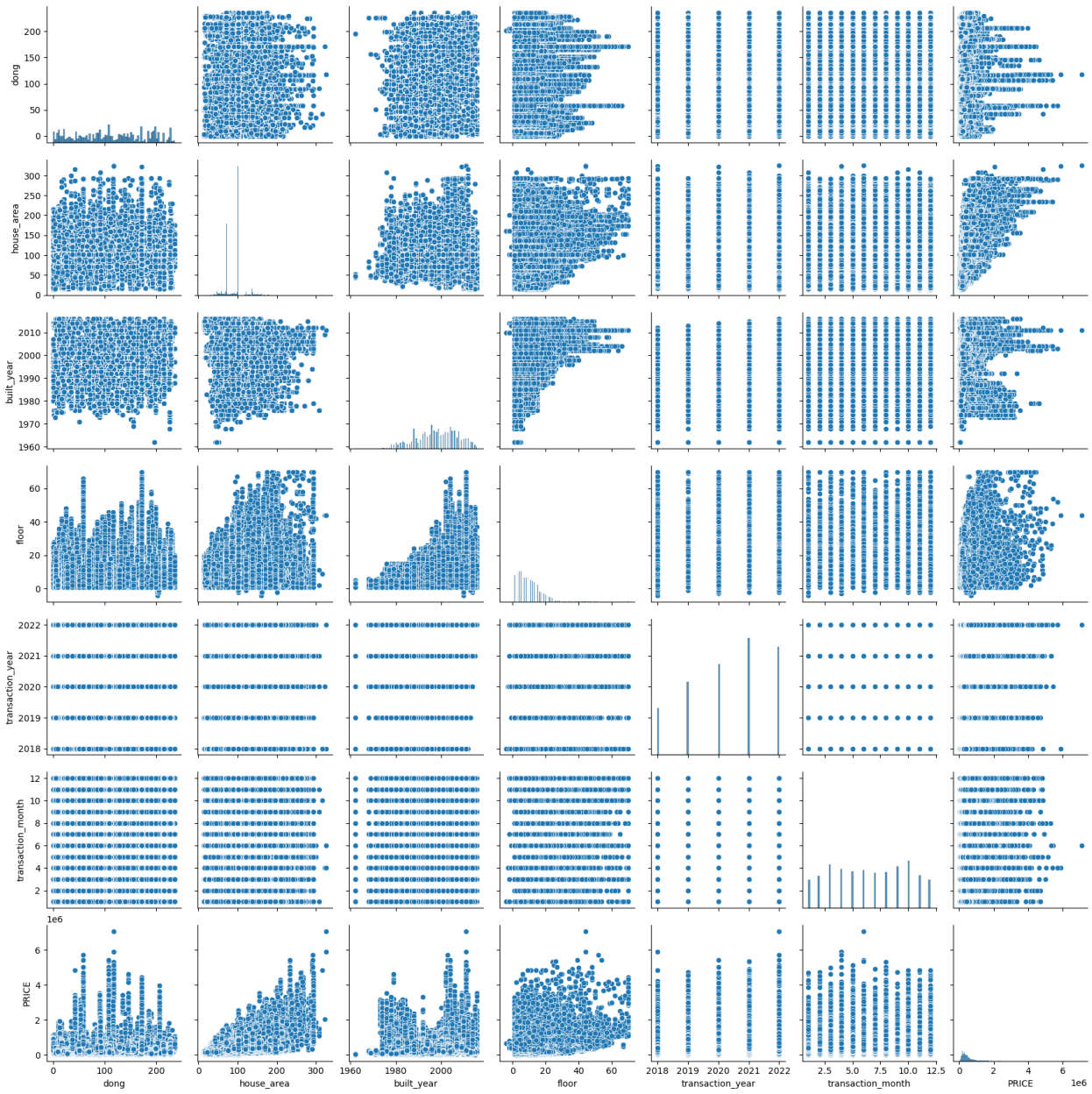

Figure 1. Pair plot of columns of original train file

As we can see, some features do have some correlation between them, which can be used to build interaction terms. The selected interaction terms are log(floor/area) and log(year/floor). The meaning of these interaction terms is quite evident, as houses with more floors tend to have more area and newer houses tend to be bigger and larger, i.e. with more floors. The interactions are designed to have these mathematical forms to yield distributions close to normal distribution. Also, the price column is log-normalized to fit the normal distribution.

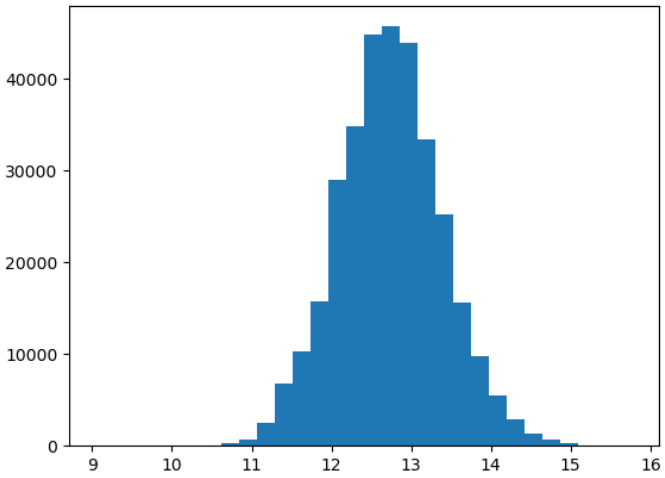

Figure 2. Log-normalized price values

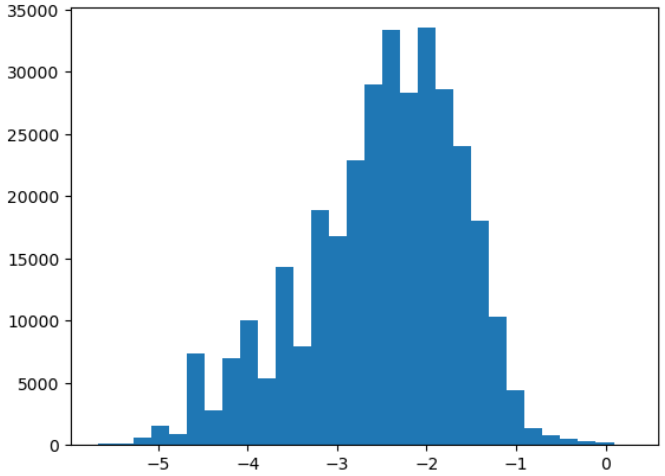

Figure 3. log(floor/area) values

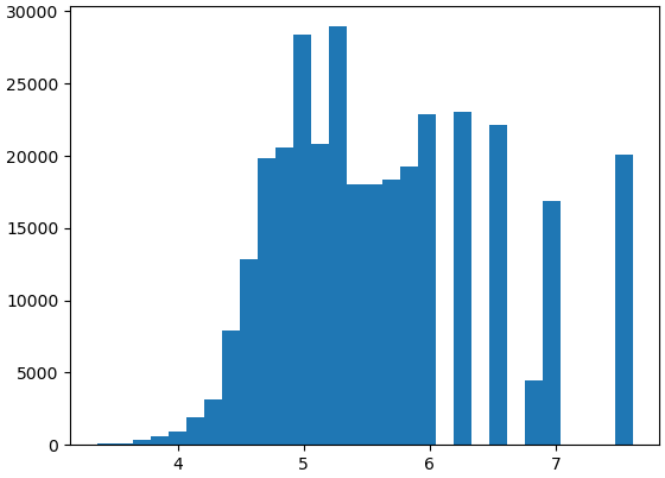

Figure 4. log(year/floor) values

One other major feature engineering uses longitude and latitude data to calculate x, y, z coordinates, as shown below.

$$x = \cos(lat) \times \cos(long)$$
$$y = \cos(lat) \times \sin(long)$$
$$z = \sin(long)$$

These formulas are taken from the original DACON competition's top-contestant notebook.[2] All these feature engineering allowed the score to go down to 63232.12303.

There were some challenges with the combined dataset. First of all, the "is_commuting_vehicle" column in daycare data has inconsistent entries, and in the park dataset, there were duplicate dongs across different gus. The first challenge was solved by creating a dummy variable, and the second challenge was solved by simply dropping the duplicates. The solution to the second challenge was deemed acceptable as the number of duplicates was low. Also, all park and daycare types except the popular 3 types were binned into a "etc" category to reduce data dimensionality. The missing values for coordinates were filled by the mode of the dong, or by the mean coordinates of the city if the whole dong had null values for coordinates.

4. Consideration for inflation (Score: **37634.23607**)
   The biggest decrease of score happened after I accounted for inflation at the very end. The implementation of this consideration was very basic, as I simply increased the predicted values for Seoul prices by 12% and predicted values for Busan prices by 11%. These values were determined experimentally by iteratively substituting various multiplying coefficients. The presumed reason on why this multiplication has worked so well is that the prices generally increase each year, not only due to supply and demand changes, but also due to simple inflation. This increase can be accounted by "year" feature only, however, as we can see, the inflation effect is not properly accounted for by the "year" feature in my models, which triggers the need for additional increase of the final output.
5. Hyperparameter tuning using Optuna and ensemble building (Score: **36319.38439**)
   I was able to further increase my results by combining the CatBoost model with other gradient boosted tree models, namely XGBoost and LightGBM. CatBoost model used was tuned in previous sections, and XGBoost and LightGBM were iteratively tuned using Optuna on GPU, which significantly decreased training time. The ensembling was done using Excel by simply averaging the output files of each model. The resulting output files were used for final Kaggle submission, which were my best results

The implementation described herein is full of shortcomings and brute-force solutions to some of them. The results can be significantly be improved by choosing a different implementation, more suitable for this task. One obvious choice would be to use time-series models, which are specifically built for tasks like this. Also, more thorough investigation into the data itself, which will lead to better data preprocessing and better and more powerful feature engineering, which can significantly improve the accuracy. By doing these fundamental changes to the

implementation, we can get rid of brute-force solutions such as the final multiplication by an arbitrary coefficient.

References:

[1] https://arxiv.org/abs/2106.03253

[2] https://dacon.io/competitions/open/21265/codeshare/741