BE903 Assignment 3

August 2024

Student ID: 95788

Housing Market Predictions

Table of Contents

Executive Summary

This report details the validation and refinement of the chosen predictive model for property sell prices in Stockholm. Upon testing multiple models and engineering combinations, it is recommended that the company use the LightGBM Regression for their housing prediction process.

To arrive at this conclusion, data cleaning, feature engineering, model comparisons, hyperparameter tuning, and model validation techniques were undertaken to ensure robust predictions and generalizability. In case of the LGBM determined as not ideal, the second model of choice is the Neural Networks (NN) model. The LightGBM Regressor was determined to be the champion model after thorough comparisons with baseline models such as Multiple Linear Regression (MLR), a Neural Network model, and an exploratory XGBoost model to establish LGBM's effectiveness by comparing LGBM to that of its close counterpart.

Ultimately, the LightGBM outperformed these alternatives, achieving a Mean Absolute Error (MAE) of 313733.53 during K-Fold Cross Validation, closely matching the training MAE of 311242.57. The model was resilient against extreme values and maintained its predictive accuracy across various validation techniques.

Feature Engineering and Tuning

In order to finalize the LGBM, considerable feature engineering was undertaken. For example, the exclusion of features with negligible predictive value, categorical encoding, creating ratio-based features and aggregation of metrics were few of the methods chosen. These efforts led to a more refined model with a significant reduction in the MAE by about 20,000 units.

Hyperparameter tuning was conducted through both GridSearch and manual adjustments, resulting in a substantial reduction in MAE. To prevent overfitting, the model was validated through cross-validation and regularization techniques, ensuring that it generalizes well to unseen data.

Robustness and Limitations

The LightGBM model was robust in handling extreme values, with minimal impact on MAE, reaffirming its selection as the champion model. However, challenges such as handling missing data through mean imputation, model drift due to changing input trends, and potential issues with underrepresented property types (e.g., row houses) were areas that were found to be needing further improvement

Key Recommendations

It is recommend that the company:

1. Looks at exploring alternative methods for handling missing data to enhance model accuracy, as considerable imputation was done to ensure data consistency,
2. Reassesses feature engineering processes with the focus being on optimizing the inclusion and transformation of features like locality, postcode, street names.
3. Develops a new metric for 'property_age', as the dearth of data from previous years would have differing predictive powers, and thus must be adjusted appropriately.
    a. Implementation of a 'weighted property age', or the exclusion of certain years from the dataset upon isolating their predictive power can be good alternatives to combat ineffective predictions.

4. Continuously monitors the model for drift, establish monitoring with KPI threshold values, recalibration protocols, and regularly communicate progress with stakeholders.

The report below details the process of model development, the specific decisions made to improve model accuracy, and the process underlying model selection, validation, and hyperparameter tuning.

Section 1: External Validation

The champion model selected was the LightGBM Regressor, which was tuned to make the final predictions. External validation for this final model was carried out in four different ways:

1. K-Fold Cross Validation
2. Comparison with existing models
3. Adversarial Testing with extreme values
4. Prediction analysis

The K-Fold Cross Validation yielded close Mean Absolute Error (MAE = 313733.53) results with the training MAE (311242.57), verifying that the model performed similarly and will have roughly the same performance when tested with unseen data (Table 1).

Compared with the existing models, the baseline Multiple Linear Regression (MLR) model and the Neural Network (NN) model both yielded higher MAEs and MAPEs (Table 2, Figure 1). It was decided that an XGBoost model will be employed for comparison purposes and also out of curiosity, to test if the tree-based learning structure would perform better than LGBM, but found that XGBoost's limitations with handling large data made it perform worse than the LGBM (Table 3).

The LGBM was also tested with outliers (Figure 3) with an iteratively increasing proportion of outliers, and the MAE obtained was closely similar to the MAE of the previous training. Thus, this shows that LGBM models are robust and can handle extreme values.

Additional precision scores were also studied, such as a histogram of residuals and error, and a Q-Q plot (Figures 3-5). The residuals had a normal distribution, and error values concentrated at 0. The Q-Q plot for predictions also revealed that the distributions were normal with a slight positive skew.

Finally, the predictions on unseen data were also analyzed. First, distributions were studied between actual vs predicted sell price (Figure 6-7, Table 4). The distributions were similar, and centered around the mean; these proved that the prediction values were in line with standard expectations, affirming the performance of the model. Then, the previous submission's performance was compared with the new predictions to analyze differences, and it was found that the old submission contained an unusually large MAE (R2 was used in the previous submission so it was run again to

obtain MAE values), and as a result the prediction distributions were incredibly skewed (Figure 8, Table 5).

Section 2: Feature Engineering

2.1. Feature Selection Process

The feature selection process for the predictive model began with a preliminary analysis of the datasets. This analysis revealed that the categorical feature '*object_type*' displayed a highly imbalanced distribution, with the majority of instances representing apartments, while *row houses* (10 instances) and *townhouses* (1 instance) are significantly underrepresented. Given this imbalance, it was determined that '*object_type*' would likely offer limited predictive value and excluded it from the model, as features with disproportionate representation can skew model outcomes and introduce bias.

To ensure a logical and systematic approach to feature selection, the remaining features were categorized into distinct groups (*Table 6)*. This categorization, while partly influenced by domain knowledge, served as a means of organizing the data for clearer focus. Although this step was largely heuristic, it laid a foundation for a more structured analysis.

- **No correlation**: Features in this category are excluded due to their negligible correlation with the target variable, following the principle that non-predictive features should be omitted to enhance model efficiency.
- **Basic**: These features are selected based on their fundamental relevance to housing characteristics, providing a baseline understanding of property attributes.
- **Locality**: Geographic and locational features are incorporated to capture the spatial impact on property values, a factor Ill-documented in real estate analytics.
- **Area + Cost**: Features related to property size and cost are deemed essential due to their direct correlation with market valuation, as supported by empirical evidence in housing economics.
- **Amenities**: These features are considered crucial for reflecting the desirability and functionality of the property, which are key determinants of market price.
- **Time**: Temporal features are included to assess the influence of construction date and market trends on property values.
- **Taxes/Costs**: Financial metrics are included to provide insights into the fiscal obligations and economic status associated with the properties, factors that significantly influence buyer decisions.

2.2. Additional Feature Engineering

In the data preparation process, additional feature engineering was done to enhance the predictive capabilities of the model. Specifically, the following techniques were implemented:

1. **Categorical Encoding with Target Encoding**: Target encoding was applied to categorical identifiers such as 'housing_coop_id', 'agency_id', and 'housing_association_org_number'.

Since traditional encoding methods like one-hot encoding may not be effective due to the potential for dimensionality explosion and sparsity this encoding replaces categories with a function of the target variable, which can lead to improved model accuracy.

2. **Ratio Features**: Ratio-based features were generated, such as 'asking_price_ratio', calculated relative to the median asking price. By standardizing prices in this manner, it is possible to reduce the impact of outliers and heteroscedasticity.

3. **Aggregated Metrics**: Various features were also aggregated to create composite metrics like 'total_units' and 'total_area'. By summing different types of areas and unit counts, the overall scale and scope of housing units can be captured. Significant insights for predictive tasks can thus be obtained by capturing the totality of a property's attributes in a single feature.

2.3 Visualization and Encoding

To visualize the relationships among these features, a correlation matrix was plotted along with correlation plots (Figure 9 - 10), which helped in identifying multicollinearity and redundancies. Following this, Pearson and Spearman correlation tests were conducted and results analyzed with their respective p-values (Table 7). These statistical tests helped quantify the strength and direction of relationships between features and the target variable. Additionally, a segment of code was developed to plot feature importance, setting a threshold of zero to specifically isolate features with non-zero importance (Fig. 11 (a), (b)). This approach helped to iteratively refine the feature set by focusing on variables that consistently demonstrated predictive power across various models, including the final tuned versions.

For features that aren't straightforward numbers, both categorical and numerical data was converted into the same format, and found that it was useful to combine binary data scores into a cumulative score based on a shared characteristic to allow for easier understanding of which features played a greater role in determining sell price. *One-hot encoding* and *label encoding* as well as some embeddings were used to allow for this.

2.4. Imputation and Standardisation

In order to handle missing values and sparse data the mean imputation method was employed. An indicator which showed if certain values were missing was also created. Ultimately, there were too many empty cells in the dataset and while mean imputation may not be a perfect fit, it was better than scrapping whole rows or whole columns of data. Then, standardization of scale allowed for proper comparison. The number of NaNs in each column was studied to ensure data had not been mishandled. This way, sparse data was filled in as necessary, and the standard scaler which was implemented is in Figure 12.

2.5. Addressing Limitations

Despite the recognized limitations of mean imputation, particularly in scenarios where the assumption of data missing at random (MAR) may not hold, it was determined that it was a preferable alternative to more aggressive data reduction techniques, which could have

compromised the robustness of the analysis. The dataset was continuously monitored for the presence of NaN values, particularly after scaling, to ensure that the integrity of the data was maintained, as mishandling missing data during preprocessing could have led to distortions in the upcoming analysis.

In addition to mean imputation, a data cleaning process was undertaken that addressed sparsity by filling in missing values where necessary. This is because the quality of the data directly influences the accuracy and reliability of the predictive models, and this approach ensured that the final dataset was well-prepared for the next stages of analysis.

Moreover, measures were taken to ensure metrics from the Annual Report dataset for Housing Associations were taken into account to evaluate risks. While it is indeed hard to balance it against the need for easy interpretability, it is required to adapt the model for company requirements. We also then decided to take into account risk features such as debts, tax payments, and savings that would play a role in the prediction models (Figure 16).

2.6. Creating the Final Dataset

Merging the Apartment data with the Housing Association data was straightforward, as both datasets shared a common identifier—‘*housing_association_org_number*’ in the Apartment data and ‘*org_number*’ in the Housing Association data. However, the process for integrating Annual Reports data presented additional challenges. Before merging, it was important to ensure that all data points were numeric, which required the conversion of any erroneous or non-numeric columns. Once the data was standardized and validated, aggregation techniques were implemented to summarize the data points, using the grouping function to consolidate information by ‘*housing_association_org_number*’ and ‘*org_number*’. This allowed us to harmonize the varying levels of granularity within the Annual Reports by one to many mapping (Figure 13).

Section 3: Model Selections

3.1. Baseline Model

The baseline model used for comparison was Multiple Linear Regression. The choice of Linear Regression as a baseline is because it’s widely used as a benchmark since it gives a quick indication of the 'minimum' expected performance; any more sophisticated model should ideally perform better than this baseline to justify its complexity. Additionally, Linear Regression's results are easy to interpret.

Thus, MLR was chosen as a base model due to its simplicity and interpretability. An MLR with DASK was employed on all features to get an estimate of baseline performance. The MSE was 2757734170624.0, while the MAE was found to be 1065121.5 (Figure 1, Table 2).

3.2. Champion Model

The Champion Model was the Light Gradient-Boosting Machine. The rationale behind employing the LGBM model is as follows:

(a) It is fast and uses low levels of memory by replacing continuous variables to discrete bins, which is Ill-suited to the dataset,
(b) Has methods to directly handle categorical features (particularly relevant), built-in feature importance, and a handy way to regularize and set custom loss functions, and
(c) LGBM works the best for extensive feature engineering and can handle the high-dimensional nature of the dataset, which, combined with its scalability, means it is possible to circumvent sampling of the training data into chunks, saving time and resources.

The model was refined five times, first running the untuned baseline LGBM, then running LGBM with important features alone, LGBM with hyperparameter tuning employed using GridSearch, LGBM with manual hyperparameter tuning and important features, and finally, performing a k-fold cross-validation to prevent overfitting.

3.2.1. Performance Increase

The baseline LGBM had a Test MAE of 406513.06, which was first minimized by condensing the feature list, resulting in an increased Test MAE of 408872.7. After employing a grid search with 24 hyperparameters and manually tuning the model, the next iteration had a Test MAE of 318746.11, which is a substantial decrease from the initial run. All model iteration results are in a combined table below. In the previous submission in May, the final MAE was much higher, at around 45,000, and this seems to be a decent decrease (Tables 8; Figures 3-5).

3.2.2. Comparison with XGBoost

Out of curiosity, the LGBM model was compared with the XGBoost model. This was done due to the algorithmic differences between the two models, and their divergence in handling categorical variables. XGBoost uses a pre-sort-based approach and grows tree-wise, while LGBM uses a histogram-based approach and grows leaf-wise. Thus it was necessary to determine if these algorithmic differences translated to differences in predictions.

And indeed, the XGBoost model, known for its inefficiencies in processing large tabular data, performed worse than the LGBM, with an MAE of 354945. Even after employing tuning, the lowest possible score obtained through the process was 344226.98, which is higher than LGBM (Tables 17-18). Thus, the decision was made to retain LGBM as the champion model, while XGBoost, a closely similar machine learning approach, was undertaken to eliminate potential oversights in making this decision.

3.3. Champion Challenger Model

The final challenger model was the Neural Network model, which was implemented after scaling, imputation, and feature engineering.Their ability to learn relationships seemed ideal for the project, and they also performed very well, as seen below. They had 11 epochs and a batch size of 32, with a validation split of 0.2. The MAE of the untuned champion challenger was 1015089.87 (Table 14, Figure 14), which is a lot higher than the LGBM. While XGBoost had closer values as discussed

previously, it cannot be taken into consideration due to its closeness in similarity to the LGBM. Thus, the NN was chosen as the final champion challenger, as it performed better than the baseline and worse than the final champion model.

Next, the Keras tuner was employed to find the best hyperparameters (Table 15), and this was tested with extreme values (Table 16, Figure 15). The MAE turned out to be 1144231.24, which is higher and indicates that the NN does not handle data distortions well, and was faced with a lot of data loss.

Section 4: Model Evaluation Methodology

A K-Fold cross-validation (k=5) was employed to reduce risks and screen for overfitting. From the outset, it was likely that a gradient boost model would work better than a neural network or random forest model due to it having the capacity to deal with tabular data, which was later confirmed by comparing MAE and MAPE values. Upon guidance, MAE emerged as the best indicator due to the data dealing with prices, as this is how customers and real estate agencies will quantify the accuracy of the predictions.

First, the data was inspected. It was clear that there were many missing values, and the asynchronicity of the columns led us to suspect that though many columns likely dealt with the same factor (e.g., geography), there would be one or two outstanding columns which held greater predictive power and were less sparse. In this way, computing power was saved and the code ran efficiently.

The initial attempts included almost all the features, but as it was slowly scaled back and condensed using Grid Search, improvements were noticeable in performance and run time. Clearly, there are some factors that actively hindered predictive power, and a feature importance chart including the given column features and the new ones obtained through feature engineering was created to analyze the differences. After this, features were reduced down to 24 and then eliminated through trial-and-error regarding the MAE.

21 final parameters were selected for hypertuning and their best values were determined through a function instead of relying on trial-and-error, since even small changes in these tuning parameters were making large waves in the errors.

Section 5: Tuning

5.1. Hyperparameter Tuning

While GridSearch yielded the first set of values, manual experimentation with values helped in obtaining potential performance increases. Table 12 outlines all the hyperparameters tuned, and the reason for their selection.

There was a ~20,000 unit decrease in the MAE after manual tuning.

5.2. Avoiding Overfitting

To ensure that additional overfitting was not introduced during the hyperparameter tuning, several strategies were implemented.

1. **Cross-Validation Validation**: Cross-validation was used to evaluate model performance. This ensures that the hyperparameters generalize Ill across different data splits.
2. **Regularization Parameters**: Regularization parameters reg_alpha (L1 regularization) and reg_lambda (L2 regularization) were adjusted, which balances model complexity and performance.
3. **Complexity Controls**: Parameters such as max_depth, num_leaves, min_child_weight, and min_split_gain were tuned to avoid an unnecessarily complex model. These parameters limit the depth of trees and the number of leaves, controlling the model's capacity to fit the training data too closely.
4. **Feature Handling**: Parameters like colsample_bytree, subsample, and subsample_freq are tuned to manage feature selection and data sampling. This reduces the risk of overfitting by preventing the model from relying too heavily on any single feature or data point.
5. **Handling Categorical Features**: Parameters were adjusted related to categorical features, such as cat_smooth and cat_l2, which help manage how categorical data is processed, and prevent excessive smoothing or overfitting to rare categories.
6. **Avoiding Data Leakage**: Data leakage was carefully avoided by splitting the train and test data at 0.2, and running the model purely on the test data.
7. **Hyperparameter Search Space**: By constraining the search space to reasonable ranges, it was possible to focus on optimizing parameters that genuinely improved model performance instead of processes that crash runtime and waste computational resources.

Section 6: Robustness and Upstreaming

6.1. Model Response to Extreme Values

LGBM is the best performer in terms of balancing between cross-validation consistency and generalization to new data. Outlier detection was performed and it was tested by introducing a few extreme values to measure model performance. The robustness of LGBM ws evident as the Mean Absolute Error on original data: 0.2449147986818385, while the Mean Absolute Error on data with outliers: 0.23436955260842146 (Table 13, Figure 3). The similarities in MAE scores indicate that the trained LGBM model handles extreme values with relative ease, especially when comparing the scores obtained with the NN model. Extreme value testing for NNs produced an MAE of 1144231.25. This reaffirmed the choice of selecting LGBM as the champion model (Table 14, Figure 15).

6.2. Upstreaming and Complexity

In this case, the model will no longer consider these columns and some predictive power will be lost, and covariate shifts can also introduce this inefficiency. To avoid this error, imputation and standardization were undertaken.

While complex models tend to have higher accuracy and are adaptable to feature interactions, there is a risk of overfitting and a lack of transparency, as well as the very real considerations of computational power they consume to be taken into account. However, complex models can have strengths such as handling large data and hyperparameter tuning. It would not have been possible to get the same MAE with the MLR model or the XGBoost model as (a) one would not have had much flexibility in tuning the MLR, and (b) the XGBoost would have been comparatively ineffective in processing large batches of data, and this would pose problems if the model were to be used on larger datasets.

6.3. Feature Importance

Feature importance is summarized in a table and visually for easier interpretation, and the values are listed in Table 7 and Figure 11. The values under the "Importance" column represent the relative contribution of each feature to the model's predictions. For example, features with higher importance scores, such as sell_date and living_area, are crucial in influencing the model's predictions, indicating they hold substantial predictive power.

These importance scores help identify which variables are most effective at predicting the target outcome and which may be redundant or unnecessary. So, based on this table, *luxury_score* was removed from the features list, and the final predictions were made with the final tuned model.

Some features performed better than others. It is surprising to observe the weight *latitude* and *longitude* held in comparison to *postcode* and *street_name* and by how little weight *asking_price* held; it was initially thought it would be a rather heavy influence and even it was even considered for weighting. It was also interesting to note how seemingly little the importance of amenities was (in terms of the *has_balcony, has_patio, has_solar_panels*, and *has_fireplace* variables, which were combined into the *luxury_score*), since these are things that, in real life, are perceived to have a great influence on valuing decisions.

The model performs weakly in predicting higher selling prices and doesn’t do well with houses classified as row houses or townhouses due to the low training data count for these house types. However, if it turns out the model is to be implemented on a larger range of object_types, it may not do as well, as several key factors are specifically related to apartments, such as floor categorisation.

6.4. Considerations and Recommendations

If the nature of the inputs change, or if the trend (that housing prices get more expensive over the years) changes, it is likely that the model will drift. In this case, it is best to recalibrate the model with new features, and perhaps remove *weighted_property_age* as that no longer reflects the dominant trend.

It is possible to tackle model drift, for example, by establishing a monitoring framework with key performance indicators (KPIs) thresholds that are set, which, when crossed will sound alerts. Recalibration upon threshold violation can be done by periodic retraining of the LGBM, implementing adaptive learning, and cross-validating on the new data, while maintaining model

versioning for rollback if needed. Documenting these changes and updating stakeholders as these changes take place, then, ensures the alignment with the company's business objectives.

6.4.1. Are there any other concerns with the data or model that you believe should be flagged/brought to the attention of validation / senior management?

Senior management would be advised to look into different ways of handling missing values and sparse data, since with clearly objective factors (such as *area*, or *floor*, or *property_age*), it is hard to rely on mean imputation. It would also be interesting to know if there was a specific method for creating the *ids* that perhaps gave more information than the currently available data points.

Additionally locality, street name, and street address, can have the potential to be transformed into one single feature that's useful instead of three, overlapping features that could distort predictions.

Section 7: Comparison to The Previous Submission

This is a small section detailing the differences in my submission from the previous group submission done in May.

There are eleven main differences:

1. Scaling and encoding was standardized for the dataset, and missing values were processed individually as opposed to a general scaler that was used for all features
2. DASK was used in conjunction with the MLR code, and this optimized MLR model saved computational resources and time
3. Cutoff dates were tested to determine whether excluding previous years (2013 - 2017) would have an impact on model performance and predictive power
4. Outlier and extreme values tests were absent from the previous submission. I implemented outlier testing for both the LGBM model and the NN model.
5. Features were condensed significantly. The previous submission contained around 35 features, and this was condensed to 24 final features in the current submission after analyzing feature importance and iteratively adding/removing features
6. The MAE was taken as a determinant of model accuracy as opposed to the previously used R2.
7. The previous submission only looked at 6 hyperparameters for tuning. I incorporated additional hyperparameters, bringing it to a total of 21 selected hyperparameters for tuning the LGBM.
8. The standard scaler was used again to scale the unseen data, as this was not properly implemented in the previous submission
9. Visualizations for models were also improved, with histograms for errors and residuals also added to the code
10. The champion challenger NN model was also modified to include a Kera tuner and was tested for extreme values in the new submission, which improved its accuracy

11. Finally, a comparison with XGBoost was implemented into this submission, and these insights helped in my understanding of the different models' strengths and weaknesses, and also demonstrated/justified the selection of LightGBM over XGBoost due to the latter's ineffectiveness in handling large amounts of data.

Appendix 1: Tables

Table 1: K-Fold Cross Validation for LGBM

```
Cross-Validation Evaluation:
Mean Absolute Error (MAE): 320675.36843929644
Root Mean Squared Error (RMSE): 654094.7704385312
Mean Absolute Percentage Error (MAPE): 6.879905920662186%

Standard Deviation of Metrics:
MAE Std Dev: 2298.6473747182345
RMSE Std Dev: 24977.924295357152
MAPE Std Dev: 0.04747360315474067%
```

Table 2: Multiple Linear Regression Baseline Performance

```
Mean Squared Error (MSE): 2757734170624.0
Root Mean Squared Error (RMSE): 1660642.75
Mean Absolute Error (MAE): 1065121.5
R-squared (R2): 0.653369643535439
```

Table 3: XGBoost Performance

```
Training Set Evaluation:
Mean Absolute Error (MAE): 235975.08536407523
Root Mean Squared Error (RMSE): 346296.2740855433
Mean Absolute Percentage Error (MAPE): 5.764702434897948%

Test Set Evaluation:
Mean Absolute Error (MAE): 344226.98138240766
Root Mean Squared Error (RMSE): 664167.4567603767
Mean Absolute Percentage Error (MAPE): 7.477072231592021%
['xgb_regressor_model.pkl']
```

Table 4: Actual and Predicted Sell Prices distribution

price_category	Actual Sell Prices	Predicted Sell Prices
100k-300k	1	0
300k-500k	1	0
500k-700k	2	3
700k-1M	73	39
1M-2M	12319	3530
2M-3M	51198	697
3M-4M	41630	32
4M-5M	23309	3
5M-6M	15296	0
6M-7M	10300	0
7M-8M	6595	0

Table 5: Distribution of prediction from the previous submission in May

price_category	Actual Sell Prices	Previously Predicted Sell Prices
100k–300k	1	0
300k–500k	1	0
500k–700k	2	0
700k–1M	73	0
1M–2M	12319	0
2M–3M	51198	322
3M–4M	41630	1168
4M–5M	23309	1024
5M–6M	15296	965
6M–7M	10300	500
7M–8M	6595	161
8M–9M	4273	79
9M–10M	2756	32

Table 5: Intuitive classification of features

no correlation	basic	locality	area + cost	energy	amenities	time	risk	construction
object_type	housing_coop_id	legal_district	living_area	energy_class	has_balcony	sell_date	association_tax_liability	construction_year
sell_price	name	postcode	additional_area	has_solar_panels	has_fireplace	fiscal_year	long_term_debt_other	is_new_construction
id	agency_id	locality	plot_area		has_patio		long_term_real_estate_debt	
	housing_association_org_number	street_name	rooms		energy_class		total_loan	
	total_rental_area	street_address	width				plot_is_leased	
	number_of_rental_units	brokers_description	height				savings	
	number_of_units	latitude	floor					
	total_commercial_area	longitude	rent					
	total_living_area		operating_cost					
	total_plot_area		asking_price					

Table 6: Spearman Correlation List and Values

Significant Spearman Correlation Coefficients and P-values:
rooms: Correlation = 0.522, P-value = 0.0000
postcode: Correlation = -0.496, P-value = 0.0000
latitude: Correlation = 0.194, P-value = 0.0000
longitude: Correlation = 0.275, P-value = 0.0000
has_solar_panels: Correlation = 0.006, P-value = 0.0203
living_area: Correlation = 0.555, P-value = 0.0000
additional_area: Correlation = -0.039, P-value = 0.0000
rent: Correlation = 0.249, P-value = 0.0000
floor: Correlation = 0.177, P-value = 0.0000
operating_cost: Correlation = 0.259, P-value = 0.0000
asking_price: Correlation = 0.226, P-value = 0.0000
has_balcony: Correlation = 0.006, P-value = 0.0202
has_fireplace: Correlation = 0.014, P-value = 0.0000
construction_year: Correlation = -0.143, P-value = 0.0000
long_term_debt_other: Correlation = 0.104, P-value = 0.0000
long_term_real_estate_debt: Correlation = -0.132, P-value = 0.0000
number_of_rental_units: Correlation = -0.108, P-value = 0.0000
number_of_units: Correlation = -0.235, P-value = 0.0000
total_commercial_area: Correlation = 0.011, P-value = 0.0000
total_living_area: Correlation = -0.111, P-value = 0.0000
total_loan: Correlation = -0.132, P-value = 0.0000
total_plot_area: Correlation = -0.050, P-value = 0.0000
total_rental_area: Correlation = -0.095, P-value = 0.0000
savings: Correlation = -0.009, P-value = 0.0003
living_area_fixed: Correlation = 0.556, P-value = 0.0000
additional_area_fixed: Correlation = 0.123, P-value = 0.0000
floor_fixed: Correlation = 0.189, P-value = 0.0000
rent_fixed: Correlation = 0.251, P-value = 0.0000
operating_cost_fixed: Correlation = 0.453, P-value = 0.0000
asking_price_fixed: Correlation = 0.566, P-value = 0.0000
long_term_debt_other_fixed: Correlation = 0.091, P-value = 0.0000
long_term_real_estate_debt_fixed: Correlation = -0.141, P-value = 0.0000
number_of_rental_units_fixed: Correlation = -0.116, P-value = 0.0000
number_of_units_fixed: Correlation = -0.242, P-value = 0.0000
plot_is_leased_fixed: Correlation = -0.344, P-value = 0.0000
savings_fixed: Correlation = -0.009, P-value = 0.0003
total_living_area_fixed: Correlation = -0.111, P-value = 0.0000
total_loan_fixed: Correlation = -0.140, P-value = 0.0000
total_plot_area_fixed: Correlation = -0.062, P-value = 0.0000
total_rental_area_fixed: Correlation = -0.104, P-value = 0.0000
construction_year_fixed: Correlation = -0.237, P-value = 0.0000
total_property_area: Correlation = 0.346, P-value = 0.0000
avg_room_area: Correlation = 0.070, P-value = 0.0000
property_volume: Correlation = 0.010, P-value = 0.0000
rent_per_sqm: Correlation = -0.496, P-value = 0.0000
asking_price_ratio: Correlation = 0.566, P-value = 0.0000
price_per_sqm: Correlation = 0.046, P-value = 0.0000
operating_cost_to_living_area_ratio: Correlation = -0.176, P-value = 0.0000
cost_efficiency_ratio: Correlation = 0.119, P-value = 0.0000

fiscal_year: Correlation = -0.142, P-value = 0.0000 plot_is_leased: Correlation = -0.338, P-value = 0.0000 sell_date_fixed: Correlation = 0.231, P-value = 0.0000 sell_year: Correlation = 0.231, P-value = 0.0000 rooms_fixed: Correlation = 0.522, P-value = 0.0000 energy_class_fixed: Correlation = -0.080, P-value = 0.0000 luxury_score: Correlation = 0.007, P-value = 0.0061	total_units: Correlation = -0.248, P-value = 0.0000 total_area: Correlation = -0.144, P-value = 0.0000 residential_to_commercial_ratio: Correlation = -0.091, P-value = 0.0000 property_age: Correlation = 0.237, P-value = 0.0000 weighted_property_age: Correlation = -0.237, P-value = 0.0000 age_new_construction_interaction: Correlation = -0.235, P-value = 0.0000

Table 7: Feature Importance list for the untuned model

Top 50 Features:

	Feature	Importance
48	post	130
40	sell_date_fixed	97
23	housing_association_org_number	91
45	living_area_fixed	76
9	living_area	69
4	street_name	68
3	locality	46
0	rooms	45
67	avg_room_area	32
13	floor	30
6	street_address	28
1	legal_district	22
7	brokers_description	20
49	rent_fixed	16
12	rent	14
69	rent_per_sqm	13
46	additional_area_fixed	11
51	asking_price_fixed	10
41	sell_year	10
71	price_per_sqm	9
66	total_property_area	9
72	operating_cost_to_living_area_ratio	5
80	age_new_construction_interaction	5
16	asking_price	4
15	agency_id	4
25	construction_year	4
24	name	3
61	total_loan_fixed	3
29	number_of_rental_units	3
39	plot_is_leased	3
64	construction_year_fixed	3
56	number_of_units_fixed	2
2	postcode	2
32	total_living_area	2
50	operating_cost_fixed	2
83	sell_month	1
14	operating_cost	1
75	total_area	1
74	total_units	1
28	long_term_real_estate_debt	1
57	plot_is_leased_fixed	1
54	long_term_real_estate_debt_fixed	1
35	total_rental_area	1
47	floor_fixed	1
5	object_type	0
8	energy_class	0
10	additional_area	0
11	plot_area	0
76	residential_to_commercial_ratio	0
77	property_age	0

Table 8: Untuned LGBM Performance

```
Training Set Evaluation:
Mean Absolute Error (MAE): 386969.15311536804
Root Mean Squared Error (RMSE): 682672.4610219309
Mean Absolute Percentage Error (MAPE): 8.755229067717037%

Test Set Evaluation:
Mean Absolute Error (MAE): 406513.06051891606
Root Mean Squared Error (RMSE): 725634.9880639797
Mean Absolute Percentage Error (MAPE): 9.100027079924471%
(386969.15311536804,
 682672.4610219309,
 8.755229067717037,
 406513.06051891606,
 725634.9880639797,
 9.100027079924471)
```

Table 9: Manually Condensed Features on an untuned LGBM

```
Training Set Evaluation:
Mean Absolute Error (MAE): 386969.15311536804
Root Mean Squared Error (RMSE): 682672.4610219309
Mean Absolute Percentage Error (MAPE): 8.755229067717037%

Test Set Evaluation:
Mean Absolute Error (MAE): 406513.06051891606
Root Mean Squared Error (RMSE): 725634.9880639797
Mean Absolute Percentage Error (MAPE): 9.100027079924471%
```

Table 10: Performance for LGBM with condensed Features and no hyperparameter tuning

```
Training Set Evaluation:
Mean Absolute Error (MAE): 390435.69562100305
Root Mean Squared Error (RMSE): 688914.8150118883
Mean Absolute Percentage Error (MAPE): 8.840994026836855%

Test Set Evaluation:
Mean Absolute Error (MAE): 408872.7627376788
Root Mean Squared Error (RMSE): 729261.1403183339
Mean Absolute Percentage Error (MAPE): 9.152621863501523%
```

Table 11: Final Champion LGBM with tuning and feature selection and manual tuning

```
Training Set Evaluation:
Mean Absolute Error (MAE): 112122.27870058158
Root Mean Squared Error (RMSE): 151519.6134709429
Mean Absolute Percentage Error (MAPE): 3.053971469514815%

Test Set Evaluation:
Mean Absolute Error (MAE): 318746.1145996769
Root Mean Squared Error (RMSE): 629425.0849366143
Mean Absolute Percentage Error (MAPE): 6.879282604598037%
['lgbm_regressor_model.pkl']
```

Table 12: List of hyperparameters selected and tunes

Hyperparameter purpose	**Hyperparameters**
Addressing model complexity and overfitting	**max_depth**
	min_split_gain
	num_leaves
	min_child_weight
	min_child_samples
	max_bin
For Regularization	**reg_alpha**
	reg_lambda
Regularization:**,** Learning and Training Efficiency	**learning_rate**
	n_estimators
	subsample,
	subsample_freq
Feature Handling	**colsample_bytree**
	cat_smooth
	cat_l2
	max_cat_threshold

	cat_quantile_max
	cat_quantile_min
	cat_max_card
	cat_min_card
Data handling and model efficiency	**min_data_in_leaf**
	min_child_samples

Table 13: Summary statistics for data with outliers for LGBM

```
Summary statistics for data with outliers:
         feature1    feature2
count  150.000000  150.000000
mean     5.096075   -0.287813
std     31.570637    0.792818
min    -71.048950   -1.701699
25%     -1.420979   -0.443710
50%      0.366426   -0.047376
75%     18.321300    0.096604
max     56.470700    0.657115
```

Table 14: Neural Networks Untuned Model performance

```
Training Set Evaluation:
Mean Absolute Error (MAE): 952788.8276246408
Root Mean Squared Error (RMSE): 2131944.479543965
Mean Absolute Percentage Error (MAPE): 20.184254131473097%

Test Set Evaluation:
Mean Absolute Error (MAE): 1015089.8687440031
Root Mean Squared Error (RMSE): 2129147.214285303
Mean Absolute Percentage Error (MAPE): 21.54318553140322%
```

Table 15: Keras Tuner results

```
Trial 3 Complete [00h 42m 07s]
val_loss: 2183235982677.3333

Best val_loss So Far: 2183235982677.3333
Total elapsed time: 02h 09m 37s

Search: Running Trial #4

Value             |Best Value So Far |Hyperparameter
128               |64                |units1
0.3               |0.5               |dropout1
2                 |2                 |num_layers
128               |128               |units_2
0.2               |0.3               |dropout_2
adam              |adam              |optimizer
96                |32                |units_3
0.5               |0.1               |dropout_3
```

Table 16: NN Model Performance for extreme values testing

```
Training Set Evaluation:
Mean Absolute Error (MAE): 1090962.033031801
Root Mean Squared Error (RMSE): 2381402.419428806
Mean Absolute Percentage Error (MAPE): 23.569964443015344%

Test Set Evaluation:
Mean Absolute Error (MAE): 1144231.246057835
Root Mean Squared Error (RMSE): 2402519.715370079
Mean Absolute Percentage Error (MAPE): 24.608426843429818%
```

Table 17: XGBoost Performance Metrics

```
Training Set Evaluation:
Mean Absolute Error (MAE): 272034.7794309211
Root Mean Squared Error (RMSE): 413247.1009649157
Mean Absolute Percentage Error (MAPE): 6.467752772664792%

Test Set Evaluation:
Mean Absolute Error (MAE): 354945.7248013885
Root Mean Squared Error (RMSE): 667168.3061105545
Mean Absolute Percentage Error (MAPE): 7.800980444042027%
['xgb_regressor_model_tuned.pkl']
```

Table 18: Best performance of XGBoost

```
Training Set Evaluation:
Mean Absolute Error (MAE): 235975.08536407523
Root Mean Squared Error (RMSE): 346296.2740855433
Mean Absolute Percentage Error (MAPE): 5.764702434897948%

Test Set Evaluation:
Mean Absolute Error (MAE): 344226.98138240766
Root Mean Squared Error (RMSE): 664167.4567603767
Mean Absolute Percentage Error (MAPE): 7.477072231592021%
['xgb_regressor_model.pkl']
```

Appendix 2: Figures

Figure 1: Multiple Linear Regression, Visualisation of Performance Metrics

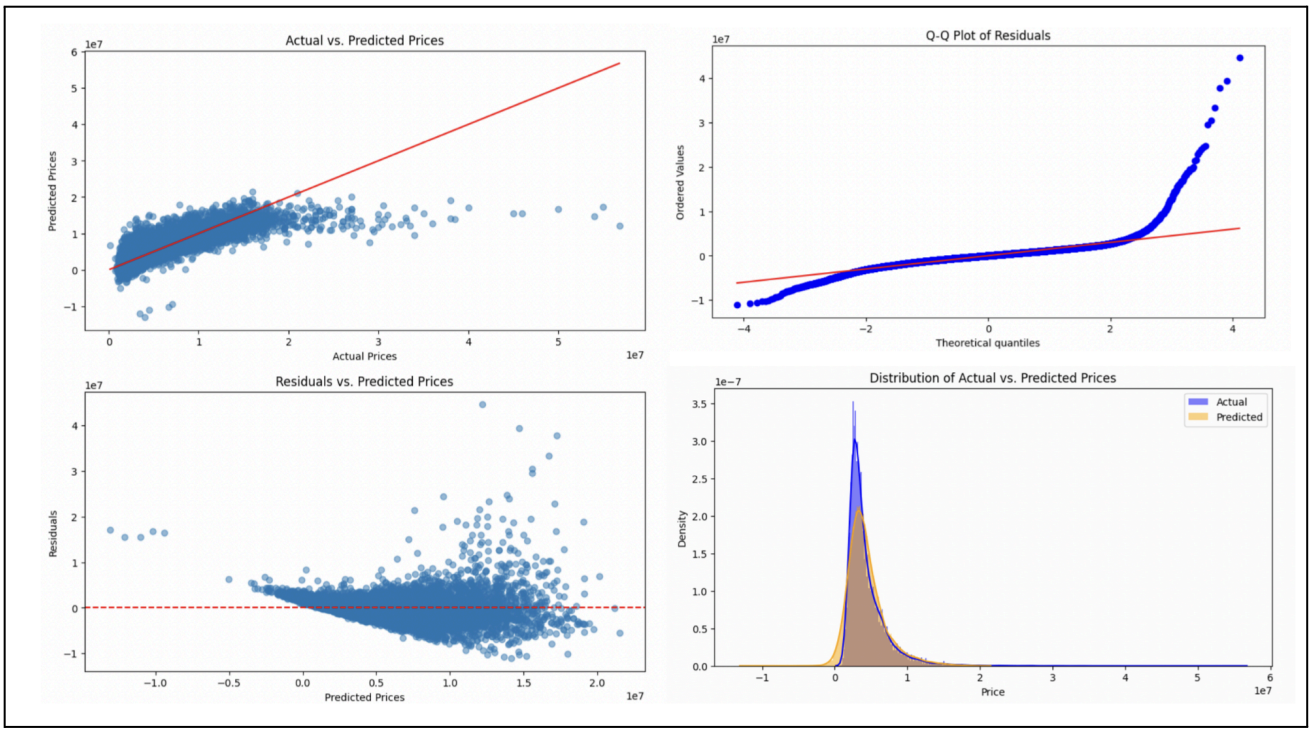


Figure 2: LGBM performance with outliers and extreme values

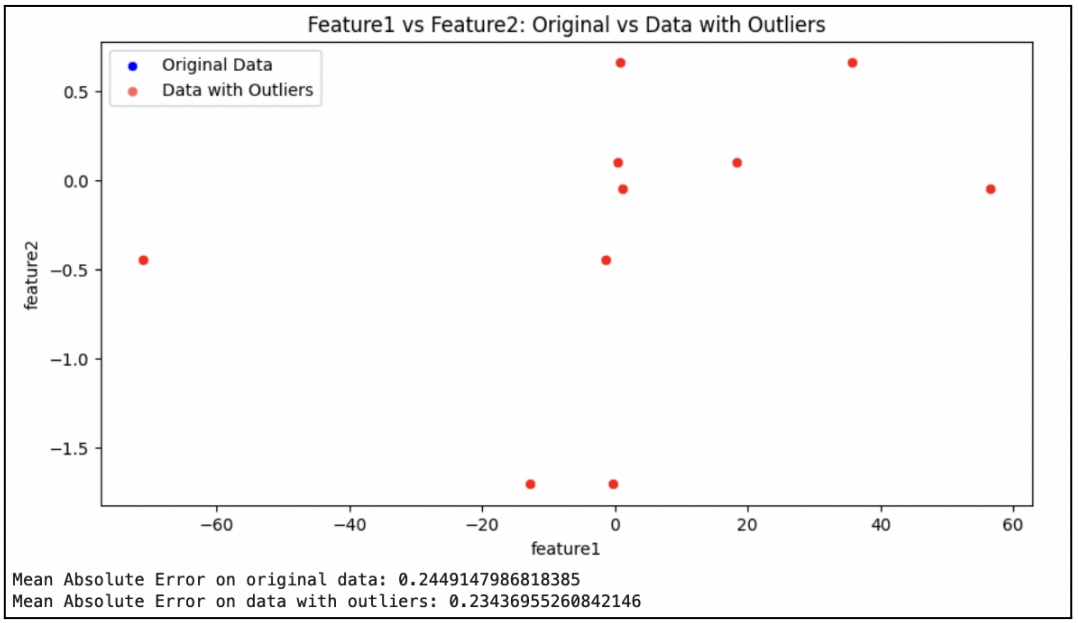


Figure 3: Actual Vs Predicted Plot for LGBM Champion Model

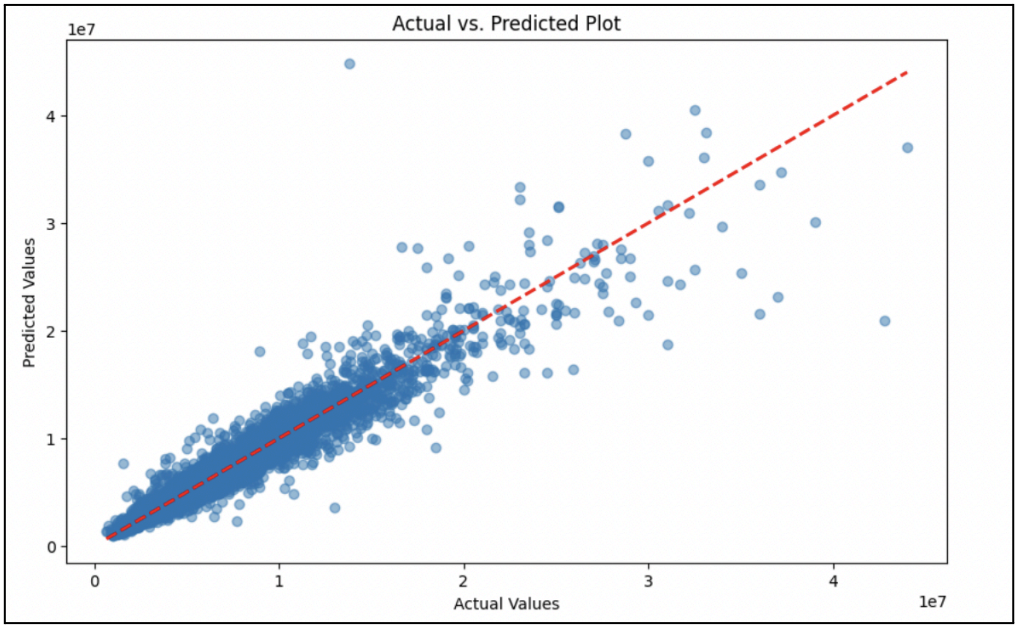


Figure 4: Error Distribution Plot for LGBM Champion Model

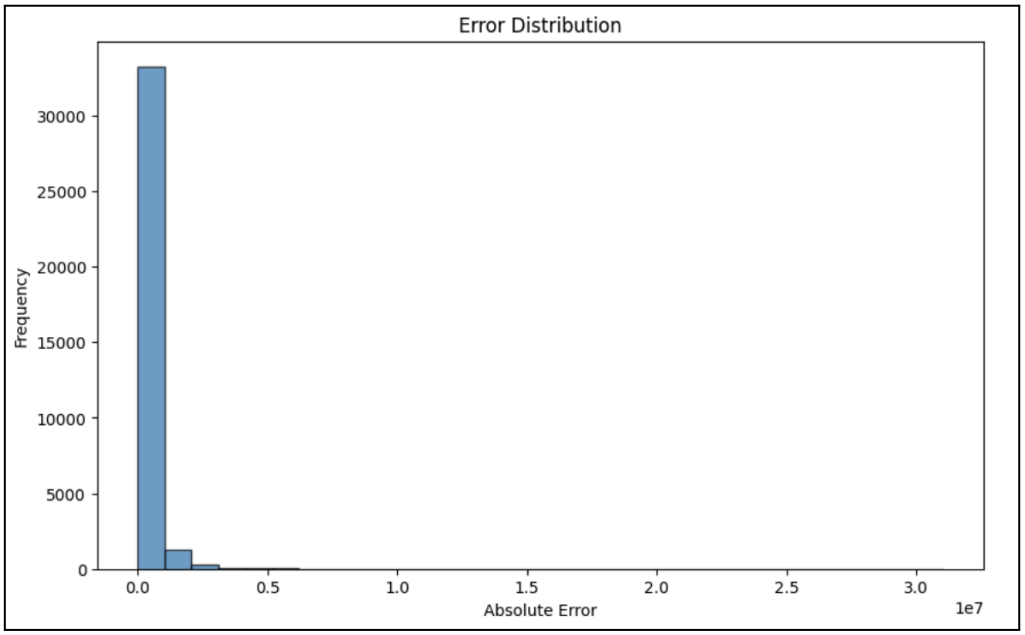


Figure 5: Q-Q Plot for LGBM Champion Model

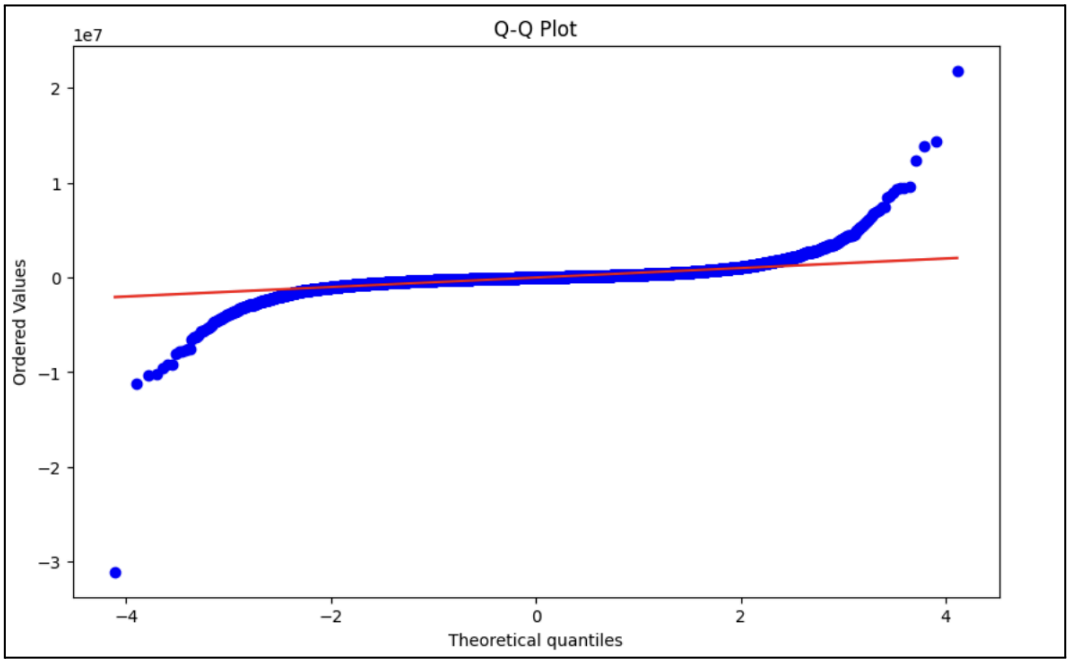


Figure 6: Comparison between actual prices and predicted prices with the LGBM Champion Model

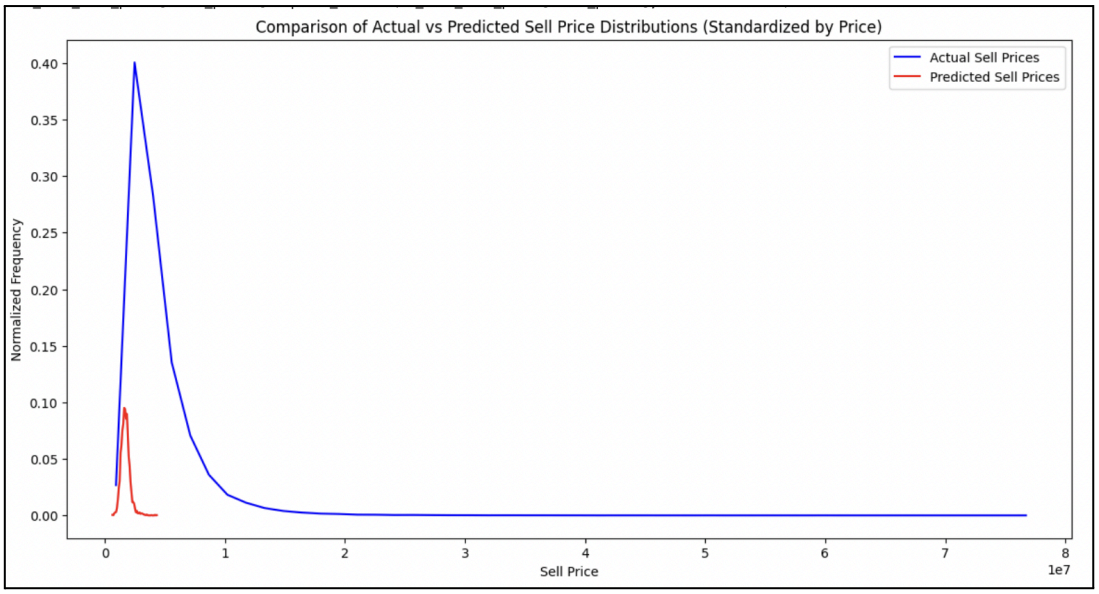


Figure 7: Sell price distribution by category, Actual Vs. Predicted

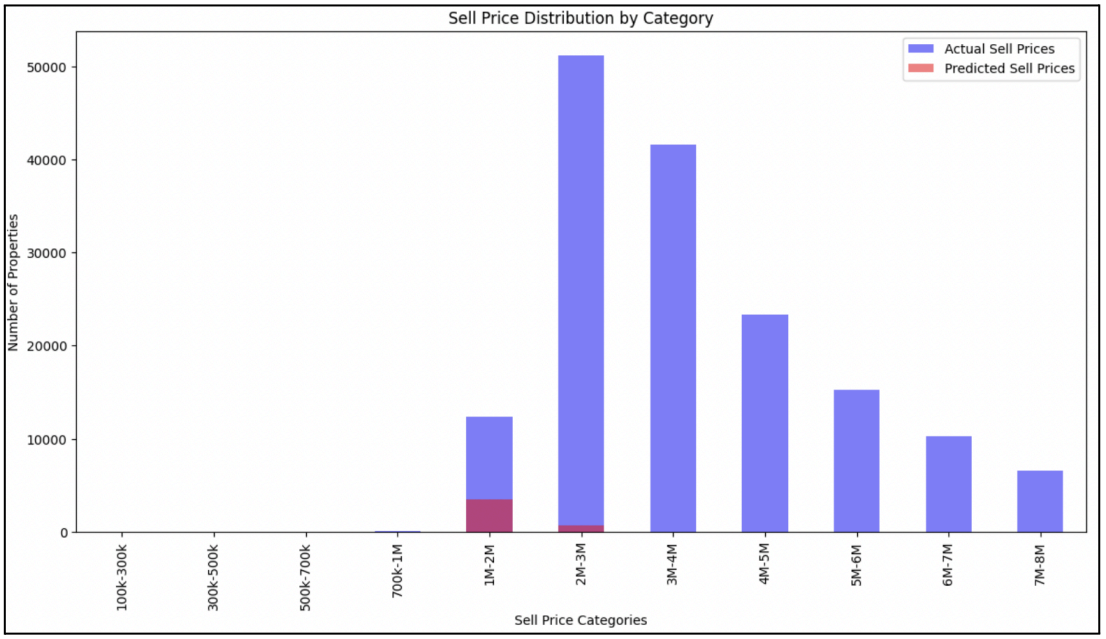


Figure 8: Sell price distribution of previous submissions' predictions

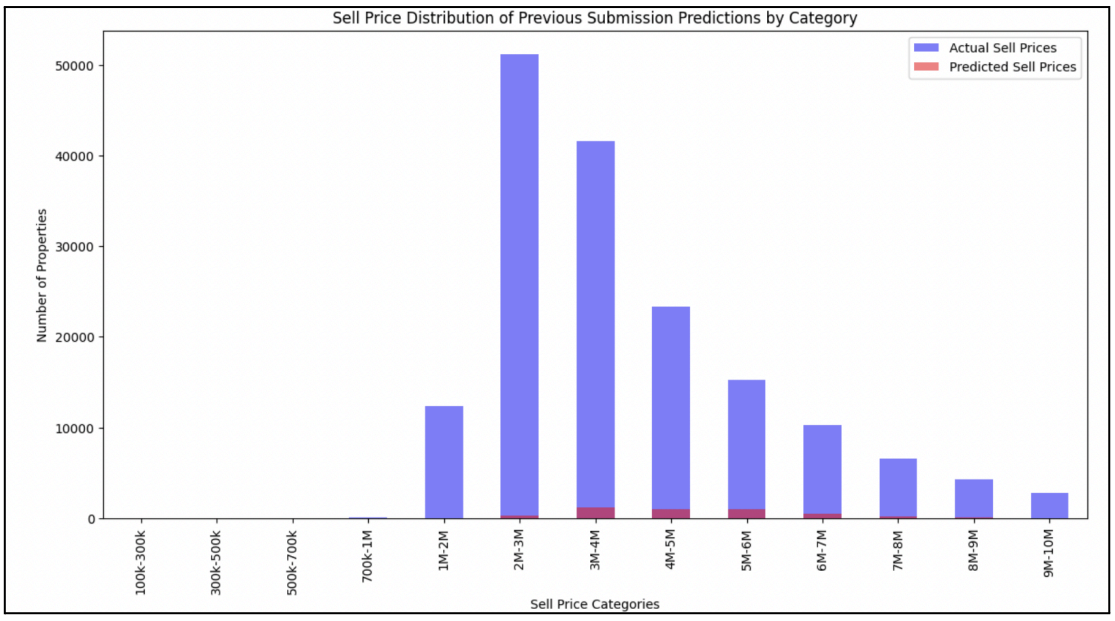


Figure 9: Correlation plots for features

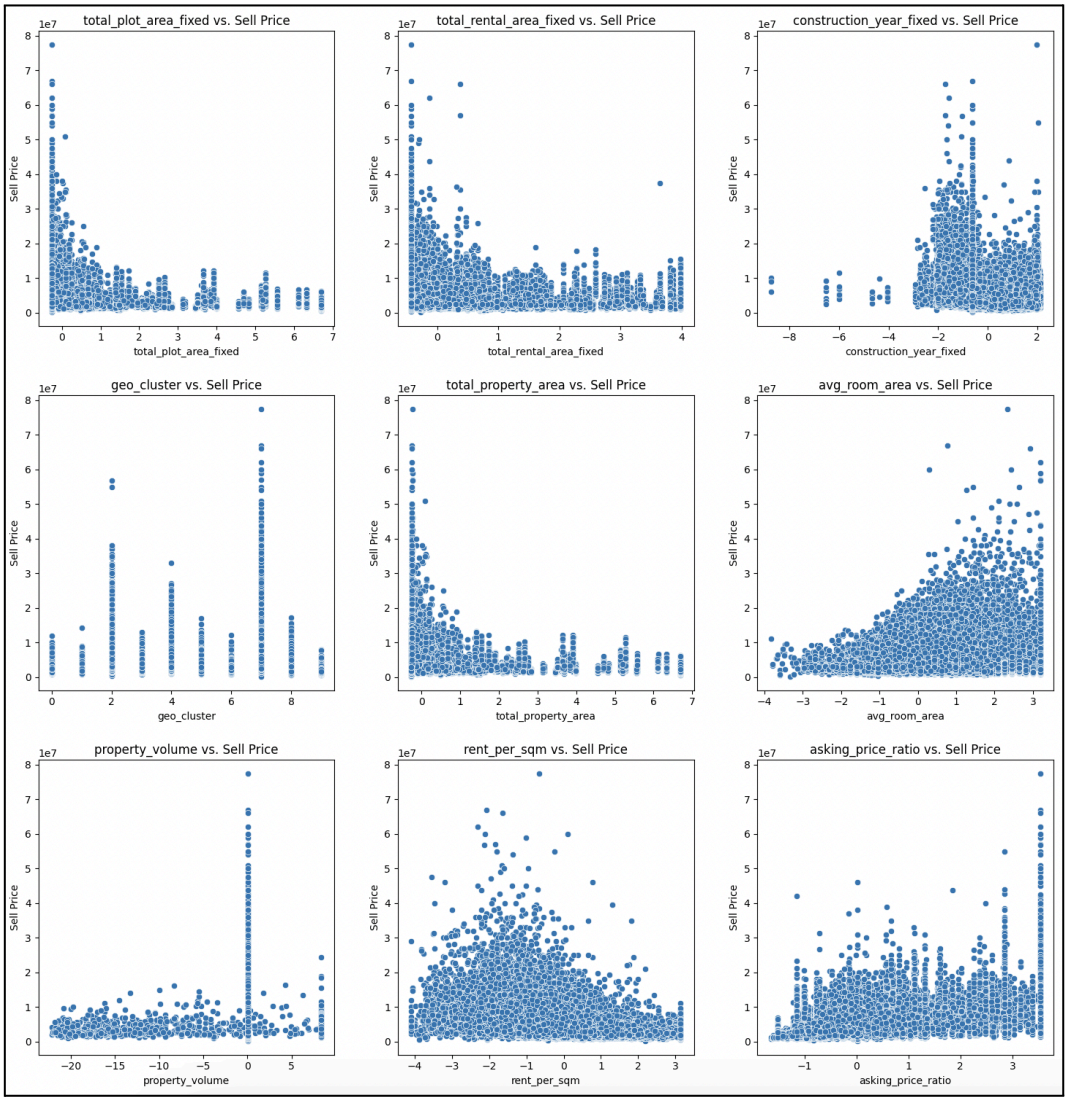


Figure 10: Correlation matrix for features

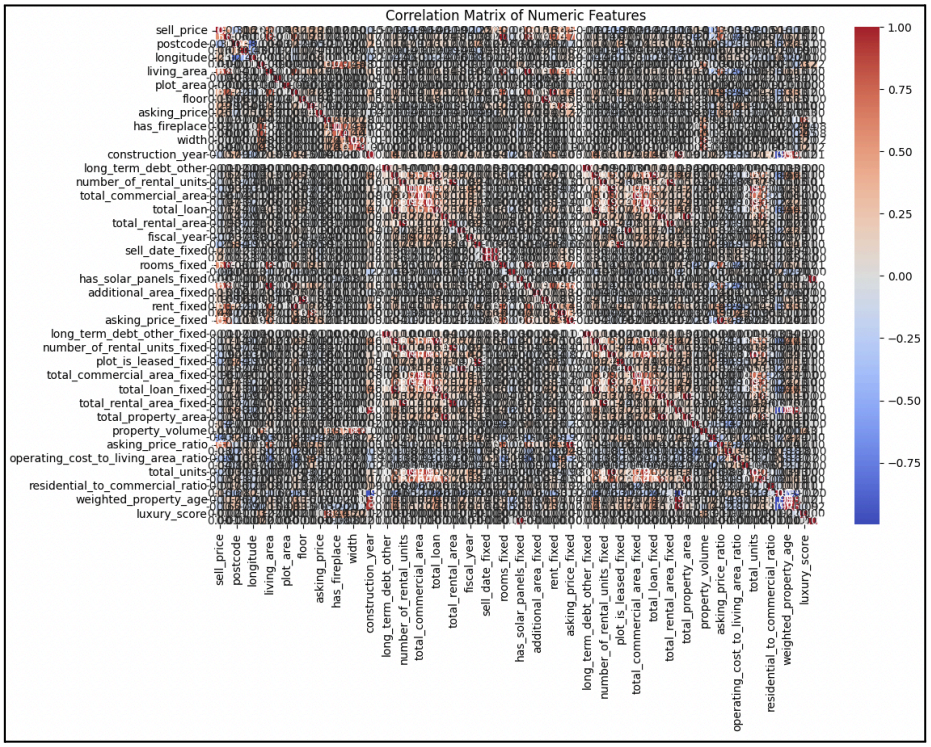
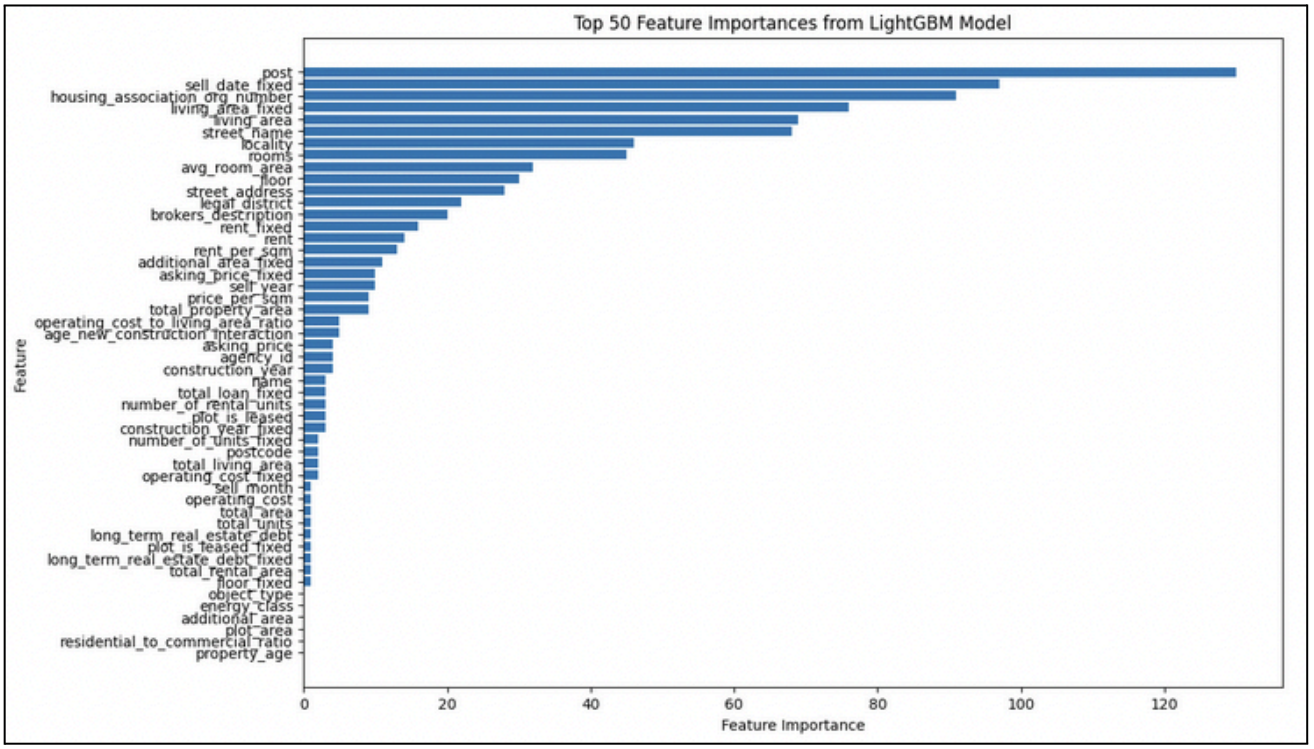


Figure 11:
(a) Feature Importance of the untuned LGBM



(b) Feature Importance List of the Final Champion LGBM

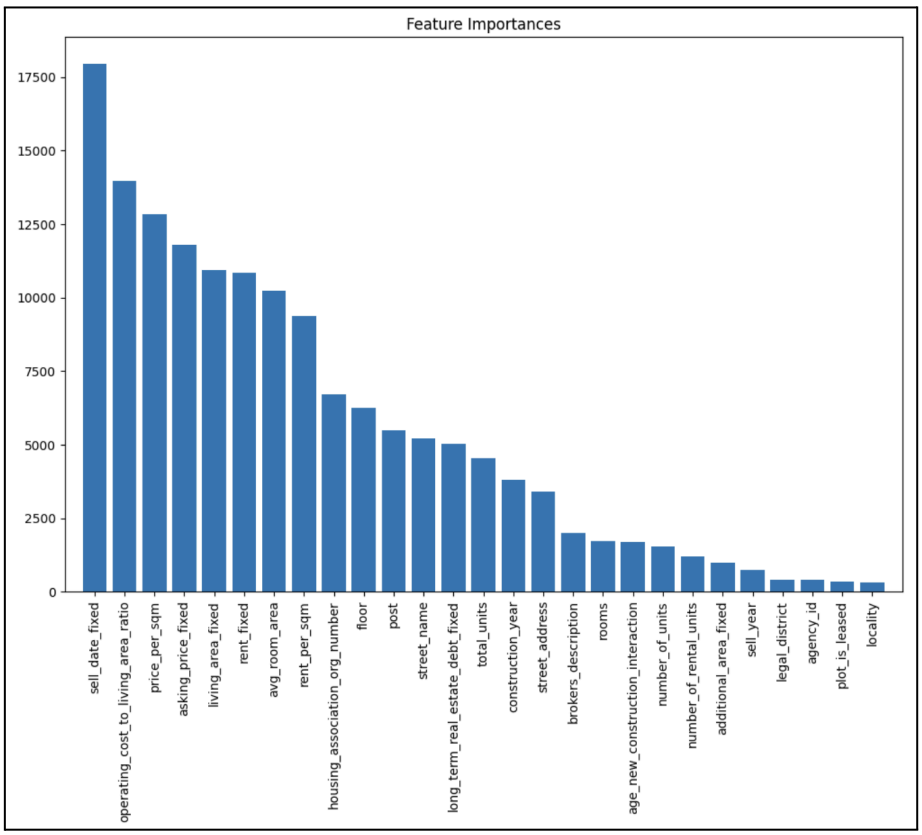


Figure 12: Standard scaler code

```python
# Scaling Dataset

from sklearn.preprocessing import StandardScaler
import joblib
import numpy as np

# Replace infinite values with NaN
df_with_sell_price.replace([np.inf, -np.inf], np.nan, inplace=True)

# Fill missing values with column means for numeric columns
numeric_cols = df_with_sell_price.select_dtypes(include=['int64', 'float64'])
df_with_sell_price[numeric_cols.columns] = numeric_cols.fillna(numeric_cols.mean())

# Cap extremely large values (example: values larger than 3 standard deviations from the mean)
for col in numeric_cols.columns:
    if col != 'sell_price':  # Exclude the 'sell_price' column
        threshold = df_with_sell_price[col].mean() + 3 * df_with_sell_price[col].std()
        df_with_sell_price[col] = np.where(df_with_sell_price[col] > threshold, threshold, df_with_sell_price[col])


# Select columns to scale (excluding the target variable 'sell_price')
columns_to_scale = df_with_sell_price.select_dtypes(include=['int64', 'float64']).columns.drop('sell_price')

# Create a dictionary to store scalers for each column
scalers = {}

# Fit and transform each column with its own scaler
for col in columns_to_scale:
    scaler = StandardScaler()
    df_with_sell_price[col] = scaler.fit_transform(df_with_sell_price[[col]])
    scalers[col] = scaler

# Create missing value indicator columns if any
for col in df_with_sell_price.columns:
    if df_with_sell_price[col].isnull().any():
        df_with_sell_price[col + '_missing'] = df_with_sell_price[col].isnull()

# Final fill of missing values (if any new ones are created)
numeric_cols = df_with_sell_price.select_dtypes(include=[np.number])
df_with_sell_price[numeric_cols.columns] = numeric_cols.fillna(numeric_cols.mean())

# Print the head of the DataFrame to verify
print(df_with_sell_price.head())

import joblib

# Save the scalers dictionary to a file
joblib.dump(scalers, 'scalers.pkl')
```

Figure 13: One to many mappings for Annual Reports

```python
aggregated_annual_reports = annual_report_df.groupby('org_number').agg({
    'association_tax_liability': 'mean',
    'long_term_debt_other': 'mean',
    'long_term_real_estate_debt': 'mean',
    'number_of_rental_units': 'sum',
    'number_of_units': 'sum',
    'total_commercial_area': 'sum',
    'total_living_area': 'mean',
    'total_loan': 'mean',
    'total_plot_area': 'sum',
    'total_rental_area': 'sum',
    'savings': 'mean',
    'fiscal_year' : 'max',
    'housing_coop_id': 'first',
    'plot_is_leased': 'max'
}).reset_index()

final_df = pd.merge(apartment_full_df, aggregated_annual_reports, left_on='housing_association_org_number', right_on='org_number', how='left')

apartment_full_df = pd.merge(apartment_df, housing_association_df, left_on='housing_association_org_number', right_on='org_number', how='left')
annual_report_df['association_tax_liability'] = pd.to_numeric(annual_report_df['association_tax_liability'], errors='coerce')

columns_to_convert = ['long_term_debt_other', 'long_term_real_estate_debt', 'total_living_area', 'total_loan', 'savings']
for col in columns_to_convert:
    annual_report_df[col] = pd.to_numeric(annual_report_df[col], errors='coerce')
```

Figure 14: Neural Network - untuned model performance

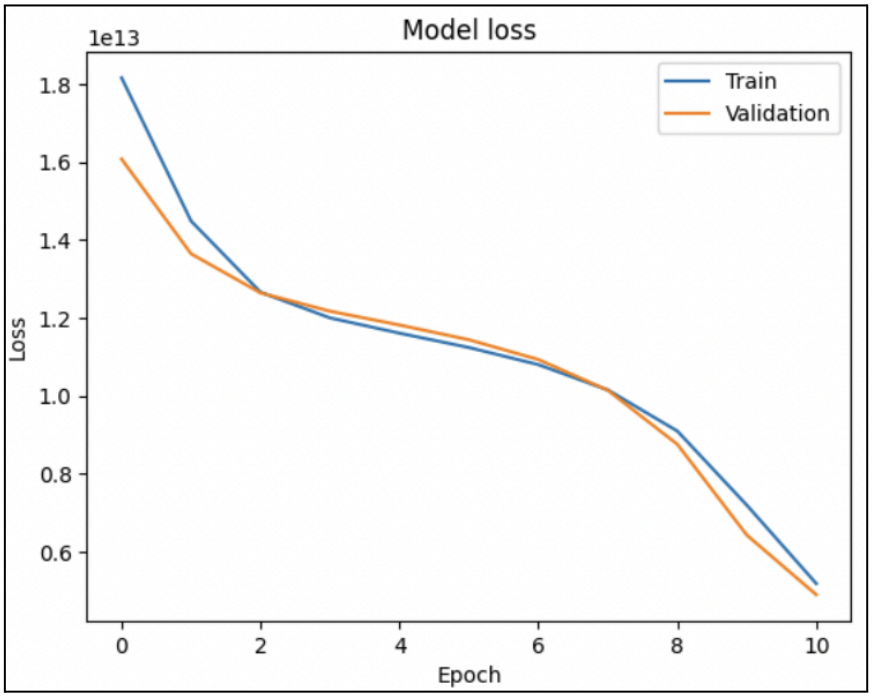


Figure 15: Neural Network tuned model with extreme values

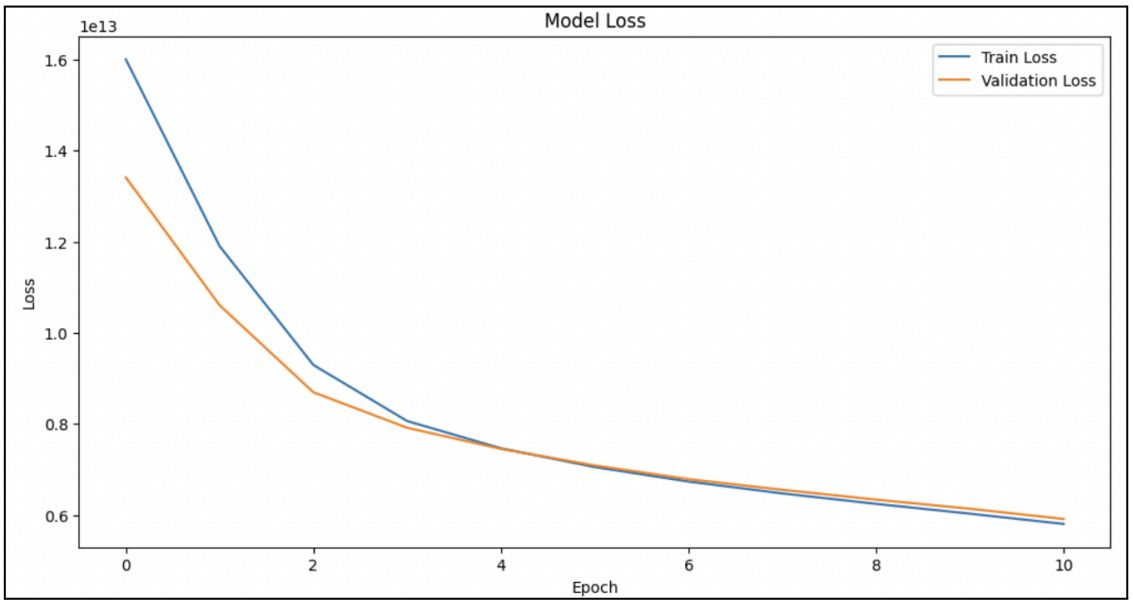


Figure 16: Risk management code

```
df_without_sell_price['total_units'] = df_without_sell_price['number_of_rental_units'] + df_without_sell_price['number_of_units']
df_without_sell_price['total_area'] = df_without_sell_price['total_commercial_area'] + df_without_sell_price['total_living_area'] + df_without_sell_price['total_plot_area'] + df_without_sell_price['total_rental_area']
df_without_sell_price['residential_to_commercial_ratio'] = df_without_sell_price['total_living_area'] / df_without_sell_price['total_commercial_area'].replace(0, 1)



df_without_sell_price['debt_to_savings_ratio'] = df_without_sell_price['total_loan'] / (df_without_sell_price['savings'] + 1)
df_without_sell_price['asset_utilization'] = (df_without_sell_price['total_living_area'] + df_without_sell_price['total_commercial_area']) / (df_without_sell_price['total_plot_area'] + 1)
df_without_sell_price['debt_load_per_unit'] = df_without_sell_price['total_loan'] / (df_without_sell_price['number_of_units'] + 1)
```