Real Estate Value Regression Modeling and Analysis

Garrett Lam

Department of Atmospheric and Oceanic Science, University of California Los Angeles

AOS C111: Introduction to Machine Learning for Physical Sciences

Dr. Alexander Lozinski

1. Introduction

The value of real estate has long been a challenging puzzle for investors and analysts to solve. While it is widely acknowledged that factors such as location, area, age, and proximity to amenities play a significant role in determining property prices, the precise weight and interaction of these factors often remain elusive. Moreover, these relationships vary considerably across different regions and markets, adding another layer of complexity. Traditional methods of manually analyzing these markets often leave much to chance, as they fail to capture the nuanced interplay of variables that ultimately influence property values. This is where machine learning comes in—offering a data-driven solution that can process large amounts of information with exceptional speed and accuracy.

To address this challenge, I applied machine learning techniques to predict real estate prices in Taipei, Taiwan. After testing several models, I found that the Gradient Boosting model provided the most accurate representation of the data. The model's objective was to predict the price per unit area of a district based on features such as house age, proximity to MRT stations, and the number of convenience stores in the area. Through careful data preprocessing, feature engineering, and model training, the Gradient Boosting model demonstrated its ability to handle the complexity of the dataset effectively.

The results were highly promising, showcasing the model's ability to produce accurate predictions when tested against the provided data. By leveraging fundamental machine learning techniques, I was able to gain insights into the factors of real estate pricing in Taipei and deliver a model with high prediction accuracy.

2. Data

The data is obtained from the Sindian District in New Taipei City, Taiwan. The dataset contains information on various properties and their associated features, which can influence their market value. The set contains 414 data points, complete with feature and resulting values.

Features Descriptions:

1. Transaction Date: The date of the real estate transaction (in fractional year format, e.g., 2013.25 for Q1 2013).
2. House Age (years): The age of the house at the time of the transaction.
3. Distance to MRT (meters): The distance to the nearest MRT (Mass Rapid Transit) station.
4. Number of Convenience Stores Nearby: The number of convenience stores within walking distance of the property.
5. Latitude: The latitude coordinate of the property.
6. Longitude: The longitude coordinate of the property.

Target Variable: Price per unit area

- The transaction price per square meter of property (in NT$)

Figure 1. Scatterplot of Price per Unit Area for each row of data

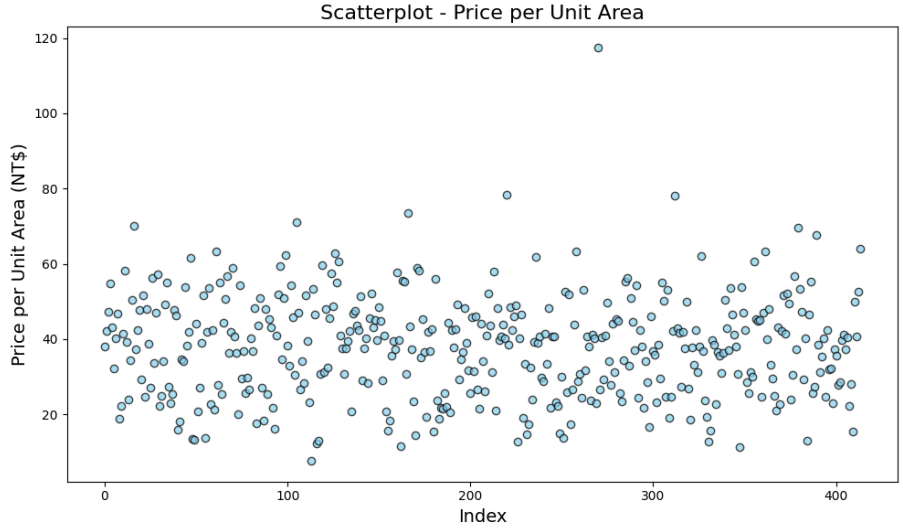
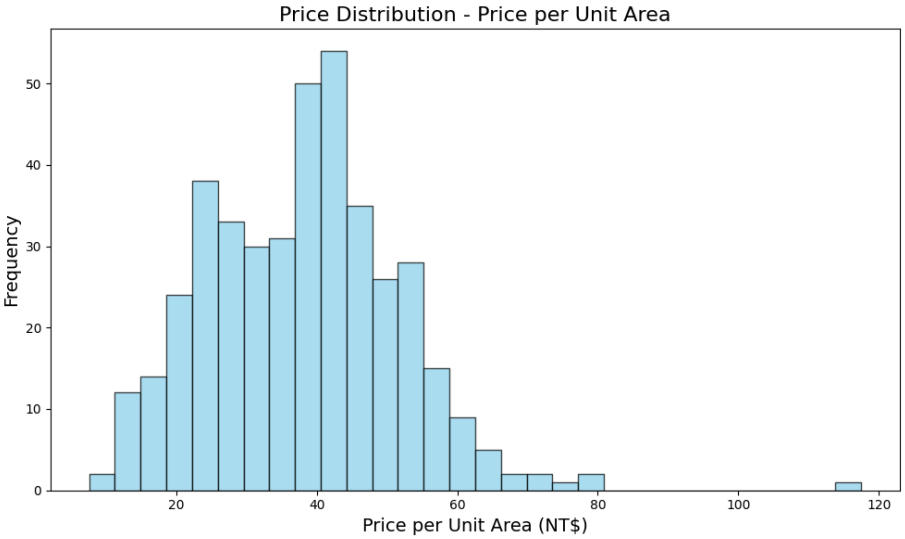


Figure 2. Histogram showing Price per Unit Area frequencies



The distribution of prices can be described by the following simple statistics:

1. Mean: The average price per unit area across all transactions is approximately 37.98 NT$.
2. Median: The price found at the center of the data set is 38.50 NT$.
3. Standard Deviation: Under the assumption of normality, the bulk of the data will be within 8.95 NT$ of the mean (37.98 NT$).

Data Preparation:

- The only data preparation I did before the initial modeling is ridding the data of redundancies found in the labeling of the variables. Each feature was unnecessarily numbered (X1, X2, …etc) and there was an additional “ID” row redundantly numbering each row.

Figure 3. Initial data preparation coded in Python

```python
# Rename columns for clarity
data = data.rename(columns={
    "No": "ID",
    "X1 transaction date": "Transaction Date",
    "X2 house age": "House Age",
    "X3 distance to the nearest MRT station": "Distance to MRT",
    "X4 number of convenience stores": "Convenience Stores",
    "X5 latitude": "Latitude",
    "X6 longitude": "Longitude",
    "Y house price of unit area": "Price per Unit Area"
})

# Drop the 'ID' column, as it does not contribute to prediction
data.drop(columns=["ID"], inplace=True)
```

Figure 4. Xindian District and surrounding areas

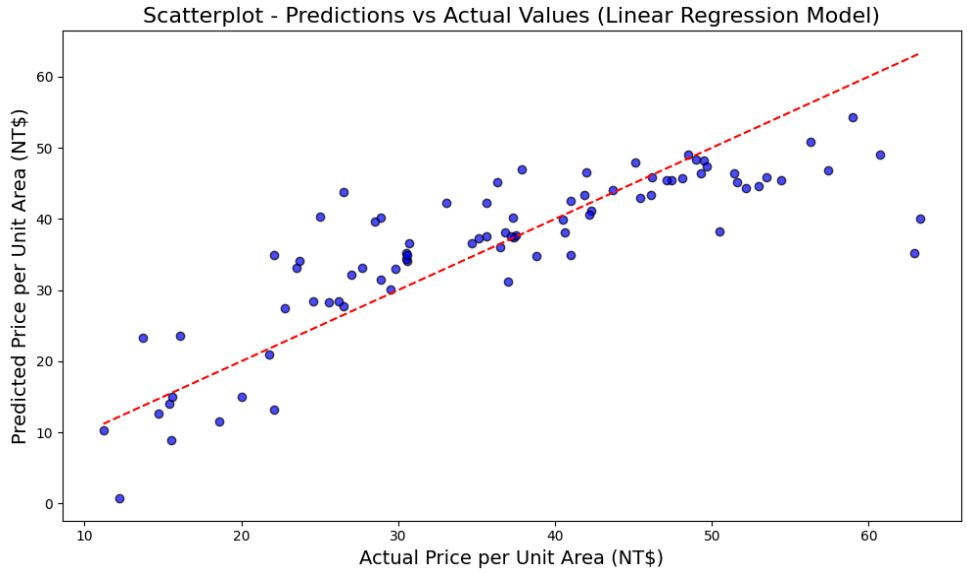
3. Modeling

For this project, I built three models, beginning with a linear/ridge regression model. The simple linear regression model produced an accuracy (r^2) of 0.6811. I then used ridge regression to add the parameter ‘alpha’ to the linear model. I created a hyperparameter grid for possible values of alpha and used the function GridSearchCV to find the best possible value for alpha as shown in the code below.

```python
# Define the Ridge Regression model
ridge_model = Ridge()

# Define a grid of hyperparameter values for alpha
param_grid = {'alpha': [0.01, 0.1, 1, 10, 100, 1000]}

# Use GridSearchCV to find the best alpha
grid_search = GridSearchCV(estimator=ridge_model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Best model and its parameters
best_ridge_model = grid_search.best_estimator_
best_alpha = grid_search.best_params_['alpha']
```

This resulted in an alpha value of 0.01, a fairly immaterial change, followed by a fairly immaterial increase in accuracy to 0.6818.

Figure 5. Test data plotted against predictions; dotted red line represents perfect predictions



I then moved to decision tree regression modeling, building an unrestrained model, then one with a limited depth to prevent overfitting. The unrestrained model performed poorly with an accuracy of 0.6038, however, after limiting the model's maximum depth and obtaining the optimal depth of 5, it performed well with an accuracy of 0.7901. This large jump in accuracy indicated that limited decision trees provided good representation of the data.

Figure 6. Decision tree with no depth limit

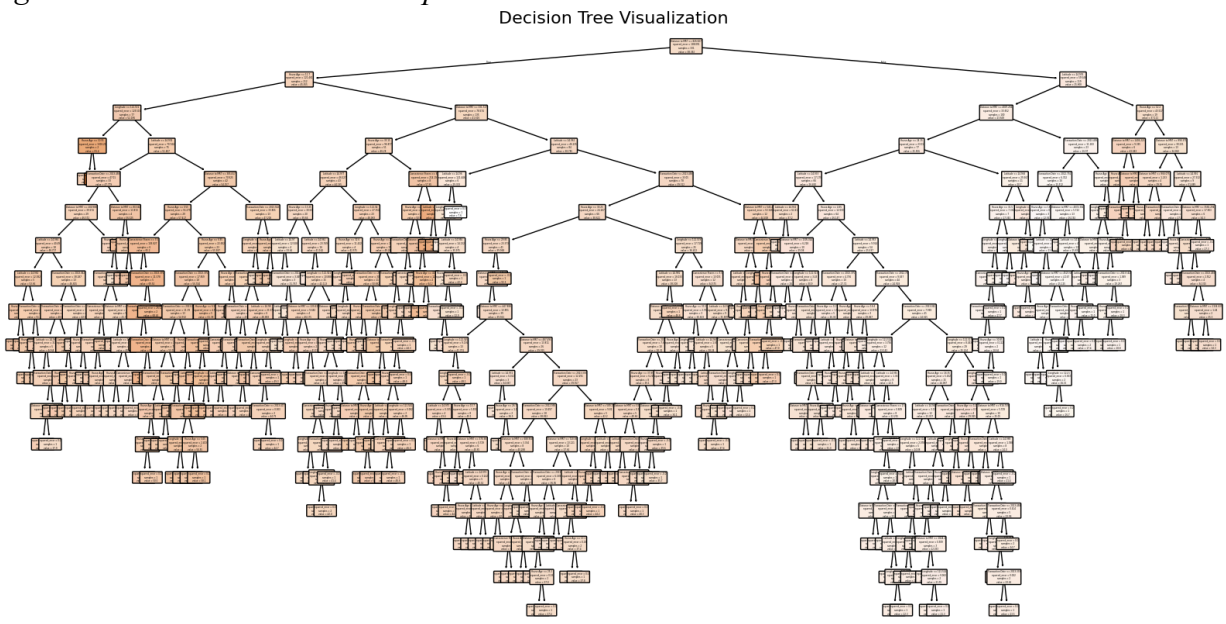
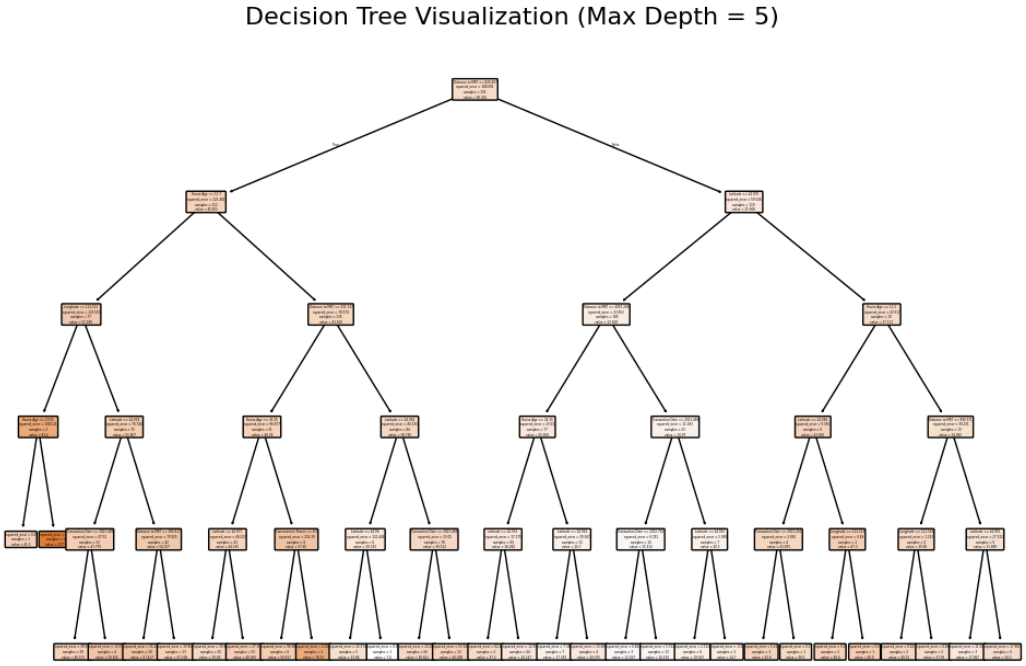


Figure 7. Decision tree with max_depth: 5



Due to the high accuracy of the limited decision tree model I decided to build a gradient boosting model, iterating limited decision trees in hopes of increasing the accuracy. To do this, I imported GradientBoostingRegressor from the sk.ensemble library. I first built the model with a basic form using 100 estimators (iterating 100 times), a 0.1 learning rate (each estimator used to determine 0.1 of the model) and a max_depth of 3 for each decision tree iteration. I then manually tested values for learning rate and max_depth (increments of 0.05 for learning rate and 1 for max_depth) and found that 0.1 and 3 are indeed the highest accuracy producing values. With the n_estimators component of the model, I used a parameter grid to test values [50, 100, 150, 200, 250, 300], finding the optimal estimator, 150. Here displayed is the code used to implement the model:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Define the parameter grid for 'n_estimators'
param_grid = {
    'n_estimators': [50, 100, 150, 200, 250, 300]
}

# Initialize the Gradient Boosting Regressor with default parameters
gb_model = GradientBoostingRegressor(random_state=42, learning_rate=0.1, max_depth=3)

# Perform Grid Search with Cross Validation to optimize 'n_estimators'
grid_search = GridSearchCV(estimator=gb_model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1)
grid_search.fit(X_train, y_train)

# Use the best model from Grid Search
best_gb_model = grid_search.best_estimator_

# Predict on the test data using the best model
y_pred_gb = best_gb_model.predict(X_test)
```

After obtaining the model I would be using, I tried two methods of feature engineering to produce more interpretable features and hopefully improve the accuracy of my model. The first features that were modified were latitude and longitude coordinates. I calculated the distance of each property to three different nearby high priced areas: Xinyi, Daan, and Zhongshan.

	Latitude	Longitude
Xinyi	25.0409	121.5720
Daan	25.0249	121.5434
Zhongshan	25.0792	121.5427

I then used the Haversine formula to calculate the distance between each given latitude and longitude coordinate for their respective data points and these coordinates.

```
# Function to calculate the distance between two geographical points
def haversine(lat1, lon1, lat2, lon2):
    # Convert degrees to radians
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    # Haversine formula
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat / 2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    radius_earth_km = 6371  # Earth's radius in kilometers
    return radius_earth_km * c
```

This produced three new features: [Distance from Xinyi, Distance from Daan, Distance from Zhongshan]. The model with these three features in place of Latitude and Longitude produced a high accuracy of 0.7703, but did not outperform the original model, thus I moved on to the second feature extraction.

I extracted three temporal features from Transaction Date: Transaction Year, Transaction Fraction, and Time Since Start.

1. Transaction Year: Transaction date in integer form to obtain year.
   - Allows for an annual view on price trends
2. Transaction Fraction: Transaction year subtracted from transaction date in order to obtain the fraction of the year that the sale took place (ex. 0.5 = middle of the year).
   - Allows for a seasonal view on price trends
3. Time Since Start: The earliest transaction date in the data set subtracted from the transaction date of the given data point.
   - Creates a continuous view of price trends, market growth or decline over the data collection period

Figure 8. Data head after change

	House Age	Distance to MRT	Convenience Stores	Latitude	Longitude	\
0	32.0	84.87882	10	24.98298	121.54024	
1	19.5	306.59470	9	24.98034	121.53951	
2	13.3	561.98450	5	24.98746	121.54391	
3	13.3	561.98450	5	24.98746	121.54391	
4	5.0	390.56840	5	24.97937	121.54245	

	Price per Unit Area	Transaction Year	Transaction Fraction	\
0	37.9	2012	0.917	
1	42.2	2012	0.917	
2	47.3	2013	0.583	
3	54.8	2013	0.500	
4	43.1	2012	0.833	

	Time Since Start
0	0.250
1	0.250
2	0.916
3	0.833
4	0.166

After building the model, analyzing the feature importances, and dropping the 'Transaction Year' feature (due to low importance and its negative effect on the model's performance), I achieved an accuracy of 0.8209 and an RMSE of 5.48. Here is the final model implementation code:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Drop 'Transaction Year' from the dataset
X_final_updated = X_updated.drop(columns=['Transaction Year'])

# Split the dataset into training and testing sets
X_train_final, X_test_final, y_train_final, y_test_final = train_test_split(X_final_updated, y_updated, test_size=0.2, random_state=42)

# Initialize the Gradient Boosting Regressor
gb_model_final_updated = GradientBoostingRegressor(random_state=42, n_estimators=150, learning_rate=0.1, max_depth=3)

# Train the model using the final training data
gb_model_final_updated.fit(X_train_final, y_train_final)

# Predict on the final test data
y_pred_final_updated = gb_model_final_updated.predict(X_test_final)

# Calculate R² score and RMSE for the final Gradient Boosting model
r2_final_updated = r2_score(y_test_final, y_pred_final_updated)
rmse_final_updated = np.sqrt(mean_squared_error(y_test_final, y_pred_final_updated))

# Display results
print(f"R² Score (Accuracy): {r2_final_updated:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_final_updated:.2f}")
```

4. Results

I found accuracies using linear regression, ridge regression, decision trees and gradient boosting models. After deciding to use a gradient boosting model, I found accuracies for three different feature combinations. The final model used to predict the value of real estate per unit was a Gradient Boosting Regression model. The final accuracy and RMSE achieved from the model was 0.8209 and 5.48 respectively.

Accuracy and RMSE for each model with base data:

	Accuracy	RMSE
Linear Regression	0.6811	7.31
Ridge Regression	0.6818	7.31
Decision Trees Unrestrained	0.6038	8.15
Decision Trees Max_Depth 5	0.7901	5.93
Gradient Boosting	0.8057	5.71

Accuracy and RMSE for Gradient Boosting models with various features:

	Accuracy	RMSE
Original Data Set	0.8057	5.71
Location Feature Derivation	0.7703	6.21
Temporal Feature Derivation	0.8209	5.48

Figure 9. Scatterplot of Predictions vs Actual Values for Final Model (Gradient Boosting with modified features)

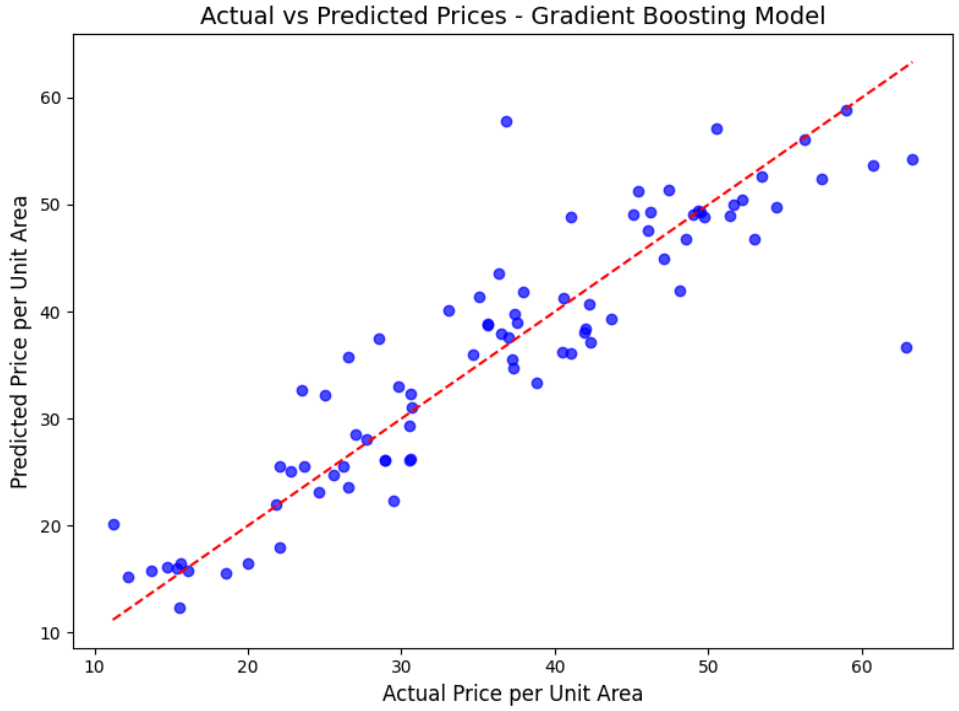


Figure 10. Histogram of the residual values displayed in figure 9

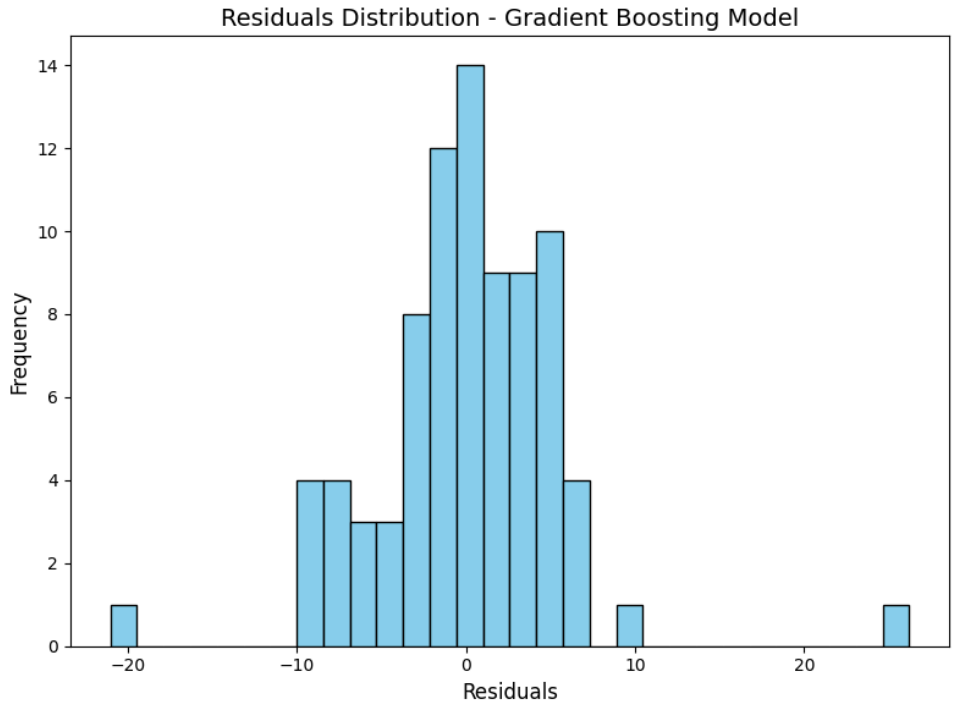
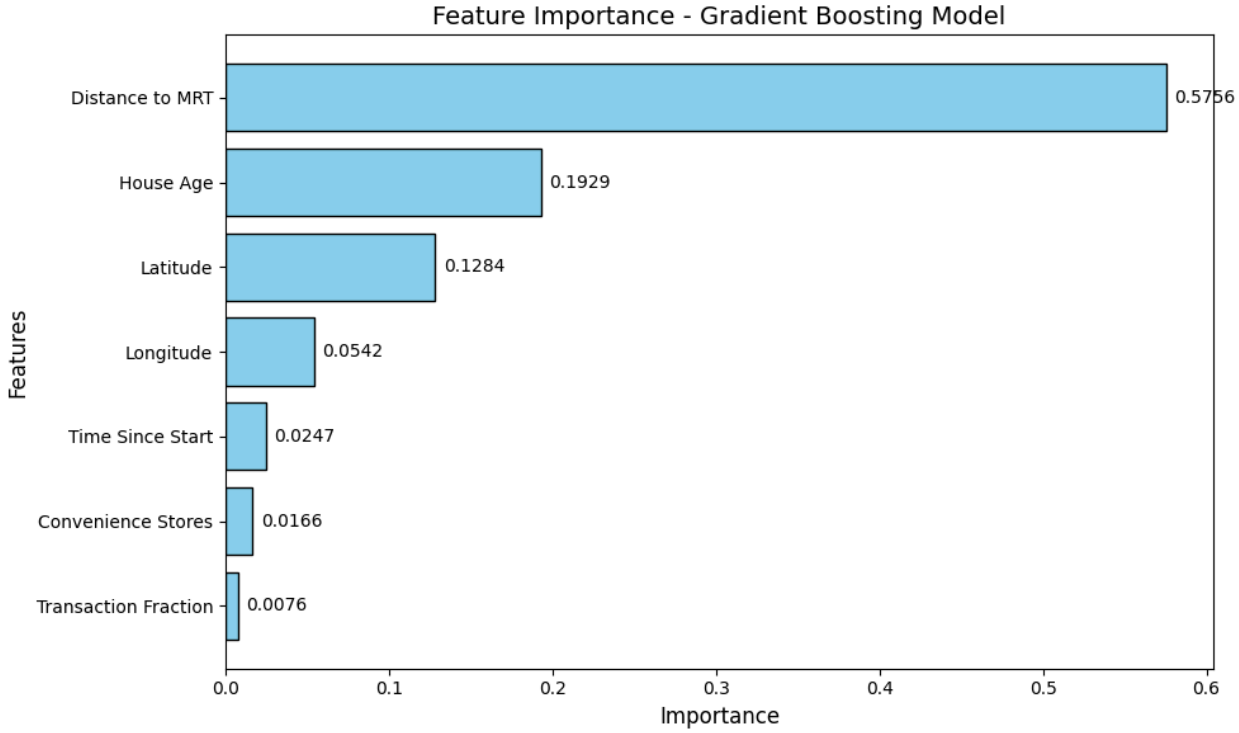


Figure 9 can be compared to Figure 5 to visualize the improvement from the first model created. The residuals of the distribution are greatly minimized as displayed in Figure 10.

Figure 11. Display of the final feature importances for the model



5. Discussion

The results of the project reveal that the Gradient Boosting Regression model outperformed other machine learning techniques in predicting real estate prices. Initial evaluations using the original dataset showed that simpler models like Linear Regression and Ridge Regression performed similarly, achieving low accuracies (0.681) and high RMSE values (7.31). These initial models struggled because they assume linear relationships between the features and the target variable. Real estate prices are influenced by complex, nonlinear interactions. Factors such as convenience and seasonal trends can rarely be measured linearly. As a result, both Linear and Ridge Regression models underfit the data, unable to capture the nuances necessary for accurate predictions.

In contrast, decision trees were able to partially account for the nonlinearity in the data. However, unrestricted decision trees (with no depth constraints) suffered from overfitting, memorizing the training data rather than generalizing patterns. This resulted in a low accuracy score of 0.6038 and a high RMSE of 8.15, as the model performed poorly on unseen test data. However, when the depth of decision trees was limited to five levels, their performance improved substantially (Accuracy 0.7901, RMSE 5.93). Limiting the depth effectively balanced model complexity, allowing the decision trees to generalize better while still capturing meaningful nonlinear patterns. As shown in Figure 6, decision trees with limited depth significantly reduced errors compared to unrestricted trees, demonstrating the importance of controlling overfitting in this dataset.

The Gradient Boosting model further built upon this success by combining multiple shallow decision trees to correct errors iteratively. This ensemble approach effectively captured complex, nonlinear relationships across features while maintaining a broad analysis of the data. Using the original dataset, Gradient Boosting achieved the best performance among all tested models, with an accuracy of 0.8057 and RMSE of 5.71. The iterative correction mechanism of Gradient Boosting allowed it to learn patterns that simpler models (linear, ridge, and individual decision trees) failed to recognize. This improvement is evident in Figure 7, which shows the Gradient Boosting model's predictions closely aligning with actual values compared to previous models.

The introduction of meaningful feature engineering further enhanced the Gradient Boosting model’s performance. Replacing geographic coordinates (Latitude and Longitude) with derived location features (distances to key districts like Xinyi and Daan) reduced accuracy slightly to an accuracy of 0.7703 and RMSE of 6.21. This indicates that these proximity features were not as meaningful as initially thought out to be. This is likely due to the fact that simple distances to high priced areas fail to properly capture area effects in complex city layouts. However, the inclusion of temporal features such as Transaction Year, Transaction Fraction, and Time Since Start significantly improved the model’s accuracy to 0.8209 and RMSE to 5.48. These features allowed the model to capture year-to-year trends and seasonal variations, offering insights into temporal factors influencing real estate pricing. As seen in Figure 11, Transaction Year did not offer a meaningful enough difference to contribute to the accuracy of the model, therefore it was dropped. Although small, the Transaction Fraction and Time Since Start features made noticeable contributions to the model.

Visual analyses further validated the Gradient Boosting model's superiority. In Figure 9, the scatterplot comparing actual versus predicted values showed points tightly clustered around the ideal diagonal line, highlighting the model's accuracy in predicting prices with minimal deviation. This contrasts sharply with scatterplots from earlier models, such as Linear Regression, shown in Figure 5, which exhibited substantial dispersion and more outliers. Figure 10, a histogram of residuals, demonstrated that the Gradient Boosting model's errors were narrowly distributed around zero, indicating minimal bias and consistent performance.

Overall, the results highlight the effectiveness of Gradient Boosting Regression, particularly when combined with meaningful feature engineering. Temporal features proved to be the most impactful, surpassing both the original dataset and location-derived features in predictive power. These findings underscore the importance of incorporating both spatial and temporal dimensions when modeling real estate prices and showcase the potential of Gradient Boosting to address complex, nonlinear relationships in structured data.

6. Conclusion

The following conclusions can be drawn from this study:

- The Gradient Boosting model produced the most accurate predictions for real estate per unit value.
- The most valuable feature in the data set with regards to predicting real estate per unit value was the Distance to the MRT.
- Linear models are often unable to capture complex and nuanced relationships, while decision tree models are able to account for nonlinearity, but must be limited to prevent overfitting.
- While not always, feature engineering can improve a model’s accuracy, bringing forth implicit influences to a modelable stage.

Further studies of real estate value can include economic implications such as GDP and inflation rates at the time of the build and selling of the property. Area metrics other than Distance to the MRT and Number of Convenience stores can be measured as well such as traffic and area popularity. The total size of the property can also be measured to see how per unit price scales with the total.

7. References

Yeh, I-Cheng. (2018). Real Estate Valuation. UCI Machine Learning Repository. https://doi.org/10.24432/C5J30W