ISYE 6740 FINAL PROJECT REPORT, SPRING 2024 MODELLING AND ANALYSIS OF BROOKLYN HOME PRICES FROM 2003 TO 2017

Ani Marellapudi, Karthik Annigeri, Ralph Andraos, & Ojas Majgaonkar
ISYE 6740, Spring 2024, Team Epsilon
Georgia Institute of Technology
Atlanta, GA 30332, USA
{am123, kannigeri3, omajgaonkar3, randraos3}@gatech.edu

ABSTRACT

This project report examines home prices in Brooklyn, New York, utilizing a dataset with over 390,000 entries and 111 features from 2003 to 2017. By employing advanced machine learning techniques such as Gradient Boosting and Random Forest, we aim to predict home price fluctuations and identify the key factors influencing these changes. This analysis processes the dataset through rigorous pre-processing, feature engineering, and data augmentation, resulting in the most impactful 41 numerical and 17 categorical predictors. The study highlights the complexities of real estate pricing influenced by diverse factors, offering insights that aid stakeholders in making informed investment and policy decisions. The outcomes demonstrate the potential of machine learning in transforming real estate analysis, providing a template for future research in other urban settings.

1 INTRODUCTION

In recent years, the real estate market has demonstrated significant volatility, influenced by various economic, demographic, technological, and geographic factors (Federal Reserve Bank of St. Louis; National Association of Realtors, 2024). Among urban areas, Brooklyn has emerged as a particularly interesting case study due to its diverse neighborhoods and complex socio-economic dynamics. This paper presents a comprehensive analysis of home prices in Brooklyn, New York, using machine learning models to predict home prices given 111 input features. Our objective is to provide a robust framework that can aid potential homeowners, investors, and policymakers in making informed decisions.

The importance of accurate home price prediction cannot be overstated, as it represents the single largest investment most individuals will make in their lifetime (U.S. Department of Housing and Urban Development, 2024). The challenges associated with predicting home prices stem from the multifaceted nature of the real estate market, where prices are influenced by an array of factors ranging from home architectural features to broader economic indicators.

Our approach utilizes a dataset encompassing over a decade of home sales in Brooklyn, with more than 390,000 data points and 111 features initially, refined to 41 numerical and 17 categorical variables through meticulous pre-processing to ensure quality and relevance. By applying Gradient Boosting and Random Forest models, we aim to harness the predictive power of the refined input features, achieving a nuanced understanding of the factors driving home prices in Brooklyn, NY.

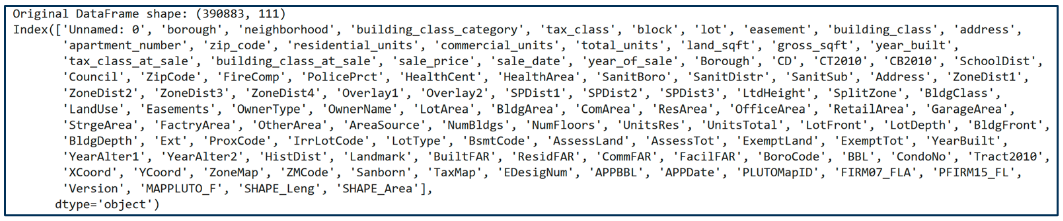
The subsequent sections of this paper detail our data pre-processing methods, exploratory data analysis (EDA), modeling techniques, results, and the implications of our findings. We discuss the strengths and limitations of our models and propose directions for future research, potentially extending our methodology to other regions and integrating more granular socio-economic data to enhance the accuracy and applicability of our predictions.

2 Data Collection and Pre-Processing

2.1 Dataset Overview

The primary dataset for this study was sourced from Kaggle, containing home sale prices in Brooklyn, NY, spanning from 2003 to 2017. Originally provided by NYC.gov, this dataset includes over 390,000 data points across 111 diverse features, encapsulating a broad spectrum of both numerical and categorical data relevant to home pricing (kag). The size of the raw dataframe was (390883, 111) as shown in Figure 1.

Initial data cleaning steps focused on the removal of extraneous data that could potentially skew our analysis. We removed columns with more than 30% missing values and manually eliminated duplicate or irrelevant columns, such as those related to dataset versioning and metadata. Additionally, variables showing more than 90% correlation with others were excluded to prevent multicollinearity (Figure 2). After these 2 steps, the dataframe size was reduced to (301220, 54).



```
Original DataFrame shape: (390883, 111)
Index(['Unnamed: 0', 'borough', 'neighborhood', 'building_class_category', 'tax_class', 'block', 'lot', 'easement', 'building_class', 'address',
       'apartment_number', 'zip_code', 'residential_units', 'commercial_units', 'total_units', 'land_sqft', 'gross_sqft', 'year_built',
       'tax_class_at_sale', 'building_class_at_sale', 'sale_price', 'sale_date', 'year_of_sale', 'Borough', 'CD', 'CT2010', 'CB2010', 'SchoolDist',
       'Council', 'ZipCode', 'FireComp', 'PolicePrct', 'HealthCent', 'HealthArea', 'SanitBoro', 'SanitDistr', 'SanitSub', 'Address', 'ZoneDist1',
       'ZoneDist2', 'ZoneDist3', 'ZoneDist4', 'Overlay1', 'Overlay2', 'SPDist1', 'SPDist2', 'SPDist3', 'LtdHeight', 'SplitZone', 'BldgClass',
       'LandUse', 'Easements', 'OwnerType', 'OwnerName', 'LotArea', 'BldgArea', 'ComArea', 'ResArea', 'OfficeArea', 'RetailArea', 'GarageArea',
       'StrgeArea', 'FactryArea', 'OtherArea', 'AreaSource', 'NumBldgs', 'NumFloors', 'UnitsRes', 'UnitsTotal', 'LotFront', 'LotDepth', 'BldgFront',
       'BldgDepth', 'Ext', 'ProxCode', 'IrrLotCode', 'LotType', 'BsmtCode', 'AssessLand', 'AssessTot', 'ExemptLand', 'ExemptTot', 'YearBuilt',
       'YearAlter1', 'YearAlter2', 'HistDist', 'Landmark', 'BuiltFAR', 'ResidFAR', 'CommFAR', 'FacilFAR', 'BoroCode', 'BBL', 'CondoNo', 'Tract2010',
       'XCoord', 'YCoord', 'ZoneMap', 'ZMCode', 'Sanborn', 'TaxMap', 'EDesigNum', 'APPBBL', 'APPDate', 'PLUTOMapID', 'FIRM07_FLA', 'PFIRM15_FL',
       'Version', 'MAPPLUTO_F', 'SHAPE_Leng', 'SHAPE_Area'],
      dtype='object')
```

Figure 1: Initial dataframe column (feature) names and size (390883, 111).

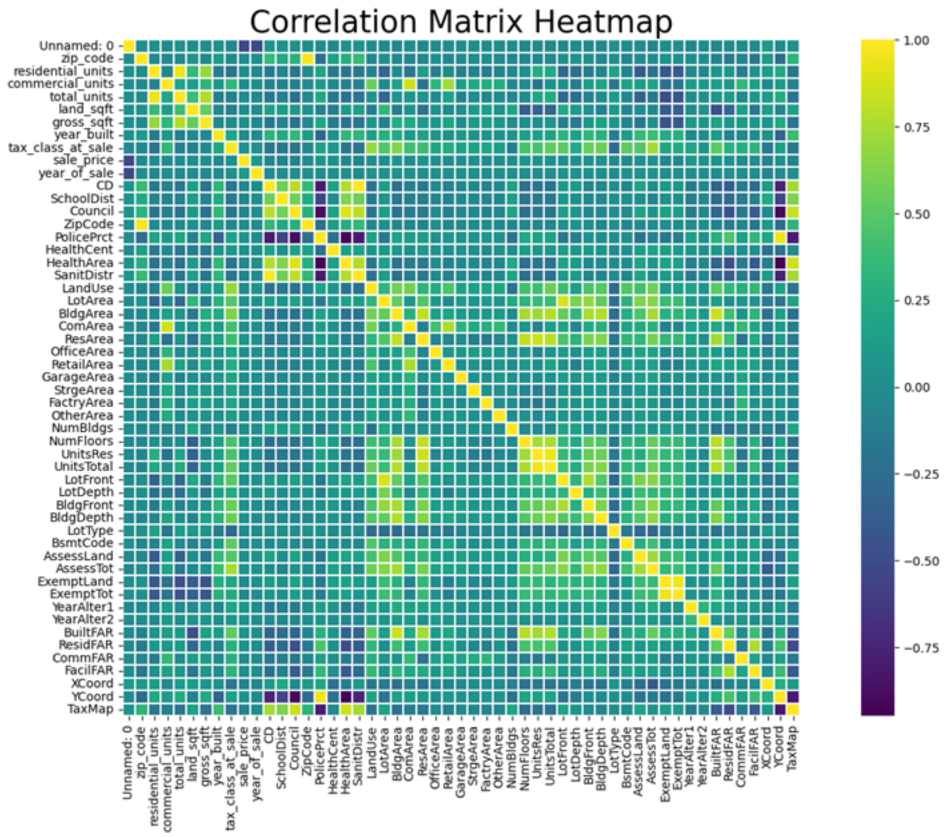


Figure 2: Initial dataset feature reduction using a correlation heatmap.

2.2 Preliminary Data Visualization

Visual inspection of the dataset revealed a long-tail distribution of sale prices, with values ranging from $20,000 to $171 million (Figure 3). The distribution of home sales showed a dip in home sale volumes during the 2008-2010 recession. Early visualization indicated no clear linear or non-linear relationships between the sale price and various property characteristics such as building area and year built (Figure 4). However, given the size and temporal span of the data set, it is understandable that single-variable visualizations do not show clear linear or non-linear relationships.

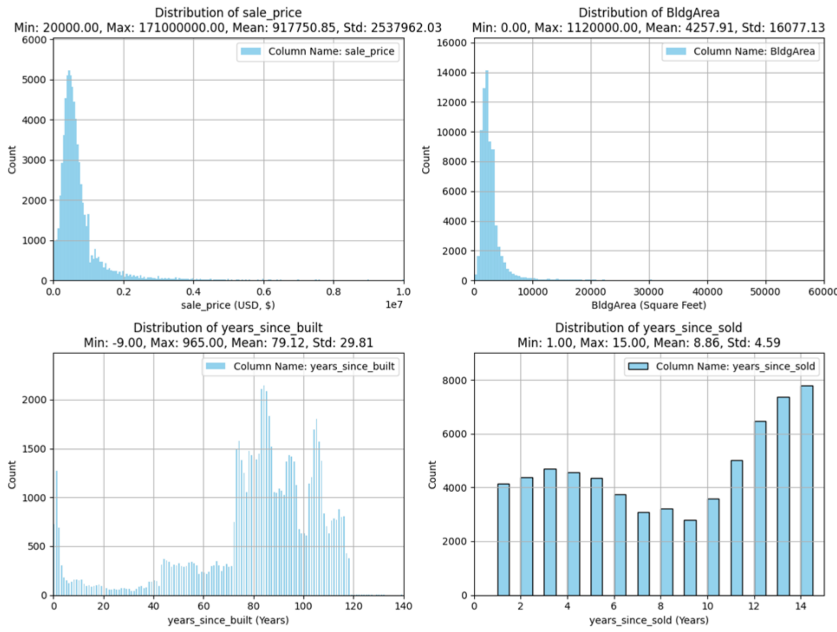


Figure 3: Distribution of sale prices, building area, years since construction, and years since sold.

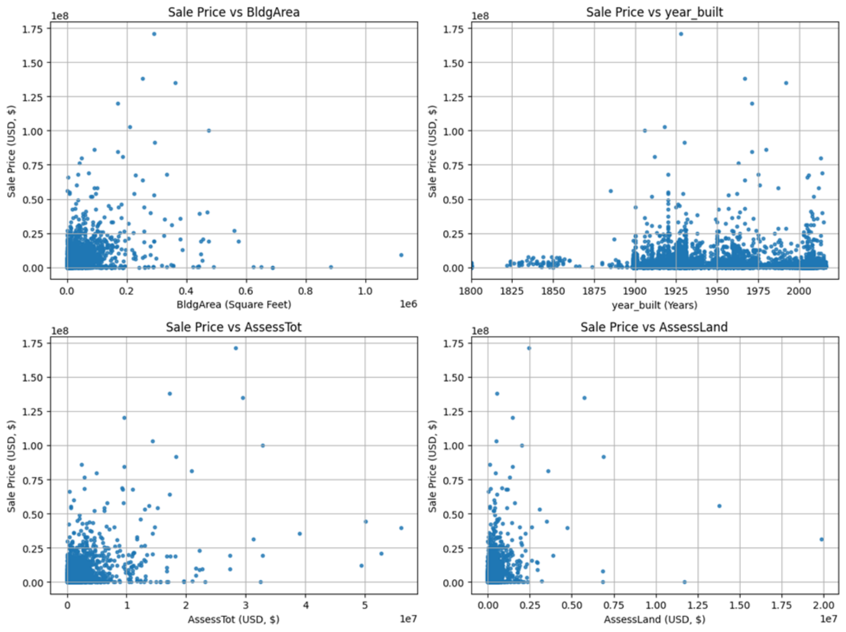
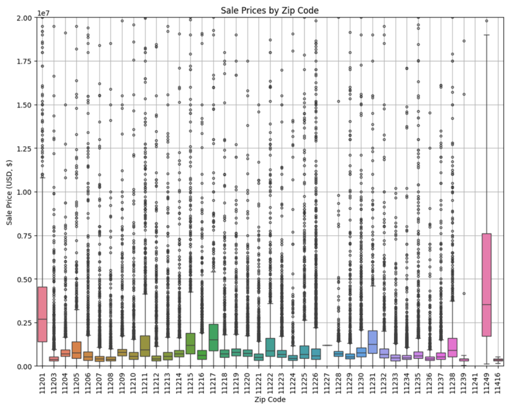
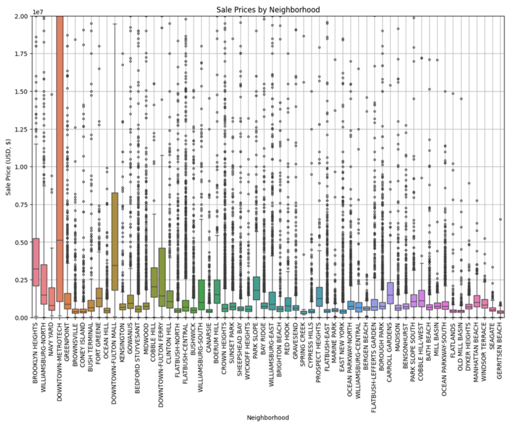


Figure 4: Building area, year built, assessed total value, and assessed land value versus sale price.

As expected, sale price showed strong dependence on categorical variables 'zip_code' and 'neighborhood.' Neighborhoods 'Downtown-Metrotech' and 'Downtown-Fulton All' seemed to have clearly higher median sale prices across all years (Figure 5).



(a)



(b)

Figure 5: Home price distributions plotted against categorical variables 'zip_code' (a) and 'neighborhood' (b).

2.3 Feature Engineering and Data Augmentation

After initial data cleaning, a more granular data processing phase was executed where additional categorical features, overlapping in impact, were removed. This included removing columns relating to proximity to fire departments and sanitation district, which were redundant given other location-based features like zip code and neighborhood.

To enhance the model's predictive power and interpretability, we engineered new features such as 'zip_code_rank' and rolling averages of prices ('average_price_3m', 'average_price_1y', 'average_price_2y'). Given that general market conditions can have a significant impact on home prices, the dataset was augmented with external economic indicators, including data from the S&P 500, IYR (iShares U.S. Real Estate ETF), and the US annual inflation rate, to capture the impact of broader economic conditions on real estate prices (Omokolade Akinsomi, 2016), as shown in Figure 6.

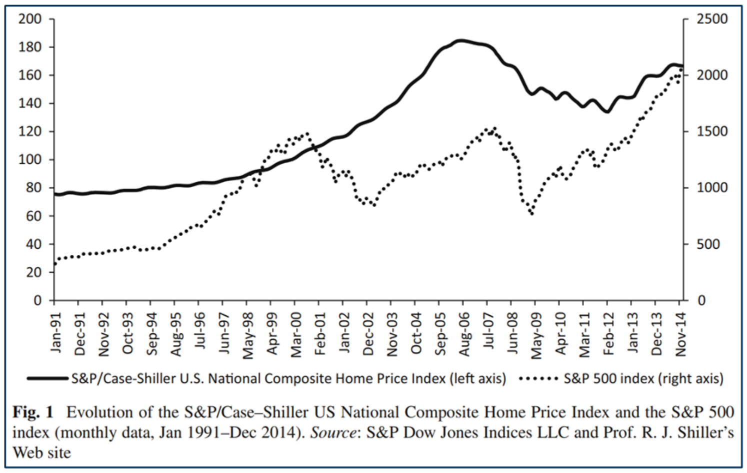
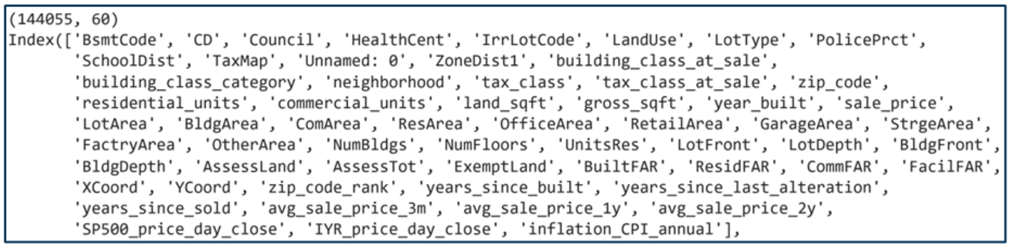


Fig. 1 Evolution of the S&P/Case–Shiller US National Composite Home Price Index and the S&P 500 index (monthly data, Jan 1991–Dec 2014). *Source*: S&P Dow Jones Indices LLC and Prof. R. J. Shiller's Web site

Figure 6: Historical data from the US between 1991 and 2014 showing the relationship between a national composite home price index and the S&P500 index.

2.4 Final Dataset Composition

The final dataframe after cleaning, pre-processing, and augmentation had 145,027 rows with 41 numerical variables and 17 categorical variables, as shown in Figure 7. Home sale price was the output variable. This pre-processed dataset not only facilitated more efficient computations but also honed the focus of our subsequent modeling efforts.



```
(144055, 60)
Index(['BsmtCode', 'CD', 'Council', 'HealthCent', 'IrrLotCode', 'LandUse', 'LotType', 'PolicePrct',
       'SchoolDist', 'TaxMap', 'Unnamed: 0', 'ZoneDist1', 'building_class_at_sale',
       'building_class_category', 'neighborhood', 'tax_class', 'tax_class_at_sale', 'zip_code',
       'residential_units', 'commercial_units', 'land_sqft', 'gross_sqft', 'year_built', 'sale_price',
       'LotArea', 'BldgArea', 'ComArea', 'ResArea', 'OfficeArea', 'RetailArea', 'GarageArea', 'StrgeArea',
       'FactryArea', 'OtherArea', 'NumBldgs', 'NumFloors', 'UnitsRes', 'LotFront', 'LotDepth', 'BldgFront',
       'BldgDepth', 'AssessLand', 'AssessTot', 'ExemptLand', 'BuiltFAR', 'ResidFAR', 'CommFAR', 'FacilFAR',
       'XCoord', 'YCoord', 'zip_code_rank', 'years_since_built', 'years_since_last_alteration',
       'years_since_sold', 'avg_sale_price_3m', 'avg_sale_price_1y', 'avg_sale_price_2y',
       'SP500_price_day_close', 'IYR_price_day_close', 'inflation_CPI_annual'],
```

Figure 7: Final dataset column (feature) names and size (145027, 60).

3 Modelling Approach 1: CatBoost

3.1 Model Description and Justification

The first model considered to predict Brooklyn home prices was Gradient Boosting, an ensemble method that involves training learners based on minimizing the differential loss function of a weak learner using a gradient descent optimization process. However, given the large number of categorical variables in the dataframe, a Gradient Boosting model with inherent categorical variable support was desired. This led to the usage of CatBoost (specifically CatBoostRegressor), an implementation of the Gradient Boosting algorithm with categorical variable support (CatBoost, 2024). The architecture is sequential, where decision trees are added stage-wise in series. Gradient Boosting, in general, focuses on reducing the bias. To avoid potential over-fitting issues, it is imperative to optimize parameters such as the regularization strength, tree depth and the number of estimators.

3.1.1 Description of Gradient Descent Algorithm

The algorithm underlying CatBoost uses an iterative sequence to update the model by stacking improved versions of the learners. This is achieved by solving an optimization problem at each step - minimizing the gradient of the loss function. The steps of the algorithm are detailed below:

Step 1: Initialize the boosting model. The initial model is given by:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma) \quad \text{for } m = 1, \ldots, M$$

Step 2: Evaluate the residuals (derivatives of the loss function). The residuals are calculated as follows:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{x_i = F_{m-1}(x_i)} \quad \text{for } i = 1, \ldots, n$$

Step 3: Fit a weak learner $h_m(x)$ (decision tree) to the residuals.

Step 4: Compute the multiplier γ_m by solving the following optimization problem.

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right)$$

Step 5: Update the model by stacking the improved learners in series.

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

The final boosted model is given by $F_M(x)$ where M is the number of estimators / weak learners in series.

3.1.2 Framework and Advantages of CatBoost

CatBoost has a number of key characteristics which make it better for data with many categorical variables and for noisy data which can cause over-fitting. Gradient Boosting, in general, is a greedy algorithm, which has a tendency to overfit the data and CatBoost helps to avoid that. The framework of the CatBoost algorithm is shown in Figure 8 on the following page.

1. **Encoding of categorical features:** CatBoost uses Ordered Target Encoding where the categories are encoded using the target over several random permutations to avoid target leakage. This works well when dealing with high cardinality features as the dimensionality of the encoded set is significantly smaller than other encoding methods such as One-Hot Encoding.
2. **Ordered boosting:** CatBoost trains the model on a subset of data while calculating residuals on another subset. This is in contrast to XGBoost and LightGBM, where the same data instance is used for both training and estimating the gradients. This allows the model to work with unseen data and helps in avoiding over-fitting (Dorogush et al., 2017).
3. **Balanced architecture:** CatBoost builds symmetric (balanced) trees unlike XGBoost and LightGBM which have asymmetric trees. In CatBoost, the feature-pair which yields the lowest log-loss is used for the split point condition at every level of the decision tree. This makes it computationally more efficient and helps in avoiding over-fitting.

Keeping these key features in mind, CatBoost is a very good choice of model selection for our dataset for the following reasons:

1. **Size of the dataset:** The dataset has around 145,000 points and 58 dimensions. CatBoost excels at computational speed compared to XGBoost and LightGBM.
2. **High cardinality features:** The dataset has 17 categorical features, some of which are having high cardinality. CatBoost excels at handling these features and uses Ordered Target Encoding which helps in reducing the training time.
3. **High variance:** The dataset has a wide variance in the target values which could bias the training in favor of outliers. CatBoost is highly robust to over-fitting compared to other Gradient Boosting models.

3.2 Model Initialization

For the Python implementation of CatBoost and Bayesian hyperparameter search, the latest versions of `catboost` and `scikit-optimize` libraries were used. The `CatBoostRegressor` and `BayesSearchCV` functions were imported from these libraries to initialize the CatBoost regression model and the hyperparameter search by Bayesian optimization.

Hyperparameter search was done for two different segments of the dataset: one search for the unrestricted dataset and one search for a restricted version of the dataset where sale prices greater than \$10M were neglected (leading to the removal of only about 1000/145,000 points). This was done to understand the impact of the outliers on the hyperparameter search and the resulting effect on the model's predictions. Ultimately, the model was trained only on the restricted set in both cases.

The cleaned and augmented, post-EDA dataset along with the .csv files containing the names of the numerical and categorical variables were imported directly onto a Google Colab cloud server (Google, 2024).

Standardization (standard scaling) was applied on the numerical features only, and each feature was transformed such that it had a mean of 0 and a standard deviation of 1 over all data points. This was done to negate the biasing effect of magnitude differences between features on the overall prediction. The standardization formula used was:

$$\bar{x} = \frac{x - \mu}{\sigma}$$

Finally, the data were split into training and testing sets. A train/test split of 80/20 was used. From the training set, we draw 25% of the data as a validation subset, specifically for tuning the hyperpa-

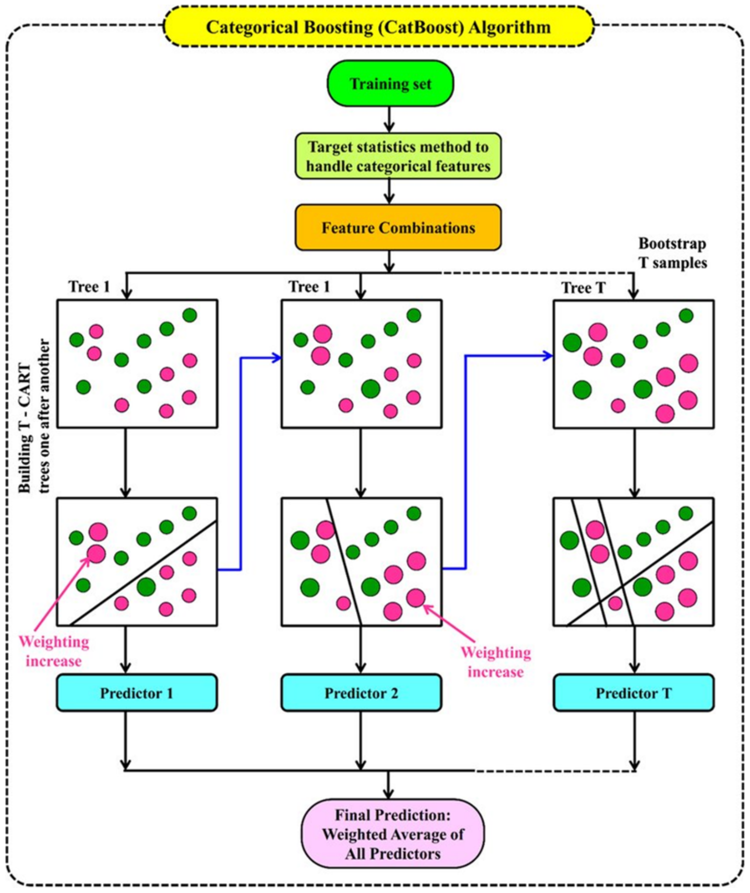

Categorical Boosting (CatBoost) Algorithm
Training set
Target statistics method to handle categorical features
Feature Combinations
Bootstrap T samples
Tree 1
Tree 1
Tree T
Building T - CART trees one after another
Weighting increase
Weighting increase
Predictor 1
Predictor 2
Predictor T
Final Prediction: Weighted Average of All Predictors

Figure 8: Framework of the CatBoost algorithm.

rameters before training the 'best' model on the full training set. This approach allowed for faster convergence of the hyperparameter tuning step.

The `CatBoostRegressor` model instance was then initialized, after which, the hyperparameters were tuned using `BayesSearchCV`. The below code snippet shows the model initialization step in Python.

```python
cbc = CatBoostRegressor(loss_function='RMSE', random_state=42,
                        one_hot_max_size=2, cat_features=categorical_features,
                        early_stopping_rounds=100)
```

The loss function for the `CatBoostRegressor` was set to RMSE (root mean squared error). The `one_hot_max_size` argument was set to 2 to ensure that categorical features with more than two unique categories were encoded using Ordered Target Encoding. One-Hot Encoding was used only for binary categorical features. The list of categorical features was passed to the model explicitly. The `early_stopping_rounds` argument was set to 100 to reduce training time. This effectively stopped the training for each learner if improvement in the loss function was not observed for more than 100 iterations.

3.3 Hyperparameter Tuning

A smaller validation set instead of the entire training set was used for hyperparameter tuning for faster convergence. Bayesian optimization was used for finding the optimal values of the hyperparameters over a specified search space. 5-fold cross-validation was performed during each iteration to avoid over-fitting. Bayesian optimization was implemented using the `BayesSearchCV` function imported from the `scikit-optimize` library.

`BayesSearchCV` is essentially a black-box optimizer which models the objective function on the basis of a Gaussian probability distribution and determines the hyperparameter values which would minimize that function. The sampling in the search space is done by acquisition functions. Each iteration of sampling the space attempts to maximize the acquisition function value, i.e., reach a trade-off between exploration and exploitation. The Gaussian process is updated with the prior (previous iteration) to get a new posterior that better approximates the objective function.

The Bayesian optimization procedure is as follows:

Step 1: For $t = 1, 2, \ldots$, find the next sampling point (x_t) by optimizing the acquisition function (u) over the Gaussian process:

$$x_t = \arg\max_x u(x|D_{t-1})$$

Step 2: Obtain a possibly noisy sample $y_t = f(x_t) + \epsilon_t$ from the objective function f.

Step 3: Add the sample to the previous samples:

$$D_t = D_{t-1} \cup \{(x_t, y_t)\}$$

Step 4: Next, update the Gaussian Process (GP).

The following hyperparameters were considered for the search:

1. **Iterations**: Equivalent to the number of estimators. It is a measure of overall model complexity.
2. **Learning rate**: Equivalent to the step-size in the optimization problem. It controls the speed of convergence.
3. **Depth**: The maximum depth of each decision tree. It is a measure of complexity of each estimator.
4. **L2 leaf reg**: Equivalent to the regularization strength in LASSO. It controls the overfitting.

The below code snippet shows the steps for defining the search space, initializing the Bayes tuner followed by fitting the tuner to the validation dataset which actively starts searching through the space to determine the best possible hyperparameter values that minimize the surrogate (objective) function.

```python
from skopt import BayesSearchCV
from skopt.space import Real, Integer

# Define the upper and lower limits for the hyperparameter search space:
search_spaces = {
    'depth': Integer(2, 10),
    'learning_rate': Real(0.01, 0.3, prior='log-uniform'),
    'iterations': Integer(100, 1000),
    'l2_leaf_reg': Integer(0, 5)
}

bayes_tuner = BayesSearchCV(
    estimator=gb,
    search_spaces=search_spaces,
    scoring='neg_mean_squared_error',
    cv=5,
    n_iter=100,
    refit=True,
    random_state=42
)

bayes_tuner.fit(X_val, y_val) # use the validation sets for hyperparameter tuning
best_model = bayes_tuner.best_estimator_
best_params = bayes_tuner.best_params_
```

Reasonably large ranges were given for the hyperparameter search spaces. The depth parameter had a maximum value of 16 and we considered a range of (2, 10). MSE was used as the scoring metric to evaluate the best model instance. The argument 'cv' was set to 5 for 5-fold cross-validation. This shuffles the validation set for each iteration to avoid over-fitting. The set of hyperparameters which give the best MSE were directly obtained using the 'best_params_' attribute.

After the 'best' model was found from hyperparameter tuning, we then trained the 'best' model on the full training set. The code snippet below shows this. Following the training, predictions on the testing set and evaluation of the prediction accuracy metrics were completed. CatBoost also has a 'get_feature_importance' method which returns the feature list sorted in the order of importance. These results were visualized and analyzed.

```
best_model.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=True, cat_features=
    categorical_features)
```

The values of the optimal hyperparameters are given in Table 1 for both segments of the dataset considered. It was observed that the iterations parameter always converged to the upper limit of the range. A grid search was attempted using GridSearchCV, only for the iterations parameter to consider values up to 2500. As the iterations were increased, the subsequent improvements in the model performance were only marginal. Thus, 2500 was decided to be a good stopping point with adequate trade-off between performance and speed of computation. Interestingly, the 'l2_leaf_reg' parameter converged to 0, indicating no LASSO (L2) regularization was incorporated in the loss function. This is indicative that over-fitting was not a concern with the CatBoost model.

Table 1: Optimal hyperparameter values determined for the CatBoost model considering different dataset segments.

Hyperparameter	**Range Specified**	**Optimal Values (Full dataset)**	**Optimal Values (Dataset with y <= $10M)**
Iterations	1000 - 2500 in steps of 250	2500	2500
Learning rate	(0.01, 0.3)	0.0191	0.0378
Depth	(2, 10)	5	8
L2 leaf reg	(0, 5)	0	0

3.4 Experimental Results

Next, we evaluated the accuracy of the predictions of the trained model on the testing set for both dataset segments discussed above. We consider two metrics: the R^2 score and the MSE/RMSE. The R^2 score is a measure of the extent to which the predictors used in the model account for the total variance in the full dataset and the MSE is simply a measure of the average deviation of the predicted values from the actual values. The MSE and RMSE values are calculated on the normalized test set targets. The coefficient of determination (R^2) and Mean Squared Error (MSE) are calculated as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Table 2: Model Performance Metrics

Model Instance	R^2 Score	MSE	RMSE
Tuned hyperparameters for full dataset	0.7107	0.3136	0.5599
Tuned hyperparameters for restricted dataset	0.7258	0.2959	0.5440

As seen in the results from Table 2, the R^2 score is reasonably good considering the noise and variance in the datasets. Visualizing the predictions against the actual values gives further insight into where the model performs well and where it performs poorly.

In Figure 9 and Figure 10, it is seen that due to sparse data in the extreme ends of the dataset, predictions are inaccurate for homes with very low prices or very high prices. High-priced houses are under-predicted and low-priced houses are over-predicted. This is expected as most of the data is centered around mean values and data is sparse at the extreme ends of the dataset. Overall, we conclude that the model does a reasonably good job at predicting home prices and the results indicate that the factors affecting the prices of very expensive houses (>=$10M) might be different, necessitating the use of separate models for different price ranges.

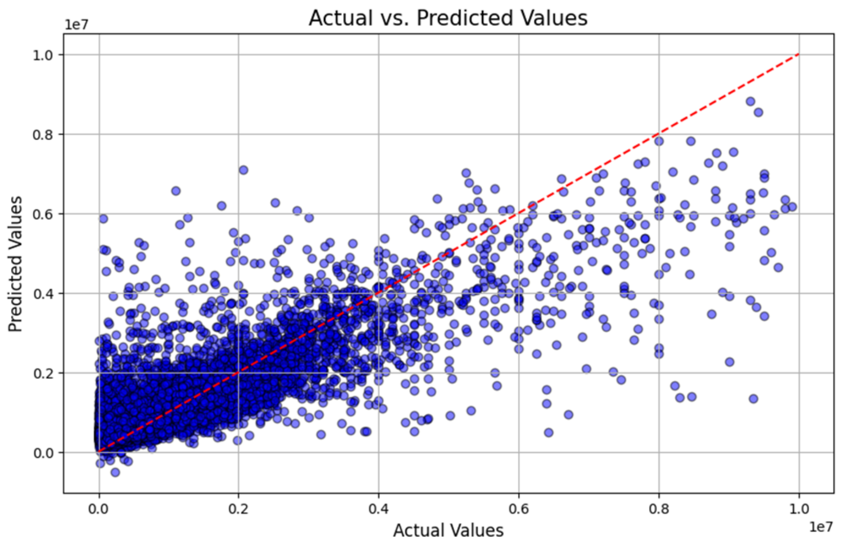


Figure 9: Visualization of performance of CatBoost model in predicting home prices under $10M, showing scatter plot of predicted versus actual home prices.

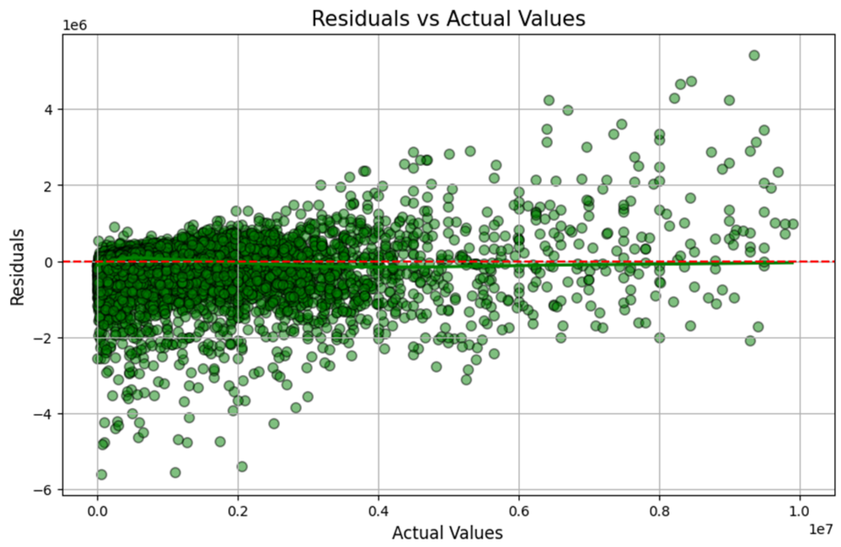


Figure 10: Visualization of performance of CatBoost model in predicting home prices under $10M, showing residuals of prediction errors.

3.5 Insights and Contributions

3.5.1 Feature Importances

Feature importance plots were then generated for the trained Catboost models to understand which features had the largest impact on predicted home prices. In addition, the differences in feature importances, when the houses priced greater than $10M were omitted from the dataset, were studied. Figure 11 shows the top features for the model trained on the entire dataset, while Figure 12 shows the top features for the model trained on the restricted dataset.

Overall, numerical variables 'AssessTot', relating the assessed total value of the home and property, and 'gross_sqft', relating to the gross square footage of the home, show high feature importance.

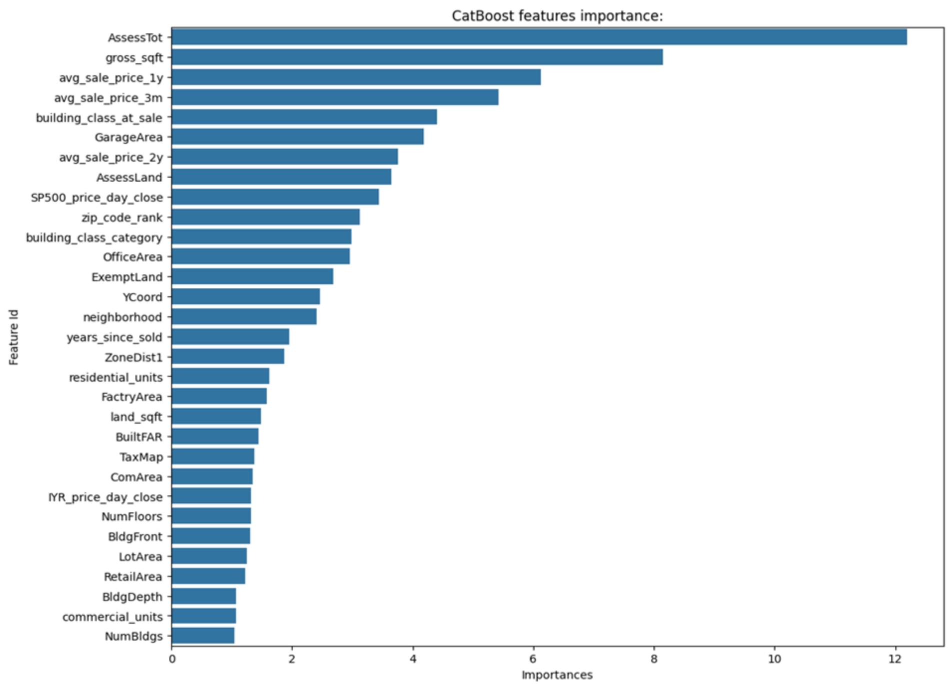


Figure 11: Feature importances of variables in the CatBoost model with the full dataset.

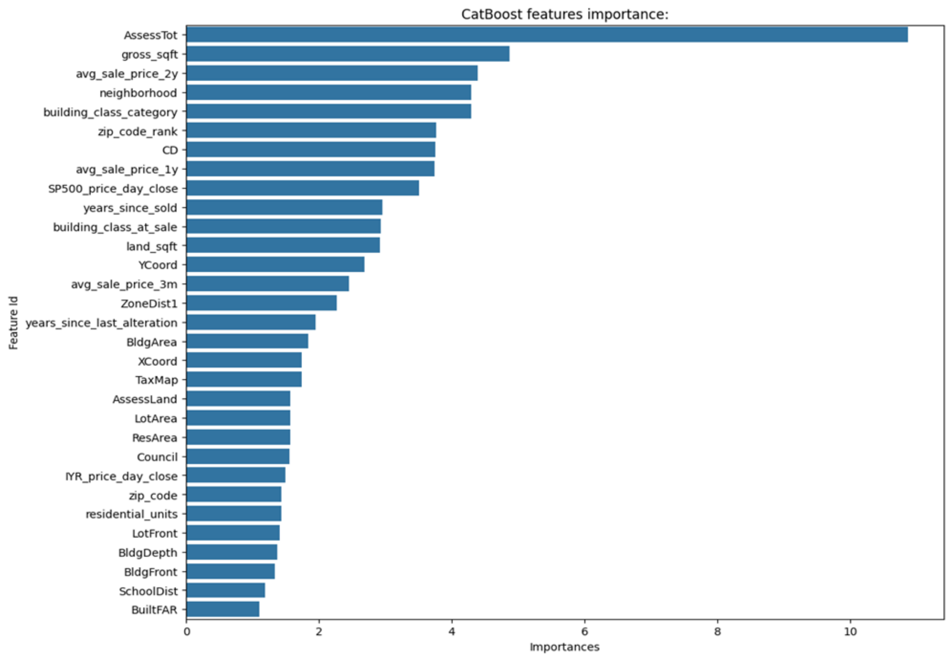


Figure 12: Feature importances of variables in the CatBoost model with the dataset restricted to home prices under $10M.

Several categorical variables related to location showed high importance such as 'neighborhood', 'zip_code_rank,' and 'CD'. This indicates that there is further scope to engineer the features and possibly combine these features into a proxy variable that encompasses the location-related details such as 'zip_code_rank,' which could simplify the model without decreasing prediction accuracy and performance. When the entire dataset is used, certain features such as ‘GarageArea’ and ‘Exempt-Land’ are given more importance considering that very expensive houses are taken into account, where these would be important factors impacting the prices.

In addition, the temporal dependence of home prices is well accounted-for by the model considering the relatively high feature-importances of the features such as ‘SP500_price_day_close,’ ‘years_since_sold,’ and ’avg_sale_price_2y’. These features were engineered to track the changes and fluctuations in market values including periods of recession (a major confounding factor for home prices). These variables contributed heavily to the final home price predictions as expected.

Feature importances were also visualized using SHAP (SHapley Additive ExPlanations) values, which are measures of the contribution of each feature to the overall prediction, as shown in Figure 13. The location of the dot horizontally shows the impact of that feature on the model’s output for that prediction. Features pushing the prediction higher are shown in red and those pushing the prediction lower are in blue. A SHAP value of zero means the feature did not change the prediction from the base value (the model’s average prediction over the training set). The further away from zero, the more impact the feature has. A vertical line of many dots suggests a common impact on the prediction, while scattered dots indicate varying impacts.

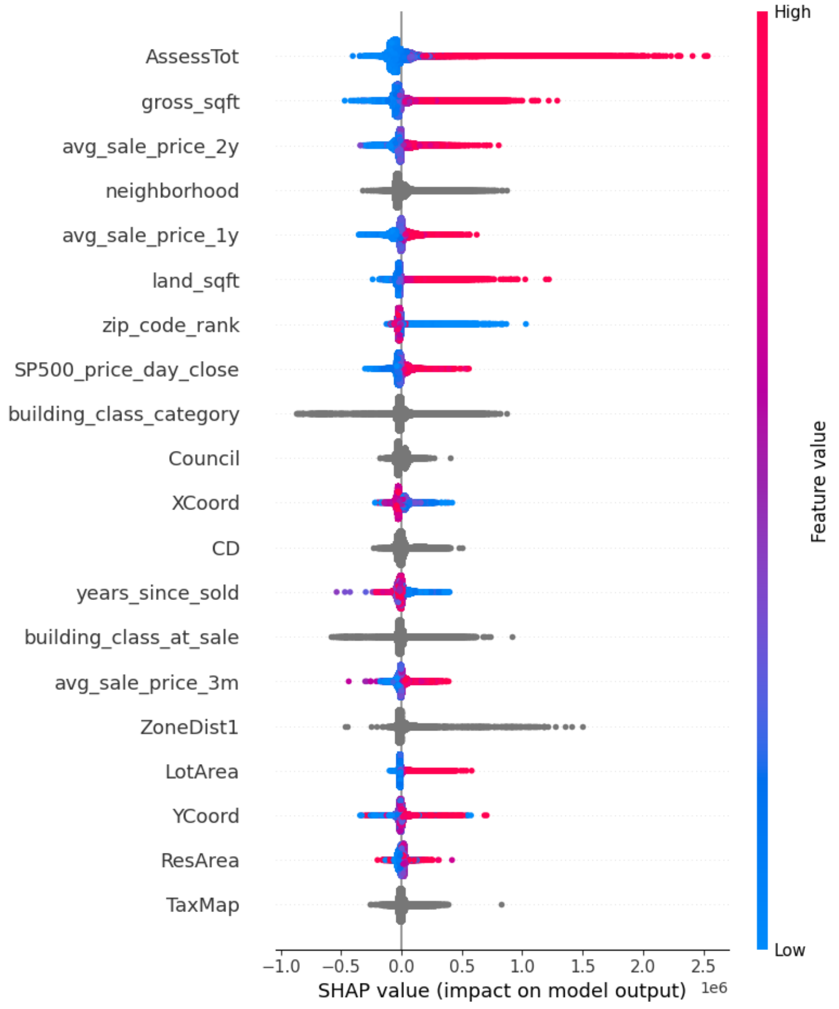


Figure 13: SHAP plot of the CatBoost model trained for home price prediction, showing another visualization method for feature importances.

4 Modelling Approach 2: Random Forest

4.1 Model Description and Justification

After an extensive evaluation of CatBoost, a decision was made to implement a Random Forest Regressor as another ensemble learning method to predict house prices. Random Forest leverages the collective strength of multiple decision trees to enhance prediction accuracy and stability, effectively countering over-fitting by averaging individual tree outcomes. This method not only surpasses the performance of single decision trees but also maintains robustness in the face of complex, non-linear data relations. One of the salient features of Random Forest is its inherent ability to rank the importance of various features, offering insight into the elements most critical to predicting the target variable. The selection of the Random Forest algorithm is further justified by its effectiveness at handling large datasets, an attribute particularly pertinent to our dataset comprising 140,000 data points with 41 numerical features. This method's scalability and efficiency ensure that the processing of our large dataset is both manageable and effective, with a tailored feature handling capability that enhances the overall predictive accuracy. Random Forest is also noise-robust, stemming from its ensemble nature.

In the practical implementation of our Random Forest model, hyperparameter optimization was paramount. The training of our models also adhered to a rigorous methodology, beginning with a train-test split of 80/20 to ensure a substantial training set while still providing a representative test set for unbiased evaluation. It is important to note that the data used for training were curated following the EDA process. Given the spread of the 'sale_price' variable, we trained distinct models across different price ranges. This stratification enabled our Random Forest models to specialize and capture the unique characteristics and trends within each price segment. Such a nuanced approach was instrumental in enhancing the model's precision and in providing more tailored predictions that reflect the varied dynamics of the real estate market.

In constructing the Random Forest model, we deliberately focused on numerical features, informed by insights gained from our preceding work with the CatBoost model. The analysis of CatBoost's feature importance revealed that significant categorical variables were predominantly related to location. This led us to the strategic decision to encapsulate the influence of location through the 'zip_code_rank' feature. We hypothesized that this singular, well-engineered feature could adequately represent the multifaceted impact of location on 'sale_price'. By distilling location into a single, ranked numerical feature, we simplified the model's feature space without compromising the nuance captured by more granular location data. This refinement allowed us to maintain the Random Forest model's performance while enhancing its interpretability and computational efficiency.

4.2 Hyperparameter Tuning

Through iterative loops, we meticulously tuned the number of trees, ensuring an optimal count that balances model complexity and performance. The tree depth was calibrated to prevent overfitting, allowing each tree to grow just deep enough to capture the nuances of the data without becoming overly specific. We also determined the minimum samples required to split a node and the minimum samples per leaf, which are critical in avoiding biases towards certain data points and maintaining a generalizable model. The final hyperparameters selections were made by identifying the model configurations that yielded the highest R^2. The results of the hyperparameter tuning shown in Figure 14.

Regarding the relationship between the number of estimators and the model R^2, as the the number of trees within the ensemble was increased, an incremental improvement in R^2 was observed, indicating a better fit to the training data. However, beyond a certain point, the gains in predictive accuracy were marginal compared to the substantial cost in computational efficiency. This diminishing return on investment led to the selection of 100 trees for the ensemble, a number that represents a balanced trade-off between model performance and computational demand.

Regarding the impact of varying the minimum number of samples required to split a node on the model's performance, it was observed that as we increased this threshold, there was a notable decline in model performance. This inverse relationship suggests that higher minimum split requirements may prevent the model from capturing essential patterns in the data. Consequently, to preserve the

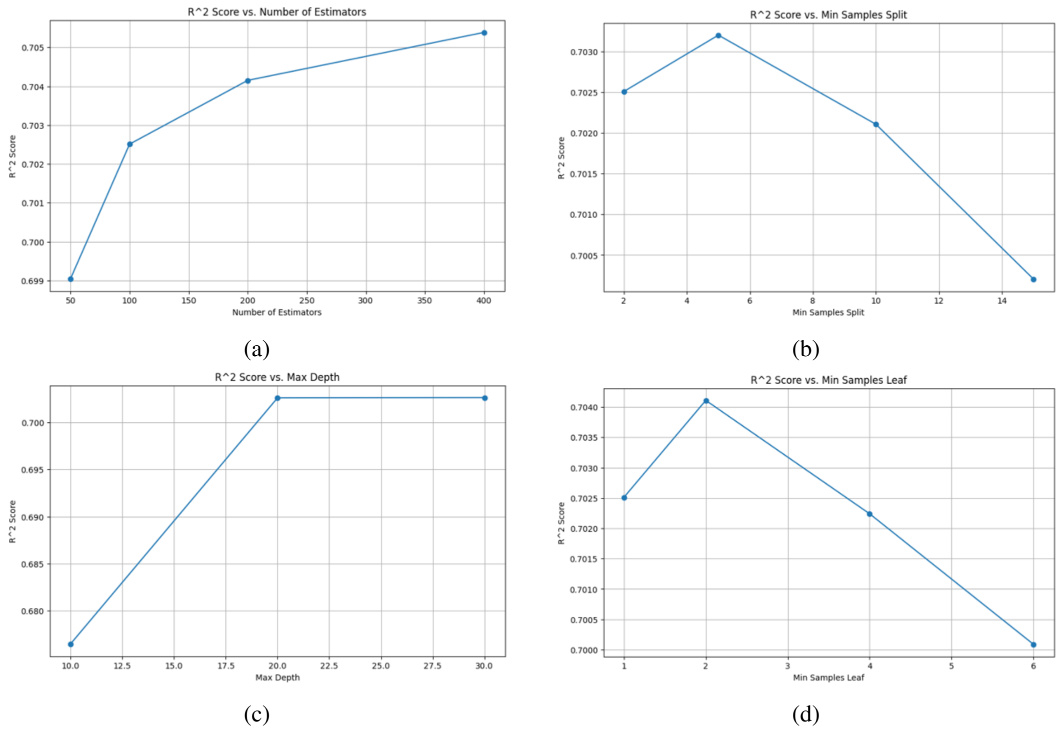


Figure 14: Hyperparameter tuning of the Random Forest model implemented to predict Brooklyn home prices.

model's sensitivity to data segmentation and its predictive accuracy, the minimum number of points needed to split a node was set to 2.

Regarding optimal tree depth, we identified an initial large improvement in performance with increasing depth, which eventually plateaued. This plateau indicates that deeper trees do not necessarily translate to better model performance and may instead lead to over-fitting. Hence, the default settings for tree depth were used, typically converging to a value of around 20. This decision aligns with the aim to balance model complexity with generalization.

Lastly, regarding the 'minimum samples per leaf' parameter, an increase in the threshold corresponded to a decrease in the model R^2. This trend suggested that larger leaf sizes may hinder the model's ability to make precise predictions, especially in the context of nuanced or sparse data regions. Consequently, we chose to maintain the initial setting of one sample per leaf to ensure the highest granularity in our predictions.

4.3 Experimental Results

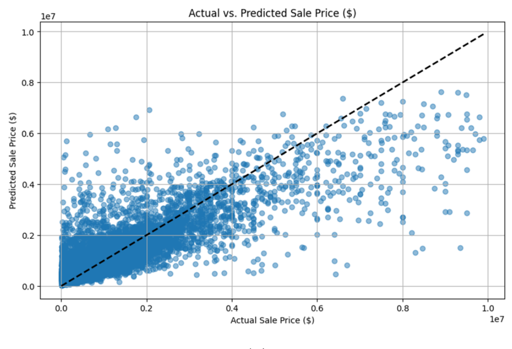
After optimizing the parameters, we trained a Random Forest Regressor on multiple subsections of our data. The results and model performance are shown in Table 3.

Table 3: Random Forest R^2 values across different home sale price ranges.

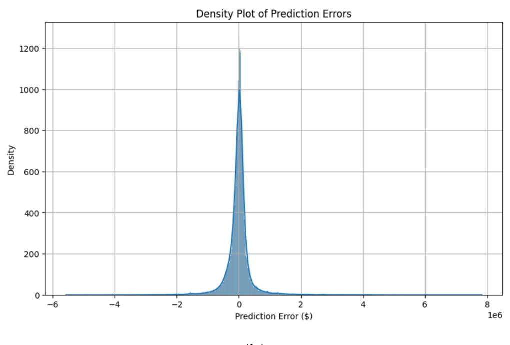
Home Sale Price Range	**$<=$ \$4M**	**$<=$ \$10M**	**\$10M-\$50M**	**$>$ \$50M**	**Whole Dataset**
Random Forest R^2	0.64	0.70	0.25	0.36	0.63

It must be noted that the size of the dataset for homes with sale prices exceeding \$10M was relatively limited, implying a potential deficiency in the volume of data required for the model to uncover reliable patterns. Furthermore, it was observed that the model's performance improved when the data was confined to a maximum sale price of \$10M. This suggests that homes at the higher end of the price spectrum may be influenced by different pricing criteria compared to those at lower price points, indicating a distinct market segment with unique characteristics. We selected the best performing model (sale price up to $10M) for subsequent analysis.

To visualize the model performance, Figure 15(a) shows predicted vs actual values. Overall, there is a good enough alignment with the dashed line representing perfect prediction. However, on the upper range of sale prices, the trained Random Forest model is undervaluing home prices while on the lower range, the model is overvaluing home prices. A density graph for prediction errors is shown in Figure 15(b), showing distribution of error centered at 0. This means that on average, the Random Forest model does a good job at predicting values with the majority of errors being small. The final performance of our model is summarized in Table 4.



(a)



(b)

Figure 15: Visualization of performance of Random Forest model in predicting home prices under $10M. (a) Scatter plot of predicted versus actual home prices, (b) density plot of prediction errors.

Table 4: Performance of final Random Forest R^2 model, considering home sale prices under $10M.

R^2	**MSE**	**RMSE**
0.70	0.30	0.55

4.4 Insights and Contributions

4.4.1 Investigation of Feature Importances

After finalizing the training and performance analysis of the Random Forest model, we wanted to gain insights about the importance of different features. Figure 16 provides a bar chart of the top 10 most important features. The most pivotal feature influencing our real estate pricing predictions was the total assessed value ('AssessTot'), with recent average sale prices over one year, two years, and three months ('avg_sale_price_1y', 'avg_sale_price_2y', 'avg_sale_price_3m') also demonstrating considerable predictive strength. Property size metrics, specifically square footage ('gross_sqft') and building area ('BldgArea'), also emerged as significant size indicators in property valuation. The role of location was distinctly highlighted by the zip code ranking ('zip_code_rank') and geographical coordinates ('YCoord'), reaffirming location as a critical determinant in real estate pricing. Additionally, the inclusion of the S&P 500 closing price ('SP500_price_day_close') as a feature acknowledges the subtle yet tangible impact of broader economic conditions on real estate markets, although their effect is less direct than that of property-specific factors.

4.4.2 Understanding Temporal Dependence of Model Performance

After reviewing the initial model results, we sought to discern the significance of temporal dynamics in property valuation. To this end, a new Random Forest model was trained using data spanning from 2003 to 2009 (using the previously optimized hyperparameters). We then evaluated this model's efficacy on a dataset covering home prices between 2010 to 2017. Figure 17 illustrates the comparison between the model's predicted values and the actual sale prices observed during this latter period.

The Random Forest model trained on 2003-2009 data produced a marked decline in performance relative to the original model (trained on data from all years, 2003-2017), with the R^2 metric falling

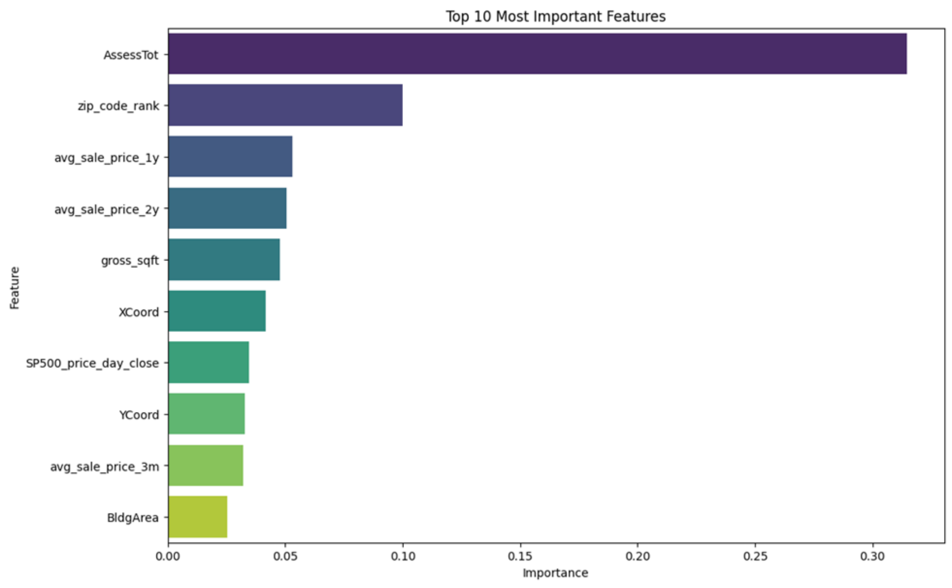


Figure 16: 10 most important features in the trained Random Forest model.

from 0.70 to 0.50. This substantial reduction in R^2 indicates that, although the model gleaned certain patterns from the historical data, it struggled to generalize effectively to more recent data. This underscores an evolution in market trends and property pricing mechanisms over time, highlighting the critical role that temporal factors play in accurate real estate valuation.

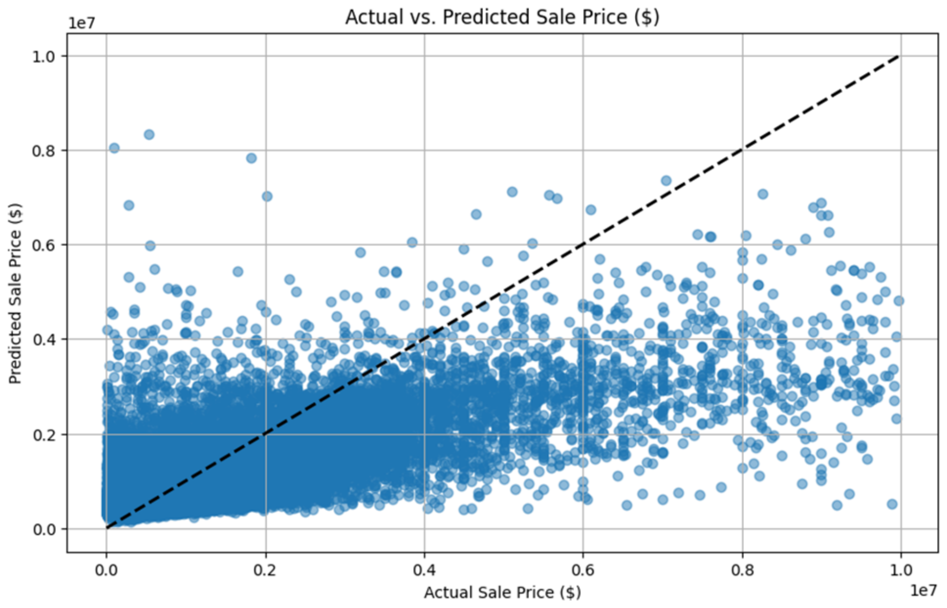


Figure 17: Actual versus predicted home sale prices for a Random Forest model trained on home prices between 2003-2009, and tested on home prices from 2010-2017.

5 Conclusion

This project on the modeling and analysis of Brooklyn home prices from 2003 to 2017 has yielded substantial insights into the real estate market through the application of advanced machine learning techniques like Gradient Boosting (CatBoost) and Random Forest. Both models demonstrated robust capabilities in predicting home prices, with Gradient Boosting achieving an R^2 score of 0.72 and Random Forest slightly behind at 0.70. These models have been instrumental in highlighting key predictive features such as 'zip_code_rank', 'YCoord', rolling averages (3mo, 1yr, 2yr), and S&P500, proving the intricate connections between property values and their geographic and economic contexts.

In our study, the CatBoost model proved to be particularly effective in handling the diverse and categorical-rich dataset of Brooklyn home prices. CatBoost, known for its robustness against over-fitting in datasets with numerous categorical features, achieved an R^2 score of 0.72. Its unique approach to handling categorical variables through Ordered Target Encoding and its use of a symmetrical tree structure helps mitigate the risk of over-fitting that is typical in traditional Gradient Boosting implementations. Additionally, CatBoost's ability to train on a subset of the data while using another subset for gradient estimation allows it to generalize better to unseen data. This makes it an excellent choice for our complex dataset, where predictive accuracy is critical for understanding nuanced market dynamics.

On the other hand, the Random Forest model also performed well with an R^2 score of 0.70, demonstrating its strength in ensemble learning through averaging multiple decision trees to reduce variance without sacrificing bias. Despite the intentional omission of categorical data, this model's performance is comparable to CatBoost, especially when considering variables like 'zip_code_rank' and 'gross_sqft' which are pivotal in predicting home prices. This finding paves the way for future model refinement and the potential simplification of complex models without substantial loss in predictive accuracy. Random Forest's inherent feature ranking capability provides valuable insights into which variables most significantly impact home values. It excels in environments with less risk of over-fitting and where the interpretability of a large number of individual trees can be leveraged to gain deeper insights into the data's underlying patterns.

Regarding real-world applications, the findings suggest that high-performance home price estimation tools could be significantly beneficial for platforms like Zillow, Opendoor, and Trulia. These tools would not only enhance consumer experiences by providing better data-driven insights but also refine investment strategies and policy-making in real estate. The feature importance analysis conducted for both models underscores the potential for real-time, dynamic pricing tools that adapt to ongoing market conditions, providing a competitive edge in the fast-paced real estate sector.

Future research could take several directions. Different models could be tailored for specific price segments or geographic areas to address the varying dynamics observed across different market segments. Additionally, incorporating time-series analyses to track homes over time and using advanced techniques like Named Entity Recognition (NER) to better utilize unstructured data such as text descriptions could enhance the predictive accuracy. This continued evolution in modeling techniques will pave the way for more sophisticated and nuanced real estate analytics, driving forward the capabilities of data science in property valuation.

A Appendix

All code written for this project, in the form of Python Jupyter notebooks, is available at the following GitHub link: `https://github.com/kannigeri3/ISYE-6740-Team-Epsilon-Project/`.

The final presentation presented to Professor Zhao on 05/02/2024 is available at the following GitHub link: `https://github.com/kannigeri3/ISYE-6740-Team-Epsilon-Project/blob/main/ISYE_6740_Final_Presentation_05_02.pdf`.

REFERENCES

Brooklyn home sales (2003-2017). https://www.kaggle.com/datasets/tianhwu/brooklynhomes2003to2017/data?select=brooklyn_sales_map.csv. Accessed: 2024-05-03.

CatBoost. Catboost: gradient boosting with categorical features support, 2024. URL https://catboost.ai. [Online; accessed 2-May-2024].

Anna Veronika Dorogush, Aleksandr Vorobev, and Andrey Gulin. Fighting biases with dynamic boosting. *arXiv preprint arXiv:1706.09516*, 2017.

Federal Reserve Bank of St. Louis. All-transactions house price index for new york. https://fred.stlouisfed.org/series/NYSTHPI. Accessed: 2024-05-02.

Google. Google colaboratory, 2024. URL https://colab.research.google.com/. Accessed: 2024-05-02.

National Association of Realtors. County Median Home Prices and Monthly Mortgage Payment. https://www.nar.realtor/research-and-statistics/housing-statistics/county-median-home-prices-and-monthly-mortgage-payment, 2024. Accessed: 2024-05-02.

Vassilios Babalos Fotini Economou Rangan Gupta Omokolade Akinsomi, Goodness C. Aye. Real estate returns predictability revisited: novel evidence from the us reits market. *Empirical Economics*, 2016. doi: 10.1007/s00181-015-1037-5. URL https://link.springer.com/article/10.1007/s00181-015-1037-5.

U.S. Department of Housing and Urban Development. Wealth accumulation and homeownership, 2024. URL https://www.huduser.gov/publications/pdf/wealthaccumulationandhomeownership.pdf. [Online; accessed 1-May-2024].