Predicting Selling Price of Apartments in Queens, NY.

Capstone Project for MATH342W: Introduction to Data Science and Machine Learning
City University of New York, Queens College

Loyd Flores

May 26, 2024

1 Abstract

This paper introduces machine learning models designed to predict the final selling price of apartments in Queens, NY. Leveraging a diverse dataset encompassing property features and historical sales data. By integrating both traditional real estate metrics and statistical methodologies, these models offers a robust tool for real estate professionals, investors, and prospective buyers to make informed decisions. The results demonstrate the model's potential to transform property valuation processes, providing precise and reliable price predictions in a dynamic and competitive market.

2 Introduction

A predictive model is a statistical tool designed to estimate an outcome, in this case, the sale price of an apartment, based on input data. Here, the unit of observation is individual apartments, and the response, or output, is the sale price in USD. The model utilizes supervised learning, where the model is trained on a labeled dataset $\mathcal{D}$, comprising known features, comprising known features X (such as the number of bedrooms, square footage, location) and their corresponding sale prices y The modeling process involves several crucial steps to ensure accuracy and reliability. Initially, the data is cleaned to remove any inconsistencies or errors. Next, the missing values are visualized and imputed to create a complete dataset. Following this, various models are selected and evaluated to determine the most effective one for this specific predictive task. Finally, the chosen model, $g(x)$, is deployed to predict sale prices for new apartments.

Preliminary performance results indicate that the model achieves high accuracy, obtaining R^2 of 97% and 38,064 **Out of sample RMSE** demonstrating its potential to transform property valuation in Queens. However, detailed performance metrics and validation will be discussed in subsequent sections.

3 The data

The dataset used in this project, titled `\textit{housing_data_2016_2017.csv}`, comprises raw housing data from 2016 and 2017 sourced from the Multiple Listing Service (MLS). This data was collected using Amazon's Mechanical Turk (MTurk) and represents a direct download from their system. Initially, the dataset contained 2,230 rows and 55 columns. After cleaning and removing outliers, the final dataset consisted of 2,179 rows and 21 columns.

The dataset is representative of the population of interest, which includes apartment sales across eight distinct areas within Queens, NY. This comprehensive coverage ensures that the dataset accurately reflects the diverse housing market in Queens, making the findings applicable and useful to residents and stakeholders in this area.

Supplementing the primary dataset with additional sources was not necessary due to its extensive coverage. However, it is essential to note the presence of outliers, which were addressed during the data cleaning process to enhance model accuracy. While the dataset provides a robust basis for predictions within the observed range, caution is advised when extrapolating beyond the historical data, as this may introduce potential inaccuracies due to unobserved variables and market

3.1 Featurization

The original dataset **D** had 55 columns, but the cleaned dataset has only 21 features. The omitted columns, such as CreationTime, URL, and RequesterFeedback, were necessary for data entry but had no predictive value in our context. These fields were essential for form completion and database storage but not useful for our analysis. The 21 columns could be classified into 5 subgroups:

1. Features providing temporal and geographical context
   (a) sale-year: Year property was sold.
   (b) zone: Obtained from extracting zip code from the full address and getting zones based off of the specific neighborhood

`Northeast Queens` 11361 11362, 11363 11364
`North Queens` 11354 11355 11356 11357 11358 11359 11360
`Central Queens` 11365 11366 11367
`Jamaica` 11412 11423 11432 11433 11434 11435 11436
`Northwest Queens` 11101 11102 11103 11104 11105 11106
`West Central Queens` 11374 11375 11379 11385
`Southeast Queens` 11004 11005 11411 11413 11422 11426 11427 11428 11429
`Southwest Queens` 11414 11415 11416 11417 11418 11419 11420 11421
`West Queens` 11368 11369 11370 11372 11373 11377 11378

```
zones = {
    1: [11361, 11362, 11363, 11364],
    2: [11354, 11355, 11356, 11357, 11358, 11359, 11360],
    3: [11365, 11366, 11367],
    4: [11412, 11423, 11432, 11433, 11434, 11435, 11436],
    5: [11101, 11102, 11103, 11104, 11105, 11106],
    6: [11374, 11375, 11379, 11385],
    7: [11004, 11005, 11411, 11413, 11422, 11426, 11427, 11428, 11429],
    8: [11414, 11415, 11416, 11417, 11418, 11419, 11420, 11421],
    9: [11368, 11369, 11370, 11372, 11373, 11377, 11378]
}
```

   (c) community-district-num: District number of property
   (d) approx-year-built
2. Binary categorical features, represented as 0 or 1
   (a) cats-allowed
   (b) dogs-allowed
   (c) coop-condo
   (d) garage-exists
3. Count variables quantifying specific attributes
   (a) num-bedrooms
   (b) num-floors-in-building
   (c) num-full-bathrooms
   (d) num-half-bathrooms

4. Categorical levels transformed into ordinal factors, ranked by mean sale price. The higher the number, the higher the rank.

(a) dining-room-type

```
dining_room_type_scaler = { 'none' : 1,
                            'dining area' : 2,
                            'combo' : 3,
                            'other' : 4,
                            'formal' : 5}
```

(b) fuel-type

```
fuel_type_scaler = { 'oil' : 1,
                     'none' : 2,
                     'gas' : 3,
                     'other' : 4,
                     'Other' : 4,
                     'electric' : 5}
```

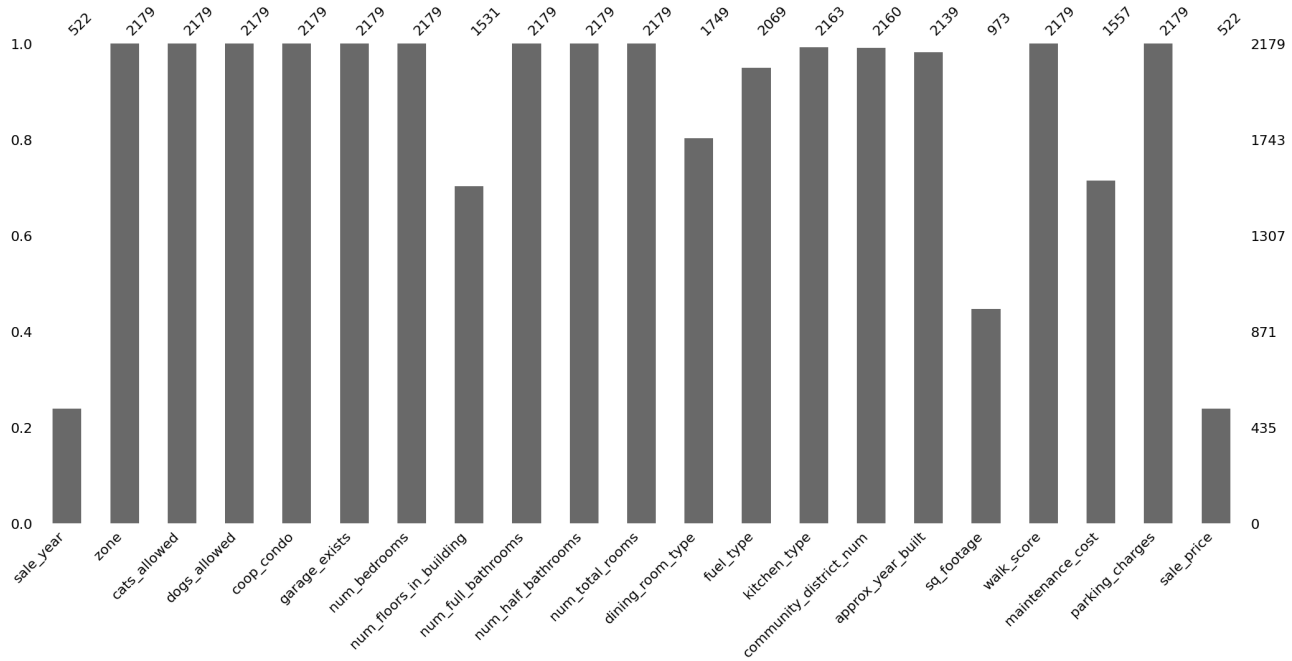
(c) kitchen-type

```
kitchen_type_scaler = { 'none' : 1,
                        'efficiency' : 2,
                        'other' : 3,
                        'eatin' : 4,
                        'combo' : 5,
                       }
```

5. Additional cost and metric-based property descriptions
   (a) sq-footage
   (b) walk-score: walkability of the area around apartment
   (c) maintenance-cost
   (d) parking-charges

3.2 Errors and Missingness

A lot of errors come from data collection. Not all entries are uniformed in type or format. For example, the feature 'dogs allowed' had 3 unique values: yes, yes98, no. Entries like this are problematic which caused major errors and had to be converted. Once all the features were standardized with data types like float the dataset could be imputed.



The bar graph illustrates the amount of missing data per column. To optimize the model, data imputation was necessary, especially for the target variable, *sale price*, which had significant missing values. The **MissForest** algorithm from scikit-learn was used to address this issue. This method leverages random forests to iteratively treat each feature as a target, using other variables as predictors to fill in the missing values.

4 Modeling

The data's intricacies are captured differently by various machine learning models. The best models are selected based on their ability to minimize error and provide the most accurate approximations.

4.1 Train + Validation, Test Split

Dataset **D** is divided into three parts: the training set for building the model, the validation set for fine-tuning, and the test set for estimating the model's performance on unseen data.

4.2 Linear Modelling

Linear regression is a statistical method that models the relationship between a dependent variable and one or more independent variables. In predicting house prices, linear regression estimates the coefficients for each feature (e.g., square footage, number of bedrooms) to create a linear equation. This equation is used to predict the house price by inputting the values of the features, providing an estimated price based on the linear relationship between the features and the target variable.

Linear Regression Results:

Mean Squared Error on Validation Set: 4348737541.035043

Root mean Squared Units (UNIT OF TARGET): 65944.95842014796

R^2 **Score on Validation Set:** 0.8515349932032527

A simple linear regression achieved an RMSE of 65,944.96, indicating that, on average, the model's predictions are off by $65,945. This is relatively good for a basic model. Additionally, the model demonstrated a strong understanding of the data's intricacies by obtaining an R^2 score of 85

4.3 Linear Regression using only top K features

Improving the linear regression model can be achieved through feature engineering. Despite data cleaning and reduction, some features may still lack predictive value. This was addressed using scikit-learn's SelectKBest to identify the most relevant features.

The model performed best with 10 features, yielding optimal in-sample results.

Mean Squared Error on Validation Set: 4637488881.221946

Root mean Squared Units (UNIT OF TARGET): 68099.11072269554

R^2 **Score on Validation Set:** 0.8416770817337977

The model with reduced features performed worse then the vanilla linear regression model. A conclusion could be made that all 21 features selected have good predictive power and reducing them would decrease the model's predictive performance.

4.4 Regression Tree Modelling

Regression tree modeling is a predictive modeling technique used for continuous target variables. It involves partitioning the data into subsets based on the values of the input features. This partitioning is performed by creating a tree structure where each node represents a decision based on a feature, and each leaf represents a predicted value. The primary objective is to minimize the variance within each subset, allowing the model to capture complex, non-linear relationships between features and the target variable.

4.4.1 Methodology

To optimize the performance of the regression tree model, a grid search with cross-validation was employed. The following steps outline the approach:

1. **Parameter Grid Definition:** A comprehensive grid of hyperparameters was defined to explore various configurations for the decision tree model. The hyperparameters included max-depth (ranging from no limit to 50), min-samples-split (ranging from 2 to 20), and min-samples-leaf (ranging from 1 to 16).
2. **Model Initialization:** An instance of the DecisionTreeRegressor was created with a fixed random state to ensure reproducibility of results.
3. **Grid Search with Cross-Validation**: The GridSearchCV method was utilized to systematically explore the parameter grid. This method performed 5-fold cross-validation and evaluated each combination of parameters based on the negative mean squared error (MSE).
4. **Model Training**: The grid search was conducted on the training data to identify the best performing hyperparameters.

5. **Best Model Selection:** The best model, identified by the grid search, was extracted for further evaluation.

4.4.2 Results

The best hyperparameters are the following:
max-depth: 10
Limits the tree to 10 levels, preventing overfitting and ensuring better generalization.

min-samples-leaf: 4
Requires at least 4 samples per leaf node, reducing over fitting and enhancing prediction robustness.

min-samples-split: 10
Requires at least 10 samples to split a node, preventing splits based on insufficient data and improving model stability.

These hyperparameters balance the model's complexity and generalization, leading to reliable and accurate predictions.

The training data was then trained on the model with the same specifications obtained the following results:

Best Hyperparameters: 'max-depth': 10, 'min-samples-leaf': 4, 'min-samples-split': 10

Mean Squared Error on Validation Set: 1683081073.2161603

Root mean Squared Units (UNIT OF TARGET): 41025.37109175443

R^2 **Score on Validation Set:** 0.9425399577195361

The regression tree performed a lot better than the regression model. It obtained an R^2 of 93% and the predictions are only off by $ 41,025.

4.5 Random Forrest Modelling

A Random Forest is an ensemble learning method that combines multiple decision trees to improve accuracy and robustness. Each tree in the forest is trained on a random subset of the training data and uses a random subset of features for splitting at each node. This randomness ensures that the trees are more independent of each other, reducing over fitting and improving generalization. By averaging the predictions of these diverse trees, a Random Forest produces a more accurate and stable prediction compared to individual decision trees.

4.5.1 Methodology

1. **Define the Parameter Grid**:
   - `n_estimators`: Number of trees in the forest. Values considered: [100, 200, 300].
   - `max_depth`: Maximum depth of the tree. Values considered: [None, 10, 20, 30].
   - `min_samples_split`: Minimum number of samples required to split an internal node. Values considered: [2, 10, 20].
   - `min_samples_leaf`: Minimum number of samples required to be at a leaf node. Values considered: [1, 2, 4, 8].
2. **Initialize the Random Forest Regressor**:
   - Create an instance of `RandomForestRegressor` with a fixed random state for reproducibility (`random_state=42`).
3. **Initialize GridSearchCV**:

- Set up `GridSearchCV` with the following parameters:
  - `estimator`: The Random Forest Regressor instance.
  - `param_grid`: The defined parameter grid.
  - `scoring`: Use negative mean squared error (`neg_mean_squared_error`) as the scoring metric.
  - `cv`: Use 5-fold cross-validation (`cv=5`).
  - `n_jobs`: Utilize all available processors for parallel computation (`n_jobs=-1`).

4. **Fit the Model**:
   - Train the model on the training data (`X_train`, `y_train`) using the grid search to find the best hyperparameters.
5. **Get the Best Model**:
   - Extract the best estimator (`bestrf`) identified by the grid search.
6. **Predict on the Validation Set**:
   - Use the best model to make predictions on the validation data (`X_val`).
7. **Evaluate the Model**:
   - Calculate the Mean Squared Error (MSE) between the actual and predicted values.
   - Compute the Root Mean Squared Error (RMSE) for interpretability.
   - Determine the R^2 score to assess the proportion of variance explained by the model.

4.5.2 Results

Best Hyperparameters:

- `max_depth`: None
- `min_samples_leaf`: 1
- `min_samples_split`: 2
- `n_estimators`: 300

Mean Squared Error on Validation Set: 962788314.0700483
Root Mean Squared Error (RMSE): 31028.830369030158
R^2 Score on Validation Set: 0.9671306046310126

Again there was an increase in performance. The random Forrest regressor fit the data better than the decision tree. It obtained an R^2 of 97% and a the lowest prediction error only at $31,028 which is quite accurate.

4.6 Bagged Regression Trees

Bagged decision trees, also known as Bootstrap Aggregating, is an ensemble learning method aimed at improving the stability and accuracy of machine learning algorithms. The process involves creating multiple subsets of the original dataset using random sampling with replacement. For each subset, a decision tree is trained independently. The final model's prediction is obtained by averaging the predictions (for regression) or voting (for classification) from all the individual trees. This approach reduces variance and helps prevent overfitting, leading to more robust and reliable models.

4.6.1 Methodology

1. **Define the Parameter Grid**:
   - `base_estimator__max_depth`: Maximum depth of the base estimator. Values considered: [None, 10, 20, 30, 40, 50].
   - `base_estimator__min_samples_split`: Minimum number of samples required to split an internal node. Values considered: [2, 10, 20].
   - `base_estimator__min_samples_leaf`: Minimum number of samples required to be at a leaf node. Values considered: [1, 2, 4, 8, 16].
   - `n_estimators`: Number of trees in the ensemble. Values considered: [10, 50, 100].
2. **Initialize the Bagging Regressor**:
   - Use a Decision Tree Regressor as the base estimator (`base_dt`) with a fixed random state (`random_state=42`).
   - Create an instance of `BaggingRegressor` using the base estimator and a fixed random state (`random_state=42`).
3. **Initialize GridSearchCV**:
   - Set up `GridSearchCV` with the following parameters:
     - `estimator`: The Bagging Regressor instance.
     - `param_grid`: The defined parameter grid.
     - `scoring`: Use negative mean squared error (`neg_mean_squared_error`) as the scoring metric.
     - `cv`: Use 5-fold cross-validation (`cv=5`).
     - `n_jobs`: Utilize all available processors for parallel computation (`n_jobs=-1`).
     - `return_train_score`: Return training scores for evaluation.
4. **Fit the Model**:
   - Train the model on the training data (`X_train`, `y_train`) using the grid search to find the best hyperparameters.
5. **Extract Results**:
   - Retrieve the cross-validation results (`bagged_dt_results`) from the grid search.
6. **Calculate RMSE and MSE**:
   - Compute the Root Mean Squared Error (RMSE) for both training and validation sets:
     - Training RMSE: `np.sqrt(-bagged_dt_results['mean_train_score'])`
     - Validation RMSE: `np.sqrt(-bagged_dt_results['mean_test_score'])`
   - Compute the Mean Squared Error (MSE) for both training and validation sets:
     - Training MSE: `-bagged_dt_results['mean_train_score']`
     - Validation MSE: `-bagged_dt_results['mean_test_score']`

4.6.2 Results

Best Hyperparameters:

- `base_estimator__max_depth`: None
- `base_estimator__min_samples_leaf`: 1
- `base_estimator__min_samples_split`: 2
- `n_estimators`: 100

Mean Squared Error on Validation Set: 961789751.960797
Root Mean Squared Error (RMSE): 31012.735318910472
R^2 Score on Validation Set: 0.9671646953364043

Despite not showing significant improvements over the Random Forest, this model represents our best performance. Given the current model and available data, it can be concluded that we have likely reached the peak performance, as there were no major leaps in improvement.

4.7 Gradient Boosted Decision Trees

Gradient Boosted Decision Trees (GBDT) is an advanced ensemble learning technique that builds models sequentially. Each new tree corrects errors made by the previous trees, focusing on the instances where the model performs poorly. This is done by minimizing a loss function, typically through gradient descent. Starting with a simple model, each subsequent tree is added to reduce the residual errors of the combined ensemble. This iterative process continues until the model achieves optimal performance. GBDT is powerful for both regression and classification tasks, offering high accuracy and the ability to capture complex patterns in data.

In this study, training gradient boosted decision trees with grid search for optimal hyperparameters resulted in a 40-minute training time, which had to be interrupted. The actual required training time is unknown. Due to limited computational power, the focus was shifted to finding the best predictive model within these hardware constraints, preventing further testing with gradient boosted decision trees.

5 Model Selection, Model Evaluation, and Results

Model selection is a crucial step in building a robust predictive model. It involves choosing the best model among various candidates to achieve optimal performance. The problem it addresses is overfitting and underfitting. Overfitting occurs when a model is too complex and captures noise in the training data, while underfitting happens when a model is too simple to capture the underlying patterns. Model selection helps to balance bias and variance, leading to better generalization on unseen data.

To ensure the selected model performs well, we evaluate it using specific metrics. For this project, we used Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and the R^2 score:

- **MSE**: Measures the average squared difference between the actual and predicted values. It penalizes larger errors more significantly.
- **RMSE**: Provides an interpretable metric by taking the square root of MSE, giving the error in the same units as the target variable.
- **R^2 score**: Indicates the proportion of variance in the dependent variable that is predictable from the independent variables. A higher R^2 score means a better fit.

Despite not showing significant improvements over the Random Forest, our Bagging Regressor model represents our best performance. Given the current model and available data, it can be concluded that we have likely reached peak performance, as there were no major leaps in improvement.

5.1 Methodology

1. **Define a Function to Evaluate and Print Model Performance**
    - The function `evaluate_model` takes the model name, the model itself, the test features (`X_test`), and the test targets (`y_test`) as inputs.
    - It predicts the target values using the model and computes the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 score.
    - It prints the performance metrics and returns them.
2. **Initialize a List to Hold Results**

- Create an empty list called `results` to store the evaluation results of each model.

3. **Predict and Evaluate Each Model on the Test Set**
   - For each model (Simple Linear Model, Best Decision Tree, Best Random Forest, and Best Bagged Decision Tree):
     - Call the `evaluate_model` function to evaluate the model's performance on the test set.
     - Append the model name and its performance metrics (MSE, RMSE, and R^2) to the `results` list.

4. **Output All Results**
   - Print a summary of the evaluation results for all models, showing the MSE, RMSE, and R^2 score for each model.

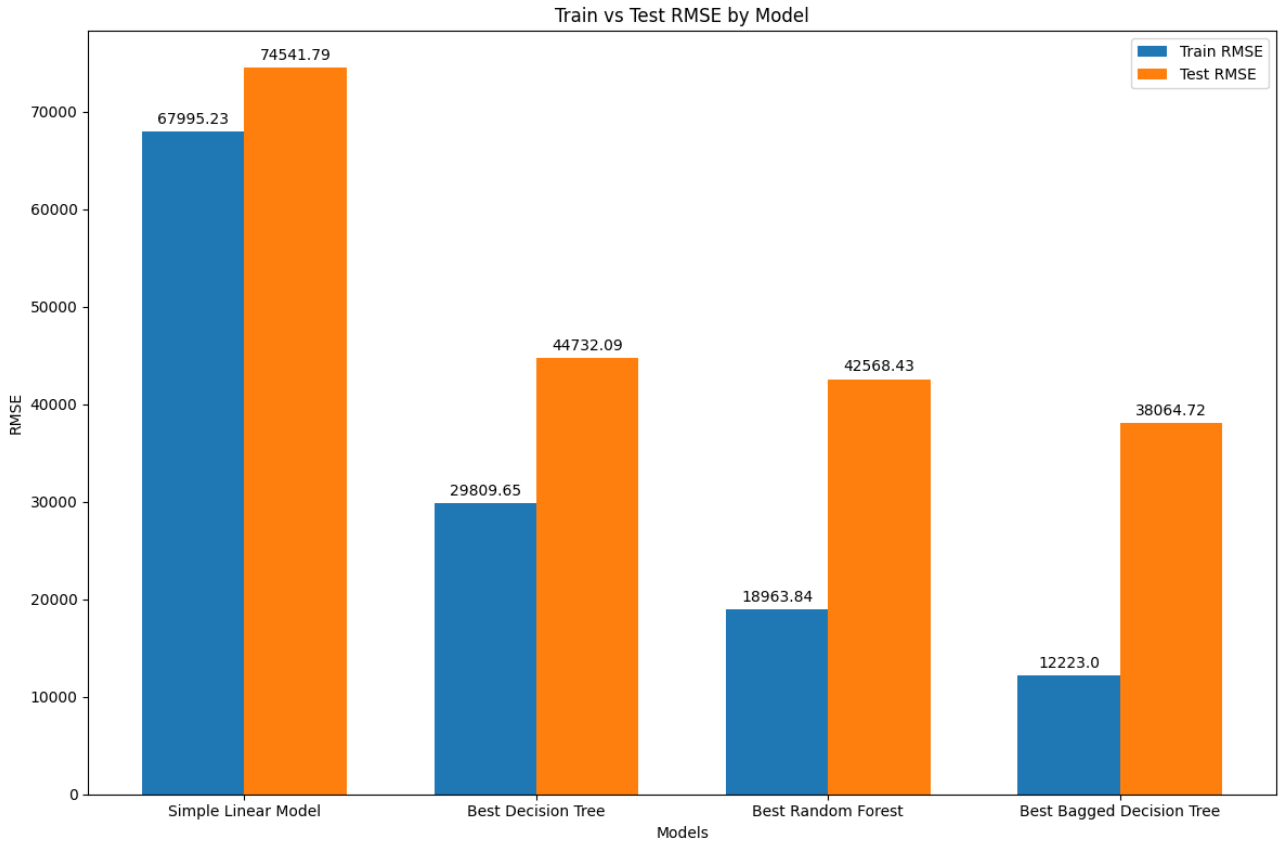
5. **Determine the Best Model Based on RMSE**
   - Identify the model with the lowest RMSE from the `results` list.
   - Print the name and RMSE of the best model.

6. **Train the Best Model on the Entire Dataset**
   - Based on the best model's name, assign the corresponding model instance to `best_model`.
   - Fit the `best_model` on the entire dataset, which includes both training and validation data (`X_train`, `X_val`, `y_train`, `y_val`).

7. **Print Final Model Details**
   - Print the name of the final model trained on the entire dataset.

5.2 Results

Model Evaluation on Test Set



- Simple Linear Model:
  - MSE: 5556479114.205616
  - RMSE: 74541.7944123001
  - R^2: 0.7996912758197945
- Best Decision Tree:
  - MSE: 2000960176.1023357
  - RMSE: 44732.09335703322
  - R^2: 0.9278662311560294
- Best Random Forest:
  - MSE: 1812071612.0233126
  - RMSE: 42568.43445586545
  - R^2: 0.9346755840763291
- Best Bagged Decision Tree:
  - MSE: 1795898789.6211407
  - RMSE: 42378.04608073785
  - R^2: 0.935258607490111

Best Model: Best Bagged Decision Tree with RMSE = 42378.04608073785

6 Final Model

In the final stage of our project, we trained a model using the entire dataset. This model leveraged the best hyperparameters identified earlier and employed the Bagging Algorithm to enhance performance and robustness.

We validated this final model using the same metrics as before. The results are as follows:

- **Best Bagged Decision Tree**:
  - Train RMSE: 12,222.997
  - Test RMSE: 38,064.717

The in-sample RMSE decreased, which is expected as the model had more data to learn from. However, the test RMSE increased, indicating a good balance. This increase in test RMSE suggests that the model did not overfit, as it is common to perform worse on unseen data. This outcome reinforces the model's generalization capability and robustness.

7 Deployment

With the best model now obtained, the next step is to make it accessible for practical use. The model could be implemented on housing applications or webpage finders such as StreetEasy or ApartmentList.com. To achieve this, the model must be exported.

A sample implementation involves creating a simple Python function. This function takes the specifics of an apartment as input and returns the predicted price using the trained model. This approach ensures that the model can be easily integrated into various platforms, providing valuable insights to users.

7.1 Methodology

```
# Example usage with the provided sample values
sale_year = 2017  # (2016, 2017)
zone = 2  # (1 - 9)
cats_allowed = 1  # (0,1)
dogs_allowed = 1  # (0,1)
coop_condo = 2  # (1:Coop, 2:Condo)
garage_exists = 0  # (0,1)
num_bedrooms = 1  # (1,..,n)
num_floors_in_building = 5  # (1,..,n)
num_full_bathrooms = 1  # (1,..,n)
num_half_bathrooms = 0  # (1,..,n)
num_total_rooms = 1  # (1,...,n)
dining_room_type = 1  # (1:none, 2:dining area, 3:combo, 4:other, 5:formal)
fuel_type = 5  # (1:oil, 2:none, 3:gas, 4:other, 5:electric)
kitchen_type = 0  # (1: none, 2:efficiency, 3:other, 4:eatin, 5:combo)
community_district_num = 26
approx_year_built = 1950  # (19xx - 20xx)
sq_footage = 1200
walk_score = 99
maintenance_cost = 500
parking_charges = 0

# Assuming best_model is already defined and trained
predicted_price, feature_description = predict_price(
    sale_year, zone, cats_allowed, dogs_allowed, coop_condo, garage_exists,
    num_bedrooms, num_floors_in_building, num_full_bathrooms, num_half_bathrooms,
    num_total_rooms, dining_room_type, fuel_type, kitchen_type, community_district_num,
    approx_year_built, sq_footage, walk_score, maintenance_cost, parking_charges
)
formatted_price = f"${predicted_price:,.0f}"
print(f"Predicted Price: {formatted_price}")
print(f"Feature Description: {feature_description}")
```

7.2 Results

In this implementation, we input the apartment's details and receive a predicted price. For example, an apartment sold in 2017, located in zone 2, allowing cats and dogs, classified as a condo, with no garage, having one bedroom, five floors in the building, one full bathroom, no half bathrooms, one total room, a dining room type of 1, using electric fuel, no kitchen, located in community district 26, built approximately in 1950, with a square footage of 1200, a walk score of 99, a maintenance cost of 500, and no parking charges, has a predicted price of $481,731.

8 Discussion

Throughout this project, several areas for improvement and future extensions have been identified. One significant aspect is the maintenance of zoning as a leveled factor, which would enhance the model's ability to account for variations within different zones. Additionally, the use of district data requires further exploration; with only a few unique values, it may not serve as an effective indicator and alternative variables should be considered.

Data cleaning emerged as a particularly crucial and challenging aspect of this project. Unlike the more intuitive and quicker process of modeling, data cleaning requires meticulous attention to detail and critical thinking. It's a tedious yet essential step to ensure the reliability of the input data, and further refinement in these techniques could substantially improve overall model performance.

There is also room for improvement in coding standards and code neatness. The complexity of the codebase increased as the project progressed, making it difficult to maintain and debug. Better coding practices, such as clear documentation, modular coding, and consistent naming conventions, would streamline future development efforts and mitigate these issues.

Although the current model shows promise, it is not yet production-ready. Effective deployment requires more comprehensive and up-to-date data, as the real estate market can change rapidly. Including such data would likely enhance the model's performance and robustness.

Future extensions could focus on incorporating additional features, such as economic indicators or detailed demographic information, to improve predictive power. Exploring more sophisticated modeling techniques and ensemble methods could also boost accuracy.

One of the primary motivations behind this project is the potential to outperform Zillow by leveraging localized data specific to Queens. While the model shows strong potential in this context, achieving this goal will require continuous improvements in data quality, model tuning, and validation against real-world outcomes.

In conclusion, while there are several areas for improvement, this project lays a solid foundation for future work. By addressing the identified gaps and extending the model's capabilities, it is feasible to develop a robust and competitive predictive model for the real estate market in Queens.

9 Acknowledgements

Thank you, Prof. Adam Kapelner, for an amazing semester. I learned more in this class than I ever thought I would. Working on this project was particularly enjoyable, especially the data cleaning aspect. I'm glad to have had this experience, as it helped me realize that I enjoy this kind of work, which inspires me for the future. Thank you for your guidance and passion for the field.

A Code Appendix

All the code could be found here: Project Repository