MILESTONE 2 - REPORT.DOCX

House Price Prediction

Table of Contents

Capstone Project: Business Report: Milestone #2

Milestone 2 Report – House Price Prediction

I. Context

Housing prices are an important reflection of the economy, and housing price ranges are of great interest for both buyers and sellers. Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. A house value is simply more than location and square footage. Like the features that make up a person, an educated party would want to know all aspects that give a house its value. For example, you want to sell a house, and you don't know the price which you may expect—it can't be too low or too high. To find house price you usually try to find similar properties in your neighbourhood and based on gathered data you will try to assess your house price.

II. Objective

The objective of this problem statement is to predict the housing prices of a town or a suburb based on the features of the locality provided and identify the most important features to consider while predicting the prices.

III. Data Dictionary

This dataset has 23 features, price being the target variable. The details of all the features are given below:

- cid: a notation for a house
- dayhours: Date house was sold
- price: Price is prediction target (in $)
- room_bed: Number of Bedrooms per house
- room_bath: Number of bathrooms per bedrooms
- living_measure: square footage of the home
- lot_measure: square footage of the lot
- ceil: Total floors (levels) in house
- coast: House which has a view to a waterfront (0 - No, 1 - Yes)
- sight: Has been viewed
- condition: How good the condition is (Overall out of 5)
- quality: grade given to the housing unit, based on grading system
- ceil_measure: square footage of house apart from basement
- basement_measure: square footage of the basement
- yr_built: Built Year
- yr_renovated: Year when house was renovated
- zipcode: zip code
- lat: Latitude coordinate
- long: Longitude coordinate
- living_measure15: Living room area in 2015 (implies-- some renovations) This might or might

- not have affected the lot size area
- lot_measure15: lotSize area in 2015 (implies-- some renovations)
- furnished: Based on the quality of room (0 - No, 1 - Yes)
- total_area: Measure of both living and lot

Comprehensive House Price Prediction Analysis

To: Executive Leadership (CMO, COO, Data Science Manager)

From: Data Science Team

Date: 14/3/2025

Subject: Comprehensive Analysis of House Price Prediction Dataset and Strategic Implications

I. Context & Objective

Housing prices serve as a key indicator of economic stability and market trends. Buyers and sellers alike are keen to understand the factors influencing property values beyond just location and size. A home's worth is determined by various features, including its condition, quality, amenities, and proximity to desirable locations. This study aims to predict house prices based on multiple features while identifying the most influential factors affecting property values.

The primary goal of the business analysis in this business report is to predict housing prices in a given town or suburb based on available property features. Additionally, this analysis seeks to determine the most significant attributes that influence property valuation.

II. Data Overview

The dataset comprises 23 features, with price as the target variable. Key attributes include:

- **Structural details**: Number of bedrooms (room_bed), bathrooms (room_bath), floors (ceil), total area (total_area), living area (living_measure), and basement size (basement_measure).
- **Property characteristics**: Year built (yr_built), year renovated (yr_renovated), condition rating (condition), and quality rating (quality).
- **Geographical attributes**: Zip code (zipcode), latitude (lat), and longitude (long).
- **Market-related factors**: Sale date (dayhours), number of views (sight), waterfront view (coast), and whether the house is furnished (furnished).

III. Milestone 1 corrections

1. Data Cleaning & Preprocessing

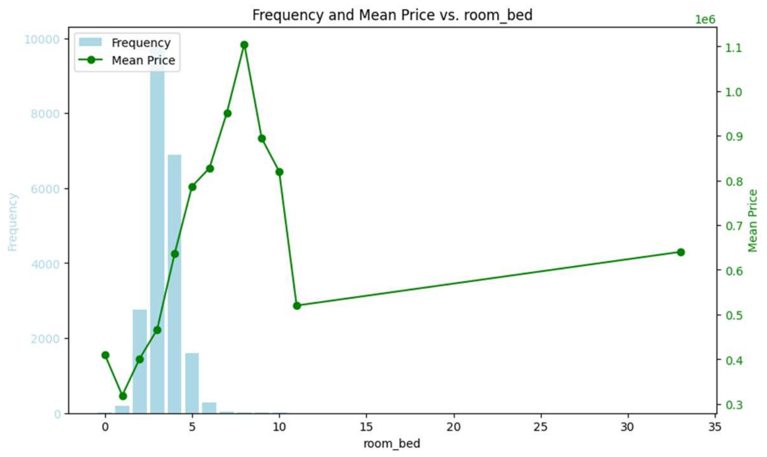
Missing Value Treatment:

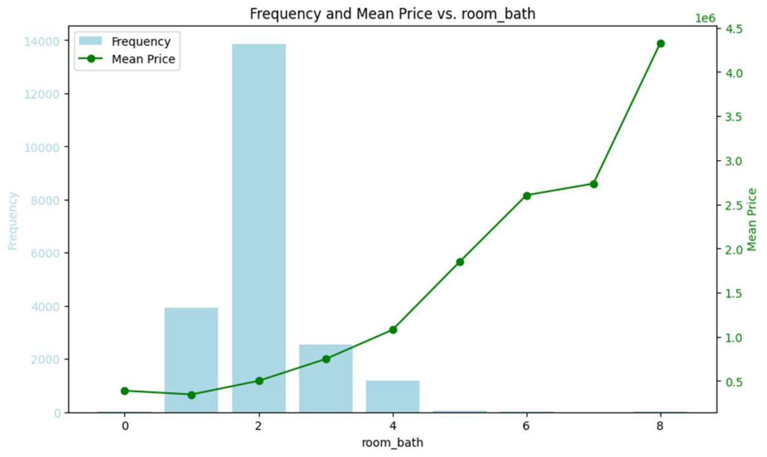
- After converting some features to numerical and coercing them, there appears some NAN values. We use then ***KNN Imputer*** to treat the columns with NANs.
- No missing values after using ***KNN Imputer***.

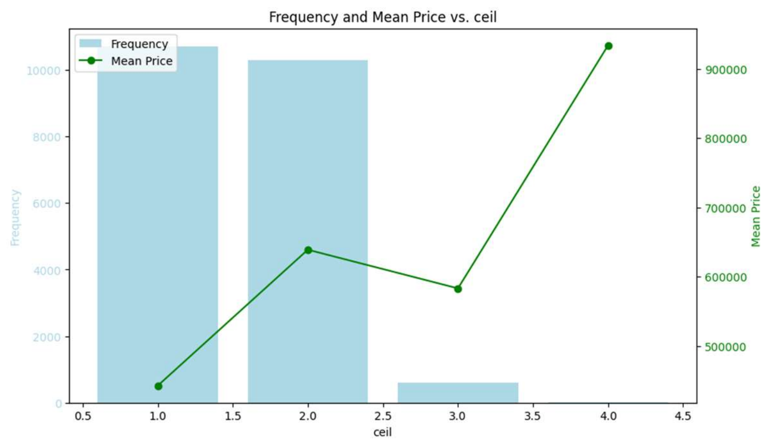
Outlier Detection & Handling:

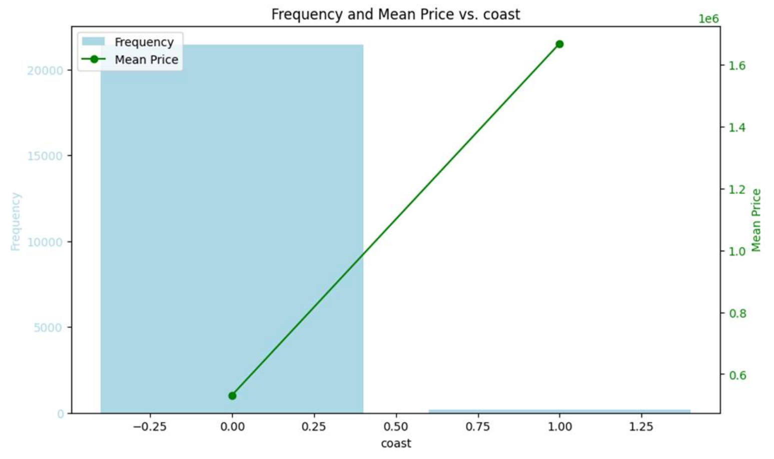
- We are not handling Outliers using Capping any longer.

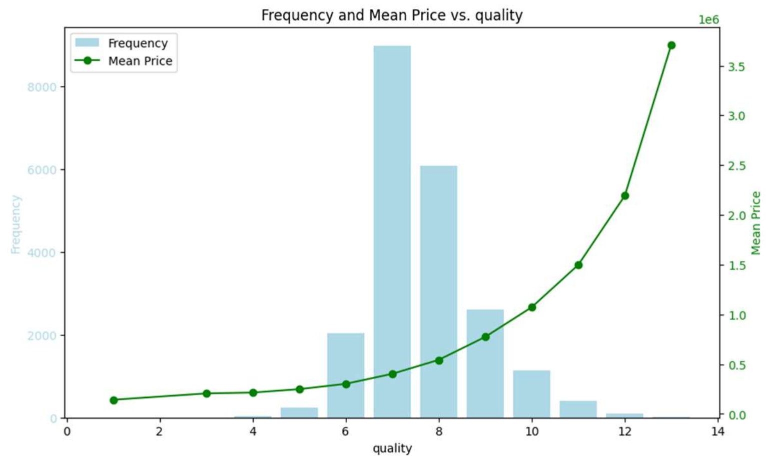
- For categorical features, we calculated the unique. Here is what we found:
    - Unique room_bed values (sorted): [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 33]
    - Unique room_bath values (sorted): [0, 1, 2, 3, 4, 5, 6, 7, 8]
    - Unique ceil values (sorted): [1, 2, 3, 4]
    - Unique coast values (sorted): [0, 1]
    - Unique quality values (sorted): [1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
    - Unique condition values (sorted): [1, 2, 3, 4, 5]
    - Unique sight values (sorted): [0, 1, 2, 3, 4]
    - Unique furnished values (sorted): [0, 1]
- Then, we calculated the frequency of each category in the specified columns and showed the corresponding mean of the price.
- We have plotted these as follows:

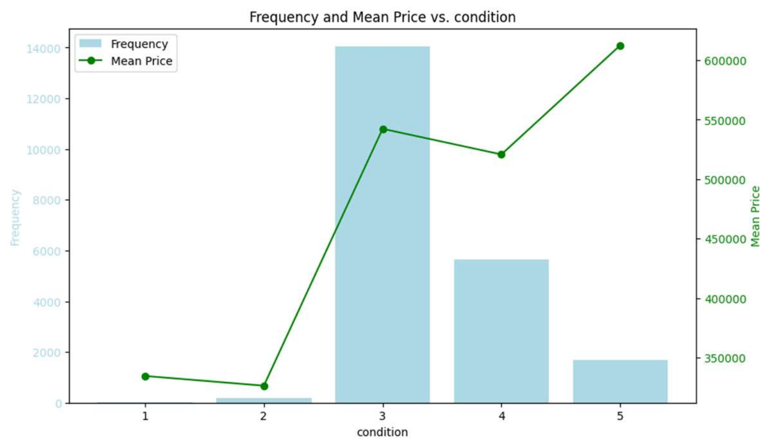


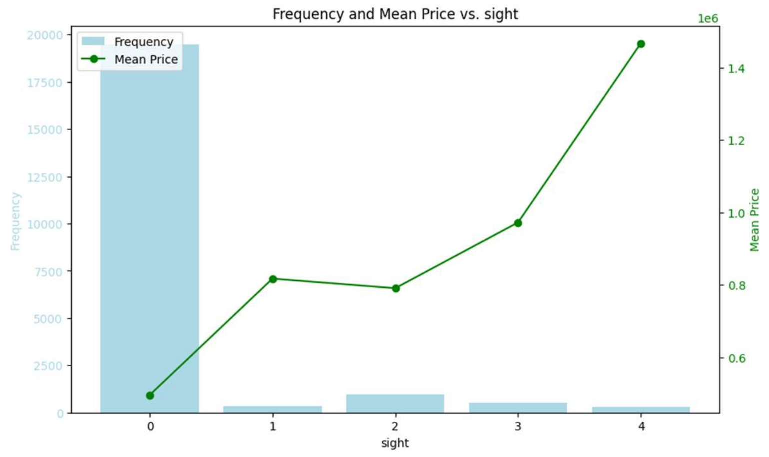


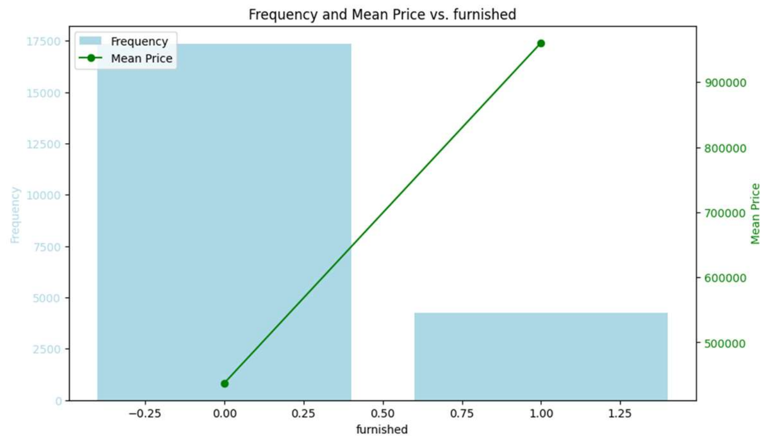

Frequency and Mean Price vs. ceil
Frequency
Mean Price
ceil
Mean Price
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
0
2000
4000
6000
8000
10000
500000
600000
700000
800000
900000


Frequency and Mean Price vs. coast
Frequency
Mean Price
coast
Mean Price
1e6
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
0
5000
10000
15000
20000
0.6
0.8
1.0
1.2
1.4
1.6

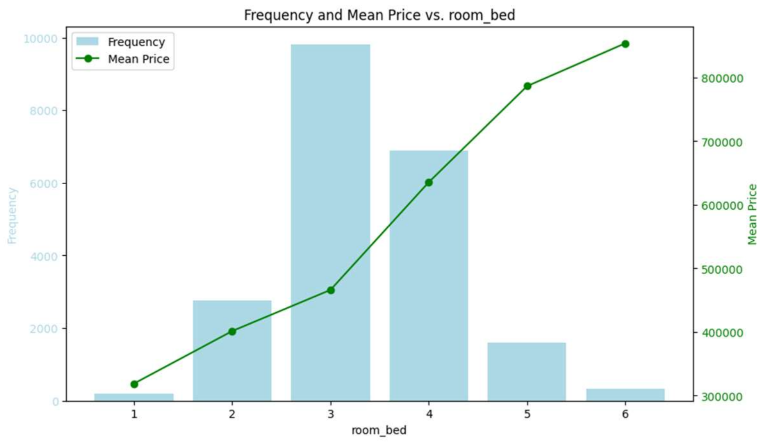

Frequency and Mean Price vs. quality
Frequency
Mean Price
quality
Mean Price
1e6
0
2
4
6
8
10
12
14
0
2000
4000
6000
8000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

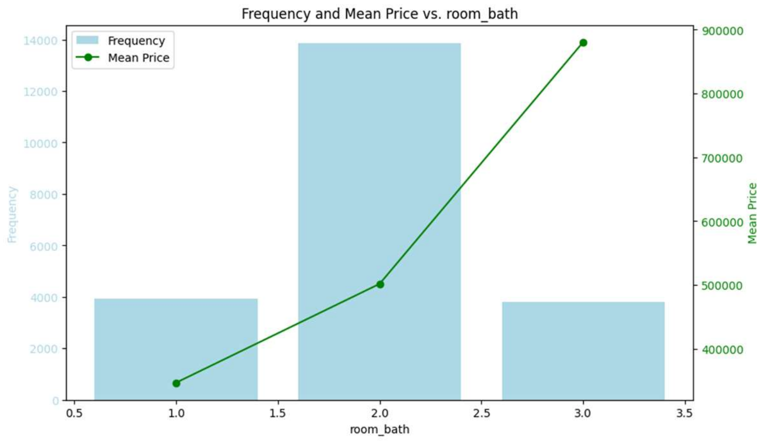


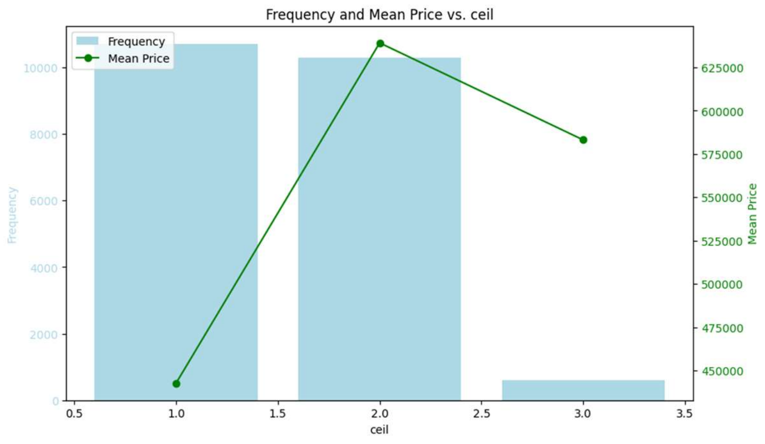


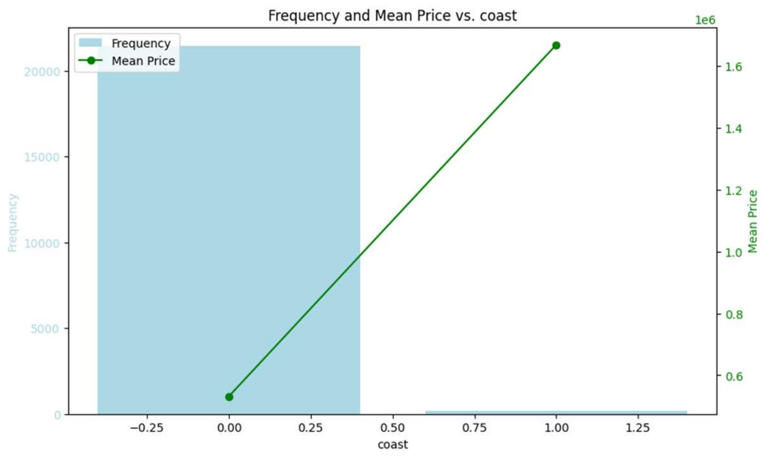


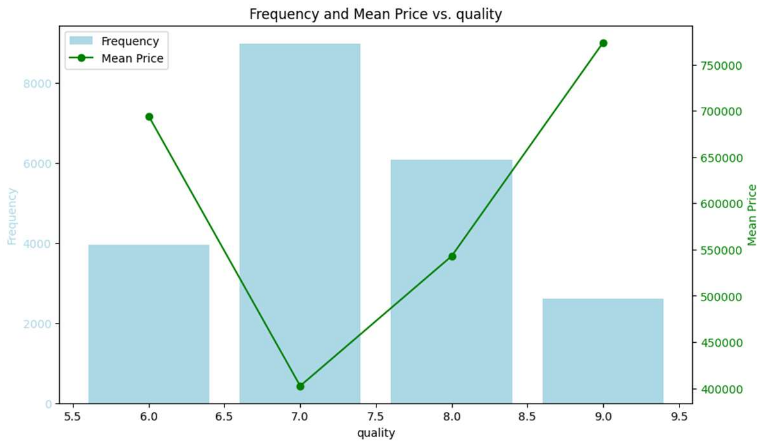
- Based on the above, I have replaced categorical feature values with low frequency with values that are closer in mean_price.
- Like this we managed to merge some feature categories and reduced their number.
- The resulting new categories are plotted as follows:

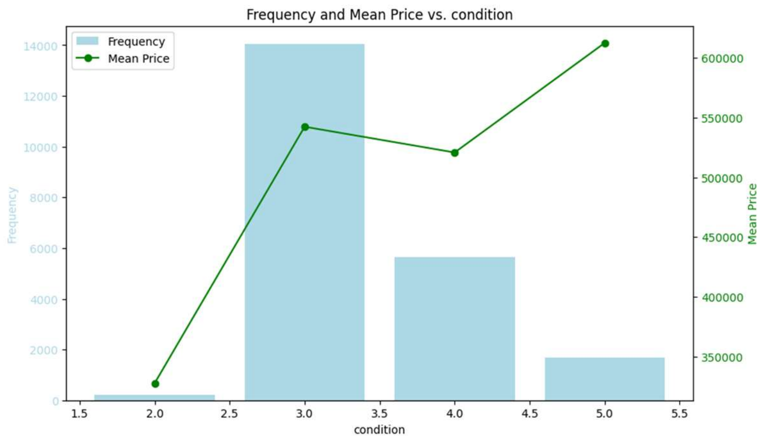

Frequency and Mean Price vs. room_bed
Frequency
Mean Price
room_bed

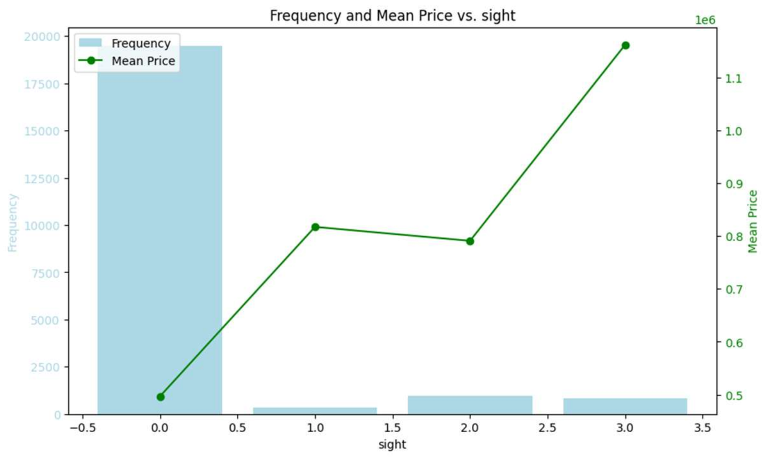

Frequency and Mean Price vs. room_bath
Frequency
Mean Price
room_bath

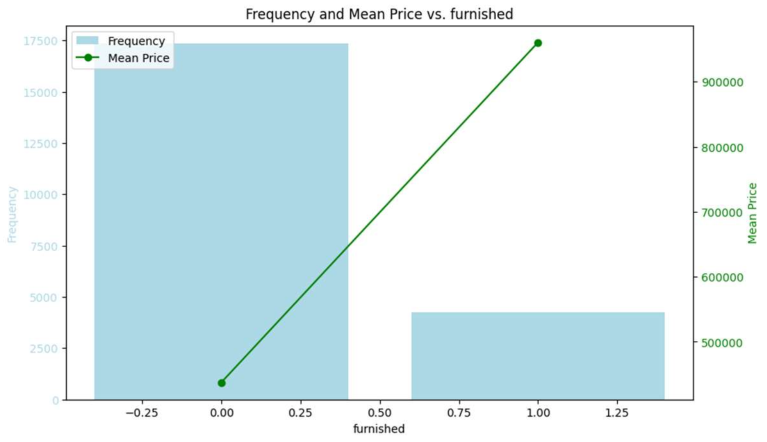

Frequency and Mean Price vs. ceil
Frequency
Mean Price
ceil


Frequency and Mean Price vs. coast
Frequency
Mean Price
1e6
coast


Frequency and Mean Price vs. quality
Frequency
Mean Price
quality


Frequency and Mean Price vs. condition
Frequency
Mean Price
condition



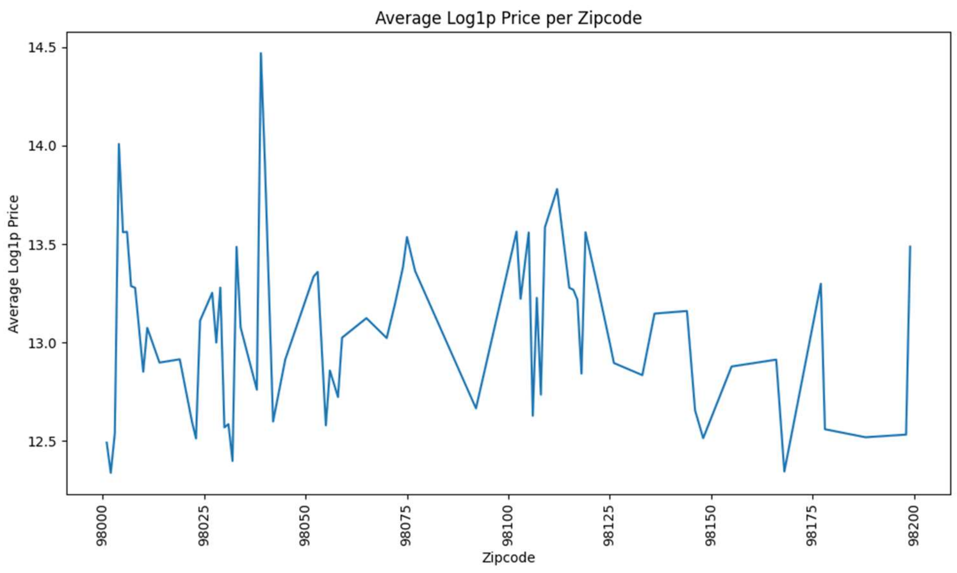


- Mitigating Outliers in Numerical Data with **Log1p Transformation**.
- Using the log1p transformation is a common technique to handle outliers in numerical features. This transformation applies the natural logarithm to the feature values after adding 1, which helps to reduce the impact of extreme values.
- This transformation can help make the data more normally distributed, which is beneficial for many machine learning algorithms.

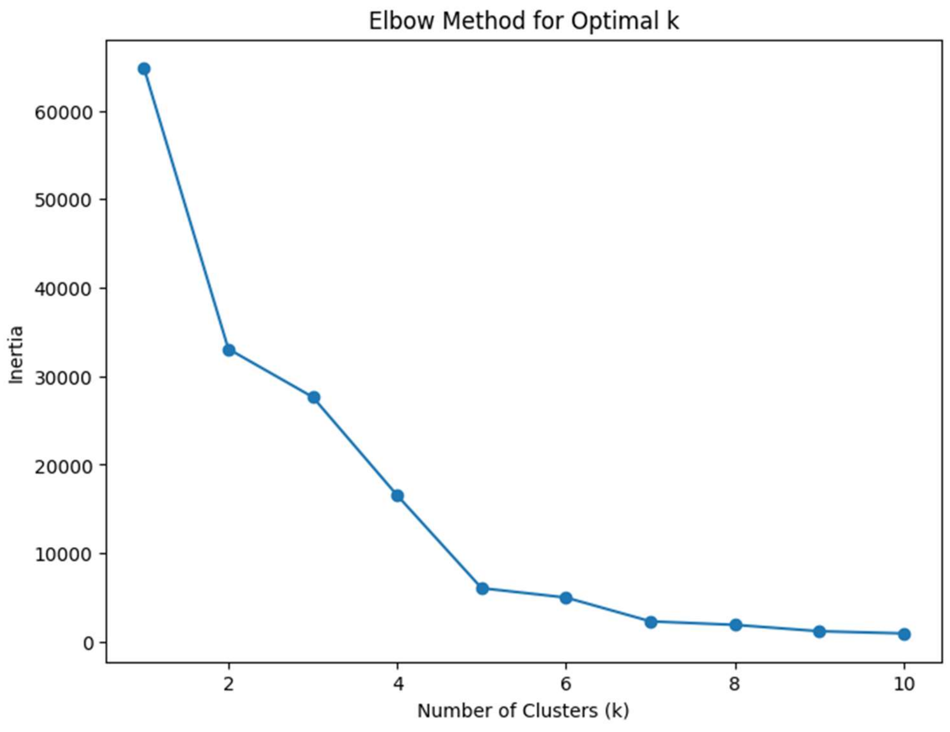
2. Exploratory Data Analysis (EDA)

Mean Price per Location (Zip Code)

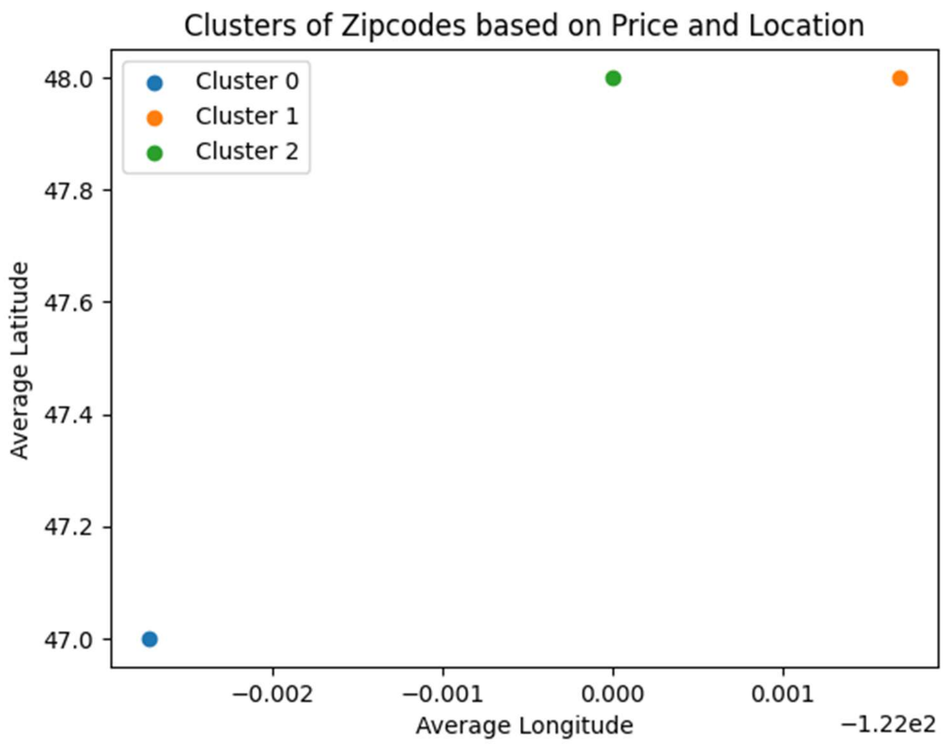
- We calculated the aggregated mean price (Log1p) per month per Zip Code location. The below plot is what we found.



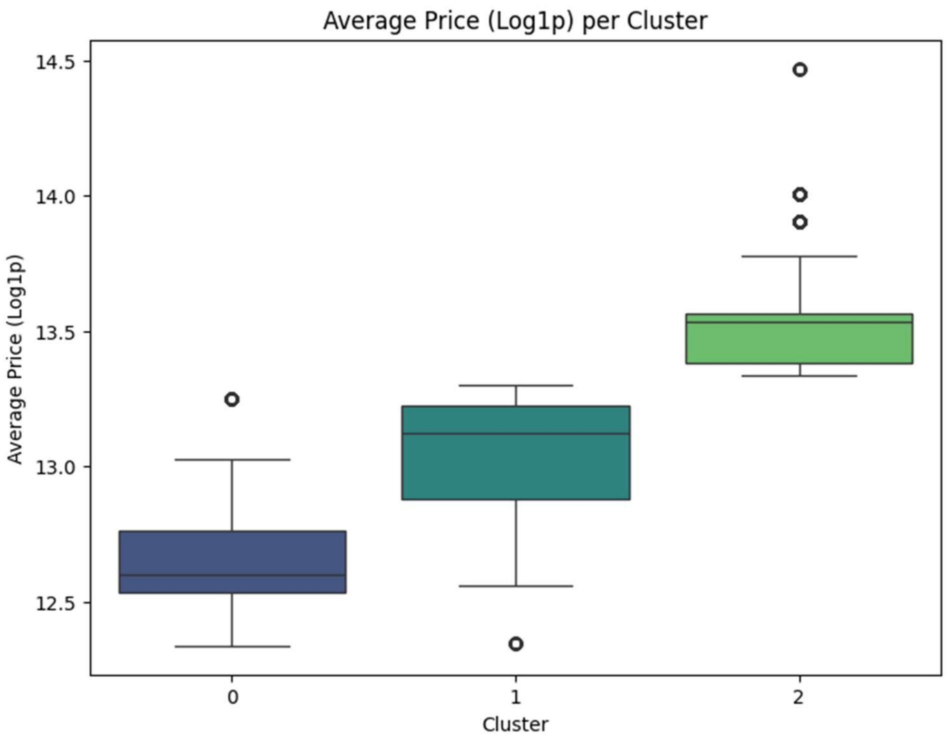
- The plot shows the relationship between zip code and the average log-transformed price.
- There are variations in average prices across different zip codes. Some zip codes show higher average prices, while others have lower average prices.
- Next step is to identify any clusters or patterns in the data. Are there groups of zip codes with similar average prices?
- We used ***KMeans Clustering***, and we plotted the Elbow method to look for the optimal number of clusters K. The ***Elbow Method*** plot is as follows:



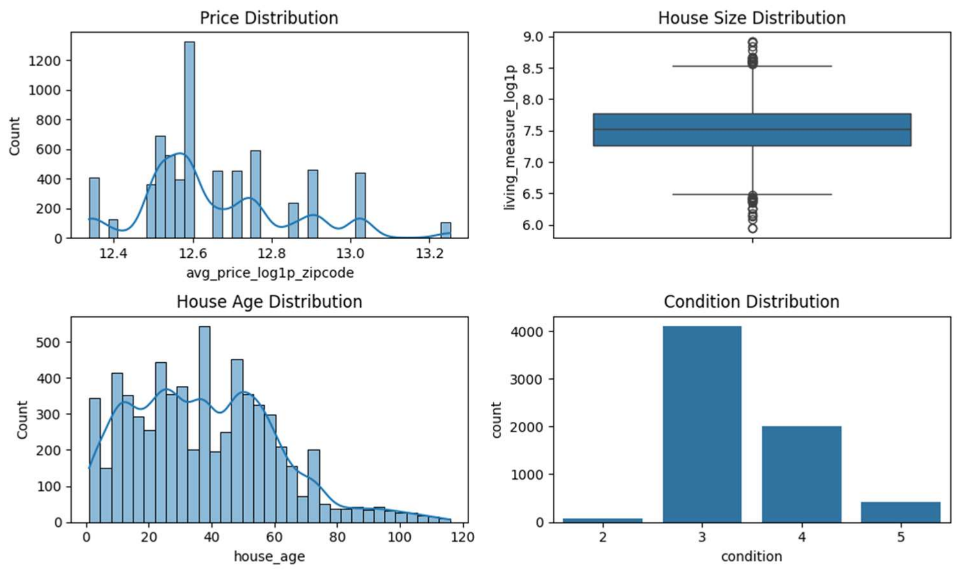
- From the above plot, the optimal K = 3.
- Below plot shows the clusters of Zip Codes based on Price and Location.



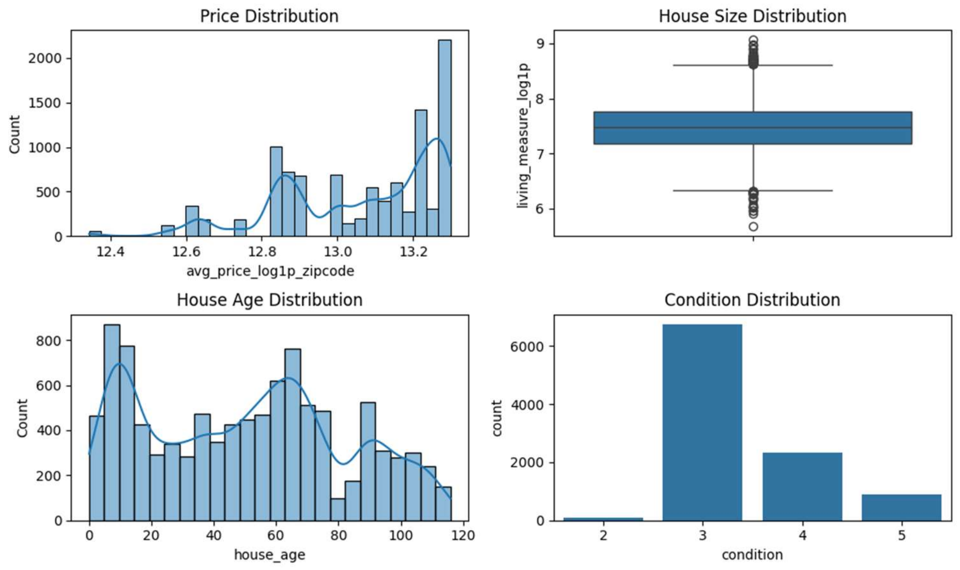
- A More straightforward representation is a boxplot of Average Price by Cluster



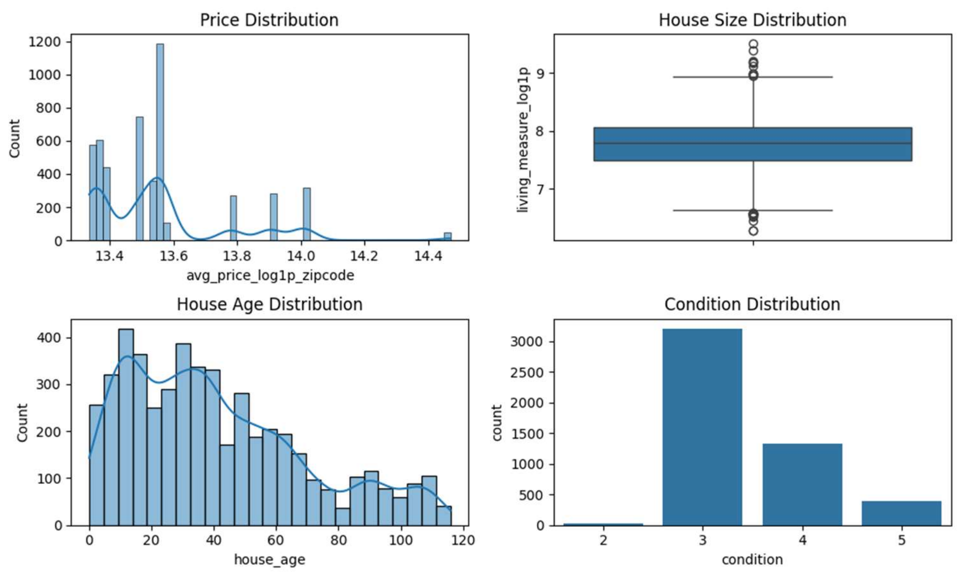
- From the above, each cluster has an average Price Distribution distinctly different.
- Analysing each Cluster characteristics; Mean Price (Log1p) per Zipcode, Living Measure (Log1p) Distribution, Condition and House Age distributions. We found the distinct characteristics plotted as below.
- **<u>Cluster #0:</u>**



- **<u>Cluster #1:</u>**

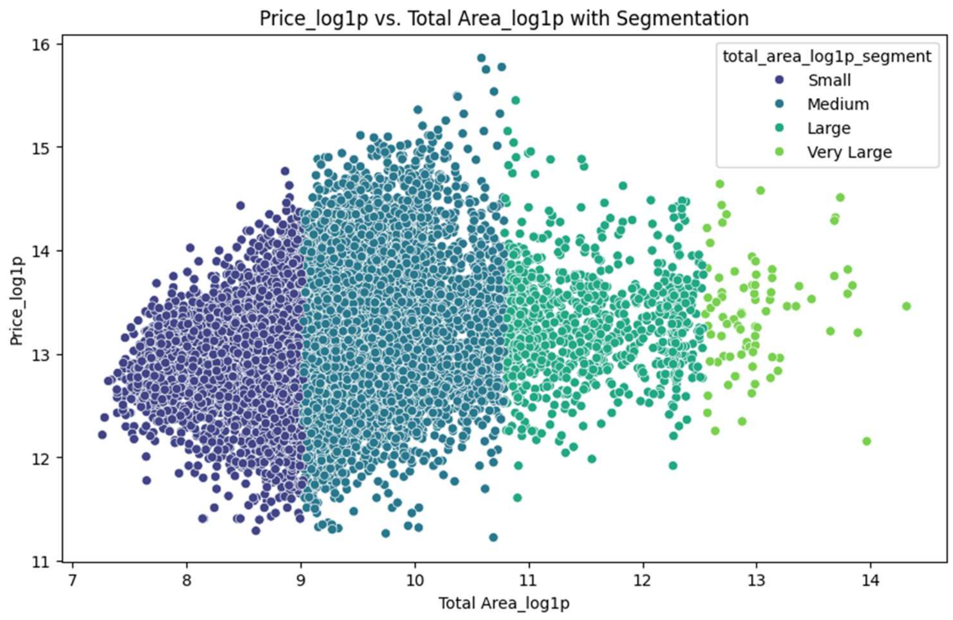


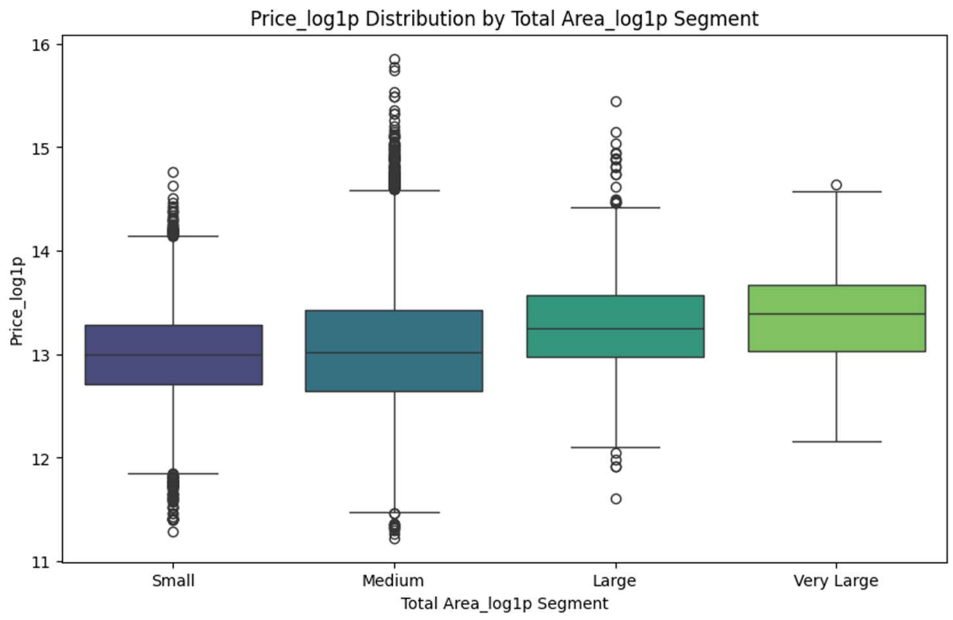
- **<u>Cluster #2:</u>**



- **Comparative Analysis:**
  - **Price**:
    - Cluster 2 has the highest median price, followed by Cluster 1, and then Cluster 0.
  - **House Size**:
    - Cluster 2 also has the largest median house size.
    - Cluster 0 and Cluster 1 have similar house sizes, with Cluster 0 being slightly larger.
  - **House Age**:
    - Cluster 1 has the oldest houses with a median age of 53 years.
    - Clusters 0 and 2 have similar house ages, around 36-37 years.
  - **Condition**:
    - All clusters have a mode condition of 3, indicating that the most common condition rating is the same across all clusters.
- These characteristics suggest that **Cluster 2** represents newer and larger houses with higher prices, while **Cluster 1** represents older houses with slightly lower prices. **Cluster 0** falls somewhere in between in terms of price and house size but has a similar house age to Cluster 2. The condition of houses is consistent across all clusters.

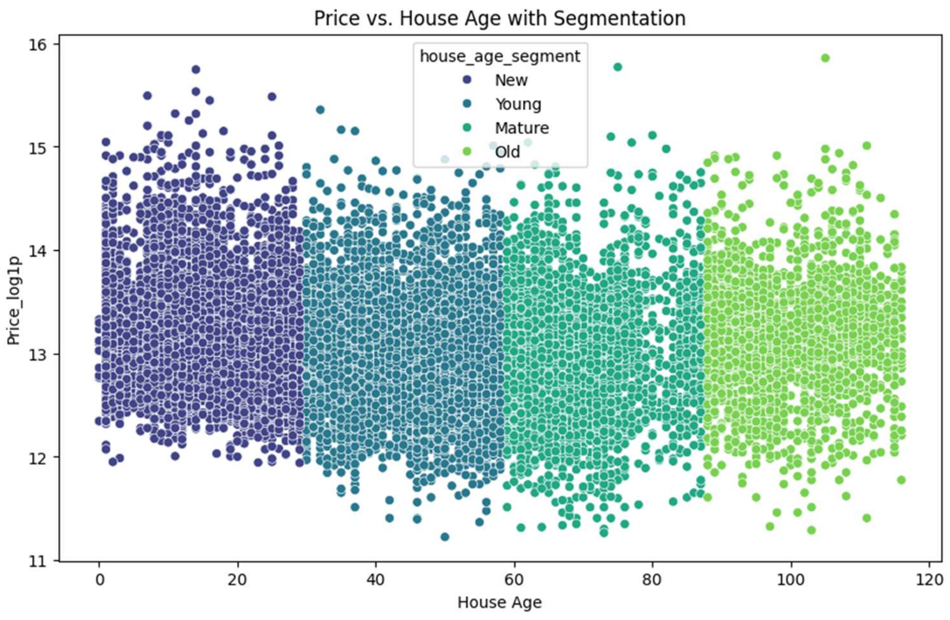
Total Area Segmentation vs. Median Price (Log1p)

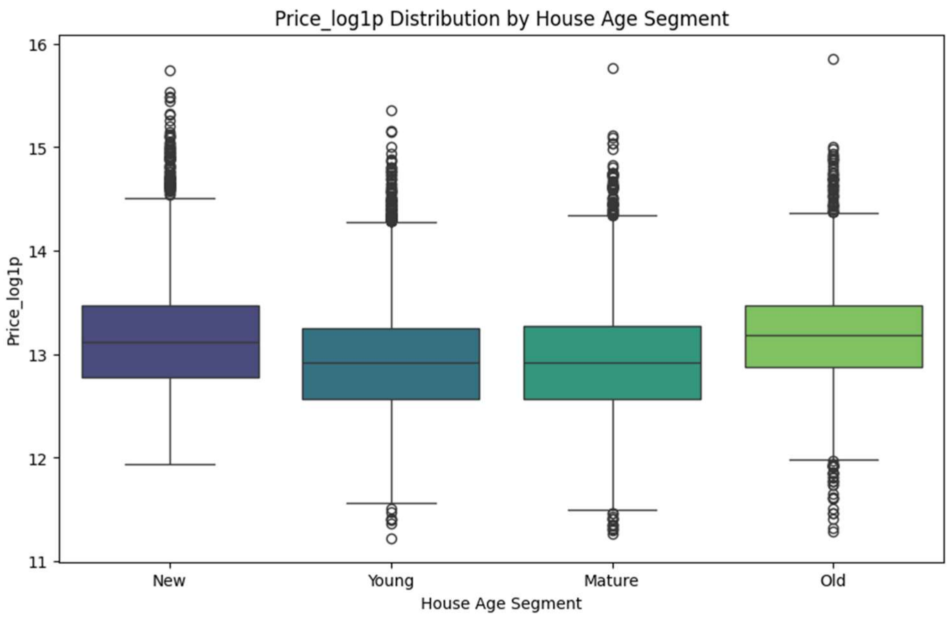




- **Price Increase with Size**: Median prices rise as property size increases, which is expected since larger properties generally have higher values.
- **Incremental Differences**:
  - Small to Medium: Minimal price difference ($0.016806).
  - Medium to Large: Significant price difference ($0.235028).
  - Large to Very Large: Largest price difference ($0.139534).
- **Overall Trend**: Median prices increase at an accelerating rate with property size, suggesting larger properties are significantly more valuable due to additional amenities, better locations, or other factors.

House Age Segmentation vs. Median Price (Log1p)





- **Old properties**: Highest median price, possibly due to unique features or prime locations.
- **New properties**: High median price, likely due to modern amenities and design.
- **Young and Mature properties**: Same median price, indicating similar market value.
- **Overall**: This analysis highlights how the age of a property can influence its market value, with both very old and very new properties commanding higher prices.

IV. Milestone 2

3. Data Preparation

Data Cleaning & Feature Selection

'cid' is a unique identifier with no predictive value, it is dropped to prevent data leakage or model bias.

Efficient Target Encoding for Zip Codes to Avoid High-Dimensionality in One-Hot Encoding

To avoid High-Demnsionality in One-Hot encoding of the categorical variable ‘zipcode’, the `target_encode` function performs target encoding on a categorical feature (like 'zipcode') to create a new numerical feature ('zip_code_target_encoded').

Target encoding replaces the categorical values with a numerical representation that reflects the average value of the target variable ('price_log1p' in this case) for each category. This helps incorporate the predictive power of the categorical feature into a model.

Here's a breakdown:

1. **Calculate the overall mean of the target variable:** `mean_target = df[target].mean()` This finds the average 'price_log1p' across the entire dataset.
2. **Group by the categorical feature and aggregate:** `agg = df.groupby(col)[target].agg(['count', 'mean'])` This groups the data by 'zipcode', then calculates the number of occurrences ('count') and the average 'price_log1p' ('mean') for each unique zip code.
3. **Smoothing:** `smooth = (counts * means + smoothing * mean_target) / (counts + smoothing)` This is the core of target encoding. It's a weighted average that combines the mean 'price_log1p' for each zip code with the overall mean 'price_log1p'. The `smoothing` parameter controls the influence of the overall mean. A higher `smoothing` value reduces the impact of zip codes with fewer observations, preventing overfitting to small sample sizes. In essence, it prevents extreme values from dominating the encoding for rare zipcodes.
4. **Mapping:** `return df[col].map(smooth)` This maps the original 'zipcode' values to the calculated smoothed means. This creates the new 'zip_code_target_encoded' column, where each zip code is now represented by a number that reflects its average 'price_log1p', adjusted for sample size using the smoothing parameter.
5. After this, the original `zipcode` column is dropped, because `zip_code_target_encoded` now captures its relevant information in a way that is usable for machine learning models.

Feature Categorization

Separating numerical and categorical features:

- Identifying numerical columns using select_dtypes(include=np.number).
- Identifying categorical columns using select_dtypes(exclude=np.number).
- Creating separate datasets for numerical (df_numerical) and categorical (df_categorical) features.

Feature Scaling (Standardization)

Standardizing numerical features using StandardScaler() to normalize values.

Encoding Categorical Features

- Appling one-hot encoding to categorical features using pd.get_dummies().
- Dropping the first category (drop_first=True) to avoid multicollinearity.

Data Integration

Merging the scaled numerical and encoded categorical data into a single dataset (df_processed).

Target Variable Handling

- Removing the target variable (price_log1p) from the feature dataset (X).
- Storing the target variable separately (y = df_processed['price_log1p']).

Train-Test Split

Spliting the dataset into training and testing sets:

- 70% training, 30% testing (test_size=0.3).
- Usind random_state=42 for reproducibility.

4. Baseline Model Performance Comparison

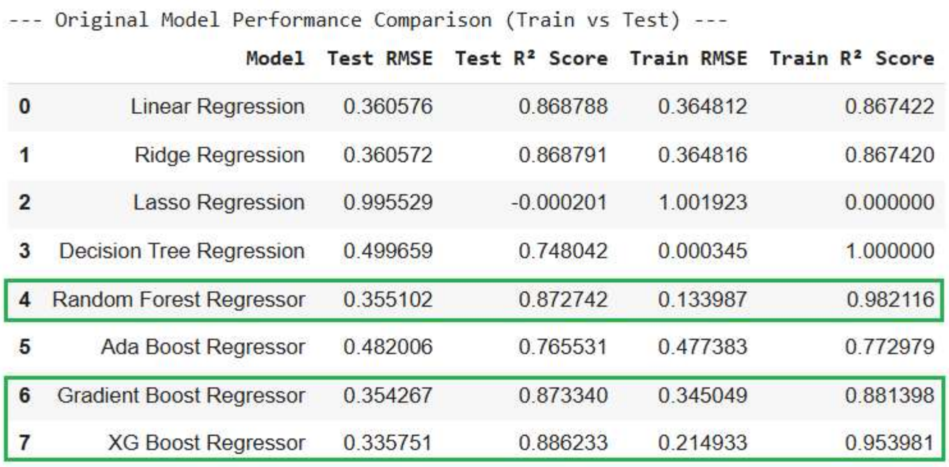
Train Models

- We define a dictionary named models that contains several regression models, each associated with a key representing the model's name.
- Here's a detailed description of the **Model Dictionary**:
    - Linear Regression: LinearRegression()
    - Ridge Regression: Ridge()
    - Lasso Regression: Lasso()
    - Decision Tree Regression: DecisionTreeRegressor()
    - Random Forest Regressor: RandomForestRegressor()
    - Ada Boost Regressor: AdaBoostRegressor()
    - Gradient Boost Regressor: GradientBoostingRegressor()
    - XG Boost Regressor: XGBRegressor()
- **Result Dictionaries:**
    - **results**: An empty dictionary intended to store the evaluation results of the models.
    - **results_fe**: Another empty dictionary intended to store evaluation results after feature engineering or other preprocessing steps.
- Create a for loop that trains and evaluates the multiple regression models:
    - **Train and Predict**:
        - For each model in models, train it with X_train and y_train.
        - Predict y_test using X_test.
    - **Evaluate**:

        - Calculate Root Mean Squared Error (RMSE).
        - Calculate R-squared (R^2) score.
    - **Store Results**:
        - Save RMSE and R^2 score for each model in the results dictionary.

Baseline Model Performance Comparison (Train vs Test)

- We added Train performance metrics RMSE and R^2 score.
- We drew a Model Performance Comparison (Train vs Test) of the Original Non-Tuned data frame.

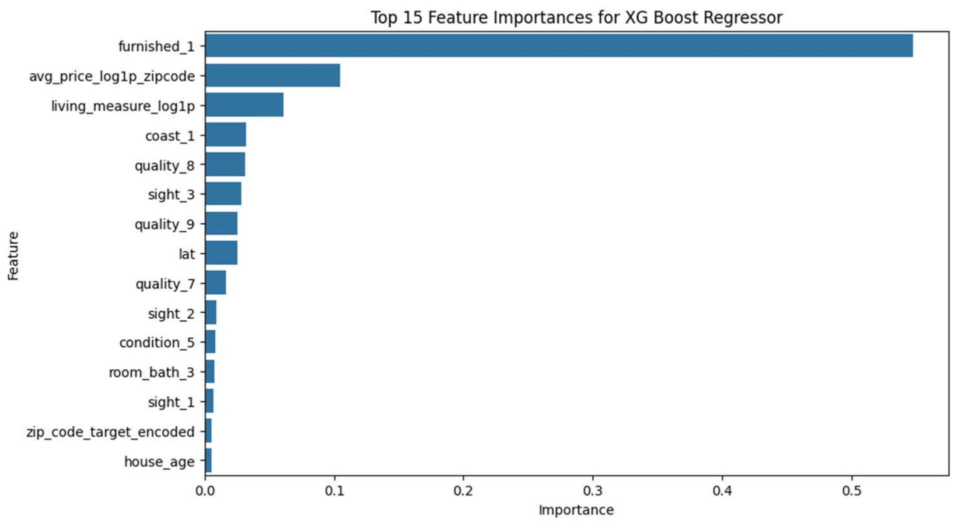


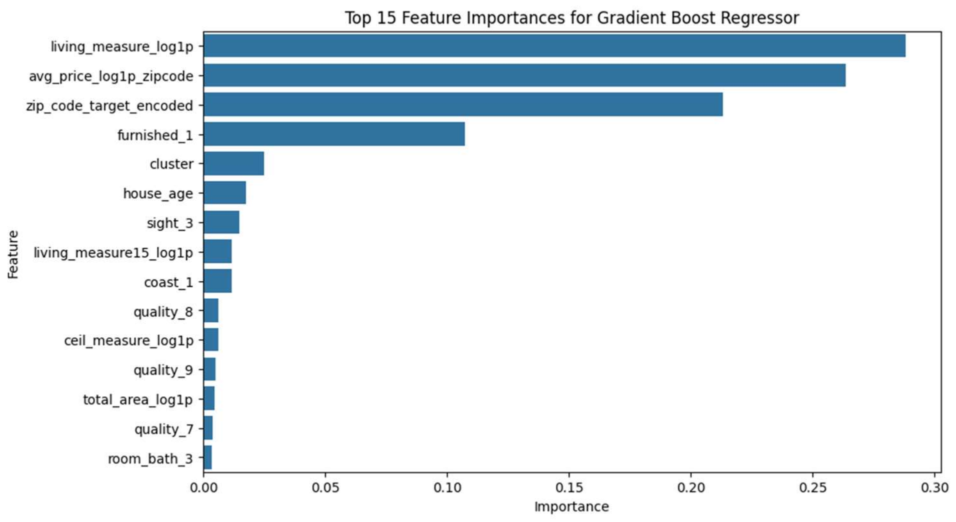
```
--- Original Model Performance Comparison (Train vs Test) ---
```

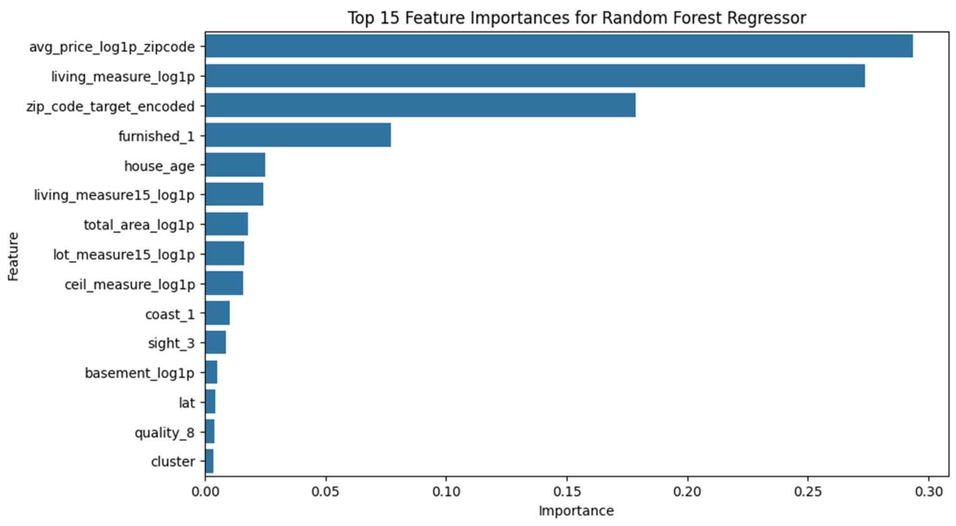
	Model	Test RMSE	Test R² Score	Train RMSE	Train R² Score
0	Linear Regression	0.360576	0.868788	0.364812	0.867422
1	Ridge Regression	0.360572	0.868791	0.364816	0.867420
2	Lasso Regression	0.995529	-0.000201	1.001923	0.000000
3	Decision Tree Regression	0.499659	0.748042	0.000345	1.000000
4	Random Forest Regressor	0.355102	0.872742	0.133987	0.982116
5	Ada Boost Regressor	0.482006	0.765531	0.477383	0.772979
6	Gradient Boost Regressor	0.354267	0.873340	0.345049	0.881398
7	XG Boost Regressor	0.335751	0.886233	0.214933	0.953981

- The table provides performance metrics for various regression models. The three best-performing models based on the lowest Test RMSE and highest Test R^2 Score are:
    - **XG Boost Regressor**:
        - **Test RMSE**: 0.335751
        - **Test R^2 Score**: 0.886233
    - **Gradient Boost Regressor**:
        - **Test RMSE**: 0.354267
        - **Test R^2 Score**: 0.873340
    - **Random Forest Regressor**:
        - **Test RMSE**: 0.355102
        - **Test R^2 Score**: 0.872742
- **XG Boost Regressor** is the best-performing model with the lowest test RMSE and highest test R^2 score, indicating excellent predictive accuracy and generalization.
- **Gradient Boost Regressor** and **Random Forest Regressor** also perform very well, with low test RMSE and high test R^2 scores, making them strong contenders.
- These three models show the best balance between low error (RMSE) and high explanatory power (R^2 Score) on the test data, indicating strong predictive performance.
- **Linear and Ridge Regression** models perform similarly and quite well, with low RMSE and high R^2 scores, indicating good predictive accuracy and generalization.
- **Ada Boost Regressor** has a higher RMSE and lower R^2 score compared to Linear and Ridge Regression, indicating it may not be the best choice for this dataset.

- The **Decision Tree Regression** model shows signs of overfitting. It performs exceptionally well on the training data but not as well on the test data. This discrepancy suggests that the model may not generalize well to unseen data.
- The **Lasso Regression** model is not performing well on this dataset. Both the training and test errors are high, and the R^2 scores are very low, indicating that the model is not fitting the data properly. This poor performance could be due to the nature of the dataset or the regularization parameter in Lasso Regression, which might be too strong, leading to underfitting.
- Created the function, **plot_feature_importances**, that visualizes the top 15 most important features of a given model.
- We plot the top 15 most important features for the top 3 performing models.



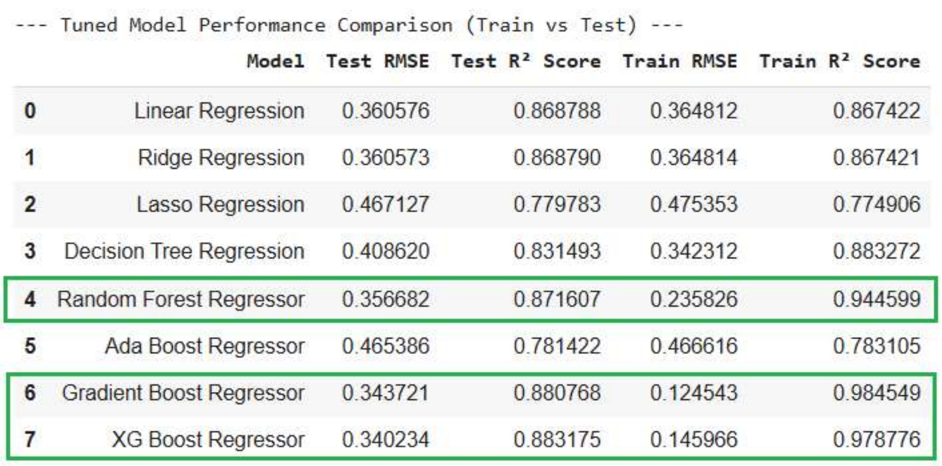




- Interpretation of Top Features for the top 3 Best Models:
  - **XG Boost Regressor:**
    - **furnished_1**: This feature has the highest importance (0.547867), indicating that whether a property is furnished is the most influential factor in the model's predictions.
    - **avg_price_log1p_zipcode**: The average price in the zip code (0.104854) is the second most important feature.
    - **living_measure_log1p**: The logarithm of the living area size (0.060682) is the third most important feature.
  - **Gradient Boost Regressor:**
    - **living_measure_log1p**: The logarithm of the living area size (0.288158) is the most important feature.
    - **avg_price_log1p_zipcode**: The average price in the zip code (0.263669) is the second most important feature.
    - **zip_code_target_encoded**: The target-encoded zip code (0.213280) is the third most important feature.
  - **Random Forest Regressor:**
    - **avg_price_log1p_zipcode**: The average price in the zip code (0.293686) is the most important feature.
    - **living_measure_log1p**: The logarithm of the living area size (0.273759) is the second most important feature.
    - **zip_code_target_encoded**: The target-encoded zip code (0.178785) is the third most important feature.
- **In summary:**
  - **XG Boost Regressor** places the highest importance on whether a property is furnished.
  - **Gradient Boost Regressor** and **Random Forest Regressor** both consider the living area size and average price in the zip code as highly important, with the Gradient Boost Regressor also emphasizing the **zip_code_target_encoded**.
  - These insights help understand which features each model prioritizes in making predictions.

Model Comparison using Hyperparameter Tuning

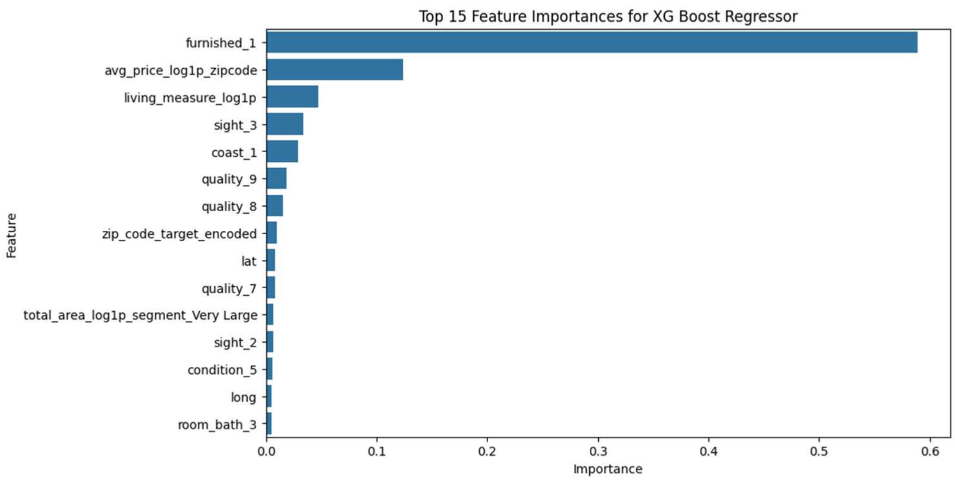
- After applying **RandomizedSearchCV** for hyperparameter tuning, we identified the best set of parameters for each model. The optimization process aimed to improve predictive performance by fine-tuning parameters using cross-validation**.**
- Here below the results of the tuned models:

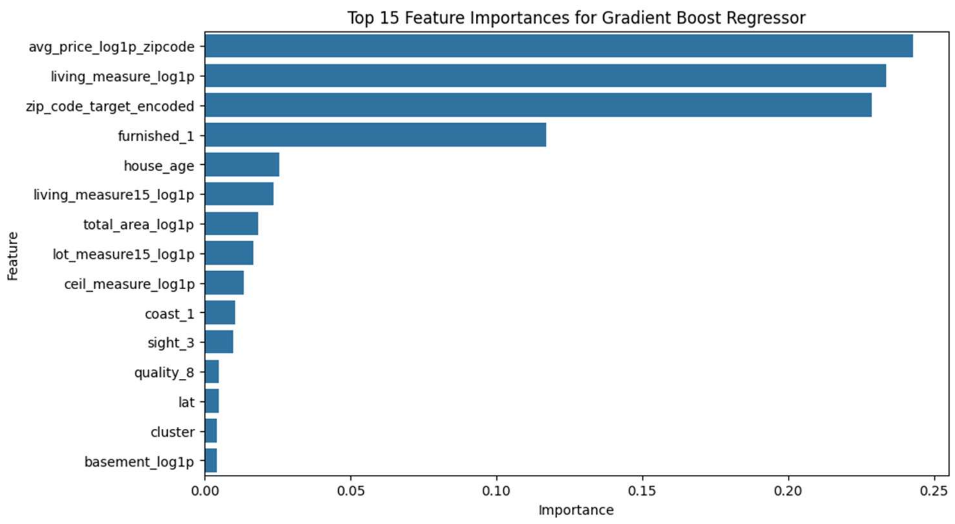


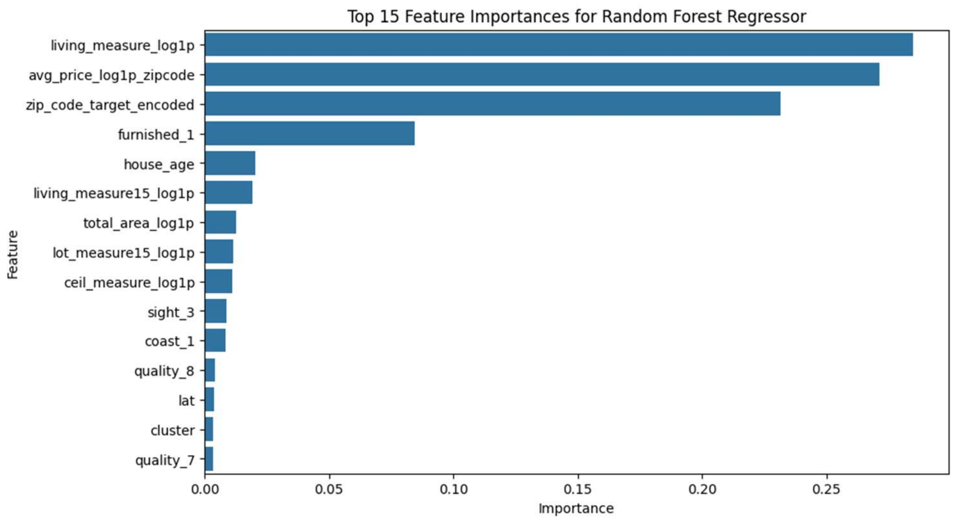
```
--- Tuned Model Performance Comparison (Train vs Test) ---
```

	Model	Test RMSE	Test R² Score	Train RMSE	Train R² Score
0	Linear Regression	0.360576	0.868788	0.364812	0.867422
1	Ridge Regression	0.360573	0.868790	0.364814	0.867421
2	Lasso Regression	0.467127	0.779783	0.475353	0.774906
3	Decision Tree Regression	0.408620	0.831493	0.342312	0.883272
4	Random Forest Regressor	0.356682	0.871607	0.235826	0.944599
5	Ada Boost Regressor	0.465386	0.781422	0.466616	0.783105
6	Gradient Boost Regressor	0.343721	0.880768	0.124543	0.984549
7	XG Boost Regressor	0.340234	0.883175	0.145966	0.978776

- The table provides performance metrics for the various tuned regression models. The three best-performing tuned models based on the lowest Test RMSE and highest Test R^2 Score are:
  - **XG Boost Regressor**:
    - **Test RMSE**: 0.340234
    - **Test R^2 Score**: 0.883175
  - **Gradient Boost Regressor**:
    - **Test RMSE**: 0.343721
    - **Test R^2 Score**: 0.880768
  - **Random Forest Regressor**:
    - **Test RMSE**: 0.356682
    - **Test R^2 Score**: 0.871607
- These three tuned models show the best balance between low error (RMSE) and high explanatory power (R^2 Score) on the test data, indicating they perform best on the test data.
- Let's plot the top 15 most important features for the top 3 performing tuned models.







Comparison of tuned and non-tuned top 3 best performing models

- **XG Boost Regressor**: The non-tuned model has a slightly better Test RMSE (0.335751 vs. 0.340234) and Test R^2 Score (0.886233 vs. 0.883175) compared to the tuned model.

- **Gradient Boost Regressor**: The tuned model performs better with a lower Test RMSE (0.343721 vs. 0.354286) and higher Test R^2 Score (0.880768 vs. 0.873326).
- **Random Forest Regressor**: The non-tuned model has a marginally better Test RMSE (0.354356 vs. 0.356682) and Test R^2 Score (0.873276 vs. 0.871607).
- Overall, the **non-tuned XG Boost Regressor** performs slightly better than its tuned counterpart, while the **tuned Gradient Boost Regressor** and **Random Forest Regressor** show improvements over their non-tuned versions.
- This comparison highlights the importance of tuning models to achieve optimal performance, though in some cases, non-tuned models can still perform exceptionally well.

Comparison of the other tuned and non-tuned models

- The **tuned Lasso Regression** model performs significantly better than the non-tuned version, both on the Test RMSE (0.467127 vs. 0.995529) and Test R^2 Score (0.779783 vs. -0.000201), and training data Train RMSE (0.475353 vs. 1.001923) and higher Train R^2 Score (0.774906 vs. 0.000000). This highlights the importance of tuning hyperparameters to improve model performance.
- The **tuned Decision Tree Regression** model outperforms the non-tuned version, with a lower Test RMSE (0.408620 vs. 0.498619) and a higher Test R^2 Score (0.831493 vs. 0.749091). This indicates that the tuned model has better accuracy and generalizes more effectively to the test data, avoiding the overfitting seen in the non-tuned model. Overall, tuning significantly improves the model's performance.

Overall Insights

- **XG Boost Regressor**: The non-tuned model performs slightly better than the tuned version, indicating that tuning may not always be necessary for optimal performance.
- **Gradient Boost Regressor**: The tuned model shows improved performance, highlighting the benefits of tuning for this model.
- **Random Forest Regressor**: The non-tuned model performs marginally better, suggesting that the default settings are already quite effective.
- **Lasso Regression**: The tuned model significantly outperforms the non-tuned version, demonstrating the importance of tuning for this model.
- **Decision Tree Regression**: The tuned model performs better, indicating that tuning helps avoid overfitting and improves generalization.

Conclusion: Tuning generally enhances model performance, especially for models like Gradient Boost, Lasso, and Decision Tree Regression. However, some models like XG Boost and Random Forest can still perform well without tuning, showing that the necessity of tuning can vary by model.

5. Model Comparison with Feature Engineering

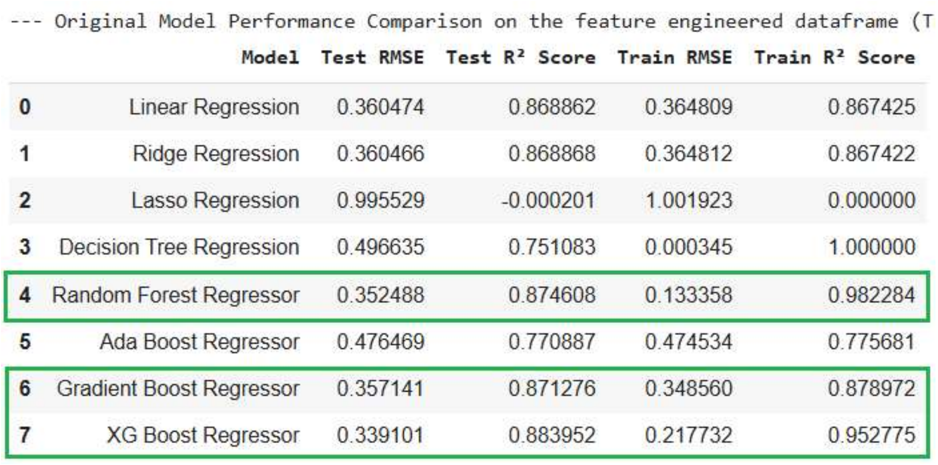
Feature Engineering

- ***Feature Engineering*** is essential in the data preprocessing phase of machine learning, as it involves creating new features from existing data to enhance predictive model performance.
- We have created a copy of the original Data Frame that we called df_fe to which we added the new features.
- Here's a concise breakdown of the feature engineering techniques I used:
- **Aggregated Features:**
    - **Feature**: premium_features_count
    - **Description**: Summarizes multiple features into one. For instance, premium_features_count totals premium features like cluster == 1, furnished == 1, and sight == 2, capturing the overall premium quality of a property.
- **Composite Features:**
    - **Features**: premium_features_score, luxury_features_score
    - **Description**: Combines multiple features into a single score. premium_features_score is a weighted sum of features (cluster == 1 with weight 3, furnished == 1 with weight 2, sight == 2 with weight 2.5). Similarly, luxury_features_score combines features (cluster == 2 with weight 3, furnished == 1 with weight 2, sight == 3 with weight 2.5) to represent the luxury level.
- **Interaction Flags:**
    - **Feature**: premium_flag, luxury_flag
    - **Description**: Binary features indicating specific conditions. premium_flag is set to 1 if cluster == 1, furnished == 1, and sight == 2, otherwise 0. luxury_flag is set to 1 if cluster == 2, furnished == 1, and sight == 3, otherwise 0.
- These techniques capture complex relationships and patterns in the data, making it easier for machine learning models to learn and make accurate predictions. By creating aggregated, composite, and interaction features, we enhance the dataset's richness and improve the model's generalization ability.

After completing feature engineering, we proceeded with the following steps: scaling numerical variables, one-hot encoding categorical features, integrating the scaled numerical and encoded categorical data, handling the target variable into X and y, and finally splitting the dataset into training and testing sets with a 70% training and 30% testing ratio (test_size=0.3).

Model Performance Comparison on the Feature Engineered data frame (Train vs Test)

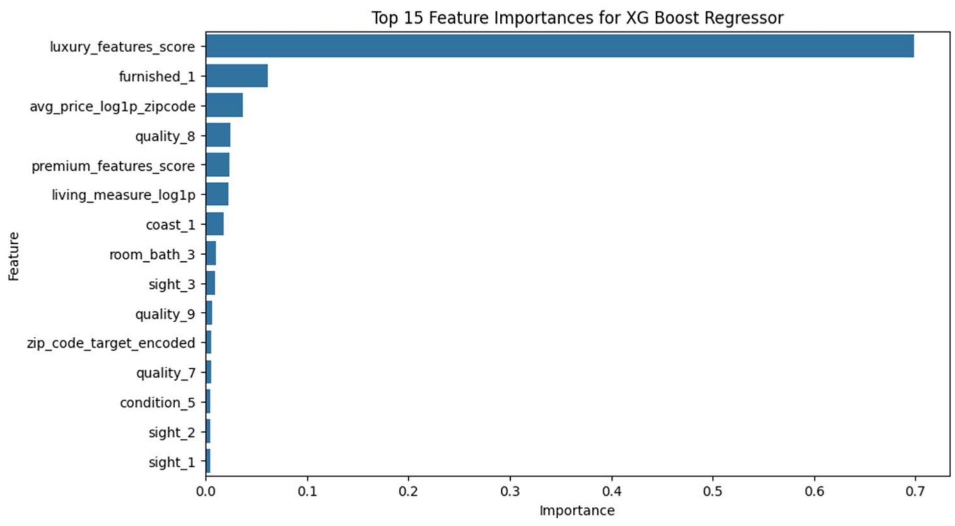
- We added Train performance metrics RMSE and R^2 score.
- We drew a Model Performance Comparison on the feature engineered Data Frame (Train vs Test) of the Original Non-Tuned data frame.



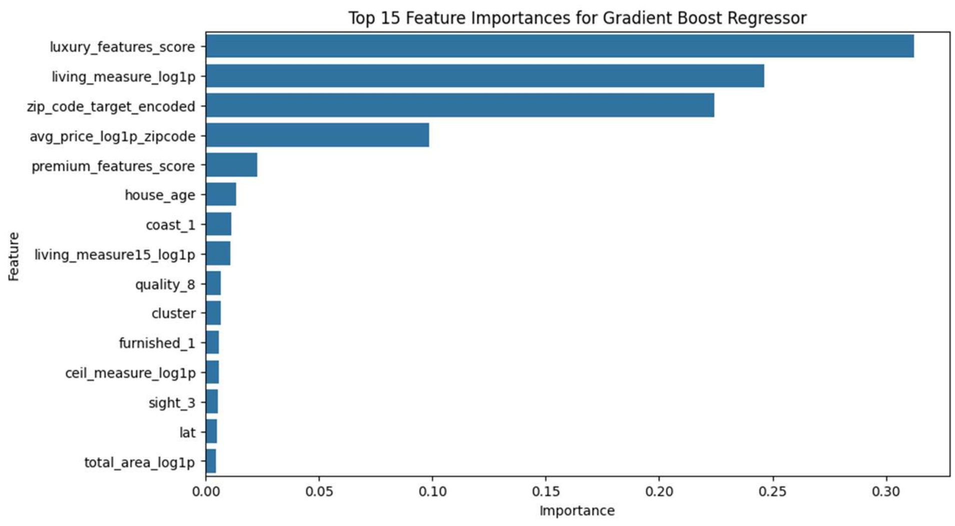
```
--- Original Model Performance Comparison on the feature engineered dataframe (T
```

	Model	Test RMSE	Test R² Score	Train RMSE	Train R² Score
0	Linear Regression	0.360474	0.868862	0.364809	0.867425
1	Ridge Regression	0.360466	0.868868	0.364812	0.867422
2	Lasso Regression	0.995529	-0.000201	1.001923	0.000000
3	Decision Tree Regression	0.496635	0.751083	0.000345	1.000000
4	Random Forest Regressor	0.352488	0.874608	0.133358	0.982284
5	Ada Boost Regressor	0.476469	0.770887	0.474534	0.775681
6	Gradient Boost Regressor	0.357141	0.871276	0.348560	0.878972
7	XG Boost Regressor	0.339101	0.883952	0.217732	0.952775

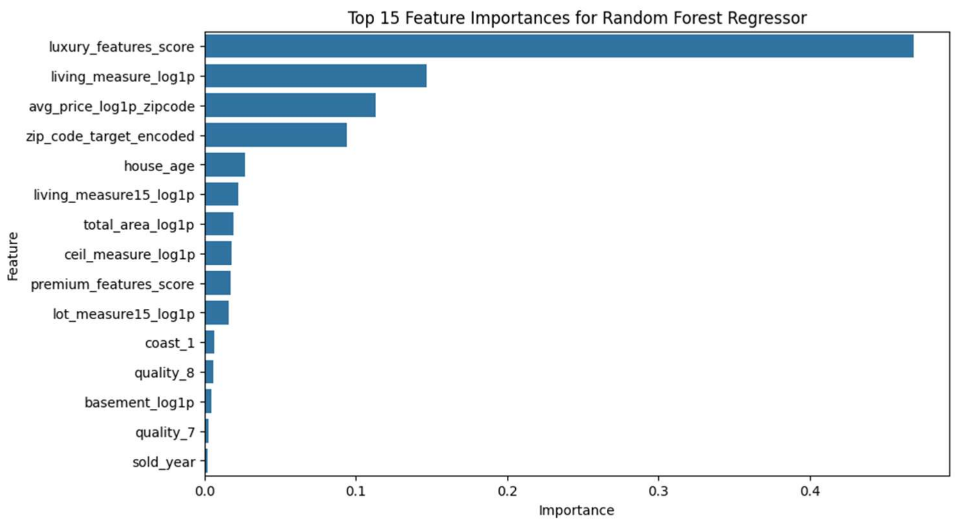
- Let’s plot the top 15 most important features for the top 3 performing feature engineered models.



Top 15 Feature Importances for XG Boost Regressor
luxury_features_score
furnished_1
avg_price_log1p_zipcode
quality_8
premium_features_score
living_measure_log1p
coast_1
room_bath_3
sight_3
quality_9
zip_code_target_encoded
quality_7
condition_5
sight_2
sight_1
Feature
Importance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7



Top 15 Feature Importances for Gradient Boost Regressor
luxury_features_score
living_measure_log1p
zip_code_target_encoded
avg_price_log1p_zipcode
premium_features_score
house_age
coast_1
living_measure15_log1p
quality_8
cluster
furnished_1
ceil_measure_log1p
sight_3
lat
total_area_log1p
Feature
Importance
0.00
0.05
0.10
0.15
0.20
0.25
0.30



Top 15 Feature Importances for Random Forest Regressor
luxury_features_score
living_measure_log1p
avg_price_log1p_zipcode
zip_code_target_encoded
house_age
living_measure15_log1p
total_area_log1p
ceil_measure_log1p
premium_features_score
lot_measure15_log1p
coast_1
quality_8
basement_log1p
quality_7
sold_year
Feature
Importance
0.0
0.1
0.2
0.3
0.4

Impact of Feature Engineering and Hyperparameter Tuning on Regression Model Performance

Top 3 Performing Models

- Here's a summary of the top 3 performing models from the feature engineered data frame, compared against the same models from the data frame without tuning or feature engineering, and the hyperparameter tuned results:
- **Top 3 Models with Feature Engineering:**
    - XG Boost Regressor: Test R^2 Score of 0.883952
    - Random Forest Regressor: Test R^2 Score of 0.874608
    - Gradient Boost Regressor: Test R^2 Score of 0.871276
- **Top 3 Models without Feature Engineering:**
    - XG Boost Regressor: Test R^2 Score of 0.886233
    - Gradient Boost Regressor: Test R^2 Score of 0.873324
    - Random Forest Regressor: Test R^2 Score of 0.872506
- **Top 3 Models with Hyperparameter Tuning:**
    - XG Boost Regressor: Test R^2 Score of 0.883175
    - Gradient Boost Regressor: Test R^2 Score of 0.881069
    - Random Forest Regressor: Test R^2 Score of 0.871148
- **Insights:**
    - **XG Boost Regressor** consistently performs well across all scenarios, showing robustness and high performance.
    - **Gradient Boost Regressor** and **Random Forest Regressor** also show strong performance, with slight improvements when hyperparameter tuning is applied.
    - **Feature engineering** and **hyperparameter tuning** both contribute to improved model performance, but the top models remain robust even without these enhancements.

Other Models Comparison:

- Let's take a closer look at the performance of the other models across the three scenarios: feature engineered, no tuning/feature engineering, and hyperparameter tuned.
- **Linear Regression:**
    - With Feature Engineering: Test R^2 Score of 0.868862
    - Without Feature Engineering: Test R^2 Score of 0.868788
    - With Hyperparameter Tuning: Test R^2 Score of 0.868788
- **Ridge Regression:**
    - With Feature Engineering: Test R^2 Score of 0.868868
    - Without Feature Engineering: Test R^2 Score of 0.868791
    - With Hyperparameter Tuning: Test R^2 Score of 0.868790
- **Lasso Regression:**
    - With Feature Engineering: Test R^2 Score of -0.000201
    - Without Feature Engineering: Test R^2 Score of -0.000201
    - With Hyperparameter Tuning: Test R^2 Score of 0.779783

- **Decision Tree Regression:**
  - With Feature Engineering: Test R^2 Score of 0.751083
  - Without Feature Engineering: Test R^2 Score of 0.746585
  - With Hyperparameter Tuning: Test R^2 Score of 0.831071
- **Ada Boost Regressor:**
  - With Feature Engineering: Test R^2 Score of 0.770887
  - Without Feature Engineering: Test R^2 Score of 0.774547
  - With Hyperparameter Tuning: Test R^2 Score of 0.780967
- **Insights:**
  - **Linear and Ridge Regression**: Both models show very consistent performance across all scenarios, indicating that they are not significantly impacted by feature engineering or hyperparameter tuning.
  - **Lasso Regression**: Shows poor performance without tuning, but hyperparameter tuning significantly improves its performance.
  - **Decision Tree Regression**: Benefits from hyperparameter tuning, showing a notable improvement in Test R^2 Score.
  - **Ada Boost Regressor**: Shows moderate performance across all scenarios, with slight improvements from hyperparameter tuning.
  - **Overall**, hyperparameter tuning generally improves model performance, especially for models like Lasso Regression and Decision Tree Regression. Feature engineering has a more varied impact, with some models benefiting more than others.

Feature Importance Comparative Analysis of Top Models Across Different Scenarios: Feature Engineering, Hyperparameter Tuning, and Baseline

Here's a table summarizing the top 3 models against the 3 different scenarios:

Scenario	*XG Boost Regressor Top 3 Features*	*Gradient Boost Regressor Top 3 Features*	*Random Forest Regressor Top 3 Features*
With Feature Engineering	luxury_features_score, furnished_1, avg_price_log1p_zipcode	luxury_features_score, living_measure_log1p, zip_code_target_encoded	luxury_features_score, living_measure_log1p, avg_price_log1p_zipcode
Baseline	furnished_1, avg_price_log1p_zipcode, living_measure_log1p	avg_price_log1p_zipcode, living_measure_log1p, zip_code_target_encoded	avg_price_log1p_zipcode, living_measure_log1p, zip_code_target_encoded
With Hyperparameter Tuning	furnished_1, avg_price_log1p_zipcode, living_measure_log1p	avg_price_log1p_zipcode, living_measure_log1p, zip_code_target_encoded	living_measure_log1p, avg_price_log1p_zipcode, zip_code_target_encoded

This table highlights the top features for each model under different scenarios.

Here are some insights based on the comparison of the top 3 models across different scenarios:

Feature Importance Consistency:

- **Luxury Features Score**: This feature is consistently important for models with feature engineering, indicating its strong predictive power when engineered features are included.
- **Avg Price Log1p Zipcode**: This feature appears in the top 3 for all models across all scenarios, highlighting its overall significance in predicting the target variable.

Impact of Feature Engineering:

- Models with feature engineering tend to prioritize the **luxury_features_score**, which is not present in models without feature engineering. This suggests that feature engineering can introduce new, highly predictive features that improve model performance.

Hyperparameter Tuning:

- The top features for models with hyperparameter tuning are similar to those without feature engineering, indicating that tuning primarily optimizes the model's parameters rather than altering feature importance significantly.

Model-Specific Insights:

- **XG Boost Regressor**: The feature furnished_1 is important in both the engineered and tuned scenarios, suggesting that this model benefits from categorical features.
- **Gradient Boost Regressor**: The feature zip_code_target_encoded is consistently important, showing that this model leverages target encoding effectively.
- **Random Forest Regressor**: The feature living_measure_log1p is consistently important, indicating that this model relies heavily on this feature for predictions.
- These insights can help in understanding how different preprocessing and tuning strategies impact model performance and feature importance.

6. Key Takeaways

Context & Objective

- Housing prices reflect economic trends, and their accurate prediction is crucial for buyers, sellers, and real estate investors.
- The study aims to predict house prices based on multiple features and identify the most influential attributes affecting property values.

Data Overview & Preparation

- The dataset includes 23 features, with price as the target variable.
- Key factors include structural details (bedrooms, bathrooms, area), property characteristics (condition, quality, renovation), geographical attributes (zip code, latitude, longitude), and market factors (sale date, waterfront view, furnishings).

Data Cleaning & Feature Engineering:

- Missing values were handled using KNN Imputer.
- Outliers were mitigated using Log1p transformation.
- Target encoding was applied to the categorical zip code feature to prevent high dimensionality from one-hot encoding.
- Feature scaling and encoding were performed for model training.
- New features were engineered (e.g., luxury and premium scores) to enhance predictive power.

Exploratory Data Analysis (EDA)

- Cluster Analysis (KMeans, k=3):
  - Cluster 2: Newer, larger houses with higher prices.
  - Cluster 1: Older houses with slightly lower prices.
  - Cluster 0: Intermediate in price and house size.
- Key Findings:
  - House size is directly correlated with price increases.
  - Oldest and newest properties tend to have the highest prices.
  - Zip codes show distinct price variations, with some commanding significantly higher values.

Model Comparison & Performance

- Baseline Models Tested:
  - XGBoost, Gradient Boosting, Random Forest, Decision Tree, Linear, Ridge, Lasso, and AdaBoost regressors.
- Top 3 best-performing models (based on R^2 and RMSE):
  - XGBoost Regressor (Best overall)
  - Gradient Boosting Regressor
  - Random Forest Regressor
- Hyperparameter Tuning:

- Improved performance for Gradient Boosting and Decision Tree, but XGBoost’s non-tuned version remained the best.

- Feature Engineering Impact:
    - The addition of luxury and premium scores improved model interpretability.
- Top predictive features:
    - Luxury Features Score (engineered feature)
    - Furnishing status
    - Zip Code’s average price
    - Living area size

7. Final Business Insights & Recommendations

- XGBoost is the best model for house price prediction due to its robust performance across all scenarios.
- Feature engineering added meaningful insights, with luxury features playing a crucial role in price determination.
- Hyperparameter tuning benefits some models (Gradient Boosting, Decision Tree) but is not always necessary for the best models.

8. Future improvements:

- Further refining feature engineering techniques.
- Testing additional models or ensemble approaches for improved accuracy.
- This analysis provides a solid foundation for predictive modelling in the real estate market, offering valuable insights into price determinants and data-driven decision-making.

Conclusion

The comparative analysis of the top models across different scenarios reveals several key insights:

1. **Feature Engineering's Impact**: Introducing engineered features like **luxury_features_score** significantly enhances model performance, as evidenced by its prominence in the top features for models with feature engineering.
2. **Consistent Importance of Key Features**: The feature **avg_price_log1p_zipcode** consistently appears across all models and scenarios, underscoring its critical role in predicting the target variable.
3. **Hyperparameter Tuning**: While hyperparameter tuning optimizes model performance, it does not drastically change the importance of features compared to the baseline models without feature engineering.
4. **Model-Specific Strengths**:
    - **XG Boost Regressor** benefits from categorical features like **furnished_1**.
    - **Gradient Boost Regressor** effectively utilizes target encoding with **zip_code_target_encoded**.
    - **Random Forest Regressor** relies heavily on **living_measure_log1p** for its predictions.

Overall, the analysis highlights the importance of feature engineering and the consistent value of certain features across different modelling approaches. These insights can guide future model development and optimization strategies.