# Housing Price Prediction Data Analysis using Decision Trees, Gradient Boosting and K-Nearest Neighbors

Student: Turcu Ciprian-Stelian

Master: High Performance Computing and Big Data Analytics

*E-mail:* ciprian.turcu@stud.ubbcluj.ro

**Abstract**

This study is based on data analysis for a housing price prediction problem using techniques such as Decision Trees, Gradient Boosting and K-Nearest Neighbors (KNN). The dataset used has a total of 545 entries with 13 in all, categorized by numeric and categorical attributes. The performance of each model was evaluated utilizing Mean Squared Error (MSE) and R-squared score. Of the three, Gradient Boosting Regression had the best predictive accuracy, capturing 66.5% of the variance in home prices. This study describes the pros and cons of each approach and their usability in data analysis, with ways that predictive performance may be improved.

## Introduction

Accurate housing prediction is indispensable for real estate planning, investment, and market analysis. Machine learning algorithms have revolutionized this domain by allowing data-driven predictions that consider many variables influencing house prices, thus enabling an easy way of analyzing data. This study compares the performances of three regression models: Decision Tree, Gradient Boosting, and KNN on a dataset of numeric and categorical features. This study tries to identify the best approach in the analysis and prediction for housing prices.

## Dataset and Processing Methodologies

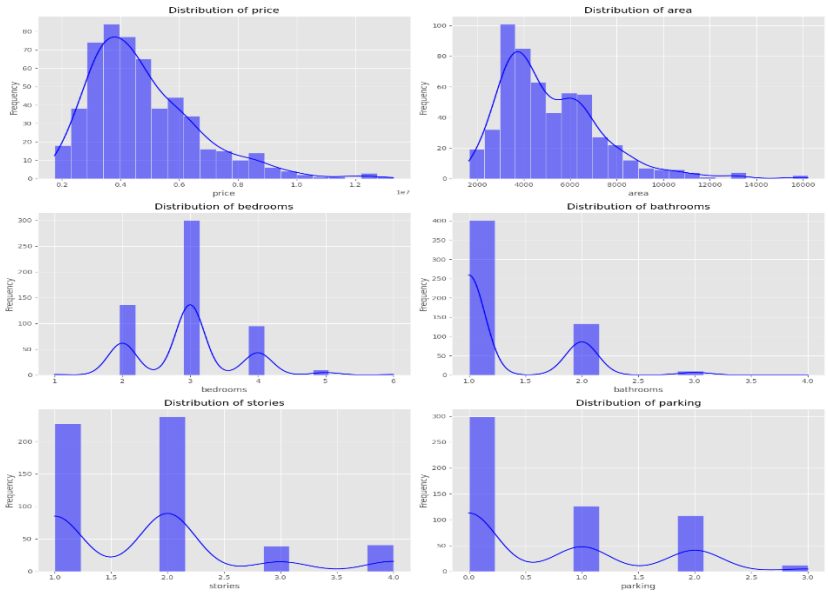
The used dataset consists of 545 entries with 13 features, providing information describing various characteristics of houses, including their prices and related attributes. The features can be categorized into numeric and categorical variables. They are split as follow:

1. **Numeric Features:**
    - **price**: Represents the house price.
    - **area**: Size of the property in square feet.
    - **bedrooms**: The number of the bedrooms in the house.
    - **bathrooms**: The number of the bathrooms in the house.
    - **stories**: The number of floors each property has
    - **parking**: The number of parking spaces available.
2. **Categorical Features:**
    - **mainroad**: Whether the property is on a main road (yes/no).
    - **guestroom**: Presence of a guestroom (yes/no).
    - **basement**: Presence of a basement (yes/no).
    - **hotwaterheating**: Presence of hot water heating (yes/no).
    - **airconditioning**: Presence of air conditioning (yes/no).
    - **prefarea**: Whether the property is in a preferred area (yes/no).
    - **furnishingstatus**: Furnishing level of the property (**furnished**, **semi-furnished**, **unfurnished**).

## Numeric Features

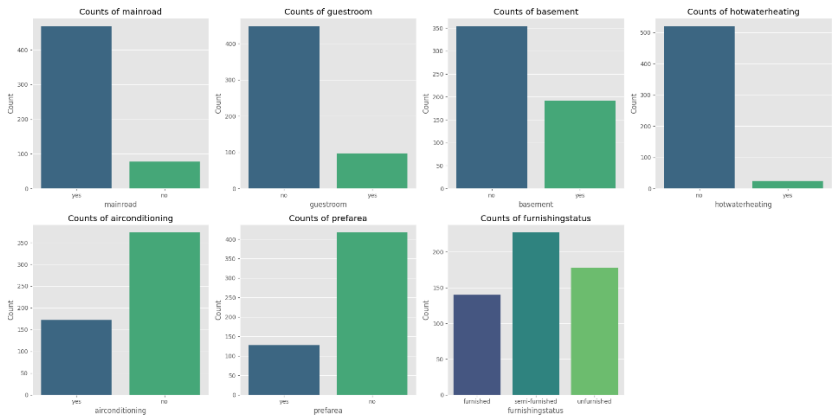
The numeric features distribution conveys valuable insights into the dataset. The house prices range from 1.75 million to 13.3 million, with the mean located around 4.77 million. The distribution is skewed towards the right indicating the presence of some high-priced outliers. Similarly, the area variable, which shows the size in square feet of houses, goes from 1,650 to 16,200 sqft, and on an average, its value is found to be roughly 5,150 sqft. This again is skewed towards the right side of its graph.

The distribution of bedrooms indicates a high concentration around 2 to 3 bedrooms, which denotes the fact that most houses stick a standard configuration. As for bathrooms the distribution of the variable closely sits at the 1 or 2 values; with a far less frequency for 3 or more bathrooms. The stories feature predominantly shows values of 1 or 2 floors, while the parking variable indicates that most properties have 0 or 1 parking spaces, with very few houses having more than 2.



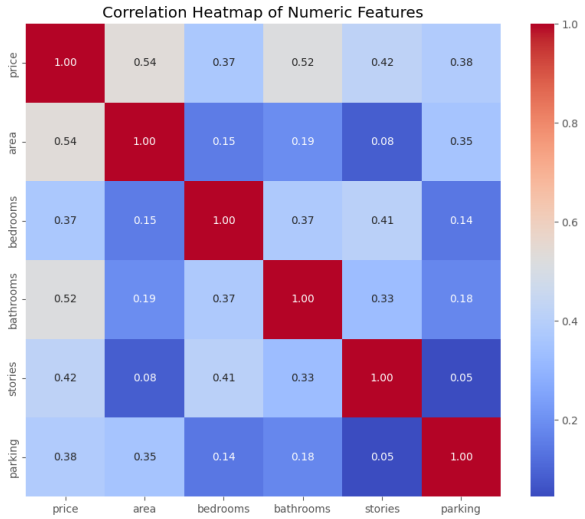

## Categorical Features

Analyzing the distribution of the categorical features provides additional context about the properties. A vast majority of the houses are located on a main road as marked by the `*mainroad*` variable. Most houses don't have a guest room `*guestroom*` or a basement `*basement*`, with very few that have hot water heating `*hotwaterheating*`. However, quite a lot of them are provided with air conditioning `*airconditioning*` and are situated in preferred areas `*prefarea*`. Regarding furnishing status `*furnishingstatus*`, the dataset includes a balanced mix of furnished, semi-furnished, and unfurnished properties.



## Correlation Analysis

The correlation heatmap displays important relations among the numeric features. The price variable, which is the target to be predicted, strongly positively correlates with area, indicating that larger houses are likely to be more expensive. Also, moderate correlations are observed between price and stories as well as price and bathrooms. The features bedrooms and parking have even weaker correlations to the target variable but may still hold some contribution toward the prediction model. On the whole, the correlations give an indication that area might be the most valuable numeric feature to predict house price.




## Methodologies Used in Data Processing

The analysis used Python-based visualization and statistical tools to explore the dataset further. Libraries like matplotlib and seaborn were used to create histograms for numeric features, bar plots for categorical variables, and a heatmap of correlations. These scripts provided a systematic way of looking at the distributions of key features, finding patterns, and understanding the relationships in the data. Methodologically, it will depend on exploratory data analysis—EDA—to discover insights into feature distributions, relationships, and potential predictors relevant to house price modeling.

## Theoretical Foundations

### 1. Decision Tree Regression

Decision Tree Regression is a supervised machine learning technique used to predict continuous outcomes by recursively partitioning data into subsets based on feature values. The decision tree regression builds a tree-like model where every internal node represents a decision on some feature, the branches represent the outcomes of the decisions, and the leaf nodes represent the predicted continuous values. [1]

The process starts with the complete dataset at its root node. The algorithm considers all possible splits over all features in order to identify the partition that will minimize the variance of the subsets

resulting from it. That is making data points in each subset as homogeneous as possible with respect to the target variable. This splitting criterion is important for the model accuracy, as it directly influences the integrity of the leaf nodes. Once the optimal split is determined, the data is divided accordingly, and the algorithm recursively applies the same process to each subset. The algorithm is based on a top-down greedy strategy known as recursive binary splitting to construct the tree. At each node, it selects the feature and threshold that would minimize a defined loss function, which, in most cases for regression tasks, is the mean squared error (MSE). This procedure is recursively repeated until a stopping criterion, usually a maximum depth of the tree or a minimum number of samples per leaf node is reached.

The foundational work on decision trees was presented by Breiman et al. [2] in their 1984 monograph, Classification and Regression Trees, which introduced the CART algorithm for constructing decision trees in both classification and regression contexts.

**Key Features and Structure**

The structure of a decision tree comprises of several components:

- **Root Node:** Represents the entire dataset, which is subsequently divided into subsets.
- **Decision Nodes:** Intermediate nodes that split the data based on specific features.
- **Leaf Nodes:** Terminal nodes that provide the predicted output.

**Advantages**

Decision Tree Regression comes with great advantages that offer strong usability reasons for data analysis tasks. First key strength is represented by the interpretability of the hierarchical structure of decision trees which makes them intuitive and easy to interpret, even for individuals without technical background. The decision-making process can be visualized with clarity on how predictions are derived. Furthermore, the flexibility in data types enable decision trees to handle both numerical and categorical data, thus they can work in various types of datasets, modeling non-linear relationships without transforming or normalizing the data. At last, compared to other algorithms, decision trees require less data preprocessing. They are also robust to outliers and can handle missing values, so there is less need for extensive data cleaning. [3] [1]

**Disadvantages**

Despite their advantages, decision trees have certain shortcomings as well. Decision trees tend to overfit, especially if they become deep; in particular, they can model the noise in the training data. This can make them generalize poorly on new, unseen data. However, such issues are usually mitigated through the use of pruning techniques or setting a constraint on the parameters of the models. They also suffer from instability, slight perturbations in the data can produce drastically different tree structures, making decision trees unstable. This susceptibility can alter the model reliability, particularly in the presence of noisy data. When there is an imbalance in the classes withing the dataset, decision trees can become biased toward the dominant class, resulting in poor predictive performance for minority classes. Resampling or adjusting class weights may help with this issue but must be used carefully. [3] [1]

## 2. Gradient Boosting Regression

Gradient Boosting Regression serves as an advanced ensemble learning technique that improves the predictive performance of multiple weak learners, usually decision trees, by sequentially combining

them into a strong composite model. It iteratively builds models by optimizing a loss function to correct errors made by previous models, thereby refining the overall predictive capability. [4]

The process initiates with the initialization of a base model; usually, it is a constant value that minimizes the loss function over the training data. The subsequent models then trained to predict the residuals, represented by the differences between the actual target values and the predictions made by the ensemble so far. Focusing on these residuals means that each new model tries to learn a part of the data aspects that the previous models have not yet captured. [4]

At every step $m$, the algorithm tries to find a new function $h_m(x)$ for the existing model $F_{m-1}(x)$ such that the updated model $F_m(x) = F_{m-1}(x) + y_m h_m(x)$ minimizes the loss function $L(y, F(x))$. Here $y_m$ represents the step size or the learning rate of the contribution made by the new function to the ensemble. Function $h_m(x)$ is then selected to approximate the negative gradient of the loos function with respect to the model's predictions evaluated over the training data. This is in line with the gradient descent optimization principle, where the model is iteratively updated in the direction that most reduces the error. [4]

The gradient boosting concept was first proposed by Jerome H. Friedman in his seminal paper, "Greedy Function Approximation: A Gradient Boosting Machine" [5], 1999, published in the Annals of Statistics. Friedman's work founded and provided a framework for gradient boosting algorithms applicable in both regression and classification tasks.

**Key Features and Structure**

The architecture of Gradient Boosting Regression involves several critical components [6]:

- **Ensemble Learning**: It creates an ensemble of weak learners where each is aiming to rectify the mistakes of its predecessors, thereby enhancing overall model accuracy, and so that the final model becomes more precise.
- **Additive Modeling**: The models are added one at a time, and every new model is introduced to improve upon the combined performance of existing models.
- **Gradient Descent Optimization**: The algorithm uses gradient descent to minimize a defined loss function, which steers the model toward making optimal predictions.

**Advantages**

Gradient Boosting Regression comes with great advantages that offer strong usability reasons for data analysis tasks. First key strength is represented by the high predictive accuracy, often results in better predictive accuracy compared to individual models by iteratively correcting its errors. Furthermore, flexibility of the gradient boosting method can be used with many loss functions and hence it is applicable to a wide variety of regression problems. At last, feature importance evaluation, this technique provides insights into feature importance and thus helps in the interpretation of the predictions of the model. [7]

**Disadvantages**

However, Gradient Boosting Regression comes with limitations as well. Computation intensity represents a first possible step back as the iterative nature and complexity in these models might result in increasing the computational demands with larger datasets. This technique is susceptible to overfitting the training data, specifically when the number of iterations is high or whenever the complexity of the model is not well controlled. Lastly, parameter sensitivity represents a challenge as performance heavily depends on the careful choice of hyperparameters, which may turn out to be time-consuming. [7]

### 3. K-Nearest Neighbors Regression

K-Nearest Neighbors Regression is a non-parametric, instance-based learning algorithm used for the prediction of continuous outcomes. The procedure works first by finding the 'k' closest training samples around a given input and then making an estimation of the target value as the mean of these neighbors' target values. In this way, it assumes that similar input features have similar output values; hence, it enables the model to learn local patterns in data. [8]

The K-Nearest Neighbors algorithm was proposed by Evelyn Fix and Joseph Hodges in 1951 as a non-parametric method for pattern classification. Their seminal work paved the way for instance-based learning algorithms, which find applications in many areas since then, including regression analysis. [9]

The algorithm starts by calculating the distance between the input sample and all samples in the training dataset. It is common to use metrics such as Euclidean distance. Once the 'k' nearest neighbors are found, the algorithm calculates the mean of their target values to generate the prediction for the input sample. The choice of 'k' affects the model a lot: smaller 'k' might give high variance and overfitting, while a larger 'k' might reduce bias and cause underfitting. So, the optimal choice of 'k' is important in finding the right trade-off between bias and variance—one of the important aspects in model tuning.

The prediction for a given input vector *x* is computed by averaging the target values of its k nearest neighbors in the feature space. This process involves several key steps:

1. Distance Calculation: For an input vector x and each data point $x_i$ in the training set, the distance $d(x, x_i)$ is computed. A common choice is the Euclidean distance defined as: $d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$ , where n is the number of features, $x_j$ is the j-th feature of x, and $x_{ij}$ is the j-th feature of $x_i$
2. Identification of Nearest Neighbors: The k training samples with the smallest distances to x are selected as the nearest neighbors.
3. Prediction: The predicted value $\hat{y}$ for the input vector x is obtained by averaging the target values $y_i$ of the k nearest neighbors: $\hat{y} = \frac{1}{k} \sum_{i \varepsilon N_k(x)} y_i$, where $N_k(x)$ denotes the set of indices corresponding to the k nearest neighbors of x.

The most critical decision to be made in KNN Regression is the choice of k. A small k tends to have high variance and may overfit, as it can capture noise in the training data. On the other hand, a large k increases bias and may underfit by over smoothing the data. Thus, choosing an optimal k is essential to balance bias and variance, a key task in model tuning. [10]

**Key Features and Structure**

The architecture of KNN Regression encompasses several critical components:

- **Instance-Based Learning:** KNN is a type of lazy learning algorithm, where no explicit model is constructed during the training phase. It memorizes the entire training dataset and makes predictions by computing similarities between the input sample and stored instances.
- **Distance Metrics:** Most algorithms depend on a distance measure, like the Euclidean distance, which quantifies how similar two data points are. This is important since the distance metric selected, to a great degree, affects the model performance in deciding which neighbors are closest.

- **Parameter `k`:** The number of neighbors is considered a very important hyperparameter. Optimal selection of 'k' balances bias and variance: with a small 'k', there may be high variance, resulting in overfitting, while large 'k' increases bias, which can lead to underfitting.

**Advantages**

K-Nearest Neighbors Regression comes with great advantages that offer strong usability reasons for data analysis tasks. First key strength is represented by the simplicity and ease of implementation, as the algorithm is quite simple to implement and comprehend; hence, it is very accessible to both practitioners and researchers. Furthermore, flexibility is an important aspect as the KNN can be applied to classification and regression problems and can model complex, non-linear relationships without assuming a specified form for the underlying distribution of data. At last, since KNN is an instance-based learner, it can immediately accommodate new data, with no requirement for retraining, hence allowing updates in dynamic environments quite efficiently. [8]

**Disadvantages**

Despite its strengths, KNN Regression has certain limitations as well. The algorithm can be presented as computationally intensive for big datasets since it involves a computation of distances between every input sample and all training samples. KNNs are sensitive to the to the choice of distance metric and irrelevant/noisy features, which can adversely affect its predictive accuracy. As the number of features increases, the distance between data points becomes less distinguishable, potentially degrading the performance of the algorithm. [8]

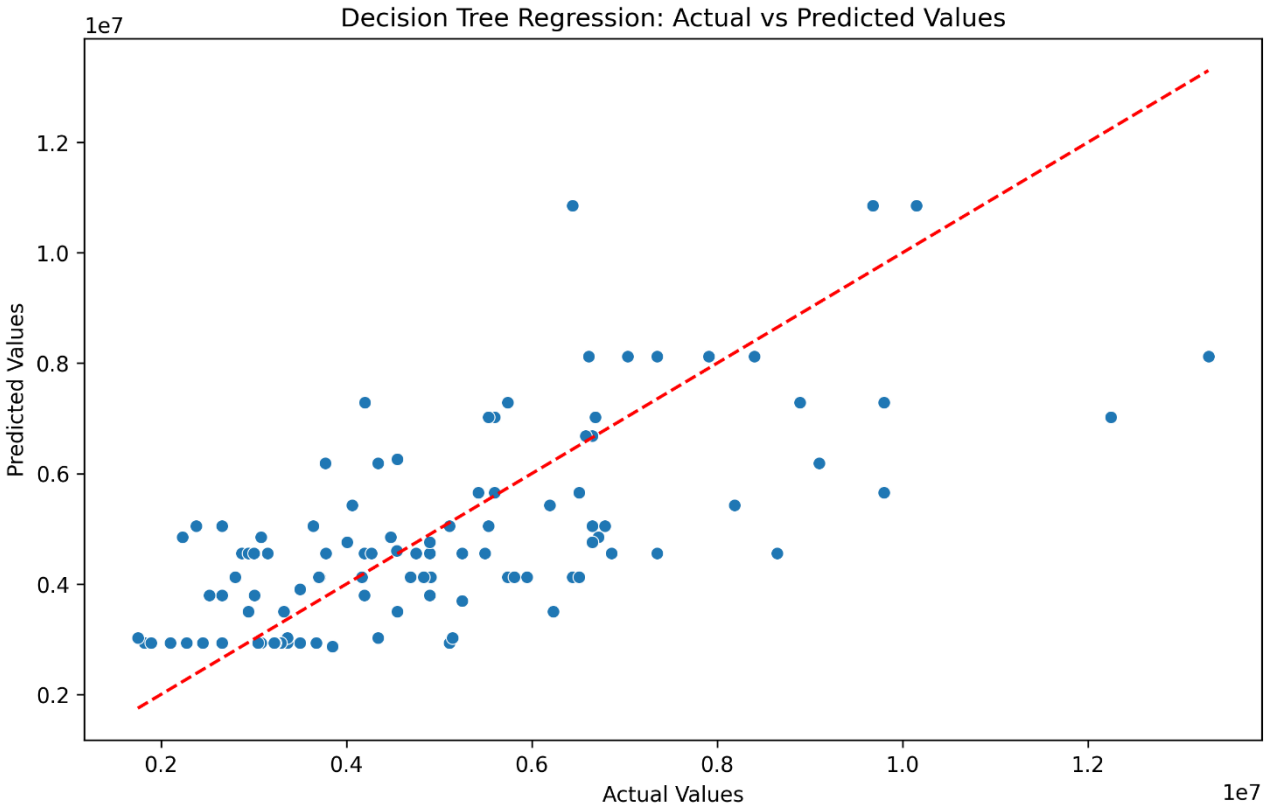
**Decision Tree Regressor Results**

The analysis began with the preprocessing of data, which started by loading the dataset with pandas [11] and understanding the structure and content of the data. It identified the missing values and handled them to maintain the quality of the data. Further, the categorical variables were transformed into numerical representations using One-Hot Encoding, making them suitable for regression modeling.

Afterwards, the data was split into predictor variables and the target variable: X, consisting of all the features, and y, which represents house prices. The data was then divided into an 80-20 split to train and test the model.

The model was trained on the training data using a regressor with the default arguments of a maximum depth of 5 to prevent overfitting. Predictions were done on the test set, and performance was gauged with Mean Squared Error and R-squared Score. The Mean Squared Error gives the average of the squared differences between actual and predicted values, while $R^2$ gives the variance in housing prices explained by the model.

Finally, an actual versus predicted scatter plot was developed to show the relationship between actual and predicted values. It includes a reference line for perfect predictions to help in assessing the predictive accuracy of the model performance.




The performance of the model on the test set gave an MSE of 2,701,167,171,509.852 and an R-squared Score of 0.4656. The huge MSE is an indication of the large scale of the target variable, which represents housing prices. This suggests large deviations between actual and predicted values. The $R^2$ score suggests that approximately 46.56% of the variability in housing prices is accounted for by the model, leaving 53.44% unexplained. This moderate predictive power implies that while the model captures some underlying patterns, it lacks critical variables or complexity to fully represent the factors influencing housing prices.

To enhance model performance, several steps can be considered. Feature engineering may introduce new variables or transform the existing ones in a way that captures better the underlying patterns, including interaction terms or polynomial features to model nonlinear relationships. Model complexity can be adjusted by trying different parameters, for example, increasing the maximum depth of the decision tree, which allows the model to capture more intricate patterns; however, caution is needed to avoid overfitting, which can be mitigated through techniques like cross-validation. It may be that other modeling techniques, such as Random Forests or Gradient Boosting Machines, could do better as it combines multiple decision trees in a way that reduces the variance. Moreover, collecting more data will give the model more information with which it may generalize better. Ensuring the model does not suffer from multicollinearity, which can adversely affect performance, is also crucial; techniques like variance inflation factor (VIF) analysis can help identify and address such issues.

In summary, the Decision Tree Regressor did a fairly good job in predicting housing prices but still had low variability that it was able to capture, thus leaving a big amount of unexplained variance. Such predictions can be improved by refinement in feature selection, modification in model parameters, using higher-order algorithms, and ensuring data quality.

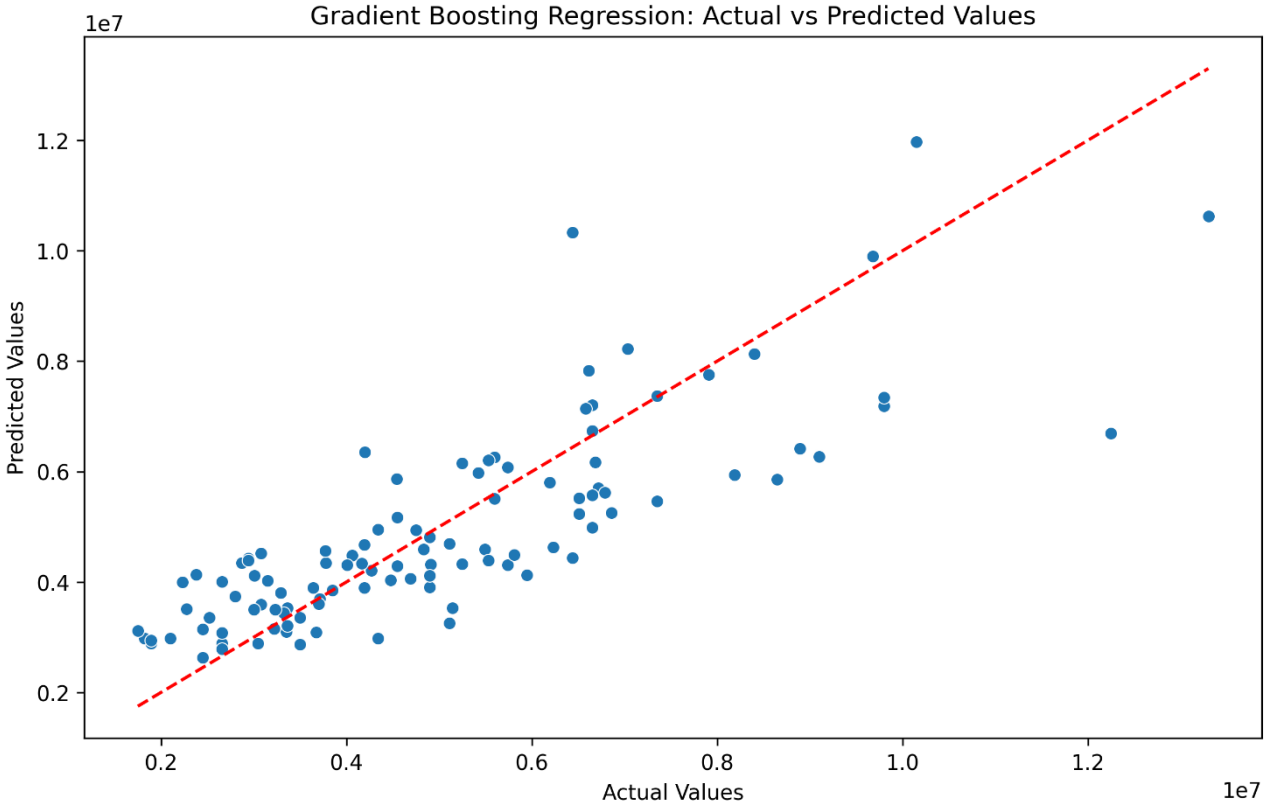
**Gradient Boosting Regression Results**

The analysis began with the preprocessing of data, where the dataset was first loaded using pandas, followed by exploratory data analysis to understand the structure and content of the data. The missing values were identified and handled appropriately to ensure the quality of the data. Further, the categorical variables were transformed into numerical representations using One-Hot Encoding, which prepared them for regression modeling.

The dataset was then preprocessed, and split into predictor variables X and target variable y, with X comprising all the features and y being the housing prices. Afterwards, the data was split into training and testing sets, keeping an 80-20 split to allow model training and evaluation in order.

The estimators are initialized to 100, the learning rate set to 0.1, and the maximum depth set to 3; hence, a Gradient Boosting Regressor was fitted on the training data. Predictions on the test set were made, with evaluation using Mean Squared Error and R-squared Score. MSE will calculate the average of the squared differences between actual and predicted values, while $R^2$ will determine what variance in the housing prices can be explained by the model.

A scatter plot showing actual vs. predicted values was done with a reference line that corresponds to perfect predictions, enabling us to gauge how far the model was from the real values.




We then get from this model the Mean Square Error: 1,693,306,118,911.061, R-squared Score: 0.6650. High value aside for MSE represents that target variable house prices is on very big scale hence differences between actual and predicted values are substantial. The $R^2$ value infers that the model explains about 66.50% of the variance in housing prices, while approximately 33.50% remains unexplained. This partial predictive power shows that there is some underlying pattern the model captures, though critical variables or complexity are not present in the model to represent the factors of housing prices fully.

A number of steps could be followed in order to raise the performance of the model: feature engineering may introduce new variables or transform the existing variables to capture underlying

patterns, for example, incorporating interaction terms or polynomial features that can model nonlinear relationships. Alternatively, the complexity of the model can be increased by fiddling with its parameters: for example, by increasing the number of estimators or optimizing the learning rate. This may allow the model to learn even more complex patterns. However, it is very important to avoid overfitting by using techniques such as cross-validation. Using other approaches to modeling, such as XGBoost or LightGBM, might lead to further improvements of the gradient boosting framework. More data can be collected to have more information from the model, which can generalize better. Besides, assurance of the model against multi-collinearity, which may seriously affect its general performance, is necessary; techniques involved in this aspect include VIF-based analysis, which assists in detecting and removing such issues.

The Gradient Boosting Regressor did good in predicting house prices, as it captured some of the variability but left much unexplained. Predictive performance can be improved by refining the feature set, adjusting model parameters, exploring advanced algorithms, and ensuring data quality for more accurate and reliable predictions.

**K-Nearest Neighbors Results**

Analysis involved the preprocessing of data by loading the dataset from pandas, performing exploratory data analysis to understand the structure and content of the data, finding missing values, and handling them accordingly and quality checking. Then, the categorical variables were encoded into numerical variables using One-Hot Encoding, thus preparing the data for regression modeling.

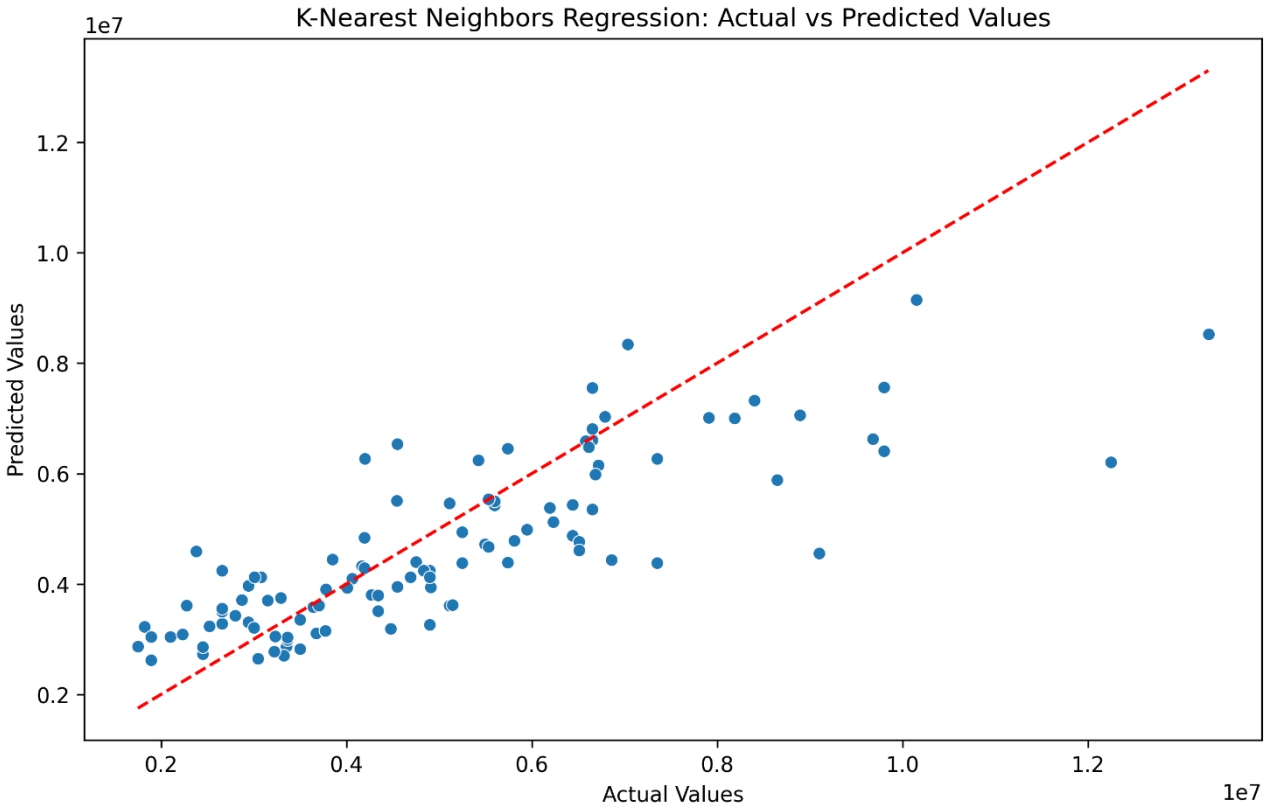
Further, the data was divided into predictor variables X and the target variable y, where X consisted of all features and y consisted of housing prices. After that, the data was split into a training set and a test set, keeping in mind to maintain an 80-20 split to allow for model training and subsequent model testing.

Because K-Nearest Neighbors operates based on distances, it is sensitive to scale; hence, the features needed to be standardized using StandardScaler to put all features on an equal footing in the distance calculations intrinsic in KNN algorithms. This is an important step because without standardization, features with a greater numeric range could end up dominating the model.

KNN regressor instance was created with n_neighbors = 5 and then fitted to the scaled training data. Next, it was used to make the prediction on the scaled test set; then, the performance of the regression model was done by calculating the Mean Squared Error (MSE), the average of the squared differences between actual and predicted, and the R-squared ($R^2$) Score as the proportion of the variance in house prices explained by the model.

Finally, an actual vs. predicted scatter plot was presented, including a reference line for perfect predictions, in an attempt to facilitate assessment of the model predictive accuracy.

With this model, the Mean Squared Error, or MSE, is 1,956,240,657,808.6606, and the R-squared Score is 0.6130. This high value of MSE infers that the target variable, housing prices, is in a large scale and has large deviations between actual and predicted values. This $R^2$ score infers that about 61.30% of the variability in housing prices is explained by this model, and the remaining 38.70% is still not captured. This postulates fair predictive power; the underlying pattern is caught somehow by the model, but the lack of critical variables or its complexity to represent the factors of house pricing.




A few things could be done to enhance model performance: feature engineering may introduce new variables or transform existing ones in ways that better bring out underlying patterns. This could mean adding interaction terms or polynomial features to model a non-linear relationship. However, changing the model complexity by the use of different parameters-for instance, in k-nearest neighbors-may let it capture a better pattern. Yet again, we need to remember the increasing risk of falling into overfitting problems, which may be tamed by the application of cross-validation techniques. Other modeling techniques, such as Random Forests and Gradient Boosting Machines, may give better results since they are assembled models that combine many decision trees in an attempt to lower their variance. Collection of more data would give the model more information from which to generalize. Multicollinearity-if the model is suffering from it-may also impact performance adversely and may be looked into by using such techniques as VIF analysis.

It follows then that the K-Nearest Neighbors Regressor had a moderate level of success in its prediction of house prices, catching some of the variability but leaving much of it unaccounted for. Hence, much better predictive performance of the models for more realistic predictions can be expected by tuning features, model parameters, more advanced algorithms, and data quality.

**Results Comparison**

A comparison of the three regression models fitted in this housing price prediction problem revealed that the best performance, in terms of a low Mean Squared Error and high R-squared, went to the Gradient Boosting Regressor at about 1.69 trillion and 0.6650, respectively, hence explaining about 66.50% of the variance in housing prices. The K-Nearest Neighbors (KNN) Regressor took the second best with an MSE approximated at 1.96 trillion and an $R^2$ score of 0.6130, hence 61.30 percent explained variance. The Decision Tree Regressor recorded the highest value of MSE at about 2.70 trillion and an $R^2$ score of 0.4656; hence, it explained 46.5%.

These results show that for this dataset, Gradient Boosting Regressor does better in predictive accuracy by building an ensemble of trees in sequence to correct the errors of its predecessors. On

the other hand, KNN Regressor depends on the average of the nearest neighbors, doing fairly well but coming in behind the first one. And lastly, the Decision Tree Regressor with the poorest performance, potentially due to its tendency to overfit or underfit the data depending on its depth.

Therefore, among the three models, Gradient Boosting Regressor is the most appropriate for the task of housing price prediction in the setting at hand.

**Conclusion**

The analysis showed that Gradient Boosting Regression outperformed Decision Tree and KNN in predicting housing prices by having the highest R-squared score and lowest Mean Squared Error. Its ensemble-based approach worked quite well in capturing complex patterns in the data. KNN showed moderate performance, while Decision Tree Regression trailed, probably due to the features of over-fitting or not enough complexity. The results show how important feature engineering, hyperparameter tuning, and algorithm selection are to achieve accurate predictions and obtain a meaningful data analysis . More advanced techniques, such as XGBoost or LightGBM, can be tried in the future, and larger datasets can be used to improve the generalizability of the model. This study has shown the potential of machine learning methods for improving housing price prediction accuracy.

## Bibliography


[1] I. D. Mienye and N. Jere, "A Survey of Decision Trees: Concepts, Algorithms, and Applications," *IEEE Access,* vol. 12, 2024.

[2] L. Breiman, "Bagging predictors," *Mach Learn,* vol. 24, p. 123–140, 1996.

[3] S. Pathak, I. Mishra and A. Swetapadma, "An Assessment of Decision Tree based Classification and Regression Algorithms," in *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, 2018, pp. 92-95.

[4] R. Zemel and T. Pitassi, "A gradient-based boosting algorithm for regression problems," *Advances in neural information processing systems,* vol. 13, 2000.

[5] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics,* pp. 1189--1232, 2001.

[6] Z. He, D. Lin, T. Lau and M. Wu, "Gradient Boosting Machine: A Survey," *CoRR,* vol. abs/1908.06951, 2019.

[7] C. Bentéjac, A. Csörgő and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review,* vol. 54, pp. 1937--1967, 2021.

[8] K. Taunk, S. De, S. Verma and A. Swetapadma, "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, 2019, pp. 1255-1260.

[9] R. Halder, M. Uddin and M. Uddin, "Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications.," *J Big Data,* vol. 11, p. 113 , 2024.

[10] A. Gupta, R. Joshi, N. Kanvinde, P. Gerela and R. M. Laban, "Metric Effects based on Fluctuations in values of k in Nearest Neighbor," *CoRR,* vol. abs/2208.11540, 2022.

[11] "Pandas," [Online]. Available: https://pandas.pydata.org/.