

**Machine Learning - Project Report Document**

|  |  |
| --- | --- |
| **Student Name** | Dawa Phuti Lepcha |
| **Batch** | AI Elite 18 |
| **Project Name** | Price Prediction for Apartments |
| **Project Domain** | Real Estate |
| **Type of Machine Learning** | Supervised ML |
| **Type of Problem** | Regression |
| **Project Methodology** | CRISP-DM |
| **Stages Involved** | * Data Collection and Understanding * Data Preparation * Model Building * Model Training * Model Evaluation |

**Business Understanding**

In the real estate business, properties such as land, buildings, and other structures are bought, sold, rented, or leased. This industry encompasses residential, commercial, and industrial sectors. Key aspects include property valuation, market analysis, financing, property management, and regulatory compliance. Real estate is influenced by economic factors, interest rates, and market trends, and it requires understanding customer needs, investment strategies, and location dynamics to successfully operate and generate profits.

**Problem Statement**

Buying or selling an apartment is a big decision, and knowing the right price can make a huge difference. It's well known that house prices depend on many factors such as location, size, number of rooms, nearby facilities, etc. Therefore, buyers, sellers, and investors make better choices if house prices are predicted accurately.

**Objective**

The objective is to develop a predictive model that estimates the prices of apartments based on a set of features such as size, location, number of BHK, etc.

**Stage 1: Data Collection and Understanding**

1. **Data Collection:** The dataset was taken from the Kaggle. It was originally scraped from the “MagicBricks” website.

Dataset link: <https://www.kaggle.com/datasets/juhibhojani/house-price/data>

1. **Data Understanding:** This dataset contains detailed descriptions of flats listed for sale on the "MagicBricks" website. It includes various apartment details from different cities across India.

Here are the features and their descriptions after cleaning the dataset:

1. **Description**: It is a test-based column, it describes the apartment in a paragraph.
2. **Location**: It gives the city where the apartment is located.
3. **Transaction**: A categorical feature that gives the different transaction types. It has three categories in the dataset: Resale, New Property, and Other, where Other consists of all the other Transaction types since they are in the   
   minority.
4. **Furnishing**: It gives the type of furnishing of the apartment. Unfurnished, Furnished or Semi-Furnished.
5. **Facing**: It gives the direction where the flat is facing. It is also a categorical column.
6. **Overlooking**: It gives the overlook that the flat provides. It is also a categorical column.
7. **Society**: Every apartment is either a part of society/community or is not a part of society. This feature provides the name of the Society the flat belongs to. If the flat is not a part of society, then it is a Standalone Building
8. **Bathroom**: It gives the count of Bathroom in the flat up to 10.
9. **Balcony**: It gives the count of Balcony in the flat up to 10.
10. **Ownership**: It gives the ownership types for the flats. They are Freehold, Leasehold, Co-operative Society, and Power of Attorney.
11. **BHK**: It gives the count of BHK in the flat up to 10.
12. **Amount**: It gives the price of the flat to buy. It is in Lakhs.
13. **Area**: It gives the area that the flat covers.
14. **Type of Car Parking**: It gives the type of Car Parking spot the flat provides. Either Open or Covered. If not provided, then it is represented by Not\_Available.
15. **No of Car Parking**: It gives the count of Car Parking spots provided by the flat.
16. **Sale Floor**: It gives the floor number that is being sold. Here, -2 means lower basement, -1 means upper basement, 0 means Ground floor and other values start from 1 up to 200.
17. **Total Floors**: It gives the total number of floors in the building in which a flat is being bought

|  |  |  |
| --- | --- | --- |
| **S No** | **Feature Name** | **Data Type** |
| 1 | Description | Object |
| 2 | Location | Object |
| 3 | Transaction | Object |
| 4 | Furnishing | Object |
| 5 | Facing | Object |
| 6 | Overlooking | Object |
| 7 | Society | Object |
| 8 | Bathroom | Int64 |
| 9 | Balcony | Int64 |
| 10 | Ownership | Object |
| 11 | BHK | Int64 |
| 12 | Amount | Float64 |
| 13 | Area | Float64 |
| 14 | Type of Car Parking | Object |
| 15 | No of Car Parking | Int64 |
| 16 | Sale Floor | Int64 |
| 17 | Total Floors | Int64 |

**Stage 2: Data Preparation**

**a) Exploratory Data Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S No** | **Type** | **Feature Names** | **Observation** |
| 1 | Missing Values | Description, Transaction, Furnishing, BHK, Amount, Area, Sale Floor, Total Floors | All these columns had  less than 5% null values. |
| 2 | Duplicates | All | There were almost 64% duplicate data. |
| 3 | Outliers | Amount, Area | These two columns had few extreme outliers. |

**b) Data Cleaning/wrangling:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S no** | **Type of Cleaning** | **Technique** | **Feature Name** | **Reason** |
| 1 | Missing value | Drop | Description, Transaction, Furnishing, BHK, Amount, Area, Sale Floor, Total Floors | Dropped to maintain the integrity of core values, as the dataset was large enough. |
| 2 | Duplicates | Drop | All | They are unnecessary. |
| 3 | Non-essential features | Drop | Index, Dimensions, Plot Area | They do not provide any useful information. |
| 4 | Encoding | OrdinalEncoder | Location, Transaction, Furnishing, Facing, Overlooking, Ownership, Type of Car Parking | Used OrdinalEncoder to encode Categorical Column. |
| 5 | Scaling | RobustScaler | Bathroom, Balcony, BHK, Amount, No of Car Parking, Area, Sale Floor, Total Floor | Used RobustScaler to scale the numerical data, as there were outliers. |

**c)Feature Selection:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S No** | **Removed Feature** | **Reason** | **Test Performed** |
| 1. | Description | Excluded this feature from the model since the description it gives is already in the dataset as a numerical and categorical column. | NA |
| 2. | Society | Excluded this feature from the model since it had numerous categories and had limited relevance for individuals seeking approximate price estimates. | NA |

**Stage 3: Model Building:**

|  |  |  |
| --- | --- | --- |
| **S No** | **Type of Problem** | **Algorithm Name** |
| 1 | Regression | KNeighborsRegressor |
| 2 | Regression | LinearRegression |
| 3 | Regression | SVR |
| 4 | Regression | RANSACRegressor |
| 5 | Regression | DecisionTreeRegressor |
| 6 | Regression | GradientBoostingRegressor |
| 7 | Regression | TheilSenRegressor |
| 8 | Regression | HuberRegressor |
| 9 | Regression | RandomForestRegressor |
| 10 | Regression | Ridge |
| 11 | Regression | Lasso |
| 12 | Regression | ElasticNet |

* + - 1. **KNeighborsRegressor:** A type of machine learning algorithm used for regression tasks, where the goal is to predict a continuous output variable based on input features. The k-NN algorithm works by identifying the 'k' nearest data points in the training set to the input point and then averaging their values to make a prediction.
      2. **LinearRegression:** A statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. The goal is to find the best-fitting straight line, represented by the equation.
      3. **SVR**: type of Support Vector Machine (SVM) used for regression tasks, which aims to find a function that deviates from the actual observed values by a value no greater than a specified margin. SVR uses kernel functions to handle non-linear relationships by mapping input features into high-dimensional spaces, providing flexibility and robustness in modelling complex data patterns.
      4. **RANSACRegressor:** n iterative algorithm used to fit a model to a dataset that may contain outliers. It works by repeatedly selecting random subsets of the data, fitting a model to these subsets, and then determining the number of inliers that fit this model within a certain tolerance. The model with the highest number of inliers is chosen as the final model, making RANSAC robust to outliers in the data.
      5. **DecisionTreeRegressor**: a machine learning model that predicts continuous values by splitting the data into subsets based on feature values, forming a tree structure. Each internal node represents a decision based on a feature, and each leaf node represents a predicted value. The model recursively partitions the data to minimize the variance within each subset, making it intuitive and capable of capturing non-linear relationships.
      6. **GradientBoostingRegressor**: an ensemble learning method that builds a series of decision trees, where each tree corrects the errors of the previous ones. The trees are added sequentially, and each one is trained to minimize the residual errors of the combined ensemble's predictions. This approach effectively improves accuracy and reduces overfitting, making Gradient Boosting a powerful tool for predictive tasks.
      7. **Random Forest Regressor**: an ensemble learning method that constructs multiple decision trees during training and outputs the average prediction of the individual trees. Each tree in the forest is built on a random subset of the data and features, which helps to reduce overfitting and improve generalization. This approach combines the strengths of multiple trees to produce more accurate and stable predictions.
      8. **TheilSenRegressor**: A robust regression technique that computes the median of all possible slopes between pairs of data points. It is less sensitive to outliers compared to ordinary least squares regression, making it useful for datasets with significant noise.
      9. **Huber Regressor**: A robust regression method that combines the properties of linear regression and robust estimation. It minimizes a loss function that is quadratic for small errors and linear for large errors, reducing the influence of outliers while maintaining efficiency for normally distributed data.
      10. **Ridge Regression**: It adds a penalty equal to the square of the magnitude of coefficients to the loss function. This regularization technique helps to prevent overfitting by shrinking the coefficients, particularly useful when dealing with multicollinearity in the data**.**
      11. **Lasso Regression**: Lasso (Least Absolute Shrinkage and Selection Operator) Regression adds a penalty equal to the absolute value of the magnitude of coefficients to the loss function. This encourages sparsity in the model by shrinking some coefficients to zero, effectively performing feature selection.
      12. **Elastic Net**: Elastic Net combines the penalties of both Ridge and Lasso regression. It adds both the L1 (absolute value) and L2 (squared value) penalties to the loss function, providing a balance between Ridge's smoothness and Lasso's sparsity. It is particularly useful when dealing with highly correlated features.

**Stage 4: Model Training:**

Splitting the dataset into training and testing sets with 85:15 Ratio.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S No** | **Algorithm Name** | **Mean Squared Error** | **Mean Absolute Error** | **Root Mean Squared Error** | **r-squared Error** | **Adjusted r-squared** |
| 1 | DecisionTreeRegressor | 2.383506 | 0.130467 | 1.543861 | 0.999892 | 0.999892 |
| 2 | KNeighborsRegressor | 4987.383825 | 25.241719 | 70.621412 | 0.774589 | 0.774237 |
| 3 | SVR | 18327.461727 | 48.400631 | 135.378956 | 0.171668 | 0.170374 |
| 4 | LinearRegression | 12415.385237 | 51.247661 | 111.424348 | 0.438872 | 0.437995 |
| 5 | TheilSenRegressor | 368718.853199 | 76.261571 | 111.424348 | -15.664690 | -15.690729 |
| 6 | HuberRegressor | 14152.330738 | 44.175670 | 118.963569 | 0.360368 | 0.359369 |
| 7 | RANSACRegressor | 55814.273630 | 67.626729 | 236.250447 | -1.522593 | - 1.526535 |
| 8 | GradientBoostingRegressor | 4984.261242 | 32.386188 | 70.599301 | 0.774730 | 0.774378 |
| 9 | RandomForestRegressor | 865.017401 | 10.034064 | 29.411178 | 0.960905 | 0.960843 |
| 10 | Ridge | 12415.385242 | 51.247456 | 111.424348 | 0.438872 | 0.437995 |
| 11 | Lasso | 12441.357490 | 51.064840 | 111.540833 | 0.437698 | 0.436819 |
| 12 | ElasticNet | 12998.748277 | 49.840567 | 114.012053 | 0.412506 | 0.411588 |

**Stage 5: Model Evaluation:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S No** | **Algorithm Name** | **Training Time** | **Predicting Time** | **Mean Squared Error** | **Mean Absolute Error** | **Root Mean Squared Error** | **r-squared Error** | **Adjusted r-squared** |
| 1 | DecisionTreeRegressor | 0.582244 | 0.027884 | 8459.977702 | 35.436810 | 91.978137 | 0.604592 | 0.603974 |
| 2 | KNeighborsRegressor | 0.192670 | 1.818614 | 6883.298237 | 30.810188 | 82.965645 | 0.678284 | 0.677781 |
| 3 | LinearRegression | 0.065665 | 0.004901 | 12801.388275 | 51.397729 | 113.143220 | 0.401680 | 0.400745 |
| 4 | HuberRegressor | 3.776718 | 0.002660 | 14098.006126 | 44.182237 | 118.735025 | 0.341077 | 0.340048 |
| 5 | GradientBoostingRegressor | 6.611134 | 0.014052 | 7085.551404 | 33.875289 | 84.175717 | 0.668830 | 0.668313 |
| 6 | RandomForestRegressor | 28.774948 | 0.385829 | 5556.702941 | 27.343809 | 74.543296 | 0.740287 | 0.739881 |
| 7 | Ridge | 0.014071 | 0.002679 | 12801.366993 | 51.397524 | 113.143126 | 0.401681 | 0.400746 |
| 8 | Lasso | 0.038083 | 0.004937 | 12797.951208 | 51.243515 | 113.128030 | 0.401840 | 0.400906 |
| 9 | ElasticNet | 0.027884 | 0.003003 | 13098.589097 | 49.834915 | 114.449068 | 0.387789 | 0.386832 |

**Observation:**

* From the above training results, we can observe that Decision Tree, KNN, Gradient Boosting, and Random Forest Regressor have learned well since they have more than 0.77 adjusted r-squared values.
* But when it came to testing, we could see that Decision Tree and Random Forest overfitted. KNN and Gradient Boosting Regressor were close to best fit.
* Moreover, the predicting time for KNN was higher than the Gradient Boosting Regressor.
* Therefore, Cross Validation to find the best fit was conducted for the Gradient Boosting Regressor.

**Conclusion:**

* The best-fit model with less prediction time was Gradient Boosting regression with max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=6, learning\_rate=0.2, loss= huber, and aloha=0.75.
* This model had the adjusted r-squared equal to 0.742769 for training and 0.669193 for testing.
* This model was chosen for deployment via Streamlit.

**Challenges Faced:**

* There was a lot of cleaning required for this dataset. And using cross-validations to find the best fit took a lot of time.