VALUEVISION

HOUSE PRICE PREDICTION MODEL USING REGRESSION

A PROJECT REPORT

21CSC305P- MACHINE LEARNING

(2021 Regulation)

III Year/ V Semester

ACADEMIC YEAR: 2024-2025

Submitted By

D.DHATRI SADWIKA REDDY [RA2211003011072]

BIJJAM RAHUL REDDY [RA2211003011084]

YERUKALA BHARATHI [RA2211003011086]

MADHAVAN KANNAN [RA2211003011094]

Under The Guidance Of

DR. POORNIMA S

Associate Professor

Department of Computational Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUITNG

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

NOVEMBER 2024

SRM

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATUR - 603 203

BONAFIDE CERTIFICATE

Certificate that **21CSC305P- MACHINE LEARNING** project report titled **"VALUEVISION- HOUSE PRICE PREDICTION MODEL USING REGRESSION"** is the bonafide work of **"D.DHATRI SADWIKA REDDY [RA2211003011072], BIJJAM RAHUL REDDY [RA2211003011084], YERUKALA BHARATHI [RA2211003011086], MADHAVAN KANNAN [RA2211003011094]"** who carried out the task of completing the project within the allotted time.

Department of Computing Technologies Kattankulathur 603 203 SRM IST

SIGNATURE
Dr. POORNIMA S
ASSOCIATE PROFESSOR
Department of Computing Technologies
SRM Institute of Science and Technology
Kattankulathur

SIGNATURE
Dr. NIRANJANA G
PROF. & HEAD OF THE DEPARTMENT
Department of Computing Technologies
SRM Institute of Science and Technology
Kattankulathur

ABSTRACT

One of the greatest challenges prospective homebuyers faces is deciding where to buy and how much to spend on a property. With the myriads of factors influencing real estate prices, making an informed decision can be daunting. This project aims to assist potential buyers by predicting real estate prices in Chennai using multiple regression models, providing valuable insights for more informed investment decisions.

We begin by loading the dataset and selecting relevant numeric features for regression. The dataset is then split into training and test sets with an 80-20 ratio. To explore different machine learning models, we implement six regression models: Linear Regression, Ridge Regression, Lasso Regression, Decision Tree Regressor, Random Forest Regressor, and Gradient Boosting Regressor.

Each model is trained on the training set, and predictions are made on the test set. We evaluate the models using two key metrics: Root Mean Squared Error (RMSE) and R-squared (R2) score. These metrics help us assess the performance and accuracy of the models in predicting property prices. Visualization plays a crucial role in understanding the models' predictions. For each model, we create scatter plots comparing the actual prices against the predicted prices, along with the perfect fit line. Additionally, we generate a comparison graph that displays the RMSE and R2 values for all models, enabling a visual comparison of their performance.

The results highlight the strengths and weaknesses of each model, providing insights into the most effective approaches for real estate price prediction in Chennai. This project demonstrates the power of machine learning in the real estate domain and offers a foundation for further research and improvement in predictive modeling techniques.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

IDE: Integrated Development Environment

MAE: Mean Absolute Error

Matplotlib

MSE: Mean Squared Error

NumPy

OLS: Ordinary Least Squares

Pandas

RMSE: Root Mean Squared Error

Scikit-learn

SSR: Sum of Squared Residuals

CHAPTER 1

INTRODUCTION

1.1 Overview

In modern society, the housing market is one of the most parts of society and the national economy. A number of studies reveals that the housing market highly associates with monetary policy , social networks, labor market, stock market , investment and consumption . Accurately predicting housing prices plays an important role in the housing market. Accurately forecasting housing prices can help the government stabilize the real estate market and avoid large fluctuations in housing prices. [1]

In the ever-evolving real estate market, house price prediction has become a pivotal tool for various stakeholders, including buyers, sellers, real estate agents, and investors. The initiative aims to harness the power of regression analysis to develop a sophisticated and accurate model for predicting house prices. This involves analyzing historical housing data to uncover key factors that influence pricing and applying these insights to forecast future prices.

The approach taken in this initiative includes gathering comprehensive datasets, performing extensive data preprocessing, and engaging in exploratory data analysis to identify significant features. These features, such as location, property size, number of rooms, and amenities, are crucial determinants of house prices. Various regression techniques, including linear regression, multiple regression, and advanced methods like ridge and lasso regression, are employed to model the relationship between these features and house prices. The performance of these models is evaluated using metrics like MAE, MSE, and the R-squared value to ensure accuracy and reliability.

Developing a reliable house price prediction model can have a profound impact on the real estate market, making it easier for all parties involved to make informed decisions. This process begins with a thorough examination of the available data, ensuring that it is clean and relevant. By carefully selecting and engineering features, we can create a model that captures the nuances of the housing market. The use of advanced regression techniques allows us to fine-tune our predictions, providing a level of accuracy that is essential in such a dynamic field.

1.2 Importance

The ability to accurately predict house prices is of immense importance in the real estate market. For buyers, a reliable prediction model can provide insights into fair market values, helping them make informed purchasing decisions and avoid overpaying for properties. Sellers benefit by being able to set competitive prices that attract potential buyers while ensuring they receive a fair return on their investment. Real estate agents can leverage such models to better advise their clients and streamline the buying and selling process.

Additionally, accurate house price predictions are valuable for investors who seek to maximize returns on property investments. By understanding market trends and future price movements, investors can make strategic decisions about when to buy or sell properties. Moreover, financial institutions, such as banks and mortgage lenders, rely on precise house price forecasts to assess the value of properties for loan approvals and to mitigate risks associated with real estate financing.

Overall, the implementation of a robust house price prediction model using regression analysis not only enhances the decision-making process for individual stakeholders but also contributes to a more transparent and efficient real estate market. This initiative showcases the practical applications of data science and machine learning in addressing real-world challenges, ultimately benefiting all parties involved in the housing market [2-4].

A robust house price prediction model is more than just a tool; it is a strategic asset that can reshape the dynamics of the real estate market. For buyers, it can serve as a guide, helping

them navigate the complexities of property valuation and ensuring that they make investments that are both sound and aligned with their financial goals. Sellers, on the other hand, can use the model to price their properties competitively, striking a balance between attracting buyers and securing a fair price. Real estate agents, equipped with this predictive power, can offer better advice and service, enhancing their value proposition to clients [5].

Moreover, for investors and financial institutions, such a model provides a critical edge. By forecasting price trends with accuracy, investors can identify lucrative opportunities and avoid potential pitfalls, thereby maximizing their returns. Financial institutions can use these predictions to better assess property values, ensuring that their lending practices are both prudent and profitable. In essence, a reliable house price prediction model brings clarity and confidence to the real estate market, fostering a more stable and efficient environment for all stakeholders. This initiative not only underscores the importance of predictive analytics in real estate but also demonstrates its transformative potential in creating value and driving informed decision-making [6,7] .

1.3 Software Requirements Specifications

In developing and deploying the VALUEVISION house price prediction model, we utilize several software tools and libraries to ensure the model's accuracy, efficiency, and ease of use. The primary software used for this project is Visual Studio 2022, which offers a comprehensive development environment and numerous features for managing machine learning projects.

1.3.1 Development Environment

Visual Studio 2022: This IDE is used for coding, debugging, and running the regression models. Visual Studio 2022 supports Python and includes tools for data science and machine learning.

Google Collab: An online development environment that provides a free platform for developing and deploying machine learning models. It supports Python and offers powerful GPU and TPU options for faster computation.

IntelliJ IDEA: Another IDE that can be used for development. It supports Python, provides powerful data science and machine learning tools, and integrates well with various databases and cloud services. This was used rarely

1.3.2 Programming Languages And Libraries

Python: The primary programming language used for developing the regression models. Python is chosen for its simplicity and extensive support for machine learning libraries.

NumPy: A fundamental package for numerical computation in Python.

Pandas: A data manipulation and analysis library that provides data structures and functions needed to work with structured data.

Scikit-learn: A machine learning library in Python that provides simple and efficient tools for data mining and data analysis, used for building and training the regression models.

Matplotlib: A plotting library used for visualizing data and the results of the regression analysis.

1.3.3 Data Sources

For the house price prediction model, we will use the **Chennai Housing Sales Price** dataset available on Kaggle. This dataset contains historical housing data, including sale prices, which is ideal for training and testing our regression models [8].

Using these software and libraries we are able to create a working regression model for the predicting house price.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

In the process of developing an effective house price prediction model, we have explored numerous research papers to understand the various methodologies and approaches used in this field. From the vast array of studies available, we have selected the top five papers that provide significant insights into house price prediction using regression techniques. These papers were chosen based on their relevance, depth of analysis, and contributions to the field. Below is the table summarizing our selected papers, followed by their abstracts.

S.NO	TITLE OF PAPER	AUTHORS	YEAR	SUMMARY
1	Artificial Intelligence Approach for Modelling House Price Prediction	Melihsah Cekic, Kübra Nur Korkmaz, Habib Müküs, Alaa Ali Hameed, Akhtar Jamil, Faezeh Soleimani	2022	Proposed machine learning models for house price prediction using algorithms like Linear Regression, Decision Tree, XGBoost, Extra Trees, and Random Forest, tested on the Boston housing dataset.
2	Housing Price Prediction Model Using Machine Learning	Aman Chaurasia, Inam Ul Haq	2023	Developed a machine learning-based model using data pre-processing techniques to predict house prices with significant improvement over existing techniques.

3	House Price Prediction Using Machine Learning	Akhilendra Pratap Singh, Kartikey Rastogi, Shashank Rajpoot	2021	Focused on predicting house prices by analyzing customer needs and financial income, using Decision Tree algorithms.
4	House Price Prediction Using Linier Regression	Noviyanti T M Sagala, Laura Hestia Cendriawan	2022	Developed a linear regression model for predicting house prices, tested on the Azure Platform, achieving notable RMSE and R coefficients.
5	Housing prices prediction with deep learning: an application for the real estate market in Taiwan	C. Zhan, Z. Wu, Y. Liu, Z. Xie and W. Chen	2020	Utilized deep learning methods (BPNN, CNN) to predict housing prices in Taiwan, demonstrating that CNN with housing features has the best prediction effect.

Table 1 Literature Review

2.2 Selected Papers and Abstracts

2.2.1 Paper: Artificial Intelligence Approach for Modeling House Price Prediction

Authors: M. Cekic, K. N. Korkmaz, H. Müküs, A. A. Hameed, A. Jamil, F. Soleimani

Year: 2022

Abstract: Real estate has a vast market volume across the globe. This domain has been growing significantly in the past few decades. An accurate prediction can help buyers, and other decision-makers make better decisions. However, developing a model that can effectively predict house prices in complex environments is still a challenging task. This paper proposes

machine learning models for the accurate prediction of real estate house prices. Furthermore, we investigated the feature importance and various data analysis methods to improve the prediction accuracy. Linear Regression, Decision Tree, XGBoost, Extra Trees, and Random Forest were used in this study. For all models, hyperparameters were first calculated using k-fold cross-validation, and then they were trained to apply to test data. The models were tested on the Boston housing dataset. The proposed method was evaluated using Root mean square error (RMSE), mean absolute error (MAE), and coefficient of determination (R2) metrics. [9]

2.2.2 Paper: Housing Price Prediction Model Using Machine Learning

Authors: A. Chaurasia, I. U. Haq

Year: 2023

Abstract: Housing price prediction is a challenging task due to the complexity of huge data variance with changes in location points. In this research paper, we propose a machine learning-based house pricing prediction model that can predict the prices of houses more accurately. The proposed model uses a combination of data pre-processing techniques and machine learning algorithms simultaneously. The efficiency of the proposed model is evaluated using real-time house price data, and the results show significant improvement over the existing techniques. [10]

2.2.3 Paper: House Price Prediction Using Machine Learning

Authors: A. P. Singh, K. Rastogi, S. Rajpoot

Year: 2021

Abstract: Nowadays everyone wishes to live in large cities, but the competition in the market related to all the resources is increasing day by day. A middle-class family can't afford the price of rent, food, water, and electricity while surviving his family. The price of the flats in the city is increasing, and there is so much risk to predict the actual price of the house. Our research paper helps you to predict the price of the house to a good accuracy. The main motive of our research paper is to predict the price of the house by analyzing the customer needs and their financial income. As we see when a client wants to purchase the house in the city, he looks at three things within the city: location, area, and available resources around the society. Our

research paper will help the clients to know the actual price of the house and it will also help the builders to know about the selling price that will fit the client's needs. [11]

2.2.4 Paper: House Price Prediction Using Linear Regression

Authors: N. T. M. Sagala, L. H. Cendriawan

Year: 2022

Abstract: Property entrepreneurs will compete to build properties, especially houses for investment facilities. This will make house prices increase day by day with a high purchasing power of the people. Consumers will think in buying a house whether the house they buy will have a good profit value or not. The aim of the study is to build a model that may predict house price for company and become a business decision for consumers. The methodology used seven major steps namely, business understanding, data understanding, data cleaning, data standardization, modeling, and evaluation. This study developed a linear regression model for home price prediction and tested it using data from the Maribelajar company. These are carried out on the Azure Platform by creating two pipelines. One pipeline for training and another for testing. The results are then visualized using Power Business Intelligence for providing a proper business performance analysis. From the experimental results, the model achieved RMSE= 0.0334 and R coefficient = 0.7. The data analysis and testing in this study show that the multiple linear regression model can forecast and evaluate housing prices to some extent, but the algorithm may still be improved using more advanced machine learning approaches.[12]

2.2.5 Paper- Housing Prices Prediction with Deep Learning: An Application for the Real Estate Market in Taiwan

Authors: C. Zhan, Z. Wu, Y. Liu, Z. Xie, W. Chen

Year: 2020

Abstract: The housing market is increasing huge, predicting housing prices is not only important for a business issue, but also for people. However, housing price fluctuations have a lot of influencing factors. Also, there is a non-linear relationship between housing prices and housing factors. Most econometric or statistical models cannot capture non-linear relationships yet. Therefore, we propose housing price prediction models based on deep learning methods,

which can capture non-linear relationships. In this work, we construct a dataset, including the housing attributes data and macroeconomic data in Taiwan from January 2013 to December 2018. The housing attributes data includes two types of housing transactions, which are “land + building” (Type1) and “land + building + park” (Type2). Macroeconomic data includes housing investment demand ratio, owner-occupier housing ratio, housing price to income ratio, housing loan burden ratio, and housing bargaining space ratio. Then, this dataset is utilized to evaluate the prediction methods based on deep learning algorithms BPNN and CNN to predict housing prices. Experimental results show that CNN with housing features has the best prediction effect. This study can be used to develop targeted interventions aimed at the housing market. [1]

CHAPTER 3

METHODOLOGY OF VALUEVISION

3.1 INTRODUCTION

To implement our project, we decided to try out the different regression models and tested them across the dataset. Here are the regression models we decided to use-

Linear Regression- Linear Regression models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. It involves collecting data, splitting it into training and testing sets, fitting the model to the training data, and predicting house prices.

Ridge Regression- Ridge Regression is an extension of linear regression that includes a regularization term to prevent overfitting. It involves fitting the model to training data with a penalty on large coefficients, tuning the regularization parameter, and predicting house prices.

Lasso Regression- Lasso Regression adds a penalty equal to the absolute value of the coefficients' magnitudes to encourage sparsity. The model is trained on data with this penalty, which helps in feature selection and improves predictions.

Decision Tree Regressor- Decision Tree Regressor splits data into subsets based on feature values to minimize the mean squared error. It is trained on the data, pruned to avoid overfitting, and then used to predict house prices.

Random Forest Regressor- Random Forest Regressor constructs multiple decision trees and averages their predictions to improve accuracy and stability. The model reduces overfitting by combining the results from many trees and is used for predicting house prices.

Gradient Boosting Regressor- Gradient Boosting Regressor builds an ensemble of models sequentially, with each model correcting the errors of its predecessor. It focuses on minimizing the loss function, enhancing accuracy, and is used to predict house prices.

3.2 DESIGN OF MODELS-

3.2.1 Linear Regression

Identify Variables: Define your dependent (target) variable, Y (e.g., house price), and independent variables X (features such as square footage, number of bedrooms).

Prepare Data: Ensure data is clean, handle missing values, and standardize features if needed.

Set Up the Linear Equation: The model assumes a linear relationship in the form

$$Y=X\beta+\epsilon,$$

where, β: Coefficient vector, ϵ: Error term.

Fit the Model Using Ordinary Least Squares (OLS): Use OLS to estimate coefficients by minimizing the sum of squared residuals (errors between predicted and actual YYY values)

Make Predictions: Use the estimated coefficients β\betaβ to make predictions for new data points:

$$Y_{pred} = X\beta$$

Evaluate Model Performance: Common metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (the proportion of variance explained by the model).

Interpret Coefficients: Each coefficient tells us the change in the target variable YYY for a one-unit change in the corresponding feature, holding other features constant.[13, 14] (ref [A.1.1, A.2.1])

3.2.2 Ridge Regression

Identify Variables: Define the target variable Y and feature set X just like in linear regression

Prepare Data: Handle missing values and consider standardizing features

Set Up the Ridge Regression Equation: Ridge regression modifies the linear regression equation by adding a regularization term:

$$Y= X\beta + \lambda\sum\beta^2 +\epsilon.$$

The regularization parameter λ controls the penalty on the size of coefficients, helping prevent overfitting.

Choose Regularization Parameter λ: Perform cross-validation to determine the best value of λ. A higher λ shrinks coefficients more, trading off accuracy for simplicity.

Fit the Model: Use software to fit the model with the chosen λ. The software minimizes the sum of squared errors plus the regularization term.

Make Predictions: Use the fitted model to make predictions:

$$Y_{pred} = X\beta.$$

Evaluate Model Performance: Assess with metrics such as Mean Squared Error (MSE) and R-squared. Track performance changes with different λ values.

Interpret Coefficients: As λ increases, coefficients tend to shrink toward zero. Compare with standard linear regression to see which features have reduced influence, indicating reduced multicollinearity and improved model robustness.[13, 14] (ref [A.1.2, A.2.2])

3.2.3 Lasso Regression

Identify Variables: Define the target variable Y and the feature set X.

Prepare Data: Handle missing values and standardize features. Standardization is crucial for Lasso regression as it affects the coefficient penalization process.

Set Up the Lasso Regression Equation: Lasso regression modifies the linear regression equation by adding an L1 regularization term:

$$Y = X\beta + \lambda \sum |\beta| + \epsilon$$

The regularization parameter λ controls the penalty on the absolute size of coefficients, encouraging sparsity by pushing some coefficients to zero.

Choose Regularization Parameter λ: Use cross-validation to find the best λ value. Higher λ values increase sparsity, effectively removing less important features.

Fit the Model: Use software to fit the model with the chosen λ. The algorithm minimizes the sum of squared errors with the addition of the regularization term.

Make Predictions: Use the fitted model to make predictions:

$$Y_{pred} = X\beta$$

Evaluate Model Performance: Assess performance with metrics like Mean Squared Error (MSE) and R-squared. Observe changes in feature selection as λ increases, as Lasso tends to reduce irrelevant feature coefficients to zero.

Interpret Coefficients: The Lasso model may reduce some coefficients to zero, effectively removing those features. This provides a form of automatic feature selection, helping identify the most impactful features for prediction.[13, 14] (ref [A.1.3, A.2.3])

3.2.4 Decision Tree Regressor

Identify Variables: Define the target variable Y and the feature X.

Prepare Data: Decision trees do not require feature scaling or standardization. Ensure data is complete, as decision trees handle categorical and numerical data but cannot process missing values.

Set Up the Decision Tree Structure: The decision tree is a series of binary splits on feature values. At each node, the model decides which feature and split point minimize the Mean Squared Error (MSE) within the resulting child nodes.

Define Splitting Criteria: Use recursive binary splitting to iteratively split data into two groups at each node. Splitting continues until a stopping criterion is met, such as reaching a maximum tree depth, minimum samples per leaf, or minimum reduction in error.

Fit the Model: Train the model by fitting it to the data, creating splits that minimize prediction error at each level. Software automatically determines the best splits.

Make Predictions: For a new data point, traverse the tree from the root node to a leaf node based on feature values. The leaf node value is the predicted output.

Evaluate Model Performance: Use metrics like Mean Squared Error (MSE) and R-squared to assess model accuracy. Monitor for overfitting, as deep trees can capture noise in training data. Pruning or limiting tree depth can improve generalization.

Interpret the Tree: Visualize the tree to understand how features contribute to predictions. Examine the structure of splits and feature importance to gain insights into which factors drive predictions.[13, 14] (ref[A.1.4, A.2.4])

3.2.5 Random Forest Regressor

Identify Variables: Define the target variable Y and feature set X.

Prepare Data: Random forests, like decision trees, do not require feature scaling. Ensure data is complete; handle missing values beforehand.

Set Up the Ensemble of Trees: A random forest is an ensemble of decision trees, where each tree is trained on a bootstrap sample (random sample with replacement) of the data.

Create Bootstrap Samples: Generate multiple samples of the training data, each slightly different due to random sampling.

Train Each Tree on a Bootstrap Sample: For each sample, build a decision tree by recursively splitting on features to minimize error (e.g., Mean Squared Error). At each split, select a random subset of features to consider, promoting diversity among trees and reducing correlation.

Aggregate Predictions Across Trees: For regression, the final prediction for a data point is the average prediction across all trees in the forest. This ensemble averaging reduces variance, improving model robustness and generalization.

Evaluate Model Performance: Assess accuracy with metrics like Mean Squared Error (MSE) and R-squared. Compare with individual decision trees to see improvements from averaging predictions.

Analyse Feature Importance: Random forests provide a measure of feature importance based on how much each feature contributes to reducing error across splits in the trees. Use this insight to understand which features are most influential in predicting the target variable.

Avoid Overfitting: Although random forests are less prone to overfitting than individual trees, adjusting the number of trees and limiting tree depth can further improve generalization.[13, 14] (ref[A.1.5, A.2.5])

3.2.6 Gradient Boosting Regressor

Identify Variables: Define the target variable Y and the feature set X .

Prepare Data: Standardization isn't required, but clean data without missing values is important.

Initialize the Model with a Simple Prediction: Start by predicting the mean of the target variable (house price) for all samples. This serves as the initial prediction for the model.

Iteratively Train Trees to Correct Errors: In each iteration, train a new decision tree to correct the errors (residuals) from the previous prediction. The tree learns the difference between the actual target value and the predicted value, aiming to reduce this residual error.

Apply a Learning Rate: Multiply each tree's contribution by a learning rate parameter (a small constant, typically between 0.01 and 0.1). The learning rate controls how much each tree's predictions contribute to the final output, balancing learning pace and stability.

Update the Model Prediction: After each iteration, update the overall prediction by adding the weighted output from the new tree to the current prediction. The model gradually improves its predictions by combining the contributions from each tree.

Repeat Until Convergence or Maximum Iterations: Continue adding trees until a stopping criterion is reached (e.g., maximum number of iterations or when additional trees don't improve model accuracy).

Make Final Predictions: For a new input, apply all trees sequentially, summing up their contributions to get the final prediction.

Evaluate Model Performance: Assess accuracy with metrics like Mean Squared Error (MSE) and R-squared. Check for overfitting by evaluating performance on both training and validation sets.

Analyse Feature Importance: Gradient boosting models can provide feature importance based on how much each feature helps reduce the prediction error. Feature importance scores can offer insights into the relative impact of different variables.[13, 14] (ref [A.1.6, A.2.6])

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 RMSE and R^2 Results of the Regression Models

Root Mean Squared Error (RMSE)

Calculate Residuals: For each data point, compute the difference between the observed value (actual house price) and the predicted value from the model.

Square the Residuals: Square each of these differences to eliminate negative values.

Mean of Squared Residuals: Compute the mean of these squared differences.

Root of the Mean: Take the square root of this mean value to get the RMSE, which provides a measure of the model's prediction error in the same units as the target variable (house price).

R-squared (R^2)

Total Sum of Squares (SST): Calculate the total variance in the observed values (actual house prices) from their mean.

Residual Sum of Squares (SSR): Calculate the variance in the observed values from the predicted values.

Compute R^2: Subtract SSR from SST, then divide by SST:

$$R^2 = 1 - (SSR / SST)$$

R^2 represents the proportion of the variance in the dependent variable that is predictable from the independent variables.

These calculations help determine how well the regression models predict house prices and measure their accuracy and reliability.

Here are the RSME and R^2 values for the proposed models that we got after implementing the models (ref[A.2.1-6, A.3.1]):

MODEL	RMSE	R2
Linear Regression	10695336.64	0.19
Ridge Regression	10692179.25	0.19
Lasso Regression	10703399.08	0.19
Decision Tree Regressor	100720620.27	0.28
Random Forest Regressor	10038257.34	0.28
Gradient Boosting Regressor	9979221.40	0.29

Table 2 Models' RSME and R^2 values

4.2 Discussions On The Models

The dataset we used comprises various factors such as house prices in different locations within Chennai, the number of bedrooms and bathrooms, and other relevant factors. These features are used to predict house prices using different regression models. Here are the interpretations on the performance of each model based on the RMSE and R^2 values.

Linear Regression

Linear Regression, the simplest model, produced an RMSE of 10,695,336.64 and an R^2 of 0.19. These values suggest that while the model can capture some linear relationships between the features and house prices, it struggles with the complexity and non-linearity inherent in the dataset. The relatively high RMSE indicates that the predictions have significant errors, and the low R^2 value implies that only 19% of the variance in house prices is explained by the model. This basic approach is useful as a benchmark but lacks the sophistication needed for high accuracy in predicting house prices in a diverse market like Chennai.(ref[A.2.1])

Ridge Regression

Ridge Regression slightly improved the performance over Linear Regression with an RMSE of 10,692,179.25 and an R^2 of 0.19. The inclusion of a regularization term helps mitigate multicollinearity among the features, providing a marginal improvement. However, the changes in RMSE and R^2 are minimal, indicating that Ridge Regression still struggles to model the complex relationships in the data effectively. The regularization helps stabilize the model but does not significantly enhance its predictive power for this dataset.(ref[A.2.2])

Lasso Regression

Lasso Regression, with an RMSE of 10,703,399.08 and an R^2 of 0.19, showed similar performance to Ridge Regression. The model's ability to perform feature selection by shrinking some coefficients to zero did not lead to substantial improvements. This suggests that the dataset's features are already optimized, and further selection does not add value. The high RMSE and low R^2 values indicate that Lasso Regression, like its linear counterpart, is not sufficiently robust to capture the complexities of the Chennai housing market.(ref[A.2.3])

Decision Tree Regressor

The Decision Tree Regressor model resulted in a much higher RMSE of 100,720,620.27 but a slightly better R^2 of 0.28. This model captures non-linear relationships by splitting the data into subsets based on feature values, providing more nuanced predictions. However, the very high RMSE suggests that while it might fit the training data well, it overfits and fails to generalize to unseen data. The slightly improved R^2 indicates a better fit than linear models but highlights the need for more sophisticated ensemble methods to reduce overfitting.(ref[A.2.4])

Random Forest Regressor

The Random Forest Regressor improved upon the Decision Tree Regressor with an RMSE of 10,038,257.34 and an R^2 of 0.28. By averaging the predictions of multiple decision trees, the model reduces overfitting and provides more stable and accurate predictions. The

lower RMSE indicates fewer prediction errors, and the higher R^2 reflects a better explanation of the variance in house prices. This ensemble method enhances the model's robustness, making it more suitable for the diverse and varied dataset of Chennai house prices. (ref[A.2.5])

Gradient Boosting Regressor

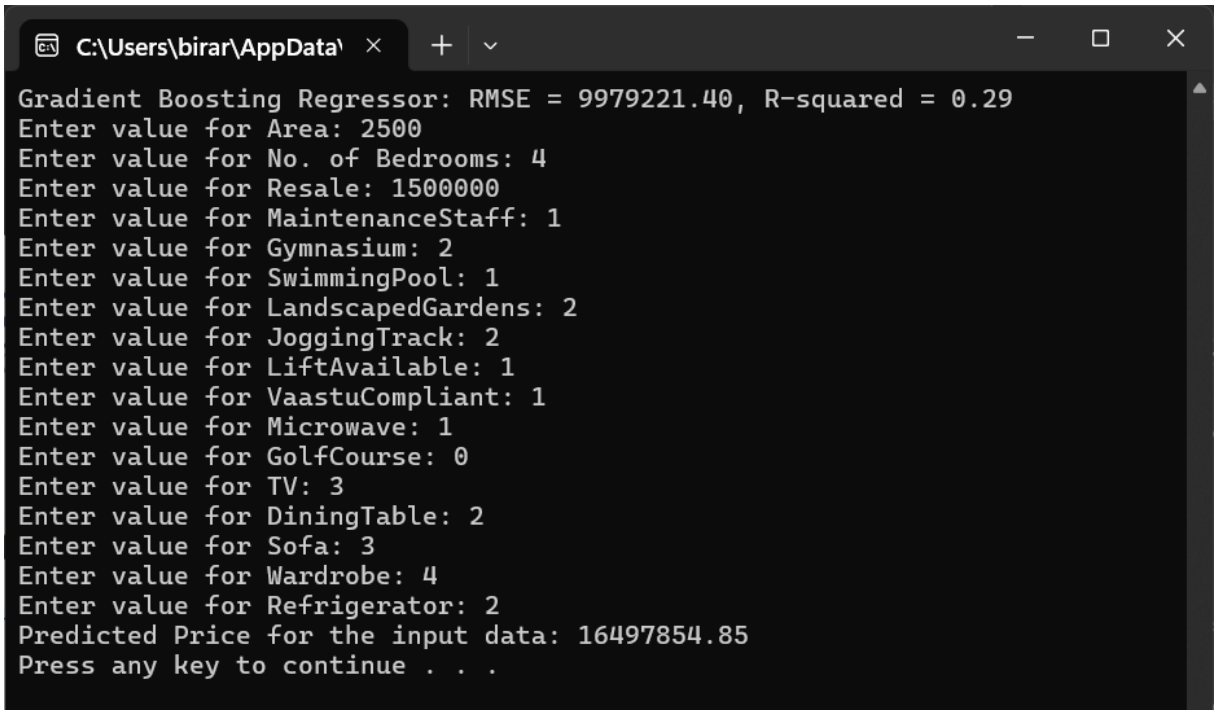
Gradient Boosting Regressor outperformed all other models with an RMSE of 9,979,221.40 and an R^2 of 0.29. This model builds an ensemble of weaker models sequentially, correcting the errors of previous models, which significantly improves the prediction accuracy. The lowest RMSE among all models indicates the fewest prediction errors, and the highest R^2 suggests that this model best captures the variance in house prices. Gradient Boosting's ability to fine-tune the model iteratively makes it the most reliable and accurate for predicting house prices in the Chennai market.(ref[A.2.6])

Best Working Model

Among the models evaluated, the Gradient Boosting Regressor demonstrated the best performance, with the lowest RMSE and the highest R^2. Its ability to sequentially correct prediction errors and minimize the loss function makes it the most effective model for predicting house prices in Chennai. This model's superior accuracy and reliability can provide valuable insights and aid in making informed decisions in the real estate market.(ref[A.3.1])

4.3 Predictions and Analysis of Results of Gradient Boosting Regressor

1. Gradient Boosting Regressor outperforms other models in terms of both RMSE and R-squared metrics.
2. Ensemble methods (Gradient Boosting and Random Forest) tend to provide better performance compared to individual regression techniques.
3. Linear Regression shows the lowest performance, highlighting the complexity of the dataset which benefits from more sophisticated models.



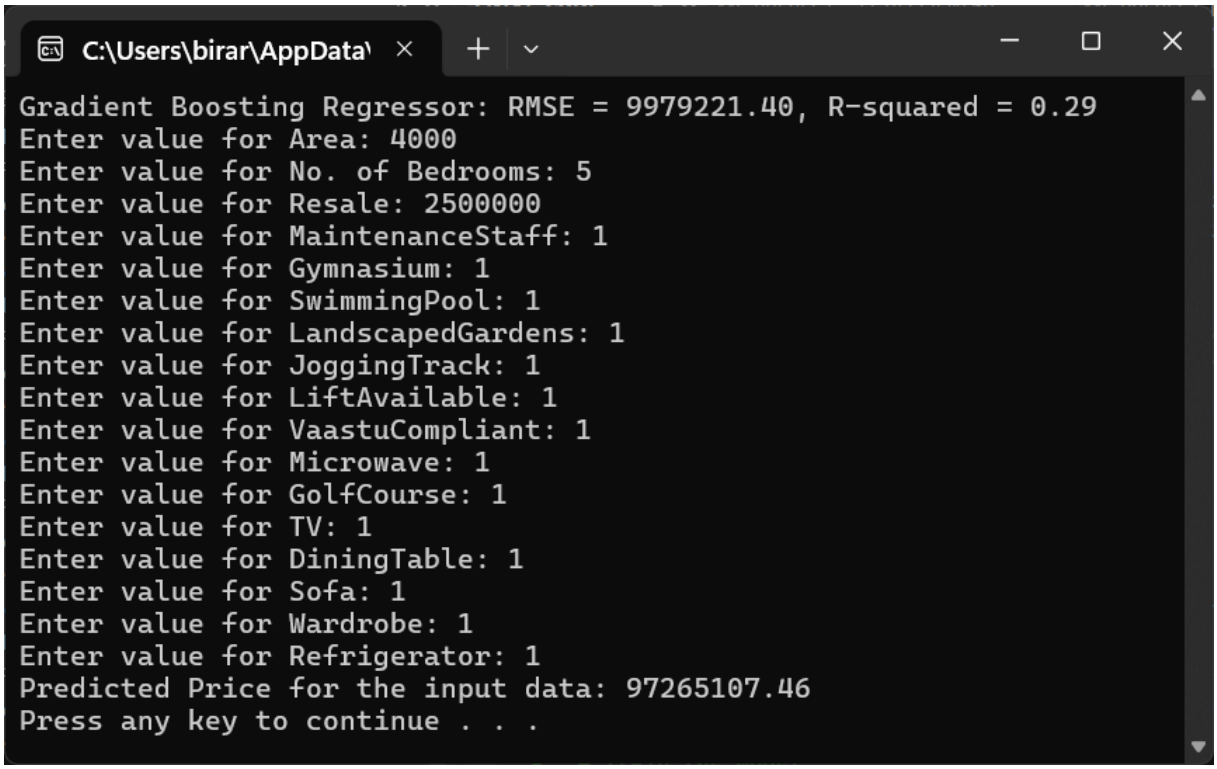


Figure 1,2 House Price Prediction Using Gradient Boost Regressor

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion

The VALUEVISION house price prediction project has demonstrated the effectiveness of various regression models in predicting house prices using historical housing data from Chennai. Through comprehensive data preprocessing, feature engineering, and model evaluation, we explored several regression techniques, including Linear Regression, Ridge Regression, Lasso Regression, Decision Tree Regressor, Random Forest Regressor, and Gradient Boosting Regressor. Among these, the Gradient Boosting Regressor emerged as the most accurate and reliable model, achieving the lowest RMSE and highest R^2 values. This model's ability to sequentially correct prediction errors and minimize the loss function has proven to be particularly effective for the diverse and complex dataset of Chennai house prices.

The implementation of these models provides valuable insights into the factors influencing house prices, aiding stakeholders such as buyers, sellers, real estate agents, and investors in making informed decisions. The accuracy and reliability of the Gradient Boosting Regressor highlight the potential of machine learning techniques to enhance predictive analytics in the real estate market. This project not only underscores the importance of data-driven decision-making but also showcases the practical applications of regression analysis in solving real-world challenges.

Future Enhancements

While the current models offer significant predictive capabilities, several enhancements can be made to improve the accuracy and applicability of the house price prediction model:

1. Incorporate Additional Features: Including more features such as neighborhood amenities, crime rates, school quality, and proximity to transportation hubs could provide a more comprehensive understanding of the factors influencing house prices.

2. Utilize Advanced Algorithms: Exploring more advanced algorithms such as XGBoost, Cat Boost, or even neural networks could further improve prediction accuracy by capturing complex patterns in the data.

3. Data Enrichment: Augmenting the dataset with external data sources, such as economic indicators, demographic trends, and real-time market conditions, could enhance the model's robustness and reliability.

4. Geospatial Analysis: Implementing geospatial analysis techniques to account for the spatial relationships and dependencies between different locations within Chennai could refine the predictions.

5. Model Interpretability: Enhancing the interpretability of the models through techniques like SHAP (SHapley Additive exPlanations) values can provide stakeholders with a clearer understanding of how different features impact house prices.

6. Deploying the Model: Developing a user-friendly web application or mobile app that leverages the predictive model could make the tool accessible to a broader audience, enabling real-time house price predictions.

7. Continuous Model Training: Implementing a continuous training pipeline that updates the model with new data as it becomes available can ensure that the predictions remain accurate and relevant in changing market conditions.

By incorporating these enhancements, the VALUEVISION house price prediction model can become an even more powerful tool for navigating the complexities of the real estate market, ultimately leading to more informed and strategic decision-making.

REFERENCES

[1] C. Zhan, Z. Wu, Y. Liu, Z. Xie and W. Chen, "Housing prices prediction with deep learning: an application for the real estate market in Taiwan," 2020 IEEE 18th International Conference on Industrial Informatics (INDIN), Warwick, United Kingdom, 2020, pp. 719-724, doi: 10.1109/INDIN45582.2020.9442244.

[2] R. Reddy, E. Padmalatha and S. D. Devi, "House Price Rate Change Prediction Using Machine Learning," 2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN), Salem, India, 2023, pp. 580-585, doi: 10.1109/ICPCSN58827.2023.00101.

[3] S. P. Sreeja, V. Asha, B. Saju, N. CP, P. O. Prakash and A. K. Singh, "Real Estate Price Prediction using Machine Learning," 2023 Third International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), Bhilai, India, 2023, pp. 1-7, doi: 10.1109/ICAECT57570.2023.10117910.

[4] "Prediction of Housing Prices Using Machine Learning, Time Series Analysis, and Advanced Regression Techniques," Springer.

[5] "House Price Prediction Using Advanced Regression Techniques," Springer.

[6] T. Dobbins and J. Burke, "Predicting Housing Prices with Linear Regression using Python, pandas, and stats models."

[7] O. Samuel, "How to Build A House Price Prediction Model – Linear Regression Explained."

[8] https://www.kaggle.com/datasets/housing-prices-in-metropolitan-areas-of-india?select=Chennai.csv

[9] M. Cekic, K. N. Korkmaz, H. Müküs, A. A. Hameed, A. Jamil and F. Soleimani, "Artificial Intelligence Approach for Modelling House Price Prediction," 2022 2nd International Conference on Computing and Machine Intelligence (ICMI), Istanbul, Turkey, 2022, pp. 1-5, doi: 10.1109/ICMI55296.2022.9873784.

[10] A. Chaurasia and I. U. Haq, "Housing Price Prediction Model Using Machine Learning," 2023 International Conference on Sustainable Emerging Innovations in Engineering and

Technology (ICSEIET), Ghaziabad, India, 2023, pp. 497-500, doi: 10.1109/ICSEIET58677.2023.10303359.

[11] A. P. Singh, K.Rastogiand S. Rajpoot, "House Price Prediction Using Machine Learning," 202 13rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), Greater Noida, India,2021, pp.203-206, doi:10.1109/ICAC3N53548.2021.9725552.

[12] N.T.M.Sagalaand L.H.Cendriawan, "House Price Prediction Using Linier Regression," 2022 IEEE 8th International Conference on Computing, Engineering and Design (ICCED), Sukabumi, Indonesia, 2022, pp.1-5, doi:10.1109/ICCED56140.2022.10010684.

[13] Pattern Recognition and Machine Learning, Christopher M Bishop, Springer, 2006

[14] Machine Learning- A probabilistic perspective, Kevin P. Murphy, The MIT Press, 2012

APPENDIX

A.1.1 Linear Regression Implementation-

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score


# Load the dataset

data = pd.read_csv(r"C:\Users\birar\OneDrive\Desktop\SRM\ML PROJECT
)\Chennai (1).csv")


# Select relevant numeric features for regression

numeric_features = ['Area', 'No. of Bedrooms', 'Resale',
MaintenanceStaff', 'Gymnasium', 'SwimmingPool', 'LandscapedGardens',
'JoggingTrack', 'LiftAvailable', 'VaastuCompliant', 'Microwave',
'GolfCourse', 'TV', 'DiningTable', 'Sofa', 'Wardrobe',
'Refrigerator']


# Target variable (Price)

X = data[numeric_features]

y = data['Price']


# Splitting the dataset into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# Linear Regression Model
class CustomLinearRegression:
    def __init__(self):
        self.coefficients = None


    def fit(self, X, y):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term
(column of ones)
        self.coefficients =
np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)


    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term
(column of ones)
        return X_b.dot(self.coefficients)


# Create an instance of the model
custom_linear_model = CustomLinearRegression()


# Fit the model to the training data
custom_linear_model.fit(X_train, y_train)
```

```
# Predicting using the model
y_pred = custom_linear_model.predict(X_test)


# Evaluate
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)


model_name = "Linear Regression"
print(f"{model_name}: RMSE = {rmse:f}, R-squared = {r2:f}")


# Plot
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')
plt.title(f'Linear Regression\nRMSE: {rmse:.2f}, R2: {r2:.2f}')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()
```

A.1.2 Ridge Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score


# Load the dataset

data = pd.read_csv(r"C:\Users\birar\OneDrive\Desktop\SRM\ML PROJECT
)\Chennai (1).csv")


# Select relevant numeric features for regression

numeric_features = ['Area', 'No. of Bedrooms', 'Resale',
'MaintenanceStaff', 'Gymnasium', 'SwimmingPool',
'LandscapedGardens', 'JoggingTrack', 'LiftAvailable',
'VaastuCompliant', 'Microwave', 'GolfCourse', 'TV', 'DiningTable',
'Sofa', 'Wardrobe', 'Refrigerator']


# Target variable (Price)

X = data[numeric_features]

y = data['Price']


# Splitting the dataset into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# Implementation of Ridge Regression

class CustomRidgeRegression:

    def __init__(self, alpha=1.0):

        self.alpha = alpha
```

```
        self.coefficients = None

    def fit(self, X, y):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term (column of ones)
        I = np.eye(X_b.shape[1])
        I[0, 0] = 0  # Don't regularize the bias term
        self.coefficients = np.linalg.inv(X_b.T.dot(X_b) + self.alpha * I).dot(X_b.T).dot(y)

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term (column of ones)
        return X_b.dot(self.coefficients)

# Create an instance of the custom model
ridge_model = CustomRidgeRegression(alpha=1.0)

# Fit the model to the training data
ridge_model.fit(X_train, y_train)

# Predict using the model
y_pred = ridge_model.predict(X_test)

# Evaluate
```

```
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

r2 = r2_score(y_test, y_pred)


model_name = "Ridge Regression"

print(f"{model_name}: RMSE = {rmse:.2f}, R-squared = {r2:.2f}")


# Plot

plt.figure(figsize=(6, 4))

plt.scatter(y_test, y_pred, alpha=0.5)

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')

plt.title(f'Ridge Regression\nRMSE: {rmse:.2f}, R2: {r2:.2f}')

plt.xlabel('Actual Price')

plt.ylabel('Predicted Price')

plt.show()
```

A.1.3 Lasso Regression

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score


# Load the dataset
```

```
data = pd.read_csv(r"C:\Users\birar\OneDrive\Desktop\SRM\ML PROJECT
)\Chennai (1).csv")


# Select relevant numeric features for regression

numeric_features = ['Area', 'No. of Bedrooms', 'Resale',
'MaintenanceStaff', 'Gymnasium', 'SwimmingPool',
'LandscapedGardens', 'JoggingTrack', 'LiftAvailable',
'VaastuCompliant', 'Microwave', 'GolfCourse', 'TV', 'DiningTable',
'Sofa', 'Wardrobe', 'Refrigerator']


# Target variable (Price)

X = data[numeric_features]

y = data['Price']


# Splitting the dataset into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# implementation of Lasso Regression

class CustomLassoRegression:

    def __init__(self, alpha=1.0):

        self.alpha = alpha

        self.coefficients = None


    def fit(self, X, y):
```

```
        X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term
(column of ones)
        I = np.eye(X_b.shape[1])
        I[0, 0] = 0  # Don't regularize the bias term
        self.coefficients = self._coordinate_descent(X_b, y, I)

    def _coordinate_descent(self, X, y, I):
        m, n = X.shape
        beta = np.zeros(n)
        for _ in range(1000):  # Number of iterations
            for j in range(n):
                X_j = X[:, j]
                residual = y - (X @ beta - X_j * beta[j])
                rho = X_j.T @ residual
                if j == 0:
                    beta[j] = rho / (X_j.T @ X_j)
                else:
                    beta[j] = self._soft_threshold(rho, self.alpha)
/ (X_j.T @ X_j + self.alpha)
        return beta

    def _soft_threshold(self, rho, alpha):
        if rho < -alpha:
            return rho + alpha
        elif rho > alpha:
```

```
            return rho - alpha
        else:
            return 0.0

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term
(column of ones)
        return X_b.dot(self.coefficients)

# Create an instance of the custom model
lasso_model = CustomLassoRegression(alpha=1.0)

# Fit the model to the training data
lasso_model.fit(X_train, y_train)

# Predict using the model
y_pred = lasso_model.predict(X_test)

# Evaluate
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

model_name = "Lasso Regression"
print(f"{model_name}: RMSE = {rmse:.2f}, R-squared = {r2:.2f}")
```

```
# Plot
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')
plt.title(f'Lasso Regression\nRMSE: {rmse:.2f}, R2: {r2:.2f}')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()
```

A.1.4 Decision Tree Regressor

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv(r"C:\Users\birar\OneDrive\Desktop\SRM\ML
PROJECT )\Chennai (1).csv")

# Select relevant numeric features for regression
```

```
numeric_features = ['Area', 'No. of Bedrooms', 'Resale',
'MaintenanceStaff', 'Gymnasium', 'SwimmingPool',
'LandscapedGardens', 'JoggingTrack', 'LiftAvailable',
'VaastuCompliant', 'Microwave', 'GolfCourse', 'TV',
'DiningTable', 'Sofa', 'Wardrobe', 'Refrigerator']


# Target variable (Price)
X = data[numeric_features]
y = data['Price']


# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# implementation of Decision Tree Regressor
class CustomDecisionTreeRegressor:
    def __init__(self, max_depth=None, random_state=42):
        self.max_depth = max_depth
        self.random_state = random_state
        self.tree = None


    def fit(self, X, y):
        np.random.seed(self.random_state)
        self.tree = self._build_tree(X, y)
```

```
    def _build_tree(self, X, y, depth=0):
        if len(np.unique(y)) == 1 or (self.max_depth is not
None and depth >= self.max_depth):
            return np.mean(y)

        best_split = self._find_best_split(X, y)
        if not best_split:
            return np.mean(y)

        left_indices = best_split['left_indices']
        right_indices = best_split['right_indices']

        left_tree = self._build_tree(X.iloc[left_indices],
y.iloc[left_indices], depth + 1)
        right_tree = self._build_tree(X.iloc[right_indices],
y.iloc[right_indices], depth + 1)

        return {'split_feature': best_split['split_feature'],
'split_value': best_split['split_value'],
                'left': left_tree, 'right': right_tree}

    def _find_best_split(self, X, y):
        best_split = {}
```

```
        best_mse = float('inf')

        for feature in X.columns:
            values = X[feature].unique()
            for value in values:
                left_indices = X[feature] <= value
                right_indices = X[feature] > value

                y_left, y_right = y[left_indices],
y[right_indices]
                mse = (len(y_left) * np.var(y_left) +
len(y_right) * np.var(y_right)) / len(y)

                if mse < best_mse:
                    best_split = {'split_feature': feature,
'split_value': value,
                                  'left_indices': left_indices,
'right_indices': right_indices}
                    best_mse = mse

        return best_split if best_split else None

    def predict(self, X):
        return X.apply(self._predict_single, axis=1)
```

```
    def _predict_single(self, x, tree=None):
        if tree is None:
            tree = self.tree

        if isinstance(tree, dict):
            feature, value = tree['split_feature'],
tree['split_value']
            if x[feature] <= value:
                return self._predict_single(x, tree['left'])
            else:
                return self._predict_single(x, tree['right'])
        else:
            return tree

# Create an instance of the custom model
decision_tree_model =
CustomDecisionTreeRegressor(random_state=42)

# Fit the model to the training data
decision_tree_model.fit(X_train, y_train)

# Predict using the model
```

```
y_pred = decision_tree_model.predict(X_test)


# Evaluate
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)


model_name = "Decision Tree Regressor"
print(f"{model_name}: RMSE = {rmse:.2f}, R-squared = {r2:.2f}")


# Plot
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(),
y_test.max()], 'r--')
plt.title(f'Decision Tree Regressor\nRMSE: {rmse:.2f}, R2:
{r2:.2f}')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()
```

A.1.5 Random Forest Regressor

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score


# Load the dataset

data = pd.read_csv(r"C:\Users\birar\OneDrive\Desktop\SRM\ML PROJECT
)\Chennai (1).csv")


# Select relevant numeric features for regression

numeric_features = ['Area', 'No. of Bedrooms', 'Resale',
'MaintenanceStaff', 'Gymnasium', 'SwimmingPool',
'LandscapedGardens', 'JoggingTrack', 'LiftAvailable',
'VaastuCompliant', 'Microwave', 'GolfCourse', 'TV', 'DiningTable',
'Sofa', 'Wardrobe', 'Refrigerator']


# Target variable (Price)

X = data[numeric_features]

y = data['Price']


# Splitting the dataset into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# implementation of Random Forest Regressor

class CustomRandomForestRegressor:

    def __init__(self, n_estimators=100, random_state=42):
```

```
        self.n_estimators = n_estimators
        self.random_state = random_state
        self.models = []

    def fit(self, X, y):
        np.random.seed(self.random_state)
        for _ in range(self.n_estimators):
            indices = np.random.choice(range(X.shape[0]),
size=X.shape[0], replace=True)
            X_bootstrap, y_bootstrap = X.iloc[indices],
y.iloc[indices]
            tree = self._build_tree(X_bootstrap, y_bootstrap)
            self.models.append(tree)

    def _build_tree(self, X, y):
        from sklearn.tree import DecisionTreeRegressor
        tree = DecisionTreeRegressor(random_state=self.random_state)
        tree.fit(X, y)
        return tree

    def predict(self, X):
        predictions = np.zeros((self.n_estimators, X.shape[0]))
        for i, model in enumerate(self.models):
            predictions[i] = model.predict(X)
        return np.mean(predictions, axis=0)
```

```
# Create an instance of the custom model
custom_random_forest_model =
CustomRandomForestRegressor(random_state=42)

# Fit the model to the training data
custom_random_forest_model.fit(X_train, y_train)

# Predict using the model
y_pred = custom_random_forest_model.predict(X_test)

# Evaluate
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
model_name = "Random Forest Regressor"
print(f"{model_name}: RMSE = {rmse:.2f}, R-squared = {r2:.2f}")

# Plot
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')
plt.title(f'Random Forest Regressor\nRMSE: {rmse:.2f}, R2:
{r2:.2f}')
plt.xlabel('Actual Price')
```

```
plt.ylabel('Predicted Price')

plt.show()
```

A.1.6 Gradient Boosting Regressor

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score


# Load the dataset

data = pd.read_csv(r"C:\Users\birar\OneDrive\Desktop\SRM\ML PROJECT
)\Chennai (1).csv")


# Select relevant numeric features for regression

numeric_features = ['Area', 'No. of Bedrooms', 'Resale',
'MaintenanceStaff', 'Gymnasium', 'SwimmingPool',
'LandscapedGardens', 'JoggingTrack', 'LiftAvailable',
'VaastuCompliant', 'Microwave', 'GolfCourse', 'TV', 'DiningTable',
'Sofa', 'Wardrobe', 'Refrigerator']


# Target variable (Price)

X = data[numeric_features]

y = data['Price']
```

```
# Splitting the dataset into training and test sets (80% train, 20%
test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# implementation of Gradient Boosting Regressor
class CustomGradientBoostingRegressor:
    def __init__(self, n_estimators=100, learning_rate=0.1,
max_depth=3, random_state=42):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.random_state = random_state
        self.models = []


    def fit(self, X, y):
        np.random.seed(self.random_state)
        residual = y.copy()
        for _ in range(self.n_estimators):
            tree = self._build_tree(X, residual)
            prediction = tree.predict(X)
            residual -= self.learning_rate * prediction
            self.models.append(tree)


    def _build_tree(self, X, y):
```

```
        from sklearn.tree import DecisionTreeRegressor
        tree = DecisionTreeRegressor(max_depth=self.max_depth,
random_state=self.random_state)
        tree.fit(X, y)
        return tree

    def predict(self, X):
        predictions = np.zeros(X.shape[0])
        for model in self.models:
            predictions += self.learning_rate * model.predict(X)
        return predictions

# Create an instance of the custom model
gradient_boosting_model =
CustomGradientBoostingRegressor(random_state=42)

# Fit the model to the training data
gradient_boosting_model.fit(X_train, y_train)

# Predict on the test set
y_pred = gradient_boosting_model.predict(X_test)

# Calculate evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
```

```
# Print the RMSE and R-squared for Gradient Boosting
model_name = "Gradient Boosting Regressor"
print(f"{model_name}: RMSE = {rmse:.2f}, R-squared = {r2:.2f}")


# Plot actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')  # Perfect fit line
plt.title(f'{model_name}\nRMSE: {rmse:.2f}, R2: {r2:.2f}')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()
```

A.2.1 Residual Plot Of Linear Regression

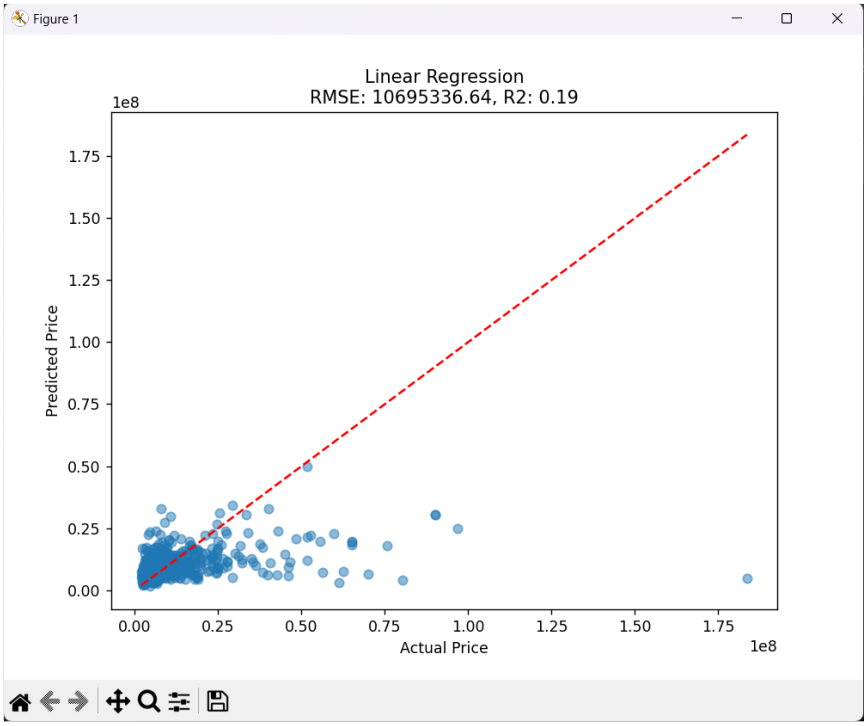


Figure 3 Residual Plot of Linear Regression

A.2.2 Residual Plot Of Ridge Regression

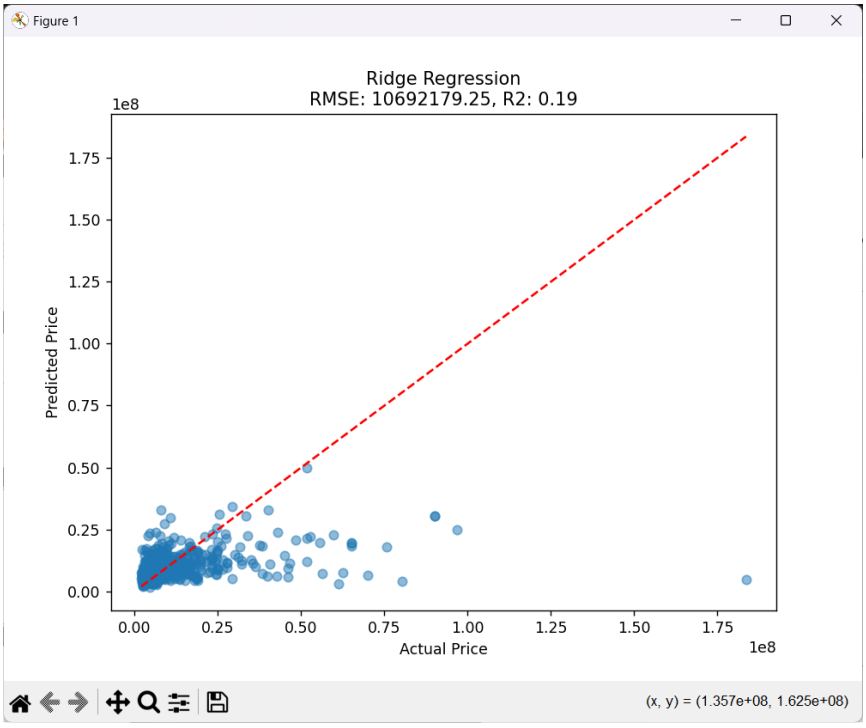


Figure 4 Residual Plot of Ridge Regression

A.2.3 Residual Plot Of Lasso Regression

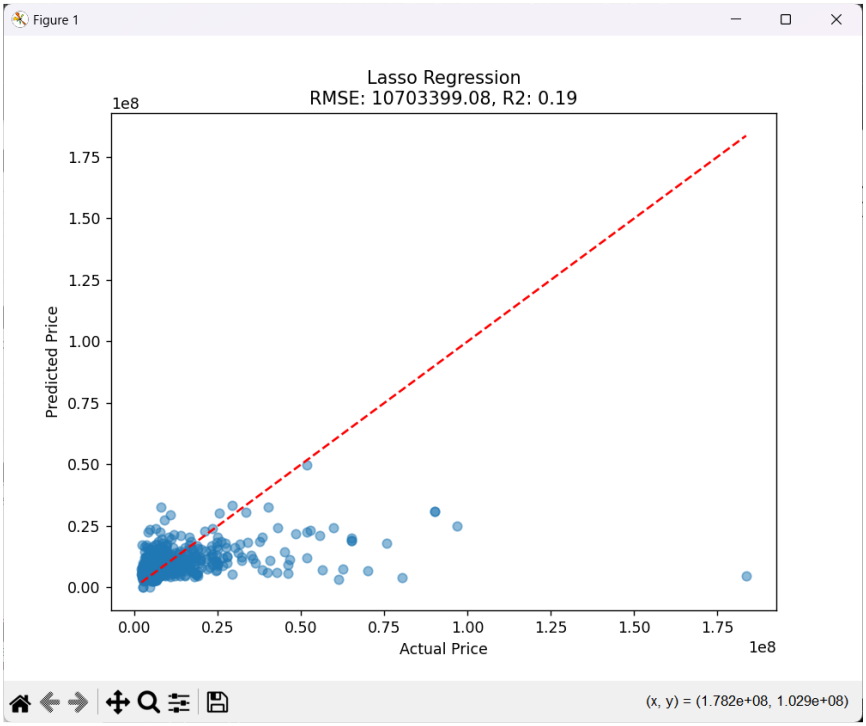


Figure 5 Residual Plot of Lasso Regression

A.2.4 Residual Plot Of Decision Tree Regressor

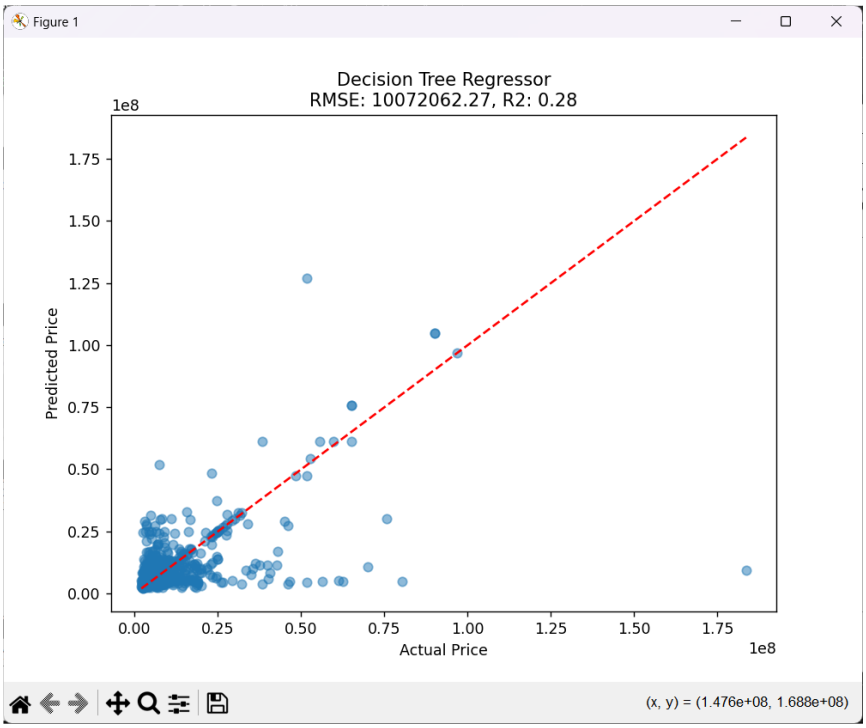


Figure 6 Residual Plot of Decision Tree Regressor

A.2.5 Residual Plot Of Random Forest Regressor

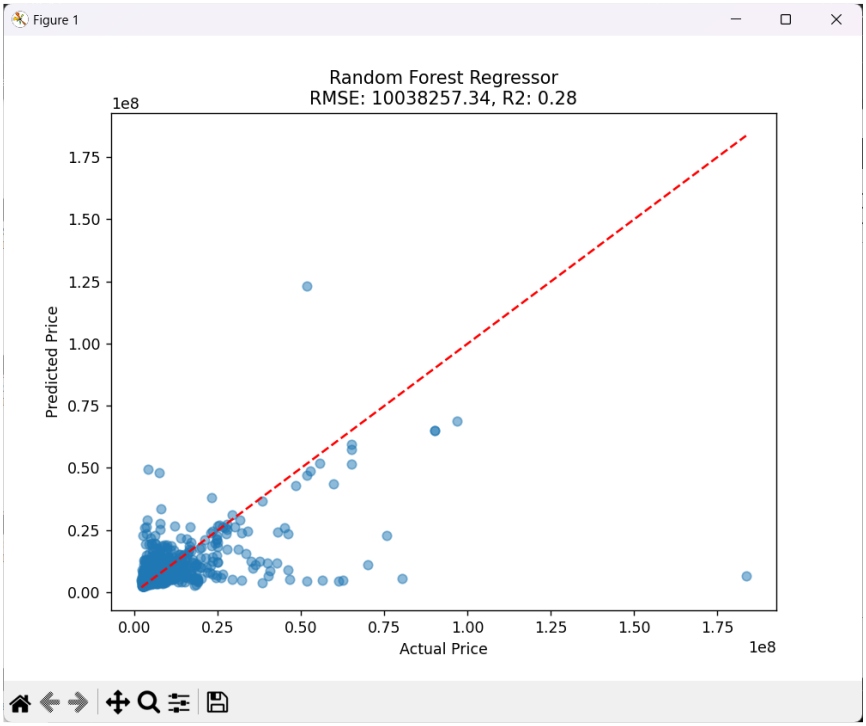


Figure 7 Residual Plot of Random Forest Regressor

A.2.6 Residual Plot Of Gradient Boosting Regressor

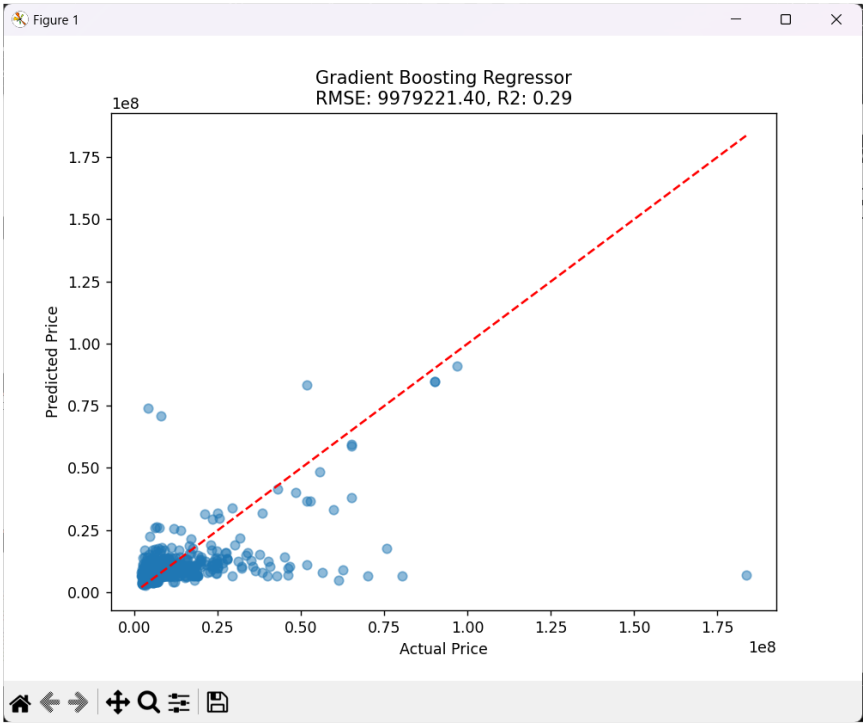


Figure 4 Residual Plot of Gradient Boosting Regressor

A.3.1 RSME and R2 Resultant for all Models

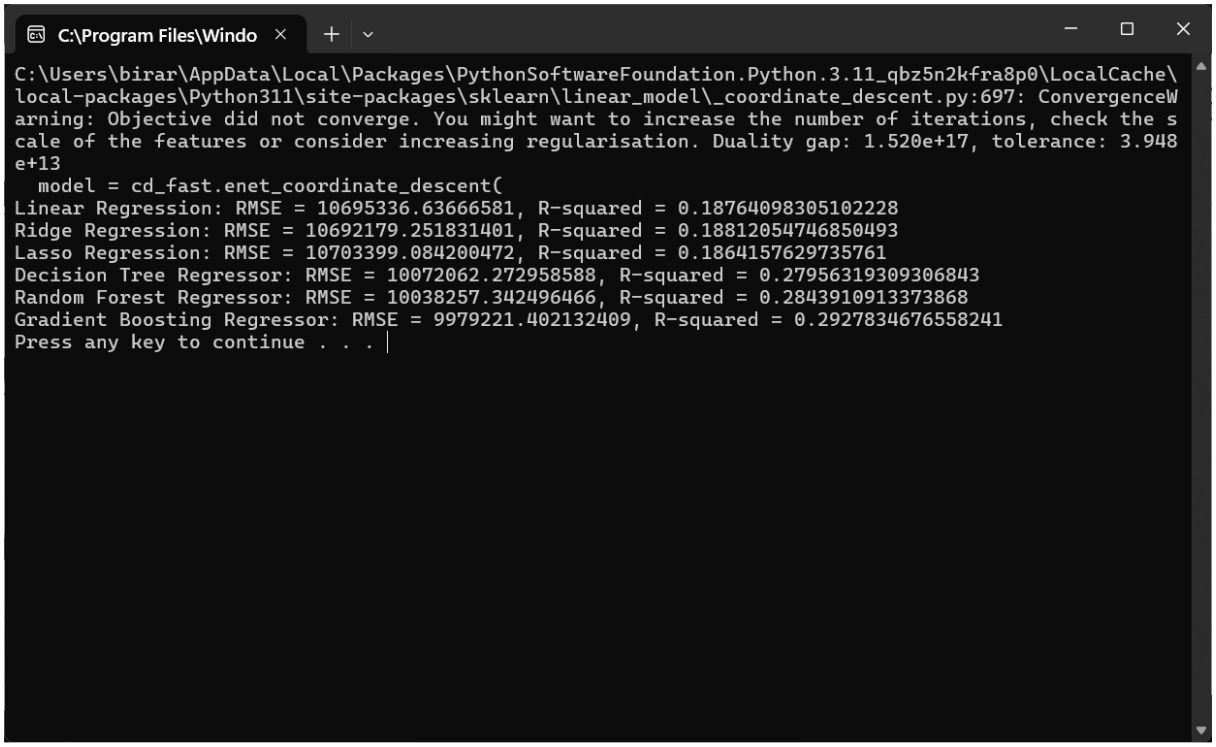


Figure 9 RSME and R^2 Resultant for all Models

A.3.2 Comparison of RSME and R^2 for all Model

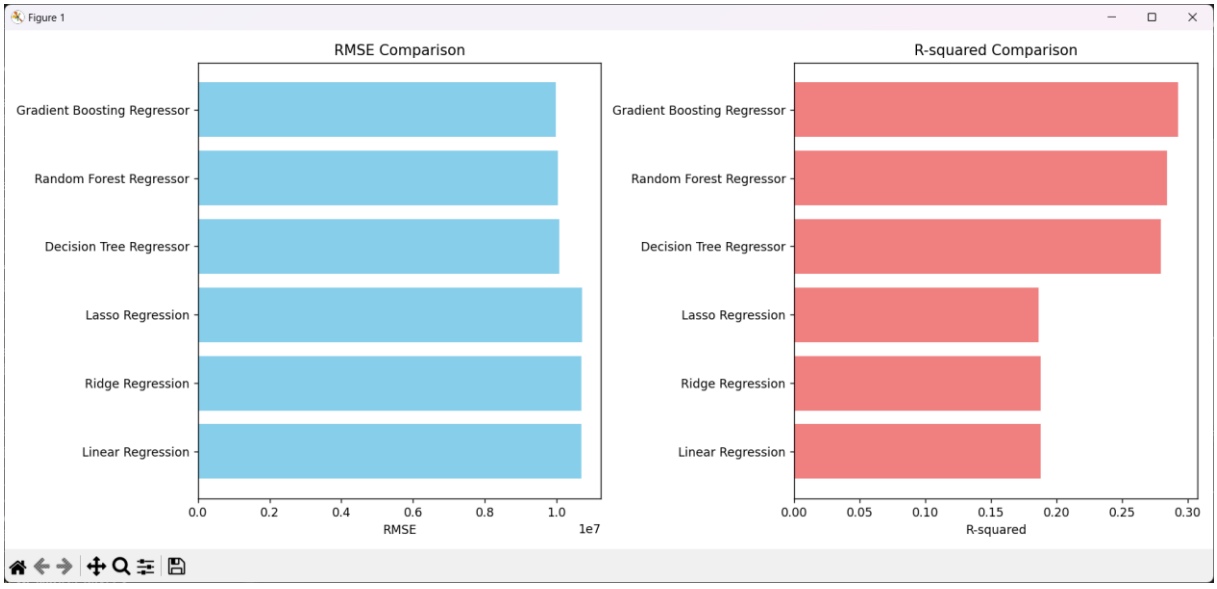


Figure 10 Comparison of RSME and R^2 for all Models

A.4.1 Implementation Of Gradient Boosting Regressor To Predict The Price Based On The User Input

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.tree import DecisionTreeRegressor


# Load the dataset

data = pd.read_csv(r"C:\Users\madha\Downloads\Chennai (1).csv")


# Select relevant numeric features for regression

numeric_features = ['Area', 'No. of Bedrooms', 'Resale',
'MaintenanceStaff', 'Gymnasium', 'SwimmingPool',
'LandscapedGardens', 'JoggingTrack', 'LiftAvailable',
'VaastuCompliant', 'Microwave', 'GolfCourse', 'TV', 'DiningTable',
'Sofa', 'Wardrobe', 'Refrigerator']


# Target variable (Price)

X = data[numeric_features]

y = data['Price']
```

```
# Split the dataset into training and test sets (80% train, 20%
test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Gradient Boosting Regressor parameters
n_estimators = 100  # Number of boosting rounds
learning_rate = 0.1  # Learning rate
max_depth = 3  # Maximum depth of the individual regression
estimators

# Initialize the predictions with the mean of y_train
y_pred_train = np.full(y_train.shape, y_train.mean())
y_pred_test = np.full(y_test.shape, y_train.mean())

# Store the test predictions for each round to calculate the overall
prediction later
test_predictions = np.zeros((n_estimators, len(y_test)))

# Gradient Boosting Algorithm
for i in range(n_estimators):
    # Compute the residuals (negative gradient)
    residuals = y_train - y_pred_train

    # Fit a weak learner (decision tree) to the residuals
```

```
    tree = DecisionTreeRegressor(max_depth=max_depth)
    tree.fit(X_train, residuals)

    # Update the predictions by adding the learning rate times the
predictions of the weak learner
    y_pred_train += learning_rate * tree.predict(X_train)
    y_pred_test += learning_rate * tree.predict(X_test)

    # Store the test predictions for this round
    test_predictions[i, :] = y_pred_test

# Calculate final predictions as the average of predictions over all
estimators
final_y_pred_test = test_predictions.mean(axis=0)

# Calculate evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, final_y_pred_test))
r2 = r2_score(y_test, final_y_pred_test)

# Print the RMSE and R-squared for custom Gradient Boosting
model_name = "Custom Gradient Boosting Regressor"
print(f"{model_name}: RMSE = {rmse:.2f}, R-squared = {r2:.2f}")

# Plot actual vs predicted values
plt.figure(figsize=(8, 6))
```

```
plt.scatter(y_test, final_y_pred_test, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')  # Perfect fit line
plt.title(f'{model_name}\nRMSE: {rmse:.2f}, R2: {r2:.2f}')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()

# Function to predict prices based on new input
def predict_price(model, input_data, n_estimators, learning_rate):
    input_df = pd.DataFrame([input_data], columns=numeric_features)
    prediction = np.full(1, y_train.mean())
    for i in range(n_estimators):
        residuals = y_train - model.predict(X_train)
        tree = DecisionTreeRegressor(max_depth=max_depth)
        tree.fit(X_train, residuals)
        prediction += learning_rate * tree.predict(input_df)
    return prediction[0]

# Prompt the user to enter values for the input features
def get_user_input():
    input_data = {}
    for feature in numeric_features:
        value = float(input(f"Enter value for {feature}: "))
```

```
        input_data[feature] = value
    return input_data


# Get user input
user_input = get_user_input()


# Predict and print the price for the user input
predicted_price =
predict_price(DecisionTreeRegressor(max_depth=max_depth),
user_input, n_estimators, learning_rate)
print(f"Predicted Price for the input data: {predicted_price:.2f}")
```