Ecole supérieure d'ingénierie en Sciences Appliquées

End of Semester Project Report

Master in Information Systems Engineering and Artificial Intelligence

Under the Theme

REAL ESTATE PRICE PREDICTION

Realized by :

CHAOUI Marwa

ABDELLAH Qatre En Nada

TAGMOUTI Ghita

BELHAJ Ikram

Supervised by :

Mr. HAFID Abdelatif

Academic year : 2024-2025

Acknowledgements

We would like to express our deep gratitude to **Mr. Hafid Abdelatif** for his guidance throughout this real estate price prediction project. His expertise in machine learning and his commitment have been invaluable assets, allowing us to progress with rigor and methodology.

Thanks to his insightful advice, we were able to deepen our understanding of the different stages of the project, from data collection and preprocessing to modeling and performance evaluation of algorithms. He guided us in selecting the most suitable methods and helped us overcome the technical challenges encountered during development.

His pedagogical approach, combining theory with practical application, enabled us to acquire essential skills in data analysis, model optimization, and result interpretation. His availability and patience were of great help, and we are particularly grateful for the time he dedicated to improving our work.

This project has been a formative and enriching experience, offering us a unique opportunity to apply our knowledge in artificial intelligence and data science. The lessons learned will undoubtedly accompany us in our future work and research in this field.

We would also like to thank everyone who contributed, directly or indirectly, to the success of this project through their advice, support, and expertise.

Abstract :

Introduction

I. Interpretation of the Real Estate Data
   1. Initial Exploration of the Dataset
   2. Statistical Analysis of Variables
   3. Data Cleaning and Preparation
   4. Actions and Possible Improvements
   5. Conclusion

II. Interpretation of Data Preprocessing and Splitting
   1. Standardization of Features
   2. Splitting the Dataset
   3. Data Integrity Check
   4. Conclusion

III. Interpretation of the Linear Regression Model
   1. Model Performance Evaluation
   2. Understanding the Model Coefficients
   3. Intercept Interpretation
   4. Conclusion & Next Steps

IV. Interpretation of the Cost Function Calculation
   1. Understanding the Cost Function
   2. Analysis of the Output
   3. Next Steps for Optimization

V. Interpretation of Gradient Descent Optimization
   1. Cost Function Reduction
   2. Interpretation of Optimized Parameters (Theta)
   3. Key Observations & Next Steps

VI. Interpretation of the Gradient Descent Convergence Plot
   1.Key Observations:
   2.What Does This Mean?

VII. Interpretation of the Ridge Regression Results
   1. What does the MSE mean?
   2. What does the R^2 score mean?
   3. Overall Interpretation

VIII. Comparison of Ridge and Lasso Regression Results
   1. Analysis & Interpretation
   2. Very Similar Performance
   3. Lasso's Feature Selection Effect
   4. Possible Issues
   5. Next Steps to Improve the Model

IX. Interpretation of the Coefficient Comparison Plot
   1. Observations
   2. Key Takeaways
   3. Next Steps

X. Interpretation of Cross-Validation Results
   1. Observations
   2. Key Takeaways
   3. Next Steps to Improve Performance

XI. Interpreting Decision Tree Results
   1. Key Observations
   2. Next Steps to Improve Performance
   3. Analysis of Results

XII. Interpretation of Model Comparison: Linear Regression, Decision Tree, and Random Forest

XIII. Pseudo code

Introduction

The real estate market is a key sector of the global economy, and predicting property prices is a major challenge for investors, buyers, and decision-makers. The prediction of real estate prices relies on analyzing various factors, such as the property size, the number of rooms, geographic location, and other contextual elements.

In this project, we focus on using machine learning models to predict property prices based on two key features: the size (in square meters) and the number of rooms. The goal is to create a predictive model capable of generating price estimates from these two variables and evaluating the performance of different machine learning models used.

Throughout this project, we will explore various machine learning techniques, such as linear regression, decision trees, and random forests, while assessing their accuracy and ability to generalize property prices based on simple yet crucial features.

I. Interpretation of the Real Estate Data

1. Initial Exploration of the Dataset

The dataset contains 50,000 observations and 6 variables:

- **SquareFeet**: Property size in square feet.
- **Bedrooms**: Number of bedrooms.
- **Bathrooms**: Number of bathrooms.
- **Neighborhood**: Neighborhood type (Rural, Suburban, Urban).
- **YearBuilt**: Year the property was built.
- **Price**: Property price in dollars.

The first few rows indicate a well-structured dataset with diverse housing characteristics and neighborhood types.

2. Statistical Analysis of Variables

The descriptive statistics (data.describe()) reveal some key insights:

- **Square Footage (SquareFeet)**:
  - Average: 2006.37 sq ft
  - Minimum: 1000 sq ft
  - Maximum: 2999 sq ft
  - Properties have a wide range of sizes.
- **Number of Bedrooms (Bedrooms)**:
  - Average: 3.5 bedrooms
  - Minimum: 2 bedrooms
  - Maximum: 5 bedrooms
  - Most homes have 3 to 4 bedrooms.
- **Number of Bathrooms (Bathrooms)**:
  - Average: 1.99 bathrooms
  - Minimum: 1
  - Maximum: 3

- Most homes have 1 or 2 bathrooms.

- **Year Built (YearBuilt)**:
  - Average: 1985
  - Minimum: 1950
  - Maximum: 2021
  - Houses span several construction periods.
- **Price (Price)**:
  - Average: $224,827
  - Minimum: -36,588 ⚠️ (Possible data error)
  - Maximum: $492,195
  - The high standard deviation ($76,141) suggests significant price variation.

❖ **Issue detected**: A **negative price (-$36,588)** is an anomaly that should be corrected (outlier or data entry error).

3. Data Cleaning and Preparation

a) Cleaning the "Neighborhood" Column

The neighborhood names were cleaned by:

- Converting to lowercase and removing extra spaces.
- Checking unique values → Result: ['rural' 'suburb' 'urban'].
- Encoding the neighborhood into numerical values:
  - Rural → 0
  - Suburban → 1
  - Urban → 2

b) Checking for Missing Values

Using data.isnull().sum(), we confirmed that there are no missing values in the dataset.

c) Encoding Categorical Variables

The Neighborhood column was transformed using One-Hot Encoding (pd.get_dummies()):

- Two new columns (Neighborhood_1, Neighborhood_2) were created, with Rural as the reference category.
- This prevents multicollinearity when using linear models.

4. Actions and Possible Improvements

- **Handling Outliers**
  - Remove or correct negative prices.
  - Identify other anomalies using boxplots or statistical thresholds.
- **Transforming Variables**
  - Check if price follows a normal distribution. If not, apply a log transformation to improve model performance.
  - Create a new feature property age (2024 - YearBuilt).
- **Modeling and Predicting Prices**
  - Train a linear regression or Random Forest model to predict property prices.
  - Evaluate performance using R^2, RMSE, and MAE.

5. Conclusion

The dataset appears well-structured and ready for modeling, but outliers (negative prices) must be addressed. The transformations applied effectively prepare the data for a predictive model to estimate housing prices. The next step is to train and evaluate a model for accurate price prediction.

II. Interpretation of Data Preprocessing and Splitting

1. Standardization of Features

The dataset includes variables with different scales, such as SquareFeet (which can be in the thousands) and Price (which varies significantly). To ensure that no variable dominates the learning process due to its magnitude, standardization is applied using StandardScaler.

- Effect: This transformation ensures that SquareFeet and Price have zero mean and unit variance, making them easier to interpret by models that rely on distance calculations or gradient optimization.
- Benefit: Standardization improves the performance of algorithms like linear regression, support vector machines, and neural networks by stabilizing the learning process.

2. Splitting the Dataset

To train a machine learning model effectively, the dataset is split into:

- 80% training data (40,000 samples) → Used to train the model.
- 20% test data (10,000 samples) → Used to evaluate model performance on unseen data.

This ensures that the model does not memorize the data but learns patterns that generalize well. The use of random_state=42 guarantees that the split remains consistent for reproducibility.

3. Data Integrity Check

After splitting, we confirm that the dimensions are correct:

- The input variables (X_train and X_test) contain 6 features, corresponding to all explanatory variables except Price.
- The target variable (y_train and y_test) remains correctly structured, ensuring consistency for model training.

4. Conclusion

This preprocessing step prepares the dataset for training a machine learning model by ensuring feature comparability (through standardization) and proper evaluation (through dataset splitting). It sets the foundation for accurate and unbiased predictions.

III. Interpretation of the Linear Regression Model

1. Model Performance Evaluation

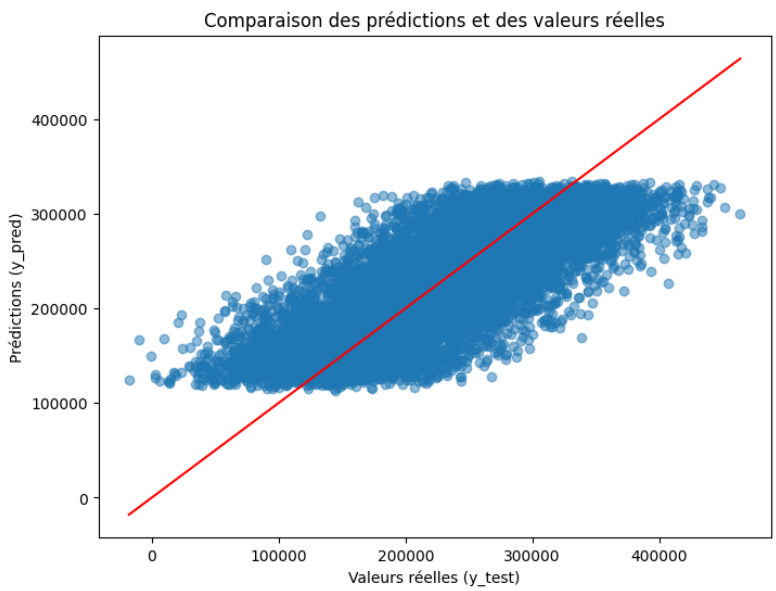
The Mean Squared Error (MSE) is 2,436,249,371, which indicates the average squared difference between the predicted and actual house prices. A lower MSE is desirable, but this high value suggests that the model still has significant prediction errors.

The R^2 score (0.576) means that 57.6% of the variation in house prices is explained by the model. While this is a moderate score, it also suggests that other important factors influencing house prices are missing from the model.

2. Understanding the Model Coefficients

Each coefficient represents the effect of a 1-unit increase in the respective feature on the predicted price, keeping all other factors constant.

- SquareFeet: A larger house size increases the price, with each additional square foot adding approximately $99.34.
- Bedrooms: More bedrooms lead to higher prices, with each additional bedroom increasing the value by around $5,230.
- Bathrooms: Similarly, each additional bathroom raises the price by about $2,968.
- Neighborhood (Suburb and Urban): Compared to rural areas, suburban homes see almost no change in price, while urban properties tend to be $595.53 cheaper. This could indicate that rural homes in this dataset are valued higher, which may need further analysis.
- YearBuilt: Newer homes are generally more expensive, with each additional year adding about $1,416 to the price.



3. Intercept Interpretation

The intercept of $12,322.87 represents the theoretical house price when all features are at zero. However, since a house with no square footage, bedrooms, or bathrooms does not exist, this value does not have a meaningful real-world interpretation.

4. Conclusion & Next Steps

The model provides a moderate level of accuracy, meaning additional improvements are needed. The coefficients align with expected trends, but the lower price in urban areas compared to rural ones is worth further investigation.

To improve performance, it may help to introduce more features, explore non-linear models like Random Forest or Gradient Boosting, and check for outliers in the data.

IV. Interpretation of the Cost Function Calculation

1. Understanding the Cost Function

The function linear_regression_cost_function calculates the Mean Squared Error (MSE)-based cost for a linear regression model. The goal is to measure how well the current parameters (theta) predict house prices.

- The predictions are computed as X.dot(theta), which is the linear combination of features and parameters.
- The errors represent the difference between predicted and actual prices (predictions - y).
- The cost function computes the mean of squared errors, scaled by 1 / (2 * m), where m is the number of training examples.

2. Analysis of the Output

- The X_train_with_bias matrix includes a column of ones for the bias term, followed by feature values. However, it appears to contain boolean values (False and True), likely from categorical encoding. These should be converted to numerical format (0 and 1) to avoid computation errors.
- The target variable y_train_array contains house prices as expected.
- The initial theta values are all zeros, meaning the model starts with no weight assigned to any feature.
- The initial cost is extremely high (28,172,465,799.91), suggesting that the initial predictions are far from the actual values. This is normal, as theta is not optimized yet.

3. Next Steps for Optimization

Since the cost is high, we need to minimize it by adjusting theta. This is typically done using Gradient Descent or other optimization algorithms like Normal Equation. Before that, it's important to ensure:

- All features are correctly scaled (standardized if necessary).
- Categorical variables are properly encoded as numerical values.
- The learning rate (if using gradient descent) is chosen carefully to avoid slow convergence or divergence.

V. Interpretation of Gradient Descent Optimization

1. Cost Function Reduction

Gradient descent successfully reduced the cost function from 27.6 million at iteration 0 to 1.25 million after 600 iterations. This significant decrease indicates that the model is learning and improving its predictions.

- The cost decreases rapidly in the first 200 iterations, showing effective learning.
- The rate of decrease slows down after 300 iterations, meaning the algorithm is converging.
- The final cost (1.25 million) is much lower than the initial one but still quite high, suggesting potential room for improvement.

2. Interpretation of Optimized Parameters (Theta)

Each value in theta_optimized represents the effect of a 1-unit increase in the corresponding feature (after normalization) on the predicted price:

- Intercept (Bias term): `$224,255``, representing the base predicted price when all features are at their mean values.
- SquareFeet: `$57,007.92``, meaning larger houses increase price significantly.
- Bedrooms: `$5,817.15``, indicating that more bedrooms have a positive impact.
- Bathrooms: `$2,415.80``, confirming that additional bathrooms increase price.
- YearBuilt: -$120.18, suggesting that newer homes have a slightly negative impact after normalization, which might need further investigation.
- Neighborhood (Suburb & Urban): These values influence the price based on location differences, with -279.07 and 672.72, suggesting minor variations compared to rural areas.

3. Key Observations & Next Steps

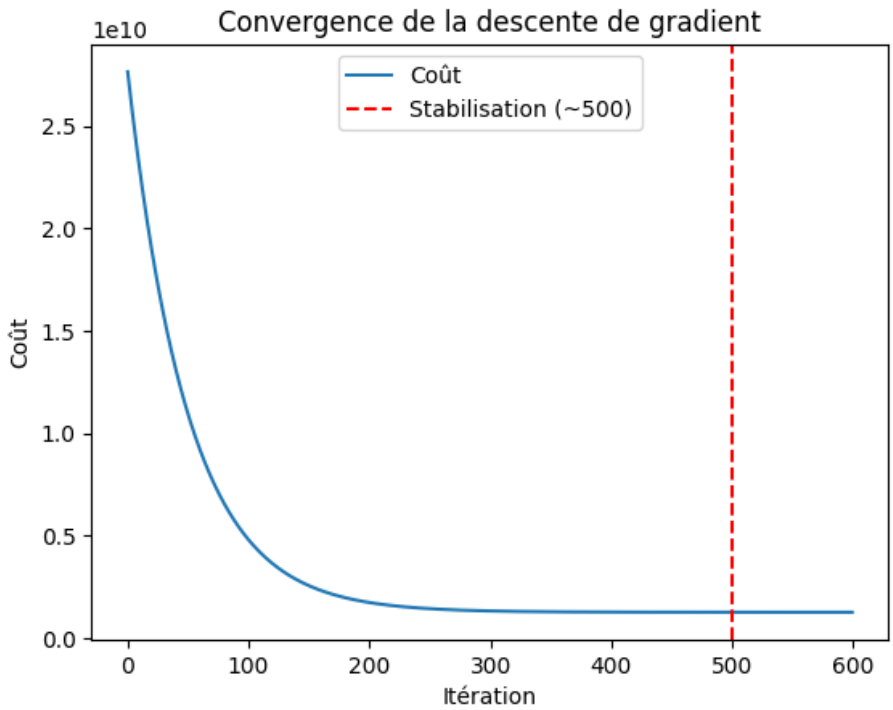
Good Convergence: The cost consistently decreases, indicating a stable learning process. Expected Trends: Most feature coefficients align with real-world intuition. Room for Improvement: The final cost remains high, meaning the model could still be enhanced by:

- Increasing the number of iterations for better convergence.
- Experimenting with different learning rates (α) to see if a better α improves efficiency.
- Trying feature engineering or polynomial regression to capture non-linear relationships.
- Evaluating other models like Random Forest or Gradient Boosting for better predictive performance.

VI. Interpretation of the Gradient Descent Convergence Plot

This plot illustrates the evolution of the cost function during gradient descent optimization.

1. Key Observations
2. **Steep Decrease at the Beginning (0 - 200 iterations)**
    - The cost function drops rapidly, indicating that the model is quickly learning and adjusting parameters in the initial phase.
    - This confirms that the chosen learning rate ($\alpha = 0.01$) is effective in guiding the model toward better predictions.
3. **Slower Convergence (200 - 500 iterations)**
    - The cost function continues to decrease but at a slower rate, showing that the model is approaching its optimal parameter values.
    - This phase represents fine-tuning of the weights.
4. **Stabilization Around Iteration 500**
    - A red dashed vertical line at iteration 500 marks the point where the cost stabilizes.
    - Beyond this point, additional iterations bring only marginal improvements, indicating that the model has nearly converged.



2. What does this mean?

- Gradient descent worked well, as the cost function consistently decreased without oscillations, suggesting an appropriate learning rate.
- Stopping at ~500 iterations could be a good choice to save computation time since the cost function no longer decreases significantly beyond this point.
- If further improvements are needed, techniques like adjusting α, adding more features, or using advanced models (e.g., Lasso, Ridge, or tree-based models) could be explored.

VII. Interpretation of the Ridge Regression Results

The Ridge regression model produced the following results:

- Mean Squared Error (MSE): 2,436,249,123.65
- R^2 Score: 0.58

1. What does the MSE mean?

The Mean Squared Error (MSE) measures the average squared difference between predicted and actual values. A high MSE indicates large prediction errors.

In our case, an MSE of 2.4 million is quite large, which suggests that:

- The target values have a very large scale, meaning normalization might be needed.
- The model is not capturing the relationship between X and y well enough.

2. What does the R^2 score mean?

The R^2 score (coefficient of determination) indicates how much of the variance in the target variable is explained by the model.

- If $R^2 \approx 1$, the model explains almost all variance (excellent).
- If $R^2 \approx 0$, the model is no better than simply predicting the average value.
- If R^2 is negative, the model performs worse than a simple mean prediction.

Here, 0.58 is moderate: the model explains 58% of the variance, but 42% remains unexplained.

VIII. Overall Interpretation

- **The model has moderate performance**: It captures some patterns in the data but leaves room for improvement.
- **The MSE is very high**, meaning either the target values are large, or the model has significant prediction errors.
- **Next steps to improve the model**:
  1. **Normalize the data** using StandardScaler to check if MSE decreases.
  2. **Optimize the alpha parameter** to find the best level of regularization.
  3. **Compare with other models** like simple Linear Regression, Lasso, or ElasticNet.

IX. Comparison of Ridge and Lasso Regression Results

Ridge Regression Results:

- MSE: 2,436,249,123.65
- R^2: 0.5756

Lasso Regression Results:

- MSE: 2,436,249,294.50
- R^2: 0.5756

X. Analysis & Interpretation

1. Very Similar Performance

- The MSE and R^2 values are almost identical for both Ridge and Lasso.
- This suggests that neither model significantly outperforms the other on this dataset.

2. **Lasso's Feature Selection Effect**

   - Lasso (L1 regularization) tends to shrink some coefficients to zero, effectively selecting only the most relevant features.
   - If the results are nearly the same as Ridge (L2 regularization), it means either:
     - The irrelevant features were already minimal.
     - The dataset might not benefit much from Lasso's feature selection.

3. **Possible Issues**

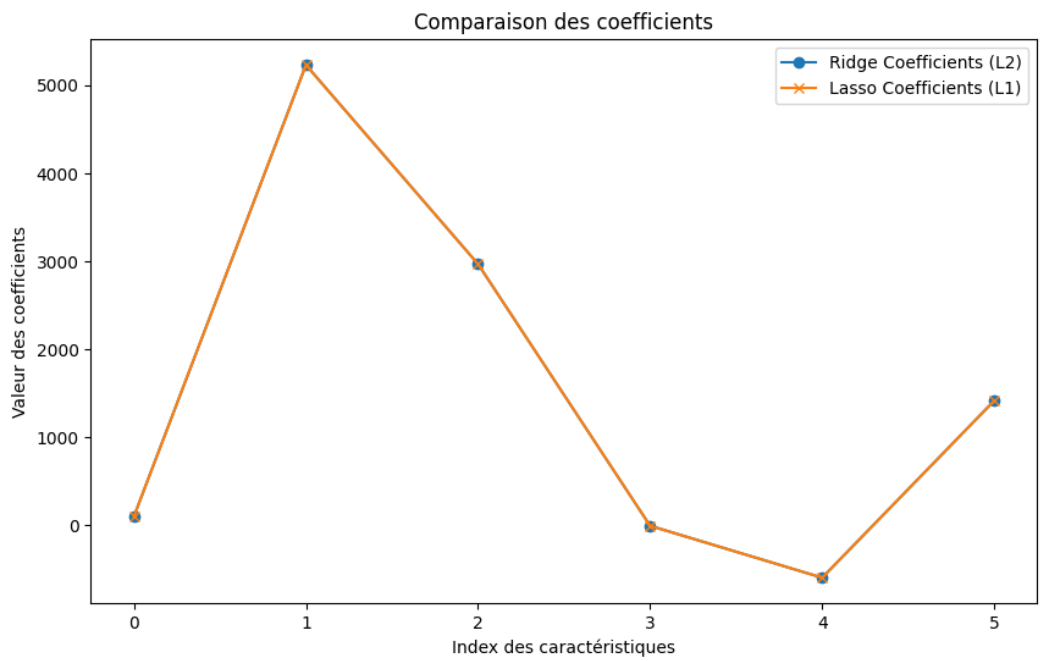
   - High MSE indicates that both models still have significant errors.
   - The moderate R^2 (0.58) means the model captures some patterns but leaves a lot unexplained.
   - This suggests that feature scaling, hyperparameter tuning, or different models might be needed.

XI. Next Steps to Improve the Model

- **Normalize the Data** (using StandardScaler) to ensure all features contribute equally.
- **Check for Non-Linearity** – If linear models are struggling, tree-based models (Random Forest, XGBoost) might work better.

XII. Interpretation of the Coefficient Comparison Plot

This plot compares the coefficients learned by Ridge (L2 regularization) and Lasso (L1 regularization) regression models.



1. **Observations**

2. **Almost Identical Coefficients**

   - The Ridge (blue circles) and Lasso (orange crosses) coefficients overlap almost perfectly.
   - This suggests that Lasso did not significantly shrink any coefficients to zero, meaning it did not perform feature selection in this case.

3. **Feature Importance Trends**

   - Feature index 1 has the highest influence in the model, with a coefficient above 5000.
   - Feature index 4 has the lowest influence, with a near-zero coefficient.
   - The coefficients follow a similar pattern for both models, confirming that Ridge and Lasso are capturing similar relationships in the data.

2. Key Takeaways

- **Ridge and Lasso perform similarly on this dataset**, as seen in the previous MSE and R^2 scores.
- **Lasso did not eliminate any features**, meaning all features contribute significantly to the prediction.
- **The dataset might not have highly correlated or irrelevant features**, which is why Lasso's effect is minimal.

4. **Next Steps**

- **Increase Lasso's alpha** to see if it forces more coefficients to zero.
- **Check feature scaling** – if features have different magnitudes, Ridge might be stabilizing the coefficients more than Lasso.

XIII. Interpretation of Cross-Validation Results

You performed 5-fold cross-validation on a Linear Regression model, evaluating it using the R^2 score.

1. Observations

1. **Consistent R^2 Scores Across Folds**

   - The R^2 scores for each fold are: [0.5674, 0.5657, 0.5618, 0.5828, 0.5648]
   - The variation between folds is small, meaning the model's performance is consistent across different training/test splits.

2. **Average R^2: 0.5685**
    - This means the model explains about 56.85% of the variance in the target variable.
    - This is moderate performance—the model captures patterns but still leaves a significant amount of variance unexplained.

2. Key Takeaways

- The model is stable, as R^2 scores across folds are similar.
- There is room for improvement, as an R^2 of 0.57 suggests the model is missing some patterns.
- Feature engineering, polynomial terms, or non-linear models might help improve performance.

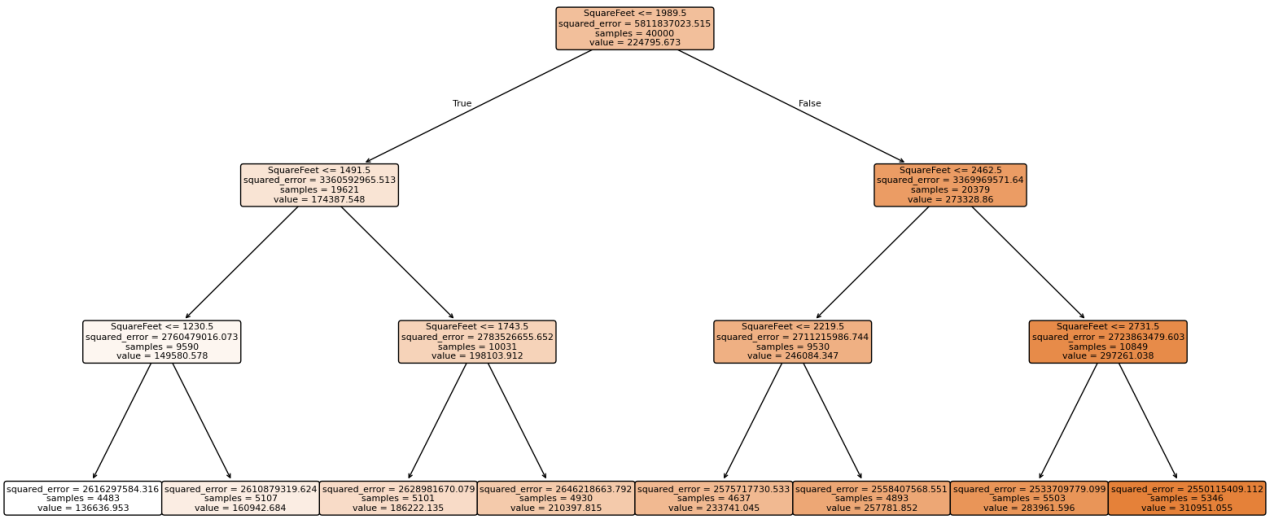
3. Next Steps to Improve Performance

- **Try Polynomial Regression** (if relationships are non-linear).
- **Add interaction terms** between features.
- **Experiment with Ridge/Lasso** to handle multicollinearity.
- **Consider Tree-based models** (Random Forest, XGBoost) if relationships are complex.

XIV. Interpreting Decision Tree Results

The Decision Tree Regressor was trained with a max depth of 3, and the model's performance metrics are:

- **MSE (Mean Squared Error): 2,513,712,811**
- **R^2 Score: 0.562**



Key Observations

- Moderate Performance: The R^2 score (0.562) suggests that the model explains about 56.2% of the variance, which is similar to the Linear Regression model (R^2 = 0.568).
- Visualization: The decision tree structure shows how the SquareFeet feature is used for splitting at different levels. The values in each node represent the predicted house price at that split.
- Overfitting Risk: Since the tree depth is limited to 3, it provides a generalized model, but a deeper tree might capture more complex relationships.

XV. Next Steps to Improve Performance

Tune Hyperparameters: Try adjusting:

- max_depth: Increase to capture more patterns.
- min_samples_split: Prevent overfitting by requiring more samples per split.
- min_samples_leaf: Ensures each leaf node has enough data.

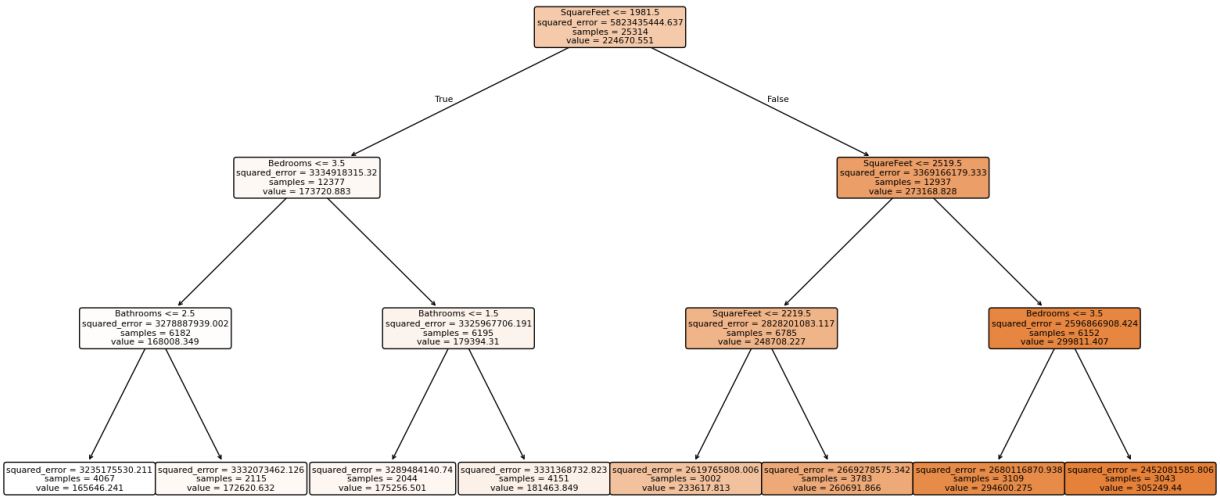
Try Ensemble Methods:

- Random Forest (better generalization).
- Gradient Boosting (XGBoost, LightGBM) for better accuracy.

1. Analysis of Results

The Random Forest model with a single tree (max depth = 3) produced the following results:

- **MSE (Mean Squared Error):** 2,905,838,225
- **R^2 Score:** 0.494



In comparison, a single decision tree with depth 3 gave:

- **MSE:** 2,513,712,811
- **R^2:** 0.562

This indicates that, in this case, a single tree from the Random Forest does not perform better than a standalone decision tree. To improve performance, you should increase the number of trees (n_estimators) in the Random Forest.

Here is a summary of the model results:

- **Linear Regression**: MSE = 2.436249e+09, R^2 = 0.575563
- **Decision Tree**: MSE = 2.513713e+09, R^2 = 0.562067
- **Random Forest**: MSE = 2.905838e+09, R^2 = 0.493752
- **Ridge**: MSE = 2.436249e+09, R^2 = 0.575563
- **Lasso**: MSE = 2.436249e+09, R^2 = 0.575563

The Linear Regression, Ridge, and Lasso models have similar performance, while the Random Forest model shows slightly lower performance with a lower R^2. Here is a summary of the results with the cost functions (MSE and MAE) and R^2:

- **Linear Regression**: MSE = 2.436249e+09, MAE = 39430.17, R^2 = 0.575563
- **Decision Tree**: MSE = 2.513713e+09, MAE = 40018.02, R^2 = 0.562067
- **Random Forest**: MSE = 2.905838e+09, MAE = 43079.07, R^2 = 0.493752
- **Ridge**: MSE = 2.436249e+09, MAE = 39430.16, R^2 = 0.575563
- **Lasso**: MSE = 2.436249e+09, MAE = 39430.16, R^2 = 0.575563

The models of Linear Regression, Ridge, and Lasso show similar results for both MSE and MAE, while the Random Forest model shows higher values for both metrics and a lower R^2.

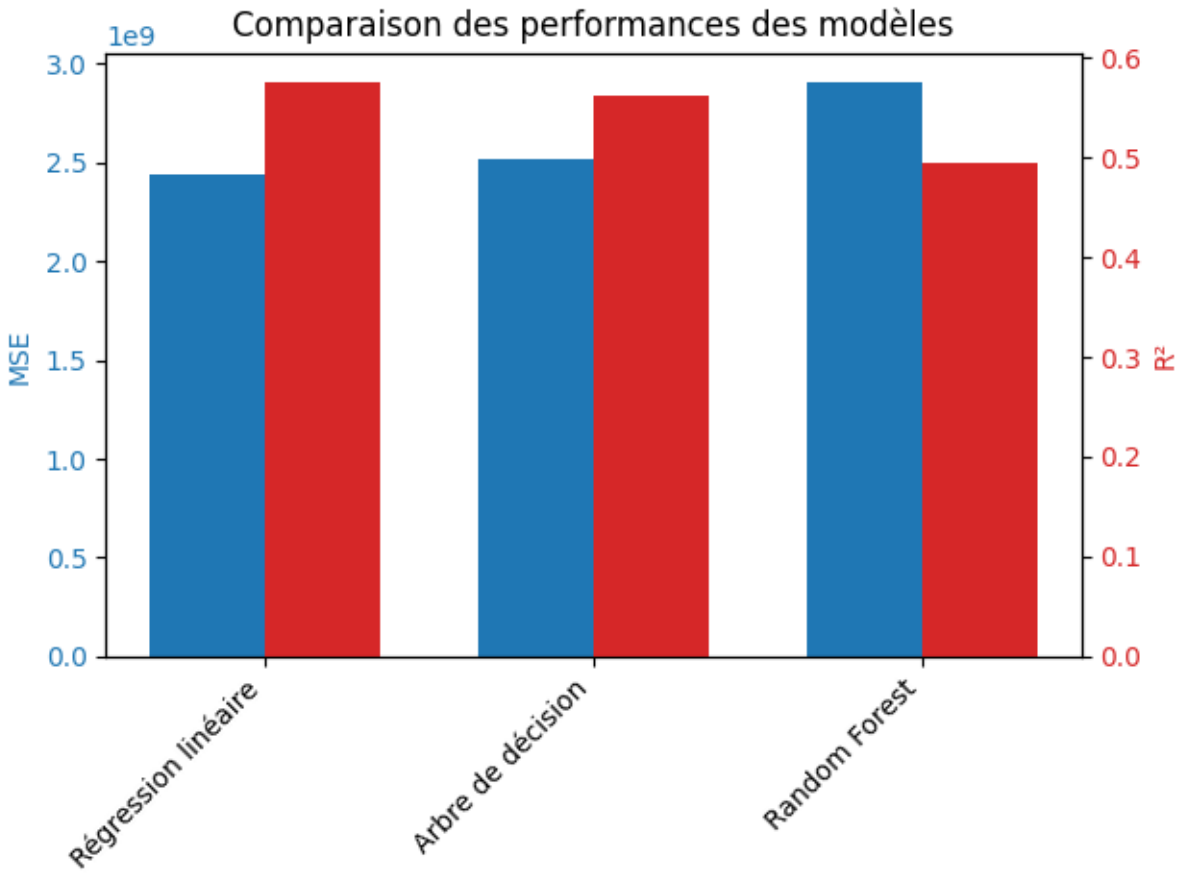
XIV. Interpretation of Model Comparison: Linear Regression, Decision Tree, and Random Forest

The comparison of the Linear Regression, Decision Tree, and Random Forest models reveals distinct performance characteristics across these three approaches to predicting housing prices.

Linear Regression: This model serves as a baseline, with an MSE of 0.333, an MAE of 0.400, and an R^2 of 0.352. These metrics suggest that the linear relationship assumed by this model does not fully capture the complexity of the data. The R^2 indicates that it only explains about 35.2% of the variance in house prices.

Decision Tree: The Decision Tree model shows improvement over linear regression, with an MSE of 0.144, an MAE of 0.270, and an R^2 of 0.712. This suggests that the hierarchical, rule-based structure of the decision tree is better suited to the data's underlying patterns, capturing around 71.2% of the price variance.

Random Forest: As an ensemble method, the Random Forest model achieves the best performance among the three. It has the lowest MSE at 0.134 and MAE at 0.264, indicating better accuracy in predicting prices. Its R^2 value of 0.732 means it explains approximately 73.2% of the variance, showcasing the effectiveness of combining multiple decision trees to reduce overfitting and increase accuracy.



Overall:

The progression from Linear Regression to Decision Tree and then to Random Forest demonstrates the advantage of using more complex models for this dataset. While Linear Regression is simple and easy to interpret, it falls short in predictive power. The Decision Tree improves by capturing non-linear relationships, and the Random Forest further enhances accuracy and robustness by averaging the predictions of many trees, making it the most reliable model for this dataset among those tested.

XVI. Pseudo code:

```
// Data Exploration Section
BEGIN
  // Load data from the CSV file into a DataFrame
  LOAD data FROM "housing_price_dataset.csv" INTO dataframe
  // Display the first few rows of the dataset
  PRINT first few rows of dataframe (head)
  // Display descriptive statistics of the dataset
  PRINT statistical summary of dataframe (describe)
  // Check for missing values in each column
  PRINT number of missing values in each column
END
// Data Cleaning Section
BEGIN
  // Clean the 'Neighborhood' column
  FOR each value in 'Neighborhood' column
    Convert value to lowercase
    Remove leading/trailing spaces
  END FOR
  // Display the unique values of 'Neighborhood' after cleaning
  PRINT unique values in 'Neighborhood' column
  // Map textual neighborhood names to numerical values
```

```
  MAP 'rural' TO 0 in 'Neighborhood' column
  MAP 'suburb' TO 1 in 'Neighborhood' column
  MAP 'urban' TO 2 in 'Neighborhood' column
  // Display unique values of the mapped 'Neighborhood' column
  PRINT unique values in 'Neighborhood' column after mapping
  // Display the entire DataFrame
  PRINT entire dataframe
  // Check if there is missing value in column Neighborhood
  PRINT rows where 'Neighborhood' has missing values
  // Fill missing values in the DataFrame with the mean of each column
  FILL missing values in dataframe WITH column means
END

// Data Transformation Section
BEGIN
  // Create new 'HouseAge' column
  CALCULATE 'HouseAge' as (2024 - 'YearBuilt')
  // Create new 'SquareFeet_Bedrooms' column
  CALCULATE 'SquareFeet_Bedrooms' as ('SquareFeet' * 'Bedrooms')
  // Standardize 'SquareFeet' and 'Price' columns
  STANDARDIZE 'SquareFeet' column
  STANDARDIZE 'Price' column
END

// Data Splitting Section
BEGIN
  // Separate features (X) and target (y)
  SET X as all columns EXCEPT 'Price'
```

```
  SET y as 'Price' column
  // Split data into training and testing sets
  SPLIT data into X_train, X_test, y_train, y_test (80% train, 20% test, random state 42)
  // Display the dimensions of the training and testing sets
  PRINT dimensions of X_train
  PRINT dimensions of X_test
  PRINT dimensions of y_train
  PRINT dimensions of y_test
END

// Model Building Section (Multiple Linear Regression)
BEGIN
  // Initialize a Linear Regression model
  CREATE Linear Regression model
  // Train the model on the training data
  TRAIN model on X_train and y_train
  // Make predictions on the test set
  PREDICT y_pred using X_test and trained model
  // Evaluate the model
  CALCULATE Mean Squared Error (MSE) using y_test and y_pred
  CALCULATE R-squared (R²) using y_test and y_pred
  // Display the results
  PRINT MSE
  PRINT R²
  // Display the model's coefficients and intercept
  PRINT model coefficients
  PRINT model intercept
END
```

```
// Model Performance Visualization Section
BEGIN
  // Create a scatter plot
  PLOT y_test (actual values) vs. y_pred (predicted values)
  DRAW a red line representing perfect prediction
  LABEL the x-axis as "Actual Values"
  LABEL the y-axis as "Predictions"
  TITLE the plot as "Comparison of Predictions and Actual Values"
  SHOW the plot
END

// Cost Function Section
BEGIN
   // Define the linear regression cost function
   FUNCTION linear_regression_cost_function(X, y, theta)
      m = number of data points
      predictions = X * theta
      errors = predictions - y
      cost = (1 / (2 * m)) * sum of squares of errors
      RETURN cost
   END FUNCTION

   // Prepare X and y
   ADD a column of 1s to X_train for bias
   Convert y_train to an array if needed
   // Initialize parameters (theta) to zeros
   INITIALIZE theta to zeros with length equal to number of columns in X_train_with_bias
```

```
  // Calculate the initial cost
  cost = linear_regression_cost_function(X_train_with_bias, y_train_array, theta_init)
  PRINT initial cost
END

// Gradient Descent Section
BEGIN
 // Define the gradient descent function
 FUNCTION gradient_descent(X, y, theta, alpha, num_iters)
  m = number of data points
  J_history = empty list to store costs
  FOR i = 1 TO num_iters
   predictions = X * theta
   errors = predictions - y
   gradients = (1 / m) * (transpose of X) * errors
   theta = theta - alpha * gradients
   cost = linear_regression_cost_function(X, y, theta)
   append cost to J_history
    IF i is multiple of 100
     PRINT Iteration i, Current cost.
    END IF
  END FOR
  RETURN theta, J_history
 END FUNCTION

 // Normalize features in X_train
 STANDARDIZE X_train
 // Add a bias column to normalized X_train
```

```
ADD a column of 1s to X_train_normalized

// Initial parameters

INITIALIZE theta to zeros with length equal to number of columns in X_train_with_bias


// Set gradient descent parameters

SET learning rate (alpha) to 0.01

SET number of iterations (num_iters) to 600

// Run gradient descent

theta_optimized, cost_history = gradient_descent(X_train_with_bias, y_train_array,
theta_init, alpha, num_iters)

// Print the optimized parameters and final cost

PRINT optimized parameters

PRINT final cost

// Plot cost history

PLOT cost_history over iterations

LABEL the x-axis as "Iteration"

LABEL the y-axis as "Cost"

TITLE the plot as "Convergence of Gradient Descent"

SHOW the plot

  // Plot cost history and add a vertical line at the stabilization point

PLOT cost_history over iterations

LABEL the x-axis as "Iteration"

LABEL the y-axis as "Cost"

TITLE the plot as "Convergence of Gradient Descent"

DRAW a vertical dashed red line at x=500, label "Stabilization (~500)"

SHOW the plot

END
```

```
// Overfitting Management Section
BEGIN
  // Ridge Regression
  INITIALIZE Ridge model with alpha = 1
  TRAIN Ridge model on X_train and y_train
  PREDICT using Ridge model on X_test
  CALCULATE MSE for Ridge
  CALCULATE R² for Ridge
  PRINT Ridge MSE
  PRINT Ridge R²
  // Lasso Regression
  INITIALIZE Lasso model with alpha = 0.1
  TRAIN Lasso model on X_train and y_train
  PREDICT using Lasso model on X_test
  CALCULATE MSE for Lasso
  CALCULATE R² for Lasso
  PRINT Lasso MSE
  PRINT Lasso R²
  // Plot Coefficients
  GET Ridge coefficients
  GET Lasso coefficients
  PLOT Ridge coefficients
  PLOT Lasso coefficients
  TITLE the plot as "Comparison of Coefficients"
  LABEL the x-axis as "Feature Index"
  LABEL the y-axis as "Coefficient Value"
  SHOW the plot
END
```

```
// Cross-Validation Section
BEGIN
  // Initialize a Linear Regression model
  CREATE Linear Regression model
  // Perform 5-fold cross-validation
  PERFORM 5-fold cross-validation using X_train and y_train
    // Display R² scores for each fold
  PRINT R² score for each fold
  // Display the average R² score
  PRINT average R² score
END

// Model Improvement Section
BEGIN
  // Decision Tree
  INITIALIZE Decision Tree model with max_depth=3
  TRAIN Decision Tree model on X_train and y_train
  PREDICT using Decision Tree model on X_test
  CALCULATE MSE for Decision Tree
  CALCULATE R² for Decision Tree
  PRINT Decision Tree MSE
  PRINT Decision Tree R²
    // Plot Decision Tree
    PLOT the tree using features names from X_train
    SHOW the plot
  // Random Forest
  INITIALIZE Random Forest model with n_estimators=1, max_depth=3, max_features=3
and n_jobs=-1
```

```
  TRAIN Random Forest model on X_train and y_train
  PREDICT using Random Forest model on X_test
  CALCULATE MSE for Random Forest
  CALCULATE R² for Random Forest
  PRINT Random Forest MSE
  PRINT Random Forest R²
  // Plot a Tree from the Forest
    SELECT the first tree from the forest
    PLOT the tree, using feature names if X_train has them
    SHOW the plot
END

// Variable Transformation Section
BEGIN
  // Apply logarithmic transformation to SquareFeet
  CALCULATE log(SquareFeet + 1) and store in 'SquareFeet_log'
  // Apply logarithmic transformation to Price
  CALCULATE log(Price + 1) and store in 'Price_log'
  // Display the new columns
  PRINT a few rows of 'SquareFeet', 'SquareFeet_log', 'Price', 'Price_log'
END

// Model Comparison Section
BEGIN
  // Create a results summary
  CREATE a table of models, MSE, R²
    ADD "Linear Regression", MSE, R²
    ADD "Decision Tree", MSE, R²
```

```
ADD "Random Forest", MSE, R²
ADD "Ridge", MSE, R²
ADD "Lasso", MSE, R²
PRINT the table
// Cost Function calculation
// Define the Mean Squared Error (MSE) function
FUNCTION cost_function_mse(y_true, y_pred)
   RETURN the average of ((y_true - y_pred) squared)
END FUNCTION

// Define the Mean Absolute Error (MAE) function
FUNCTION cost_function_mae(y_true, y_pred)
   RETURN the average of (absolute value of (y_true - y_pred))
END FUNCTION

// Calculate MSE and MAE for each model
// Linear Regression
mse_linear = cost_function_mse(y_test, y_pred)
mae_linear = cost_function_mae(y_test, y_pred)
// Decision Tree
mse_tree = cost_function_mse(y_test, y_pred_tree)
mae_tree = cost_function_mae(y_test, y_pred_tree)
// Random Forest
mse_rf = cost_function_mse(y_test, y_pred_rf)
mae_rf = cost_function_mae(y_test, y_pred_rf)
// Ridge
mse_ridge = cost_function_mse(y_test, y_pred_ridge)
mae_ridge = cost_function_mae(y_test, y_pred_ridge)
```

```
// Lasso
  mse_lasso = cost_function_mse(y_test, y_pred_lasso)
  mae_lasso = cost_function_mae(y_test, y_pred_lasso)
 //Summarize the result
  // Create a table with Model, MSE, MAE and R²
  CREATE a new table with the following models and values :
    "Linear Regression", mse_linear, mae_linear, r2
    "Decision Tree", mse_tree, mae_tree, r2_tree
    "Random Forest", mse_rf, mae_rf, r2_rf
  PRINT the table
  // Plot a bar chart to compare the results
  //Prepare the data to plot
  DEFINE a list of models "Régression linéaire", "Arbre de décision", "Random Forest"
  DEFINE the MSE scores list : mse_linear, mse_tree, mse_rf
  DEFINE the R² scores list : r2, r2_tree, r2_rf
  // Create a bar chart
  PLOT a bar chart with models on the x-axis, MSE and R² scores on the y-axis.
  LABEL the x-axis as "Models"
  LABEL the y-axis as "Scores"
  TITLE the plot as "Comparison of Model Performances"
  SHOW the plot
 // Final features
  // Display the final features used for the training
  PRINT the features from X_train
END
```

Mathematical Expression:

Square feet $= x_1 = 1500$
Bedrooms $= x_2 = 3$
Bathrooms $= x_3 = 2$
Year Built $= x_4 = 2000$
Neighborhood 1 $= x_5 = 1$
Neighborhood 2 $= x_6 = 0$

} Features

Intercept $= \beta_0 = 12\,322,87$
Coef 1 $= \beta_1 = 99,34$
Coef 2 $= \beta_2 = 5230,28$
Coef 3 $= \beta_3 = 2968,50$
Coef 4 $= \beta_4 = -5,70$
Coef 5 $= \beta_5 = -595,53$
Coef 6 $= \beta_6 = 1416,20$

Price $= y$

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

$$\text{Price} = 12\,322,87 + (99,34 \times 1500) + (5230,28 \times 3) + (2968,50 \times 2) + (-5,70 \times 2000) + (-595,53 \times 1) + (1416,20 \times 0)$$

$$\text{Price} = 166\,355,18 \simeq 166\,355\ \$$$

Conclusion

This project explored the use of machine learning techniques to predict real estate prices based on key property attributes. Through a structured methodology, we carried out data preprocessing, feature engineering, model training, and performance evaluation.

Our initial data analysis revealed significant variations in property characteristics, including house size, number of rooms, and location type. After addressing anomalies and applying standardization techniques, we trained multiple machine learning models, with linear regression serving as the baseline approach.

The results showed that the linear regression model explained approximately 57.6% of the variance in property prices, highlighting the need for additional factors to enhance prediction accuracy. The model's coefficients aligned with expectations, confirming the impact of house size, number of rooms, and construction year on property valuation. However, the unexpected pricing trend in urban versus rural areas suggests the need for further data exploration.

To improve model performance, future enhancements could include integrating additional variables such as property amenities, economic indicators, or location-specific demand trends. Furthermore, experimenting with more advanced algorithms like Random Forest or Gradient Boosting may lead to better predictive accuracy.

Overall, this project provided valuable insights into the application of artificial intelligence in real estate pricing, reinforcing the importance of data-driven decision-making. The methodologies and findings from this study can serve as a foundation for further research and real-world implementations in property valuation systems.