Group ID: 2025-Y2-S1-MLB-B8G2-08

Group Member	IT Number
Athapaththu A.A.W.S	IT24102285
Masachchi U.R	IT24102210
Gunawardhna H.D.S.A	IT24102452
Kaldera H.P.I.D	IT24102276
Nidurshan G	IT24102266
Sandaru P.H.B	IT24102295

Introduction and Problem Statement

Predicting residential house prices is a central task in real-estate analytics with direct applications in property valuation, investment decisions, mortgage lending, and urban planning . The objective of this project is to build a robust regression model that accurately predicts house prices using the Kaggle housing.csv dataset. We aim to:

- Minimize prediction error (MSE/RMSE/MAE) on unseen data.
- Explain model decisions and surface key drivers of price.
- Identify and mitigate potential biases in the dataset and model.

Problem formulation: Given a set of features describing houses (structural attributes, location proxies, condition, etc.), estimate the target variable median house value using supervised regression.

Dataset Description

Source: Kaggle (https://www.kaggle.com/datasets/camnugent/california-housing-prices)

Size: 9+1(targe) columns and 20,640 rows

Key Features:
longitude,latitude,house_median_age,total_rooms,total_bedrooms,population,households,median_income,ocean_proximity

Target Variable: median_house_value

Relevance and Limitations: The dataset provides a wide range of structural and locational features making it suitable for price modelling.

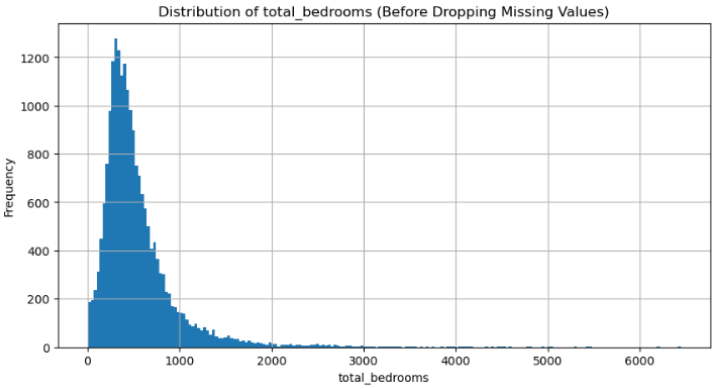
Limitations: dataset may be geographically biased (represents a specific region), may reflect historical prices (temporal bias), and may underrepresent minority housing types. Missing values and categorical levels with low frequency can also challenge generalization.

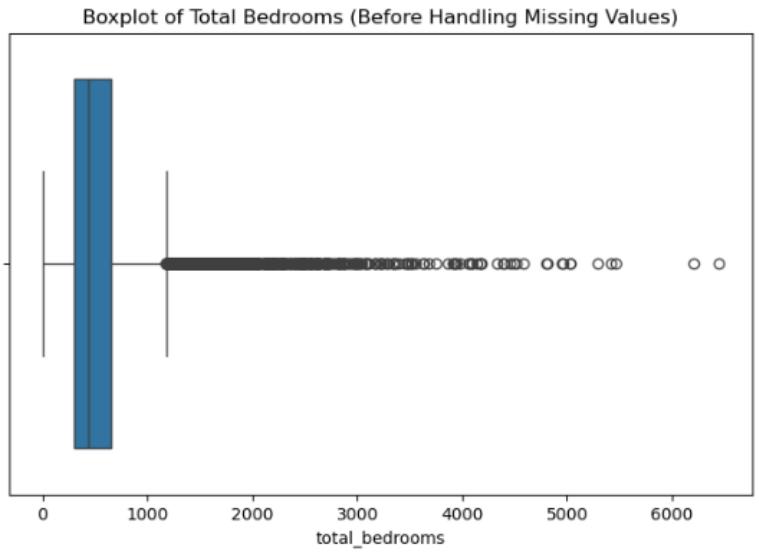
Preprocessing and Exploratory Data Analysis (EDA)

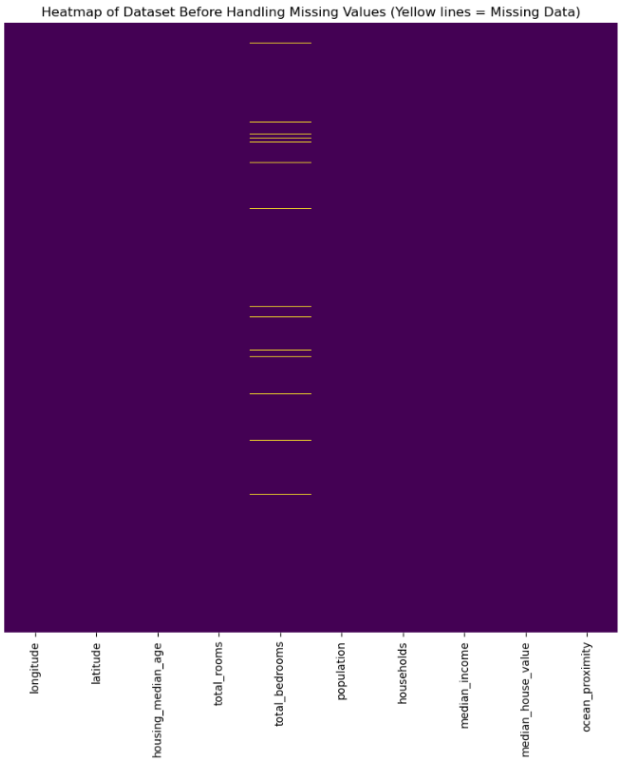
1. Handling Missing Values

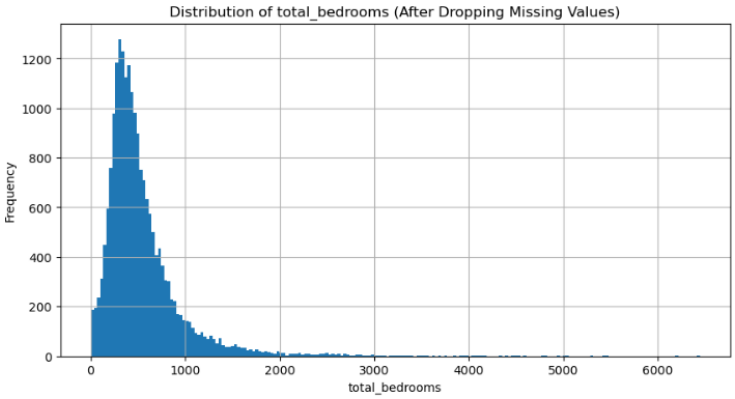
During the data preprocessing stage, we identified missing values in the dataset. Out of all the features, only total_bedrooms contained missing entries, with 207 missing values. Considering that the dataset has a total of 20,640 records, these missing values account for approximately 1% of the dataset. All other features had complete data with 20,640 entries each.

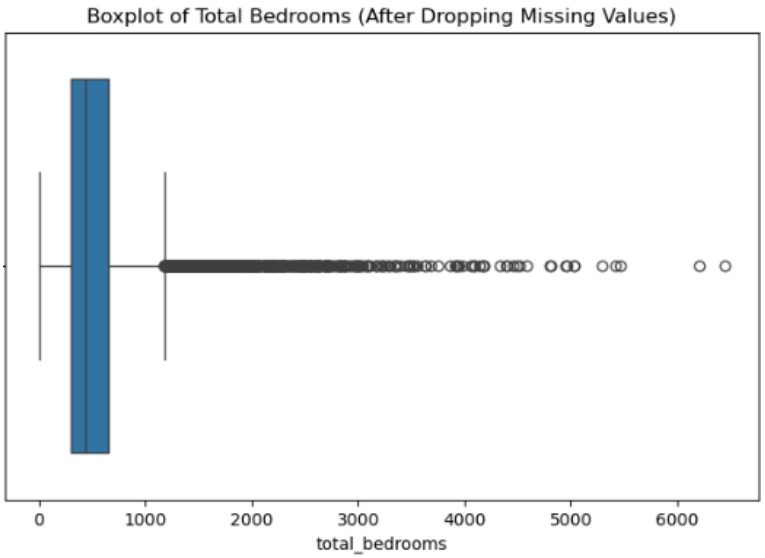
Given the minimal proportion of missing data, we decided to handle this by **dropping the rows containing missing values** [4] in the total_bedrooms column. This approach was chosen because the small number of affected rows is unlikely to significantly impact the overall analysis or model performance, while simplifying preprocessing and maintaining data integrity.

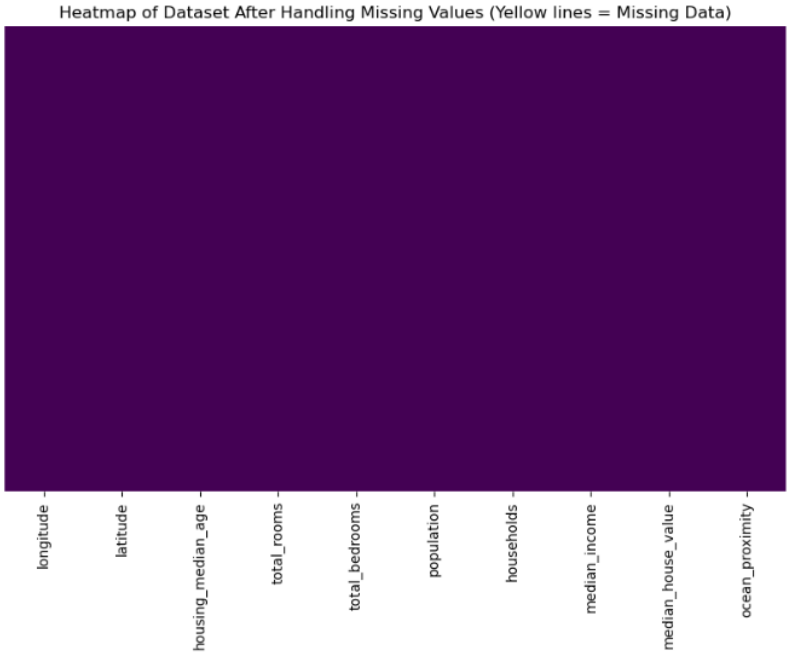








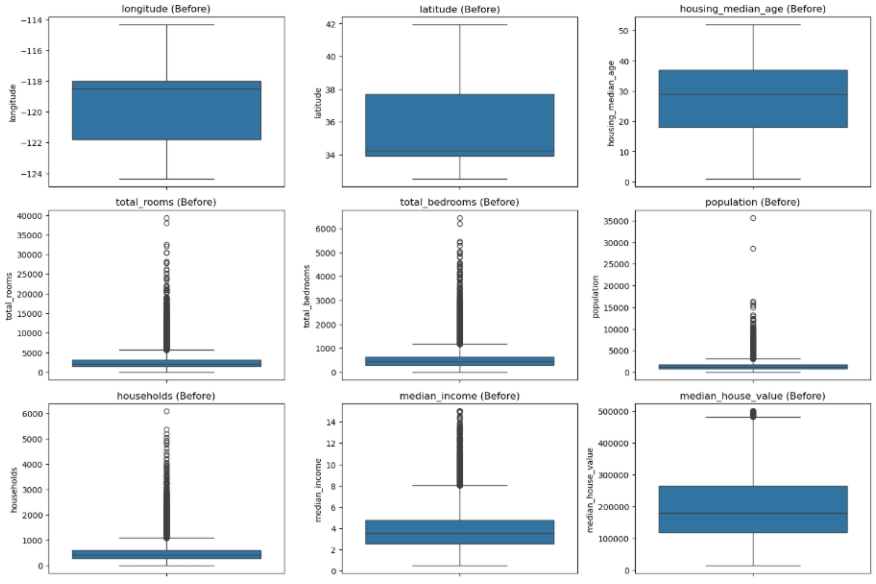


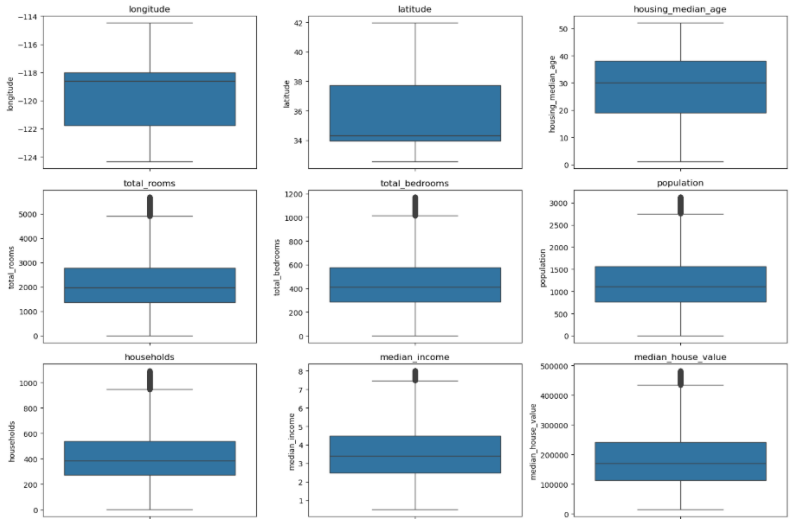


2. Outlier Detection and Removal

During exploratory data analysis, we observed that some features contained extreme outliers that were significantly distant from the majority of data points. These outliers can disproportionately influence statistical measures and machine learning model performance, leading to skewed results and reduced predictive accuracy.

To address this, we **removed the extreme outliers** from the dataset. This step helps ensure that the models learn from the typical patterns in the data rather than being biased by rare or anomalous values. Removing these extreme values improves model stability and reliability while preserving the integrity of the majority of the dataset [1].



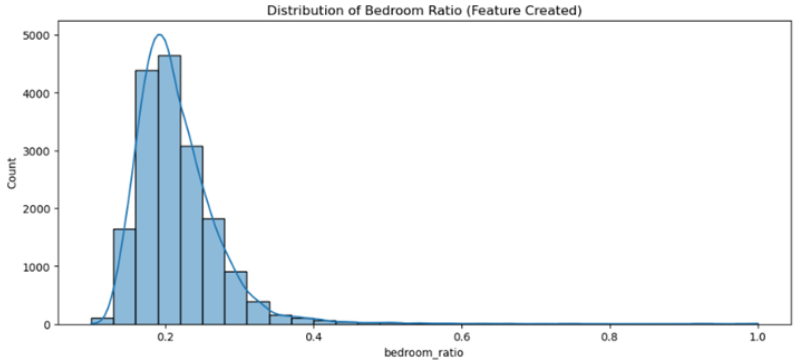


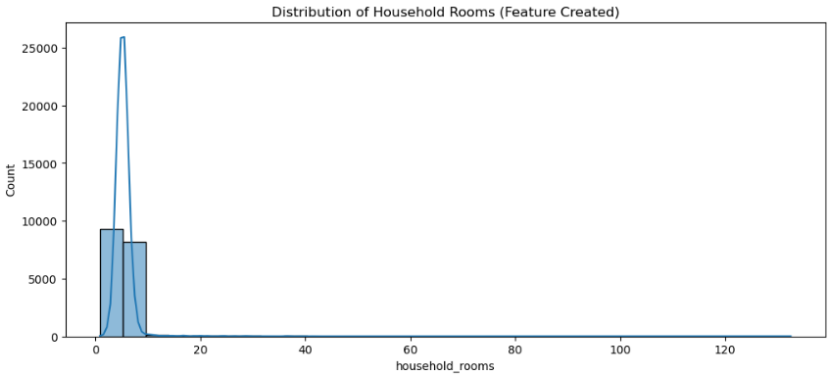
3. Feature Creation

To enhance the predictive power of the model, we created two additional features: bedroom_ratio and household_rooms.

- **Bedroom Ratio (bedroom_ratio)**: This feature was calculated as the ratio of total_bedrooms to total_rooms. It captures the proportion of bedrooms within a property, providing a normalized measure of bedroom availability relative to the overall size of the house. This helps the model better understand how the distribution of rooms affects housing prices, independent of the absolute number of rooms.
- **Household Rooms (household_rooms)**: This feature was calculated as the ratio of total_rooms to households. It represents the average number of rooms per household, offering insight into living space availability per household. This normalized metric allows the model to differentiate between densely and sparsely occupied areas, which can significantly impact house prices.

By creating these features, we introduced more meaningful, normalized representations of the original data [6] , helping the model capture relationships that may not be evident from the raw total_rooms, total_bedrooms, or households features alone.



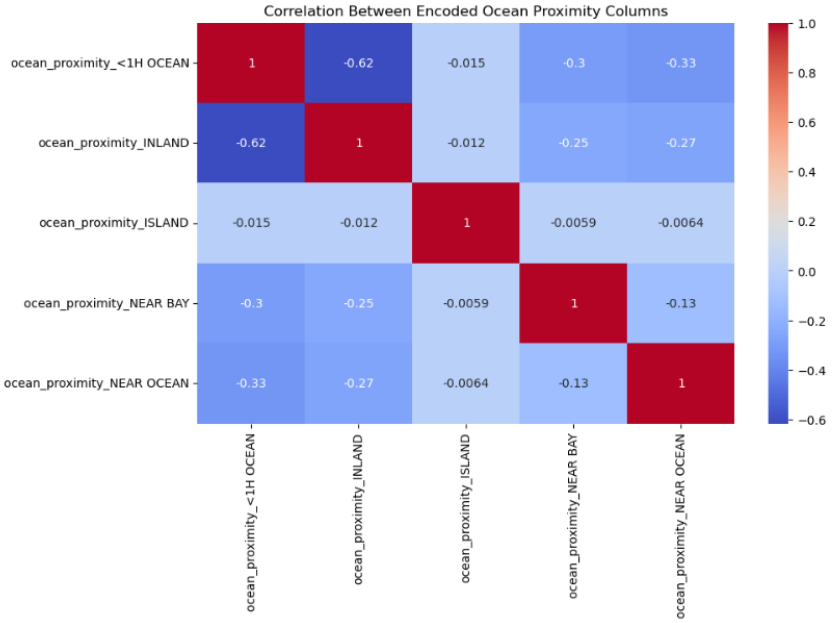


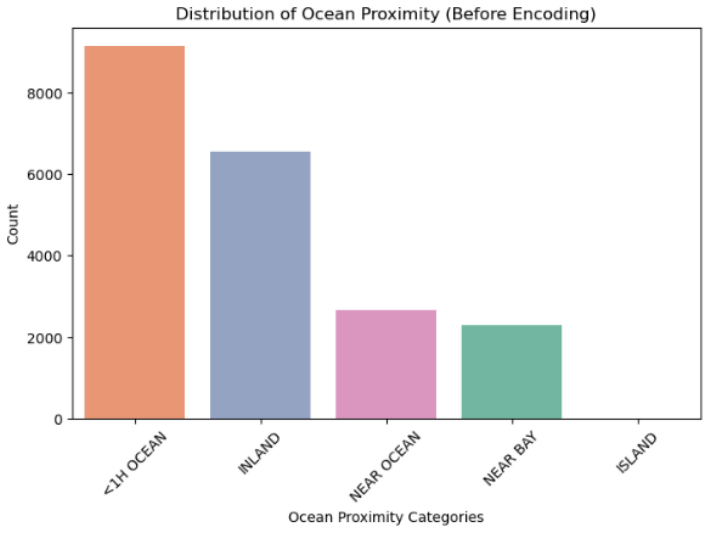
4. Encoding Categorical Variables

The dataset contains the categorical feature ocean_proximity, which represents the location of a property relative to the ocean. Machine learning models generally require numerical input, so categorical variables must be converted into a suitable numeric format.

We applied **one-hot encoding** to transform the ocean_proximity categories into binary indicator columns. This approach creates a separate column for each category, with a value of 1 indicating the presence of the category and 0 otherwise. One-hot encoding was chosen because it avoids introducing artificial ordinal relationships that could mislead the model .

This preprocessing step ensures that the location information is retained in a machine-readable format, allowing the model to leverage differences in ocean proximity without bias. Additionally, we visualized the distribution of ocean_proximity before encoding and analyzed the correlation between the encoded columns to confirm that the transformation preserved meaningful relationships among the categories.





5. Normalization

To ensure that all numerical features contribute equally to the machine learning model, we applied **Min-Max normalization** to the dataset. This scaling technique transforms each feature to a standard range of 0 to 1, calculated using the formula:

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Normalization was applied to all numerical columns except the target variable [5] median_house_value. This step is particularly important for models sensitive to the scale of features, such as distance-based algorithms or gradient-based optimization methods, as it prevents features with larger magnitudes from dominating the learning process.

Before normalization:

	min	max
longitude	-124.350000	-114.490000
latitude	32.540000	41.950000
housing_median_age	1.000000	52.000000
total_rooms	2.000000	5694.000000
total_bedrooms	2.000000	1173.000000
population	3.000000	3131.000000
households	2.000000	1092.000000
median_income	0.499900	8.011300
bedroom_ratio	0.100000	1.000000
household_rooms	0.846154	132.533333

After normalization:

	min	max
longitude	0.0	1.0
latitude	0.0	1.0
housing_median_age	0.0	1.0
total_rooms	0.0	1.0
total_bedrooms	0.0	1.0
population	0.0	1.0
households	0.0	1.0
median_income	0.0	1.0
bedroom_ratio	0.0	1.0
household_rooms	0.0	1.0

6. Feature Selection

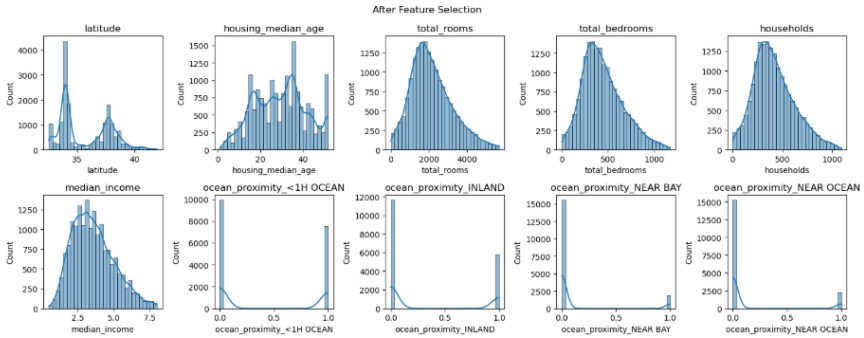
To improve model performance and reduce complexity, we performed feature selection to identify the most relevant predictors for the target variable median_house_value. Using the **SelectKBest** method with f_regression as the scoring function, we selected the top 10 numerical features that exhibit the strongest correlation with the target [3].

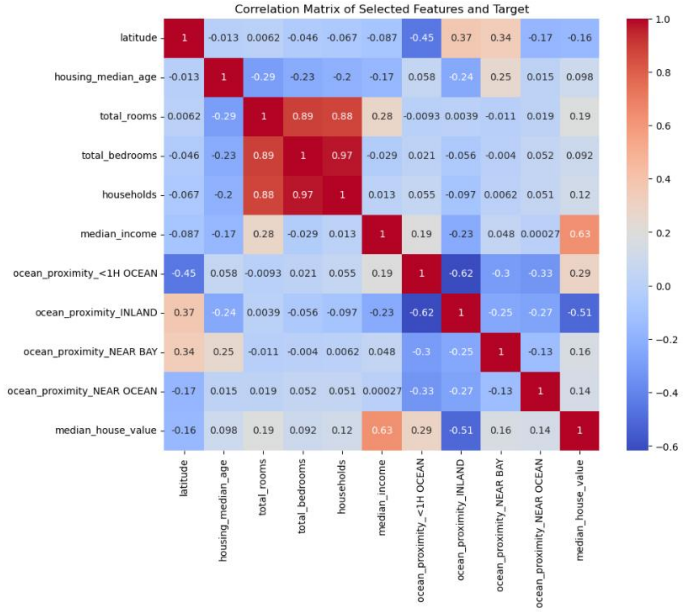
This approach helps in multiple ways:

- Reduces Dimensionality: By keeping only the most informative features, we simplify the dataset and decrease computational cost.

- Improves Model Performance: Removing irrelevant or weakly correlated features can enhance the model's predictive accuracy and reduce overfitting.
- Enhances Interpretability: Focusing on the most significant features makes the model easier to analyze and understand.

After feature selection, the dataset was reduced to these top 10 features along with the target variable, providing a more concise and relevant dataset for training the machine learning model.





Model Design and Implementation

To ensure both interpretability and performance, a diverse set of regression models was selected, ranging from simple linear approaches to advanced ensemble techniques. This allows comparison between baseline models and more complex architectures to identify the best-performing approach [2].

1. Linear Regression

- Why chosen: Serves as a baseline model to understand the linear relationship between features and the target variable.
- Advantage: Simple, interpretable, fast to train, and useful for comparison.

2. Lasso Regression

- Why chosen: Performs both regression and feature selection by applying L1 regularization.
- Advantage: Useful in reducing overfitting and removing less important features.

3. Decision Tree Regressor

- Why chosen: Captures non-linear relationships and feature interactions.
- Advantage: Easy to interpret and does not require feature scaling.

4. Random Forest Regressor

- Why chosen: An ensemble of decision trees that reduces overfitting and improves stability.
- Advantage: Handles non-linearity well and is robust to noise.

5. Gradient Boosting Regressor

- Why chosen: Builds trees sequentially by minimizing errors of the previous model.
- Advantage: High predictive performance and effective handling of complex patterns.

6. Extra Trees Regressor

- Why chosen: Similar to Random Forest but uses more randomness in feature splits.
- Advantage: Faster training and can reduce variance further than random forests.

<u>Why this combination is appropriate?</u>

- Coverage of both baseline and advanced models.
- Mix of linear and non-linear approaches.
- Regularized methods to reduce overfitting.
- Tree-based ensemble models to capture complex relationships.

- Comparative evaluation enables selecting the most optimal model.

Parameter Estimation for Random Forest Regressor

n_estimators = [100, 200, 300]
These values allow testing different ensemble sizes.

A higher number of trees generally improves performance and stability, but also increases computational cost.

The chosen range provides a balance between accuracy and efficiency.

max_depth = [None, 10, 20, 30]
Controls how deep each tree can grow.

None lets trees grow fully, capturing complex relationships.

Depth limits (10–30) help prevent overfitting and improve generalization.

min_samples_split = [2, 5, 10]
Defines the minimum number of samples needed to split a node.

Lower values allow deeper splits and more complex trees.

Higher values reduce overfitting by restricting unnecessary splits.

min_samples_leaf = [1, 2, 4]
Sets the minimum number of samples required at a terminal leaf node.

A leaf size of 1 allows maximum flexibility.

Larger leaf sizes smooth the model and reduce variance.

Parameter Estimation for Linear Regression

Linear Regression
This model does not have any tuneable parameters. The default settings from the library were used.

Ridge Regression
The alpha (regularization strength) values tested were 0.001, 0.01, 0.1, 1, 10, and 100.
Five-fold cross-validation was used to select the best model based on the lowest root mean squared error (RMSE).

Lasso Regression
The alpha values tested were 0.001, 0.01, 0.1, 1, 10, 100, and 1000.
This includes a wider range to test stronger regularization.
Five-fold cross-validation and RMSE scoring were used.

ElasticNet Regression
Two parameters were tuned:

* Alpha values: 0.001, 0.01, 0.1, 1, and 10
* L1 ratio values: 0.2, 0.5, and 0.8
Five-fold cross-validation was applied, and models were compared using RMSE.
Parameter Estimation for Lasso Regression

α (alpha) = [0.001, 0.01, 0.1, 1, 10, 100]
These values control the strength of the L1 regularization applied to the regression model.

- Lower alpha values (e.g., 0.001) allow the model to retain more features but may risk overfitting.
- Higher alpha values (e.g., 10, 100) shrink more coefficients to zero, simplifying the model and reducing overfitting.
- The chosen range allows testing different levels of regularization to find the best trade-off between bias and variance.

Coefficient Estimation

Once α is chosen, Lasso estimates the best-fitting coefficients (β) by minimizing the loss function with L1

During training:

- Lasso uses the coordinate descent optimization algorithm to estimate the best-fitting coefficients.
- When α is high, more coefficients are reduced to exactly 0 — performing automatic feature selection.
- When α is low, more features are retained, possibly improving fit but increasing the risk of overfitting.

Estimated coefficients reflect the importance of each feature:

- Positive values increase the prediction.
- Negative values decrease the prediction.
- Zero means the feature is excluded from the model.

Parameter Estimation for Decision Tree Regression

Parameters Considered

Parameter	Description	Values Tested
criterion	Function to measure the quality of a split	'squared_error'
splitter	Strategy used to choose the split at each node	'best', 'random'
max_depth	Maximum depth of the tree	None, 5, 10, 15, 20, 30
min_samples_split	Minimum number of samples required to split an internal node	2, 5, 10, 20
min_samples_leaf	Minimum number of samples required to be at a leaf node	1, 2, 5, 10
max_features	Number of features to consider when looking for the best split	None, 'sqrt', 'log2'

Method Used

- **Tool:** GridSearchCV from *scikit-learn*
- **Model:** DecisionTreeRegressor(random_state=42)
- **Evaluation Metric:** neg_root_mean_squared_error (negative RMSE — smaller values are better)
- **Cross-Validation:** 5-fold (KFold(n_splits=5, shuffle=True, random_state=42))
- **Parallel Execution:** n_jobs = -1 (utilized all CPU cores for faster computation)

The grid search exhaustively tested all combinations of hyperparameters using 5-fold cross-validation.
For each parameter set, the average RMSE across folds was computed.
The configuration with the **lowest RMSE** was selected as the **best model**.

Best Parameter Combination (Example Output)

```
Best params: {
  'criterion': 'squared_error',
```

```
    'splitter': 'best',
    'max_depth': 15,
    'min_samples_split': 5,
    'min_samples_leaf': 2,
    'max_features': 'sqrt'
}
```

Parameter Estimation for Gradient Boosting Regression

n_estimators = [100, 200, 300]
Determines the total number of boosting stages (trees) in the ensemble.
Testing across 100–300 allows the model to balance between learning capacity and efficiency.
Higher values generally enhance prediction stability and accuracy, but also increase computation time.
The chosen range provides an effective trade-off between model complexity and runtime.

learning_rate = [0.01, 0.05, 0.1]
Controls the contribution of each tree to the overall model.
Smaller values (e.g., 0.01) slow down learning, requiring more trees but improving generalization.
Larger values (e.g., 0.1) speed up learning but may risk overfitting.
The selected value of **0.05** offers a moderate learning pace that maintains both stability and accuracy.

max_depth = [3, 4, 5]
Defines how deep each individual regression tree can grow.
Shallower trees (e.g., depth 3) capture only simple relationships, while deeper trees (e.g., 5) can model more complex patterns.
The chosen **max_depth = 4** provides sufficient complexity to capture interactions in the data while preventing overfitting.

random_state = [42]
Ensures reproducibility of results by fixing the random number generator.
This allows consistent model behavior across multiple runs and comparisons among different parameter configurations.

Parameter Estimation for Extra Tree Regressor

n_estimators = [100, 200, 300]
Specifies the **number of trees** in the ensemble.

- A higher number of trees generally increases stability and accuracy by reducing variance.
- However, more trees also increase computational cost.
- The chosen range (100–300) provides a good balance between model performance and training time.

max_depth = [None, 10, 20]
Controls the **maximum depth** of each decision tree.

- None allows trees to grow fully, capturing complex, non-linear relationships.
- Shallower depths (10–20) limit tree complexity, helping prevent overfitting and improving generalization.
- Testing these values helps find the optimal trade-off between bias (too shallow) and variance (too deep).

min_samples_split = [2, 5, 10]
Determines the **minimum number of samples required to split an internal node**.

- A smaller value (2) allows deeper, more complex trees, which can capture fine-grained patterns but risk overfitting.
- Larger values (5 or 10) make the model simpler and more robust by preventing unnecessary splits.
- This parameter helps control the model's complexity and overfitting level.

min_samples_leaf = [1, 2, 4]
Defines the **minimum number of samples required at a leaf node**.

- A leaf size of 1 allows trees to fit the data very closely, increasing variance.
- Higher values (2 or 4) create smoother, more generalized predictions by reducing overfitting.
- This helps balance flexibility and stability in the final model.

Evaluation and Comparison

Below mentioned are the values for the evaluation done by the metrics (MSE,RMSE,MAE,R^2)

```
===== MODEL COMPARISON (GROUP MEMBERS) =====

Model Performance Comparison:
                                  R² Score       RMSE        MAE
Gradient Boosting (Buddhima)         0.697  51385.914  36597.980
Random Forest Regressor (Uththara)   0.681  52722.491  37765.454
Extra Trees Regressor (Imesh)        0.665  54049.108  39191.376
Linear Regression (Samadhi)          0.575  60860.840  45772.024
Decision Tree (Nidurshan)            0.333  76220.115  52370.319
Lasso Regression (Wathmavi)          0.168  85113.274  72348.275
```

This shows the comparison between the trained models.

Evaluation Interpretation

The comparison results highlight notable performance differences among the six models evaluated. Gradient Boosting achieved the highest performance overall, with an R^2 score of 0.697, indicating that it explains roughly 69.7% of the variance in the target variable. It also produced the lowest RMSE (51,385.91) and MAE (36,597.98), confirming its superior predictive accuracy and consistency across samples.

The Random Forest Regressor closely followed, recording an R^2 of 0.681, with an RMSE of 52,722.49 and MAE of 37,765.45. While slightly behind Gradient Boosting, it still demonstrated strong generalization and reliable performance, outperforming most other models.

The Extra Trees Regressor also performed competitively, achieving an R^2 of 0.665 with moderate RMSE and MAE values (54,409.18 and 39,191.38, respectively). This suggests that its ensemble averaging effectively reduced variance, though not to the same level as Gradient Boosting or Random Forest.

Linear Regression produced a mid-range R^2 score of 0.575, with comparatively higher RMSE (60,860.84) and MAE (45,772.02). This indicates that the simple linear approach could not capture complex, non-linear relationships in the dataset.

The Decision Tree displayed weaker generalization, with an R^2 of 0.333 and significantly higher error metrics (RMSE = 76,220.12, MAE = 52,370.32). This outcome suggests overfitting, where the model performs well on training data but poorly on unseen data.

Finally, the Lasso Regression showed the lowest performance, with an R^2 of 0.168, RMSE of 85,113.27, and MAE of 72,348.28, indicating that the regularization penalized coefficients too aggressively, leading to underfitting.

In summary, Gradient Boosting emerged as the most effective model, offering the best balance between accuracy and generalization. It marginally outperformed Random Forest, while both ensemble models clearly outshone the simpler linear and single-tree approaches, making Gradient Boosting the most suitable model for this regression task.

Ethical Considerations and Bias Mitigations

Potential Biases

- Geographic bias: If dataset represents a limited region, the model may not generalize to other regions with different value drivers.
- Historical bias: Historical prices reflect past discrimination or unequal investment in neighbourhoods.

Mitigation Strategies applied

- Outlier Handling (Data-Level Mitigation): Identify the extreme outliers and treat them carefully.
- Feedback Mechanism (Policy-Level Mitigation): Allow affected parties (e.g. homeowners) to challenge or feedback on predictions, especially if they believe model predictions are unfair.
- Continuous Monitoring and Retraining (Deployment Level Mitigation): Monitor distribution drift, error metrics over time, and retrain or recalibrate as new data arrives. Also monitor for new biases.
- Audit by External Users(Deployment Level Mitigation): Have third parties audit the model for fairness and bias.

These practices ensure fairness and ethical compliance in predictive modelling [7].

Reflections and Lessons Learned

Throughout this project, we gained valuable insights into the complexities of working with real-world data - particularly housing and socioeconomic data, which often carries historical and structural biases. One key takeaway was the importance of not just optimizing for model accuracy, but also being aware of ethical implications and fairness concerns, especially when working with geographic and income-based features.

As a team, we collaborated effectively by dividing responsibilities based on our individual strengths, yet we helped each other out when we came through difficulties.

As for future enhancements, we would like to try out Geospatial modelling. This will allow us to analyze spatial patterns in housing prices more effectively and create interactive map visualizations to enhance interpretability.

References

[1] Aggarwal, C. C. (2017). *Outlier analysis* (2nd ed.). Springer.

[2] Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

[3] Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*, 1157–1182.

[4] Han, J., Kamber, M., & Pei, J. (2011). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.

[5] Jain, A. K., Duin, R. P. W., & Mao, J. (2005). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(1), 4–37.

[6] Kuhn, M., & Johnson, K. (2019). *Feature engineering and selection: A practical approach for predictive models*. CRC Press.

[7] Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2019). A survey on bias and fairness in machine learning. *ACM Computing Surveys, 54*(6), 1–35.