# India Housing Price

## What Problem Does It Solve?

a. **Best Possible Location from City Name and Budget:**
   Purpose:
   To determine the optimal location within a city for a given budget.
   Implementation:
   Input: The user is prompted to enter a city code corresponding to one of the six cities.
   Data Filtering: The dataset is filtered based on the selected city.
   Price Range Determination: The minimum and maximum prices in the selected city are identified.
   Price Binning: The price range is divided into bins of 50,000 units each.
   Prediction:
   For each price bin, data within the current price range is filtered. If data exists for the current price range, features (excluding 'Price', 'City Name', and 'Location') are extracted.
   The trained model predicts the price based on these features.
   The prediction with the highest price is identified as the best possible area.
   Output:
   Each price bin displays the best possible area and features in a tabular format.
b. **Best Possible Features for Different Price Range:**
   Purpose:
   To identify the best possible features (such as the number of bedrooms, amenities, etc.) for different price ranges within a city.
   Implementation:
   Input: The user inputs the city number to focus on a specific city.
   Data Filtering: The dataset is filtered to include only data from the selected city.
   Price Binning: The data is divided into price bins of 50,000 units each.
   Feature Prediction:
   For each price bin, the dataset is filtered to include properties within the current price range. If properties exist in the current price range, their features (excluding 'Price', 'City Name', and 'Location') are used to make predictions.
   The model predicts the most valuable features based on the available data.
   The best features within the price range are identified and displayed.
   Output:
   The best possible features for each 50,000-unit price range are displayed in a tabular format.

---

## Was ML required? Where did the conventional methods fail?

a. **Conventional Methods**

Rule-Based Systems:

Approach: Manually define rules to determine the best locations and features based on historical data.

Limitations:

Requires extensive domain knowledge and time to create effective rules. Difficulty scaling and adapting to new data or changing trends. It may not capture complex patterns and interactions between features.

Statistical Analysis:

Approach: Use statistical techniques like regression analysis to model the relationship between prices and features.

Limitations:

Simple linear regression might not capture non-linear relationships effectively. Assumptions such as normality, homoscedasticity, and independence might not hold in real-world data. They need to be more expansive in handling large feature sets and complex interactions between them.

b. **Machine Learning Methods**

Flexibility:

ML models can handle a wide variety of features and capture complex non-linear relationships that are often present in housing data.

Automation:

Once trained, ML models can automate the prediction process, making it easier to handle new data without manual rule updates.

Accuracy:

ML models, especially ensemble methods like Random Forests and Gradient Boosting, often provide higher prediction accuracy than simple statistical methods.

Scalability:

ML models can easily be retrained with new data, making them adaptable to changing market trends and new data inputs.

c. **Specific Scenarios Where Conventional Methods Fail**

Handling High-Dimensional Data:

Housing data often includes many features (location, number of bedrooms, amenities, etc.). Conventional methods need help with high-dimensional data, whereas ML models are designed to handle such complexity.

Capturing Complex Interactions:

The relationship between housing prices and features is complex and often non-linear. ML models, especially non-linear models like decision trees and gradient boosting, can capture these interactions more effectively than linear statistical models.

Adaptability and Learning:

ML models can learn from data and improve over time. In contrast, rule-based systems remain static and require manual updates to reflect new trends or data patterns.

---

## Specific Models and Trade-off Between Them

**a. Linear Regression**

Description:

A simple and interpretable model that assumes a linear relationship between the features and the target variable.

Strengths:

- Easy to implement and understand.
- Computationally efficient and fast to train.
- It works well when there is a linear relationship between the features and the target.

Weaknesses:

- Struggles with non-linear relationships.
- Assumes independence of features and homoscedasticity (constant variance of errors).
- Sensitive to outliers.

**b. Decision Tree Regressor**

Description:

A non-linear model that splits the data into subsets based on feature values, making decisions at each node to minimise error.

Strengths:

- It can capture non-linear relationships and interactions between features.
- Easy to interpret and visualise.
- No need for feature scaling.

Weaknesses:

- Prone to overfitting, especially with deep trees.
- Sensitive to small changes in the data (high variance).

**c. Random Forest Regressor**

Description:

An ensemble method that combines multiple decision trees to improve accuracy and reduce overfitting.

Strengths:

- Reduces overfitting by averaging multiple trees (bagging).
- Handles non-linear relationships and feature interactions well.
- Robust to outliers and noise in the data.

Weaknesses:

- Computationally intensive and slower to train.
- It is less interpretable compared to a single decision tree.

**d. Gradient Boosting Regressor**

Description:

Another ensemble method builds trees sequentially, each trying to correct the errors of the previous one (boosting).

Strengths:

- Can achieve high accuracy by focusing on hard-to-predict cases.
- Handles non-linear relationships and feature interactions effectively.

- Typically outperforms other models in terms of predictive accuracy.

Weaknesses:

- Even more computationally intensive and slower to train than Random Forests.
- It is sensitive to hyperparameters and requires careful tuning.
- It can still overfit if not adequately regularised.

**Trade-Offs Between the Models**

a. Complexity vs. Interpretability
   - Linear Regression: Low complexity, high interpretability. It is best when the features and target relationship are approximately linear.
   - Decision Trees: Moderate complexity, moderate interpretability. It helps capture non-linear relationships but can overfit.
   - Random Forests: High complexity, lower interpretability. Better at generalisation due to reduced overfitting.
   - Gradient Boosting: Highest complexity, lower interpretability. It offers the best accuracy but at the cost of increased computational requirements and the need for tuning.

b. Training Time and Resources
   - Linear Regression: Fast to train and requires minimal computational resources.
   - Decision Trees: Relatively fast but can slow down with large datasets and deep trees.
   - Random Forests: Slower due to the training of multiple trees but parallelisable.
   - Gradient Boosting: Slowest to train due to its sequential nature but offers high accuracy.

c. Handling Non-Linear Relationships
   - Linear Regression: Ineffective for non-linear relationships.
   - Decision Trees, Random Forests, Gradient Boosting: All capable of capturing non-linear relationships, with Gradient Boosting typically offering the best performance.

d. Risk of Overfitting
   - Linear Regression: Less prone to overfitting in high-dimensional spaces but may underfit if the relationship is non-linear.
   - Decision Trees: High risk of overfitting without proper pruning.
   - Random Forests: Reduced risk of overfitting compared to individual decision trees due to averaging.
   - Gradient Boosting: There is a risk of overfitting, but it can be mitigated with regularisation techniques and careful tuning.

## Any Alternate Approach Possible

a. **Hybrid and Ensemble Methods**
   Stacking Ensemble
   Description:
   Combines multiple models by training a meta-model to aggregate their predictions.
   Strengths:
   - Can leverage the strengths of different models.
   - Often leads to improved performance.

   Weaknesses:
   - It is more complex to implement and interpret.
   - Increases computational cost and training time.

b. **Support Vector Regression (SVR)**
   Description:
   Uses support vector machines for regression tasks.
   Strengths:
   - Effective in high-dimensional spaces.
   - Works well with non-linear relationships using kernel functions.

   Weaknesses:
   - Computationally intensive for large datasets.
   - Requires careful tuning of hyperparameters (e.g., kernel type, C, epsilon).

---

## Data Collection

The dataset for each city has been taken from Kaggle from the following resources: https://www.kaggle.com/datasets/ruchi798/housing-prices-in-metropolitan-areas-of-india.

---

## Model Evaluation Metrics

a. **Mean Absolute Error (MAE):**
   The code evaluates models based on the Mean Absolute Error (MAE). This metric measures the average magnitude of errors in a set of predictions while considering their direction. It's the average of absolute differences between prediction and actual observation over the test sample, where all individual differences have equal weight.
   Usefulness: MAE is helpful because it provides a straightforward interpretation of the prediction error, clearly indicating how far off predictions are from actual values on average. Lower MAE values indicate better model performance.
   <u>Usefulness:</u>
   Interpretability → MAE is easy to interpret because it provides the average magnitude of errors in the same units as the target variable.
   Robustness to Outliers → MAE is less sensitive to outliers than MSE, making it a good choice when outliers are not a primary concern.

b. **Mean Squared Error (MSE):**
   MSE is the average of the squared differences between the predicted values and the actual values.
   <u>Usefulness:</u>
   Penalisation of Large Errors → MSE penalises more significant errors more heavily because of the squaring, making it helpful in identifying models that need to minimise substantial mistakes.
   Smooth Differentiation → MSE is smooth and differentiable, making it suitable for optimisation algorithms.

c. **R-squared:**
   $R^2$, also known as the coefficient of determination, measures the proportion of variance in the dependent variable that is predictable from the independent variables.
   <u>Usefulness:</u>
   Explained Variance → $R^2$ indicates how well the independent variables explain the variability of the dependent variable.
   Model Fit → Higher $R^2$ values indicate a better model fit to the data.

## Optimisation

a. **Use Scikit-Learn Pipelines**
   Scikit-Learn pipelines can streamline the process of transforming data and training models. This also makes the code more readable.
b. **Optimize Hyperparameter Tuning:**
   Use GridSearchCV or RandomizedSearchCV to tune hyperparameters to find the best model parameters efficiently.
c. **Code Refactoring:**
   Refactor the code to modularise different parts such as data loading, preprocessing, model training, and evaluation.