House Price Prediction (California Housing) — Model Comparison Report

1) Objective

The goal of this project is to **predict house prices** using the **California Housing dataset**. I built a baseline machine learning pipeline and then improved it using proper **data preprocessing (feature scaling)** and **model comparison**.
To find the best approach, I trained multiple regression models and compared their performance using standard evaluation metrics.

2) Dataset

This project uses the **California Housing dataset** available from scikit-learn.

- **Target (HousePrice):** Median house value for each district (this is the value we predict).
- **Features (inputs):** The dataset contains numerical features that describe each district, such as:
    - **Median Income** (important factor for pricing)
    - **House Age**
    - **Average Rooms / Bedrooms**
    - **Population**
    - **Average Occupancy**
    - **Location-related features** (Latitude and Longitude)

The dataset is already clean and numeric, so it can be used directly for regression after preprocessing.

3) Methodology

3.1 Data Loading and Splitting (X and y):

- The dataset was loaded using fetch_california_housing.
- Features were stored in **X** and the target house price values were stored in **y**.
- The data was split into: **80% training data / 20% testing data**
  using train_test_split(test_size=0.2, random_state=42).

This ensures the model is trained on one part and evaluated on unseen data (test set).

3.2 Feature Scaling (StandardScaler):

I applied **StandardScaler** to the input features.
StandardScaler converts each feature to a similar scale by making:

- mean ≈ 0
- standard deviation ≈ 1

Why scaling is needed:
Some models (especially Linear and Ridge Regression) perform better when features are on similar ranges. Without scaling, features with large values may dominate the learning process, which can reduce accuracy and stability.

Important note:
To avoid data leakage, the scaler was:

- **fit only on the training data**
- then used to transform both training and testing data.

3.3 Models Trained:

I trained and compared the following regression models:

1. **Linear Regression**
    - A simple baseline model that assumes a linear relationship between features and house price.
2. **Ridge Regression**
    - Similar to Linear Regression but includes **regularization** (penalty term) to reduce overfitting and improve generalization.
3. **Decision Tree Regressor**
    - A non-linear model that can capture complex patterns by splitting data into rules (if-else conditions).

3.4 Evaluation Metrics:

I evaluated each model using:

- **RMSE (Root Mean Squared Error):** lower is better (less prediction error)
- **R^2 Score:** higher is better (explains more variance in house prices)

4) Results

4.1 Model Comparison Table

Model	RMSE	R^2
Decision Tree	**0.702829**	**0.623042**
Ridge Regression	**0.745557**	**0.575816**
Linear Regression	**0.745581**	**0.575788**

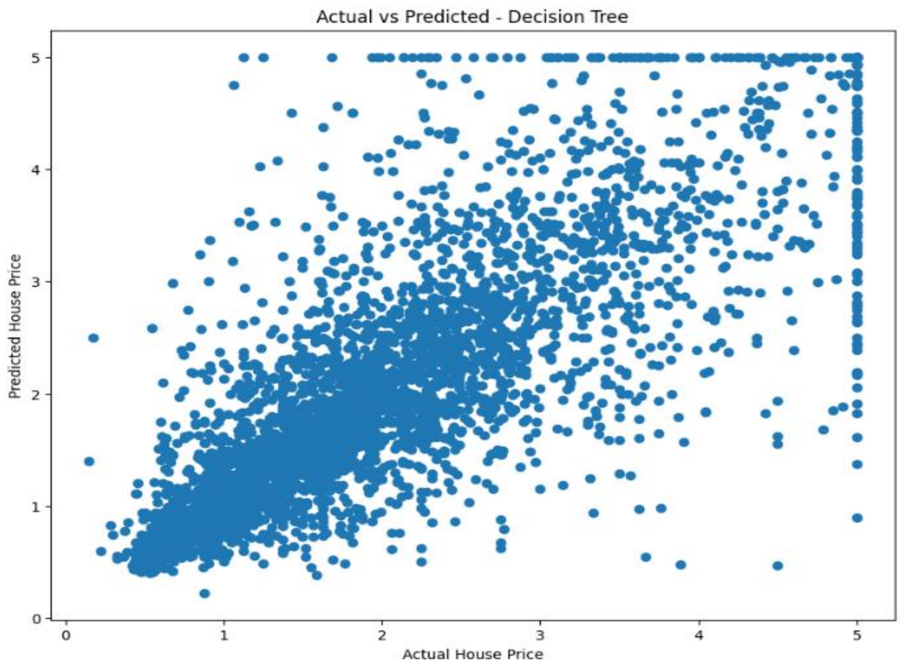
4.2 Best Model

The **Decision Tree Regressor** performed best because it had:

- **Lowest RMSE = 0.702829**
- **Highest R^2 = 0.623042**

This indicates the Decision Tree captured more complex (non-linear) patterns in the dataset compared to Linear and Ridge regression models.

I plotted **Actual vs Predicted** values for the best model (Decision Tree).
In a perfect model, points would lie close to a diagonal line (Actual ≈ Predicted). The scatter plot shows a clear positive trend, meaning the model predictions generally follow the true values, though there is still spread (error), especially at higher prices.



Caption: *Figure 3: Actual vs Predicted plot for the best model (Decision Tree).*

5) Conclusion

In this project, I built an end-to-end regression pipeline to predict California house prices. The workflow included data loading, splitting into train/test sets, feature scaling using StandardScaler, training multiple models, and evaluating them using RMSE and R^2.

Based on the results, the **Decision Tree Regressor** was selected as the best model because it achieved the **lowest error (RMSE 0.702829)** and the **highest R^2 (0.623042)** among the tested models.

Possible Improvements (Future Work)

To improve performance further, the following can be done:

- **Hyperparameter tuning** (e.g., tree depth, min samples split)
- Use **cross-validation** for more stable evaluation
- Try more powerful models like **Random Forest**, **Gradient Boosting**, or **XGBoost**
- Add **feature engineering** to create new informative features