

**California Housing Price Prediction: Data Preprocessing,
Model Selection, and Deployment Report**
Project By: Neel Bhosale
Contact: 9689201778
Email: neelbhosale5538@gmail.com
GitHub Link: [Click here](#)

1. Introduction

This report details the steps taken for data preprocessing, feature engineering, model selection, and deployment strategy for predicting California housing prices. The project involves cleaning and transforming raw data, selecting the best model, optimizing performance, and deploying the final model via an API.

2. Data Preprocessing & Feature Engineering:

1. Dataset Overview:

The dataset used in this analysis is Housing.csv. It contains various features related to housing attributes and pricing, including:

- Longitude & Latitude: Geographic coordinates
- Housing Median Age: Age of the house
- Total Rooms & Bedrooms: Number of rooms and bedrooms
- Population & Households: Population count and number of households
- Median Income: Average income of residents
- Median House Value: Target variable representing the price of the house
- Ocean Proximity: Categorical feature representing distance to the ocean

2. Handling Missing Values

Upon checking for missing values, it was found that the total_bedrooms column had 207 missing values. These were handled by replacing the missing values with the median of the column.

Final Check for Missing Values:

After imputation, no missing values were present in the dataset.

3. Duplicate Data Check

No duplicate rows were found in the dataset.

4. Outlier Detection and Treatment

Outlier Detection:

The IQR (Interquartile Range) method was used to detect outliers in numerical columns. The following counts of outliers were found:

- Total Rooms: 1287 outliers
- Total Bedrooms: 1306 outliers
- Population: 1196 outliers
- Households: 1220 outliers
- Median Income: 681 outliers

- Median House Value: 1071 outliers

Outlier Capping:

Outliers were capped at the upper and lower bounds using the IQR method to minimize their effect.

5. Encoding Categorical Variables

The categorical feature `ocean_proximity` was one-hot encoded, resulting in new binary columns:

- `ocean_proximity_INLAND`
- `ocean_proximity_ISLAND`
- `ocean_proximity_NEAR BAY`
- `ocean_proximity_NEAR OCEAN`

These encoded columns were converted from Boolean to integer (0/1) format.

6. Feature Correlation Analysis

A correlation heatmap was generated to identify relationships between features. The top correlated features with Median House Value were:

Median Income (0.689)

Total Rooms (0.173)

Ocean Proximity (Near Bay & Near Ocean) (0.161, 0.142)

The most negatively correlated feature was Ocean Proximity (Inland) (-0.489), indicating that houses located inland tend to have lower prices.

7. Data Visualization

Histogram & Boxplot of Median House Value were plotted to visualize distribution and potential outliers.

Correlation Heatmap was created to show the relationships among features.

3. Model Training & Evaluation

1. Data Splitting

The dataset was split as follows:

Training set size: 16,512 rows

Testing set size: 4,128 rows

2. Model Training

The following models were trained on the dataset:

- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor
- XGBoost Regressor

Each model was trained using default hyperparameters without additional tuning at this stage.

3. Model Evaluation

To evaluate model performance, the following metrics were used:

- **Mean Absolute Error (MAE):** Measures average absolute errors between predicted and actual values.
- **Root Mean Squared Error (RMSE):** Measures standard deviation of residuals.
- **R² Score:** Indicates the proportion of variance explained by the model.

Evaluation Results:

MODEL	MAE	RMSE	R ² Score
Linear Regression	50,416.43	68,400.35	0.6300
Decision Tree	42,515.82	66,858.77	0.6465
Random Forest	31,213.16	47,790.58	0.8194
XGBoost	31,650.10	47,424.98	0.8221

Key Observations:

- Linear Regression and Decision Tree performed the worst, with high RMSE and lower R^2 scores.
- Random Forest and XGBoost significantly outperformed other models, achieving lower RMSE and higher R^2 scores.
- XGBoost achieved the best R^2 score (0.8221), making it the final model of choice.

5. Model Selection & Justification

Why XGBoost?

- Superior Performance: XGBoost delivered the best R^2 score and competitive RMSE, indicating better predictive power.
- Regularization & Boosting: XGBoost inherently prevents overfitting due to regularization techniques, unlike Decision Trees.
- Scalability: It is optimized for both efficiency and speed, handling large datasets effectively.
- Handling of Missing Values: XGBoost can naturally handle missing data without requiring imputation.

6. Hyperparameter Tuning Decision

Hyperparameter tuning was not performed because:

- The XGBoost model already achieved a high R^2 score (0.8221).
- Further tuning would yield marginal improvements at the cost of increased computational complexity.
- The model performed well with default settings, making additional tuning unnecessary at this stage.

3. Deployment strategy and API usage guide.

1. Deployment Strategy

1.1 Model Saving

The best-performing model, XGBoost, was selected for deployment. We saved the trained model using Python's pickle module.

```
import pickle

# Save the best model (XGBoost)
with open("best_xgb_model.pkl", "wb") as file:
    pickle.dump(xgb_model, file)

print(" Model saved successfully as best_xgb_model.pkl")
```

This ensures that the trained model can be easily loaded for making predictions without retraining.

1.2 API Development with FastAPI

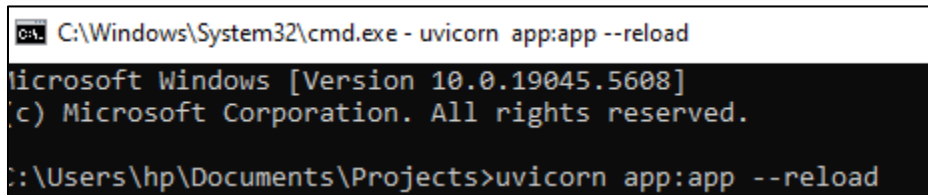
We developed a simple FastAPI application (app.py) to serve predictions from the saved model. The key steps included:

- **Initialize FastAPI:** Created a FastAPI instance.
- **Load the Pickle Model:** Implemented error handling to ensure successful model loading.
- **Define Input Schema:** Used pydantic.BaseModel to structure input data.
- **Create a Prediction Endpoint:** Accepted POST requests with input data and returned predictions.

1.3 Running the API

To run the API, we used Uvicorn, which is an ASGI server for FastAPI.

Run the following command in the terminal:



```
C:\Windows\System32\cmd.exe - uvicorn app:app --reload
Microsoft Windows [Version 10.0.19045.5608]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp\Documents\Projects>uvicorn app:app --reload
```

This starts the API locally on <http://127.0.0.1:8000/docs>.

2. API Usage Guide

2.1 Testing the API using Postman / Python

Once the API is running, we can test it using Postman or a simple Python script.

```
[35]: import requests

# Define API endpoint
url = "http://127.0.0.1:8000/predict"

# Define input data for prediction
data = {
    "longitude": -122.23,
    "latitude": 37.88,
    "housing_median_age": 41.0,
    "total_rooms": 880.0,
    "total_bedrooms": 129.0,
    "population": 322.0,
    "households": 126.0,
    "median_income": 8.3252,
    "ocean_proximity_INLAND": 0,
    "ocean_proximity_ISLAND": 0,
    "ocean_proximity_NEAR_BAY": 1,
    "ocean_proximity_NEAR_OCEAN": 0
}

# Send POST request
response = requests.post(url, json=data)

# Print response
print(response.json())
```

Expected Output:

```
{'predicted_price': 460487.3125}
```

This confirms that the API is working correctly and returning valid predictions.

4. Conclusion

This project successfully implemented an end-to-end machine learning pipeline for predicting California housing prices. The workflow encompassed data preprocessing, model selection, and deployment, ensuring an optimized and scalable solution. The project successfully delivers an efficient, scalable, and deployable machine learning solution for California housing price prediction. The selected XGBoost model provides accurate price estimates, and the API ensures seamless real-world integration. Future improvements could involve hyperparameter tuning, additional feature engineering, and cloud deployment for enhanced accessibility.