California Housing Price Prediction Project

1. Introduction

The **California Housing Price Prediction** project aims to estimate house prices based on multiple factors, including geographical location, population, and median income. The model is trained using **scikit-learn** and deployed using **Flask** on **Render**, making it accessible through a web interface and API.

This report outlines the steps taken for **data preprocessing, feature engineering, model selection, optimization, and deployment**.

---

2. Data Preprocessing and Feature Engineering

2.1 Dataset Overview

The dataset used for this project is derived from the **California housing dataset**, which includes features such as:

- **Longitude & Latitude** (Geographical coordinates)
- **Housing Median Age**
- **Total Rooms & Total Bedrooms**
- **Population & Households**
- **Median Income** (Primary factor influencing house prices)
- **Ocean Proximity** (Categorical variable)

2.2 Data Cleaning

To prepare the dataset, the following preprocessing steps were applied:

- **Handling Missing Values**: Missing values in total_bedrooms were filled using the **median value**.
- **Outlier Detection & Removal**: Outliers in total_rooms, total_bedrooms, population, households, median_income, and median_house_value were identified and removed using the **datasist library**.
- **Data Type Handling**: The categorical feature ocean_proximity was **encoded using Label Encoding** to convert it into numerical format.

2.3 Feature Engineering

- **Feature Correlation Analysis**: A **heatmap** was used to visualize relationships between variables.
- **Histogram & Pair Plot**: Plotted for understanding feature distributions.
- **Standardization**: Used **StandardScaler** to scale numerical features.

---

3. Model Selection and Optimization

3.1 Model Selection

Several regression models were tested:

- **Linear Regression**
- **Ridge & Lasso Regression**
- **Decision Tree Regressor**
- **Random Forest Regressor**
- **Gradient Boosting Regressor** (Best-performing model)
- **Support Vector Regressor (SVR)**
- **K-Nearest Neighbors (KNN) Regressor**

3.2 Hyperparameter Tuning

Each model was tuned using **GridSearchCV** with a 5-fold cross-validation.

- **Ridge Regression**: Tuned alpha values.
- **Lasso Regression**: Tuned alpha values.
- **Decision Tree**: Tuned max_depth and min_samples_split.
- **Random Forest**: Tuned n_estimators and max_depth.
- **Gradient Boosting**: Tuned n_estimators, learning_rate.
- **SVR**: Tuned C and kernel type.
- **KNN**: Tuned n_neighbors.

3.3 Model Performance Metrics

Each model was evaluated using:

- **R^2 Score (Coefficient of Determination)**
- **Mean Absolute Error (MAE)**
- **Root Mean Squared Error (RMSE)**

The **Gradient Boosting Regressor** had the amongst **highest R^2 score** with stable results for all the cross validations and was selected as the final model.

3.4 Model Saving

The trained **best model** was saved as a **Pickle file (best_model.pkl)** using joblib for easy deployment.

---

4. Deployment Strategy

4.1 Web Framework: Flask

The Flask framework was used to develop a REST API to serve predictions. The API receives input from the frontend, processes it using the trained model, and returns predicted house prices.

4.2 Deployment on Render

The application is hosted on **Render**, a cloud-based deployment platform. The deployment steps include:

1. **Pushing the project to GitHub**.
2. **Connecting the repository to Render**.
3. **Setting up a Flask web service with Gunicorn for production**.
4. **Running startup.sh to start the Flask app**.

The deployment link for this project is : https://california-housing-fqqt.onrender.com

4.3 Folder Structure for Deployment

```
/california-house-price-prediction
│-- /templates            # Contains the frontend HTML files
│   ├── index.html         # Main UI file
│-- app.py                # Flask application
│-- best_model.pkl         # Trained ML model
│-- requirements.txt       # Python dependencies
│-- startup.sh            # Start script for Render deployment
│-- README.md              # Project documentation
```

---

5. API Usage Guide

5.1 API Endpoints

1 Home Route (/)

- **Method:** GET
- **Description:** Serves the frontend (index.html).

2 Prediction Route (/predict)

- **Method:** POST
- **Input (JSON):**

```
{
    "features": [longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, "ocean_proximity"]
}
```

- **Output (JSON):**

```
{
  "predicted_price": 350000.0
}
```

5.2 Testing the API

To test the API locally, use:

```
curl -X POST "http://127.0.0.1:5000/predict" \
    -H "Content-Type: application/json" \
    -d '{"features": [ -122.23, 37.88, 41, 880, 129, 322, 126, 8.3252, "NEAR BAY"] }'
```

---

6. Conclusion

This project demonstrates the **end-to-end workflow of a machine learning application**, from **data preprocessing** to **deployment on Render**. The Flask API makes it easy to integrate with different front ends and applications.

Future improvements include:

- **Adding more advanced feature selection techniques.**
- **Deploying the model as a microservice using Docker.**
- **Improving UI/UX for better user experience.**

🚀 **Project Repository:** GitHub

---

7. Author

Umashankar Kandpal

📧 Contact: umashankarkandpal612@gmail.com