California Housing Price Prediction Report

1 Data Preprocessing and Feature Engineering

1.1 Data Loading and Initial Analysis

- The dataset was fetched using `fetch_california_housing()` from `sklearn.datasets`.
- Converted the dataset into a Pandas DataFrame.
- Added the target variable (`Target`) to the dataset.
- Conducted an initial exploration with `.head()`, `.info()`, and `.describe()` to understand the dataset structure.

1.2 Handling Missing Values

- Verified that no missing values existed using `df.info()`.
- If missing values had been present, they would have been filled using median values (`df.fillna(df.median())`).

1.3 Feature Engineering

- Created two additional meaningful features:
  - **PopulationPerHousehold** = Population / AveOccup
  - **RoomsPerPerson** = AveRooms / Population
- Standardized all features using `StandardScaler()` to ensure uniformity and prevent any single feature from dominating.

1.4 Correlation Analysis

- Computed the correlation matrix and visualized it with a heatmap.
- Identified features most correlated with the target variable (`MedInc`, `AveRooms`, `HouseAge`).
- Created scatter plots to observe trends between these features and the target variable.

2 Model Selection and Optimization

2.1 Model Choice

- **Random Forest Regressor** was chosen for its capability to handle non-linear relationships and robustness in structured tabular data.

2.2 Hyperparameter Tuning

- Used `GridSearchCV` with 5-fold cross-validation to optimize:
  - `n_estimators`: Number of trees in the forest ([50, 100])
  - `max_depth`: Maximum depth of trees ([None, 10, 20])
  - `min_samples_split`: Minimum samples required to split an internal node ([2, 5])
- The best parameters found: {`max_depth: None, min_samples_split: 2, n_estimators: 100`}.

2.3 Model Evaluation

- The model was evaluated using:
  - **Root Mean Squared Error (RMSE)**: 0.5074
  - **Mean Absolute Error (MAE)**: 0.3295
  - **R^2 Score**: 0.8035
- The trained model was saved using `joblib.dump()`.

3 Deployment Strategy and API Usage

3.1 Deployment with Flask

- A Flask API was created (`myapp.py`) to expose the trained model for real-time predictions.
- The model (`california_rf_model.joblib`) was loaded within the API.
- A list of expected features was predefined.
- Default median values were used if any input was missing.

3.2 API Endpoint

- **Endpoint**: `/predict` (POST request)
- Accepts JSON input with the following features:

```
{
    "MedInc": 3.5,
    "HouseAge": 30,
    "AveRooms": 5.0,
    "AveBedrms": 1.1,
    "Population": 1000,
    "AveOccup": 2.5,
    "Latitude": 34.0,
    "Longitude": -118.0,
    "PopulationPerHousehold": 3.0,
    "RoomsPerPerson": 1.2
}
```

3.3 Running the Flask API

1. Install dependencies:

   ```
   pip install flask joblib numpy
   ```

2. Start the API:

   ```
   python myapp.py
   ```

3. Send a Postman POST request:
4. Expected Output:

   ```
   {"prediction": 2.45}
   ```

4 Bonus Point Work

- I created a good looking frontend UI to interact with the model.
- Implemented logging and error handling in the API.

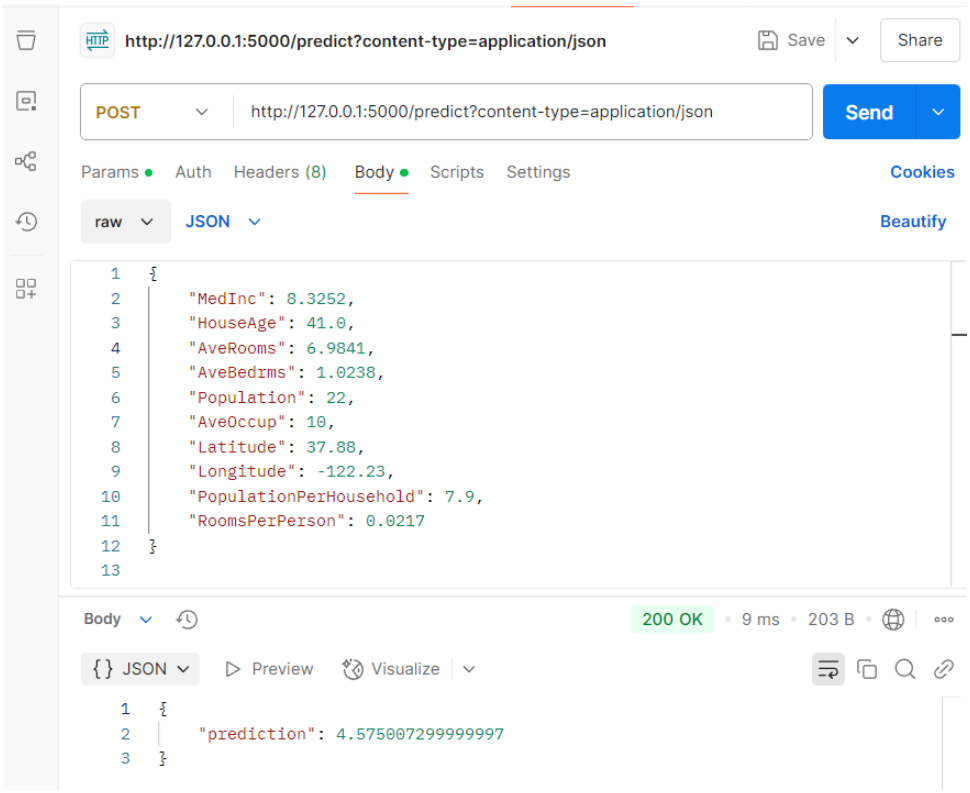

Figure 1: Postman Image

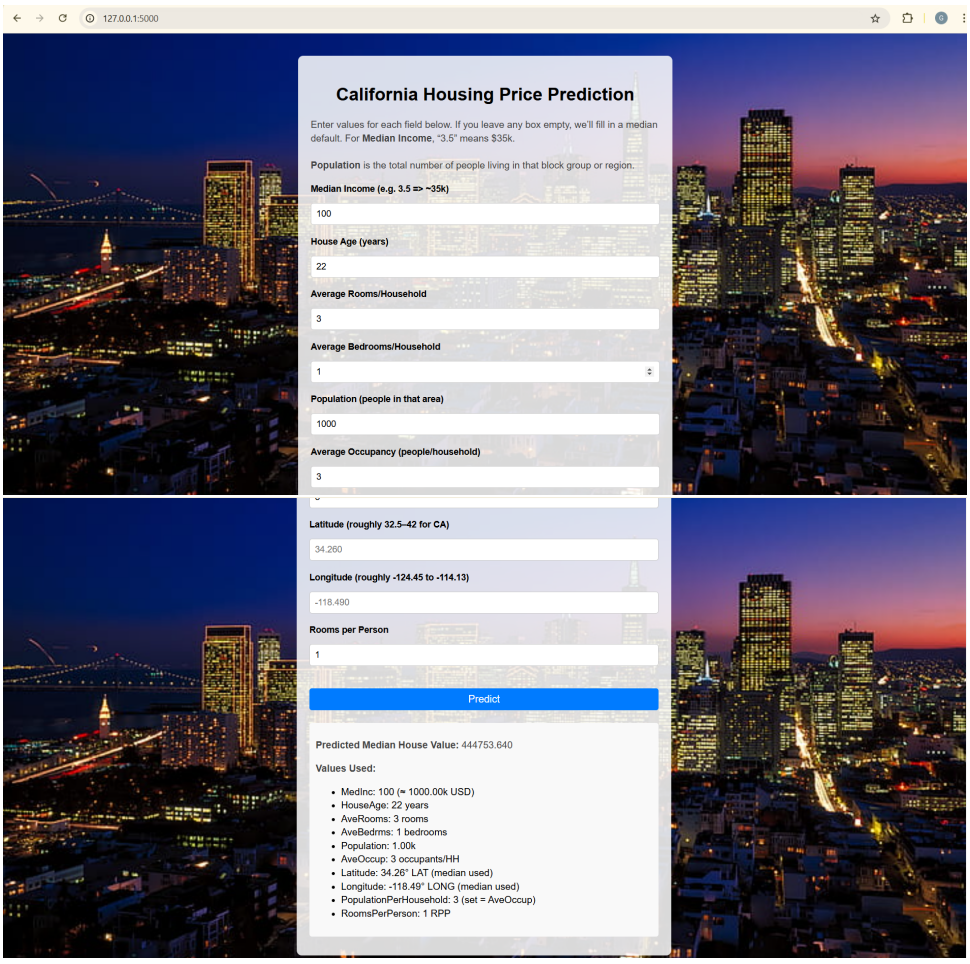

Figure 2: UI of my App.