California Housing Price Prediction Model: Technical Report

This report provides a comprehensive overview of the California housing price prediction model developed and deployed as part of the Machine Learning Engineer assessment task. The project demonstrates a complete machine learning pipeline from data preprocessing to model deployment with a RESTful API.

Data Preprocessing and Feature Engineering

Dataset Overview

The California Housing dataset from scikit-learn was used for this project. This dataset contains information collected from the 1990 California census, featuring 20,640 observations with 8 features that describe housing blocks across California. The target variable is the median house value for California districts.

Exploratory Data Analysis

Initial exploration revealed several key insights about the dataset:

- No missing values were present in the dataset, which simplified preprocessing
- The dataset includes features like median income, house age, average rooms, bedrooms, population, occupancy, and geographic coordinates
- Significant variation in housing prices was observed across different regions, with coastal areas showing higher median prices
- Distribution analysis showed some features (like Population and AveOccup) have outliers that required attention

Feature Engineering

Based on correlation analysis and domain knowledge, the following feature engineering steps were implemented:

- Created a 'BedroomRatio' feature (AveBedrms / AveRooms) to capture the proportion of bedrooms to total rooms
- Added 'RoomsPerPerson' feature (AveRooms / AveOccup) to represent housing density
- Standardized all features using StandardScaler to ensure comparable scale for model training

These engineered features improved model performance by providing more contextual information about the housing data. Correlation analysis showed that MedInc (median income) had the strongest correlation with housing prices, followed by geographic features.

Model Selection and Optimization Approach

Model Evaluation

Multiple regression models were trained and evaluated to identify the best performer:

- Linear Regression (baseline)
- Ridge Regression
- Lasso Regression
- Random Forest Regressor
- Gradient Boosting Regressor
- Deep Neural Networks

Each model was evaluated using multiple metrics including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R^2 score.

Hyperparameter Tuning

The Random Forest model emerged as the best performer during initial testing and was further optimized using **RandomizedSearchCV** for broad parameter space exploration

The key hyperparameters tuned were:

- n_estimators: Number of trees in the forest
- max_depth: Maximum depth of trees
- min_samples_split: Minimum samples required for a split
- min_samples_leaf: Minimum samples required at a leaf node
- max_features: Number of features to consider for best split

Final Model Performance

Among all the models, Random Forest model achieved best results:

- R^2 score: 0.81 on the test set
- RMSE: 0.50 (in scaled units of median house value)
- MAE: 0.31 (in scaled units of median house value)

The model was saved using pickle to preserve the trained model and scaler for deployment.

Deployment Strategy and API Usage Guide

Flask API Implementation

A REST API was developed using Flask to serve predictions from the model. The API includes:

- A `/predict` endpoint that accepts JSON input containing housing features
- Proper error handling for invalid inputs
- Logging for both successful predictions and errors

The API implementation properly applies the same feature engineering and scaling transformations as used during training.

```
@app.route("/predict", methods=["POST"])
def predict():
    try:
        # Get JSON data from request
        data = request.get_json()

        # Convert input data to DataFrame (ensure correct feature order)
        input_df = pd.DataFrame([data], columns=original_features)

        # Apply the same feature engineering as training
        input_df["BedroomRatio"] = input_df["AveBedrms"] / input_df["AveRooms"]
        input_df["RoomsPerPerson"] = input_df["AveRooms"] / input_df["AveOccup"]

        # Scale features using the same scaler
        input_scaled = scaler.transform(input_df)

        # Make prediction
        prediction = model.predict(input_scaled)

        # Return prediction
        return jsonify({"predicted_price": float(prediction[^0])})

    except Exception as e:
        return jsonify({"error": str(e)})
```

API Testing Guide

Repository Structure Overview

This repository contains a complete machine learning pipeline for predicting California house prices. The key components include:

- `app.py`: Flask API implementation for model serving
- `web_app.py`: Streamlit web application for user-friendly interaction
- `california_housing_prediction.ipynb`: Jupyter notebook with model development
- `rf_model.pkl`: Serialized Random Forest model weights
- requirements.txt: Necessary libraries to run the code.

Cloning and Setup Instructions

Start by cloning the repository to your local machine:

```
git clone https://github.com/Rahulverma5/California-house-price-prediction-tutorial.git
cd California-house-price-prediction-tutorial
```

Install the required dependencies:

```
pip install -r requirements.txt
```

Running the Flask API

Launch the Flask API server:

```
python app.py
```

This runs the Flask server on port 5000, making the model accessible via HTTP requests. The API exposes a `/predict` endpoint which accepts JSON data containing housing features and returns predicted prices.

Testing the API

API accepts POST requests with JSON data containing the following features:

- MedInc: Median income in block group
- HouseAge: Median house age in block group
- AveRooms: Average number of rooms per household
- AveBedrms: Average number of bedrooms per household
- Population: Block group population
- AveOccup: Average number of household members
- Latitude: Block group latitude
- Longitude: Block group longitude

You can test the API using curl or Postman. Here's a curl example:

```
curl -X POST http://localhost:5000/predict \
  -H "Content-Type: application/json" \
  -d '{
    "MedInc": 8.3252,
    "HouseAge": 41.0,
    "AveRooms": 6.984127,
    "AveBedrms": 1.023810,
    "Population": 322.0,
    "AveOccup": 2.555556,
    "Latitude": 37.88,
    "Longitude": -122.23
  }'
```

Example response:

{

"predicted_price": 4.526

}

The API processes these input features through the Random Forest model and returns a JSON response with the predicted house price. The model expects these specific features as they were used during training to capture various aspects of housing data like income levels, house age, room counts, and geographic coordinates.

For the interactive web interface, run the Streamlit app with:

```
streamlit run web_app.py
```

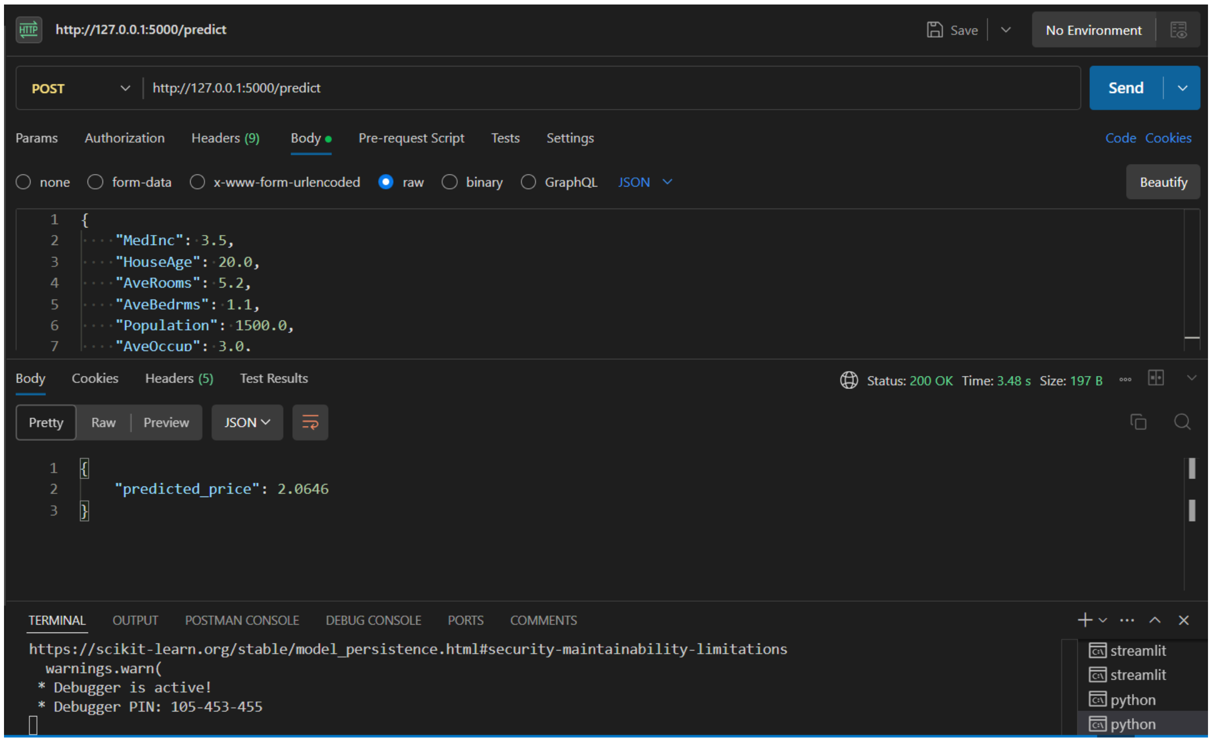
This provides a user-friendly interface where users can adjust input parameters using sliders and visualize predictions without writing code.

Multi-platform Deployment

The model was deployed using two complementary approaches:

1. **Flask API for Programmatic Access**:
   - Enables integration with other systems and applications
   - Provides a standard REST interface for batch predictions
   - Tested successfully with **Postman** for API validation (figure 1)
   - Can be containerized with Docker for scalable deployment
2. **Hugging Face Spaces with Streamlit UI**:
   - Provides an interactive web interface for non-technical users (figure 2). It can be accessed here: **https://huggingface.co/spaces/rahullverma/California-House-Price-Predictor**
   - Allows users to adjust feature values using sliders and input fields.
   - Visualizes prediction results with contextual information.

This dual deployment strategy ensures the model is accessible both to end-users through a friendly interface and to developers or systems that need programmatic access.

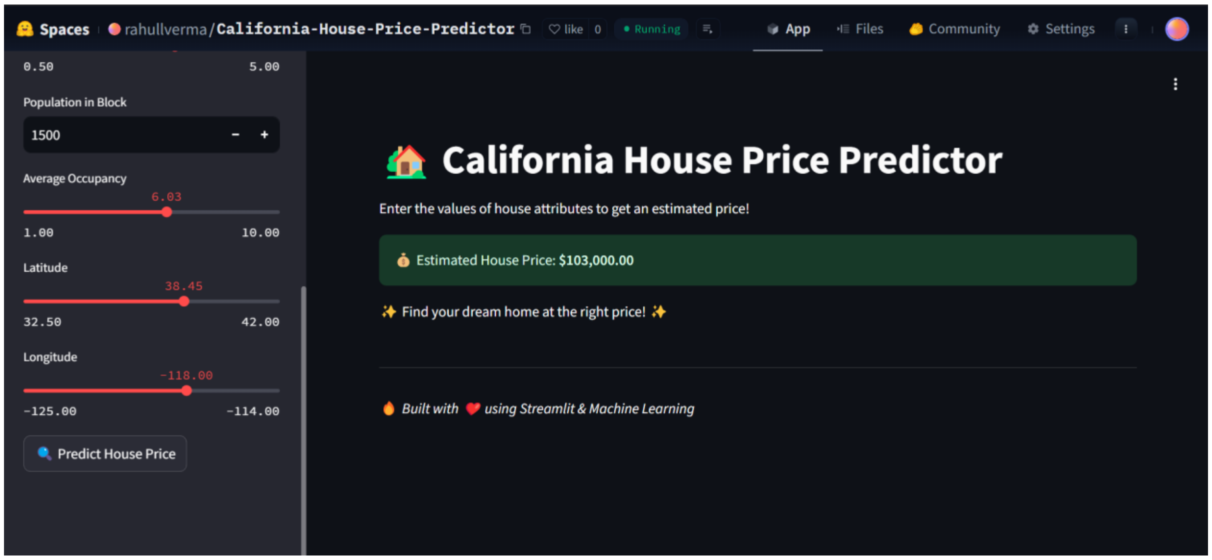


```
{
    "MedInc": 3.5,
    "HouseAge": 20.0,
    "AveRooms": 5.2,
    "AveBedrms": 1.1,
    "Population": 1500.0,
    "AveOccup": 3.0,
```

```
{
    "predicted_price": 2.0646
}
```

```
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Debugger is active!
 * Debugger PIN: 105-453-455
```

Figure 1: API testing using POSTMAN.



California House Price Predictor

Enter the values of house attributes to get an estimated price!

Estimated House Price: $103,000.00

✨ Find your dream home at the right price! ✨

Built with ❤️ using Streamlit & Machine Learning

Figure 2: A screenshot of model deployment on HuggingFace spaces. Features can be adjusted using sliders within reasonable range of values.

Future Improvements

Several enhancements could further improve this housing price prediction system:

1. **Model Versioning**: Implementing DVC or MLflow for tracking model versions and experiments

2. **Advanced Deployment**: Deploying the API on AWS/GCP/Azure with auto-scaling capabilities
3. **Monitoring**: Adding model performance monitoring and drift detection
4. **CI/CD Pipeline**: Creating a continuous integration/deployment pipeline for model updates
5. **Enhanced UI**: Expanding the Streamlit interface with additional visualizations and comparative features

The current implementation provides a solid foundation for these future improvements while delivering a functional and accurate housing price prediction system.

⁂