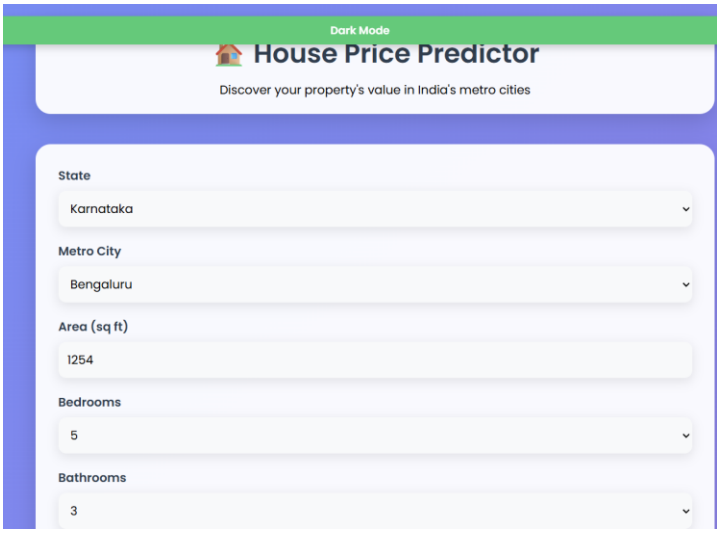
HOUSE PRICE PREDICTION MODE

House price Model is a full-stack web application that provides personalized Price recommendations to users based on their preferences. The platform uses a blend of collaborative filtering and content-based filtering approaches powered by a Random Forest algorithm to deliver tailored Price suggestions.



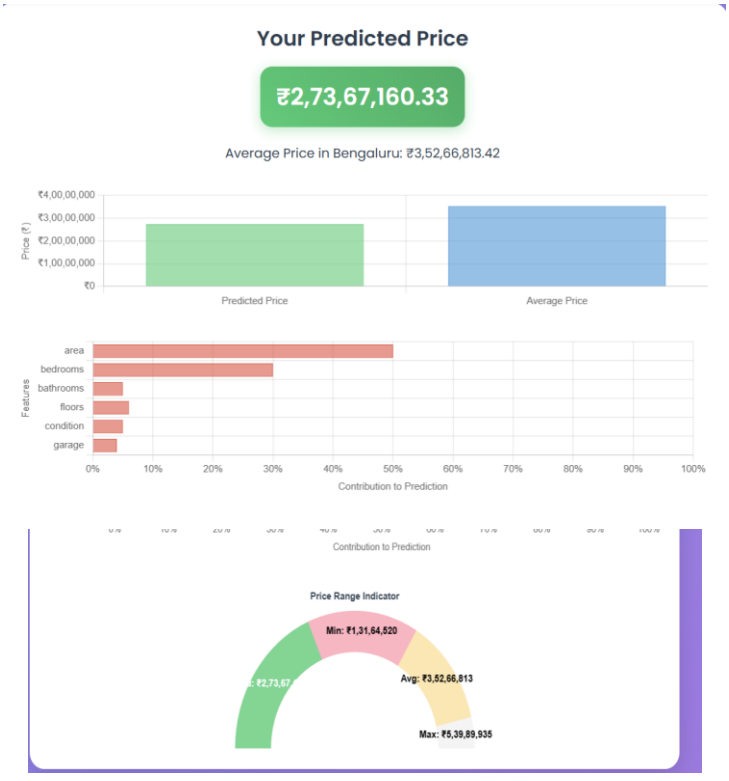


Table of Contents

Overview

House price prediction is a machine learning task aimed at estimating the market value of residential properties based on their features. It is a regression problem where the goal is to predict a continuous output variable (house price) using input features such as location, size, number of bedrooms, and other property characteristics. This task is widely used in real estate, financial planning, and urban development.

Features

- **Personalized Recommendations**: Using a Q-learning algorithm to tailor suggestions based on
- user preferences
- **Movie Discovery**: Swipe interface to easily browse through movie options
- **Search Functionality**: Find specific movies by title or genre
- **Detailed Movie Information**: View comprehensive details including cast, plot, ratings, and
- streaming availability
- **User Interaction**: Like/dislike movies and provide 5-star ratings
- **Responsive Design**: Works across desktop and mobile devices

-

Caching System: Optimized performance with intelligent caching of API responses and frequent queries **Recommendation Explanations**: Provides context for why specific movies are being recommended

Technology Stack

Backend

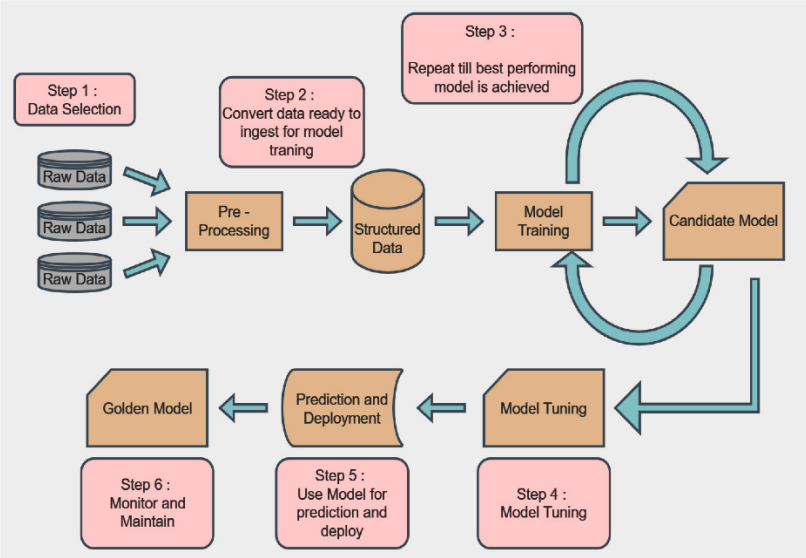
- **Python 3.x**: Core programming language
- **Flask**: Web framework for building the application
- **NumPy**: For mathematical operations and vector calculations
- **SQLite3**: Local database for storing user preferences and caching
- data **Requests**: Library for making HTTP requests to external APIs

Frontend

HTML5/CSS3: Structure and styling
JavaScript: Client-side

System Architecture

The House Price platform consists of three main layers that work together to deliver personalized movie recommendations:



-

Architecture Overview

Step 1: Data Selection - Gather raw data (e.g., house features like size, location, and age) from various sources to form the dataset.

Step 2: Pre-processing - Clean and transform raw data into structured data by handling missing values, encoding categories, and scaling features for model training.

Step 3: Model Training - Train a machine learning model (e.g., Random Forest) on the structured data, iterating and refining until optimal performance is achieved, resulting in a candidate model.

Step 4: Model Tuning - Adjust the candidate model's hyperparameters (e.g., number of trees in a Random Forest) to enhance prediction accuracy.

Step 5: Deployment and Use - Deploy the tuned model into a production environment to make house price predictions for new data.

Step 6: Monitor and Maintain - Continuously monitor the deployed model (golden model) to ensure it performs well over time, maintaining or updating it as needed to adapt to new trends or data shifts

Backend Architecture

The backend of House price Model is built with Flask and follows a modular design pattern. Here's a breakdown of the key

Backend Architecture of House Price Prediction System

- **User Interface Layer (API Layer)**:

  Flask app with CORS for client requests.

  Endpoints: /predict (POST), /metrics (GET), /health (GET).

  Validates inputs and returns JSON responses.

- **Application Logic Layer**:

  Data Preprocessing: Loads CSV, encodes categories, filters outliers, creates Bedrooms Bathrooms feature.

  Model Training: Random Forest Regressor (300 trees, max depth 10).

  Prediction: Transforms inputs, predicts price, computes city stats.

- **Evaluation**: Computes R^2, MAE, RMSE metrics.

- **Data Storage Layer**:

  Local CSV file (House1_Price_Multiplied.csv).

  In-memory storage for model, encoders, and dataset.

  No persistent database.

- **Key Components**:

  Data Management: Handles preprocessing and feature engineering.

  Model Management: Trains and evaluates Random Forest model.

  API Integration: RESTful endpoints for predictions and metrics.

  Performance: Parallel processing (n_jobs=-1), in-memory efficiency

Frontend Interface

The frontend offers a clean, intuitive interface for users to interact with the recommendation system:

1. Header

- Contains a title: **"Intelligent Web Assistant for Text Generation"**
- Displayed with centered styling and a distinct font.

2. Input Section

- A **form** with a labeled <textarea> for users to enter their text prompt (prompt).
- A **"Generate" button** to submit the form.

3. Response Section

- A div (id="response") displays the generated output from the assistant after submission.

-

Styling

- Uses **Tailwind CSS** for styling.
- The design is clean, responsive, and modern:
    - Centered layout
    - Input area with padding, border, and rounded corners
    - Button styled with hover effects and transitions

Behavior

- Uses **JavaScript** to:
    - Capture form submissions
    - Send a POST request to /generate endpoint on the backend
    - Display the assistant’s response in the response are

Installation Guide

☑ **Prerequisites**

- Python 3.8 or later
- Pip (Python package installer)
- A modern browser (for frontend)
- (Optional) Virtual environment

Create Virtual Environment (Optional but Recommended)

```
python -m venv venv
source venv/bin/activate  # For Linux/macOS
venv\Scripts\activate     # For Windows
```

2. 📄 Create requirements.txt

Create a file named requirements.txt with the following contents:

```
flask
flask-cors
numpy
pandas
scikit-learn
```

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

3. Dataset Setup

Ensure the dataset file is placed correctly. In app.py, this line:

python

Copy Edit

```
DATASET_PATH = r' C:\Users\Anshu2006\OneDrive\Desktop\DATA DCIENCE PROJECT\House1_Price_Multiplied.csv'
```

Change to a relative path like:

python

Copy Edit

```
DATASET_PATH = 'House1_Price_Multiplied.csv'
```

Place the CSV file in the **same directory** as app.py.

-

4. 🚀 Run Backend Server

bash

Copy Edit

python app.py

- This starts the Flask server at http://0.0.0.0:5000/
- Test it by visiting http://localhost:5000/health in your browser. You should see:

json

Copy Edit

{"status": "healthy"}

5. Launch Frontend

Simply open index.html in your browser (e.g., double-click or use Live Server in VS Code).

Connect Frontend to Backend

Ensure that your JavaScript in index.html sends requests to the correct backend routes:

- /generate (for text generation if applicable)
- /predict (for price prediction in your case — make sure JS matches this route