Project- House Price Prediction

Company- Deccan AI

Github link- <https://github.com/VaibhavSh026/House-Price-Predictor>

Index:

1. **Introduction**
2. **Data Preprocessing and Feature Engineering**

* 2.1 Data Cleaning
* 2.2 Feature Scaling
* 2.3 Feature Selection
* 2.4 Encoding

1. **Model Selection and Optimization**

* 3.1 Model Used
* 3.2 Hyperparameter Tuning
* 3.3 Model Evaluation

1. **Deployment Strategy**

* 4.1 Deployment Platform
* 4.2 Flask API Development
* 4.3 Endpoint and Request Details

1. **API Usage Guide**

* 5.1 Example Request and Response

1. **Conclusion**
2. Introduction

The **House Price Prediction** project leverages **machine learning** to estimate property prices based on various input features. With the real estate market being highly dynamic, accurate price prediction can help buyers, sellers, and investors make informed decisions. This project is designed to address this need by creating a **regression-based predictive model**, which is capable of analyzing key features such as **property size**, **location attributes**, and other relevant factors to generate a reliable price estimate.

The project incorporates a systematic approach starting from **data preprocessing** to **feature engineering**, followed by **model selection**, **optimization**, and **deployment**. The end goal is to offer a scalable and user-friendly **Flask API** that enables real-time predictions. Additionally, the use of tools like **DVC** ensures efficient model versioning, tracking performance improvements, and maintaining consistency.

Deployment on **Render** makes the model accessible over the web, with a detailed **API usage guide** to ensure ease of integration with other applications or services. This report outlines the step-by-step methodology followed throughout the project, offering insights into each critical phase from preprocessing to deployment.

**Key Objectives:**

* Build a reliable regression model for house price prediction.
* Ensured scalability through a robust deployment strategy.
* Provided version control to monitor and optimize performance.
* Created an easy-to-use API for real-time predictions.

1. Data Preprocessing and Feature Engineering

**2.1 Data Cleaning**

* **Handling Missing Values:** Missing values were imputed using mean for numerical features and median where outliers were present.
* **Outlier Removal:** Outliers were detected using IQR (Interquartile Range) and removed to enhance model stability.

**2.2 Feature Scaling**

* **Standardization:** Standardization was applied to normalize feature distributions, ensuring the mean is 0 and standard deviation is 1.
* **Min-Max Scaling:** Used for features with wide value ranges to scale values between 0 and 1, improving model convergence.

**2.3 Feature Selection**

* **Correlation Analysis:** Highly correlated features were identified using a correlation matrix and removed to avoid multicollinearity.
* **Recursive Feature Elimination (RFE):** RFE was implemented to iteratively select the most relevant features, improving model performance.

**2.4 Encoding**

* **One-Hot Encoding:** Applied to categorical variables with no inherent order, creating binary columns for each category.
* **Label Encoding:** Used for ordinal categorical variables where the order of categories holds meaningful information.

**3. Model Selection and Optimization**

**3.1 Model Used**

* Implemented a **Regression-based Model** for continuous house price estimation, offering a balance of simplicity and performance.

**3.2 Hyperparameter Tuning**

* Applied **Grid Search Cross-Validation (Grid Search CV)** to explore different combinations of hyperparameters.
* Optimized parameters such as learning rate, number of estimators, and regularization factors to achieve the best performance.

**3.3 Model Evaluation**

* Evaluated model performance using key metrics:
  + **R-squared (R²):** To measure the proportion of variance explained by the model.
  + **Mean Absolute Error (MAE):** To assess the average prediction error.
  + **Root Mean Square Error (RMSE):** To penalize larger errors more heavily.
    1. **Deployment Strategy**

**Deployment**

* **Deployed on Render with Flask API for real-time predictions.**
* **Utilized Model Versioning tools like DVC for tracking improvements.**

**Platform:**

* **Render: Deployed on a cloud-based platform that simplifies web app and API management.**

**API Framework:**

* **Flask: Utilized Flask to create a lightweight API for real-time predictions, enabling quick client-server communication.**

**Real-time Prediction Features:**

* **API Endpoints: Defined endpoints for processing input data and serving predictions.**
* **Input Handling: Designed to validate and format incoming data efficiently.**
* **Response Format: Returns predictions in JSON format for easy integration.**

**Model Versioning**

**Tools:**

* **DVC (Data Version Control): Employed for tracking datasets and model changes, enhancing project management.**

**Key Benefits of DVC:**

* **Tracking Improvements: Monitors changes to models, enabling performance evaluation of different versions.**
* **Collaboration: Facilitates teamwork by allowing seamless updates and sharing of datasets and models.**
* **Reproducibility: Ensures experiments are repeatable by capturing the state of data and models over time.**
* **Storage Management: Supports various storage backends for handling large datasets without clogging the main repository.**
  + 1. API User Guide

**The application features a dedicated API endpoint that allows users to obtain house price predictions using a straightforward request/response format. Below are the details for utilizing the prediction API:**

**Endpoint**

* **URL: http://127.0.0.1:5000/predict**

**This is the API endpoint where users send their data for processing.**

**Request Specifications**

* **Method: POST**
* **Content Type: JSON**

**To use the API, send a POST request to the specified endpoint with a JSON payload containing the relevant features. The structure of the JSON payload should resemble the following example:**

**json**

**{**

**"feature1": 1200,**

**"feature2": 3,**

**"feature3": 2**

**}**

**Parameters:**

* **feature1: Represents a numeric value, such as the area of the house in square feet.**
* **feature2: Indicates the number of bedrooms in the house, specified as an integer.**
* **feature3: Represents another relevant feature, such as the number of bathrooms, also specified as an integer.**

**Response Format**

**Upon successfully receiving and processing the request, the API responds with a JSON object containing the predicted price of the house. An example of a typical response is as follows:**

**json**

**{**

**"predicted\_price": 250000**

**}**

**Response Details:**

* **predicted\_price: This field contains the estimated market price of the house based on the input features, represented as a numeric value (in this example, $250,000).**

Conclusion

This project presents a comprehensive approach to house price prediction by integrating effective data preprocessing, robust model selection, and efficient deployment strategies. By utilizing DVC for model versioning and implementing a user-friendly Flask API, the system guarantees both scalability and reliability. Continuous optimization coupled with version control allows the model to adapt and maintain its performance, offering valuable insights into real estate market trends. This integrated method ensures that the prediction system not only meets current demands but is also prepared for future advancements in data science and machine learning.