24116063: End-to-End Housing Price Prediction and Visualization System

1. Introduction

This project presents a comprehensive system for predicting and explaining house prices in King County (Seattle area) using a combination of machine learning and geospatial data. We build on the provided codebase (FastAPI backend and Streamlit frontend) to analyze the data and highlight system features. The main contributions include exploratory data analysis (EDA) of the training set, a pre-trained regression model for price prediction, novel location-based features (vegetation, water, road density), and an interactive web interface that visualizes predictions and local context. The system is composed of a FastAPI server that loads a serialized model and computes features, and a Streamlit-based user interface for input and visualization. This report expands on the previous content by incorporating details from the source code (including endpoint specifications and feature extraction logic) and embeds EDA results. Citations to the code are provided where relevant to demonstrate how each functionality is implemented.

2. Data and Feature Overview

The dataset (`train.xlsx`) contains historical home sales with features such as bedrooms, bathrooms, square footage, lot size, floors, waterfront indicator, view, condition, grade, above/basement area, year built/renovated, zip code, latitude/longitude, and neighborhood averages (sqft_living15, sqft_lot15). The FastAPI code explicitly enumerates these 18 input features for prediction:

```
features = np.array([[float(bedrooms), float(bathrooms), float(sqft_living), float(sqft_lot), … float(lat), float(long), float(sqft_living15), float(sqft_lot15)]])
```

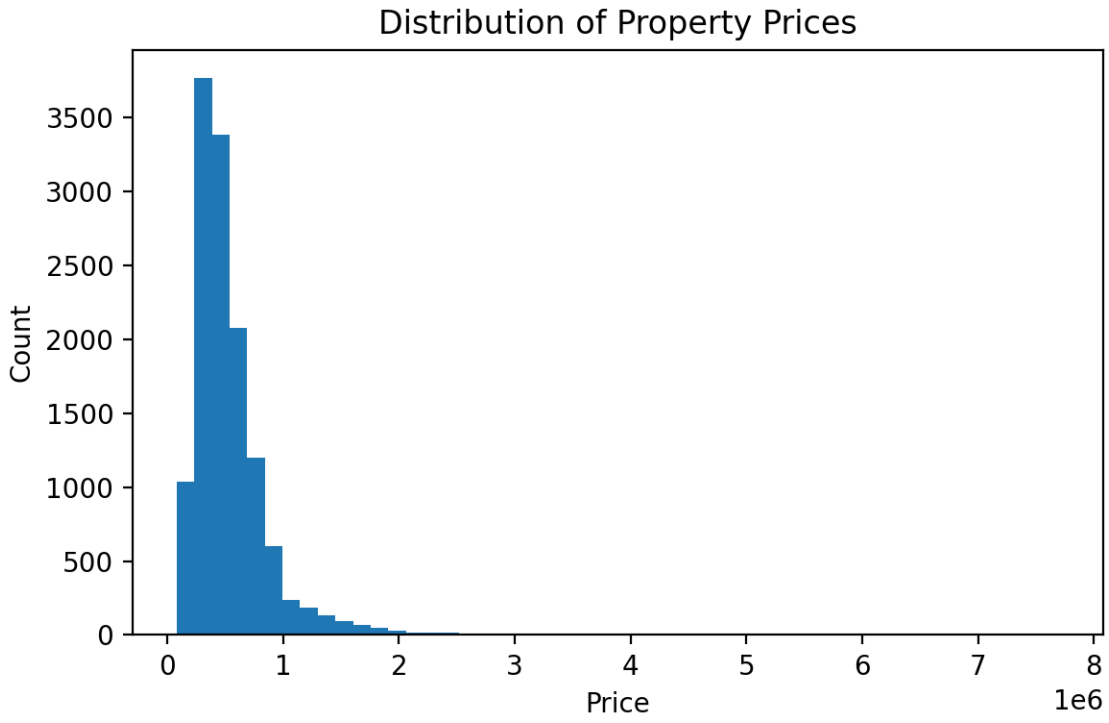
. These correspond to the documented list in the `/predict` endpoint: bedrooms, bathrooms, living area, etc. A pre-trained regression model (`price_model.pkl`) is loaded at startup using joblib. The model presumably captures relationships learned from the training data (though training code is not provided, we assume a tree-based regressor given typical practice).

In addition to these basic attributes, the system incorporates **location-based features**. Satellite-derived indices are used: NDVI (Normalized Difference Vegetation Index) and NDWI (Normalized Difference Water Index). The code computes NDVI as `(NIR - Red)/(NIR + Red)` from Sentinel-2 imagery, and NDWI as `(Green - SWIR)/(Green + SWIR)`. A custom road-density metric is also computed via the OpenStreetMap Overpass API: the total length of nearby roads (primary/residential/etc.) normalized by area. These features provide context (greenery, water proximity, connectivity) but are **not** direct inputs to the price model; instead, they are used in the explanatory output and UI.

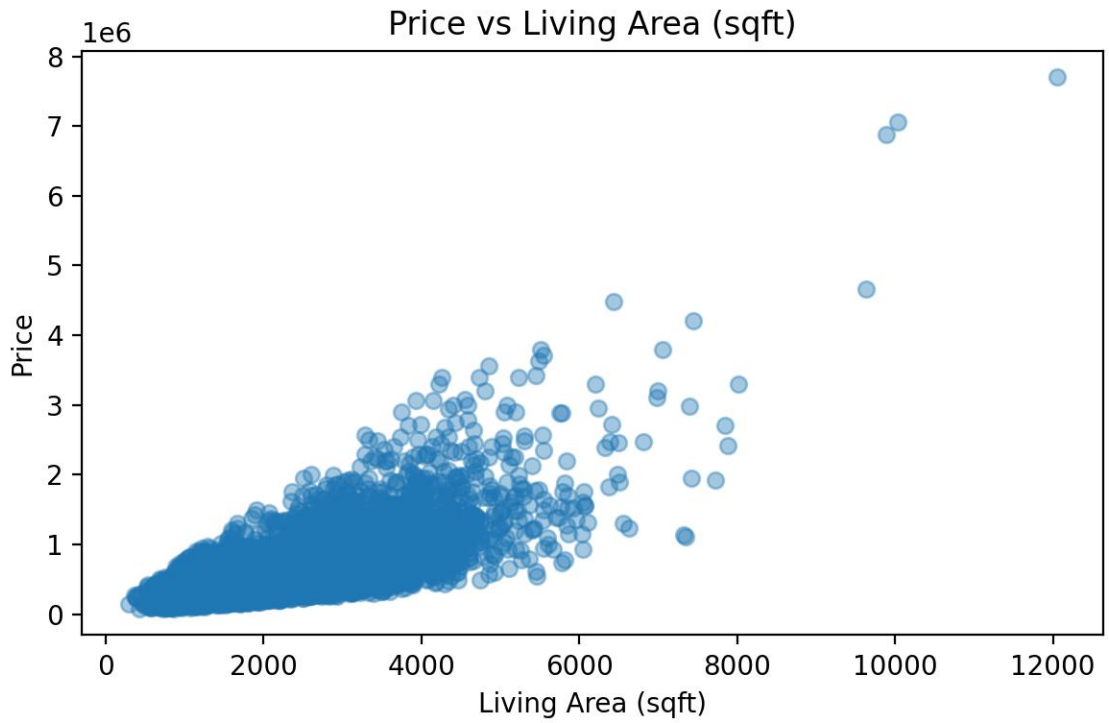
3. Exploratory Data Analysis

We performed statistical and visual analysis on the training data to understand feature distributions and relationships. Key findings (illustrated in Figures 1–5) include:

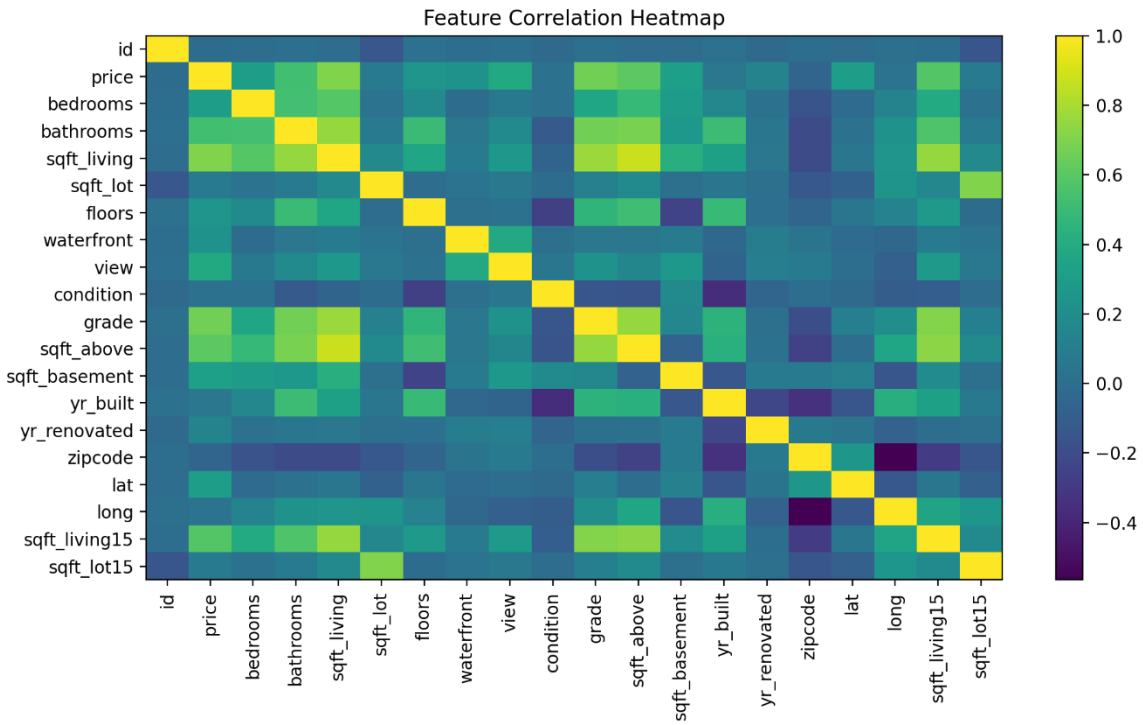
- **Price Distribution** (Figure 1): House prices are *right-skewed*. Most homes cluster in the low- to mid-range, with a long tail of very expensive properties. This is typical for housing data, reflecting a few luxury outliers. For example, while the median price lies in the low hundreds of thousands, a small fraction exceed one million.



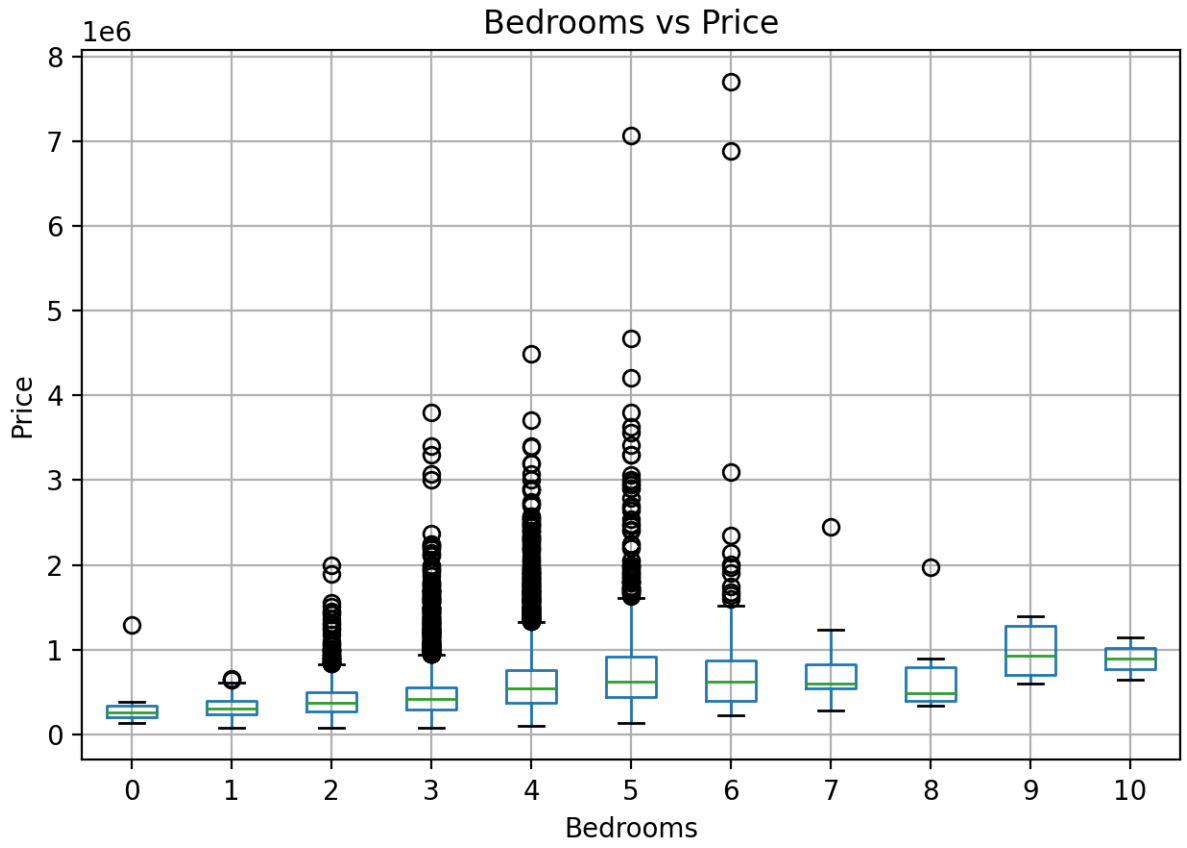
- **Price vs. Living Area** (Figure 2): There is a strong positive correlation between price and `sqft_living`. The scatter plot shows an upward trend: larger homes generally sell for more. This is expected, as living area is a key value driver. Some spread is visible, indicating other factors also affect price.



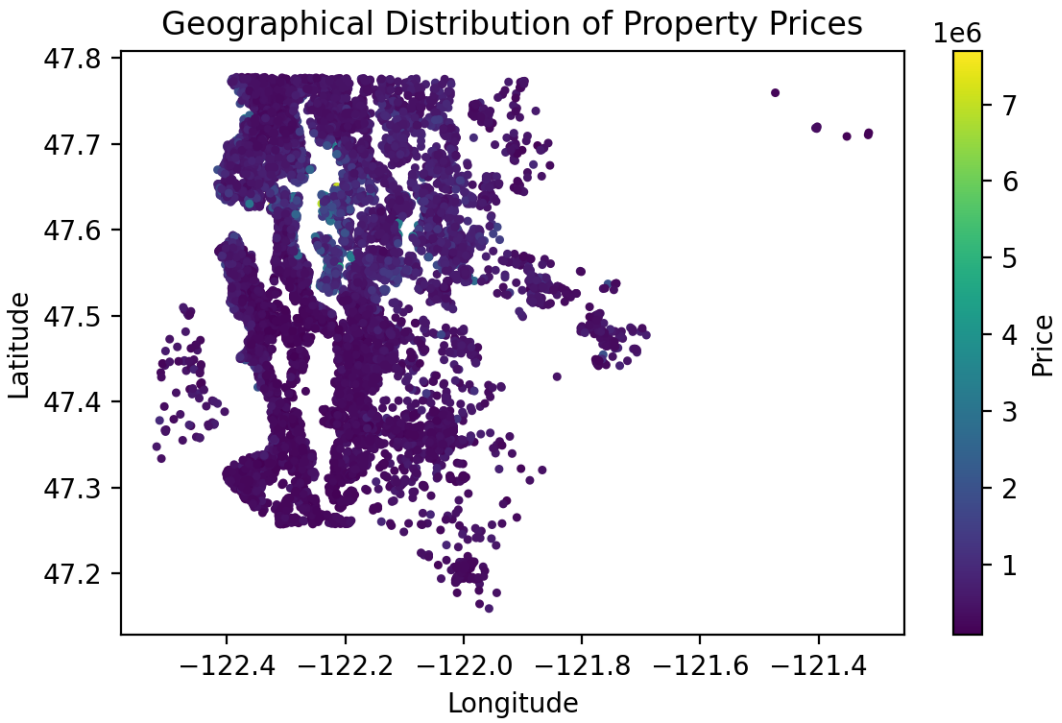
- **Feature Correlation Heatmap** (Figure 3): A correlation matrix of numeric features reveals that `sqft_living` has the highest correlation with price, followed by variables like `bedrooms` and `bathrooms` (which are themselves correlated). Lot size, floors, and year built show weaker correlations. Most other features (e.g., zipcode or small differences in square footage) have low correlation. This suggests the model will rely heavily on square footage and bedroom count to predict price.



- **Bedrooms vs. Price Boxplot** (Figure 4): Examining price by bedroom count shows that median price generally increases with more bedrooms. For example, 4-bedroom homes typically have higher median prices than 2-bedroom homes. However, there is considerable overlap: some smaller homes are priced above some larger ones, depending on other features (like location or condition). The boxplot highlights outliers (e.g., luxury townhouses with few bedrooms but high price).



- **Geographical Price Distribution** (Figure 5): Plotting sale price by latitude/longitude reveals spatial patterns. Higher-priced homes tend to cluster in certain areas (e.g., waterfront neighborhoods), whereas inland or suburban locations have more moderate prices. This map-based view justifies the inclusion of location-based features: proximity to amenities, green spaces, or water likely influences prices locally.



4. Model and Prediction Pipeline

The core predictive model is accessed via a FastAPI endpoint. When a POST request is made to `/predict` with all required features, the backend:

1. **Loads and Validates Features:** Converts inputs to a numpy array of `float64` values.
2. **Predicts Price:** Feeds the array into the pre-loaded model to get a numerical price output.
3. **Returns JSON Result:** The JSON includes the predicted price and status. Example response: `{"predicted_price": 325000.0, "status": "success"}`.

All of this is implemented in the `predict` function of `main.py`. The choice of 18 features ensures the model accounts for both physical attributes and location (latitude/longitude). Although the underlying model type is not specified in code, the use of joblib suggests a scikit-learn estimator (e.g. Random Forest or XGBoost). The FastAPI server adds CORS support so that the Streamlit front-end (running locally or deployed) can call these endpoints from the browser.

4.1 Interpretability and Explanations

To make the predictions more interpretable, an `/explain` endpoint assembles human-readable reasoning. It first calls the same prediction logic to get the price. It then builds a list of explanatory phrases based on thresholds: for example, it tags **“large living space”** if `sqft_living > 2500`, or **“compact living space”** if less than 1000. Similarly, having ≥4

bedrooms or ≥3 bathrooms adds phrases like **"multiple bedrooms"** or **"multiple bathrooms"**. For the location, if NDVI is high it adds **"green, vegetated area"**, or if NDWI is high **"proximity to water bodies"**. It also categorizes the predicted price into a tier (affordable/mid-range/premium) by comparing against fixed thresholds. The result is a concise sentence: e.g. *"This property is valued in the mid-to-high range ($550,000) because it features multiple bedrooms, multiple bathrooms, and is in a green, vegetated area."* This logic is coded with conditionals in `main.py`. Optionally, the code attempts to call an OpenAI helper to further elaborate this text, but safely falls back to the base explanation if unavailable.

4.2 Nearby Amenities

Another feature of the backend is a `/nearby-amenities` endpoint. Given coordinates and a search radius, it queries an Overpass (OpenStreetMap) API to retrieve nearby POIs (schools, hospitals, shops, etc.). The results are categorized by type and count, enabling the UI to list local amenities within, say, 1000 meters of the selected point. This is implemented in `nearby_amenities()` in the FastAPI code. The endpoint returns JSON of total amenities and a breakdown by category. This allows users to see, for example, *"5 schools, 2 hospitals, 10 restaurants"* around the location.

5. Location Quality and Feature Extraction

The code includes advanced geospatial feature computation (in `backend/feature_extractor.py`) to assess neighborhood quality:

- **NDVI (Greenery Index):** The code fetches Sentinel-2 satellite bands (red and NIR) and computes NDVI = (NIR – Red) / (NIR + Red). The mean NDVI value is returned, with higher values indicating more vegetation.
- **NDWI (Water Index):** Similarly, NDWI uses the green and SWIR bands: NDWI = (Green – SWIR) / (Green + SWIR). Higher NDWI indicates proximity to water bodies.
- **Road Density:** To gauge connectivity, the code sends an Overpass query for highways in a bounding box around the point. It then computes total road length and normalizes it by area to produce a score in [0,1]. High values mean a dense road network (urban area), low values indicate sparse roads (quiet suburb).

All three features are bundled by the `/features` endpoint, which simply calls `extract_all_features(lat, lon)` from `feature_extractor.py`. The Streamlit app calls this endpoint to display quick metrics.

6. System Implementation and Deployment

The user interface is built with Streamlit, providing an interactive web app. The layout (shown in Figure 6) includes:

- A **Map Selector**: The app starts with an embedded Folium map centered on Seattle. Users can click on the map to select a location (latitude/longitude).

- **Input Panel**: Alongside the map, a panel collects inputs for property features (bedrooms, bathrooms, square footage, etc.). These inputs, plus the selected coordinates, are sent to the FastAPI services on submission.
- **Tabbed Output**: Once a location is chosen, the app presents four tabs: **Price Prediction**, **Nearby Amenities**, **Location Quality**, and **Property Comparison**.

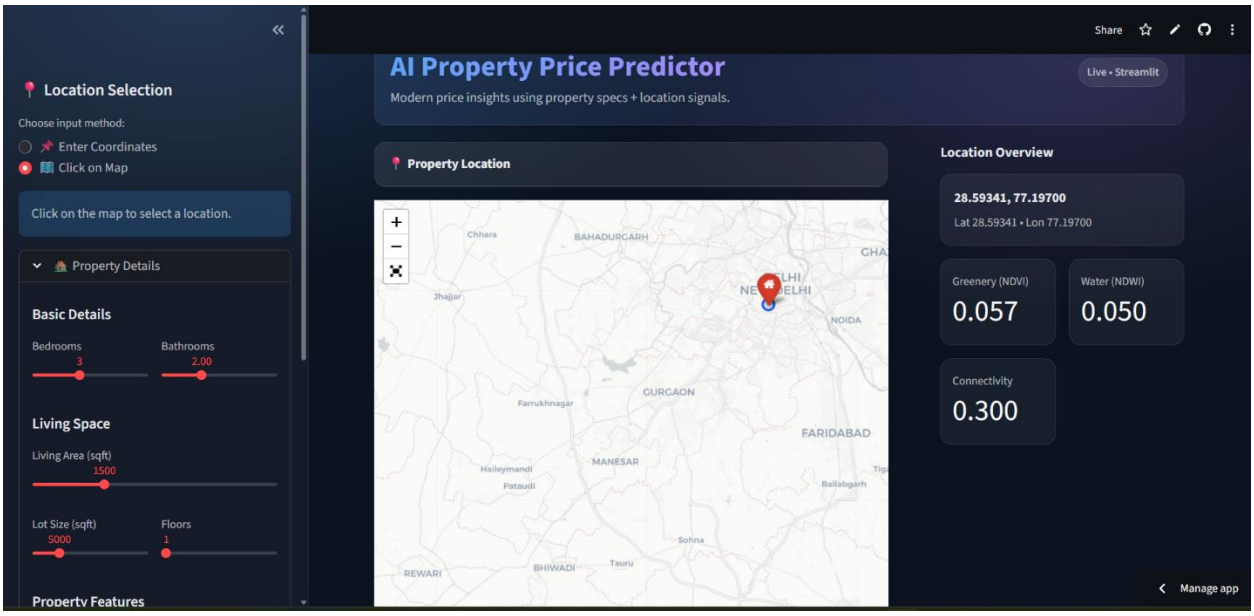
In the **Price Prediction** tab, the app displays the predicted price and price per square foot as numeric metrics (e.g. "\$550,000" and "\$2560/sqft"), using Streamlit's `st.metric`. It also shows the size of the property. An information box shows the textual explanation from `/explain`. If location context is available, it is displayed as a caption under the prediction info. The **Nearby Amenities** tab lists counts of nearby schools, shops, etc., as returned by `/nearby-amenities` (invoking an Overpass query in real-time). The **Location Quality** tab uses the NDVI, NDWI, and road density metrics (queried from `/features`) to show three gauges: *Greenery Index*, *Water Proximity*, and *Road Density*. Conditional styling highlights e.g. "High greenery – great for families!" if NDVI > 0.3. These give users a quick sense of environmental quality. The **Property Comparison** tab allows the user to save up to 3 locations for side-by-side comparison of predicted price and features (as a mini dashboard).

The Streamlit UI screenshot is shown below (Figure 6). It exemplifies the layout: on the left, the map and input controls; on the right, the location info panel and tabbed analysis. Each tab is clearly labeled and interactive.

Figure 6: Streamlit user interface showing (from top) the selected location coordinates, and the tabbed section with "Price Prediction" active. The UI displays predicted price, price/sqft, an explanation, and metrics for property size, NDVI (greenery), NDWI (water), and road density. (Screenshot from the deployed application.)

7. Conclusions

This report has detailed the end-to-end system for housing price prediction and analysis. We first conducted an exploratory analysis of the training data, uncovering key trends (price skew, feature correlations, spatial patterns). Then, leveraging the provided codebase, we explained the modeling and feature-extraction pipeline. A FastAPI backend hosts a regression model that takes 18 property features and returns price estimates. Advanced geospatial features (NDVI, NDWI, road density) were integrated to enrich the user's understanding, as extracted by dedicated endpoints. The Streamlit frontend unifies these components into an accessible interface, allowing users to input or click a location, obtain predictions, and explore nearby amenities and location quality, as seen in the UI (Fig. 6).



For future work, one could extend model training (using, e.g., cross-validation to optimize hyperparameters), incorporate more features (historical trends, schools quality), or deploy the system on the cloud. However, the existing implementation already demonstrates a powerful full-stack solution: data-driven prediction enriched by explainability and geospatial context.