# **Dynamic Property Analysis and Search**

Made By : Submitted To :

**Akaspreet Danish Bansal**

## **Project Overview**

### **Problem Statement**

The task is to implement a Gen AI-based filtering system to efficiently handle and analyze property data provided in a JSON format. The dataset includes columns such as address, latitude, longitude, bedrooms, bathrooms, property type, area, postcode, sector, etc. The system should support advanced NLP capabilities for analysis and property recommendations. For instance, it should be able to analyze queries like "Tell me the change in property type over the years" and recommend properties based on specific criteria like "I want a 3BHK house in a posh area with good online facilities."

## **Application Structure**

**1. gemini\_handler.py**

* **Purpose:** Handles interactions with the Gemini API to interpret user queries and generate Python code for data analysis.
* **Key Functions:**
  + interpret\_query(self, query): Sends a prompt to the Gemini API to interpret a user query and extract parameters.
  + generate\_and\_execute\_analysis(self, query, parameters, df): Generates Python code for data analysis based on the interpreted query and executes it on the provided DataFrame.

**2. llm\_handler.py**

* **Purpose:** Manages encoding and searching functionalities using SentenceTransformers.
* **Key Functions:**
  + encode\_query(self, query): Encodes a user query into embeddings.
  + encode\_properties(self, df): Encodes property data into embeddings.
  + search\_properties(self, query\_embedding, property\_embeddings, top\_k=5): Searches for the top-k most similar properties based on query embeddings.

**3. llm\_model.py**

* **Purpose:** Initializes and handles operations related to the language model from Hugging Face's transformers library.
* **Key Functions:**
  + initialize\_llm(): Initializes the tokenizer and model.
  + encode\_query(tokenizer, model, query): Encodes a query using the model.
  + search\_properties(df, query\_embedding, top\_k=5): Placeholder for implementing similarity search logic.

**4. search\_engine.py**

* **Purpose:** Orchestrates the property search process using embeddings.
* **Key Functions:**
  + \_\_init\_\_(self, df, llm\_handler): Initializes the search engine with property data and the LLM handler.
  + search(self, query, parameters, top\_k=5): Combines the user query and parameters to perform a property search.

**5. utils.py**

* **Purpose:** Provides utility functions for executing generated analysis code.
* **Key Functions:**
  + execute\_analysis\_code(code, df): Executes the generated analysis code and saves the result.

**6. app.py**

* **Purpose:** Main Streamlit application for user interaction, data loading, and handling property analysis and search.
* **Key Components:**
  + Data loading and preprocessing using load\_and\_preprocess\_data().
  + Initializing handlers and search engine.
  + Input field for user queries.
  + Displaying query interpretation, search results, analysis results, and generated code.

**7. data\_loader.py**

* **Purpose:** Loads and preprocesses the property data from a JSON file.
* **Key Functions:**
  + load\_data(file\_path): Loads data from a JSON file.
  + preprocess\_data(df): Preprocesses the data by converting date columns, handling missing values, and parsing the lease term.

**8. gemini\_api.py**

* **Purpose:** Initializes and interacts with the Gemini API.
* **Key Functions:**
  + initialize\_gemini(api\_key): Configures the Gemini API with the provided API key.
  + generate\_code(model, prompt): Generates code using the Gemini model based on the provided prompt.

### **How It Solves the Problem Statement**

### **Approach Overview**

#### **1. Data Handling and Preprocessing**

* **File:** data\_loader.py
* **Functionality:**
  + Load data from a JSON file.
  + Preprocess the data by converting date columns, handling missing values, and parsing the lease term.
* **Models/Tools Used:** Pandas for data manipulation

#### **2. Interpreting User Queries with Gemini API**

* **File:** gemini\_handler.py
* **Functionality:**
  + Send user queries to the Gemini API to interpret the intent and extract parameters.
  + Generate Python code for data analysis based on interpreted queries.
* **Models/Tools Used:** Gemini API for natural language understanding and code generation

#### **3. Encoding and Searching Properties with SentenceTransformers**

* **File:** llm\_handler.py
* **Functionality:**
  + Encode user queries and property data into embeddings using SentenceTransformers.
  + Search for similar properties based on query embeddings.
* **Models/Tools Used:** SentenceTransformers for embedding generation and similarity search.

4. **Executing Generated Analysis Code**

* **File:** utils.py
* **Functionality:**
  + Execute the Python code generated by the Gemini API for data analysis.
* **Models/Tools Used:** Built-in Python functions for code execution.

5. **Orchestrating the Search Process**

* **File:** search\_engine.py
* **Functionality:**
  + Combine user queries and parameters to perform a property search.
* **Models/Tools Used:** Custom logic using encoded embeddings for similarity search.

6. **Streamlit Application for User Interaction**

* **File:** app.py
* **Functionality:**
  + Provide an interactive user interface for data loading, query input, and displaying results.
* **Models/Tools Used:** Streamlit for building the web application.

### **Models Used and Their Purpose**

1. **Gemini API**
   * **Purpose:** Interprets natural language queries and generates Python code for data analysis.
   * **Why:** Provides a powerful way to understand complex user queries and automate code generation for analysis tasks.
2. **SentenceTransformers (all-MiniLM-L6-v2)**
   * **Purpose:** Encodes user queries and property data into embeddings for similarity search.
   * **Why:** Efficient for generating high-quality embeddings suitable for similarity search, ensuring relevant property recommendations.

### **Overall Workflow**

1. **Data Loading and Preprocessing:**
   * Load and preprocess the property data from a JSON file.
2. **Query Interpretation and Code Generation:**
   * Use Gemini API to interpret user queries and generate code for analysis.
3. **Embedding and Similarity Search:**
   * Encode queries and property data using SentenceTransformers.
   * Perform similarity search to find relevant properties based on user queries.
4. **Executing Analysis and Displaying Results:**
   * Execute generated analysis code and display results using Streamlit.
   * Provide an interactive UI for users to input queries and view recommendations or analysis results.

**Results**















