Property AI Agent – Intelligent RAG System for Property Analytics

A next-generation Retrieval-Augmented Generation (RAG) system that fuses semantic search, AI-driven analytics, and natural language querying for property datasets. Designed with modularity, transparency, and scalability in mind — enabling powerful property data exploration through conversational AI.

Key Features

- Hybrid AI Architecture: Combines semantic vector search with AI-generated pandas analytics
- Dynamic Analytics: Auto-generates and executes pandas code for live data insights
- Semantic Search: Understands natural language queries and extracts intelligent constraints
- Interactive UI: Modern Flask-based web app with live panels and real-time query execution
- Mobile Responsive: Works seamlessly across all device types
- Transparent AI Execution: Inspect generated code, intermediate data, and search matches
- Multimodal Analytics: Runs Gemini and ChatGPT analytics in parallel for more reliable results

Note: Current version processes queries in approximately 30 seconds per response (full RAG + analytics + vector search). Speed optimizations are under development.

System Architecture

- User Query

    - ↓
    - [Analytics Agent (Gemini + ChatGPT)]
    - ↓
    - [Vector Search Engine (Pinecone)]
    - ↓
    - [AI Response Composer (OpenAI/Gemini)]
    - ↓
    - Flask Web Interface → Result Panels

Core Components

Component	Description
Property RAG Agent	Core orchestrator coordinating vector, analytics, and LLM services
Analytics Agent	Generates and executes pandas code for live data insights
Vector Search Engine	Pinecone-powered semantic property retrieval
AI Response Composer	Combines Gemini + ChatGPT outputs for final response
Web Interface	Flask + JavaScript frontend for real-time interactions

---

Technology Stack

Backend

- Python 3.12+
- Flask (threaded)

- Pandas / NumPy
- python-dotenv (for API key management)

AI/ML Services

- Pinecone – Vector database for semantic search
- Sentence Transformers (all-MiniLM-L6-v2) – Text embeddings
- Google Gemini – LLM for analytics and responses
- OpenAI (GPT-4o-mini) – Secondary LLM for code generation and final composition

Frontend

- HTML5 / CSS3 / JavaScript (ES6+)
- Marked.js – Markdown rendering
- Font Awesome – Icons
- Responsive Flask templates – For real-time panels

Installation

Prerequisites

- Python 3.12 or higher
- pip package manager
- Valid API keys for:
    - Pinecone
    - Google Gemini

- OpenAI

Setup Steps

- # 1. Clone the repository
- git clone <repository-url>
- cd simplyphi
-
- # 2. Install dependencies
- pip install -r requirements.txt
-
- # 3. Prepare data
- # Required files:
- # - property_embeddings_ready.csv
- # - property_data_cleaned.csv (optional for analytics)
-
- # 4. Configure environment variables
- cp .env.example .env
- # then edit .env with your API keys
-
- # 5. Run the app
- python app.py

Access the App

Visit: http://localhost:7860
Start asking property-related questions in natural language.

Usage Examples

Example Queries

Query	Intent
highest crime rate in which city?	City-level risk analytics
2 bedroom apartments under £1500	Price and room filtering

new homes with 3+ bedrooms	Multi-constraint filtering
studio apartments in low crime areas	Safety-based search
compare prices between studio and 2 bedroom	Comparative analytics
properties with flood risk	Risk-based search
best value property type	Value estimation
most expensive location	Market insights

Interactive Features

- Prebuilt Query Chips – One-click test queries
- Live Panels – Toggle between:
    - Pandas Code (Gemini / ChatGPT)
    - Data Outputs (Gemini / ChatGPT)
    - Vector Matches (Pinecone)
- Auto-Refresh Panels – Resets between each query
- Timing Display – View full processing duration (about 30 seconds currently)

Data Pipeline Overview

- Raw Property Data
- ↓
- Data Cleaning & EDA
- ↓
- Feature Engineering

- ↓
- Embedding Generation (Sentence Transformer)
- ↓
- Pinecone Vector Index
- ↓
- RAG Query (Analytics + Vector Search)
- ↓
- Final AI-Composed Response

Data Statistics

- Records: 147,666+
- Embedding Dimension: 384
- Vector Database: Pinecone (AWS Serverless)
- Search Speed: ~100 ms
- Response Time: ~30 seconds (end-to-end)

Configuration

Environment Variables (.env)

- PINECONE_API_KEY=your_pinecone_key
- GEMINI_API_KEY=your_gemini_key
- OPENAI_API_KEY=your_openai_key
-
- # Optional
- SAMPLE_UPLOAD_LIMIT=1000
- HEADLESS=1

Performance and Optimization

Current Focus Areas

- Reducing LLM latency (Gemini + OpenAI parallel)
- Smarter caching of embeddings and intermediate responses
- Asynchronous I/O for API calls and vector queries

Implemented Optimizations

- Model caching (loaded once)
- Persistent Pinecone connections
- Local response caching for repeated queries
- Batched vector similarity search

UI/UX Design

Visuals

- Dark mode interface
- Glassmorphic panels with gradient accents
- Smooth animations and transitions

Usability

- Responsive design (mobile-first)
- Keyboard accessible
- Real-time error messages
- Session resets for each query cycle

Multimodal Analytics (Gemini + ChatGPT)

The system executes analytics through two LLMs in parallel:

Model	Role
Gemini	Generates and executes pandas code
ChatGPT (GPT-4o-mini)	Generates alternate pandas logic
Pinecone	Provides semantic context vectors
Response Composer	Merges results, prioritizing OpenAI output with Gemini fallback

Frontend Toggle Panels

- Gemini Code
- ChatGPT Code
- Gemini Data
- ChatGPT Data
- Vector Matches

Each toggle allows deep inspection of the AI reasoning process.

Troubleshooting

Issue	Possible Fix
No response generated	Check API keys, data files, and internet connection

Processing failed	Ensure dependencies installed and APIs are reachable
Slow response	Currently expected (~30s). Check network and API latency
Empty results	Use different query phrasing or validate dataset content

Enable debug logs in the terminal for detailed traces.

Roadmap and Future Enhancements

Planned Features

- Multi-language support
- Interactive data visualizations (Plotly or Altair)
- Export results to CSV or Excel
- User authentication
- RESTful API endpoints

Performance Goals

- Reduce response time below 5 seconds via:
    - Redis caching
    - Asynchronous model execution
    - CDN for assets
    - Query batching and streaming