CityScape: An AI-Powered NYC Travel Guide

Project Overview

CityScape is an intelligent travel assistant that helps users explore New York City through natural conversation. Built using OpenAI's language models and Chroma Vector DB, the system provides personalized recommendations for restaurants, museums, attractions, and experiences across NYC. The project leverages Retrieval Augmented Generation (RAG) to ground its responses in accurate, up-to-date information about NYC establishments.

Technical Architecture

Core Components

1. **Vector Database (Chroma DB)**
   - Stores embeddings of NYC venue information
   - Enables semantic search capabilities
   - Created and managed through `create_database.py`
2. **Query Engine**
   - Handles user interactions through `query_data.py`
   - Processes natural language queries
   - Manages conversation context and user preferences
3. **Knowledge Base**
   - Structured data about NYC venues in `nyc.txt`
   - Information includes locations, prices, ratings, and descriptions
   - Data is chunked and embedded for efficient retrieval

Key Features

1. Intelligent Query Processing

- Classification of user queries into categories (food, entertainment, museums, etc.)
- Context-aware responses that maintain conversation flow
- Support for follow-up questions about venues

2. Personalization

- Stores user preferences (e.g., cuisine preferences)
- Remembers preferences across sessions
- Tailors recommendations based on stored preferences

3. Trip Planning

- Allows users to save venues to a plan
- Generates logical day itineraries
- Provides practical tips and timing suggestions

Implementation Details

Database Creation (`create_database.py`)

```
- Loads venue data from nyc.txt

- Splits text into manageable chunks

- Creates embeddings using OpenAI's embedding model

- Stores vectors in Chroma DB
```

Query Processing (`query_data.py`)

```
- Handles user input processing

- Manages conversation state

- Integrates with OpenAI's API

- Performs similarity search in vector database

- Generates contextual responses
```

Key Classes and Functions

NYCGuide Class

- Main class handling user interactions
- Manages user preferences and saved plans
- Processes queries and generates responses

Key Methods

1. `classify_query()`: Categorizes user input
2. `get_recommendations()`: Generates venue suggestions
3. `handle_follow_up()`: Processes follow-up questions
4. `add_to_plan()`: Saves venues to user's plan

5. `generate_day_plan()`: Creates structured itineraries

Features and Capabilities

1. Venue Recommendations

- Provides detailed information about:
    - Restaurants and eateries
    - Museums and cultural institutions
    - Bookstores and shops
    - Entertainment venues
    - Local attractions

2. Natural Language Understanding

- Processes various query types:
    - Direct questions about specific venues
    - General recommendations
    - Follow-up questions
    - Preference statements

3. Contextual Awareness

- Maintains conversation context
- Understands implicit references
- Provides relevant follow-up information

4. Preference Management

- Captures explicit user preferences
- Stores preferences in JSON format
- Uses preferences for future recommendations

5. Trip Planning

- Saves venues to a plan
- Generates optimized day itineraries
- Considers factors like:
    - Location proximity
    - Operating hours
    - Meal timing
    - Travel time between venues

User Experience

Interaction Flow

1. User starts conversation
2. System offers various interaction options:
    - Ask for recommendations
    - State preferences
    - Save places to plan
    - Request specific information
3. System provides detailed responses
4. User can ask follow-up questions
5. System generates final plan upon exit

Response Types

- Venue recommendations
- Practical information (location, prices, hours)
- Navigation suggestions
- Cultural context
- Local tips and insights

Technical Requirements

Dependencies

- python-dotenv
- langchain
- openai
- chromadb
- tiktoken
- Other supporting libraries

Environment Setup

- Requires OpenAI API key
- Environment variables in .env file
- Chroma DB for vector storage

Future Enhancements

1. Integration with mapping services
2. Real-time availability checking
3. Booking/reservation capabilities
4. Multi-city support
5. User authentication and profiles
6. Integration with review platforms
7. Mobile application development

Best Practices

- Regular database updates
- Error handling for API failures
- User preference validation
- Response quality monitoring
- Performance optimization

Conclusion

CityScape demonstrates the power of combining modern AI technologies with structured data to create an intelligent travel assistant. The system's ability to understand context, remember preferences, and generate practical itineraries makes it a valuable tool for anyone exploring New York City. Its modular architecture allows for future expansions and improvements while maintaining robust current functionality.