AI ANALYTICS AGENT

PROJECT CREATED BY SAGAR MARU (*KAGGLE* || *GITHUB* || *LINKEDIN*)

TABLE OF CONTENTS

---

1. INTRODUCTION

The AI Analytics Agent System is a cutting-edge, modular platform designed to bridge the gap between natural language understanding and actionable data insights. In an era where data is the new oil, organizations need more intuitive and intelligent systems to interact with their vast reservoirs of structured and unstructured information. Traditional methods of querying and data extraction require significant technical knowledge, which often restricts insights to those with deep domain expertise in analytics or SQL.

This project was conceived to empower a wider audience—including business analysts, researchers, and non-technical users—to perform complex data operations through simple natural language queries. By integrating large language models with modern data processing frameworks like LangChain, OpenAI, FAISS, and Pandas, the system provides real-time, conversational access to data.

Whether it's analyzing CSV files, retrieving relevant context from lengthy documents, or executing advanced SQL queries, the system supports a seamless experience across

different data formats. With security, scalability, and extensibility as core principles, this AI-powered agent system stands as a comprehensive solution for modern data analysis needs. Through the Streamlit interface and FastAPI backend, it offers both usability and performance, making it a valuable tool for data-driven decision-making.

---

2. PROJECT GOALS

The main objectives of this project include:

- Enabling intuitive analytics for end users using natural language inputs.
- Creating separate intelligent agents for structured data (DataFrames), unstructured documents (text files), and SQL databases.
- Maintaining a secure, scalable, and modular architecture.
- Building a user-friendly UI and robust backend using FastAPI and Streamlit.
- Integrating LangChain and OpenAI's models to facilitate LLM-based reasoning and retrieval-augmented generation (RAG).

---

3. SYSTEM OVERVIEW

The system comprises three core agents:

- **Data Analytics Agent**: For analyzing tabular data (CSV, Excel) using pandas.
- **Context Agent**: For extracting and searching through unstructured document text using FAISS-based vector stores.
- **SQL Agent**: For querying structured data from uploaded SQLite databases.

Each agent runs through a common API architecture secured by token-based authentication. A responsive Streamlit frontend enables users to interact with the system by uploading files and asking questions in natural language.

---

4. CORE AGENTS

4.1 DATA ANALYTICS AGENT

Implemented in `llm_agent.py`, this agent:

- Accepts CSV or Excel files.
- Converts user queries into pandas code for descriptive analytics.
- Supports multi-turn conversations to build context.
- Uses LangChain and ChatOpenAI for query interpretation.
- Uploads are previewed and tokenized before forwarding for LLM-based analysis.

4.2 CONTEXT AGENT

Implemented in `rag_agent.py`, this agent:

- Accepts .txt files and generates embeddings using OpenAIEmbeddings.
- Stores embeddings in FAISS vectorstore for similarity search.
- Executes RAG queries for deep text understanding.
- Enables users to extract context-rich answers from large documents.

4.3 SQL AGENT

Implemented in `sql_agent.py`, this agent:

- Accepts SQLite databases.
- Uses LangChain's SQLDatabaseChain for query parsing and data retrieval.
- Supports SQL schema introspection, summarization, and reporting.
- Responds with natural language insights and relevant SQL queries.

---

5. SYSTEM ARCHITECTURE

- Modular Python architecture using FastAPI.
- Each agent functions as an independent module, allowing for easy scalability.
- API layer communicates with frontend via secure endpoints.
- LangChain acts as the middleware between user queries and agent logic.

---

6. TECHNOLOGIES USED

- **Language Models**: OpenAI GPT-4o-mini (ChatOpenAI)
- **Frameworks**: LangChain, FastAPI, Streamlit
- **Vector Stores**: FAISS
- **Data Processing**: Pandas
- **Embedding**: OpenAIEmbeddings
- **Authentication**: Bearer Token Auth
- **File Management**: tempfile, file uploader widgets

---

7. AUTHENTICATION SYSTEM

- Every API call is protected with a bearer token mechanism.
- Tokens are generated via a `/auth/token` endpoint using `SECRET_ID` and `SECRET_KEY`.
- Tokens have an expiry duration (default: 1 hour) and are refreshed if expired.
- Token management is handled through Streamlit session state.

8. EMBEDDING AND VECTOR STORE MANAGEMENT

- Text documents are chunked and embedded using `OpenAIEmbeddings`.
- FAISS vectorstore is used for efficient semantic search.
- Embeddings are cached and reused for optimization.
- Index is re-generated only when a new file is uploaded.

---

9. USER INTERFACE

- Built using Streamlit with a modern layout.
- Users can:
- Upload DataFrame (CSV/Excel), Document (TXT), or SQL (SQLite) files.
- Ask natural language queries.
- View responses in conversational UI blocks.
- Session state tracks history for continuity across interactions.

---

10. API IMPLEMENTATION

- Main server logic is in `main.py` using FastAPI.
- APIs:
  - `/auth/token`: Auth token generation
  - `/update-df`: Update DataFrame for analysis
  - `/update-context`: Upload and embed text
  - `/update-sql`: Register SQL DB
  - `/chat, /context, /sql`: Handle respective queries
- Run via terminal: `uvicorn app.main:app --reload`

---

11. ENVIRONMENT CONFIGURATION

`.env` file contains:

- `OPENAI_API_KEY`: For model access
- `SECRET_ID` and `SECRET_KEY`: For token generation

Sensitive keys are loaded via st.secrets in production.

---

12. KEY DESIGN DECISIONS

- **LangChain Usage**: To manage chains for each agent, including RAG and SQL parsing.
- **Session State**: Used for maintaining continuity and chat histories across agents.
- **Separation of Logic**: Isolated core agent logic into different files for clarity.
- **Caching**: Default datasets/documents are cached to avoid reloading.
- **UI Responsiveness**: File preview, markdown messages, and loading states handled clearly.

---

13. CHALLENGES FACED

- **Token Expiry Handling**: Ensuring auto-refresh without breaking UI flow.
- **Chat History Loss**: Ensuring chat histories persist across file uploads by tracking file names.
- **RAG Memory Management**: Handling large documents efficiently within memory limits.
- **Session State Sync**: Maintaining state between frontend and backend especially during file changes.
- **Model Rate Limits**: Implementing retry logic for OpenAI APIs.

---

14. FUTURE ENHANCEMENTS

- 🔐 Role-based user authentication (admin/user)
- 📈 Integration with BI dashboards like Power BI/Tableau
- ☐ Fine-tuned models or local LLMs for privacy
- 🌐 Expand to cloud vectorstores (e.g., Pinecone)
- 📤 File summarization before vector embedding
- 🌲 Real-time DB connectors (PostgreSQL, MySQL)
- 📚 PDF document support for context agent
- 🎨 Enhanced Streamlit UI with theme selector and multi-file uploads

---

15. CONCLUSION

The AI Analytics Agent System signifies a major step forward in making data analytics more accessible, intelligent, and user-friendly. By leveraging the power of large language models and combining it with robust backend data processing tools, the system bridges the divide between technical complexity and business insight. Its modular design allows each agent—be it for tabular data, text documents, or databases—to operate independently while contributing to a cohesive user experience.

The integration of technologies like FAISS for vector search, LangChain for LLM chaining, and FastAPI for backend flexibility ensures that the system remains performant and scalable. The front-end, built with Streamlit, provides a clean, interactive platform that makes the system approachable for both technical and non-technical users alike.

As data continues to grow in volume and complexity, tools like this AI Analytics Agent System become essential for informed decision-making. With room for future enhancements like real-time DB support, PDF embedding, and cloud deployment, the system lays a strong foundation for AI-driven data applications. Its success lies in its simplicity of use, depth of capability, and adaptability to future AI advancements.

APPENDICES

A. FILE STRUCTURE

```
.
├── app
│   ├── main.py
│   ├── llm_agent.py
│   ├── rag_agent.py
│   ├── sql_agent.py
├── ui
│   ├── streamlit_app.py
├── data
│   ├── retail_transactions_dataset.csv
│   ├── retail_transactions_description.txt
│   ├── retail_transactions_data.db
├── .env
├── requirements.txt
```

B. RUN INSTRUCTIONS

- API Server:
  - `uvicorn app.main:app --reload`
- Streamlit UI:
  - `streamlit run ui/streamlit_app.py`

C. KEY DEPENDENCIES

- fastapi
- uvicorn
- pandas
- streamlit
- openai
- langchain
- langchain-core
- langchain-community
- langchain-openai
- langchain-experimental
- faiss-cpu
- tiktoken
- tabulate
- sqlalchemy
- pymysql
- psycopg2-binary
