Documentation for the AI Data Analysis Agent

Overview

This script acts as an **AI-powered data analysis agent**, allowing users to interact with datasets using natural language queries.. It uses an LLM (Large Language Model) via Groq API to generate Python code based on user questions and executes it safely to return results or visualizations.

🔧 Key Features

- Load `.csv` or `.json` datasets
- Natural language query support
- Automatic classification of query complexity
- Safe execution of generated Python code
- Visualization generation (histograms, heatmaps, etc.)
- Smart suggestions for possible analyses
- Saves visualizations to output folder with unique timestamps

📦 Components

1. Core Modules

- `pandas`, `numpy`: For data manipulation
- `matplotlib.pyplot`, `seaborn`: For visualization
- `langchain_groq`, `ChatGroq`: To connect to Groq's LLM
- `dotenv`: For loading API keys from `.env`

2. Tools

- `smart_analysis_tool`: Handles basic and complex analysis queries
- `visualization_tool`: Generates and displays plots
- `suggestion_tool`: Provides suggested analysis questions

3. Utilities

- Dataset encoding detection
- Code safety checks (blocks dangerous operations)
- Output formatting and truncation
- Image viewer (cross-platform)

🔄 Workflow

High-Level Flow

```
Start → Load Dataset → Initialize Tools & LLM → Main Loop:
    ↓
User Query → Classify → Execute Code/Generate Plot → Return Result
    ↓
Repeat Until Exit
```

📈 Smart Analysis Tool Flow

```
User Query → Classify as Simple or Complex
            ↓
       Simple → Execute basic pandas code
            ↓
       Complex → Generate + execute advanced code → Summarize result
```

📊 Visualization Tool Flow

```
User asks for visualization → Generate Matplotlib/Seaborn code
                            ↓
```

```
                    Execute code
                        ↓
Save plot to output folder with unique timestamp
                        ↓
                 Open image file
                        ↓
            Return path + insights
```

🗺 Flowchart

```
+-----------------------+
|    Start Program      |
+-----------------------+
           |
           v
+-----------------------+
|  Load Dataset (.csv/.json)|
|  Detect Encoding      |
+-----------------------+
           |
           v
+-----------------------+
| Initialize LLM & Tools|
| (Analysis, Viz, Suggest)|
+-----------------------+
           |
           v
+-----------------------+
| Ask for User Query    |
+-----------------------+
           |
           v
+-----------------------+
| Classify Query Type   |
| (Simple / Complex / Viz)|
+-----------------------+
```

```
           |
     +-----+--------+
     |              |
     v              v
+----------------+ +------------------+
| Execute Pandas | | Generate Plot Code |
| Code & Return  | | Save Image        |
| Result         | | Show Summary      |
+----------------+ +------------------+
     |
     v
+-----------------------+
| Loop Until Exit       |
+-----------------------+
```

Installation

Requirements:

```
langchain==0.3.26
langchain_groq==0.3.4
matplotlib==3.10.3
numpy==2.3.1
pandas==2.3.0
python-dotenv==1.1.1
seaborn==0.13.2
```

Environment Setup:

Create a `.env` file in the root directory:

```
GROQ_API_KEY=your_api_key_here
```

🗂 Folder Structure

```
project/
│
├── output/                # Generated plots are saved here
├── .env                   # Stores API key
├── main.py                # Main script
```

📌 Usage Guide

1. Run the Script

```
python main.py
```

2. Load a Dataset

When prompted:

```
📂 Enter dataset path: data.csv
```

3. Ask Questions

Examples:

```
❓ Query: show head
❓ Query: describe data
❓ Query: visualize correlation matrix
❓ Query: check missing values
```

4. Get Suggestions

Type:

❓ `Query: suggestions`

To see recommended analysis steps.

Special Commands

Command	Description
`suggestions`	Get list of suggested analyses
`exit`	Quit the program

⚙ Configuration

You can customize these settings at the top of the script:

```
MAX_DISPLAY_ROWS = 20          # Max rows shown in output
MAX_OUTPUT_LENGTH = 1000       # Truncate long outputs
```

Visualizations

All visualizations are saved in the `output/` folder with filenames like:

`plot_20250405_123456.png`

They include:

- Histograms
- Correlation matrices
- Scatter plots
- Bar charts

- Line graphs

🛡️ Safety Features

- Blocks unsafe code patterns (`os.system`, `eval`, etc.)
- Limits output size
- Prevents infinite loops through timeout handling
- Input sanitization

Error Handling

The system handles:

- Dataset encoding errors
- API rate limits
- Authentication failures
- Timeout issues
- Invalid user queries

Errors are gracefully displayed with emoji warnings.

Example Queries

Basic Exploration

- `show first 5 rows`
- `describe the data`
- `how many rows and columns`

Statistical Analysis

- `check missing values`
- `find average salary by department`

- `correlation between age and income`

Visualization

- `create histogram of age`
- `visualize correlation matrix`
- `bar chart of categories`

Conclusion

This tool bridges the gap between natural language and data science, allowing both technical and non-technical users to explore datasets effortlessly. With smart suggestion capabilities and robust error handling, it’s suitable for exploratory data analysis in research, business intelligence, or educational contexts.

Integration with Route Agent

To integrate the Data Analysis Agent as a tool within a larger **Router Agent**, follow these steps:

```
def run_data_analysis(query: str) -> str:
    """
    Wrapper function to be used as a Tool in a router agent.
    Accepts a query string, runs it through the existing agent, and
returns the output.
    """
    global agent
    if not global_df or not schema:
        return "❌ Dataset not loaded. Please load a dataset before
querying."
    if not agent:
        return "❌ Agent not initialized. Please check LLM setup."
    try:
        result = agent.invoke({"input": query})
        return result["output"]
    except Exception as e:
        return f"❌ Error running analysis: {str(e)}"
```

Integrate into Router Agent

Now include this tool along with others in the router agent:

```
from data_analysis_agent import run_data_analysis

data_analysis_tool = Tool(

                name="DataAnalysisAgent",

                 func=run_data_analysis,

                 description="Use this tool for analyzing CSV/JSON
datasets. It supports both basic and advanced analysis queries." )
```

```
router_tools = [
    data_analysis_tool,
    other_tool_1,
    other_tool_2,
    # Add more tools here
]

router_agent = initialize_agent(
    tools=router_tools,
    llm=router_llm,  # A different or same LLM instance
    agent="zero-shot-react-description",  # Or use a multi-agent
router
    verbose=True
)
```