

Assignment 3

AI-Powered SQL Query & Visualization System

Multi-Agent Analytics Platform using PostgreSQL, Polars, LangChain, LangGraph, and Streamlit

1. Introduction

This document provides complete technical documentation for an AI-driven analytics system that converts natural-language questions into SQL queries, executes them against a PostgreSQL database, generates intelligent visualization code, validates and executes that code, and displays the final output using Streamlit.

The system uses a multi-agent workflow orchestrated through LangGraph and integrates with a Streamlit-based user interface.

It satisfies all assignment requirements, including:

- Creating a PostgreSQL database in 3NF
- Loading CSV data using Polars
- Building a five-agent architecture
- Integrating LangChain, LangGraph, and Streamlit
- Producing both textual and graphical outputs

2. System Overview

The platform processes user input through a pipeline of **five coordinated AI agents**:

1. **SQL Generator Agent** – Converts English questions into SQL and executes them on PostgreSQL.
2. **Visualization Code Generator Agent** – Produces Python visualization code based on the query and DataFrame.
3. **Code Validator Agent** – Ensures the generated code is valid, safe, and uses correct DataFrame columns.
4. **Secure Execution Agent** – Executes the validated code in a protected Python environment and captures the output.

5. **Streamlit UI Agent** – Coordinates all agents and displays results in a web interface.

Each agent is implemented in its own Python file, and the system is modular, secure, and scalable.

3. Technologies Used

- **Programming Language:** Python
- **Database:** PostgreSQL
- **CSV Loader:** Polars
- **AI Framework:** LangChain + LangGraph
- **LLM Model:** GPT-4o Mini
- **Visualization Libraries:** Matplotlib, Plotly
- **Web Framework:** Streamlit
- **Security Tools:** Python AST for code safety validation
- **Environment Management:** python-dotenv

4. System Architecture

The system operates in a sequential pipeline:

Step 1 – User Input

The user enters a natural-language question in the Streamlit UI.

Step 2 – SQL Generation & Execution

- The SQL Generator Agent interprets the question.
- It dynamically reads the database schema.
- Generates a valid PostgreSQL SELECT query.
- Executes the SQL using Polars + SQLAlchemy.
- Returns the result as a pandas DataFrame.

Step 3 – Visualization Code Generation

- The Visualization Agent receives the DataFrame.
- Detects chart type based on:
 - Query intent
 - Row count
 - Data types
 - Presence of categorical or datetime columns
- Generates executable Matplotlib code using real DataFrame columns.

Step 4 – Code Validation

- The Validator Agent scans the generated Python code.
- Ensures:
 - No unsafe imports
 - No exec/eval usage
 - No placeholder variable names
 - Only real DataFrame columns are referenced
- Returns a JSON object:
 - “is_valid”: true/false
 - “feedback”: explanation

Step 5 – Secure Execution

- The Runner Agent safely executes the visualization code inside a sandbox.
- Captures the rendered Matplotlib output as PNG bytes.
- Prevents filesystem access or dangerous operations.

Step 6 – Streamlit Display

- The final visualization or table is displayed in the Streamlit UI.
- Intermediate steps (SQL, DataFrame preview, generated code, validation output) are also shown.

5. Folder Structure

```
project/
|
└── doc/
    └── readme.md
|
└── src/
    ├── agent_sql_generator.py
    ├── agent_code_generation.py
    ├── agent_code_validator.py
    ├── agent_code_runner.py
    ├── streamlit_sql_agent.py
    ├── create_table_and_load_CSV.ipynb
    ├── RobotVacuumDepot_MasterData.csv
    ├── .env.example
    └── requirements.txt
```

6. Environment Setup

6.1 Virtual Environment Creation

```
python -m venv venv
venv\Scripts\activate  (Windows)
source venv/bin/activate (Mac/Linux)
```

6.2 Install Dependencies

```
pip install -r src/requirements.txt
```

6.3 Environment Variables

Create a .env file inside src/:

PG_HOST=localhost

PG_PORT=5433

PG_USER=postgres

PG_PASSWORD=admin

PG_DB=robot_vacuum

OPENAI_API_KEY=your_api_key_here

Do **not** submit .env.

Instead, submit .env.example.

7. PostgreSQL Setup

1. Create database:
2. CREATE DATABASE robot_vacuum;
3. CREATE SCHEMA robot_vacuum;
4. Run the Jupyter notebook:
 - o create_table_and_load_CSV.ipynb
 - o This notebook:
 - Creates all tables in **3NF**
 - Loads CSV with **Polars**
 - Inserts data into PostgreSQL

8. Running the Application

From the project root folder:

streamlit run src/streamlit_sql_agent.py

The browser UI will open and show:

- Input box for natural-language questions

- Buttons for each agent stage
- SQL output
- DataFrame preview
- Visualization code
- Validation results
- Final chart/table

9. Supported Question Types

Text/Table Questions

- “Which warehouses are below restock threshold?”
- “Which manufacturers have the highest review rating?”
- “Which ZIP code has the most delayed deliveries?”

Graph Questions

- “Plot monthly revenue trends over time”
- “What is the percentage distribution of delivery statuses?”
- “Compare average shipping cost by carrier”
- “Show average review rating per manufacturer”

The system supports:

- Line charts
- Bar charts
- Scatter plots
- Histograms
- Pie charts
- Table fallback for unsupported or minimal data

10. Security Measures

The execution environment is protected by:

- AST-based code safety validation
- Blocking dangerous imports (os, subprocess, sys, etc.)
- No exec/eval allowed
- Code executed in isolated namespace
- Only Matplotlib operations permitted
- No file I/O allowed

11. Submission Requirements (All Met)

The project satisfies all assignment requirements:

- ✓ src and doc directories
- ✓ requirements.txt included
- ✓ .env.example included
- ✓ Jupyter notebook included
- ✓ All agents implemented
- ✓ LangGraph pipeline implemented
- ✓ Streamlit front-end implemented
- ✓ PostgreSQL used with 3NF schema
- ✓ CSV loaded via Polars
- ✓ Charts + tables both supported

12. Conclusion

This project demonstrates a fully automated AI analytics workflow combining:

- Natural-language processing
- SQL synthesis
- Intelligent visualization

- Code validation
- Secure execution
- UI presentation

It is modular, scalable, and meets all academic and technical requirements.