RTGS-Style AI Analyst: System Documentation

Version: 1.0

Date: September 7, 2025

Author: Nulu Veera Manicharan

1.0 Abstract

The RTGS-Style AI Analyst is a prototype of a terminal-based agentic system designed to bridge the gap between raw, unstructured public data and actionable, evidence-based insights for policymakers. The system ingests any tabular dataset, leverages a Large Language Model (LLM) to dynamically create a custom plan for data cleaning and analysis, executes that plan, and generates a comprehensive, human-readable report complete with visualizations and AI-driven interpretations. It is built to be a robust, "CLI-first" tool that automates the most time-consuming aspects of data analysis, allowing a decision-maker to go from raw data to insight with a single command.

2.0 Introduction & Problem Statement

The primary challenge in data-driven governance is the **"last mile problem."** While government portals, such as the Telangana Open Data Portal, provide a wealth of public data, this data is often raw, messy, and not immediately usable for analysis. Policymakers and their staff are required to invest significant time and technical expertise in cleaning, standardizing, and transforming this data before any meaningful insights can be extracted.

This project addresses that challenge directly. The RTGS AI Analyst acts as an automated data scientist, taking on the role of ingesting, cleaning, analyzing, and interpreting data. The system is designed to be fully autonomous after receiving an input file, producing a final analytical brief that not only shows *what* the data says but also explains *why* the findings are relevant, making it a true decision-support tool.

3.0 System Architecture

The system is built on a modular, **agentic architecture** orchestrated by the **LangGraph** framework. This design moves away from a rigid, monolithic script and instead models the workflow as a graph of specialized, independent agents. Each agent has a single responsibility and communicates with the others by passing a central GraphState object, which contains all the data, plans, and artifacts.

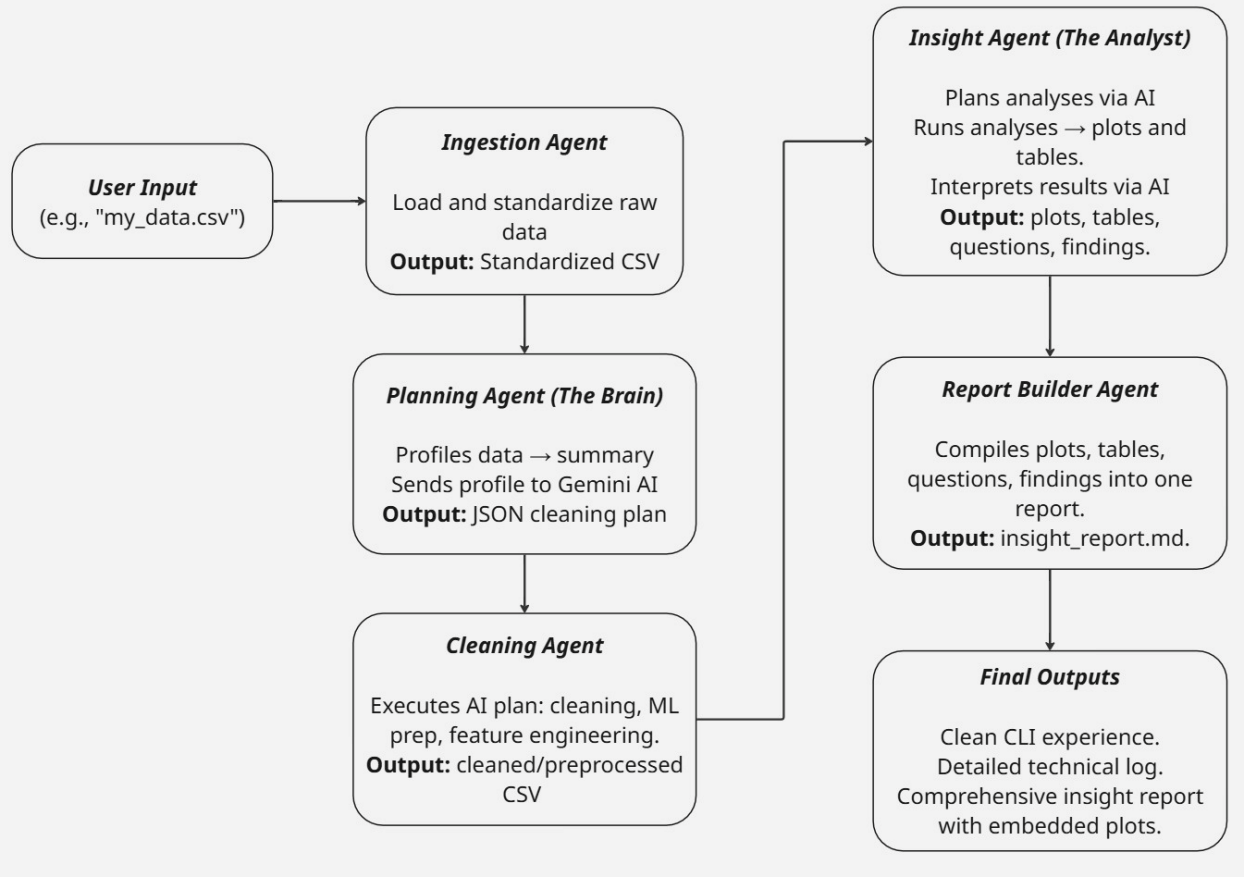
This state-driven approach makes the system highly extensible, robust, and easy to debug.

The Agents:

- **The Ingestion Agent (ingestion.py):** The initial point of contact with the data. This agent is responsible for robustly loading datasets, intelligently handling both well-formed and malformed CSV files (including those with encoding issues or structural errors). Its primary output is a standardized DataFrame with clean column names.
- **The Planning Agent (The Brain) (ai_planner.py, profiler.py):** The heart of the system's intelligence. This agent first performs a comprehensive diagnostic check by programmatically profiling the data to create a detailed statistical summary. It then sends this profile to the Google Gemini AI with a sophisticated prompt, instructing the AI to act as an expert data scientist. The AI's response is a custom, multi-step JSON plan for cleaning, transformation, machine learning preprocessing, and feature engineering, complete with a data-driven **reason** for every proposed action.
- **The Cleaning & Preprocessing Agent (cleaning.py):** The workhorse of the pipeline. This agent is a dynamic executor that reads the AI-generated plan from the state. It is equipped with a library of "power tools"—including text cleaning utilities, custom functions, and scikit-learn modules—to perform a wide range of advanced operations, from removing currency symbols and scaling numeric features to encoding categorical data for machine learning.
- **The Insight & Interpretation Agent (insight.py):** The "Chief Analyst." This is a two-stage agent. It first performs a high-level analysis of the final, cleaned data and makes an AI call to generate a **list of valuable analytical questions** to investigate. It then loops through these questions, executes each analysis (e.g., correlations, distributions, group-by summaries), generates a plot, and makes a final, **batched AI call** to interpret the results, providing a concise, data-driven finding for each visualization.
- **The Report Builder Agent (insight_report.py):** The final scribe. This agent gathers all the artifacts from the preceding agents—the AI-generated dataset summary, the analytical questions, the data tables, the plots, and the AI-generated interpretations—and compiles them into a single, professional, multi-page Markdown report.

4.0 Process Flow

The flow of data and logic through the system is sequential and managed by the LangGraph state machine.



5.0 Technology Stack

- **Core Language:** Python 3.10+
- **Agentic Framework:** LangGraph (for orchestration and state management)
- **AI Model:** Google Gemini (gemini-2.0-flash)
- **Data Manipulation:** pandas
- **Machine Learning Preprocessing:** scikit-learn (for scaling and encoding)
- **Natural Language Processing:** NLTK (for stopwords in word frequency analysis)
- **Command-Line Interface:** Typer
- **Console UI & Logging:** rich, Python logging module
- **Plotting & Visualization:** Matplotlib, seaborn

6.0 Key Features & Capabilities

- **Dynamic Preprocessing:** The system is not limited to a fixed set of rules. It adapts to any tabular dataset, with the AI generating a custom plan for everything from text cleaning and categorical normalization to advanced machine learning preprocessing like Min-Max scaling.
- **Automated Feature Engineering:** The AI proactively suggests and creates new, valuable columns from the existing data (e.g., calculating BMI from height and weight, or converting age-in-days to age-in-years).
- **Automated EDA & Interpretation:** The system doesn't just create plots; it performs a full suite of Exploratory Data Analysis (EDA) tasks. It uses the AI to first decide *what* to analyze and then, critically, to interpret the results, providing human-like observations about what the findings mean.
- **Robustness & Error Handling:** The system is designed to be resilient. It features a robust ingestion agent that can handle malformed files, an intelligent logger that separates user feedback from technical tracebacks, and a master error handler that ensures the application always exits gracefully with a clear, actionable message.
- **Transparency & Auditability:** Every run produces two key reports: a detailed technical log file for developers and a professional, human-readable insight_report.md that explains the dataset, the questions asked, the data supporting the findings, the visualizations, and the final AI-driven interpretations.
- **Clearing the past files:** Every run clears the old run files so that there will not be a problem in understanding which output belongs to which input. So, if you want to store the result you can save a copy of the cleaned dataset files and insight, run reports.