Framework - LangChain  
LLM - OpenAI ( gpt-4o-mini )  
Programming Language - Python  
  
The tool is focused on building synthetic and mock data on following domains :-

* Data Template / Product Template
* Products
* Customers
* Orders
* Order Details / Order LineItems

Detailed Approach on each of the domains :-

1. **Product Template**

Utilizes OpenAI's GPT-4o-mini LLM to craft dynamic, domain-specific DataFrame schemas through a user-driven synthesis workflow. Users define the target domain and column requirements, enabling the LLM to generate tailored column structures with contextual precision. The tool includes an intuitive interface for schema refinement, featuring:

* **Adaptability**: Real-time adjustments for precise alignment with domain-specific requirements.
* **Column Customization**: Interactive and dynamic exclusion of unwanted columns via checkboxes and changing the predefined data type options from the dropdown.
* **Reference Data Integration**: Users can provide sample data for each column to influence product generation in the later stage.

Finally, users can export the refined schema as a configuration JSON file, supporting seamless integration into data workflows. This system exemplifies the synergy of LLM-powered adaptability and user-centric design in data generation.

1. **Products**

The system enables users to generate synthetic product datasets by leveraging OpenAI's LLM (e.g., GPT-4o-mini). Users interact through a UI where they:

* **Domain Customization**: Select a product domain and specify its variant/type.
* **Template Integration**: Optionally upload a JSON schema defining product attributes.
* **Data Generation**: Specify the volume of synthetic data required, which the LLM processes based on inputs.
* **Output Delivery**: Generate and export the structured product dataset as a downloadable CSV.

This setup effectively combines domain-driven design with AI-powered data synthesis for scalable and flexible dataset creation.

1. **Customers**

The tool leverages a user-driven interface with dynamic data generation capabilities. Upon selecting a customer from the domain dropdown, the user also specifies the region for which mock customer address data needs to be generated. This selection, alongside the number of records required, triggers a custom Python service. The service uses the **Faker** library to generate 50 mock addresses corresponding to the selected region. Additionally, the Python service ensures the accuracy of names based on the customer's gender, adjusting first and last names accordingly.

The system also enforces proper data typing, with fields such as phone numbers and employee numbers being generated as integers. The mock addresses are then randomly associated with the generated customer records, ensuring variety and realism.

This architecture is highly reliant on Python's data manipulation capabilities and the **Faker** library's ability to produce realistic, region-specific mock data for various customer-related fields. The overall flow integrates backend logic with frontend user input, ensuring that generated data matches the user’s specifications while adhering to realistic formats.

1. **Orders**

The tool leverages Python to generate synthetic, relationally consistent order data. It incorporates a **domain-specific selector** and a record count parameter, alongside a **file ingestion interface** allowing multi-CSV uploads. These files contribute domain-specific static keys (e.g., customerNumber) to instantiate primary or foreign key relationships.

The Order data generation mechanism uses the **Faker library** for creating dynamic, probabilistic mock data, enriched with **custom deterministic logic** to map CSV-derived keys into the synthesized records. This design enables the emulation of relational database schemas by maintaining referential integrity between the generated datasets, facilitating their deployment in environments requiring realistic yet synthetic database representations for testing, ETL workflows, or schema validation.

1. **Order Details / Order LineItems**

The tool is a Python-driven system for generating relational datasets, emphasizing interdependencies and domain-specific integrity. Users configure order detail records via a domain selector, specifying the desired count and uploading a dependent CSV (such as an order file) if needed. The Python service enforces a validation constraint, ensuring the number of generated order detail records does not exceed the order count in the uploaded CSV, preserving referential integrity.

Mock data is synthesized using the **Faker library** combined with custom logic, which randomly assigns single or multiple product codes to each order detail record. This creates relational data structures suitable for database setups, ensuring the generated datasets maintain logical connections to other data sources, making them usable for database initialization, testing, or prototyping scenarios.