DataAnalystAgent
Documentation

Team Caramel Popcorn

June 1, 2025

Project Overview

DataAnalystAgent is an autonomous AI agent that leverages advanced language models to analyze SQL databases, generate summaries, answer questions, and create visualizations—all through a natural language interface. No SQL or coding required!

Data Schema Overview

The DataAnalystAgent is designed to work with SQL databases, such as the included `Chinook` and `Northwind` sample databases. Understanding the schema (tables, columns, and relationships) is essential for generating meaningful queries and insights.

Chinook Database Schema

The **Chinook** database models a digital media store, similar to iTunes. It contains tables for artists, albums, tracks, customers, invoices, and more.

Table	Description	Key Columns
Artist	Music artists	ArtistId, Name
Album	Music albums	AlbumId, Title, ArtistId
Track	Music tracks	TrackId, Name, AlbumId, GenreId, MediaTypeId
Genre	Music genres	GenreId, Name
MediaType	Media types (e.g., MPEG, AAC)	MediaTypeId, Name
Customer	Customer information	CustomerId, FirstName, LastName, Email
Invoice	Sales invoices	InvoiceId, CustomerId, InvoiceDate, Total
InvoiceLine	Line items for each invoice	InvoiceLineId, InvoiceId, TrackId, UnitPrice, Quantity
Employee	Employees and support reps	EmployeeId, FirstName, LastName, ReportsTo

Key Relationships:

- `Album.ArtistId` → `Artist.ArtistId`
- `Track.AlbumId` → `Album.AlbumId`
- `Track.GenreId` → `Genre.GenreId`

- `Track.MediaTypeId` → `MediaType.MediaTypeId`
- `Invoice.CustomerId` → `Customer.CustomerId`
- `InvoiceLine.InvoiceId` → `Invoice.InvoiceId`
- `InvoiceLine.TrackId` → `Track.TrackId`
- `Customer.SupportRepId` → `Employee.EmployeeId`

Northwind Database Schema

The **Northwind** database is a classic business sample database, modeling a trading company with orders, products, customers, employees, and suppliers.

Table	**Description**	**Key Columns**
Customers	Customer information	CustomerID, CompanyName, ContactName
Orders	Orders placed by customers	OrderID, CustomerID, EmployeeID, OrderDate
Order Details	Line items for each order	OrderID, ProductID, UnitPrice, Quantity
Products	Products for sale	ProductID, ProductName, SupplierID, CategoryID
Categories	Product categories	CategoryID, CategoryName
Suppliers	Product suppliers	SupplierID, CompanyName
Employees	Employees and reporting structure	EmployeeID, LastName, FirstName, ReportsTo
Shippers	Shipping companies	ShipperID, CompanyName

Key Relationships:

- `Orders.CustomerID` → `Customers.CustomerID`
- `Orders.EmployeeID` → `Employees.EmployeeID`
- `Order Details.OrderID` → `Orders.OrderID`
- `Order Details.ProductID` → `Products.ProductID`
- `Products.SupplierID` → `Suppliers.SupplierID`
- `Products.CategoryID` → `Categories.CategoryID`
- `Orders.ShipVia` → `Shippers.ShipperID`
- `Employees.ReportsTo` → `Employees.EmployeeID`

How the Agent Uses the Schema:

- The agent inspects table and column names to generate SQL queries.
- It uses foreign key relationships to join tables and provide richer answers.
- Schema information is used to generate summaries, visualizations, and to validate user questions.

You can view your database schema using the `/api/schema/<database_name>` endpoint or by browsing the SQL files in the `database/` folder.

Example Questions and Outputs

Example 1: Top Customers by Order Value

Question:

List out the top 5 customers who have highest value of products?

Agent Output (JSON):

```
{
  "result_type": "aggregated summary",
  "visual_goal": "bar chart",
  "question_description": "What are the top 5 customers with the
     highest total order value?",
  "answer_description": "The top 5 customers ranked by the total
     value of their orders.",
  "plot_arguments_description": {
    "x": "CustomerID",
    "y": "TotalValue"
  }
}
```

Visualization: A bar chart showing `CustomerID` on the X-axis and `TotalValue` on the Y-axis.

Example 2: Product Sales by Category

Question:

Show total sales for each product category as a pie chart.

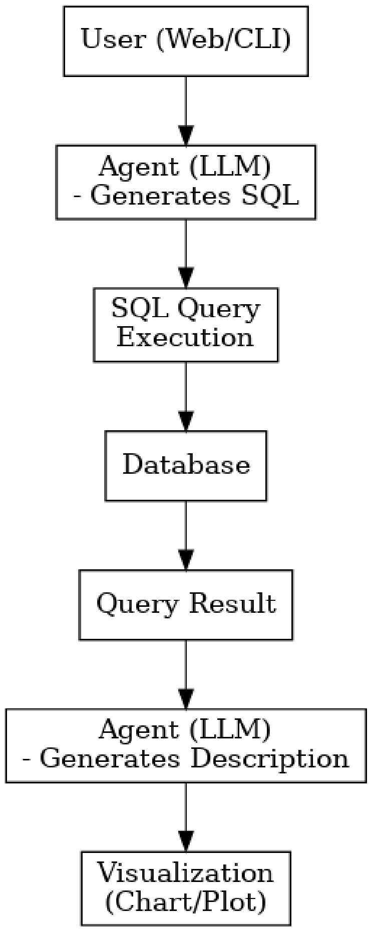
Agent Output (JSON):

```
{
  "result_type": "category summary",
  "visual_goal": "pie chart",
  "question_description": "Total sales by product category.",
  "answer_description": "Distribution of sales across categories.",
  "plot_arguments_description": {
    "labels": "CategoryName",
    "values": "TotalSales"
  }
}
```

Visualization: A pie chart showing the proportion of sales for each category.

System Architecture Explanation

High-Level Architecture



- **User Interface:** Web interface (`templates/index.html`) or CLI (`main.py`)
- **Flask Backend (`app.py`):** Handles API requests, routes, and serves the UI
- **Core Agent (`core/`, `agent_types.py`):** Orchestrates LLM, SQL, and visualization tools
- **LLM & Tools (`models.py`, `tools.py`, `prompts.py`):** Generates SQL, interprets results, creates summaries and chart configs
- **Visualization (`visualizer.py`):** Plots charts using matplotlib/seaborn (CLI) or Chart.js (web)
- **Database (`database/`):** Stores user or example databases (e.g., Chinook, Northwind)

Detailed Architecture Components

1. **User Entry Point**
    - **Components:**
        - Web Interface (`templates/index.html`)
        - CLI Interface (`main.py`)
    - **Functions:**
        - Accept natural language questions
        - Display results and visualizations
        - Handle database selection

2. **Query Generation**
    - **Components:**
        - LLM-based Agent (`core/agent_types.py`)
        - Query Generator (`tools.py`)
    - **Functions:**
        - Analyze user's natural language question
        - Understand database schema
        - Generate appropriate SQL query
        - Validate query structure

3. **Database Interaction**
    - **Components:**
        - SQL Execution Engine (`core/`)
        - Database Connector
    - **Functions:**
        - Execute generated SQL queries
        - Handle database connections
        - Process result sets
        - Manage error handling

4. **Result Analysis**
    - **Components:**
        - LLM Insight Generator (`core/insight_generator.py`)
        - Result Processor

- **Functions:**
  - Process query results
  - Generate natural language descriptions
  - Identify key insights
  - Determine visualization approach

5. Visualization Generation

- **Components:**
  - Chart.js (Web Interface)
  - matplotlib/seaborn (CLI)
  - Visualization Engine (`visualizer.py`)
- **Functions:**
  - Generate appropriate chart types
  - Handle data formatting
  - Create interactive visualizations
  - Export static images (CLI)

Data Flow Process

1. User submits natural language question
2. Agent processes question and generates SQL
3. Query is executed against database
4. Results are processed and analyzed
5. Insights are generated from analysis
6. Appropriate visualizations are created
7. Complete response is returned to user

Getting Started

1. **Clone the repository**

   ```
   git clone https://github.com/yourusername/DataAnalystAgent.git
   cd DataAnalystAgent
   ```

2. **Install dependencies**

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

3. **Prepare your database**

   - Place your SQLite file in the `database/` folder.

4. **Run the agent**

    - For web:

      ```
      python app.py
      ```

    - For CLI:

      ```
      python main.py
      ```

Project Structure

```
DataAnalystAgent/

 core/           # Core logic for agent orchestration, SQL execution, and insight genera
 database/       # Example and user-uploaded databases
 utils/          # Utility functions and helpers used across the project
 templates/      # HTML templates for web interface (if used)
 __init__.py     # Marks the directory as a Python package
 agent_types.py  # Defines different agent types and their configurations
 app.py          # Flask web application entry point (for web interface)
 config.py       # Configuration settings (database, dialect, etc.)
 main.py         # Command-line entry point for running the agent
 models.py       # LLM model configuration and setup
 prompts.py      # System and tool prompts for the agent
 requirements.txt# Python dependencies
 schemas.py      # Database schema definitions and helpers
 tools.py        # Tool definitions (SQL, visualization, etc.)
 README.md       # Project overview and instructions
 ...
```

Customization and Contributing

- **Add new tools:** Extend `tools.py` for more capabilities.
- **Change prompts:** Edit `prompts.py` for different agent behaviors.
- **Web interface:** Use `app.py` and `templates/` for a Flask-based UI.

Contact

For questions or support, open an issue or contact the maintainer.