Data Analysis Chatbot with Guardrails

Overview

This chatbot allows users to interact with a CSV-based dataset using natural language queries. It leverages **LangChain**, **OpenAI GPT models**, **Pandas**, and **Guardrails** to process user queries, validate responses, and ensure safe interactions. The bot supports querying, data visualization, and answering domain-specific questions related to **finance**, **delivery**, and **restaurant data**.

Code Main File: pandas_dataframe_agent.py

Key Features

1. **Natural Language Query Processing**:
   - Users can query data from a CSV file (`combined_data_v6.csv`) using simple, conversational language.
2. **Data Visualization**:
   - Generates charts and visualizations using **Plotly**.
3. **Guardrails for Valid Responses**:
   - Ensures responses adhere to valid topics.
4. **Moderation API Integration**:
   - Screens user input for inappropriate content.
5. **Advanced Query Execution**:
   - Executes Python code for complex calculations using **Python REPL**.
6. **Error Handling and Logging**:
   - Logs errors and events to a log file for debugging and monitoring.

System Flow

1. User Interaction Flow

1. **User Input**:
   - User provides a query related to the dataset.
2. **Moderation Check**:

    - Input is screened using OpenAI Moderation API to ensure it doesn't contain inappropriate or unsafe content.
3. **Query Processing**:
    - The bot processes the query using LangChain's **Pandas DataFrame Agent**.
4. **Guardrails Validation**:
    - Ensures the generated response is relevant to supported topics (finance, delivery, restaurant).
5. **Response Generation**:
    - Outputs results or visualizations based on the dataset.

---

2. Technical Flow

1. **Environment Setup**:
    - Load configuration variables (API keys, file paths) from `.env` and `config.yaml`.
2. **Data Loading**:
    - Load CSV data into a **Pandas DataFrame**.
3. **Agent Initialization**:
    - Create a **Pandas DataFrame Agent**:
        - Includes tools like Python REPL for calculations.
        - Provides a detailed prompt specifying the dataset columns and expected behavior.
4. **LLM Interaction**:
    - **OpenAI GPT (gpt-4o)** processes queries, performs tasks, and generates responses.
5. **Validation**:
    - Guardrails restrict responses to predefined topics.
    - Moderation API ensures input and output adhere to ethical guidelines.
6. **Output Formatting**:
    - Responses are formatted for readability.
    - Visualizations are rendered using **Plotly**.
7. **Error Handling**:
    - Logs any errors or invalid inputs for debugging.

Key Methods and Functions

1. Initialization Functions

- **`load_yaml(file_path)`**:
  - Reads configurations from `config.yaml`.
  - Returns API keys and settings as a dictionary.
- **`load_dotenv()`**:
  - Loads environment variables, specifically `OPENAI_API_KEY`.
- **`create_pandas_dataframe_agent()`**:
  - Initializes the agent with a DataFrame and a detailed prompt.
  - Supports Python REPL for calculations and visualizations.

2. Query Handling

- **`ask_dataframe_agent(user_question)`**:
  - Processes user questions and generates responses.
  - Steps:
    1. Cleans and pre-processes the question.
    2. Call the agent with the processed query.
    3. Validates the response using Guardrails.
    4. Returns the final response or an error message.

3. Moderation

- **`check_moderation(text)`**:
  - Uses OpenAI Moderation API to validate user input.
  - Flags inappropriate content and logs flagged categories.

4. Response Validation and Formatting

- **`validate_question(question)`**:
  - Ensures the question doesn't contain forbidden operations like `delete`, `drop`, or `remove`.
- **`format_response(response)`**:
  - Cleans and formats the chatbot's response for better readability.

5. Logging

- **`logging.basicConfig()`**:
  - Configures logging to track events and errors in `pandas_agent.log`.

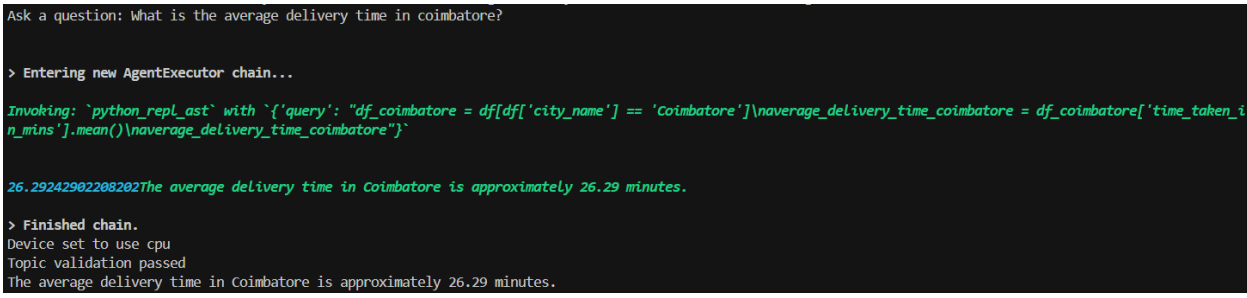
Use Cases

1. Finance Data Analysis

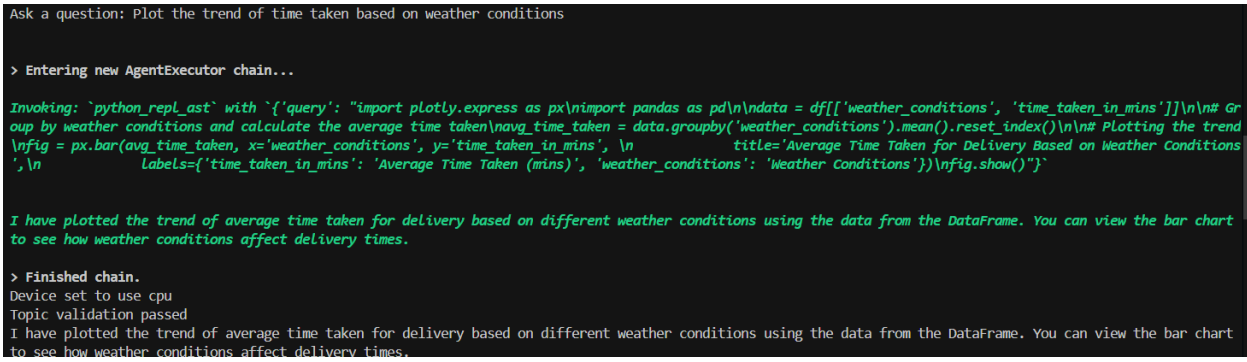
- **Example Queries**:
  - "What is the average transaction value?"
  - "Show the top 5 customers by revenue."
- **Visualization**:
  - "Generate a bar chart of monthly expenses."

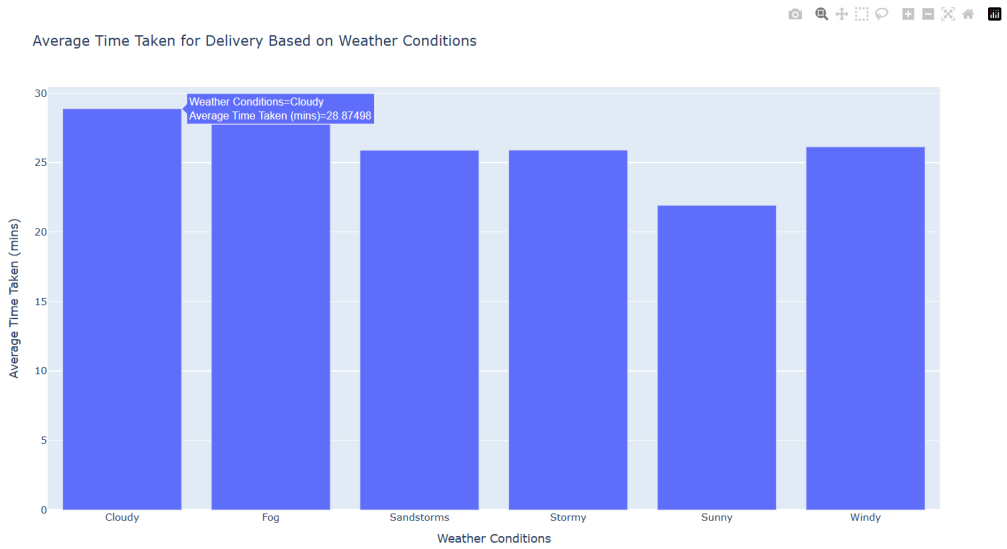
2. Delivery Metrics

- **Example Queries**:
  - "What is the average delivery time in Coimbatore?"



- **Visualization**:
  - "Plot the trend of time taken based on weather conditions."





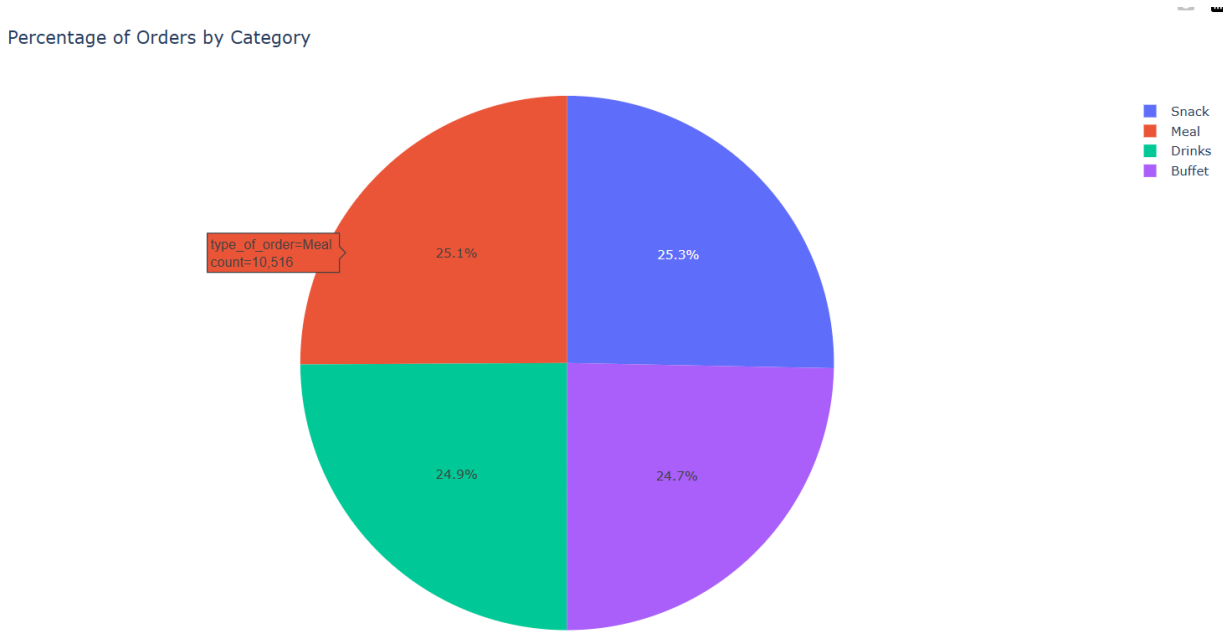
3. Restaurant Insights

- **Example Queries**:
  - "What are the top 3 most popular menu items?"

```
Ask a question: What is the top 2 popular items ordered?


> Entering new AgentExecutor chain...

Invoking: `python_repl_ast` with `{'query': "df['type_of_order'].value_counts().head(2)"}`


type_of_order
Snack     10610
Meal      10516
Name: count, dtype: int64The top 2 popular items ordered are "Snack" with 10,610 orders and "Meal" with 10,516 orders.
```

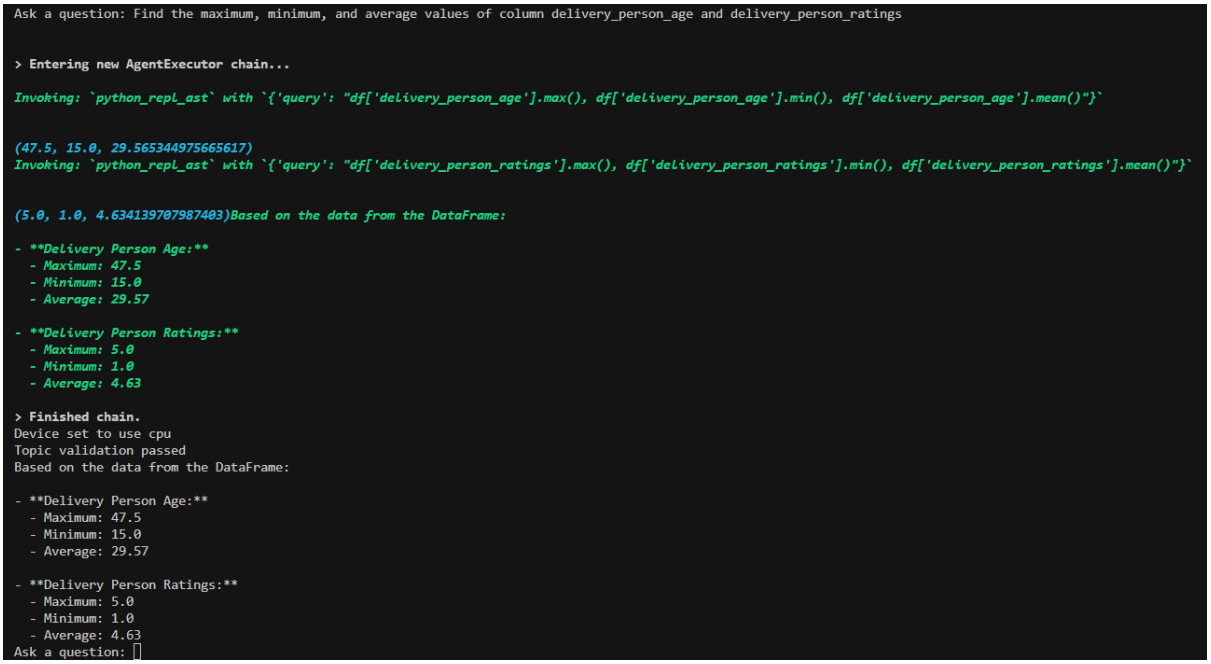
- **Visualization**:
  - "Create a pie chart showing the percentage of orders by category."

```
Ask a question: Create a pie chart showing the percentage of orders by category.


> Entering new AgentExecutor chain...

Invoking: `python_repl_ast` with `{'query': "import plotly.express as px\n\n# Count the number of orders by type_of_order\ntype_of_order_counts = df['type_of_order'].value_counts()\n\n# Create a pie chart\nfig = px.pie(type_of_order_counts, values='type_of_order', names=type_of_order_counts.index, title='Percentage of Orders by Category')\nfig.show()"}`


ValueError: Value of 'values' is not the name of a column in 'data_frame'. Expected one of ['count'] but received: type_of_order
Invoking: `python_repl_ast` with `{'query': "# Correcting the error by converting the series to a DataFrame\nimport plotly.express as px\n\n# Count the number of orders by type_of_order\ntype_of_order_counts = df['type_of_order'].value_counts().reset_index()\ntype_of_order_counts.columns = ['type_of_order', 'count']\n\n# Create a pie chart\nfig = px.pie(type_of_order_counts, values='count', names='type_of_order', title='Percentage of Orders by Category')\nfig.show()"}`


I have created a pie chart showing the percentage of orders by category using the data from the DataFrame. The chart visualizes the distribution of different types of orders.

> Finished chain.
Device set to use cpu
Topic validation passed
I have created a pie chart showing the percentage of orders by category using the data from the DataFrame. The chart visualizes the distribution of different types of orders.
```



4. General Data Insights

- **Example Queries**:
  - "Find the maximum, minimum, and average values of column delivery_person_age and delivery_person_ratings? “



Error Handling

1. **Invalid Input**:
    - User inputs containing inappropriate content are flagged by the Moderation API.
    - The chatbot informs the user about flagged content.
2. **Out-of-Scope Queries**:
    - Guardrails restrict responses to valid topics.
    - The bot responds with: *"Your question appears to be outside the scope of our supported topics."*
3. **Agent Errors**:
    - Logs exceptions and provides user-friendly feedback for unexpected errors.

Flow Diagram

```
User Input
    ↓
[Moderation Check] -----> [Flagged Content?]
       ↓ No                        ↓ Yes
```

```
  [Query Processing]     "Input inappropriate"
        ↓
[Agent Invokes Tools]
        ↓
[Guardrails Validation]
        ↓
[Response/Visualization] ---> [Invalid Topic?]
                                          ↓ Yes
                                  "Outside scope"
```

Enhancements for Future

1. **Dynamic Dataset Support**:
    - Allow users to upload their datasets for on-the-fly analysis.
2. **Advanced Guardrails**:
    - Include more granular validation for specific business use cases.
3. **Improved Visualizations**:
    - Incorporate interactive dashboards with **Plotly Dash** or **Streamlit**.
4. **Context Awareness**:
    - Enable multi-turn conversations with persistent context for in-depth queries.