**NiFi Flow Documentation**

**Overview**

This NiFi flow is designed to automate the process of ingesting data from the Kaggle API, processing the data, and pushing it to a database. It includes logging functionality to capture each step of the execution and error handling to ensure smooth operation.

**Flow Components**

**1. GenerateFlowFile**

* **Purpose**: To create a FlowFile that acts as the trigger for the flow. This is necessary to initiate the data fetching process from the Kaggle API.
* **Configuration Settings**:
  + **Scheduling Strategy**: Timer-driven (configured to run every two hours).
  + **Custom Properties**: None required.

**2. ExecuteStreamCommand (Kaggle API Data Fetch)**

* **Purpose**: To execute a Python script that interacts with the Kaggle API to fetch data.
* **Configuration Settings**:
  + **Command Path**: Path to the Python interpreter.
  + **Command Arguments**: Path to the kaggle\_data\_fetch.py script and any necessary arguments for the API request.
  + **Input FlowFile**: FlowFile from the GenerateFlowFile processor.
  + **Output Destination**: Logs the output of the command execution to a specified log file.

**3. ExecuteStreamCommand (Data Processing)**

* **Purpose**: To run a Python script that processes the fetched data and checks if the target database table exists.
* **Configuration Settings**:
  + **Command Path**: Path to the Python interpreter.
  + **Command Arguments**: Path to the data\_processing.py script, including parameters for the database connection and data processing.
  + **Input FlowFile**: FlowFile from the previous ExecuteStreamCommand processor.
  + **Output Destination**: Logs the output of the command execution to a specified log file.

**4. RouteOnAttribute (Unique Data Check)**

* **Purpose**: To determine if the data already exists in the database, enabling appropriate actions based on the existence of the data.
* **Configuration Settings**:
  + **Routing Criteria**: Based on attributes set by the previous processing script (e.g., checking if records exist).
  + **Relationships**:
    - **Exists**: Route to the processor handling existing data.
    - **Not Exists**: Route to the processor for inserting new data.

**5. LogAttribute (Logging)**

* **Purpose**: To log attributes of the FlowFile for monitoring and debugging purposes.
* **Configuration Settings**:
  + **Log Level**: INFO
  + **Attributes to Log**: Key attributes that provide insight into the processing status.

**6. UpdateAttribute (Setting Output Paths)**

* **Purpose**: To modify the FlowFile attributes to include output paths for reports and logs.
* **Configuration Settings**:
  + **Attribute Names**: output\_path, log\_file\_path, etc.
  + **Attribute Values**: Derived from previous processor outputs or set dynamically.

**7. PutDatabaseRecord (Inserting Data into Database)**

* **Purpose**: To insert processed data into the target database.
* **Configuration Settings**:
  + **Database Connection Pooling Service**: Reference to the connection pool configured for the target database.
  + **Record Reader**: Configuration for reading the FlowFile content.
  + **Table Name**: Name of the database table where the data will be inserted.

**8. LogMessage (Completion Log)**

* **Purpose**: To log a message indicating the successful completion of the data processing and insertion.
* **Configuration Settings**:
  + **Log Level**: INFO
  + **Message**: Custom message summarizing the results of the flow execution.

**Error Handling**

* Implement error handling mechanisms, such as:
  + Sending notifications via email if any of the processors fail.
  + Logging error messages for failed processors using the LogMessage processor.

**Conclusion**

This documentation provides an overview of the NiFi flow components, their purposes, and configuration settings. Maintaining thorough documentation is crucial for future maintenance and onboarding of new team members, ensuring smooth operation and understanding of the flow.

**Python Script Documentation**

**Overview**

This documentation covers the Python scripts used in the NiFi flow for data ingestion from the Kaggle API and processing the retrieved data. The scripts are designed to ensure robust data handling, including fetching, processing, and inserting data into a database.

**1. kaggle\_data\_fetch.py**

**Purpose**

This script is responsible for interacting with the Kaggle API to fetch the required dataset.

**Usage**

To run this script, it should be called by the NiFi ExecuteStreamCommand processor, passing any required arguments for authentication and dataset selection.

**Configuration**

* **Input Arguments**:
  + dataset\_name: The name of the dataset to be fetched from Kaggle.
  + kaggle\_json\_path: The path to the kaggle.json file for API authentication.

**2. data\_processing.py**

**Purpose**

This script processes the data fetched from Kaggle, checking if the relevant database table exists and preparing the data for insertion.

**Usage**

The script is executed after fetching data, called by the NiFi ExecuteStreamCommand processor. It takes necessary parameters for database connection and data processing.

**Configuration**

* **Input Arguments**:
  + db\_connection\_string: Connection string for the target database.
  + table\_name: The name of the database table to check or insert data into.
  + data\_file\_path: The path to the fetched data file (e.g., CSV or JSON). (This could be used during testing phase)

**3. sql\_analysis.py**

**Purpose**

This script executes SQL queries on the database to analyze the ingested data and produce relevant insights or summaries.

**Usage**

This script is executed after data has been processed and inserted into the database, typically called by the NiFi ExecuteStreamCommand processor.

**Configuration**

* **Input Arguments**:
  + db\_connection\_string: Connection string for the target database.
  + query: The SQL query to be executed.

**4. output\_generation.py**

**Purpose**

This script generates outputs based on the results of the SQL analysis, such as reports or summaries, and prepares them for further processing or export.

**Usage**

This script is executed after the SQL analysis is complete. It can be called from NiFi to generate required output files.

**Configuration**

* **Input Arguments**:
  + analysis\_results: Data structure (e.g., DataFrame, list) containing results from SQL queries.
  + output\_file\_path: Path where the output report or summary will be saved.

**5. file\_content\_generation.py**

**Purpose**

The file\_content\_generation.py script generates a structured overview and insights based on credit card transaction data by leveraging the Cohere AI API. The script takes a dataset, formulates a prompt, and requests detailed analysis and recommendations regarding spending habits.

**Usage**

This script is intended to be executed after data processing and analysis have been completed. It can be integrated into a larger data pipeline, typically invoked through the NiFi ExecuteStreamCommand processor or directly within a Python workflow.

**Configuration**

* **Environment Variables**:
  + COHERE\_API\_KEY: The API key required to authenticate requests to the Cohere API. This should be set as an environment variable before running the script.

**Explanation of Key Components**

1. **Environment Variable Check**:
   * The script first checks for the presence of the COHERE\_API\_KEY environment variable. If it's not set, an error is logged, and a ValueError is raised to prevent further execution.
2. **Cohere Client Initialization**:
   * A Cohere client instance is created using the provided API key. This client is responsible for making requests to the Cohere API.
3. **Prompt Construction**:
   * The script constructs a prompt string that includes information about the credit card transactions and asks the AI for specific insights.
   * The prompt specifies the format for the expected output, guiding the AI in generating structured results.
4. **API Call and Response Handling**:
   * The script sends the prompt to the Cohere API using the co.chat() method and awaits a response.
   * If the response is empty, an error is logged, and an empty list is returned.
5. **Response Parsing**:
   * The script extracts the content from the response, specifically targeting the portion that contains the JSON-like dictionary.
   * It uses string manipulation to locate the starting and ending braces {} to slice the response accordingly.
6. **Dictionary Conversion**:
   * The sliced string content is parsed into a Python dictionary using json.loads().
   * This dictionary is then converted into a list of tuples for easier handling and reporting.
7. **Error Handling**:
   * The script includes a broad exception handler that logs any errors encountered during execution, ensuring that issues are tracked without crashing the application.

**Logging**

* Logging is implemented to capture significant events, such as missing API keys or errors during API calls. This helps maintain visibility over the script's execution flow and aids in debugging.

**Conclusion**

The file\_content\_generation.py script efficiently generates actionable insights from credit card transaction data by leveraging advanced AI capabilities. Its structured output is essential for further analysis or reporting, making it a valuable component in the data processing workflow.