Overview of the Process

This script is designed to process soccer event data, extract player offensive metrics, and store the results in a PostgreSQL database. The data engineering process ensures that data flows smoothly from raw sources to structured tables while maintaining data integrity and handling errors effectively.

I opted for a full refresh approach instead of an incremental update because the data needs to be computed by combining all the files. This ensures that the derived metrics are accurate and consistent, reflecting the complete dataset rather than just recent changes.

The pipeline can be broken down into the following key stages:

1. Data Extraction
2. Data Transformation
3. Data Ingestion to Staging
4. Data Insertion into Production Table
5. Finalization

1.Data Extraction

The data is stored as JSON files in a GitHub repository. The extraction step fetches these files so they can be processed and transformed into a usable format.

GitHub API is used list all JSON files in the /events directory. Once the list of files is obtained, each file is read using its URL.

2. Data Transformation

The JSON files are parsed to extract specific event data related to offensive metrics (e.g., shots, goals, assists). For each player, various offensive metrics are calculated and stored in a dictionary.

3. **Data Ingestion to Staging**

Data Loading: The transformed data is written to a CSV file for data persistency, which is then loaded into the staging table using the COPY command for efficiency

Table Creation: Before inserting data into the staging table, any existing staging table is dropped to ensure that only the most recent data is processed. The table is then recreated with the appropriate schema. If the COPY operation fails, the transaction is rolled back to ensure no partial or corrupted data is stored

The SQL for creating the table is dynamically generated by iterating over the columns of the DataFrame. This ensures that the order of columns in the table matches the order in the CSV file, allowing the COPY command to work correctly without needing to specify the column order explicitly.

The staging layer serves as a temporary holding area for raw data before it undergoes transformations and is loaded into the production environment. This separation prevents errors or inconsistencies in the raw data from affecting the final dataset, allowing for early detection and correction.

For example, if a column's data type is incorrectly specified and data is directly inserted into the production table, it could result in errors or data loss. By using a staging layer, such issues can be identified and resolved before the data reaches the production environment, minimizing downtime and potential data corruption.

Furthermore, using CSV files can help mitigate the impact of database downtime or other disruptions. If data cannot be written to the database, it can be temporarily stored in CSV format, reducing the time needed to re-run the data loading process once the issue is resolved.

4. Data Insertion into Production Table

Once data has been staged, it is ready to be moved into the production table where it will be available for queries and analysis. The production table is only dropped and recreated if the data copying to the staging table is successful. This step ensures that the production table only contains complete and verified data.

Table Creation: The production table is dropped and recreated to maintain data integrity and prevent conflicts with outdated information. By using a fixed schema defined in the CREATE SQL statement, the risk of errors, such as unintended adding, removing, or modifying columns or data types, being propagated to the production environment is minimized. This approach helps ensure data consistency and reliability.

Data Insertion: The data from the staging table is inserted into the production table. The SQL query includes type casting to ensure that data types in the database align with the intended schema.

5. Finalization

The database connection is closed in a finally block to ensure resources are freed regardless of whether the process succeeds or fails.

Key Considerations

- **Table Management**: The script carefully manages table creation and deletion. Staging tables are dropped and recreated to ensure clean data loading. Production tables are only altered after successful staging table operations, preventing the accidental loss of validated data.
- **Data Integrity**: By separating staging and production tables, the script ensures that only validated, complete data is loaded into the production environment. This step mitigates risks associated with partial or corrupted data entries.