## ****Challenge Overview****

Existing Reconciliation tools processes huge number of transactions daily and monthly. Business users spend significant time manually identifying data anomalies post-reconciliation for the breaks and fixing them, making the process tedious and error prone.

## ****Problem Statement****

Implement an anomaly detection system within the reconciliation process, by leveraging historical data to identify patterns, trends, and expected ranges. The system should be able to:

* Automatically detect data anomalies by comparing real-time data against historical baselines.
* Provides insights into potential root causes of detected anomalies.
* Integrates with existing reconciliation tools to streamline the anomaly identification process.
* Reduces manual effort and minimizes human error in anomaly detection.

Business users like Reconcilers, Operations user will interact with their existing reconciliation tools to see identified anomalies and additional insights generated from AI.

## ****Expectations from Participants****

Participants will:

* Use **LLMs** (such as OpenAI’s GPT, LLaMA, or Falcon) with historical data to detect and categorize anomalies in real-time reconciliation data.
* Build a **classification system** outlining the reason for an anomaly in predefined buckets (and not as free text reasoning). If a new reason is detected, that should be marked as new with the details.
* Implementtechniques (e.g., clustering, anomaly detection) to identify the patterns in the data and validate data consistency/accuracy.
* Develop an **interactive tool** to allow the reconcilers to give **feedback** on the detected anomalies (like false positives/ false negatives) that can be used in future to refine the accuracy.
* Use **Agentic AI** to look at reconciliation breaks (including anomalies) and provide concise summaries of break resolutions.
* These agents should be able to learn from and mimic Operator/Reconcilers with a human in the loop to take corrective action for a break.
* Streamline the workflow with **Operator Assist Agents** - Create Resolution Tasks, Call APIs, Send Emails, Create Tickets etc.
* Develop an end-to-end pipeline that accepts reconciliation details, current and historical data as input, outputs:
  + Anomaly detection results
  + Classification of detected anomaly
  + Break resolution summaries
  + Tries to autonomously act on the break resolutions.
* Ensure the solution is scalable, explainable, and efficient.

## ****Technical Constraints****

* The solution **must be built using freely available tools** (e.g., OpenAI’s free-tier API, Hugging Face models, Scikit-learn, Pandas, LangChain, etc.).
* It should be implemented using **Python**.
* The dataset used for validation will be provided as **CSV or Excel files**.

## ****Sample Test Dataset****

A reconciliation is the process of matching two data sources like payment, settlement data etc. The outcome of a reconciliation could be a **match** or **break**. A break can have further sub classification. A reconciler needs to manually analyze each break and take corrective actions to either fix the break or document its validity.

Some definitions before we jump to the data:

**Key Columns**: These are the columns based on which **LHS (Left hand Side)** or **Source 1** data is matched with **RHS (Right Hand Side)** or **Source 2** data.

**Criteria Columns**: These are the column which are matched between LHS and RHS to mark the reconciliations as a **Match** or **Break**. These can be exact match or matching with some tolerance/buffer and is typically done my reconciliation rules engine.

**Derived Columns**: New columns created during reconciliation process to facilitate break analysis.

**Historical Columns**: Suggested column which can be used in conjunction with date columns to establish historical trends and patterns in the data.

**Date Columns**: Date columns which can be used as a marker to study point in time data.

**Comment Columns**: Documentation of corrective action taken or analysis. This could be empty in some cases.

**Case 1**: General Ledger (GL) vs IHub Reconciliation

**Key Columns**: Company, Account, AU, Currency

**Criteria Columns**: GL Balance, IHub Balance

**Derived Columns**: Balance Difference

**Historical Columns**: Account, Secondary Account, Primary Account

**Date Columns**: As of Date

A sample data for IHub reconciliation is added in the attachment.

**Case 2**: Catalyst vs Impact Reconciliation

**Key Columns**: Trade ID

**Criteria Columns**: Inventory Code, CUSIP, Trade Date, Settlement Date, Buy or Sell, Price, Quantity

**Historical Columns**: CUSIP, Inventory Code

**Date Columns**: Recon Date

A sample data for Catalyst reconciliations is added in the attachment.



**Reconciler actions** could include cross verifying the data in source systems (LHS or RHS or both), correcting source system data and reposting to reconciliation systems, create JIRA tickets or email the concerned teams.

Please assume any missing data/columns that will be critical for your AI solution to work and justify the assumptions. For example, you may assume at least **one year** of historical data will be available.

## ****Submission Requirements****

Participants must submit:

* A **GitHub repository** with the code, including a README file with setup instructions.
* A **presentation (PDF/Slides)** explaining the approach, challenges, and results.
* A **demo video (optional but recommended)** showcasing the solution in action.

Git template folder structure for Hackathon

* Repository information: <Team Repo>
* Code templates and Folder structure

/<Team Repo>

|-- artifacts/

|-- arch/ #Architecture documents

|-- demo/ #presentation or Demo to support

|-- code

|-- src/ #source code

|-- test/ # test cases

## ****Tools & Resources****

Participants can use any freely available tools, including but not limited to:

* **LLMs**: OpenAI GPT (free-tier API), Hugging Face Transformers
* **Machine Learning**: Scikit-learn, PyCaret, AutoML frameworks
* **Data Processing**: Pandas, NumPy
* **Code Generation**: LangChain, GPT-based code generation
* **Visualization & Interactivity**: Streamlit, Gradio for interactive UI