**Project Title: Advanced Pipeline Orchestration for Multi-Source Data Integration and Export**

**Objective**:  
Following the initial project of ingesting and transforming PayPal data, this project aimed to enhance the ETL process by orchestrating a complex pipeline. The focus was to handle multiple data sources, implement error handling, and automate the export of transformed data as a CSV file.

**Project Steps:**

**1. Pipeline Orchestration and Design**

* **Purpose**: Build a cohesive data pipeline in Azure Data Factory that manages multiple sources, transformation processes, and data export.
* **Approach**:
  + Created a pipeline that sequentially controls the flow from data ingestion to final output, ensuring each step completes successfully before triggering the next.
  + Designed the pipeline with modular steps, making it easy to manage individual processes and to troubleshoot specific sections if needed.

**2. Multi-Source Data Ingestion**

* **Objective**: Integrate data from multiple sources and trigger each based on the successful ingestion of the prior source.
* **Process**:
  + **Source Copying**: Set up two initial data copy activities to bring in data from separate sources.
  + **Sequential Execution**: Configured each copy activity to execute only after the prior step completes successfully.
* **Outcome**: The pipeline now ingests each data source in sequence, ensuring that only validated data proceeds to the transformation stage.

**3. Error Handling with Failure Notifications**

* **Purpose**: Enhance reliability by adding failure handling and notifications to specific pipeline steps.
* **Implementation**:
  + Added a failure path connected to both data copy steps. If an ingestion step fails, the failure path triggers, sending a notification with an error message and code.
  + This allows for immediate identification of any issues and provides quick insights for troubleshooting.
* **Outcome**: The error handling process ensures that any ingestion issues are flagged instantly, reducing downtime and improving pipeline reliability.

**4. Pipeline Execution for Data Transformation**

* **Objective**: Continue with data transformation processes only when prior steps are completed successfully.
* **Process**:
  + Created an **Execute Pipeline** step in ADF to carry out data transformations on the ingested data.
  + The transformation step involves cleansing, merging, and filtering, as in the previous project, to prepare data for storage in an Azure SQL Database.
* **Result**: Successfully transformed data is stored in the SQL database, ready for final processing and export.

**5. Final Data Export as CSV**

* **Purpose**: Provide a final, accessible version of the transformed data in a widely usable format.
* **Steps**:
  + Designed a new data flow in ADF to extract data from the Azure SQL Database.
  + Used a **Sink** to export the final transformed dataset as a CSV file, which is stored in Azure Blob Storage.
* **Outcome**: The transformed data is now easily accessible in CSV format, facilitating further analysis or sharing as needed.

**Key Features and Learnings:**

* **Advanced Orchestration**: Designed a complex pipeline to handle multi-source ingestion and step-by-step processing.
* **Failure Management**: Implemented robust error handling, allowing for immediate response to any pipeline issues.
* **Automation**: Built a pipeline with automated triggers and failure notifications to ensure reliability and efficiency.
* **Data Export**: Enabled seamless data export into Azure Blob Storage as a CSV, ready for easy access and use.

**Reflections**

* This project built on the foundation of the first Azure project, reinforcing my skills in managing end-to-end ETL workflows.
* Gained deeper knowledge of handling complex pipeline architectures in Azure Data Factory, including error handling and automation.

**Screenshots**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**