Project Report: Automated Financial Data Pipeline

Introduction

This report outlines a comprehensive project that integrates various Azure cloud services and data processing tools to establish an automated pipeline for the extraction, loading, transforming (ETL), and visualization of financial data. The project's objective is to streamline the monitoring and analysis of stock market trends and company financials over a ten-year period.

Data and Methods

The project leverages several technologies and platforms to construct the ELT pipeline:

Azure Functions were implemented to automate the retrieval of historical stock data, sourced from the yfinance library, for indexes and companies over the previous decade. This Python-based function is scheduled to run daily and stores the extracted data in CSV format on Azure Blob Storage within a container named 'cont-output'.

Alteryx is utilized for data cleansing and transformation. Financial data files for four major companies such as Apple, Microsoft, Amazon, and META were processed. These files, represent annual and quarterly balance sheets and income statements and were sourced from Yahoo Finance Plus. The Alteryx workflows, specifically tailored for each financial statement type, were designed to ensure accurate data structuring.

Logic Apps orchestrates the workflow, scheduling the Azure Functions and managing data updates. With its configuration, it monitors the modifications in the dataset, triggering notifications upon successful updates.

The data are persisted in Blob Storage, which acts as a data lake, holding the CSV files. Daily data is managed by Azure Functions, and Alteryx manages quarterly and annual updates.

Azure Synapse Analytics plays a pivotal role in loading the CSV files from blob storage into SQL tables. Through a dedicated SQL pool, the data undergoes further transformations, ensuring it is primed for analysis.

Azure Function

The Azure Function, is executed upon receiving an HTTP request. The script performs several operations essential for the data pipeline:

- The script begins by logging the initiation of the process and initializing the DefaultAzureCredential, which is a secure method for accessing Azure services without hard-coding credentials in the script.
- Utilizing the BlobServiceClient with the specified account URL, the script connects to the Azure Blob Storage where the data files will be stored.
- The script checks for existing files that start with 'function_' prefix and deletes them to make room for updated data, ensuring the container only contains the latest data.
- After downloading the data using yfinance library, the script then converts the data into CSV format in preparation for upload.
- Finally, if the upload is successful, the function returns an HTTP response indicating success. If any exceptions are caught during the process, an error message is logged, and an HTTP response indicating failure is returned.

Alteryx

Alteryx workflows enable the processing of complex data manipulation tasks through a visual programming interface. For this project, a total of sixteen files containing quarterly and annual financial data for Apple, Microsoft, Amazon, and META were downloaded from Yahoo Finance Plus.

Income Statement Workflow Analysis
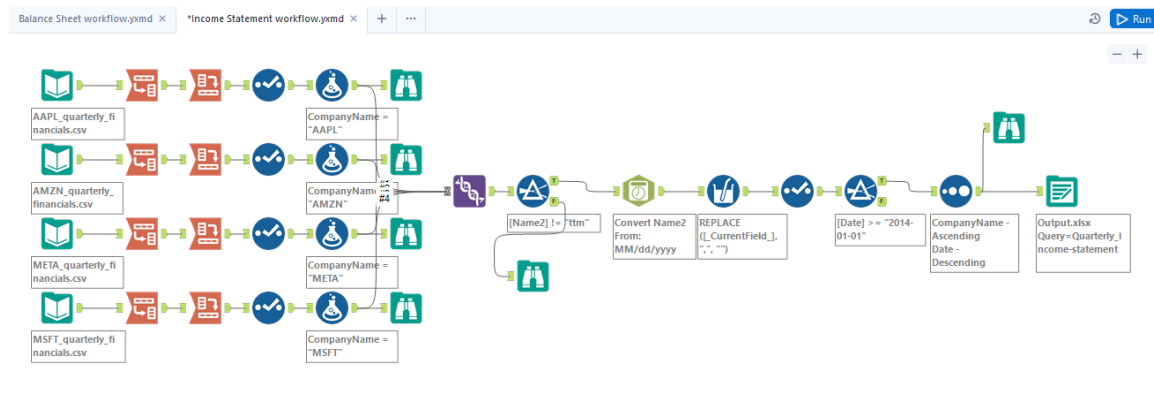
The first image represents the workflow for transforming the income statements. This workflow begins with the input data nodes for each company, where CSV files are read into the workflow. The data is then cleansed and structured using a series of tools: Select, Formula, Filter, and Sort.

Specifically, the workflow assigns company names to individual records, reformats dates, and replaces missing values. The process involves filtering the records to include only relevant data from 2014 onwards, ensuring a standardized dataset for analysis.

Annual Balance Sheet Workflow Analysis
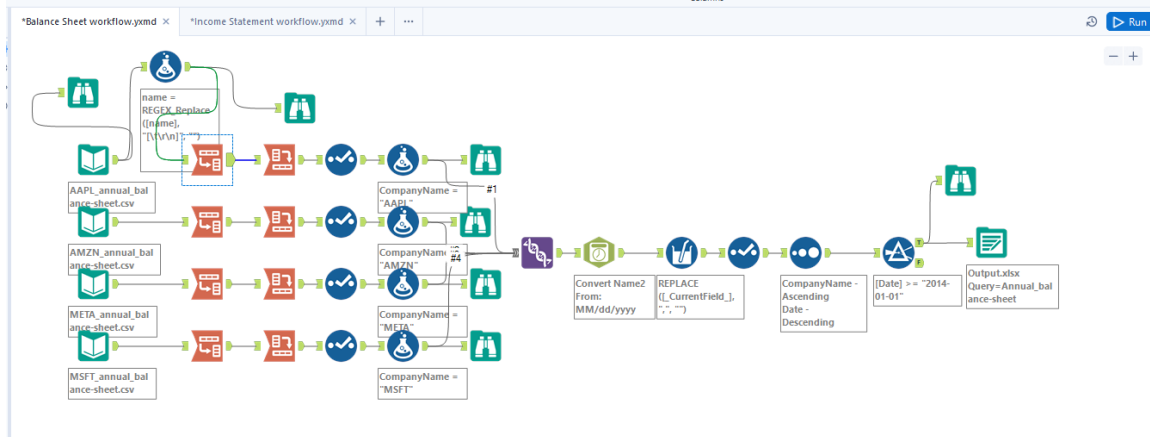
The second image details the workflow for processing the annual balance sheets. Similarly, it utilizes input data nodes for the annual CSV files and applies a consistent sequence of tools to prepare the data for further analysis.

The workflow incorporates REGEX tools for name standardization, a necessary step due to the variation in naming conventions across different data files. After the initial transformation steps, the data is sorted and outputted for subsequent use.

Workflow Adaptability and Final Output

Both workflows were designed with adaptability in mind, allowing for minimal changes when switching from quarterly to annual data processing. The final output of each workflow is a clean, well-structured dataset that is directly uploaded to Azure Blob Storage for persistent storage.

These workflows provide a reproducible and automated approach to handle financial data, significantly reducing manual effort and the risk of errors associated with data handling. The output files are ready for integration into the larger data pipeline, where they will contribute to insightful financial analysis and decision-making.
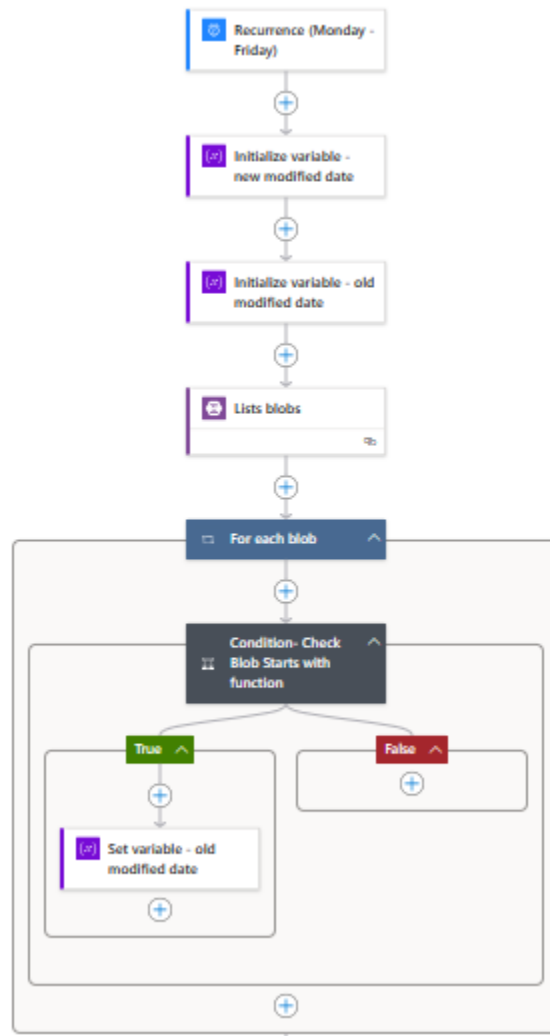
The Azure Function script is a crucial component of the financial data pipeline. It is efficiently designed to handle data extraction, transformation, and loading with minimal intervention, aligning with modern cloud-based ETL practices. This approach not only enhances data management and accuracy but also optimizes resource usage by avoiding redundant storage and ensuring up-to-date information is readily available for further analysis.
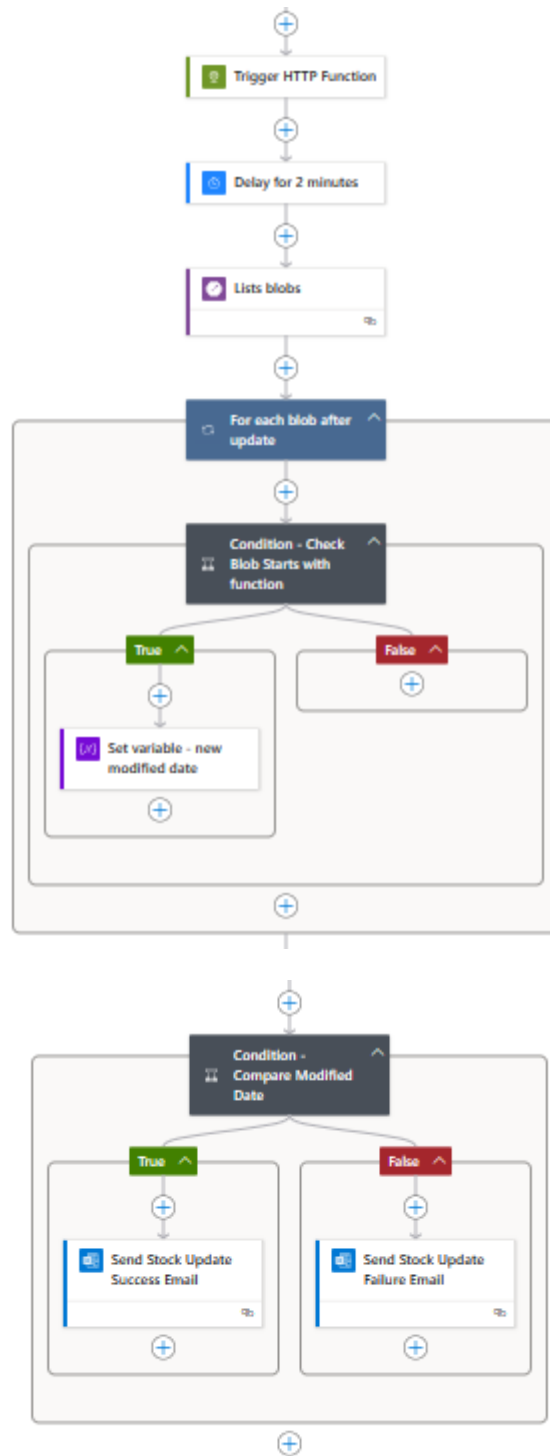
This report provides an overview and a detailed explanation of the Azure Function script used in the project. It offers insights into the technical aspects and operational flow of the script, reflecting its importance and efficacy within the larger context of the data pipeline.

Logic app

The project employs Azure Logic Apps to automate the process of triggering Azure Functions responsible for data updates, thereby ensuring data remains up-to-date.

The Logic App serves as an automated scheduler and monitor for data updates. Its design ensures that the financial data, vital for decision-making and analysis, is regularly refreshed and that any updates are communicated promptly.

```
                    ⊕
          ┌──────────────────────┐
          │ 🟩 Trigger HTTP Function │
          └──────────────────────┘
                    ⊕
          ┌──────────────────────┐
          │ ⏱ Delay for 2 minutes  │
          └──────────────────────┘
                    ⊕
          ┌──────────────────────┐
          │ 🧭 Lists blobs          │
          │                     ⊡ │
          └──────────────────────┘
                    ⊕
    ┌─────────────────────────────────────────┐
    │  ┌──────────────────────┐               │
    │  │ ↻ For each blob after  ⌄ │           │
    │  │    update              │             │
    │  └──────────────────────┘               │
    │             ⊕                            │
    │  ┌──────────────────────┐               │
    │  │ ⬚ Condition - Check  ⌄ │             │
    │  │    Blob Starts with    │             │
    │  │    function            │             │
    │  └──────────────────────┘               │
    │       ┌─────────┴──────────┐            │
    │  ┌─────────┐          ┌─────────┐        │
    │  │ True  ⌄ │          │ False ⌄ │        │
    │  └─────────┘          └─────────┘        │
    │  ┌────────────┐       ┌────────────┐     │
    │  │     ⊕      │       │     ⊕      │     │
    │  │ ┌────────┐ │       └────────────┘     │
    │  │ │[x] Set variable - new │            │
    │  │ │   modified date       │            │
    │  │ └────────┘ │                          │
    │  │     ⊕      │                          │
    │  └────────────┘                          │
    │             ⊕                            │
    └─────────────────────────────────────────┘

                    ⊕
    ┌─────────────────────────────────────────┐
    │  ┌──────────────────────┐               │
    │  │ ⬚ Condition -       ⌄ │             │
    │  │    Compare Modified    │             │
    │  │    Date                │             │
    │  └──────────────────────┘               │
    │       ┌─────────┴──────────┐            │
    │  ┌─────────┐          ┌─────────┐        │
    │  │ True  ⌄ │          │ False ⌄ │        │
    │  └─────────┘          └─────────┘        │
    │  ┌────────────┐       ┌────────────┐     │
    │  │     ⊕      │       │     ⊕      │     │
    │  │ ┌────────┐ │       │ ┌────────┐ │     │
    │  │ │ Send Stock Update │ │ Send Stock Update │
    │  │ │ Success Email     │ │ Failure Email     │
    │  │ │             ⊡ │   │ │             ⊡ │   │
    │  │ └────────┘ │       │ └────────┘ │     │
    │  │     ⊕      │       │     ⊕      │     │
    │  └────────────┘       └────────────┘     │
    └─────────────────────────────────────────┘
                    ⊕
```

1. Recurrence (Monday - Friday): The Logic App is scheduled to initiate the workflow every weekday, consistent with the trading days of the financial markets, ensuring data is refreshed on a daily basis.
2. Initialize variable - new modified date: This action sets up a variable to capture the date and time when the data was last modified, post-update, to serve as a point of comparison.
3. Initialize variable - old modified date: Parallel to the new modified date, this step records the pre-update last modified date, facilitating a before-and-after comparison.
4. List blobs: A retrieval action that compiles a list of all the blobs, which are essentially the data files stored within Azure Blob Storage.
5. For each blob: The workflow enters a loop, iterating through each blob to determine if an update is necessary.
6. Condition - Check Blob Starts with function: A check is performed to identify which blobs are related to the Azure Function and hence require an update.
7. Trigger HTTP Function: This crucial action sends out an HTTP request to the Azure Function, prompting it to commence the data update process.
8. Delay for 2 minutes: A pause is introduced, allowing the data update process, which is asynchronous, to complete fully before moving forward.
9. List blobs: The blob list is refreshed post-update to ensure the most current status is checked.
10. For each blob after update: The Logic App iterates through the blob list once more, this time to verify the completion of the update process.
11. Condition - Check Blob Starts with function: The condition is reiterated to ensure the process is examining the correct blobs post-update.
12. Condition - Compare Modified Date: The decisive moment where the pre and post-update modified dates are compared. If the new date is more recent, the update is considered successful.
13. Send Stock Update Success Email: On successful update, an automated email is dispatched, informing relevant stakeholders that the latest data is available.
14. Send Stock Update Failure Email: Should the update fail, a notification is sent out, alerting to the need for manual intervention or review.

By automating updates and notifications, the Logic App streamlines workflows, mitigates the risk of outdated information, and keeps essential stakeholders informed.

Blob Storage

The project's data storage strategy employs Azure Blob Storage within the 'storestockanalysis' account. Within this storage account, a container named 'cont-output' has been designated to store the financial data files.

Name	Modified	Access tier	Archive status	Blob type
☐ 📄 Annual_balance-sheet.csv	3/15/2024, 9:59:21 PM	Hot (Inferred)		Block blob
☐ 📄 Annual_income-statement.csv	3/15/2024, 9:59:21 PM	Hot (Inferred)		Block blob
☐ 📄 function_company_data.csv	4/2/2024, 5:24:58 PM	Hot (Inferred)		Block blob
☐ 📄 function_index_data.csv	4/2/2024, 5:24:58 PM	Hot (Inferred)		Block blob
☐ 📄 function_volume_data.csv	4/2/2024, 5:24:58 PM	Hot (Inferred)		Block blob
☐ 📄 Quarterly_income-statement.csv	3/15/2024, 9:59:21 PM	Hot (Inferred)		Block blob
☐ 📄 Quaterly_balance-sheet.csv	3/15/2024, 9:59:21 PM	Hot (Inferred)		Block blob

Data files within the 'cont-output' container are managed according to their update frequency and source. Files with display names beginning with 'function_' are recognized as files that need daily updates. The Azure Function is tasked with updating these files every day, ensuring that the data they contain is always current. Other data files, which do not require daily updates, are uploaded manually after being transformed through Alteryx processes. These files are typically generated on a less frequent basis, such as quarterly or annually, and thus do not need daily automation.

Post storage, the financial data files from the Blob Storage are ingested into Azure Synapse Analytics. This process is essential for advanced data analytics, leveraging Synapse Analytics' capacity for extensive data querying and analysis.

Azure Synapse Analytics

SQL Pool and Data Loading in Synapse Analytics

For this project, a SQL pool within Azure Synapse Analytics has been allocated to manage and structure the storage of various data tables. Synapse pipelines have been employed to facilitate the transfer of CSV files from Azure Blob Storage into these tables, which allows for comprehensive manipulation of data including merging tables and executing SQL/T-SQL queries.

Data Copy Pipeline

Below is the synapse pipeline which is used to load data from blob storage to azure synapse analytics tables:
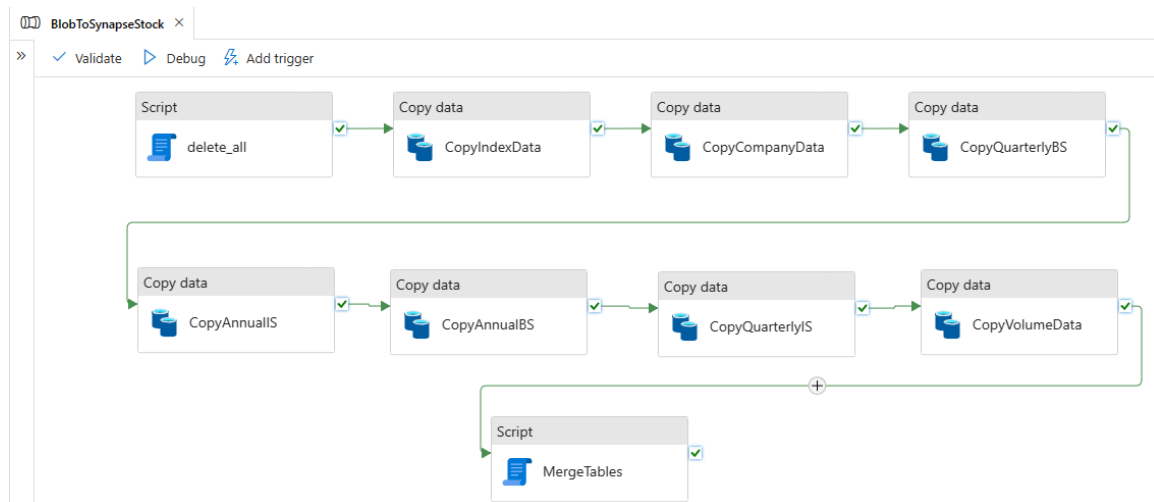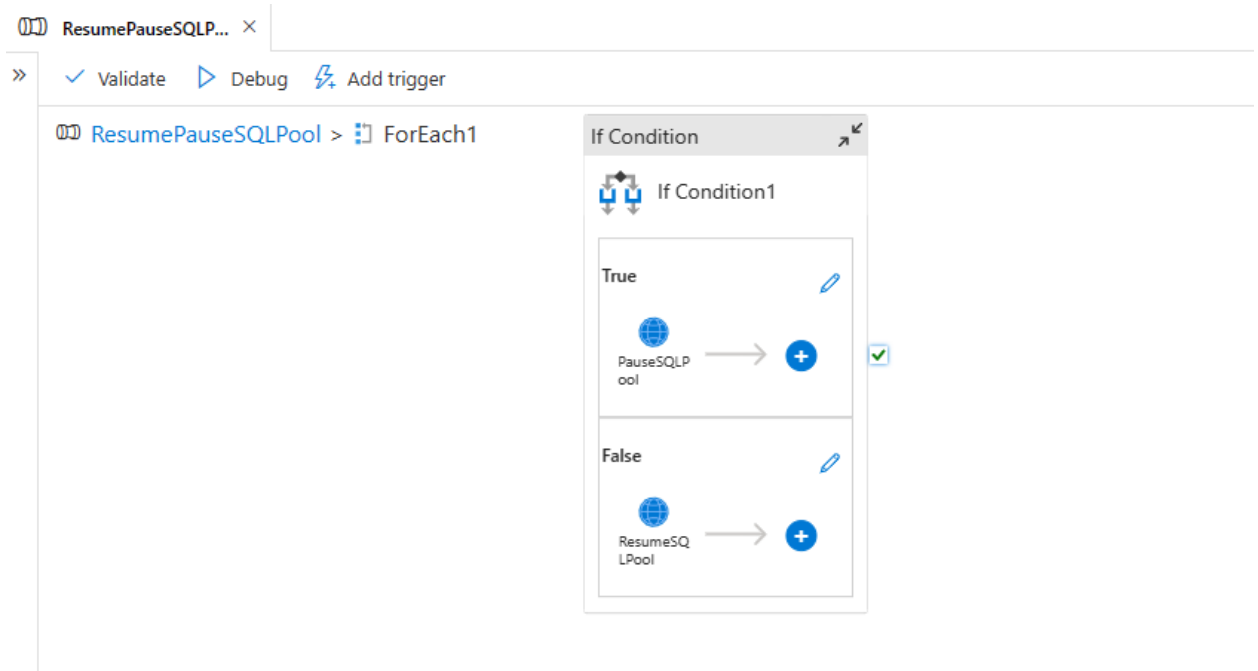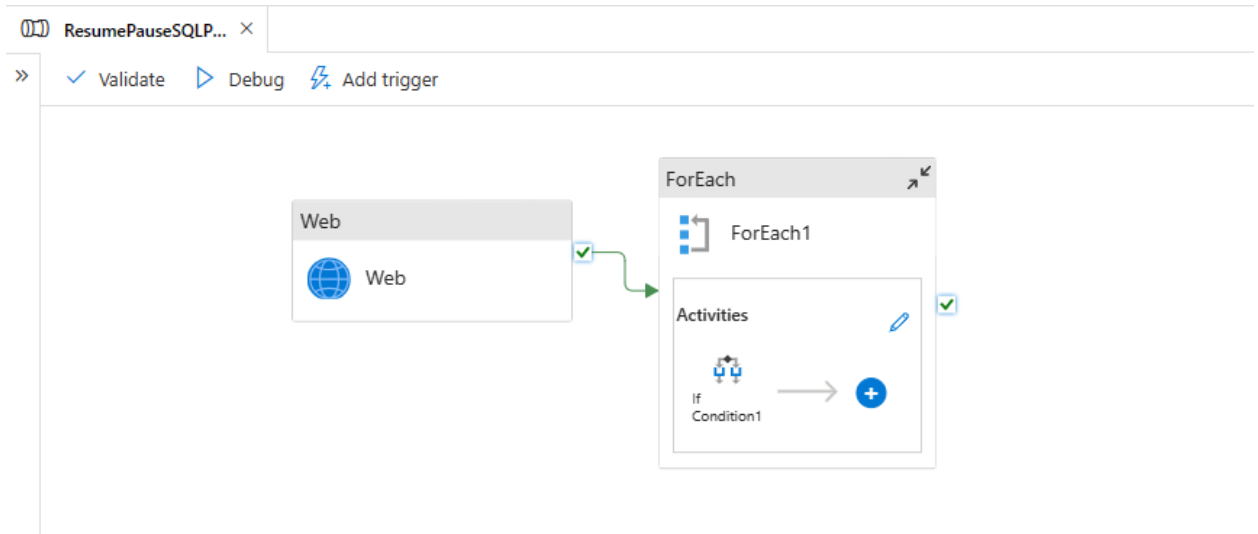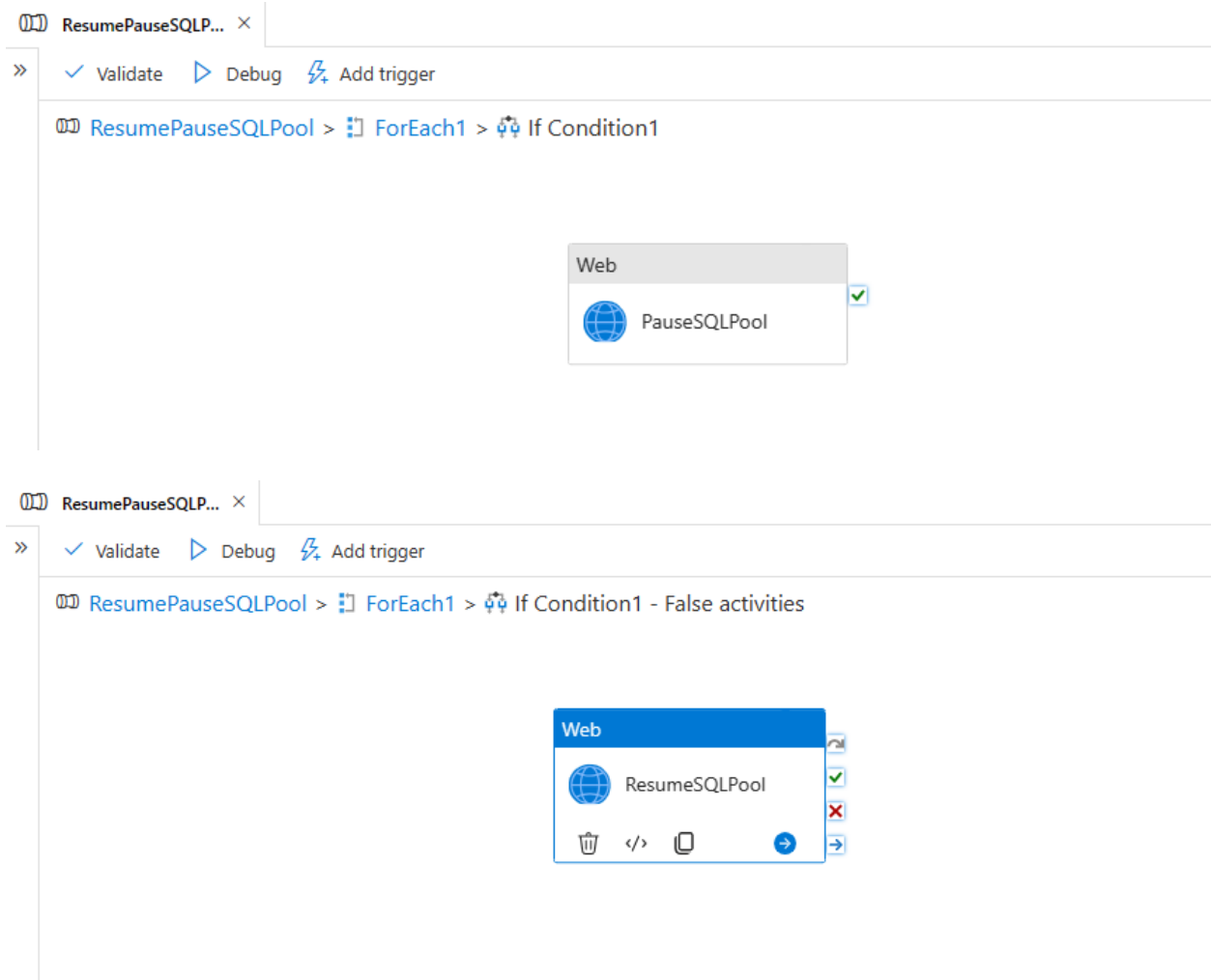

- Initial Table Clearing: The first action in the pipeline employs an SQL script that executes a stored procedure, DeleteTableData, deleting existing data within the tables to prepare for a fresh data load. This ensures that the tables only contain the most recent and relevant data.
- Data Copy Actions: The subsequent actions, from the second to the seventh, perform the data copying operation. This batch of actions systematically moves the seven different files from blob storage into the tables within the SQL pool of Synapse Analytics.
- Table Merging: The final action within the data loading pipeline involves an SQL script designed to merge related tables. Specifically, it merges the annual balance sheet with the annual income statement and does the same for the quarterly income statement and quarterly balance sheet, creating a consolidated view of financials.

SQL Pool Management Pipeline

In addition to the primary data loading pipeline, a separate Synapse pipeline has been established to control the SQL pool, aimed at optimizing cost efficiency. This pipeline's responsibility is to automate the starting and stopping of the SQL pool based on usage, effectively minimizing compute charges when the system is idle.

ResumePauseSQLP... ✕

✓ Validate ▷ Debug ⚡ Add trigger

Web

🌐 Web

ForEach

ForEach1

Activities ✏️

If
Condition1 → ➕

ResumePauseSQLP... ✕

✓ Validate ▷ Debug ⚡ Add trigger

ResumePauseSQLPool > ForEach1

If Condition

If Condition1

True ✏️

🌐
PauseSQLP
ool → ➕

False ✏️

🌐
ResumeSQ
LPool → ➕

- A web activity to get the sql pool list in the synapse workspace.
- A "For Each" loop, which iterates over the sql pools in the synapse workspace.
- An "If Condition" activity, \performs a logical check, which decide whether to pause or resume the SQL pool based on the current status of the pool. If the pool is running, it will pause it and if the pool is already paused it will restart it.

This conditional mechanism ensures that the SQL pool is only active when necessary, contributing to an efficient use of resources and cost savings.

The utilization of Azure Synapse Analytics and its SQL pool demonstrates a strategic approach to managing and processing large volumes of financial data. The implementation of these pipelines not only streamlines the workflow but also introduces a cost-effective solution for resource management within the analytics environment.

Analysis of the Results

Data pulled into Power BI from Azure Synapse Analytics undergoes a final transformation, with relationships defined between tables to aid in the analytical process. The suite of visualizations produced provides insights into stock market behaviors and company financial health.

Conclusion and Recommendations

The project successfully creates an automated data pipeline that enables the processing and visualization of financial data efficiently. The integration of cloud-based ELT with Power BI visualization exemplifies a model for making data-driven decisions in financial analysis.

For continuous improvement, the pipeline's responsiveness to data source changes and enhanced error handling should be considered. Further research could involve expanding the data sources and incorporating predictive analytics and machine learning for trend forecasting.