# ETL TECHNICAL REPORT

**Submitted By:**

**Muhammad Waqas Ali**

**DS-019/2023-24**

# 1. Introduction

This project implements an ETL (Extract, Transform, Load) pipeline designed to automate the process of fetching, cleaning, transforming, and loading stock market data from multiple data sources into a MongoDB database. The primary goal of this pipeline is to streamline data processing tasks, ensure data integrity, and provide timely insights from various stock-related datasets, which include APIs, Google Sheets, CSV files, and BigQuery.

The pipeline extracts raw data from these diverse sources, performs necessary transformations—such as cleaning missing values, converting timestamps, and handling different formats—and loads the processed data into a MongoDB database for easy querying and analysis.

A key feature of this pipeline is automation: it is scheduled to run at regular intervals, ensuring that the data is updated automatically without manual intervention. Furthermore, the integration of Continuous Integration and Continuous Deployment (CI/CD) practices ensures that any changes to the codebase are tested, validated, and deployed in an efficient and reliable manner. This approach minimizes human errors, accelerates the development and deployment cycle, and ensures that the pipeline remains robust and scalable as the data grows and evolves.

## 2. ETL Pipeline Design

### 2.1. Data Sources

Explain the sources from which the data is extracted:

- **API (Alpha Vantage)**: For real-time stock price data (e.g., IBM).
- **Google Sheets**: For stock data collected and stored in Google Sheets (e.g., Apple stock data).
- **CSV (Yahoo)**: For historical stock data.
- **JSON**: For daily stock data (e.g., Alibaba stock data).
- **BigQuery**: For historical stock data (VOD stock data from BigQuery).

### 2.2. Data Cleaning and Transformation

Describe the steps taken to clean and transform the data before loading:

- **Cleaning**: Handling missing values, correcting data types, removing duplicates.
- **Transformation**: Converting timestamps to consistent formats, converting values (e.g., currency or volume), handling categorical values.
- **Combining**: Merging all the data sources into one unified dataset.

Provide visual diagrams (optional) to illustrate the flow from raw data to cleaned data.

## 3. Automation and Scheduling

### 3.1. Scheduler Setup

Following are the steps of scheduler setup:

- **Scheduler Tool**: Python `schedule` module (or cron jobs, if applicable) is used to trigger the ETL process at specific times (e.g., once a day).
- **Logging**: Logs are maintained for each run to monitor successful execution and troubleshoot any errors.

### 3.2. Scheduler Code Example

The scheduler.ipynb script automates the execution of the ETL and database loading processes using the schedule library in Python. It defines two functions: run_etl_pipeline to execute the ETL process by running the etl_pipeline.ipynb notebook, and run_load_to_db to load the transformed data into MongoDB by executing the load_to_db.ipynb notebook. These processes are scheduled to run daily at specific times—ETL at midnight and database loading at 1:00 AM. The script ensures that both processes run automatically without manual intervention, with detailed logging to track the success or failure of each task. It uses a continuous loop to check and execute pending tasks, providing a fully automated solution for maintaining up-to-date data pipelines.

## 4. CI/CD Pipeline (GitHub Actions)

### 4.1. Introduction to CI/CD

CI/CD ensures that the ETL pipeline is reliably tested and deployed. Every time code is updated, it goes through automated tests and is deployed to production without manual intervention.

### 4.2. CI/CD Tools Used

- **GitHub Actions**: Automated workflows for testing, validation, and deployment.
- **Testing**: Unit tests are run to ensure the ETL pipeline behaves as expected.
- **Validation**: Ensures the schema of the data and other rules (e.g., types, ranges) are met.
- **Deployment**: Upon successful testing, changes are deployed to production.

### 4.3. GitHub Actions Workflow File (ci_cd.yml)

Explain the workflow file (`ci_cd.yml`) used to automate the CI/CD process:

```yaml
name: CI/CD Pipeline for ETL

on:
  push:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.x'
      - name: Install dependencies
        run: |
          pip install -r requirements.txt
      - name: Run tests
        run: |
          python -m unittest discover tests/

  deploy:
    runs-on: ubuntu-latest
    needs: test
    steps:
      - uses: actions/checkout@v2
      - name: Deploy to production
        run: |
          ./deploy.sh
```

### 4.4. CI/CD Benefits

- **Reduces Manual Errors**: Automation ensures the pipeline runs the same way every time.
- **Rapid Feedback**: Developers get immediate feedback if something goes wrong, speeding up the iteration process.
- **Improves Data Integrity**: Tests validate that the pipeline behaves as expected, reducing the chances of incorrect data being processed.
- **Accelerates Deployment**: New updates to the pipeline are automatically deployed to production without manual intervention.

**NOTE:** I have use drive to upload all files.

## 5. Database Loading (MongoDB)

### 5.1. Data Upsertion into MongoDB

The process of loading data into MongoDB involves reading the cleaned and transformed data from a CSV file and inserting it into a MongoDB collection. In the load_to_db.ipynb file, the pandas library is used to load the final cleaned data from a CSV file stored in Google Drive. The script connects to MongoDB using the pymongo library, authenticating through a URI that points to the MongoDB Atlas cluster. The data is converted into a list of dictionaries using the to_dict() function of the DataFrame, which is suitable for insertion into MongoDB. The insert_many() function of the pymongo library is then used to upsert the data into the target MongoDB collection, ensuring that the latest data is inserted or updated as necessary. This process allows for efficient data storage and retrieval in MongoDB for future analysis or reporting.

### 5.2. Data Retrieval from MongoDB

Once the ETL pipeline has processed and loaded the stock data into MongoDB, retrieving this data for further analysis or validation becomes a crucial step. In this pipeline, data retrieval from MongoDB is facilitated through the use of Python's `pymongo` library, which provides a simple and efficient way to interact with MongoDB databases.

The process begins by establishing a connection to the MongoDB database using the connection URI, which includes the authentication credentials required to access the database. The following steps are involved in the data retrieval process:

1. **Establishing Connection**: A connection is made to the MongoDB cluster using the `MongoClient` class from `pymongo`. The URI string contains the necessary credentials for accessing the database. This URI connects to the appropriate cluster and database instance.
2. **Selecting the Database and Collection**: Once the connection is established, the next step is selecting the specific database and collection within MongoDB where the stock data

has been stored. The `db` object refers to the target database, and the `collection` object refers to the collection holding the stock data.

3. **Querying the Data**: After selecting the correct collection, data can be retrieved using the `find()` method, which allows for querying the collection. This method returns a cursor that can be iterated over to extract documents (rows of data). The data is then converted into a format that can be easily manipulated and analyzed, such as a Pandas DataFrame.
4. **Data Processing**: Once the data is retrieved, it can be processed and analyzed using various Python libraries, like `pandas`. The retrieved data is typically transformed into a DataFrame for easy analysis, allowing for tasks such as data cleaning, transformation, and visualization.
5. **Validation**: Data validation can also be carried out at this stage by checking for anomalies, missing values, or incorrect formats. This ensures the integrity and consistency of the data retrieved from MongoDB before further analysis or processing.

## 6. Technical Justification

### 6.1. Tools & Technologies Chosen

- **Python**: Chosen for its vast ecosystem of libraries for data manipulation (pandas, requests, etc.), as well as its ease of use and integration capabilities.
- **MongoDB**: A NoSQL database is used due to its ability to handle large amounts of unstructured data (stock data) efficiently.
- **Google Sheets API**: Used to fetch data directly from Google Sheets for integration with cloud-based systems.
- **BigQuery**: Used for handling large datasets of historical stock data.
- **CI/CD (GitHub Actions)**: Chosen for its simplicity and integration with GitHub repositories, automating testing, and deployments.

### 6.2. Benefits of the Chosen Tools

- **Python**: Ease of use and flexibility for implementing ETL pipelines.
- **MongoDB**: Scalability and flexible schema to store large datasets.
- **Google Sheets**: Quick integration for small to medium-scale data.
- **CI/CD**: Ensures automatic testing and deployment, reducing human errors and ensuring reliability in production environments.

## 7. Conclusion

Provide a summary of the project:

- The pipeline automates the extraction, transformation, and loading of stock data from multiple sources.
- The automation and CI/CD integration ensure that the pipeline is reliable and efficient, with minimal manual intervention required.

- The project demonstrates the power of modern tools like Python, MongoDB, and CI/CD in building robust data processing systems.