**Project01**

***Financial Modelling Prep (FMP) Stock Data ETL, Reporting & Modelling Project (Local to Azure Implementation)***

## Project Description and Background

My background in data has to date has primarily been in reporting and I had taken some time away from work to obtain the Microsoft Azure Data Engineering Associate certification to expand my competencies upstream in ETL and ETL using cloud tools. The next logical step in my mind following this certification was to put the knowledge into practice by building pipelines in Azure, but I thought it would be prudent to start with a local prototype of any project first. This would let me get familiar with the specific tools, concepts, and scripting conventions before tackling a cloud implementation which would be relatively quick once I had laid the groundwork with this local phase.

Initially, I developed separate scripts to fetch, preprocess, and import stock data into a Postgres database. The primary dataset consisted of API requests for price quotes of a selection of stocks, covering East Coast US trading hours (approximately 1:30 am to 8 am AEDT). Developing this didn’t take me long and I found this setup basic, so I added reporting features. This started with a daily PDF report generated within my pipeline (using Pandas for descriptive analysis), then added a Power BI dashboard fed by my stock price pipeline in conjunction with my Postgres database retaining the stock price data.

The visuals from the intraday-focused stock quote API were limited and I was dissatisfied that the dataset, at one table, did not lend itself to demonstrating Kimball modelling competencies. I created a pipeline for a company info dimension table, populating my dashboard with company profiles. I built a pipeline for an earnings statement fact table, which lets me easily compare companies' performance over years rather than months as permitted by my stock price charts. To add more depth and historical filtering options to my stock quote dataset, I developed scripts which merged datasets from different endpoints and loaded historical stock prices spanning back to the year 2000 into my existing table.

I was tired of manually running my scripts and instead of waiting to create my planned cloud setup (scheduled runs using Data Factory) to manage orchestration, I incorporated Airflow into this phase of the project. This meant adjusting my scripts to fit its DAG file structure and setting up scheduling within the daily pipeline (managed at the DAG's default argument level). I also switched to Airflow's built-in modules to handle API requests and securely manage environment variables like my Postgres sever password and API key.

Persistent issues with Airflow on macOS (unwanted task forking and unsuccessful troubleshooting via Airflow config) led me to switch the project to a Windows PC. Next, I containerised my Airflow pipelines using a Dockerfile, packaging it with all the dependencies (Airflow, Postgres, Apache Spark, Java, Plotly, Pandas, SQLAlchemy, etc.) after which I used Docker Compose to streamline the setup of these containers and configure Docker-network communication. While Docker wasn't a part of my original plan, I figured using it here it would streamline future projects and give me a chance to gain hands-on experience with containerisation.

I'm nearing the end of this phase of the project and am currently using Apache Spark (w/ PySpark) to analyse my stock quote dataset (currently at ~2 million rows), I'm performing transformations and initial analysis that will go into a report weekly report. Once this is completed, I'll deploy this project to Azure using Data Factory, Azure Functions, Data Lake Gen2 for storage likely semi-structured only and Azure Databricks for transformations.

## Tools Used and Libraries Used

* Docker
* Git / Github
* Python 3.11
* Apache Airflow 2.8.0
* Apache Spark 3.5.0
* Pandas 2.0
* Pyspark 3.5.0
* ReportLab
* Postgres 14
* Power BI

## Pipeline #1 - Daily Stock Price Pipeline

This Apache Airflow DAG (Directed Acyclic Graph), named Stock\_Price\_Daily\_Pipeline, automates the process of fetching, preprocessing, analysing, and storing stock price data. It is designed to run weekdays at 14:30 UTC to correspond with the start of trading on the East Coast of the United States.

**Workflow Steps**

1. **Fetch Stock Data (fetch\_stock\_data)**:
   * Retrieves stock price data for a list of symbols from an API.
   * Collects data over 6.5 hours from the start time.
   * Saves the data to a JSON file.
2. **Preprocessing (preprocessing)**:
   * Loads data from the JSON file.
   * Validates and flattens the data structure.
   * Selects relevant fields for analysis.
   * Performs data type conversions.
   * Saves the cleaned data to a Parquet file.
3. **Data Analysis (data\_analysis)**:
   * Loads the preprocessed data.
   * Identifies top price movements.
   * Calculates volatility.
   * Determines stocks trading furthest above their 50-day moving average.
   * Creates a PDF report with tables.
4. **Import to Database (import\_to\_database):**
   * Loads preprocessed data from the Parquet file.
   * Inserts data into a PostgreSQL database.

**Key Components**

* **Operators:** Utilises PythonOperator for task execution.
* **Data Handling:** Employs Pandas and ReportLab.
* **Scheduling:** Runs on weekdays at a specific time.
* **Environment Variables:** Stores API keys and database passwords securely.

**Execution Flow (Sequential)**

fetch\_stock\_data >> preprocessing >> data\_analysis >> import\_to\_database

**Pipeline #1 – Enrichment - Historical Stock Data Processing Pipeline**

This Airflow DAG, named historical\_stock\_fact\_dag, automates fetching, processing, and storing historical stock data which has been configured for manual or externally scheduled triggering. It need only be used if data parameters change following the initial load, or in the event of repopulating the database.

**Workflow Description**

1. **Fetch Historical Data (fetch\_historical\_data)**
   * Retrieves stock price and market capitalisation data from financial APIs.
   * Handles API calls, responses, and potential errors.
   * Saves data into separate JSON files.
2. **Preprocess Data (preprocess\_data)**
   * Loads JSON data and normalises nested structures.
   * Selects relevant fields, converts date formats, and renames columns.
   * Merges price and market capitalisation data.
   * Fills missing values and performs data type conversions.
   * Saves the cleaned data as a Parquet file.
3. **Import Data to Database (import\_data)**
   * Loads the Parquet file into a DataFrame.
   * Inserts the data into a PostgreSQL database (requires correct credentials).

**Key Components**

* **PythonOperator:** Provides flexibility for custom data processing logic.
* **Environment Variables:** Securely store API keys and database passwords.
* **Pandas and JSON:** Data manipulation and handling.
* **Logging:** Tracks execution status, errors, and progress.

**Execution Flow (Sequential)**

fetch\_historical\_data >> preprocess\_data >> import\_data

## Pipeline #2 - Earnings Statement Pipeline

The DAG consists of three main tasks:

1. **Fetch and Save Data**: Retrieves annual earnings data for specified stock symbols from an API and saves the raw data in JSON format.
2. **Process and Save Data**: Reads the raw JSON data, normalises it into a structured format using Pandas, and then saves it as a Parquet file for efficient storage and access.
3. **Load Data to Database**: Loads the processed data from the Parquet file into a PostgreSQL database, making it available for analysis and reporting.

**Workflow Description**

1. Fetch and Save Data (**fetch\_and\_save\_data**)

* Utilises Airflow's **HttpHook** to make GET requests to an external API, fetching earnings data for a list of stock symbols.
* The fetched data is logged and stored in a JSON file, **earnings\_data.json**.

2. Process and Save Data (**process\_and\_save\_data**)

* Reads the stored JSON data, flattens nested structures, and converts it into a Pandas DataFrame.
* The DataFrame is then saved as a Parquet file, **earnings\_data.parquet**, optimising for space and read efficiency.

3. Load Data to Database (**load\_data\_to\_database**)

* Loads the Parquet file into a DataFrame and inserts the data into a PostgreSQL database table named **Earnings\_Fact**.
* Database connection details are managed through environment variables for security.

**Key Components**

* **PythonOperator**: Executes Python functions for each task within the DAG, enabling custom data manipulation and interaction with external services.
* **Airflow Variables and Environment Variables**: Securely manage sensitive information such as API keys and database credentials.
* **Pandas**: Used for data processing and transformation, facilitating the conversion of raw JSON into a structured and queryable format.
* **SQLAlchemy**: Handles database connections and operations, allowing for seamless integration with PostgreSQL.

**Execution Flow (Sequential)**

fetch\_and\_save\_task >> process\_and\_save\_task >> load\_data\_to\_db\_task

## Pipeline #3 - Company Info Pipeline

This Airflow DAG, named **company\_info\_dimension\_dag**, automates the collection, processing, and storage of company information to a dimension table in a postgres database. This pipeline is scheduled to run on a yearly basis and uses a Type-0 slowly changing dimension (i.e. overwrites the data).

**Workflow Description**

1. **Fetch Data (fetch\_data)**
   * Retrieves company profiles for specified stocks from an API (financialmodelingprep.com).
   * Saves data as a JSON file (**companyinfo\_data.json**).
2. **Process Data (process\_data)**
   * Loads the JSON file and validates its structure.
   * Converts data into a tabular format (DataFrame).
   * Performs data cleaning and type conversions.
   * Saves the processed data as a Parquet file (**companyinfo\_data.parquet**).
3. **Load Data (load\_data)**
   * Loads the Parquet file.
   * Inserts data into the **CompanyInfo\_Dim** table in a PostgreSQL database.

**Key Components**

* **PythonOperator:** Executes custom Python functions for flexibility.
* **Airflow Variables:** Securely store API keys and database credentials.
* **Pandas:** For data manipulation and transformation.
* **JSON and Parquet:** Efficient data storage formats.
* **Logging:** Tracks execution status and errors.

**Execution Flow (Sequential)**

fetch\_data\_task >> process\_data\_task >> load\_data\_task

## Weekly Report

(This report is automated by Airflow and uses Apache Spark for efficient transformation and analysis of the complete stock price dataset (~2 million rows at present). This will incorporate machine learning functionality and is currently under development.)

## Power BI Dashboard

This dashboard leverages data gathered from my pipelines to highlight practical chart design, interactive features, and intuitive navigation. It prioritises best practices in data modelling (STAR schema) and integrates date/time intelligence functions. While I have primarily used inbuilt aggregations for the charts, this proof of concept is scalable for more complex DAX calculations as user requirements expand.

**Overview Tab**

Market Cap Growth

* Tracks growth in market capitalisation over time for selected companies. Useful for analysing historical performance.

Valuation and Profitability

* Compares companies' price-to-earnings ratios with return on equity. Assists in assessing financial health and investment potential.

Market Share

* Tree map visualisation of market capitalisation by company enabling a quick view of companies' market performance.

**Stock Tabs**

Daily Price Change

* Line chart of daily stock price percentage changes – contrasted against price fluctuations of an ETF matching the Dow Jones Industrial Average to give a view of daily price volatility within a given month.

Volume vs. Price

* Bar and line chart comparing trading volume to stock prices. Illustrates the relationship between market activity and price changes and identifies trading days of interest for further investigation.

Intraday Price Movement

* Detailed line chart for intraday stock price analysis to explore shore term price movement, to the minute, for a particular stock.

**Technical Features**

**Data Model**

* STAR schema model drawing from the established pipelines drawing from stock price quotes, company information tables and earnings fact table for analysis.
* **Data from additional endpoints (e.g. –** [**FMP endpoints**](https://site.financialmodelingprep.com/developer/docs)**, say a balance sheet fact table) could be incorporated into the model by creating new tables and establishing relationships using appropriate keys and cardinality to maintain the integrity of the fact-dimension structure.**

**DAX Measures**

As mentioned above I have leaned on inbuilt aggregation features for this Power BI model and used only a few DAX measures to provide (largely) cosmetic flavour, acknowledging the scope for more complex aggregations. Measures used:

* **First Two Sentences** extracts the first two sentences from a 'description' field. This handles cases where there might be fewer than two sentences.
* **Closing Price2** calculates the latest stock price in a day, ensuring data accuracy.
* **Chart Title Date2** provides dynamic chart titles, improving user interaction.

**User Experience**

* Slicers are present on all tabs. On ‘Overview’ the ‘symbol’ slicer interacts with all charts and the ‘Year’ slicer interacts only with the bottom row – as the top row is an innate long-term comparison of market cap.
* On the company tabs ‘Month’ slicer interacts with all charts which reside within month windows and the day slicer interacts with the Intraday ‘Price Movement Chart’ which is day to day.