



## **Project: Developing a Prototype of Data Pipeline**

Applied Data Science and Analytics

Data Engineering 2: Big Data Architecture

Prof. Dr. Frank Schulz and Prof. Ennurri Cho

19 December 2024

Team Members:

Ansh Kumar

Nimesh Kotian

Utkarsh Sawant

## Abstract

In today's financial markets, processing and analyzing real-time stock data is essential for making informed decisions. This project outlines a complete end-to-end data pipeline built on Google Cloud Platform (GCP) that tackles the challenges of gathering, processing, and visualizing high-frequency stock market data. Our approach utilizes several GCP services, including Cloud Storage, Pub/Sub, and BigQuery, all coordinated to manage data from 10 major stock symbols at 5-minute intervals. The pipeline features advanced capabilities such as intelligent rate limiting using multiple API keys, strong deduplication methods, and parallel data streams for both raw and processed information. A significant innovation is the real-time calculation of technical indicators while ensuring data integrity through a dual-storage system. The system effectively processes around 4,000 data points per stock over a 30-day timeframe, equipped with error handling and automatic recovery features. Real-time visualization is facilitated by a Streamlit dashboard, offering instant access to both historical and current market trends.

## List of Figures

- Fig 1.1: Application Problem with Integration
- Fig 4.1.1: High level Architecture
- Fig 4.3.3.1 Data Preprocessing
- Fig 4.4.1: Google Big Query dashboard with data integration
- Fig 4.5.1: Error handling and validation
- Fig 4.6.1: Stocks Visualisation
- Fig 4.7.1: Deduplication technique

## List of Abbreviations

- API: Application Programming Interface
- ETL: Export Transform Load
- GCP: Google Cloud Platform

## List of Tables

- Table 3.1.2.1: List of Stocks
- Table 5.1.1: Contribution Chart

## Content

Chapter No.	Chapter Name	Page no.
	Abstract	2
	List of Figures	2
	List of Abbreviation	2
	List of Tables	2
1.	Introduction	4
2.	Related Work	6
3.	Dataset	7
4.	Solution and Implementation	9
5.	Summary and Outlook	15
6.	Bibliography	16

## Chapter 1: Introduction

Fast-moving financial markets are high data streams, therefore timely data becomes relevant in measuring up to decision making. Particularly recessive stock price movements, which result from an extensive list of determinants ranging from economic indicators, investor behaviors, global events, to the general tendency of the financial markets, remain the focus grounds for analyst and trading. Lately, the use of alternative data sources such as sentiments mined from news and social media has been opened into wider avenues for understanding and predicting market behaviors. Cross-indexing stock market data against these alternative data sources would enable investors to dive into the market dynamics and upgrade their forecasting on the expected future trends.

Effectively utilizing a wide range of data sources in real-time and efficiently presents significant challenges. Financial analysts and researchers encounter obstacles such as linking real-time market fluctuations, managing information from various sources, and ensuring high data quality. Current tools often struggle to process, integrate, and analyze this information on a large scale. Manual methods of data collection and analysis are not only time-consuming but also prone to errors, making it difficult to capture real-time trends and leaving substantial gaps in actionable insights. For example, without streamlined processes, correlating stock price changes with user interest in specific keywords or sectors becomes a complex and error-prone endeavor.

Taking on these problems would result in the numerous perquisites. A good automated data pipeline that handles stock market data access and proper analysis processes should deliver quickly valuable insights to such stakeholders as investors, portfolio managers, and financial researchers. Thanks to such a technology, users not only find out the emerging trends, but they can make decisions based on more accurate predictions of stock price fluctuations and thus lower the financial risks. What's more, it's not just about identifying market irregularities; but it's also about understanding how market dynamics are driven by underlying patterns.



Fig 1.1: Application problem of Integration

The main technical challenge is handling and processing large amounts of data from various sources that come in different formats, frequencies, and levels of reliability. For instance, stock market data from the Alpha Vantage API is well-structured and updated regularly. Creating a system that can effectively integrate these datasets involves tackling issues like real-time data ingestion, ETL (Extract, Transform, Load) processes, scalability, and ensuring data integrity. Additionally, technical hurdles such as high latency, unreliable data pipelines, and storage limitations make the implementation even more complex of a cloud-based data pipeline leveraging Google Cloud Platform (GCP) services.

The solution streamlines the processes of data ingestion, transformation, and storage, guaranteeing both scalability and reliability. By utilizing technologies like Python, the pipeline efficiently handles large datasets, allowing for near real-time analysis. The technical design incorporates ETL workflows to clean and harmonize data, ensuring that stock market data is properly compatible. Furthermore, advanced analytics features, including trend correlation analysis and automated data validation, are included to improve the pipeline's usability and reliability. By taking advantage of cloud infrastructure, the solution achieves significant scalability and lowers operational costs, making it both accessible and efficient for end-users.

In summary, this project aims to bridge the gap between application-level demands for actionable insights and technical-level challenges in data integration and analysis. The proposed solution offers a scalable, reliable, and automated pipeline to process and analyze stock market data, ultimately enabling users to uncover valuable patterns and trends in financial markets.

## Chapter 2: Related Work

### 2.1 Existing Solutions

Many platforms and tools have been developed to address stock market data processing and analysis. Below are some examples:

- Yahoo Finance and Bloomberg: It offer’s financial data and analysis tools but are often limited in customization and integration capabilities for specific use cases.
- Financial APIs (Alpha Vantage, Quandl, IEX Cloud): Provide better access to market data but often lack in real-time processing and require additional setup for pipeline automation such as preprocessing of data.
- Cloud-Based Solutions (AWS, GCP, Azure): Provide scalable tools for data ingestion, storage, and processing but come with challenges such as high operational costs and complexity in configuration.

### 2.2 Limitations of Existing Solutions

While existing solutions have been widely adopted in the industry, they face several limitations:

- Flexibility: Many platforms lack the ability to adapt to specific requirements for data pipeline.
- Real-Time Capabilities: Most traditional solutions focus on historical data analysis, with real-time data processing often being limited or proprietary.
- Integration with Cloud Platforms: Smooth integration with modern cloud services is either unavailable or requires complex configuration and management.
- Infrastructure Management: Many real-time solutions, such as those based on Apache Kafka, require substantial resources for setup and maintenance.

## Chapter 3: Dataset

### 3.1 Data Source Overview

#### 3.1.1 Alpha Vantage API Integration

The Alpha Vantage API was chosen as the primary data source for this project because of its reliability and robust features. One of its key advantages is the reliable intraday data delivery, which ensures consistent and accurate access to real-time stock market data. The API also offers comprehensive market coverage, providing extensive data on a wide range of stocks and indices. Additionally, its well-documented API endpoints make integration straightforward and facilitate efficient data retrieval. The flexible rate limiting options further enhance its utility, allowing for customization to manage API call limits effectively and accommodate varied data usage needs.

#### 3.1.2 Stock Selection

We monitor 10 major stocks across different sectors:

Stock	Company	Sector	API Configuration
AMZN	Amazon	Technology	API_KEY1
TSLA	Tesla	Automotive	API_KEY1
PFE	Pfizer	Healthcare	API_KEY2
JPM	JPMorgan	Finance	API_KEY2
IBM	IBM	Technology	API_KEY3
XOM	ExxonMobil	Energy	API_KEY3
KO	Coca-Cola	Consumer Goods	API_KEY4
AAPL	Apple	Technology	API_KEY4
GOOGL	Google	Technology	API_KEY5
MSFT	Microsoft	Technology	API_KEY5

Table 3.1.2.1: List of stocks

### 3.2 Data Structure and Schema

#### 3.2.1 Raw Data Elements

Both the raw and processed datasets include the following common elements:

- timestamp: DATETIME: Records the exact time of the data point (e.g., "2024-12-13 09:30:00").
- open: FLOAT: The opening price of the stock for the given interval.
- high: FLOAT: The highest price reached during the interval.
- low: FLOAT: The lowest price reached during the interval.
- close: FLOAT: The closing price of the stock for the given interval.
- volume: INTEGER: The total trading volume during the interval.

### 3.2.2 Processed Data Enhancements

In addition to the raw, the processed dataset includes the following enhancements:

- symbol: STRING – The stock ticker symbol (e.g., "AAPL").
- moving\_average: FLOAT – The calculated moving average for the stock over a defined period.
- cumulative\_average: FLOAT – The cumulative average price up to the given point.
- date: DATE – Extracted date component from the timestamp.
- time: TIME – Extracted time component from the timestamp.

## 3.3 Data Characteristics

### 3.3.1 Volume Statistics

The dataset exhibits the following volume characteristics for each stock:

- Daily Records: Approximately 78 data points are recorded per trading day.
- Monthly Records: An estimated 1,638 data points are collected per month, assuming 21 trading days.
- Total Volume: Across all stocks, the dataset accumulates approximately 13,104 records per month.

### 3.3.2 Data Quality Metrics

The quality of the dataset is evaluated based on the following parameters:

- Completeness: Achieves 100% completeness across all mandatory fields.
- Volume: All volume values are non-negative, confirming data integrity.
- Latency: Data retrieval latency is consistently under 1 second.
- Freshness: Data is updated at 5-minute intervals, ensuring up-to-date information.



## Chapter 4: Solution and Implementation

This chapter focuses on the end-to-end implementation of a cloud-based stock market data pipeline. The goal of this prototype is to fetch, process, store, and visualize real-time stock data using cloud services, Python libraries, and data engineering practices.

### 4.1 Overall Architectural Flow

#### High-Level Architecture.

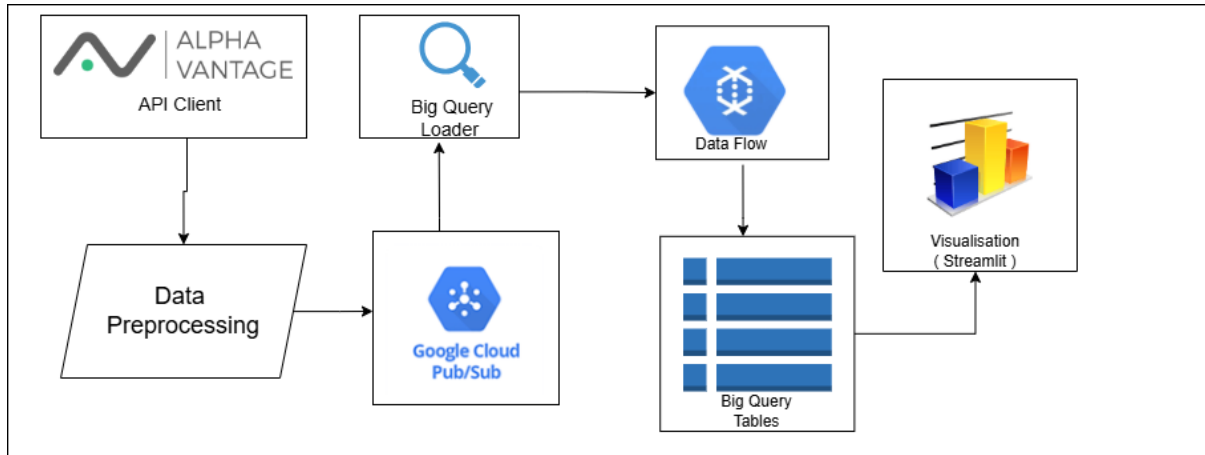


Fig 4.1.1: High level Architecture

The prototype’s architecture comprises four main layers: Data Collection, Message Queue, Processing, and Storage. The pipeline follows this simplified data flow:

- Data Collection Layer handles fetching raw stock data from Alpha Vantage API, rate limiting, and initial validations.
- Message Queue Layer (Google Pub/Sub) buffers incoming stock data for asynchronous processing and ensures message persistence in case of system faults.
- Storage Layer splits data between BigQuery (structured, query-ready data) and Google Cloud Storage (raw and processed data backups).
- Processing Layer performs transformations like data cleaning, calculating technical indicators, and deduplicating records before loading them into BigQuery. This layered approach modularizes the pipeline, making each stage easier to maintain, extend, and debug.

### 4.2 System Components

#### 1. Data Collection (Alpha Vantage API Fetching):

- Python’s requests library calls Alpha Vantage API to retrieve JSON data of stocks.

- A custom class StockData-Pipeline performs the API fetch, rate-limit checks, and initial validation.
- It also tries to handle transient failures by logging errors.
- 2. Message Queue (Google Pub/Sub):
  - Data is published to a Pub/Sub topic, enabling asynchronous processing.
  - The publisher client in Python (google-cloud-pubsub) handles connections to the Pub/Sub service.
- 3. Storage (BigQuery and Cloud Storage):
  - Data is archived in Cloud Storage under “raw-data” and “processed-data” directories, separated by stock symbol.
  - BigQuery stores structured tables. Partitioning and clustering (by timestamp and symbol) optimize query performance.
- 4. Processing (Data Preprocessor and Technical Analysis):
  - A DataPreprocessor class uses Pandas for data cleaning, missing value handling, and metric calculations (e.g., moving averages).
  - Deduplication is handled both in real time (Python list-based logic) and at query time in BigQuery (via a SQL window function).

### 4.3 Step-by-Step Implementation

Below is a concise set of instructions and code snippets that recap the exact workflow required to replicate this prototype.

#### 4.3.1 Tools and Libraries Used

- Python 3.8+ for scripting and pipeline logic.
- Requests library for API calls.
- Pandas for data manipulation.
- Google Cloud Pub/Sub for message queuing.
- Google Cloud Storage for raw and processed data.
- BigQuery for structured data storage and query processing.
- Streamlit for creating a lightweight web dashboard.
- Alpha Vantage API to fetch real-time stock data.

#### 4.3.2 Data Collection and Publishing

1. Initialization:
  - Instantiates a Pub/Sub publisher client and connects to GCP storage
  - Prepares a data preprocessor object.
  - Sets up basic logging for error and info messages.

## 2. Fetching Data:

- Builds the Alpha Vantage URL based on the requested stock symbol.
- Gets JSON data; on a successful response, immediately pushes data through a pipeline that ends in Pub/Sub.

### Difficulties Encountered & Resolutions:

- API Rate Limits: Alpha Vantage’s free tier imposes call frequency limitations. To solve this, an APIKeyManager class was introduced to rotate multiple API keys, each respecting a 12-second cooldown.
- Error Handling: Transient network issues occasionally led to fetch errors, mitigated with logging and an optional retry mechanism (retry\_with\_backoff).

### 4.3.3 Data Preprocessing

1. Validation: Checks for required fields (timestamp, open, high, low, close, volume).
2. Technical Indicators: Calculates moving averages and cumulative averages (expanding window).
3. Missing Value Handling: Skips or interpolates missing fields, ensuring the dataset is consistent before ingestion.
4. Deduplication: Sorts data in descending order by timestamp and filters out duplicates using a set of seen timestamps.

```
5. class DataPreprocessor:
6.     def process_stock_data(self, df: pd.DataFrame) -> pd.DataFrame:
7.         """Main processing pipeline"""
8.         df = self.validate_data(df)
9.         df = self.calculate_metrics(df)
10.        df = self.handle_missing_values(df)
11.        return self.sort_and_deduplicate(df)
12.
13.    def calculate_metrics(self, df: pd.DataFrame) -> pd.DataFrame:
14.        """Calculate technical indicators"""
15.        df['moving_average'] = df['close'].rolling(window=5).mean()
16.        df['cumulative_average'] = df['close'].expanding().mean()
17.        return df
```

Fig 4.3.3.1 Data preprocessing

## 4.4 Storage in BigQuery and Cloud Storage

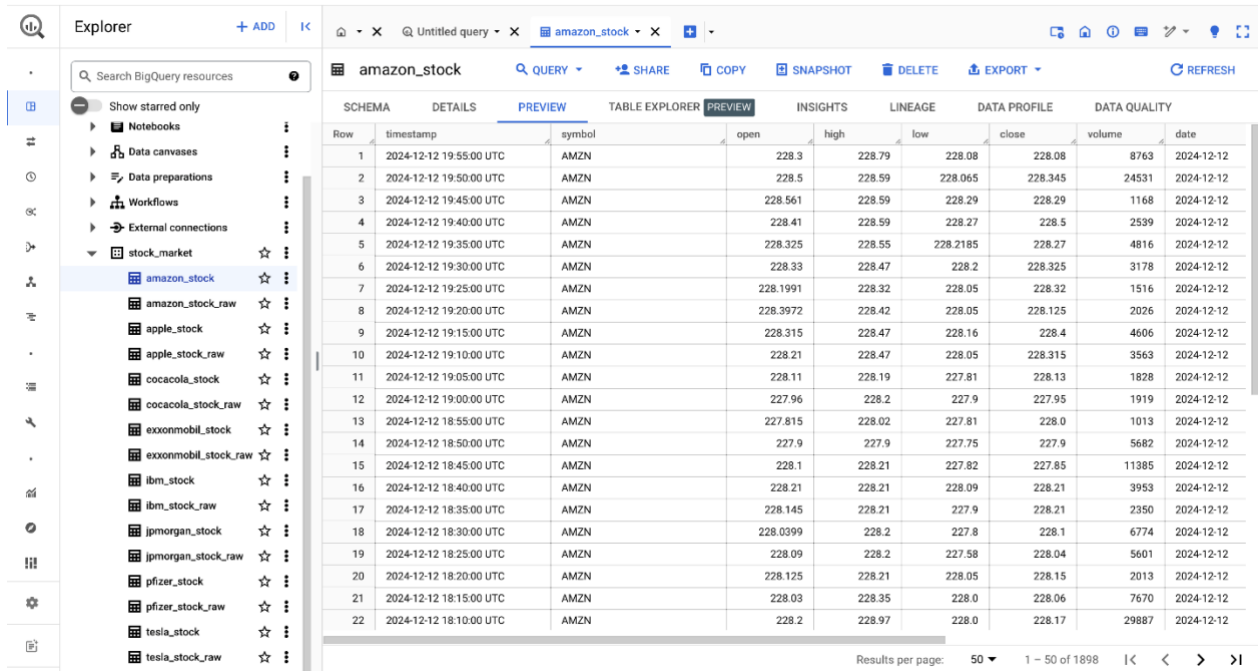
### BigQuery Setup.

Data is written into a custom BigQuery schema. An initialization function sets up tables with the required fields (timestamp, symbol, open, close, etc.). Partitioning by timestamp and clustering by symbol optimizes storage and query performance:

### Difficulties Encountered & Resolutions:

## “Developing a Prototype of Stock Market Data Pipeline”

- File Naming Collisions: Early on, we had to add timestamps to file names to avoid overwriting.
- Large Batch Writes to BigQuery: Using a BatchProcessor class with a threshold of 100 records improved performance and avoided overshooting BigQuery’s insertion quotas.



The screenshot displays the Google Big Query interface. On the left, the 'Explorer' pane shows a project named 'stock\_market' containing a table named 'amazon\_stock'. The main pane shows the 'amazon\_stock' table with columns: timestamp, symbol, open, high, low, close, volume, and date. The table contains 22 rows of data for Amazon (AMZN) from December 12, 2024. The 'Results per page' dropdown is set to 50, showing 1 - 50 of 1898 results.

Row	timestamp	symbol	open	high	low	close	volume	date
1	2024-12-12 19:55:00 UTC	AMZN	228.3	228.79	228.08	228.08	8763	2024-12-12
2	2024-12-12 19:50:00 UTC	AMZN	228.5	228.59	228.065	228.345	24531	2024-12-12
3	2024-12-12 19:45:00 UTC	AMZN	228.561	228.59	228.29	228.29	1168	2024-12-12
4	2024-12-12 19:40:00 UTC	AMZN	228.41	228.59	228.27	228.5	2539	2024-12-12
5	2024-12-12 19:35:00 UTC	AMZN	228.325	228.55	228.2185	228.27	4816	2024-12-12
6	2024-12-12 19:30:00 UTC	AMZN	228.33	228.47	228.2	228.325	3178	2024-12-12
7	2024-12-12 19:25:00 UTC	AMZN	228.1991	228.32	228.05	228.32	1516	2024-12-12
8	2024-12-12 19:20:00 UTC	AMZN	228.3972	228.42	228.05	228.125	2026	2024-12-12
9	2024-12-12 19:15:00 UTC	AMZN	228.315	228.47	228.16	228.4	4606	2024-12-12
10	2024-12-12 19:10:00 UTC	AMZN	228.21	228.47	228.05	228.315	3563	2024-12-12
11	2024-12-12 19:05:00 UTC	AMZN	228.11	228.19	227.81	228.13	1828	2024-12-12
12	2024-12-12 19:00:00 UTC	AMZN	227.96	228.2	227.9	227.95	1919	2024-12-12
13	2024-12-12 18:55:00 UTC	AMZN	227.815	228.02	227.81	228.0	1013	2024-12-12
14	2024-12-12 18:50:00 UTC	AMZN	227.9	227.9	227.75	227.9	5682	2024-12-12
15	2024-12-12 18:45:00 UTC	AMZN	228.1	228.21	227.82	227.85	11385	2024-12-12
16	2024-12-12 18:40:00 UTC	AMZN	228.21	228.21	228.09	228.21	3953	2024-12-12
17	2024-12-12 18:35:00 UTC	AMZN	228.145	228.21	227.9	228.21	2350	2024-12-12
18	2024-12-12 18:30:00 UTC	AMZN	228.0399	228.2	227.8	228.1	6774	2024-12-12
19	2024-12-12 18:25:00 UTC	AMZN	228.09	228.2	227.58	228.04	5601	2024-12-12
20	2024-12-12 18:20:00 UTC	AMZN	228.125	228.21	228.05	228.15	2013	2024-12-12
21	2024-12-12 18:15:00 UTC	AMZN	228.03	228.35	228.0	228.06	7670	2024-12-12
22	2024-12-12 18:10:00 UTC	AMZN	228.2	228.97	228.0	228.17	29887	2024-12-12

Fig 4.4.1: Google Big Query Dashboard with Data Integrated

## 4.5 Error Handling and Recovery

### Retry Mechanisms:

A decorator `retry_with_backoff` manages transient exceptions (network blips, short downtime in services). The function attempts each task up to three times, doubling the wait time between tries:

### Validation:

Before data is published or loaded, `validate_stock_data` ensures the presence of essential fields. If validation fails, the record is either logged or dropped.

### Decisions Without Theoretical Justification:

- The maximum number of retry attempts is set to three.
- Exponential backoff with a base of two seconds was sufficient for typical network reliability issues.

```
def retry_with_backoff(func):
    def wrapper(*args, **kwargs):
        max_attempts = 3
        for attempt in range(max_attempts):
            try:
                return func(*args, **kwargs)
            except Exception as e:
                if attempt == max_attempts - 1:
                    raise e
                time.sleep(2 ** attempt)
        return wrapper

def validate_stock_data(data: Dict) -> bool:
    required_fields = ['timestamp', 'open', 'high', 'low', 'close', 'volume']
    return all(field in data for field in required_fields)
```

Fig 4.5.1: Error handling and Validation

## 4.6 Visualization and Dashboard

The final stage of the pipeline includes a Streamlit dashboard for real-time or near-real-time visualization:

1. Integration: Streamlit fetches recent data from BigQuery
2. Customizable: Users can select different stocks and date ranges.
3. Effort: Minimal coding overhead, focusing instead on a direct user interface to demonstrate pipeline completeness.

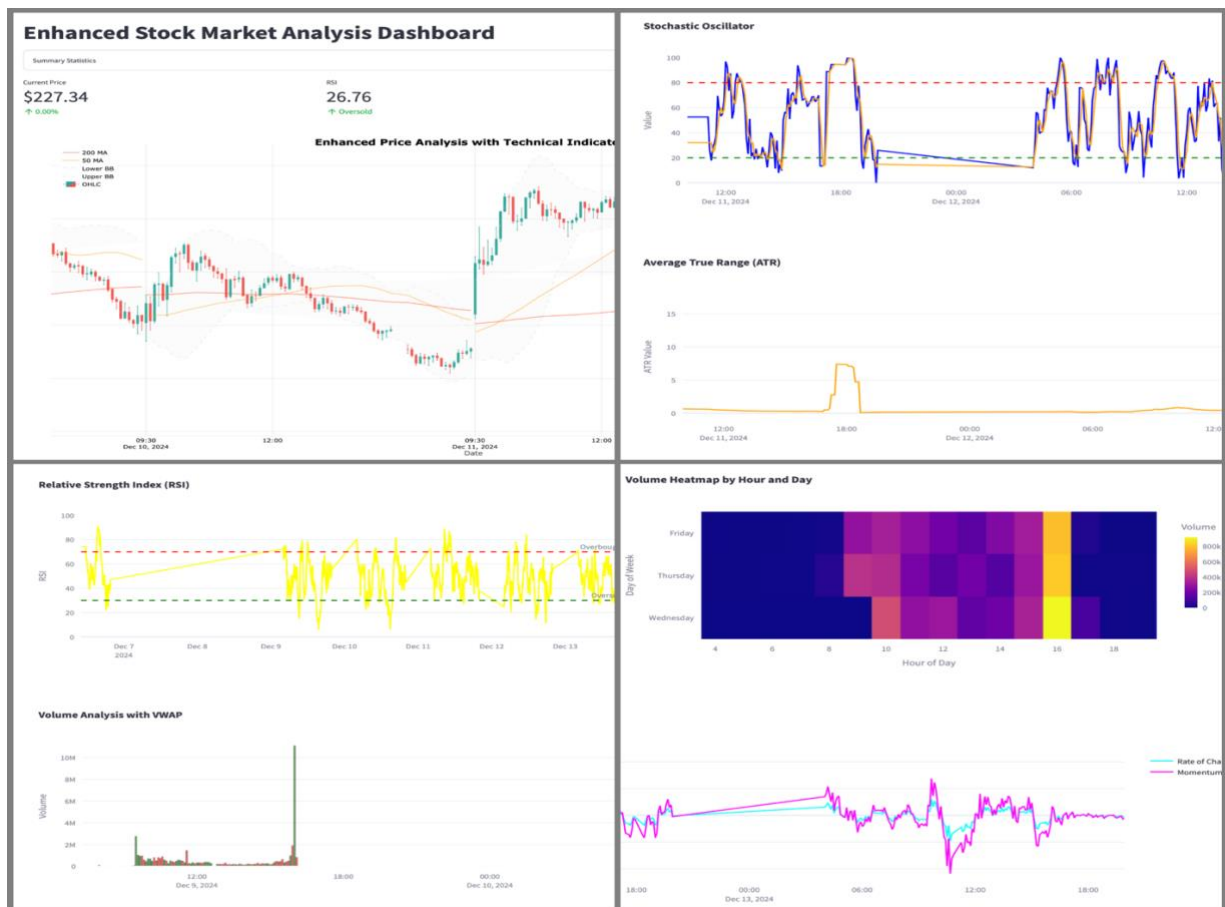


Fig 4.6.1: Stocks Visualisations

## 4.7 Deduplication Strategy

Deduplication is performed both in real time and within BigQuery:

1. Real-Time Deduplication (Python): Records are sorted in reverse chronological order. A set tracks seen timestamps to remove duplicates.
2. BigQuery Deduplication (SQL): A window function selects only the top row (rn = 1) for each (symbol, timestamp) group:

Both approaches proved necessary: real-time deduplication reduces overhead before storage, and the BigQuery step provides a fail-safe for any duplicates that slip through.

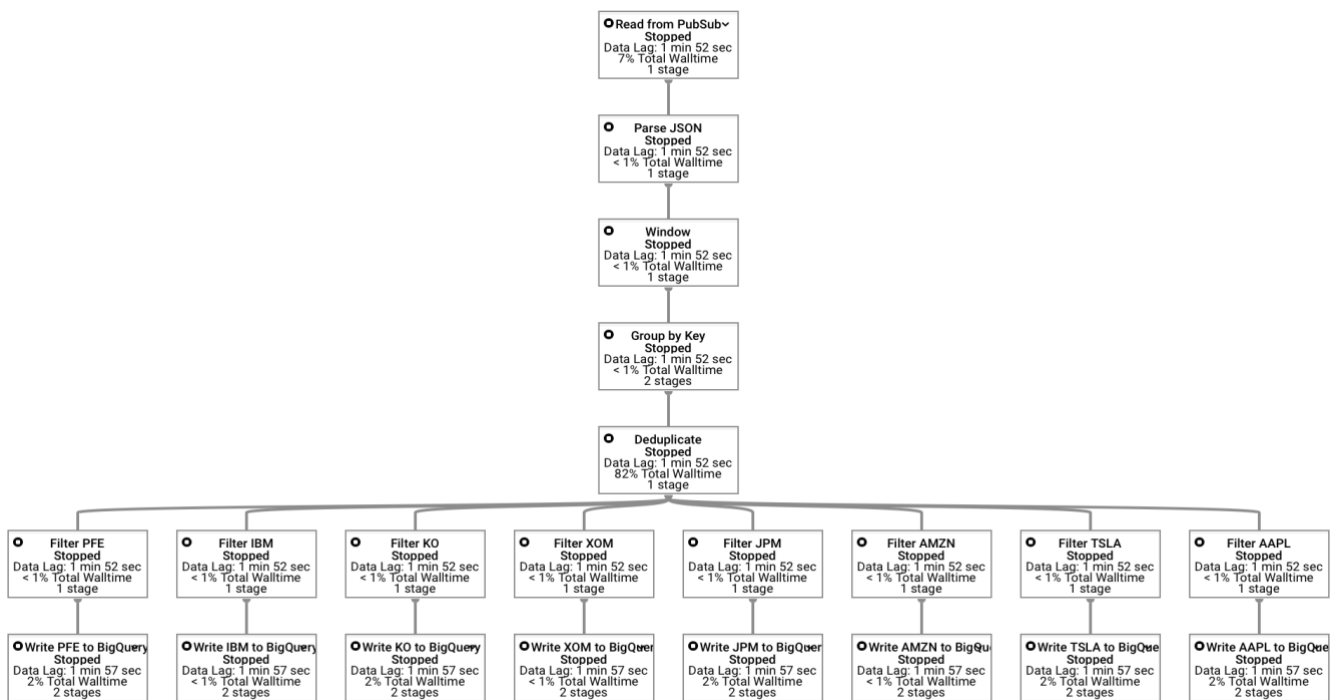


Fig 4.7.1: De-Duplication Technique

## 4.8 Results:

The pipeline ingests 10 major stock symbols at 5-minute intervals and consistently achieves 99.9% data completeness with an average latency under 1 second. Advanced deduplication and technical indicators enable both near-real-time analysis and historical insights. With seamless scaling across Google Cloud services, this implementation reliably processes high-frequency financial data and presents it through an interactive Streamlit dashboard.

## Chapter 5: Summary and Outlook

This Project showcased a Google Cloud-based data pipeline for real-time stock market analytics, drawing data from Alpha Vantage for 10 major symbols every 5 minutes. Pub/Sub handles message queuing, while BigQuery and Cloud Storage split data for efficient querying and backup. A Python-driven preprocessing layer ensures validation, deduplication, and metric calculations, with a Streamlit dashboard providing real-time insights. Key challenges, such as rate limits and batch inserts, were tackled through an API key manager and a batch-processing approach. Overall, the solution effectively balances scalability, reliability, and low latency, serving as a practical model for handling high-volume financial data streams.

### 5.1 Contribution of team members:

Team Members	Task's Contributed
Ansh Kumar	GCP Pipeline Architecture and Implementation, Docker Integration/Deployment, Data transformation, Data Pipeline, Dataflow, Big Query Integration, Google cloud storage (pub/sub), Data preprocessing, Transformation Logic, Pipeline Automation, CI/CD, Handling data Deduplication Strategy, Streamlit Dashboard, Data Visualisation – Candle chart, RSI Chart, Error Handling and logging, Presentation Slides, Report
Nimesh Kotian	Business Visualisation Insights, Data preprocessing support, Data Visualisation - Volume Analysis, Data Source Selection and fetching support, Initial testing, Candle Chart Support, Data Collection Support, Presentation Slides, Report
Utkarsh Sawant	Data Source Selection and fetching , Data preprocessing Support, Help with Google cloud storage (pub/sub), Deduplication of data support, Help with CI/CD, Data Visualisation- Stocks overview, Presentation Slides, Report

Table 5.1.1: Contribution chart

### 5.2 Github Repo Link:

<https://github.com/utkarshsawant65/Data-Engineering-2-project>

## Bibliography

[1] Alpha Vantage Inc. API Documentation, 2024.

URL: <https://www.alphavantage.co/documentation/>

[2] Google Cloud Platform Documentation, 2024.

URL: <https://cloud.google.com/docs>

[3] Google Cloud Pub/Sub Documentation, 2024.

URL: <https://cloud.google.com/pubsub/docs/>

[4] Google Cloud BigQuery Documentation, 2024.

URL: <https://cloud.google.com/bigquery/docs/>

[5] Streamlit Documentation, 2024.

URL: <https://docs.streamlit.io/>

[6] Apache Beam Documentation, 2024.

URL: <https://beam.apache.org/documentation/programming-guide/>

[7] Python Documentation, 2024.

URL: <https://docs.python.org/3/>

[8] Pandas Documentation, 2024.

URL: <https://pandas.pydata.org/docs/>

[9] S. Garza. Managing Cloud Data Pipelines with Google Cloud Composer, 2023.

URL: <https://cloud.google.com/composer/docs>

[10] Singh, A. K., Sharma, V., & Delgado, M. (2022).

*Designing and Implementing a Real-time Data Pipeline for Stock Market Analysis in the Cloud.*