Data Orchestration in Connection to Stock Fundamentals

Yuxuan Zhou

Master of Science
School of Informatics
University of Edinburgh
2023

Abstract

The modern financial market is characterized by an abundance of diverse data, such that the time devoted to data preparation frequently outweighs other crucial duties, particularly in the field of stock market fundamental analysis. This dissertation addresses this problem by introducing a comprehensive system for automatically orchestrating stock fundamental data. The system aims to automate the complex process involving the acquisition, updating, processing, and storage of data. The focus is on collecting information from reliable financial sources. In light of a comprehensive analysis of stock fundamentals, a curated list of influential factors concentrating on valuation, performance, sensitivity, and geopolitics has been generated. The system combines cutting-edge data-centric technologies, including a real-time engine for big data processing(Flink) and message middleware(Kafka). The combination of these technologies not only increases the system's efficiency, but also highlights its potential for improving stock market fundamental analysis.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Acknowledgements

Throughout this entire dissertation journey, I would like to express my deepest gratitude to my supervisor, Felipe Costa Sperb, for his consistent guidance, invaluable feedback, and endless encouragement. His knowledge and insights were crucial to the completion of this project. In addition, I am grateful to the team members for providing the necessary assistance when I was in need, which greatly facilitated the completion of my project. Furthermore, I would like to express my gratitude to the University of Edinburgh, not only for providing me with a stimulating academic environment, but also for instilling in me the values and principles that have guided my research activities. The school's education, resources, and opportunities have had a significant impact on my academic and professional development. Lastly, I am grateful to my parents for their unwavering encouragement and support throughout this process. Their belief in my abilities sustained me during the most challenging times.

Table of Contents

Chapter 1

Introduction

1.1 Background and Motivation

In the fast-paced and ever-changing financial markets, access to reliable and timely information is of great essence for making proper decisions and gaining a competitive edge. Among the vast array of data sources available, stock market fundamental analysis, which involve the evaluation of companies' balance sheets, income statements, and cash flow statements, has been proved as a key driver for estimating companies' value, understanding market trends, predicting stock performance, and supporting investment strategies[18], which prompts the finance industry to continuously seek ways to leverage this crucial method and relevant data with great potential analytical value.

However, despite the acknowledged significance of stock fundamental analysis, exploiting its full potential remains a practical challenge. With the booming development of machine learning and artificial intelligence in stock market analysis and prediction, the technical obstacles is relates to the complex nature of data acquisition and manipulation in the critical stage of data preparation. Stock fundamentals data is often scattered across a wide range of data sources, characterized by inconsistent, unstructured formats[18], and irregular update frequencies. Moreover, the analytical models commonly utilized for analysing such data typically necessitate a dataset of sufficient quality and quantity, along with timely data. Manual aggregation and pre-processing of data, which might introduce inefficiencies, delays, and inaccuracies into analytical systems, are inadequate to fulfill the requirements of modern data preparation for advanced analytical systems.

As a result, building an efficient, automated data pipeline system capable of acquiring, updating, and handling the intricacies of data related to stock fundamentals holds paramount importance in improving the accuracy and efficiency of model analysis and prediction, thereby further providing financial professionals, traders, and investors with more accurate and timely information to enhance their decision-making capabilities and market competitiveness.

1.2 Project Objective

This project is a collaborative endeavor aimed at constructing a prototype for a stock trading system, which aims to automate the core processes related to stock trading analysis and prediction. The design of this trading system involved students engaging in essential tasks such as data acquisition and orchestration, financial modeling, and formulation of financial trading strategies. This project specifically concentrates on the initial aspect of the system's development, namely, data acquisition and orchestration.

To address the aforementioned challenges and fulfill the motivation of this dissertation, the objective of this project is to create a stock fundamental data pipeline system which is capable of acquiring, real-time updating, processing, formatting, and storing data. Therefore, the orchestration system could be seamlessly integrated with the downstream analytical modelling components of prototyped the stock trading system to support more efficient and intelligent and effective data-driven investment decision-making. The key data focus of the project will be related to company valuation estimates, metrics of company performance, key exogenous sensitivity factors, and geopolitical indicators (referred collectively as 'stock factors' hereafter).

Specifically, the project includes the selection of data source APIs and database, acquisition of raw data, identification and implementation of influential stock factors, batch processing design and storage of historical data, automated stream processing design and database updates for real-time data, and user-friendly interactive interface. The data orchestration system adopts Kafka distributed messaging middleware, Apache Flink distributed processing framework, and MySQL relational database. By the combination of technologies, the system can leverage the strengths of each component: Kafka's high throughput and reliability in the aspect of data messaging, Apache Flink's real-time data processing capabilities, and MySQL's stability and scalability in the aspect of data storage. And this could help the system overcome the limitations of existing 'off-the-shelf' products such as latency, low-scalability, low-reliability, and so on and empower the system to efficiently process data, conduct real-time analysis, and effectively manage stock fundamentals data, providing a powerful solution for financial market analysis.

1.3 Dissertation Structure

- **Related Works:** This chapter reviews the relevant literature from the conceptual and technical aspects necessary to substantiate the design of the data orchestration framework to be developed. The conceptual part focus on the stock fundamental analysis and relevant influential factors. And the technical part concentrates on distributed data processing frameworks, distributed message oriented middlewares.
- **Methodology:** This chapter provides a clear outline of the processes, methods, and technologies of the project.
- **Implementation:** The chapter provides an in-depth description of the system development process. This encompasses the requirements specification and comprehensive implementation procedure which includes system architecture and designs.
- **Evaluation and Discussion:** This chapter describes the system's execution status and output results. In addition, it discusses the system's efficacy, output results, and other relevant aspects.
- **Conclusions:** This chapter provides a concise overview of the article and the project, while also highlighting potential directions for further research.

Chapter 2

Related Works

2.1 Stock Fundamental Analysis

Stock fundamental analysis is widely recognised as a field of knowledge encompassing principles and established methodologies aimed at evaluating the inherent value of stocks in financial markets[26]. It utilises a comprehensive framework to assess the expected economic factors, thereby identifying sectors which show the potential of increasing in sales and profits. This analysis further enables the evaluation of the financial robustness of companies, the effectiveness of management, and the identification of business prospects based on historical financial statements and prevailing market conditions[26]. Therefore, the process involves assessing the expected fair value of stocks and afterwards comparing them to market values that arise from the interplay of supply (sellers) and demand (buyers) of stocks, facilitating the identification of potential investment opportunities.

According to Efficient Market Hypothesis(EMH)[14], all publicly available information is already fully reflected in market prices. Put simply, the EMH postulates that investors are unable to systematically extract additional information from public sources or from historical stock price movements that is not already impounded in stock prices and that would yield a return greater than the average market return. However, for some markets, particularly emerging markets, the EMH is not entirely effective and stock market prediction models using an array of public information and historical price movements have been shown to produce better results than the average market returns[30]. Moreover, machine learning and artificial intelligence methods have been widely used for accurate prediction of stock market[9], including the researches based on stock fundamental indexes[24, 8, 19]. In the period of data pre-

processing, [24] applied three famous feature selection methods, Principal Component Analysis (PCA), Genetic Algorithms (GA) and decision trees (CART) to select influential factors and developed back-propagation neural network as prediction model. Based on fundamental data, [8] proved that artificial neural networks (ANN) and decision trees perform better on stock price prediction than the hybrid model.
In summary, the literature reviewed in this section underscores the challenge of predicting stock market prices due to market efficiency, wherein much of the public information influencing buy and sell decisions is already incorporated into stock prices. Nonetheless, the literature also highlights that predictive models can yield price forecasts that can support traders achieving above average returns. The key to achieving such results lies in the capacity of predictive models to extract relevant insights from stock market factors.

2.1.1 Stock Fundamental Data

In the current trend dominated by machine learning and artificial intelligence, data preparation and processing plays an important role in the overall stock fundamental analysis[18]. The fundamental analysis employs the company's economic standing, employees, board of directors, financial status, annual report, balance sheets, and income statement, as well as special circumstances such as unnatural or natural disasters and geopolitical or economic data, to analysis and forecast the stock price[18]. Moreover, information from the market environment such as national productivity , inflation rate, foreign currency exchange rate, or interest rates could also be influential in stock fundamental analysis[17]. However, the such data comes from many different sources and feature complex formats and structures, predominantly falling into the categories of semi-structured or unstructured data[13]. Consequently, there exists significant challenges in the data processing.

2.1.2 Stock Fundamental Indicators

Stock factors are the key measures that help to evaluate the economic state of market and financial performance of a company[18]. These factors help investors and financial analysts estimate the intrinsic value of a company's stock, forecast the market trend, and thus make better investment decisions. Here are some typical categories of fundamental factors:

- Valuation

These are the metrics used to evaluate the intrinsic value of a company. They include indexes such as the Price to Earnings (P/E) ratio which comes from multiplier model[26] , Price to Book (P/B) ratio, Earnings per Share (EPS) and so on. These indexes provide insight into how the market values a company relative to its earnings, book value, and many other fundamental influential factors to determine whether a stock of the company is underpriced or overpriced. For example, a higher value of P/E suggests that a stock is overpriced relative to its earnings, but it can also indicate the market's expectation of higher growth in the future.

- Performance

  The performance factors refer to the metrics used to evaluate the company's financial health and market performance, such as share price change, volume, etc. Changes in the share price of a company over time can reveal how the market's perception of the company has evolved. In addition, trading volume is a significant indicator of stock performance that can reveal information about liquidity and investor interest. And sudden surges in volume may be indicative of important news or events affecting the stock price. Based on the analysis on the performance factors, investors and analysts can construct a picture of a stock's past performance and potentially gain insight into its future performance.

- Sensitivities

  The factors are external elements to which a company's performance is particularly sensitive. This can include changes in interest rates, foreign currency exchange rates, or commodity prices. A company's sensitivity to these factors can significantly impact its future profitability and therefore its stock price. For example, for multinational companies, changes in currency exchange rates can have a significant impact on profitability and revenue in the cross-border business involving payments in different currencies.

- Geopolitical

  Geopolitical factors refer to the macroeconomic and geopolitical factors which can include political stability, trade policies, and geopolitical tensions. These factors can impact the broader economy and therefore have an indirect impact on individual companies.

To summarize, stock fundamental analysis is an important part of analysis and forecasting of financial market. However, current researches show high demand and standard for data while fundamental data always has complex structures and comes from a wide range of sources. Therefore, successfully and efficiently processing and generating the important factors of stock fundamentals is the key to accurate stock fundamental analysis.

2.2 Distributed Message Oriented Middleware(MOM)

Message Oriented Middleware(MOM), which consists of a message delivery mechanism or a message queue pattern[10], simplifies and enhances the consistency and reliability of data exchange between systems. MOM enables two or more applications to exchange data packaged as messages in a distributed system, in which Participants are classified as either message producers or message consumers based on their methods of information management[28]. Producers transmit messages to the MOM. Once a message arrives at the middleware, it is placed in a queue based on the message's control information. And then the MOM sends data to consumers via a peer-to-peer or publish-subscribe model. Also, consumers can use the interface to designate the message information they require by sending message queue names or subscription conditions to the MOM. And the typical MOMs are as followed:

- **Message Queuing Telemetry Transport(MQTT):** MQTT[20] is a lightweight, efficient, and reliable protocol that emphasises minimal network bandwidth consumption and reduced memory footprint. It is widely used in Internet of Things (IoT) applications, where it facilitates efficient and stable communication between a large number of devices with limited capabilities. Due to the issue with throughput, MQTT is unsuitable for the transmission of large message payloads.
- **RabbitMQ:** RabbitMQ[25] is an open source messaging system that initially implemented Advanced Message Queuing Protocol (AMQP) and has since been expanded to support Streaming Text Oriented Messaging Protocol (STOMP), MQ Telemetry Transport (MQTT), and other protocols via a plug-in architecture. It is known for its robustness, ease of use, and platform independence, which makes it the preferred option for enterprise-level development.
- **ActiveMQ:** ActiveMQ[12] is a an open-source message queue server which supports various user languages such as Java, C, C++, etc. It is suitable for systems

requiring stable, efficient, and flexibility. Nevertheless, ActiveMQ kernels were developed earlier and the maintenance may not be guaranteed.

- **RocketMQ:** RocketMQ[15] is a widely adopted, high performance, non-logged and reliable distributed messaging system. Considering its high throughput, reliability, and scalability, it is particularly useful in e-commerce transactions, big data, and IoT. However, RocketMQ only offers development interfaces in Java, C++, and Go.

- **ZeroMQ:** ZeroMQ[21] is a high-performance asynchronous messaging library used widely in distributed and real-time scenarios, including finance and the IoT. ZeroMQ has exceptional performance, but its costly development costs and customization requirements make it a less common choice.

- **Kafka:** Kafka[27] is LinkedIn's open-source distributed event streaming platform. The architecture of Kafka enables it to process millions of messages per second, providing the high throughput required to manage real-time data flows. Due to its distributed nature, Kafka can be scaled horizontally by simply adding more hardware. In addition, Kafka's built-in storage system allows it to store vast quantities of data for extended periods, enabling both real-time and batch processing. Furthermore, Kafka's fault-tolerant design prevents messages from being lost due to individual node failures. In conclusion, Apache Kafka's high throughput, scalability, durability, and fault-tolerance make it an ideal choice for distributed messaging middleware in systems requiring the processing of large quantities of real-time data.

In summary, Message Oriented Middlewares, including MQTT, RabbitMQ, ActiveMQ, RocketMQ, ZeroMQ, and Kafka, each possess distinct application scenarios. MQTT is predominantly utilised within the context of the Internet of Things (IoT), RabbitMQ is deemed appropriate for web applications, ActiveMQ is considered suitable for enterprise applications on a large scale, and RocketMQ is specifically tailored to cater to big data scenarios. Conversely, ZeroMQ is better suited for distributed applications that necessitate optimal performance and lowl latency. When considering the real-time data part of the data orchestration system, Kafka is an ideal choice for the middleware due to its ability to offer high throughput, persistence, distributed functionalities, and stream processing capabilities. These features make Kafka particularly well-suited for efficiently managing substantial volumes of real-time stock data.

2.3 Data Stream Processing Framework

Responding to an increasing need for huge amount real-time data processing in the current data-driven world[16], data Stream Processing frameworks enjoy great popularity. To serve the booming demands of streaming data processing, many computation engines have sprung up[11], such as Apache Storm[4], Apache Spark Streaming[3], Apache Flink[2].

- **Apache Storm:** Apache Storm[4] is a distributed real-time computation system for processing high-velocity, high-volume data. It was initially developed by Nathan Marz at BackType and was subsequently open-sourced after being acquired by Twitter[23]. Storm consists of three primary elements: streams, topologies, and nodes. The stream, which is an unbounded sequence of tuples, is the fundamental data processing format. The topology functions as the data graph, and Storm applications consist of topologies which form a graph of data transformations[11]. In the data transformation graph, there are bolts and spouts[23]. Spouts are accountable for receiving data from an external source and sending it to the topology as a stream. Bolts, on the other hand, are responsible for data processing and transformation. The distributed cluster for Storm has two types of nodes: Master Node(Nimbus) and Worker Nodes(Supervisor). Master nodes are responsible for distributing and managing topologies and worker nodes execute the actual topology tasks.

- **Apache Spark Streaming:** Apache Spark Streaming[3] divides the live data stream into micro-batches, which permits the use of the same code for both batch and streaming processing.In the design of large-scale streaming computing systems, error management and straggler processing are two significant concerns. Due to the real-time nature of streaming systems, it is crucial to recoup from errors rapidly. Discretized Stream(DStream), the basic abstraction in Apache Spark Streaming, provides a new recovery mechanism: parallel recovery[29]. In the event of a node failure, the system promptly initiates the reconstruction of the Resilient Distributed Dataset(RDD) of the failed node, using the resources of the other nodes in the cluster. The current recovery mechanism has a higher rate of speed compared to the replication and upstream replay methods of Storm. In all, Apache Spark Streaming is a powerful tool for processing real-time data streams, utilizing the convenience of batch processing, and ensuring high fault tolerance and scalability.

- **Apache Flink:** Apache Flink[2] is an open-source platform for distributed processing and computation, specifically designed for handling large-scale data streams. It supports both batch and streaming processing jobs composed of stateful interconnected tasks[7]. The architecture consists of three main components: JobManager, TaskManager, and Client. JobManager is responsible for coordinating and monitoring the execution of jobs, handling task scheduling, and coordinating fault recovery. TaskManager executes data processing tasks and returns the results to the Job Manager. Client is the user interface for submitting jobs to the JobManager for execution. Flink is a powerful big data stream processing framework that provides robust capabilities for handling large-scale data streams, ensuring high throughput and low latency performance. It is widely used in real-time data analysis like quality monitoring of Telco networks, large-scale event-driven applications like fraud detction, or data pipeline applications like real-time search index building in e-commerce[2].

In conclusion, each of the data stream processing framework possesses its own characteristics. For instance, when the parallelism parameter for the number of processing cores is adjusted, Storm shows a faster processing rate and shorter reaction time[11]. However, it also demonstrates a larger rate of message loss in the event of failures. To choose an appropriate framework, the user should consider the application needs as well as framework characteristics such as data processing rate and fault tolerance.

Chapter 3

Methodology

3.1 Data Collection

For this project, the fundamental stock data is collected from three primary sources: Alpha Vantage[1], Yahoo Finance[6], and Wharton Research Data Service(WRDS)[5].

- **Alpha Vantage**

  Alpha Vantage offers a comprehensive and reliable range of data, including real-time and historical stock market data, foreign exchange rates, commodities prices, technical indicators, and other relevant financial information. Users can access the data by calling the APIs that the company offers.

- **Yahoo Finance**

  Yahoo Finance provides financial news, data, and commentary including stock quotes, press releases, financial reports, and original content. However, in September 2021, the official Yahoo Finance API has been discontinued. The project utilized the yfinance Python package, which allows users to download Yahoo Finance data directly into Python, simplifying the data collection process.

- **Wharton Research Data Service(WRDS)**

  Wharton Research Data Service(WRDS) is a research platform provided by the Wharton School of the University of Pennsylvania. This platform offers diverse financial datasets, catering to users with varying backgrounds. Users can not only retrieve data via a user-friendly web-based interface, but also do data extraction using programming languages such as Python, SAS, and R.

The detailed list of variables retrieved from Alpha Vantage, Yahoo Finance, and WRDS is provided in Appendix ”1. Detailed List of Raw Data”.

3.1.1 Stock Fundamental Factors

The stock fundamental metrics are categorized as four aspects(Table 3.1).

Table 3.1: Category of Stock Fundamental Factors

Category	Meaning
Valuation	Intrinsic value of a company
Performance	Financial health and performance of a company in the market
Sensitivities	External elements to which company’s performance is sensitive
Geopolitical	Macro-economic and geopolitical factors

Subsequently, a comprehensive description of the factors inside each category is provided in the Tables 3.2, 3.3, 3.5, 3.4, 3.6 below. Their respective formulations can be found in Appendix ”2. Formulations of Stock Factors”.

3.2 Data Processing

3.2.1 Kafka Message Stream

Apache Kafka is a distributed, high-throughput publish-subscribe messaging system that supports partitioning, multi-copy, and multi-subscriber[27, 22]. It is extensively used in scenarios including application decoupling, asynchronous processing, flow limiting and peak shaving, and message-driven. The overall architecture of Kafka is shown in Figure 3.1, which helps decouple data pipeline. And here are some important components of Kafka:

Broker: Broker in Kafka cluster is the server node used to store topic data. The higher the number of brokers, the higher the cluster throughput.

Topic: A Topic is a category of messages, similar to a database table name, and messages from different Topics are stored separately on one or more brokers.

Partition: A topic is divided into one or more partitions (at least one), and the more partitions there are, the greater the throughput, but the more resources are required, which may result in greater unavailability.

Table 3.2: Valuation Factors(1)

Factor	**Description**
Market Capitalization	Aggregate value of a company's outstanding shares of stock
Enterprise Value	Total value of a business, including not only equity holders but also debt holders
EPS	Portion of a company's profit allocated to each outstanding share of common stock, acting as a profitability indicator
EV/EBITDA	Valuation metric that compares a company's Enterprise Value (EV) to its Earnings Before Interest, Taxes, Depreciation, and Amortization (EBITDA)
EV/Sales	Valuation metric that relates a company's Enterprise Value (EV) to its total sales or revenue
P/E	Valuation metric that relates a company's current share price to its per-share earnings
PEG	Valuation metric that relates a company's P/E Ratio to its expected earnings growth rate
Price/Sales	Valuation metric that compares a company's market capitalization to its total sales or revenue
Book Value	Net value of a company's assets once all liabilities have been subtracted
Price/Book Value	Valuation metric that relates a company's current share price to its book value per share
Book Value Per Share(BVPS)	Accounting value of a share based on the company's equity available to shareholders
Revenue	Total monetary inflow for a company during a specific period
Cash/Share	The amount of cash and cash equivalents a company holds, relative to its total number of outstanding shares
P / FCF	Valuation metric that relates a company's current share price to its per-share free cash flow
FCF Yield	Valuation metric that relates a company's annual free cash flow to its market capitalization, offering insights into the relative value of a company based on its ability to generate free cash flow
Graham Number	stock's maximum intrinsic value based on its earnings and book value
Total equity/Total liability	How much of company's assets funded by equity versus funded by liabilities

Table 3.3: Valuation Factors(2)

Factor	**Description**
DuPont Analysis	Decomposes Return on Equity (ROE) into its driving components to provide a detailed understanding of a company's performance
Total debt/Capitalization	Proportion of a company's capital that is derived from debt (both short-term and long-term) compared to the total capital (sum of debt and equity)
Debt/EBITDA	Company's ability to pay off its incurred debt
FCF to Sales	Firm's ability to convert sales into cash
Interest Coverage Ratio	Company's ability to meet its interest obligations from its operating earnings
DFL	Sensitivity of a company's earnings per share (EPS) to fluctuations in its operating income as a result of changes in its capital structure
Joel Greenblatts Earnings Yield	Variation of the traditional earnings yield, where instead of using market capitalization, the denominator is the enterprise value
CROIC	Efficiency of a company in turning its invested capital into free cash flow
Piotroski F-score	Metric used to determine the financial strength of a company
Altman's Z-Score	Metric developed by Edward I. Altman in 1968 to predict the likelihood of a publicly traded manufacturing company going bankrupt within the next two years

Table 3.4: Sensitive Factors

Factor	**Discription**
Commodity prices	Market prices for raw materials that are traded on national and international commodity markets
Energy Prices	Prices of different forms of energy
Foreign Currency Exchange Rate	Value of one country's currency in relation to another currency
Interest Rates	Rate a bank offers to its savers or investors
Inflation	Rate of the general level of prices for goods/services is rising, and subsequently, purchasing power is falling

Table 3.5: Performance Factors

Factor	**Description**
Share price change	Market's expectations of a company
Trading volume	Number of shares or contracts traded in a particular security or market during a specific period
52 weeks low	Lowest price at which the stock has traded during the previous 52 weeks
52 weeks high	Highest price at which the stock has traded during the previous 52 weeks
Dividend	Payment made by a corporation to its shareholders
Split	Corporate action that increases the number of shares by dividing its existing shares

Table 3.6: Geopolitical Factors

Factor	**Description**
CBOE Brexit High 50	Performance 50 of UK companies least impacted by Brexit
CBOE Brexit Low 50	Performance of 50 UK companies most affected by Brexit

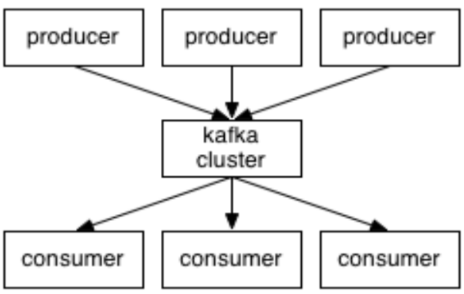


Figure 3.1: Overall Architecture of Kafka

Producer: Producer, data publisher, publishes the message to the relevant topic. The broker containing the topic receives the message and appends it to the segment file. And the user can specify a storage partition for the data.

Consumer: The consumer can read data from the broker and can consume data from multiple topics subscribed. Each consumer belongs to a specific Group.

Some of the outstanding features of kafka make it a good choice for this project. First of all, Kafka acts as an intermediate layer that can receive data from multiple sources and distribute it to multiple consumers without requiring direct connections between sources and consumer, which means that data producers and data consumers(such as Apache Flink) can scale and modify independently without affecting the entire system. Furthermore, Kafka provides data durability so that data will not be lost even in case of system failures. Moreover, there's a good integration and compatibility between Apache Flink and Apache Kafka. Flink could read and write data from and to Kafka with Kafka connector.

The project uses the publish-subscribe mode of Apache Kafka in the real-time data part. In publish-subscribe mode, messages are persisted to the selected topic. Each customer can subscribe to one or more topics and consume all data within the topic, allowing the same data to be consumed by multiple customers without being promptly deleted. According to different data types such as share prices, commodity prices, the publishers publish the required data to the corresponding topics. And the consumer(Flink) pull the data from the topics they subscribed.

To summarize, Kafka offers features such as data buffering, decoupling, fault tolerance, and the capacity to handle large-scale real-time data streams. When combined with Flink, these characteristics enable the creation of a robust and scalable real-time data processing framework.

Also, considering Kafka's principle that data within the same partition is ordered and data across different partitions is unordered, it is important to note that when consuming data from a topic with several partitions, the guarantee of data order cannot be ensured. Hence, in this project, in order to ensure the sequential consumption of real-time data collected, it is necessary to set the partition value to 1 in each topic.

3.2.2 Apache Flink

Apache Flink is a framework and distributed processing engine designed for stateful computation of both unbounded and bounded data streams[2]. Flink could adapt to all common cluster environments and can perform computations at memory speed and at arbitrary scale. Currently, the prevailing streaming computing frameworks available in the market include Apache Storm, Spark Streaming, Apache Flink. However, Apache Flink is the sole framework capable of concurrently supporting low latency, high throughput, and Exactly-Once mechanism.

Flink is a stream processing system that is capable of performing batch processing as well[7]. It achieves this by utilising its streaming computing engine to process batch data, hence achieving a seamless integration of batch processing with stream processing. In contrast to Spark's strategy, Flink treats batch processing as a special instance of stream processing. In the Flink framework, data is generally processed as a stream, which aligns more closely with real-world scenarios. The project uses flink's batch processing mode for historical data part and stream processing mode for real-time data part.

- **Batch Processing for Historical Data**

  Considering the massive quantity of historical stock fundamental data(e.g., the historical data collected for the duration of this project is SP500's financial reports for past 15 years, daily stock data for past 5 years, interest rates for past 70 years, etc.) and the capacity of Flink's batch processing to effectively manage huge data volumes in a single operation, it is appropriate to employ Flink's batch processing mode for the historical data part. Raw data collected from data sources undergoes pre-processing before being transferred to the batch processing environment of Flink. Temporary tables are then created by employing Flink's Table APIs to receive the data. In addition, Flink offers some basic functions like *map()* and customized methods to facilitate further data processing tasks in the Flink environment[2]. The ultimate results are temporarily held for the step of output or storage.

- **Stream Processing for Real-time Data**

  In the real-time data section, Apache Flink builds a connection with Apache Kafka through the Kafka Connector and reads the latest data from the Kafka topics, which are continuously updated at specific intervals. Due to the presence of duplicate and irrelevant information, as well as historical data that has already existed in the database, it is necessary to preprocess real-time data in order to ensure its suitability and effectiveness. The preprocessing stage encompasses many tasks including as data purification, data format conversion, and feature extraction, all aimed at preparing the data for the next step. After that, the data would be sent to the Flink stream processing environment for computational or analytical procedures. Furthermore, Apache Flink offers a diverse selection of pre-existing connectors designed for different types of sinks, including databases, filesystems, and messaging queues. Consequently, it is a straightforward process to direct the final results to the desired sink.

3.3 Data Storage and Updating

Data storage and updating uses MySQL, an open-source relational database management system(RDBMS) that uses Structured Query Language(SQL) to add, access, and manage the contents of a database. MySQL offers high-performance database interactions and can efficiently manage large volumes of data. Furthermore, ACID-compliant(Atomicity, Consistency, Isolation, Durability) of MySQL ensures that all transactions are processed reliably, and in the event of a system failure, the database could recover to a consistent state. Also, MySQL is easy to integrate with various software and platforms, including Apache Flink, Visual Studio Code. Due to its robust features, high reliability, and excellent performance, it is a good choice for the data orchestration system in connection to stock fundamentals.

Chapter 4

Implementation

This section provides an in-depth description of the requirements specification, environment prerequisites, architecture, and implementation details of the project. And the detailed code content can be found in the appendix("3. Data Collection and Pre-Processing", "4. Batch Processing and Storage", "5. Real-time Stream Processing and Storage").

4.1 Requirements Specification

The data orchestration system is specifically created to have the capability of obtaining, updating in real-time, processing, formatting, and storing data with a primary emphasis on firm Valuation, Performance, Sensitivities, and Geopolitics. As a result, the system can autonomously produce the required data set and serve as the upstream component of stock trading systems, thereby enhancing the efficacy and effectiveness of data-driven investment decision-making. The following are the requirements for the system.

4.1.1 Functional Requirements

1. **Data Collection**

   The system should be able to collect data of various sources, including companies' balance sheets, income statements, statements of cash flow, real-time stock data, commodity and energy prices, interest rates and so on.

   The system should be able to use APIs of Alpha Vantage, Yahoo Finance, and Wharton Research Data Service(WRDS) to fetch the accurate and sufficient historical and real-time data, which will be used as inputs to the subsequent stages of the system..

2. **Data Processing**

   The system should possess the capability to perform pre-processing and filtering on the raw data in order to ensure that the data meets the input criteria of the metric functions, if the raw data is used as an intermediate operand. Alternatively, the system should ensure that the raw data is transformed into its final format.

   The system should be able to generate the influential fundamental factors by computation or prediction on the basic elements from raw data.

   The system should be able to realize the automatic process of processing real-time data with the combination the kafka and flink.

3. **Data Storage and Updating**

   The system should be able to store all the data in a scalable and reliable data storage system and handle the data in the database.
   The system should be able to automatically update the content of the database to ensure the data is latest.

4. **User Interface**

   The system should provide easy user interfaces for both the downstream of stock trading system and analysts. Users can have access to the datasets(Json/CSV) via APIs which is designed with the parameters based on the desired data categories, companies, time periods, and factor names. The flexible combination of parameters allows the user to access the datasets according to their unique needs.

4.1.2 Non-functional Requirements

1. **Performance**: The system should be able to handle a high volume of incoming data and provide low latency processing.

2. **Scalability**: The system should be capable of scaling up to handle increased workloads. Also, the system could be easy to add more factors or functions if needed.

3. **Security**: The system should be able to ensure data privacy and confidentiality.

4. **Usability**: The interfaces created of the system should be user-friendly, intuitive, and accessible.

4.1.3 Constraints

1. **Budget**: The system ought to be designed and executed within the designated budgetary constraints, such as the exclusion of paid resources and reliance solely on resources provided by the university.
2. **Technology Stack**: The system should be built on open-source software or framework.
3. **Data Source**: The system should fetch the data from public data source or use data source APIs with official permission.

4.2 Data Flow Model

The project collects raw data on stock fundamentals from data sources via APIs and separates it into historical and real-time data segments. For historical data, the final result is stored in the database after batch processing. For the real-time data part, real-time data is subscribed via Kafka, then delivered into the Flink environment for stream processing, and then the database contents are updated. Finally, the user can acquire the data sets required for various later stock trading analysis via the system's interactive interface. The data flow model is shown as Figure 4.1.

4.3 Environmental Prerequisites

The project is developed by Python and utilises the technologies like kafka, Apache-flink, and MySQL. The Table 4.1 shows the specific environment prerequisites.

Table 4.1: Environmental Prerequisites

Environmental Prerequisites
Python 3.9.16
open-jdk 11.0.13
MySql 8.0.33
kafka_2.12-3.5.0
apache-flink 1.17.1

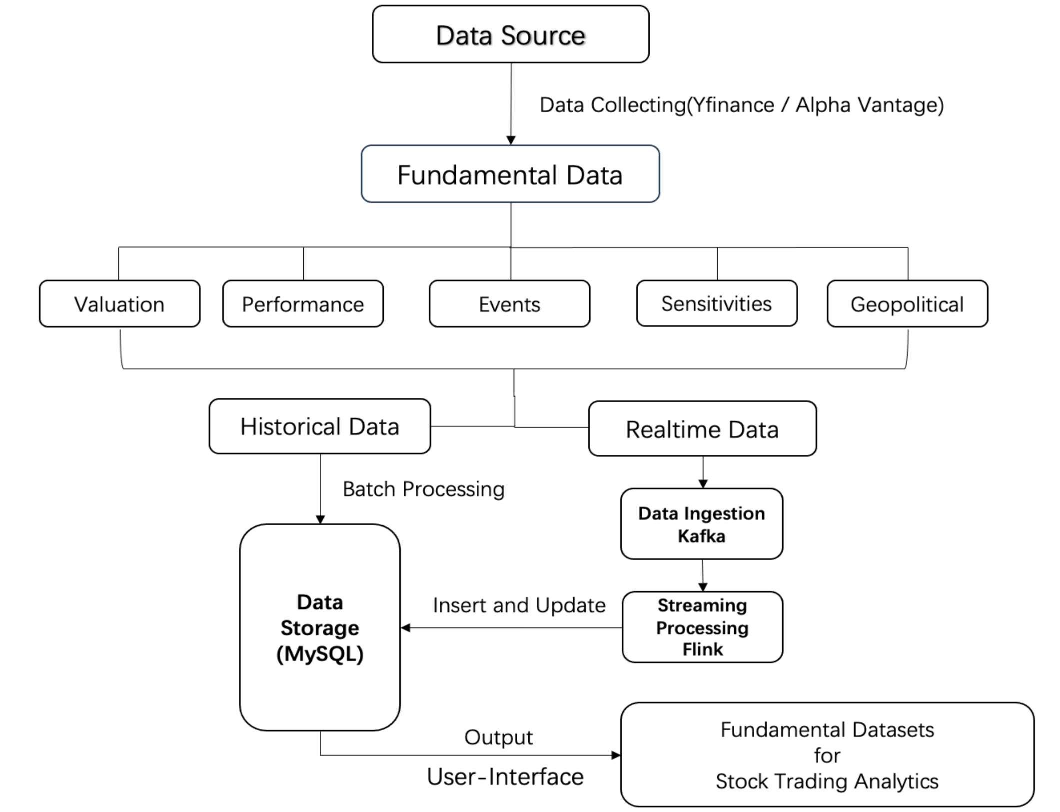


Figure 4.1: Data Flow Model

4.4 Data Collection

The project employs the APIs of Alpha Vantage, yfinance and WRDS to gather stock fundamental data, ensuring sufficient coverage and complementarity of data.

Alpha Vantage is a provider of APIs for historical and real-time financial data of various types. The project uses the APIs to get historical and real-time data like financial reports of companies that compose the S&P500 index, commodity prices, foreign exchange, inflation rates, interest rates and some other required data in the format of Json. The API provided by Alpha Vantage allows for customization like functions, intervals, and other relevant parameters according to the requirements(Figure 4.2).

```python
#collect data from alpha vantage
def fetch_data_from_alpha(function, index):
    url = f'https://www.alphavantage.co/query?function={function}&symbol={index}&apikey=740YDHNMV1Q7Y2A9'
    r = requests.get(url,verify=False)
    data = r.json()
    return data
```

Figure 4.2: API of Alpha Vantage

Yfinance is a Python library that provides users with the capability to retrieve financial data from Yahoo Finance. This includes a wide range of information such as stocks, cryptocurrencies, currencies, options, commodities, and other relevant data. This system uses APIs of yfinance to get historical and real-time stock data, financial reports of S&P500. The Figure 4.3 is an example of collecting historical stock data by yfinance API.

```
#get S&P500 index list
sp500_components = pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')[0]
symbols = sp500_components['Symbol'].tolist()
#fetch historic stock data for 3y
for symbol in symbols:
    stock = yf.Ticker(symbol)
    try:
        history = stock.history(period='3y')
    except Exception as e:
        print(f"Failed to get stock price for {symbol}: {e}")
        continue
    else:
        if not history.empty:
            history['Symbol'] = symbol
```

Figure 4.3: API of Yfinance

However, the fundamental data provided by Alpha Vantage and yfinance is not sufficient for the fundamental analysis. The time range available in these APIs for financial reports of S&P500 is only 5 years, which can limit the training capabilities of most machine learning models and trading simulations typically employed in stock trading systems(e.g., 5 years of stock fundamentals that are published on a quarterly basis would provide 20 data points per company). As a result, the project utilizes WRDS as a main source of historical financial reports which provides past annual financial reports for about 15 years. And the financial reports from the other sources serve as the main raw data of real-time part

Given that the valuation part requires various basic items from the financial reports and the data queries are complex, data extraction is done by Python(Figure 4.4).

```
db = wrds.Connection()
def wrds_annual_hist(symbol):
    annual_report_query = f"""
        SELECT tic, datadate, ebit,ebitda,at, act, ivst, lt, lct, dltt, teq, tstk, re, csho, cshi, dt, ceq,seq,cstk,chech,icapt,xint,idit,xt,
        epsfi,epspx,ni,oiadp,xoprar,cogs,revt,capx,fincf,ivncf,oancf
        FROM comp.funda
        WHERE tic = '{symbol}'
        AND indfmt='INDL'
        AND datafmt='STD'
        AND popsrc='D'
        AND consol='C'
        AND datadate >= '2008-01-01'

    """
    annual_report_data = db.raw_sql(annual_report_query)
```

Figure 4.4: API of WRDS

And the raw data collected from the data sources(Table 4.2) includes:

Table 4.2: Raw Data List

Alpha Vantage	Yahoo Finance	WRDS
Income Statement	Income Statement	Income Statement
Balance Sheet	Balance Sheet	Balance Sheet
Cash Flow	Cash Flow	Cash Flow
FX Exchange Rates	Stock Data	
Commodity Price	CBOE Brexit	
Energy Price		
Inflation		
Interest Rate		

4.5 Historical Data

For the historical data part, the types of the raw data crawled, intervals, and their corresponding time ranges are shown in the table 4.3 below.

Table 4.3: Historical Data (Type, Interval, Time Range)

Type	Interval	Time Range
Financial Reports	Yearly	Past 15 years
Inflation	Yearly	Past 64 years
Financial Reports	Quarterly	Past 5 quarters
FX Exchange Rates	Monthly	Past 20 years
Commodity/Energy Price	Monthly	Past 38 years
Interest Rate	Monthly	Past 70 years
Stock Data	Daily	Past 5 years
Geopolitical Data	Daily	Past 5 years

Prior to sending the data set into a batch processing environment, it is imperative to do basic pre-processing on the raw data.

1. Data Cleaning: For certain instances of the S&P500 symbol list, it is possible that there can be a lack of data that meets the specified criteria. Under these circumstances, the project chooses to assign a value of 0 to the rows and columns that have missing values.

2. Format Conversion: The data obtained from Alpha Vantage is in the Json format.

In order to ensure consistency in formatting, it is necessary to transform the data into a standardised format of Dataframe, which can make the further processing more easier.

3. Data Filtering and Aggregation: It is necessary to perform a filtering procedure in data pre-processing. Useless or replicated items are removed while preserving the important components. Afterwards, according to the project target, some distinct datasets are merged and combined in order to facilitate the subsequent phases of data processing. For example, for financial reports, the essential elements necessary for generating fundamental valuation metrics in the following phase are retained, while extraneous components are eliminated. Furthermore, it is important to acknowledge that financial reports consist of a balance sheet, income statement, and statement of cash flow, which are derived from three distinct data sources. Therefore, the process of aggregation is a crucial step that cannot be disregarded.

4.5.1 Batch Processing by pyFlink

Given the significant amount of historical data, it is preferable to do the main processing operation within the Flink framework. The initial stage in batch processing is the selection of the execution mode, which enables the configuration of the settings for the batch processing operation. By implementing this functionality, Flink could acknowledge the limited scope of the dataset it is manipulating, thereby facilitating the use of optimization strategies that are specifically tailored for datasets with restricted boundaries. Subsequently, a Table Environment is instantiated, serving as the main mechanism for accessing the functions of the Table API. The implementation of this phase would guarantee that all operations are in accordance with batch processing techniques. Additionally, the Table Environment offers an interface to set the configurations, including parallelism, the number of task slots, memory allocation, and other related parameters. The typical example of code snippet is shown in Fig 4.5.

```
from pyflink.table import EnvironmentSettings, TableEnvironment

env_settings = EnvironmentSettings.in_batch_mode()
table_env = TableEnvironment.create(env_settings)
table_env.get_config().get_configuration().set_integer('parallelism',4)
```

Figure 4.5: Flink Batch Mode Environment

Once the environment has been initialized, the subsequent step involves the processing of data. Most factors in the aspect valuation need to be calculated and generated according to their respective formulas while Flink's inbuilt operations are not able to meet these needs. However, Flink provides extensibility by allowing users to define their own functions according to their needs through User-defined functions(UDF). Through UDF, data processing can be conducted more flexible in the environment of the Flink. Using the generation of the Graham Number as an illustrative example(Figure 4.6), the entire procedure includes the definition, registration, and application stages. UDFs can only be utilised in subsequent SQL queries or DataStream/-DataSet APIs once the registration process has been completed.

```
@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),DataTypes.DOUBLE(),DataTypes.DOUBLE()],
     result_type=DataTypes.DOUBLE())
def an_Graham_basic(totalAsset,totalLib,shareOutstanding,basiceps):
    if totalAsset is None or totalLib is None or shareOutstanding is None or basiceps is None or shareOutstanding==0.0:
        ratio = 0.0
    else:
        bv = totalAsset - totalLib
        if (22.5 * (bv/shareOutstanding ) * basiceps)>0.0:
            ratio = np.sqrt(22.5 * (bv/shareOutstanding ) * basiceps)
        else:
            ratio = 0.0
    return ratio
table_env.register_function('GrahamBasic',an_Graham_basic)
```

Figure 4.6: UDF of Graham Number

Additionally, in the Flink environment, the final outcome should be stored in a temporary sink prior to its output. And the schema of the sink should be strictly consistent with that of the result query.

4.6 Real-time Data

4.6.1 Timing Mechanism for Updating

Considering the differing frequency of updates for different data types, the project employs a scheduling mechanism(***schedule***) to make certain functions to execute at specified times (*.at()* method) or at regular intervals (*.every()* method). For financial reporting, the update date is consistently positioned at the end of each quarter and year. This arrangement allows for the convenient scheduling of automated data collecting tasks, which can be scheduled at the initial day of each quarter and year. In the case of stock data, the update frequency is significantly higher, necessitating a daily execution interval. Furthermore, it is crucial to acknowledge that ***schedule*** only executes the predetermined jobs upon explicit invocation of the *run_pending()* function. Hence, it is imperative to realize an automation process by incorporating a loop design with a sleep mechanism to mitigate the program's excessive frequency in checking for pending tasks.

4.6.2 Data Ingestion into Kakfa

To ensure efficient, orderly, and reliable collection of a significant amount of real-time financial data, as well as to establish a stable buffer and persistence for the data, the real-time part of the project utilises the Kafka messaging middleware to subscribe and deliver newly updated data in real time.

Given the variety of data sources and types involved in this project, it is crucial to build different Kafka topics based on the data types and storage architecture. This approach will ensure more efficient listening and subsequent processing for real-time data. Once the topics have been set up, the primary task is to write the real-time messages obtained from the data sources into the Kafka topics. The first step of the task is to define the type information(*type_info()*) based on the nature and structure of the input data. And then the Flink stream execution environment generates a data stream from the input data set and converting the data into the Json format. This serialisation process facilitates the transfer and storage of the data by enhancing its accessibility and compatibility. Subsequently, the data stream is written to Kafka by choosing the correspondig topic, serialization setting, and other relevant configuration details through the Flink connector(*FlinkKafkaProducer*) intended for connecting with Kafka.

4.6.3 Stream Processing by pyFlink

The initial stage in Flink's stream processing framework is consuming the real-time data from the Kafka messaging platform. The starting point is to create the deserialization schema and constructing the deserialization builder, which enables the conversion of the serialized format into a data structure that can be manipulated by Flink. Subsequently, it is necessary to initialize the Kafka consumer(*FlinkKafkaConsumer*) in Flink, wherein the topic of the data to be retrieved, the deserialization pattern, and the related configuration are specified. Once the configured consumer is integrated into Flink's stream processing execution environment, the real-time data in Kafka could be seamlessly consumed by Flink. Moreover, real-time data is substantially less than historical data, thus processing it before sending it to Kafka or reading and then processing it in Flinks by techniques similar to those in batch mode has a similar effect. The data processing in this project employs the first method, wherein the data is processed prior to being transmitted to kafka. Consequently, subsequent to the consumption of data from Kafka, Flink could efficiently store and output real-time data through straightforward processes.

By concurrently executing the tasks of subscribing data in Kafka and consuming and processing data in Flink on the Flink's server, it becomes feasible to automate the process of obtaining, processing, and storing real-time data.

4.7 Logging and Fault Tolerance Mechanism

Given the special nature of financial data, even after carefully selecting reliable data sources, the data we acquire from the sources frequently has defaults or is not available for specific symbol. Furthermore, the values of the same factors may vary due to the adoption of different computational and quantitative criteria by different data sources and firms. Hence, in situations involving missing data, the implementation of a simple substitution of information from other sources of data is not feasible. In general, when dealing with a significant quantity of data, the significance of fault tolerance and logging systems cannot be overstated. These techniques play a crucial role in error management and ensuring the uninterrupted execution of the script.

Logging Mechanism: The present project employs module *logging* combined with the function *print()* as a logging mechanism, which is extremely effective in the stages of development and debugging. This approach enhances logging context by providing additional details and enables increased flexibility in terms of output options and more precise control.

Fault Tolerance Mechanism: (1) Exception Handling: In the process of data acquisition and processing, *try-expect* statements is used to mitigate the risk of a complete system failure resulting from a singular error occurrence. This is especially important given the substantial volume of data and the long process of the procedure. Hence, during the acquisition or processing of data, it is imperative to promptly catch and throw faults or exceptions, such as the incapacity of a single object to acquire data or exceptions related to data format, in order to ensure the smooth running of the programme. (2) Integrity of Data and Enforceability of Calculations: The project has the mechanism whereby the default value is set to zero. Furthermore, given the metric calculations play an important role in the project, conditional judgements are commonly employed to prevent the circumstances such as an empty dataset, negative values under the square root, zero denominators. (3) Track Record of Tasks Performed: Considering the intricacy and huge volume of the data and the diversity of tasks, I implemented a procedure of tracking and recording executed tasks. This approach aimed to guarantee the successful completion of all data acquisition and task execution, hence mitigating the occurrence of duplicate or missing tasks.

4.8 Database Design and Storage

The design of the tables in the MySQL database for this project is based on the categories of fundamental factors as well as the elements and structure of different kinds of datasets.

Valuation: The factors of valuation include fundamental variables and valuation metrics. The fundamental variables are derived from the quarterly and yearly financial reports of the S&P 500 index over a span of 15 years. The metrics are obtained through the following utilisation of computations and operations on the fundamental variables. The design of the table for the valuation factors is shown in Figure 4.7.

Performance: The factors of performance are mainly the daily stock data of S&P500. The design of the table for performance factors is shown in Figure 4.8.

Sensitivities: Sensitivities include multiple factors such as commodities prices, energy prices, inflation rates, interest rates, and foreign currency exchange rates. Nevertheless, the structure of foreign currency exchange rates diverges from other factors. Consequently, two distinct tables have been designed to accommodate the sensitive factors. The design for foreign currency exchange rates is depicted in Figure 4.9, whereas the design for other factors(commodities prices, energy prices, inflation rates, interest rates) is displayed in Figure 4.10.

Geopolitical: Geopolitical factors are composed of CBOE Brexit indexes. Based on the structure of the factors, the design of the table is shown as Figure 4.11.

In the context of data storage, Flink, in both its batch and stream processing environments, establishes a connection with the database by utilising a connector that facilitates interaction between Flink and JDBC. The final outcome is preserved within the temporary table provided by Flink. It is imperative that the structure of the temporary table closely matches with the corresponding table in the receiving database. Subsequently, the data is stored in the database by manipulating the temporary table using *SQL* statements in Flink.

mysql> DESC valuationTotal;

Field	Type	Null	Key	Default	Extra
symbol	varchar(10)	NO	PRI	NULL	
fiscalDateEnding	timestamp(6)	NO	PRI	NULL	
interval	varchar(10)	NO	PRI	NULL	
ebit	double	YES		NULL	
ebitda	double	YES		NULL	
totalAssets	double	YES		NULL	
totalCurrentAssets	double	YES		NULL	
shortTermInvestments	double	YES		NULL	
totalLiabilities	double	YES		NULL	
totalCurrentLiabilities	double	YES		NULL	
longTermDebt	double	YES		NULL	
totalShareholderEquity	double	YES		NULL	
treasuryStock	double	YES		NULL	
retainedEarnings	double	YES		NULL	
commonStockSharesOutstanding	double	YES		NULL	
ShareIssued	double	YES		NULL	
TotalDebt	double	YES		NULL	
CommonStockEquity	double	YES		NULL	
StockholdersEquity	double	YES		NULL	
CommonStock	double	YES		NULL	
CashAndCashEquivalents	double	YES		NULL	
InvestedCapital	double	YES		NULL	
InterestExpense	double	YES		NULL	
InterestIncome	double	YES		NULL	
TotalExpenses	double	YES		NULL	
DilutedEPS	double	YES		NULL	
BasicEPS	double	YES		NULL	
NetIncome	double	YES		NULL	
OperatingIncome	double	YES		NULL	
OperatingExpense	double	YES		NULL	
CostOfRevenue	double	YES		NULL	
TotalRevenue	double	YES		NULL	
FreeCashFlow	double	YES		NULL	
FinancingCashFlow	double	YES		NULL	
InvestingCashFlow	double	YES		NULL	
OperatingCashFlow	double	YES		NULL	
currentClosePrice	double	YES		NULL	
pToEDiluted	double	YES		NULL	
pToEBasic	double	YES		NULL	
DilutedPEG	double	YES		NULL	
BasicPEG	double	YES		NULL	
revenueGrowth	double	YES		NULL	
piotroskiFscore	bigint	YES		NULL	
evToEbitda	double	YES		NULL	
enterpriseValue	double	YES		NULL	
marketCaptation	double	YES		NULL	
evToSales	double	YES		NULL	
priceToSales	double	YES		NULL	
bv	double	YES		NULL	
priceToBv	double	YES		NULL	
bvToShare	double	YES		NULL	
cashToShare	double	YES		NULL	
priceToFCF	double	YES		NULL	
FCFYield	double	YES		NULL	
GrahamBasic	double	YES		NULL	
GrahamDiluted	double	YES		NULL	
totalEquityToTotalAsset	double	YES		NULL	
Dupont	double	YES		NULL	
debtToCapital	double	YES		NULL	
DFL	double	YES		NULL	
debtToEbitda	double	YES		NULL	
InterestCoverageRatio	double	YES		NULL	
FCFToSales	double	YES		NULL	
altmanZscore	double	YES		NULL	
JoelGreenblattsEarningsYield	double	YES		NULL	
croic	double	YES		NULL	

66 rows in set (0.01 sec)

Figure 4.7: Design of Valuation Table

```
mysql> DESC stock_price;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Date        | varchar(15) | NO   | PRI | NULL    |       |
| Symbol      | varchar(10) | NO   | PRI | NULL    |       |
| Open        | double      | YES  |     | NULL    |       |
| High        | double      | YES  |     | NULL    |       |
| Low         | double      | YES  |     | NULL    |       |
| Close       | double      | YES  |     | NULL    |       |
| Volume      | bigint      | YES  |     | NULL    |       |
| Dividends   | double      | YES  |     | NULL    |       |
| StockSplits | double      | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

Figure 4.8: Design of Performance Table

```
mysql> DESC fx_monthly;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Date       | varchar(15) | NO   | PRI | NULL    |       |
| FromSymbol | varchar(8)  | NO   | PRI | NULL    |       |
| ToSymbol   | varchar(8)  | NO   | PRI | NULL    |       |
| Open       | double      | YES  |     | NULL    |       |
| High       | double      | YES  |     | NULL    |       |
| Low        | double      | YES  |     | NULL    |       |
| Close      | double      | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

Figure 4.9: Design of Fx Table

```
mysql> DESC sensitivities;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| name     | varchar(40) | NO   | PRI | NULL    |       |
| interval | varchar(10) | YES  |     | NULL    |       |
| unit     | varchar(40) | YES  |     | NULL    |       |
| date     | varchar(15) | NO   | PRI | NULL    |       |
| value    | double      | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

Figure 4.10: Design of Sensitivities Table

```
mysql> DESC geopolitical;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| Date  | varchar(15) | NO   | PRI | NULL    |       |
| Name  | varchar(20) | NO   | PRI | NULL    |       |
| Open  | double      | YES  |     | NULL    |       |
| High  | double      | YES  |     | NULL    |       |
| Low   | double      | YES  |     | NULL    |       |
| Close | double      | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Figure 4.11: Design of Geopolitical Table

Chapter 5

Evaluation & Discussion

5.1 Execution and Output

5.1.1 Data Acquisition

The project collected data on corporate valuations, performance, sensitivity, and geopolitical factors from three reliable data sources, namely Alpha Vantage, Yahoo Finance, and WRDS. Figure 5.1 depicts a representative sample of the actual data collected(Symbol: IBM, Data Type: Financial Report). Upon conducting the data quality assessments, it has been concluded that the data exhibits a notable level of accuracy and completeness, while also being regularly updated. The content of the raw dataset for each category is list as follows:

Valuation: quarterly and yearly financial reports of S&P500

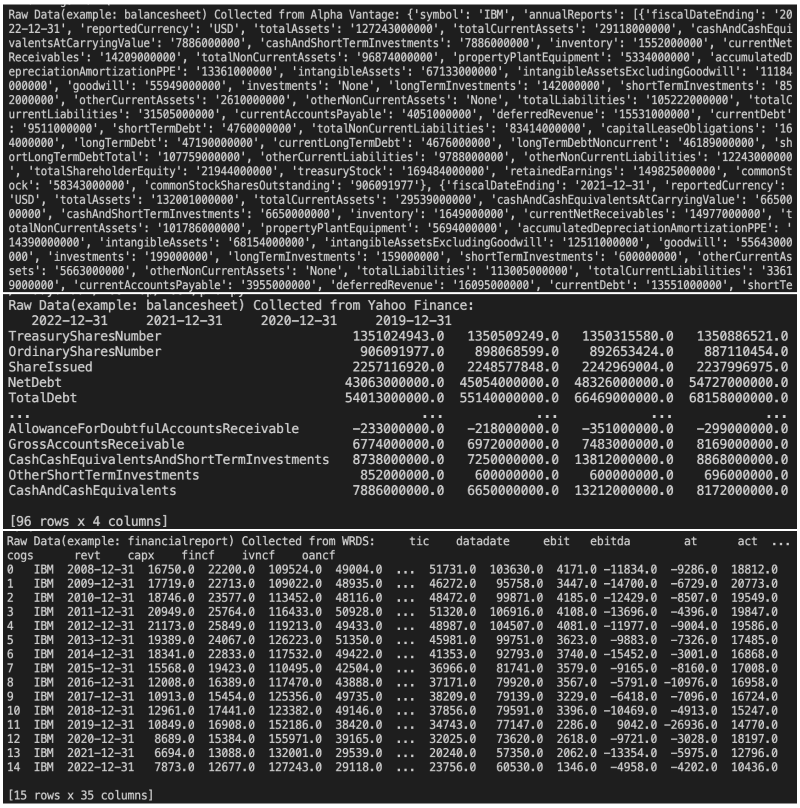
Performance: stock data of S&P500

Sensitivities: Commodity/Energy Prices(West Texas Intermediate(WTI) crude oil, Brent crude oil, natural gas, copper, aluminum, wheat, corn, cotton, sugar, coffee, global price index of all commodities), interest rates, inflation rates, foreign currency exchange rates(*EUR to USD, GBP to USD, USD to JPY, AUD to USD, USD to CAD, USD to CHF, NZD to USD, GBP to EUR, USD to CNY, EUR to JPY*)

Geopolitical: CBOE Brexit Low 50 index and CBOE Brexit High 50 index

5.1.2 Batch Processing and Storage of Historical Data

The process of batch processing and storage of historical data involves three main components: pre-processing of raw data , data processing in the Flink's batch environment of Flink, and data storage. The efficiency of the execution of the whole process is




```
Raw Data(example: balancesheet) Collected from Alpha Vantage: {'symbol': 'IBM', 'annualReports': [{'fiscalDateEnding': '20
22-12-31', 'reportedCurrency': 'USD', 'totalAssets': '127243000000', 'totalCurrentAssets': '29118000000', 'cashAndCashEqui
valentsAtCarryingValue': '7886000000', 'cashAndShortTermInvestments': '7886000000', 'inventory': '1552000000', 'currentNet
Receivables': '14209000000', 'totalNonCurrentAssets': '96874000000', 'propertyPlantEquipment': '5334000000', 'accumulatedD
epreciationAmortizationPPE': '13361000000', 'intangibleAssets': '67133000000', 'intangibleAssetsExcludingGoodwill': '11184
000000', 'goodwill': '55949000000', 'investments': 'None', 'longTermInvestments': '142000000', 'shortTermInvestments': '85
2000000', 'otherCurrentAssets': '2610000000', 'otherNonCurrentAssets': 'None', 'totalLiabilities': '105222000000', 'totalC
urrentLiabilities': '31505000000', 'currentAccountsPayable': '4051000000', 'deferredRevenue': '15531000000', 'currentDebt'
: '9511000000', 'shortTermDebt': '4760000000', 'totalNonCurrentLiabilities': '83414000000', 'capitalLeaseObligations': '16
4000000', 'longTermDebt': '47190000000', 'currentLongTermDebt': '4676000000', 'longTermDebtNoncurrent': '46189000000', 'sh
ortLongTermDebtTotal': '107759000000', 'otherCurrentLiabilities': '9788000000', 'otherNonCurrentLiabilities': '12243000000
', 'totalShareholderEquity': '21944000000', 'treasuryStock': '169484000000', 'retainedEarnings': '149825000000', 'commonSt
ock': '58343000000', 'commonStockSharesOutstanding': '906091977'}, {'fiscalDateEnding': '2021-12-31', 'reportedCurrency':
'USD', 'totalAssets': '132001000000', 'totalCurrentAssets': '29539000000', 'cashAndCashEquivalentsAtCarryingValue': '66500
00000', 'cashAndShortTermInvestments': '6650000000', 'inventory': '1649000000', 'currentNetReceivables': '14977000000', 't
otalNonCurrentAssets': '101786000000', 'propertyPlantEquipment': '5694000000', 'accumulatedDepreciationAmortizationPPE': '
14390000000', 'intangibleAssets': '68154000000', 'intangibleAssetsExcludingGoodwill': '12511000000', 'goodwill': '55643000
000', 'investments': '199000000', 'longTermInvestments': '159000000', 'shortTermInvestments': '600000000', 'otherCurrentAs
sets': '5663000000', 'otherNonCurrentAssets': 'None', 'totalLiabilities': '113005000000', 'totalCurrentLiabilities': '3361
9000000', 'currentAccountsPayable': '3955000000', 'deferredRevenue': '16095000000', 'currentDebt': '13551000000', 'shortTe
```

```
Raw Data(example: balancesheet) Collected from Yahoo Finance:
   2022-12-31     2021-12-31     2020-12-31     2019-12-31
TreasurySharesNumber                        1351024943.0   1350509249.0   1350315580.0   1350886521.0
OrdinarySharesNumber                         906091977.0    898068599.0    892653424.0    887110454.0
ShareIssued                                 2257116920.0   2248577848.0   2242969004.0   2237996975.0
NetDebt                                    43063000000.0  45054000000.0  48326000000.0  54727000000.0
TotalDebt                                  54013000000.0  55140000000.0  66469000000.0  68158000000.0
...                                                  ...            ...            ...            ...
AllowanceForDoubtfulAccountsReceivable      -233000000.0   -218000000.0   -351000000.0   -299000000.0
GrossAccountsReceivable                     6774000000.0   6972000000.0   7483000000.0   8169000000.0
CashCashEquivalentsAndShortTermInvestments  8738000000.0   7250000000.0  13812000000.0   8868000000.0
OtherShortTermInvestments                    852000000.0    600000000.0    600000000.0    696000000.0
CashAndCashEquivalents                      7886000000.0   6650000000.0  13212000000.0   8172000000.0

[96 rows x 4 columns]
```

```
Raw Data(example: financialreport) Collected from WRDS:     tic    datadate     ebit   ebitda        at       act  ...
cogs       revt     capx     fincf     ivncf     oancf
0   IBM  2008-12-31  16750.0  22200.0  109524.0  49004.0  ...  51731.0  103630.0  4171.0 -11834.0  -9286.0  18812.0
1   IBM  2009-12-31  17719.0  22713.0  109022.0  48935.0  ...  46272.0   95758.0  3447.0 -14700.0  -6729.0  20773.0
2   IBM  2010-12-31  18746.0  23577.0  113452.0  48116.0  ...  48472.0   99871.0  4185.0 -12429.0  -8507.0  19549.0
3   IBM  2011-12-31  20949.0  25764.0  116433.0  50928.0  ...  51320.0  106916.0  4108.0 -13696.0  -4396.0  19847.0
4   IBM  2012-12-31  21173.0  25849.0  119213.0  49433.0  ...  48987.0  104507.0  4081.0 -11977.0  -9004.0  19586.0
5   IBM  2013-12-31  19389.0  24067.0  126223.0  51350.0  ...  45981.0   99751.0  3623.0  -9883.0  -7326.0  17485.0
6   IBM  2014-12-31  18341.0  22833.0  117532.0  49422.0  ...  41353.0   92793.0  3740.0 -15452.0  -3001.0  16868.0
7   IBM  2015-12-31  15568.0  19423.0  110495.0  42504.0  ...  36966.0   81741.0  3579.0  -9165.0  -8160.0  17008.0
8   IBM  2016-12-31  12008.0  16389.0  117470.0  43888.0  ...  37171.0   79920.0  3567.0  -5791.0 -10976.0  16958.0
9   IBM  2017-12-31  10913.0  15454.0  125356.0  49735.0  ...  38209.0   79139.0  3229.0  -6418.0  -7096.0  16724.0
10  IBM  2018-12-31  12961.0  17441.0  123382.0  49146.0  ...  37856.0   79591.0  3396.0 -10469.0  -4913.0  15247.0
11  IBM  2019-12-31  10849.0  16908.0  152186.0  38420.0  ...  34743.0   77147.0  2286.0   9042.0 -26936.0  14770.0
12  IBM  2020-12-31   8689.0  15384.0  155971.0  39165.0  ...  32025.0   73620.0  2618.0  -9721.0  -3028.0  18197.0
13  IBM  2021-12-31   6694.0  13088.0  132001.0  29539.0  ...  20240.0   57350.0  2062.0 -13354.0  -5975.0  12796.0
14  IBM  2022-12-31   7873.0  12677.0  127243.0  29118.0  ...  23756.0   60530.0  1346.0  -4958.0  -4202.0  10436.0

[15 rows x 35 columns]
```


Figure 5.1: Raw Data from Data Sources

shown in Table5.1, which reveals gaps between the actual data volume and the anticipated data volume. These differences arise from the unavailability of data for some companies or specific time periods. And the throughput of the valuation factors is quite poor in comparison to that of others, mostly because of the extensive computation of numerous metrics during data processing. Moreover, the final form of the data saved in the database is illustrated by the instances depicted in Figure 5.2, 5.3, 5.4, 5.5.

Data Type	Target	Actual	Throughput
Valuation	10000rows	9684rows	1.24rows/s
Performance	630000rows	613158rows	466.3rows/s
Sensitivities	8560rows	8440rows	109rows/s
Geopolitical	2350rows	2348rows	242row/s

Table 5.1: Efficiency of Historical Data Part (Approximately)

```
                    symbol: IBM
          fiscalDateEnding: 2009-12-31 00:00:00.000000
                  interval: annual
                      ebit: 17719000000
                    ebitda: 22713000000
               totalAssets: 109022000000
        totalCurrentAssets: 48935000000
      shortTermInvestments: 1791000000
          totalLiabilities: 86267000000
   totalCurrentLiabilities: 36002000000
              longTermDebt: 21932000000
    totalShareholderEquity: 22755000000
             treasuryStock: 81243000000
          retainedEarnings: 62070000000
commonStockSharesOutstanding: 1305337000
               ShareIssued: 2127016999.9999998
                 TotalDebt: 24154000000
         CommonStockEquity: 22637000000
        StockholdersEquity: 22637000000
               CommonStock: 425000000
    CashAndCashEquivalents: -558000000
           InvestedCapital: 44687000000
           InterestExpense: 1122000000
            InterestIncome: 94000000
             TotalExpenses: 82333000000
                DilutedEPS: 10010000
                  BasicEPS: 10120000
                 NetIncome: 13425000000
           OperatingIncome: 17719000000
          OperatingExpense: 0
             CostOfRevenue: 46272000000
              TotalRevenue: 95758000000
              FreeCashFlow: 17326000000
         FinancingCashFlow: -14700000000
         InvestingCashFlow: -6729000000
         OperatingCashFlow: 20773000000
         currentClosePrice: 78.41683959960938
              pToEDiluted: 0.000004840521569494958
                pToEBasic: 0.000004787907958501884
                DilutedPEG: 0.0000004002394223665738 5
                  BasicPEG: 0.0000004135840493677341 6
             revenueGrowth: -7.596255910450641
            piotroskiFscore: 7
                evToEbitda: 5.594699165783266
           enterpriseValue: 127072402152.4353
           marketCaptation: 102360402152.4353
                 evToSales: 1.3270160420271444
             priceToSales: 1.0689488309325101
                        bv: 22755000000
                priceToBv: 4.498369683693048
                 bvToShare: 17.432279939969526
               cashToShare: -0.4274758165898921
                priceToFCF: 5.907907315735617
                  FCFYield: 0.16926467301484502
              GrahamBasic: 63002.62012274618
             GrahamDiluted: 62659.2790373432
   totalEquityToTotalAsset: 0.20763699069912495
                    Dupont: 0.5930556169103679
             debtToCapital: 0.5162103823384839
                       DFL: 1.0676025787792975
              debtToEbitda: 1.0634438427332364
     InterestCoverageRatio: 15.792335115864528
                FCFToSales: 0.18093527433739218
              altmanZscore: 2.9236755412893105
JoelGreenblattsEarningsYield: 0.13944019078780295
                     croic: 0.38771902342963277
```

Figure 5.2: Valuation

```
      Date: 2018-08-15
    Symbol: IBM
      Open: 106.63524032981353
      High: 107.7199179197099
       Low: 106.22381789816683
     Close: 107.65260314941406
    Volume: 4436609
 Dividends: 0
StockSplits: 0
```

Figure 5.3: Performance

```
    name: Crude Oil Prices Brent
interval: monthly
    unit: dollars per barrel
    date: 2023-06-01
   value: 74.84
```

Figure 5.4: Sensitivities

```
 Date: 2018-08-16
 Name: CBOE Brexit High 50
 Open: 9764.6201171875
 High: 9764.6201171875
  Low: 9764.6201171875
Close: 9764.6201171875
```

Figure 5.5: Geopolitical

5.1.3 Real-time Updating Mechanism

To begin, the ZooKeeper and Kafka services must be started. Following this, it would be helpful to create distinct Kafka topics based on the various databases and the update frequency of the data. The subsequent step is to initiate the Flink server and execute scripts on the Flink platform(Figure 5.6). These scripts utilise the schedule mechanism so that tasks can be executed at the same frequency as data updates. This enables the automatic retrieval of real-time data from the data source, processing, and storage in the appropriate tables. Consequently, the objective of implementing an automatic update function for real-time data is met.

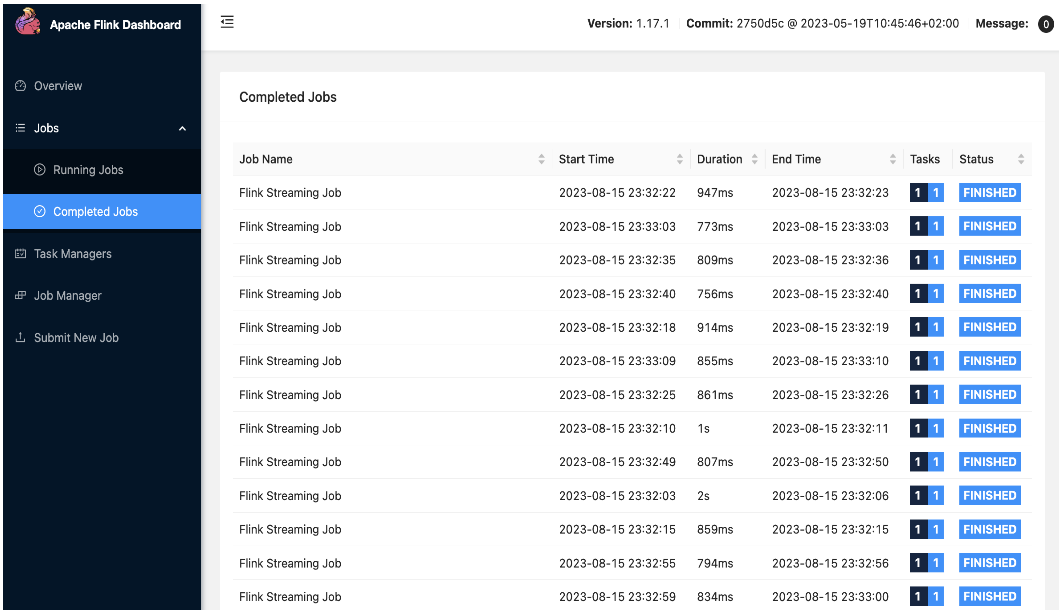


Figure 5.6: Stream Processing Tasks

5.1.4 User Interface

To accomplish a seamless connection with large-scale stock trading analysis systems, this project provides some interfaces for automatic generation of datasets(Json/CSV) and charts. By accessing the APIs(Table 5.2) designed with the parameters based on the desired data categories, companies, time periods, and factor names, users can obtain the required data sets. The outcome and log of calling one API is shown as Figure 5.7 and Figure 5.8.(Code details are shown in Appendix ”6. User Interface”)

Table 5.2: User Interface

Category	API(Flexible Parameters)
Valuation	http://127.0.0.1:5000/valuation_dataset?symbol=xx&interval=xx&start_data=YYYY-MM-DD&end_data=YYYY-MM-DD&format=xx
Performance	http://127.0.0.1:5000/performance_dataset?symbol=xx&start_data=YYYY-MM-DD&end_data=YYYY-MM-DD&format=xx
FX	http://127.0.0.1:5000/fx_dataset?from_symbol=xx&to_symbol=xx&start_data=YYYY-MM-DD&end_data=YYYY-MM-DD&format=xx
Sensitivities	http://127.0.0.1:5000/sensitivities_dataset?name=xx&start_data=YYYY-MM-DD&end_data=YYYY-MM-DD&format=xx
Geopolitical	http://127.0.0.1:5000/geopolitical_dataset?name=xx&start_data=YYYY-MM-DD&end_data=YYYY-MM-DD& format=xx

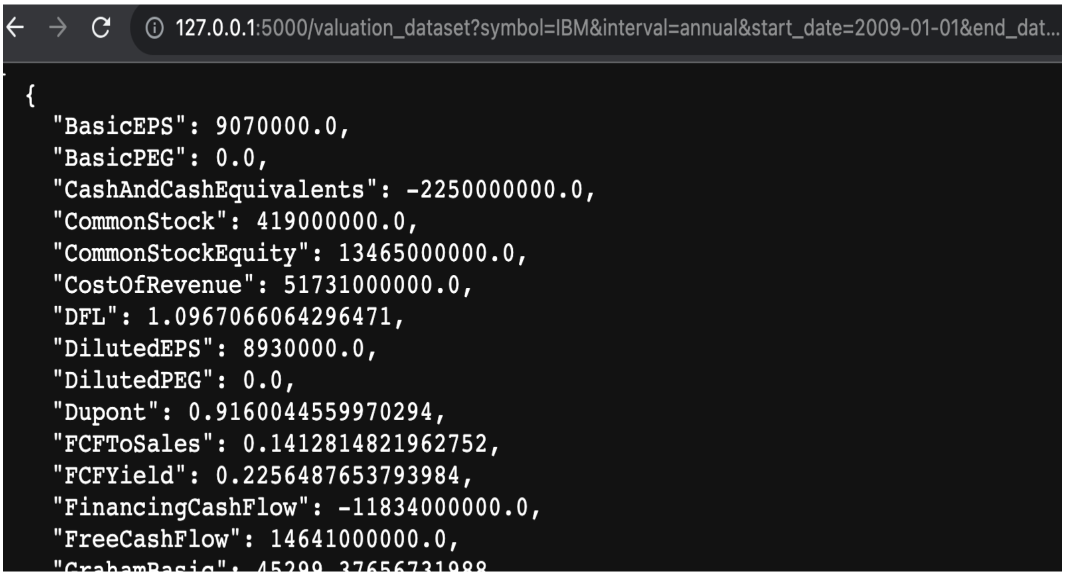


Figure 5.7: Outcome of Calling API

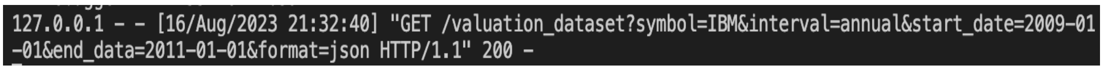


Figure 5.8: Log of Calling API

5.2 Discussion

The project builds a stock fundamental data pipeline system that effectively automates the processes of acquiring, processing, updating, and storing relevant data. This data orchestration system offers application programming interfaces (APIs) that are designed to be easily navigable by users. These APIs facilitate the integration of the system with downstream analysis and decision-making modules inside large stock trading systems. Consequently, the system is able to supply the necessary datasets for later analyses conducted within the whole system.

With regards to data collection, it is important to acknowledge that although the chosen data sources are generally regarded as reliable and trustworthy, there may be instances where they lack the requisite reliability and fail to give clients with the comprehensive data they need. Moreover, when it comes to data processing, the efficiency of valuation part is hindered by the heavy computational tasks, leading to low throughout. However, overall, the system successfully deploys an automated framework to coordinate the pipeline of acquiring, updating, processing, and storing stock fundamental data. To some extent, the system meets the requirements of a large-scale stock trading system in terms of data volume, efficiency, and the functioning of upstream data module. The system could handle vast data volumes efficiently and meet the demands of large-scale stock trading. As an upstream data module, the system stands out in its role, processing and verifying data from diverse sources accurately and in real-time. This combination of volume handling, efficiency, and data accuracy positions the system well for large-scale stock trading systems.

Chapter 6

Conclusions

In the contemporary financial industry landscape, data assumes a pivotal role within stock trading analysis systems. Through the autonomous provisioning of requisite datasets for subsequent model analysis, an automated data orchestration system holds the potential to enhance efficiency and productivity, affording investors the ability to remain informed about market dynamics and capitalize on emerging opportunities in a timely and informed manner.

This dissertation reveals the feasibility and potential of some data-related technologies such as big data real-time processing frameworks as well as messaging middleware for enhancing the efficiency and accuracy of data-related tasks in stock trading systems. By designing and implementing an automated data orchestration system, this project successfully resolves the issues of tedious, inefficient, time-consuming data acquisition, processing, update, and storage tasks, satisfying the requirements of a large-scale stock trading system in terms of data volume, efficiency, and the functioning of upstream data module.

Nevertheless, the project still has limitations. Firstly, with regards to data sources, it should be noted that while the three data sources are generally considered reliable and trustworthy, there may still be instances where certain data is unavailable. Furthermore, the impact of the network and other related factors can potentially result in instability in the functioning of data acquisition. In the context of missing data, due to the inconsistency in the criteria employed by various data sources and companies for generating financial reports and related content, it becomes challenging to seamlessly substitute one data source with another. As a result of time constraints to develop and implement the data orchestration system, the default issue in this project can only be addressed by setting the value to zero. It is necessary to make enhancements to the error mechanism in future iterations. Furthermore, the list of variables lacks comprehensiveness in terms of fundamental factors. There exist more fundamental factors that influence decisions relating to stock analysis. In the future, the list of factors may be expanded through additional exploration, thereby enhancing the comprehensiveness of the system. Meanwhile, it is better to employ machine learning algorithms to enhance the accuracy of predictive variables. In relation to the volume of data, an increase in data volume could lead to a more accurate and dependable system. Furthermore, there exists potential for additional enhancement in the throughput rate of the system, particularly in parts involving considerable amounts of calculation tasks, such as the valuation part. It is essential to significantly enhance the throughput rate in order to enhance the overall efficiency of the system.

In conclusion, this dissertation presents a robust and efficient prototype of an automated data orchestration platform tailored for stock fundamental analysis, with applications that were aimed to be extended to data-driven stock trading systems at wide. Moreover, it holds promise as a pivotal data-driven instrument for researchers engaged in model analysis within the realm of finance. The findings contribute substantial insights into the viable integration of real-time data tools by analysts and investors to amplify decision-making efficacy in stock trading systems. Furthermore, this work points toward an auspicious avenue for forthcoming research at the crossroads of finance, technology, and data-driven decision making.

Bibliography

[1] Alpha vantage. https://www.alphavantage.co/.

[2] Apache flink project. http://flink.apache.org/.

[3] Apache spark project. http://spark.apache.org/.

[4] Apache storm project. http://storm.apache.org/.

[5] Wharton research data service. https://wrds-www.wharton.upenn.edu/.

[6] Yahoo finance. https://finance.yahoo.com/.

[7] Paris Carbone, Gyula Fóra, Stephan Ewen, Seif Haridi, and Kostas Tzoumas. Lightweight asynchronous snapshots for distributed dataflows.

[8] Tsung Sheng Chang. A comparative study of artificial neural networks, and decision trees for digital game content stocks price prediction. *Expert Systems with Applications*, 38:14846–14851, 11 2011.

[9] Yingjun Chen and Yongtao Hao. A feature weighted support vector machine and k-nearest neighbor algorithm for stock market indices prediction. *Expert Systems with Applications*, 80:340–355, 9 2017.

[10] Yuting Chen, Ting Mao, and Bo Yu. A reliable messaging middleware for financial institutions. *ACM International Conference Proceeding Series*, pages 108–112, 11 2017.

[11] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, and Paul Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, pages 1789–1792, 7 2016.

[12] Manuel Díaz, Cristian Martín, and Bartolomé Rubio. State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. *Journal of Network and Computer applications*, 67:99–117, 2016.

[13] Adanma Cecilia Eberendu. Unstructured data: an overview of the data of big data. *International Journal of Computer Trends and Technology*, 38:46–50, 8 2016.

[14] Eugene F Fama. Efficient capital markets: Ii. *The journal of finance*, 46(5):1575–1617, 1991.

[15] Fu Ge, Zhang Xinhua, and Li Chao. Study of message data subscription based on multi-application big data analysis. *Netinfo Security*, (11):44–49, 2017.

[16] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. Benchmarking distributed stream data processing systems. *Proceedings - IEEE 34th International Conference on Data Engineering, ICDE 2018*, pages 1519–1530, 10 2018.

[17] Mahinda Mailagaha Kumbure, Christoph Lohrmann, Pasi Luukka, and Jari Porras. Machine learning techniques and data for stock market forecasting: A literature review. *Expert Systems with Applications*, 197:116659, 7 2022.

[18] Isaac Kofi Nti, Adebayo Felix Adekoya, and Benjamin Asubam Weyori. A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review 2019 53:4*, 53:3007–3057, 8 2019.

[19] Fagner A. De Oliveira, Cristiane N. Nobre, and Luis E. Zárate. Applying artificial neural networks to prediction of stock price and improvement of the directional prediction index – case study of petr4, petrobras, brazil. *Expert Systems with Applications*, 40:7596–7606, 12 2013.

[20] Bruce Snyder, Dejan Bosnanac, and Rob Davies. *ActiveMQ in action*, volume 47. Manning Greenwich Conn., 2011.

[21] P Sommer, Florian Schellroth, M Fischer, and Jan Schlechtendahl. Message-oriented middleware for industrial production systems. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1217–1223. IEEE, 2018.

[22] KHIN ME ME THEIN. Apache kafka: Next generation distributed messaging system khin me me thein. 12 2014.

[23] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. Storm @twitter. 2014.

[24] Chih Fong Tsai and Yu Chieh Hsiao. Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. *Decision Support Systems*, 50:258–269, 12 2010.

[25] Alvaro Videla and Jason JW Williams. *RabbitMQ in action: distributed messaging for everyone*. Manning, 2012.

[26] Ahmed. S. Wafi, Hassan Hassan, and Adel Mabrouk. Fundamental analysis models in financial markets – review study. *Procedia Economics and Finance*, 30:939–947, 1 2015.

[27] Zhenghe Wang, Wei Dai, Feng Wang, Hui Deng, Shoulin Wei, Xiaoli Zhang, and Bo Liang. Kafka and its using in high-throughput and reliable message distribution. In *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pages 117–120. IEEE, 2015.

[28] Jiang Yongguo, Liu Qiang, Qin Changshuai, Su Jian, and Liu Qianqian. Message-oriented middleware: A review. *Proceedings - 5th International Conference on Big Data Computing and Communications, BIGCOM 2019*, pages 88–97, 8 2019.

[29] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, and Scott Shenker. Discretized streams: Fault-tolerant streaming computation at scale.

[30] Xiangzhou Zhang, Yong Hu, Kang Xie, Shouyang Wang, E. W.T. Ngai, and Mei Liu. A causal feature selection algorithm for stock prediction modeling. *Neurocomputing*, 142:48–59, 10 2014.

Appendices

This section only includes the code for some of the functions. For detailed code, please refer to the code files.

.1 Detailed List of Raw Data

- **Alpha Vantage**

  Balance Sheet: 'symbol', 'fiscalDateEnding', 'totalAssets', 'totalCurrentAssets', 'shortTermInvestments', 'totalLiabilities', 'totalCurrentLiabilities', 'longTermDebt', 'totalShareholderEquity', 'treasuryStock', 'retainedEarnings', 'commonStockSharesOutstanding'

  Income Statement: 'symbol', 'fiscalDateEnding', 'ebit', 'ebitda'

  (Financial Reports for real-time part)

  FX rate: EUR to USD, GBP to USD, USD to JPY, AUD to USD, USD to CAD, USD to CHF, NZD to USD, GBP to EUR, USD to CNY, EUR to JPY

  Commodity/Energy Price: West Texas Intermediate(WTI) crude oil, Brent crude oil, natural gas, copper, aluminum, wheat, corn, cotton, sugar, coffee, global price index of all commodities

  Interest Rate

  Inflation Rate

- **Yahoo Finance(Yfiance)**

  Balance Sheet: 'ShareIssued', 'TotalDebt', 'CommonStockEquity', 'StockholdersEquity', 'CommonStock', 'CashAndCashEquivalents','InvestedCapital'

  Income Statement: 'InterestExpense', 'InterestIncome', 'TotalExpenses', 'DilutedEPS', 'BasicEPS', 'NetIncome', 'OperatingIncome', 'OperatingExpense', 'CostOfRevenue', 'TotalRevenue'

  Statement of Cash Flow: 'FreeCashFlow', 'FinancingCashFlow', 'InvestingCashFlow', 'OperatingCashFlow'

  (Financial Reports for real-time part)

  Stock Data

  CBOE Brexit High/Low 50

- **WRDS**

  Financial Statement: tic, datadate, ebit, ebitda, at, act, ivst, lt, lct, dltt, teq, tstk, re, csho, cshi, dt, ceq, seq, cstk, chech, icapt, xint, idit, xt, epsfi, epspx, ni, oiadp, xoprar, cogs, revt, capx, fincf, ivncf, oancf('symbol', 'fiscalDateEnding', 'ebit', 'ebitda', 'totalAssets', 'totalCurrentAssets', 'shortTermInvestments', 'totalLiabilities', 'totalCurrentLiabilities', 'longTermDebt', 'totalShareholderEquity', 'treasuryStock', 'retainedEarnings', 'commonStockSharesOutstanding', 'ShareIssued', 'TotalDebt', 'CommonStockEquity', 'StockholdersEquity', 'CommonStock', 'CashAndCashEquivalents', 'InvestedCapital', 'InterestExpense', 'InterestIncome', 'TotalExpenses', 'DilutedEPS', 'BasicEPS', 'NetIncome', 'OperatingIncome', 'OperatingExpense', 'CostOfRevenue', 'TotalRevenue', 'FreeCashFlow', 'FinancingCashFlow', 'InvestingCashFlow', 'OperatingCashFlow')

.2 Formulations of Stock Factors

1. **Market Capitalization**

$$\text{Market Capitalization} = P \times Q$$

Where:

- P = Current share price of the company's stock.
- Q = Total number of outstanding shares of the company.

Typically:

- Large-Cap: $10 billion.
- Mid-Cap: $2 billion to $10 billion.
- Small-Cap: $300 million to $2 billion.
- Micro-Cap: $50 million to $300 million.
- Nano-Cap: $50 million.

2. **Enterprise Value**

$$\text{EV} = \text{Market Capitalization} + \text{Total Debt} - \text{Cash and Cash Equivalents}$$

Where:

- Market Capitalization: The total value of all of a company's outstanding shares of stock.
- Total Debt: The sum of a company's long-term and short-term debt.
- Cash and Cash Equivalents: Assets that are cash or can be converted into cash swiftly.

3. **Earnings Per Share (EPS)**

$$\text{EPS} = \frac{\text{Net Income} - \text{Preferred Dividends}}{\text{Weighted Average Number of Common Shares Outstanding}}$$

Where:

- Net Income: The company's total earnings after deducting all expenses and taxes.
- Preferred Dividends: Dividends paid to preferred shareholders.
- Weighted Average Number of Common Shares Outstanding: The average number of shares over a certain reporting period, considering any changes in the number of shares over that period.

Typically:

- Positive EPS: Indicates profitability.
- Growing EPS: Can be a sign of financial health and potential future profitability.

4. **EV/EBITDA**

$$\text{EV/EBITDA} = \frac{\text{Enterprise Value (EV)}}{\text{Earnings Before Interest, Taxes, Depreciation, and Amortization (EBITDA)}}$$

Typically:

- 7x: Might be considered undervalued.
- 7x - 12x: Typically seen as a fair range.
- 12x: Could suggest overvaluation.

5. **EV/Sales**

$$\text{EV/Sales} = \frac{\text{Enterprise Value (EV)}}{\text{Total Sales (Revenue)}}$$

Typically:

- A low EV/Sales ratio might indicate potential undervaluation relative to the company's sales.
- A high EV/Sales ratio can suggest potential overvaluation.

6. **P/E(Price-to-Earnings Ratio)**

$$\text{P/E Ratio} = \frac{\text{Current Share Price}}{\text{Earnings Per Share (EPS)}}$$

Typically:

- A high P/E ratio might suggest that investors are expecting higher future earnings growth.
- A low P/E ratio could indicate lower expectations for future growth or perceived higher risk.

7. **PEG**

$$\text{PEG Ratio} = \frac{\text{P/E Ratio}}{\text{Annual EPS Growth Rate (expressed as a percentage)}}$$

Typically:

- A PEG ratio of 1 suggests the stock may be fairly valued given its growth rate.
- A PEG ratio less than 1 could indicate potential undervaluation relative to the company's growth prospects.
- A PEG ratio greater than 1 can suggest potential overvaluation given the expected growth rate.

8. **Price-to-Sales (P/S)**

$$\text{P/S Ratio} = \frac{\text{Market Capitalization}}{\text{Total Sales (Revenue)}}$$

Typically:

- A low P/S ratio might indicate potential undervaluation or challenges facing the company.
- A high P/S ratio could suggest potential overvaluation or high expected growth.

9. **Book Value**

$$\text{Book Value} = \text{Total Assets} - \text{Total Liabilities}$$

Typically:

- The book value provides an accounting-based perspective on a company's intrinsic value.
- Stocks trading below their book value might be seen as undervalued, though there could be reasons such as expected losses or operational challenges.

10. **Price/Book (P/B)**

$$\text{P/B} = \frac{\text{Total Number of Outstanding Shares}}{\text{Book Value}}$$

Typically:

- A P/B ratio below 1 might indicate the stock is undervalued relative to its book value, or there might be underlying challenges with the company.
- A P/B ratio above 1 typically suggests the market perceives additional value not captured in the book value, possibly due to expected growth or other strategic assets.

11. **Book Value Per Share(BVPS)**

$$\text{BVPS} = \frac{\text{Book Value}}{\text{Total Number of Outstanding Shares}}$$

12. **Revenue**

$$\text{Revenue} = \text{Quantity of Goods or Services Sold} \times \text{Selling Price Per Unit}$$

13. **Cash/Share**

$$\text{Cash Per Share} = \frac{\text{Total Cash and Cash Equivalents}}{\text{Total Number of Outstanding Shares}}$$

Where:

- Total Cash and Cash Equivalents include money market securities, bank accounts, and short-term marketable securities that can be easily converted into cash.
- Total Number of Outstanding Shares represents all the shares that have been authorized, issued, and purchased by investors.

Typically:

- A higher Cash Per Share can be a positive indicator of a company's strong liquidity position.
- A lower value might hint at potential liquidity challenges or signify that the company is investing its cash back into the business or returning it to shareholders.

14. **P/FCF**

$$\text{P/FCF Ratio} = \frac{\text{Share Price}}{\text{Free Cash Flow Per Share}}$$

Where the Free Cash Flow Per Share is given by:

$$\text{Free Cash Flow Per Share} = \frac{\text{Total Free Cash Flow}}{\text{Total Number of Outstanding Shares}}$$

And the Free Cash Flow is determined by:

$$\text{Free Cash Flow} = \text{Operating Cash Flow} - \text{Capital Expenditures}$$

Typically:

- A lower P/FCF ratio might indicate that the stock is undervalued based on its cash-generating abilities.
- Conversely, a higher P/FCF can suggest potential overvaluation unless the market expects significant future growth in free cash flow.

15. **FCF Yield**

$$\text{FCF Yield} = \frac{\text{Free Cash Flow}}{\text{Market Capitalization}} \times 100\%$$

Typically:

- A higher FCF Yield might suggest the company is undervalued and producing substantial free cash flow relative to its market value.
- A lower FCF Yield could indicate a potentially overvalued company or one generating a smaller amount of free cash flow in relation to its market capitalization.

16. **Graham Number**

$$\text{Graham Number} = \sqrt{22.5 \times EPS \times BVPS}$$

Typically:

- If the stock's current market price is below the Graham Number, it might be undervalued.
- If the stock's market price is higher than the Graham Number, it might suggest overvaluation.

17. **Total Equity/Total Liability**

$$\text{Equity to Liability Ratio} = \frac{\text{Total Equity}}{\text{Total Liability}}$$

Where:

- Total Equity is the residual interest in the assets of the company after deducting liabilities. It typically includes common stock, retained earnings, and additional paid-in capital.
- Total Liability represents all the debts and obligations the company owes, both in the short-term and long-term.

Typically:

- A ratio greater than 1 implies that the company has more equity than liabilities, indicating a potentially stronger financial position and less risk for creditors.

- Conversely, a ratio less than 1 suggests more liabilities than equity, which might indicate financial risk if not managed appropriately.

18. **DuPont Analysis**

    (a) Net Profit Margin (Profitability):

    $$\text{Net Profit Margin} = \frac{\text{Net Income}}{\text{Sales}}$$

    (b) Total Asset Turnover (Efficiency):

    $$\text{Total Asset Turnover} = \frac{\text{Sales}}{\text{Total Assets}}$$

    (c) Equity Multiplier (Leverage):

    $$\text{Equity Multiplier} = \frac{\text{Total Assets}}{\text{Shareholder's Equity}}$$

    Combining these components, the DuPont Analysis represents ROE as:

    $$\text{ROE} = \text{Net Profit Margin} \times \text{Total Asset Turnover} \times \text{Equity Multiplier}$$

19. **Total Debt/Capitalization**

    $$\text{Debt to Capitalization Ratio} = \frac{\text{Total Debt}}{\text{Total Debt} + \text{Shareholder's Equity}}$$

    Where:

    - Total Debt includes both short-term (current) and long-term debts.
    - Shareholder's Equity represents the owners' residual interest in the assets after deducting liabilities.

    Typically:

    - A higher ratio implies that the company is more leveraged, which might be riskier, especially during economic downturns or periods of rising interest rates.
    - A lower ratio indicates that the company might have a conservative capital structure, potentially meaning less financial risk, but also suggesting that it may not be capitalizing on the financial benefits of leverage.

20. **Debt/EBITDA**

$$\text{Debt to EBITDA Ratio} = \frac{\text{Total Debt}}{\text{EBITDA}}$$

Where:

- Total Debt includes both short-term (current) and long-term debts.
- EBITDA represents Earnings Before Interest, Taxes, Depreciation, and Amortization.

Typically:

- A higher Debt to EBITDA Ratio can suggest that the company may face challenges in servicing its debt, especially if earnings decline.
- Conversely, a lower ratio indicates that the company is generating a comfortable level of earnings relative to its debt, implying a potentially stronger financial position.

article

21. **Free Cash Flow to Sales Ratio**

$$\text{FCF to Sales Ratio} = \frac{\text{Free Cash Flow (FCF)}}{\text{Sales}}$$

Where:

- Free Cash Flow (FCF) is the cash generated after deducting capital expenditures from operating cash flow.
- Sales represents the company's total revenue.

Typically:

- A higher ratio indicates that a company is more efficiently converting its sales into cash, which can be used for various corporate activities like growth investments or shareholder returns.
- A lower ratio may imply operational inefficiencies or significant capital expenditures relative to the cash generated from sales.

22. **Interest Coverage Ratio**

$$\text{Interest Coverage Ratio} = \frac{\text{Earnings Before Interest and Taxes (EBIT)}}{\text{Interest Expense}}$$

Where:

- Earnings Before Interest and Taxes (EBIT) represents the company's operating profit.
- Interest Expense refers to the total interest payable on debts.

Typically:

- A higher Interest Coverage Ratio suggests that the company can easily meet its interest obligations using its operating profit.
- A low ratio may indicate potential difficulties in covering interest payments, especially if there are significant fluctuations in earnings.

23. **Degree of Financial Leverage(DFL)**

$$\text{DFL} = \frac{\text{EBIT}}{\text{EBIT} - \text{Interest Expense}}$$

Where:

- EPS stands for Earnings Per Share.
- EBIT represents Earnings Before Interest and Taxes.
- Interest Expense is the cost of debt for the period under consideration.

Typically:

- A DFL greater than 1 means financial leverage magnifies the effect of EBIT changes on EPS.
- A DFL of 1 implies that changes in EBIT have a direct proportional effect on EPS.
- A DFL less than 1 suggests that financial leverage dampens the impact of EBIT changes on EPS.

24. **Joel Greenblatt's Earnings Yield**

$$\text{Earnings Yield} = \frac{\text{Earnings Before Interest and Taxes (EBIT)}}{\text{Enterprise Value (EV)}}$$

Where:

- EBIT stands for Earnings Before Interest and Taxes, representing the company's operating earnings.
- Enterprise Value (EV) encompasses the market capitalization, minus cash and cash equivalents, plus total debt, minority interest, and preferred shares.

Typically:

- A higher Earnings Yield suggests that the company might be undervalued, indicating a potentially attractive investment opportunity.

25. **Cash Return on Invested Capital(CROIC)**

$$\text{CROIC} = \frac{\text{Free Cash Flow (FCF)}}{\text{Invested Capital}}$$

Where:

- Free Cash Flow (FCF) represents the net cash generated by the company's operations after accounting for capital expenditures.
- Invested Capital typically includes the sum of equity and debt, subtracting cash and cash equivalents.

Typically:

- A high CROIC value signifies that the company efficiently converts its investments into cash returns.

26. **Piotroski F-Score**

(a) Positive net income for the current year.

(b) Positive Return on Assets (ROA) for the current year.

(c) Positive operating cash flow for the current year.

(d) Cash flow from operations greater than net income.

(e) Decrease in long-term debt compared to the previous year.

(f) Increase in the current ratio compared to the previous year.

(g) No issuance of new shares in the last year.

(h) Increase in gross margin compared to the previous year.

(i) Increase in the asset turnover ratio compared to the previous year.

Typically:

- An F-Score between 7-9 denotes a strong financial position.
- An F-Score between 4-6 indicates average financial health.
- An F-Score between 0-3 suggests potential financial weaknesses.

27. **Altman's Z-Score**

$$Z = 1.2A + 1.4B + 3.3C + 0.6D + 1.0E$$

Where:

$$A = \frac{\text{Working Capital}}{\text{Total Assets}}$$
$$B = \frac{\text{Retained Earnings}}{\text{Total Assets}}$$
$$C = \frac{\text{Earnings Before Interest and Tax (EBIT)}}{\text{Total Assets}}$$
$$D = \frac{\text{Market Value of Equity}}{\text{Total Liabilities}}$$
$$E = \frac{\text{Sales}}{\text{Total Assets}}$$

Typically:

- $Z > 2.99$: Company is in the "Safe" zone.
- $1.81 < Z < 2.99$: Company is in a "Grey" zone.
- $Z < 1.81$: Company is in the "Distressed" zone, with a high risk of bankruptcy.

.3 Data Collection and Pre-Processing

```
#collect data from alpha vantage
def fetch_data_from_alpha(function, index):
    url = f'https://www.alphavantage.co/query?function={function}&symbol={index}&apikey=74OYDHNMV1Q7Y2A9'
    r = requests.get(url,verify=False)
    data = r.json()
    return data
```

```
#get S&P500 index list
def get_lists()
    sp500=pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')[0]
    symbols = sp500['Symbol'].tolist()
    return symbols
```

```
def income_statement_process(isdf, symbol):
    isdf.insert(0, 'symbol', symbol)
    df = isdf[['symbol', 'fiscalDateEnding', 'ebit', 'ebitda']]
    return df
def balance_sheet_process(bsdf, symbol):
    bsdf.insert(0, 'symbol', symbol)
    df = bsdf[['symbol', 'fiscalDateEnding', 'totalAssets','totalCurrentAssets',
                'shortTermInvestments', 'totalLiabilities', "totalCurrentLiabilities",
                 'longTermDebt', 'totalShareholderEquity', 'treasuryStock',
                 'retainedEarnings', 'commonStockSharesOutstanding']]
    return df
def cash_flow_process(cfdf, symbol):
    cfdf.insert(0, 'symbol', symbol)
    df = cfdf[['symbol', 'fiscalDateEnding']]
    return df
```

```
def AF_financial_report_collect(symbol):
```

```
    is_data=fetch_data_from_alpha('INCOME_STATEMENT',symbol)
    bs_data = fetch_data_from_alpha('BALANCE_SHEET', symbol)
    cf_data = fetch_data_from_alpha('CASH_FLOW', symbol)
    is_adf = pd.DataFrame(is_data['annualReports'])
    is_qdf = pd.DataFrame(is_data["quarterlyReports"])
    bs_adf = pd.DataFrame(bs_data['annualReports'])
    bs_qdf = pd.DataFrame(bs_data["quarterlyReports"])
    cf_qdf = pd.DataFrame(cf_data["quarterlyReports"])
    cf_adf = pd.DataFrame(cf_data['annualReports'])
    annual_df = pd.merge((pd.merge(income_statement_process(is_adf, symbol),
    balance_sheet_process(bs_adf, symbol), on=['symbol','fiscalDateEnding'], how='outer')),
    cash_flow_process(cf_adf, symbol), on=['symbol', 'fiscalDateEnding'], how='outer')
    quater_df = pd.merge((pd.merge(income_statement_process(is_qdf,symbol),
    balance_sheet_process(bs_qdf,symbol),on=['symbol','fiscalDateEnding'],how='outer')),cash_flow_process(cf_qdf,symbol), on=['symbol', 'fiscalDateEnding'], how='outer')
    annual_df['fiscalDateEnding'] = pd.to_datetime(annual_df['fiscalDateEnding'])
    quater_df['fiscalDateEnding'] = pd.to_datetime(quater_df['fiscalDateEnding'])
    columns_to_convert = ['ebit','ebitda','totalAssets','totalCurrentAssets',
    'shortTermInvestments','totalLiabilities',"totalCurrentLiabilities",
    'longTermDebt', 'totalShareholderEquity', 'treasuryStock', 'retainedEarnings', 'commonStockSharesOutstanding']
    for col in columns_to_convert:
        quater_df[col] = pd.to_numeric(quater_df[col], errors='coerce')
        annual_df[col] = pd.to_numeric(annual_df[col], errors='coerce')
    return annual_df, quater_df
```

```
def annual_financial_report_YF(symbol):
    #yearly balance sheet yahoo
    yf_balance_sheet = yf.Ticker(symbol).get_balance_sheet()
    yf_balance_sheet_filter = yf_balance_sheet.loc[['ShareIssued', 'TotalDebt',
         'CommonStockEquity','StockholdersEquity', 'CommonStock',
            'CashAndCashEquivalents','InvestedCapital']].transpose()
    yf_balance_sheet_filter = yf_balance_sheet_filter.reset_index(drop=False)
    yf_balance_sheet_filter = yf_balance_sheet_filter.rename(columns={yf_balance_sheet_filter.columns[0]: 'fiscalDateEnding'})
    yf_balance_sheet_filter.insert(0,'symbol',[symbol,symbol,symbol,symbol])
    #yearly income statement yahoo
    yf_income_statement = yf.Ticker(symbol).get_income_stmt()
    yf_income_statement_filter = yf_income_statement.loc[[ 'InterestExpense', 'InterestIncome','TotalExpenses', 'DilutedEPS', 'BasicEPS', 'NetIncome',
    'OperatingIncome', 'OperatingExpense','CostOfRevenue', 'TotalRevenue']].transpose()
    yf_income_statement_filter = yf_income_statement_filter.reset_index(drop=False)
    yf_income_statement_filter = yf_income_statement_filter.rename(columns={yf_income_statement_filter.columns[0]:'fiscalDateEnding'})
    yf_income_statement_filter.insert(0,'symbol',[symbol,symbol,symbol,symbol])
    #yearly cash flow yahoo
    yf_cash_flow = yf.Ticker(symbol).get_cash_flow()
    yf_cash_flow_filter = yf_cash_flow.loc[['FreeCashFlow',  'FinancingCashFlow',
         'InvestingCashFlow', 'OperatingCashFlow']].transpose()
    yf_cash_flow_filter = yf_cash_flow_filter.reset_index(drop
```

```
        =False)
    yf_cash_flow_filter = yf_cash_flow_filter.rename(columns={
        yf_cash_flow_filter.columns[0]:'fiscalDateEnding'})
    yf_cash_flow_filter.insert(0,'symbol',[symbol,symbol,
        symbol,symbol])
    yf_annual_fr =pd.merge(pd.merge(yf_balance_sheet_filter,
        yf_income_statement_filter,
    on=['symbol', 'fiscalDateEnding'], how='outer'),
        yf_cash_flow_filter,on=['symbol', 'fiscalDateEnding'],
        how='outer')
    #yf_annual_fr['fiscalDateEnding'] =
    yf_annual_fr['fiscalDateEnding'].dt.date.astype('object')
    columns_to_convert = ['ShareIssued', 'TotalDebt', '
        CommonStockEquity',
    'StockholdersEquity', 'CommonStock', '
        CashAndCashEquivalents','InvestedCapital', '
        InterestExpense', 'InterestIncome','TotalExpenses', '
        DilutedEPS', 'BasicEPS', 'NetIncome','OperatingIncome',
         'OperatingExpense','CostOfRevenue', 'TotalRevenue','
        FreeCashFlow', 'FinancingCashFlow','InvestingCashFlow',
         'OperatingCashFlow']
    for col in columns_to_convert:
         yf_annual_fr[col] = pd.to_numeric(yf_annual_fr[col],
            errors='coerce')
    yf_annual_fr.insert(1,'interval','annual')
    return yf_annual_fr
```

```
def quarter_financial_report_YF(symbol):
    #quartery balance sheet yahoo
    yf_balance_sheet_quarter = yf.Ticker(symbol).
        quarterly_balance_sheet
    rows_to_select_bs = ['Share Issued','Total Debt','Common
        Stock Equity',
    'Stockholders Equity','Common Stock','Cash And Cash
        Equivalents',"Invested Capital"]
    common_rows_bs = yf_balance_sheet_quarter.index.
        intersection(rows_to_select_bs)
```

```
#yf_balance_sheet_quarter_filter =
   yf_balance_sheet_quarter.loc[common_rows_bs].transpose(
   )
zero_bs = pd.DataFrame(0.0, index=rows_to_select_bs,
columns=yf_balance_sheet_quarter.columns)
zero_bs.update(yf_balance_sheet_quarter.loc[common_rows_bs
   ])
yf_balance_sheet_quarter_filter = zero_bs.transpose()
yf_balance_sheet_quarter_filter.columns =
   yf_balance_sheet_quarter_filter.columns.str.replace(' '
   , '')
yf_balance_sheet_quarter_filter =
   yf_balance_sheet_quarter_filter.reset_index(drop=False)
yf_balance_sheet_quarter_filter =
   yf_balance_sheet_quarter_filter.rename(columns={
   yf_balance_sheet_quarter_filter.columns[0]: '
   fiscalDateEnding'})
yf_balance_sheet_quarter_filter.insert(0,'symbol',symbol)
#quarterly income statement yahoo
yf_income_statement_quarter = yf.Ticker(symbol).
   quarterly_income_stmt
rows_to_select_is = ['Interest Expense', 'Interest Income'
   ,'Total Expenses', 'Diluted EPS', 'Basic EPS', 'Net
   Income', 'Operating Income', 'Operating Expense',
     'Cost Of Revenue', 'Total Revenue']
common_rows_is = yf_income_statement_quarter.index.
   intersection(rows_to_select_is)
zero_is = pd.DataFrame(0.0, index=rows_to_select_is,
   columns=yf_income_statement_quarter.columns)
zero_is.update(yf_income_statement_quarter.loc[
   common_rows_is])
yf_income_statement_quarter_filter = zero_is.transpose()
yf_income_statement_quarter_filter.columns =
   yf_income_statement_quarter_filter.columns.str.replace(
   ' ', '')
yf_income_statement_quarter_filter =
   yf_income_statement_quarter_filter.reset_index(drop=
```

```
False)
    yf_income_statement_quarter_filter = yf_income_statement_quarter_filter.rename(columns={yf_income_statement_quarter_filter.columns[0]:'fiscalDateEnding'})
    yf_income_statement_quarter_filter.insert(0,'symbol',symbol)
    #quarterly cash flow yahoo
    yf_cash_flow_quarter = yf.Ticker(symbol).quarterly_cash_flow
    rows_to_select_cf = ['Free Cash Flow', 'Financing Cash Flow','Investing Cash Flow', 'Operating Cash Flow']
    common_rows_cf = yf_cash_flow_quarter.index.intersection(rows_to_select_cf)
    zero_cf = pd.DataFrame(0.0, index=rows_to_select_cf,
    columns=yf_cash_flow_quarter.columns)
    zero_cf.update(yf_cash_flow_quarter.loc[common_rows_cf])
    yf_cash_flow_quarter_filter  = zero_cf.transpose()
    yf_cash_flow_quarter_filter.columns = yf_cash_flow_quarter_filter.columns.str.replace(' ', '')
    yf_cash_flow_quarter_filter = yf_cash_flow_quarter_filter.reset_index(drop=False)
    yf_cash_flow_quarter_filter = yf_cash_flow_quarter_filter.rename(columns={yf_cash_flow_quarter_filter.columns[0]:'fiscalDateEnding'})
    yf_cash_flow_quarter_filter.insert(0,'symbol',symbol)
    yf_quarter_fr = pd.merge(pd.merge(yf_balance_sheet_quarter_filter, yf_income_statement_quarter_filter,on=['symbol', 'fiscalDateEnding'], how='outer'), yf_cash_flow_quarter_filter,on=['symbol', 'fiscalDateEnding'], how='outer')
    columns_to_convert = ['ShareIssued', 'TotalDebt', 'CommonStockEquity',
     'StockholdersEquity', 'CommonStock','CashAndCashEquivalents',
```

```
        'InvestedCapital', 'InterestExpense', 'InterestIncome', 'TotalExpenses', 'DilutedEPS', 'BasicEPS', 'NetIncome', 'OperatingIncome', 'OperatingExpense',
        'CostOfRevenue', 'TotalRevenue','FreeCashFlow', 'FinancingCashFlow',
        'InvestingCashFlow', 'OperatingCashFlow']
    for col in columns_to_convert:
        yf_quarter_fr[col] = pd.to_numeric(yf_quarter_fr[col], errors='coerce')
    yf_quarter_fr.insert(1,'interval','quarter')
    return yf_quarter_fr
```

```
db = wrds.Connection()
def wrds_annual_hist(symbol):
    annual_report_query = f"""
        SELECT tic, datadate, ebit,ebitda,at, act, ivst, lt, lct, dltt, teq, tstk, re, csho, cshi, dt, ceq,seq, cstk,chech,icapt,xint,idit,xt,
        epsfi,epspx,ni,oiadp,xoprar,cogs,revt,capx,fincf,ivncf,oancf
        FROM comp.funda
        WHERE tic = '{symbol}'
        AND indfmt='INDL'
        AND datafmt='STD'
        AND popsrc='D'
        AND consol='C'
        AND datadate >= '2008-01-01'
    """
    annual_report_data = db.raw_sql(annual_report_query)
    pd.set_option('display.max_columns', None)
    new_colum_names =['symbol', 'fiscalDateEnding',  'ebit', 'ebitda','totalAssets', 'totalCurrentAssets', 'shortTermInvestments','totalLiabilities', 'totalCurrentLiabilities', 'longTermDebt','totalShareholderEquity', 'treasuryStock', 'retainedEarnings','commonStockSharesOutstanding',  'ShareIssued',
```

```
    'TotalDebt', 'CommonStockEquity', 'StockholdersEquity', 'CommonStock', 'CashAndCashEquivalents','InvestedCapital',  'InterestExpense',
    'InterestIncome', 'TotalExpenses', 'DilutedEPS', 'BasicEPS','NetIncome', 'OperatingIncome', 'OperatingExpense', 'CostOfRevenue','TotalRevenue', 'FreeCashFlow', 'FinancingCashFlow','InvestingCashFlow', 'OperatingCashFlow']
    annual_report_data.columns = new_colum_names
    annual_report_data['OperatingExpense']=
    annual_report_data['OperatingExpense'].fillna(0.0)
    annual_report_data.iloc[:, 2:] *= 1000000
    annual_report_data['TotalExpenses'] = annual_report_data['TotalRevenue'] - annual_report_data['NetIncome']
    annual_report_data['FreeCashFlow'] = annual_report_data['OperatingCashFlow'] - annual_report_data['FreeCashFlow']
    annual_report_data.insert(2, 'interval', 'annual')
    annual_report_data['fiscalDateEnding']= pd.to_datetime(annual_report_data['fiscalDateEnding'], format="%Y-%m-%d")
    return annual_report_data
```

.4 Batch Processing and Storage

```
def get_close_price(row):
#     row    ['fiscalDateEnding']      NaT
if pd.isna(row['fiscalDateEnding']):
    return 0.0

ticker = yf.Ticker(row['symbol'])
start = row['fiscalDateEnding'] - pd.Timedelta(days=1)
end = row['fiscalDateEnding']
history = ticker.history(start=start, end=end)

#            history
```

```
    if history.empty:
        return 0.0

    return history['Close'].iloc[0]

env_settings = EnvironmentSettings.in_batch_mode()
table_env = TableEnvironment.create(env_settings)
table_env.get_config().get_configuration().set_integer('parallelism',4)

jars = []
for file in os.listdir(os.path.abspath(os.path.dirname(__file__))):
    if file.endswith('.jar'):
        file_path = os.path.abspath(file)
        jars.append(file_path)

str_jars = ';'.join(['file:///' + jar for jar in jars])
table_env.get_config().get_configuration().set_string("pipeline.jars", str_jars)

create_sink_sql = '''
CREATE TABLE valuationTotal(
symbol STRING,
fiscalDateEnding TIMESTAMP(6),
`interval` STRING,
ebit DOUBLE,
ebitda DOUBLE,
totalAssets DOUBLE,
totalCurrentAssets DOUBLE,
shortTermInvestments DOUBLE,
totalLiabilities DOUBLE,
totalCurrentLiabilities DOUBLE,
longTermDebt DOUBLE,
totalShareholderEquity DOUBLE,
treasuryStock DOUBLE,
retainedEarnings DOUBLE,
```

```
commonStockSharesOutstanding DOUBLE,
ShareIssued DOUBLE,
TotalDebt DOUBLE,
CommonStockEquity DOUBLE,
StockholdersEquity DOUBLE,
CommonStock DOUBLE,
CashAndCashEquivalents DOUBLE,
InvestedCapital DOUBLE,
InterestExpense DOUBLE,
InterestIncome DOUBLE,
TotalExpenses DOUBLE,
DilutedEPS DOUBLE,
BasicEPS DOUBLE,
NetIncome DOUBLE,
OperatingIncome DOUBLE,
OperatingExpense DOUBLE,
CostOfRevenue DOUBLE,
TotalRevenue DOUBLE,
FreeCashFlow DOUBLE,
FinancingCashFlow DOUBLE,
InvestingCashFlow DOUBLE,
OperatingCashFlow DOUBLE,
currentClosePrice DOUBLE,
pToEDiluted DOUBLE,
pToEBasic DOUBLE,
DilutedPEG DOUBLE,
BasicPEG DOUBLE,
revenueGrowth DOUBLE,
piotroskiFscore BIGINT,
evToEbitda DOUBLE,
enterpriseValue DOUBLE,
marketCaptation DOUBLE,
evToSales DOUBLE,
priceToSales DOUBLE,
bv DOUBLE,
priceToBv DOUBLE,
bvToShare DOUBLE,
```

```
cashToShare DOUBLE,
priceToFCF DOUBLE,
FCFYield DOUBLE,
GrahamBasic DOUBLE,
GrahamDiluted DOUBLE,
totalEquityToTotalAsset DOUBLE,
Dupont DOUBLE,
debtToCapital DOUBLE,
DFL DOUBLE,
debtToEbitda DOUBLE,
InterestCoverageRatio DOUBLE,
FCFToSales DOUBLE,
altmanZscore DOUBLE,
JoelGreenblattsEarningsYield DOUBLE,
croic DOUBLE,
PRIMARY KEY (symbol,fiscalDateEnding,`interval`) NOT ENFORCED
) WITH (
            'connector' = 'jdbc',
            'url' = 'jdbc:mysql://localhost:3306/mydatabase?
               useSSL=false',
            'driver' = 'com.mysql.jdbc.Driver',
            'table-name' = 'valuationTotal',
            'username' = 'root',
            'password' = '12345678'
)
'''
table_env.execute_sql(create_sink_sql)

#ev/ebitda
@udf(input_types=[DataTypes.DOUBLE(), DataTypes.DOUBLE(),
   DataTypes.DOUBLE(), DataTypes.DOUBLE(), DataTypes.DOUBLE()]
   ,
     result_type=DataTypes.DOUBLE())
def an_ev_to_ebitda(shareOutstanding, close, totalDebt, cash,
   ebitda):
    #date = an_fr['fiscalDateEnding']
```

```
    #Enterprise Value = Market Capitalization + Total Debt - Cash and Cash Equivalents
    if shareOutstanding is None or close is None or totalDebt is None or cash is None or ebitda is None or ebitda == 0.0 :
        ev_to_ebitda = 0.0
    else:
        ev_to_ebitda = (shareOutstanding * close + totalDebt - cash) /ebitda
    #an_fr['ev/ebitda'] = (an_fr['commonStockSharesOutstanding'] * current_close_price(date,symbol) + an_fr['TotalDebt'] - an_fr['CashAndCashEquivalents']) / an_fr['ebitda']
    return ev_to_ebitda
table_env.register_function('evToEbitda',an_ev_to_ebitda)

#ev
@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(), DataTypes.DOUBLE(),DataTypes.DOUBLE()],
     result_type = DataTypes.DOUBLE())
def an_ev(shareOutstanding, close, totalDebt, cash):
    if shareOutstanding is None or close is None or  totalDebt is None or cash is None:
        ev =0.0
    else:
        ev = shareOutstanding * close + totalDebt - cash
    return ev
table_env.register_function('ev',an_ev)

#marketCap
@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
     result_type=DataTypes.DOUBLE())
def an_marketcap(shareOutstanding, close):
    if shareOutstanding is None or close is None:
        marketcap = 0.0
    else:
        marketcap = shareOutstanding * close
```

```
    return marketcap
table_env.register_function('marketCap',an_marketcap)

# EV/Sales
@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),
    DataTypes.DOUBLE(),DataTypes.DOUBLE(),DataTypes.DOUBLE()],
     result_type=DataTypes.DOUBLE())
def an_ev_to_sales(shareOutstanding, close, totalDebt, cash,
    totalRevenue):
    if shareOutstanding is None or close is None or totalDebt
       is None or cash is None or totalRevenue is None or
       totalRevenue==0.0:
        ev_to_sales=0.0
    else:
        ev_to_sales = (shareOutstanding * close + totalDebt -
           cash) / totalRevenue
    return ev_to_sales
table_env.register_function('evToSales',an_ev_to_sales)

# Price/Sales = Market Capitalization / Total Revenue (or
    Sales)
@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),
    DataTypes.DOUBLE()],
     result_type=DataTypes.DOUBLE())
def an_price_to_sales(shareOutstanding, close,totalRevenue):
    if shareOutstanding is None or close is None or
       totalRevenue  is None or totalRevenue==0.0:
        price_to_sales =0.0
    else:
        price_to_sales = shareOutstanding*close/totalRevenue
    return price_to_sales
table_env.register_function('price/sales', an_price_to_sales)

#bv
@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
     result_type=DataTypes.DOUBLE())
def an_bv(totalAsset,totalLib):
```

```
    if totalAsset is None or totalLib is None:
        bv = 0.0
    else:
        bv = totalAsset - totalLib
    return  bv
table_env.register_function('bv',an_bv)

#P/B = Market Capitalization / Total Book Value
@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),
   DataTypes.DOUBLE(),DataTypes.DOUBLE()],
     result_type=DataTypes.DOUBLE())
def an_price_to_bv(shareOutstanding, close,totalAsset,totalLib
   ):
    if shareOutstanding is None or close is None or totalAsset
        is None or totalLib is None or totalAsset == totalLib:
        pb = 0.0
    else:
        bv = totalAsset - totalLib
        pb = shareOutstanding * close / bv
    return pb
table_env.register_function('P/B',an_price_to_bv)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),
   DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_bv_to_share(totalAsset,totalLib,shareOutstanding):
    if totalAsset is None or totalLib is None or
       shareOutstanding is None or shareOutstanding ==0.0:
        bs  = 0.0
    else:
        bv = totalAsset - totalLib
        bs = bv/shareOutstanding
    return bs
table_env.register_function('bv/share',an_bv_to_share)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
```

```
def an_cash_to_share(cash,shareOutstanding):
    if cash is None or shareOutstanding is None or
       shareOutstanding ==0.0:
        ratio = 0.0
    else:
        ratio = cash/shareOutstanding
    return ratio
table_env.register_function('cash/share',an_cash_to_share)


@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),
   DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_price_to_FCF(shareOutstanding,close,fcf):
    if shareOutstanding is None or close is None or fcf is
       None or fcf==0.0:
        ratio = 0.0
    else:
        ratio = shareOutstanding*close/fcf
    return ratio
table_env.register_function('price/FCF',an_price_to_FCF)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),
   DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_FCF_Yield(shareOutstanding,close,fcf):
    if shareOutstanding is None or close is None or fcf is
       None  or shareOutstanding ==0.0 or close ==0.0:
        ratio = 0.0
    else:
        ratio = fcf/(shareOutstanding * close)
    return ratio
table_env.register_function('FCF Yield',an_FCF_Yield)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),
   DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
```

```
def an_Graham_basic(totalAsset,totalLib,shareOutstanding,
    basiceps):
    if totalAsset is None or totalLib is None or
        shareOutstanding is None or basiceps is None or
        shareOutstanding==0.0:
        ratio = 0.0
    else:
        bv = totalAsset - totalLib
        if (22.5 * (bv/shareOutstanding ) * basiceps)>0.0:
            ratio = np.sqrt(22.5 * (bv/shareOutstanding ) *
                basiceps)
        else:
            ratio = 0.0
    return ratio
table_env.register_function('GrahamBasic',an_Graham_basic)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),
    DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_Graham_du(totalAsset,totalLib,shareOutstanding,dueps):
    if totalAsset is None or totalLib is None or
        shareOutstanding is None or dueps is None or
        shareOutstanding==0.0:
        ratio = 0.0
    else:
        bv = totalAsset - totalLib
        if (22.5 * (bv/shareOutstanding ) * dueps)>0.0:
            ratio = np.sqrt(22.5 * (bv/shareOutstanding ) *
                dueps)
        else:
            ratio = 0.0
    return ratio
table_env.register_function('GrahamDu',an_Graham_du)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_EA(shareequity, totalAsset):
```

```
    if shareequity is None or totalAsset is None or totalAsset ==0.0:
        ratio = 0.0
    else:
        ratio =shareequity/totalAsset
    return ratio
table_env.register_function('total equity/total asset', an_EA)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(), DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_Dupont(netIncome, totalRe, totalAs, shareEq):
    if netIncome is None or totalRe is None or totalAs is None or shareEq is None or totalRe==0.0 or totalAs==0.0 or shareEq==0.0:
        ratio=0.0
    else:
        ratio = (netIncome/totalRe)*(totalRe/totalAs)*(totalAs/shareEq)
    return ratio
table_env.register_function('Dupont',an_Dupont)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_debt_to_capital(totalde,equity):
    if totalde is None or equity is None:
        ratio =0.0
    else:
        if (totalde+equity)==0.0:
            ratio = 0.0
        else:
            ratio = totalde/(totalde+equity)
    return ratio
table_env.register_function('debt/capital',an_debt_to_capital)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
```

```
def an_DFL(ebit,inex):
    if ebit is None or inex is None:
        ratio = 0.0
    else:
        if (ebit-inex)==0:
            ratio =0.0
        else:
            ratio = ebit/(ebit-inex)
    return ratio
table_env.register_function('DFL',an_DFL)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_debt_to_ebitda(totalDebt, ebitda):
    if totalDebt is None or ebitda is None or ebitda==0.0:
        ratio = 0.0
    else:
        ratio = totalDebt/ebitda
    return ratio
table_env.register_function('debit/ebitda',an_debt_to_ebitda)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_InterestCoverageRatio(oi,ie):
    if oi is None or ie is None or ie==0.0:
        ratio  = 0.0
    else:
        ratio = oi/ie
    return ratio
table_env.register_function('InterestCoverageRatio',
   an_InterestCoverageRatio)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_FCF_to_sales(cash,totalre):
    if cash is None or totalre is None or totalre==0.0:
        ratio =0.0
```

```
    else:
        ratio = cash / totalre
    return ratio
table_env.register_function('FCF/sales',an_FCF_to_sales)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),DataTypes.DOUBLE(),
                  DataTypes.DOUBLE(),DataTypes.DOUBLE(),DataTypes.DOUBLE(),
                  DataTypes.DOUBLE(),DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_altman_zscore(totalCurrentAssets,totalCurrentLiabilities,retainedEarnings,
                     totalAssets,ebit,commonStockSharesOutstanding,close,
                     totalLiabilities,TotalRevenue):
    if totalCurrentAssets is None or totalCurrentLiabilities is None or retainedEarnings is None or totalAssets is None or ebit is None or commonStockSharesOutstanding is None or close is None or totalLiabilities is None or TotalRevenue is None:
        ratio=0.0
    else:
        if totalCurrentLiabilities==0.0 or totalAssets==0.0 or totalLiabilities==0.0:
            ratio = 0.0
        else:
            ratio = 1.2 * ((totalCurrentAssets / totalCurrentLiabilities) / totalAssets)
            ratio += 1.4 * (retainedEarnings / totalAssets)
            ratio += 3.3 * (ebit / totalAssets)
            ratio += 0.6 * ((commonStockSharesOutstanding * close) / totalLiabilities)
            ratio += 1.0 * (TotalRevenue / totalAssets)
    return ratio
table_env.register_function('altman_zscore',an_altman_zscore)
```

```
@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE(),DataTypes.DOUBLE(),
                  DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_JoelGreenblattsEarningsYield(ebit, commonStockSharesOutstanding,close,TotalDebt, CashAndCashEquivalents):
    if ebit is None or commonStockSharesOutstanding is None or close is None or TotalDebt is None or CashAndCashEquivalents is None:
        ratio = 0.0
    else:
        if (commonStockSharesOutstanding*close+TotalDebt-CashAndCashEquivalents)==0.0:
            ratio =0.0
        else:
            ratio = ebit/(commonStockSharesOutstanding*close+TotalDebt-CashAndCashEquivalents)
    return ratio
table_env.register_function('JoelGreenblattsEarningsYield', an_JoelGreenblattsEarningsYield)

@udf(input_types=[DataTypes.DOUBLE(),DataTypes.DOUBLE()],
      result_type=DataTypes.DOUBLE())
def an_croic(FreeCashFlow,InvestedCapital):
    if FreeCashFlow is None or InvestedCapital is None or InvestedCapital==0.0:
        ratio=0.0
    else:
        ratio = FreeCashFlow / InvestedCapital
    return ratio
table_env.register_function('croic',an_croic)

PROCESSED_SYMBOLS_FILE = "valuation_processed_symbols.txt"

def load_processed_symbols():
```

```
    if os.path.exists(PROCESSED_SYMBOLS_FILE):
        with open(PROCESSED_SYMBOLS_FILE, "r") as f:
            return set(f.read().splitlines())
    return set()


def save_processed_symbol(symbol):
    with open(PROCESSED_SYMBOLS_FILE, "a") as f:
        f.write(symbol + "\n")


def process_symbol(symbol):
    processed_symbols = load_processed_symbols()
    if symbol in processed_symbols:
        print(f"{symbol} has already been processed. Skipping...")
        return

    try:
        start_time = time.time()
        annual = wrds_annual_hist.wrds_annual_hist(symbol)
        if annual is None or annual.empty:
            print(f"No data available for {symbol}. Skipping...")
            return

        annual.fillna(0.0, inplace=True)
        annual['currentClosePrice'] = annual.apply(get_close_price, axis=1)
        annual = valuation_metrics_function.mini_factors_generate(annual, symbol)
        table = table_env.from_pandas(annual)
        result = table.select(table.symbol, table.fiscalDateEnding, table.interval, table.ebit, table.ebitda, table.totalAssets,
                              table.totalCurrentAssets, table.shortTermInvestments, table.
```

```
        totalLiabilities,
        table.totalCurrentLiabilities, table.longTermDebt, table.totalShareholderEquity,
        table.treasuryStock, table.retainedEarnings, table.commonStockSharesOutstanding,
        table.ShareIssued, table.TotalDebt, table.CommonStockEquity, table.StockholdersEquity,
        table.CommonStock, table.CashAndCashEquivalents, table.InvestedCapital, table.InterestExpense, table.InterestIncome,
        table.TotalExpenses, table.DilutedEPS, table.BasicEPS, table.NetIncome,
        table.OperatingIncome, table.OperatingExpense, table.CostOfRevenue,
        table.TotalRevenue, table.FreeCashFlow, table.FinancingCashFlow,
        table.InvestingCashFlow, table.OperatingCashFlow,
        table.currentClosePrice, table.pToEDiluted, table.pToEBasic, table.DilutedPEG, table.BasicPEG,
        table.revenueGrowth,table.piotroskiFscore,
        an_ev_to_ebitda(table.commonStockSharesOutstanding, table.currentClosePrice,table.TotalDebt, table.CashAndCashEquivalents, table.ebitda).alias('evToEbitda'),
        an_ev(table.commonStockSharesOutstanding, table.currentClosePrice,table.TotalDebt, table.CashAndCashEquivalents).alias('enterpriseValue'),
        an_marketcap(table.commonStockSharesOutstanding, table.
```

```
currentClosePrice).alias('marketCaptation'),
        an_ev_to_sales(table.commonStockSharesOutstanding, table.currentClosePrice,table.TotalDebt,table.CashAndCashEquivalents,table.TotalRevenue).alias('evToSales'),
        an_price_to_sales(table.commonStockSharesOutstanding, table.currentClosePrice,table.TotalRevenue).alias('priceToSales'),
        an_bv(table.totalAssets, table.totalLiabilities).alias('bv'),
        an_price_to_bv(table.commonStockSharesOutstanding, table.currentClosePrice,table.totalAssets,table.totalLiabilities).alias('priceToBv'),
        an_bv_to_share(table.totalAssets, table.totalLiabilities,table.commonStockSharesOutstanding).alias('bvToShare'),
        an_cash_to_share(table.CashAndCashEquivalents,table.commonStockSharesOutstanding).alias('cashToShare'),
        an_price_to_FCF(table.commonStockSharesOutstanding, table.currentClosePrice,table.FreeCashFlow).alias('priceToFCF'),
        an_FCF_Yield(table.commonStockSharesOutstanding, table.currentClosePrice,table.FreeCashFlow).alias('FCFYield'),
        an_Graham_basic(table.totalAssets, table.totalLiabilities,table.commonStockSharesOutstanding,table.
```

```
BasicEPS).alias('GrahamBasic'),
    an_Graham_du(table.totalAssets, table.totalLiabilities,table.commonStockSharesOutstanding,table.DilutedEPS).alias('GrahamDiluted'),
    an_EA(table.StockholdersEquity,table.totalAssets).alias('totalEquityToTotalAsset'),
    an_Dupont(table.NetIncome,table.TotalRevenue, table.totalAssets,table.StockholdersEquity).alias('Dupont'),
    an_debt_to_capital(table.TotalDebt, table.StockholdersEquity).alias('debtToCapital'),
    an_DFL(table.ebit,table.InterestExpense).alias('DFL'),
    an_debt_to_ebitda(table.TotalDebt,table.ebitda).alias('debtToEbitda'),
    an_InterestCoverageRatio(table.OperatingIncome,table.InterestExpense).alias('InterestCoverageRatio'),
    an_FCF_to_sales(table.FreeCashFlow,table.TotalRevenue).alias('FCFToSales'),

    an_altman_zscore(table.totalCurrentAssets,table.totalCurrentLiabilities,table.retainedEarnings,
    table.totalAssets,table.ebit,table.commonStockSharesOutstanding,table.currentClosePrice,
    table.totalLiabilities,table.TotalRevenue).alias('altmanZscore'),
    an_JoelGreenblattsEarningsYield(table.ebit,table.commonStockSharesOutstanding,table.currentClosePrice,table.TotalDebt,table.CashAndCashEquivalents).alias('JoelGreenblattsEarningsYield'),
```

```
                    an_croic(table.FreeCashFlow,table.InvestedCapital).alias('croic')
                    )

        table_env.create_temporary_view('temporary_table', result)

        table_env.execute_sql("INSERT INTO valuationTotal SELECT * FROM temporary_table").wait()
        table_env.drop_temporary_view('temporary_table')
        print(f"{symbol} stored to database successfully!")
        end_time = time.time()  # End time of the processing

        duration = end_time - start_time
        throughput = len(annual) / duration
        print(f"{symbol} processed in {duration:.2f} seconds with a throughput of {throughput:.2f} rows/second.")

        # Once the symbol is processed and stored successfully, save it to the file
        save_processed_symbol(symbol)
    except Exception as e:
        print(f"Error processing {symbol}: {e}")

def process_group(symbols_group):
    for symbol in symbols_group:
        process_symbol(symbol)

group_size = math.ceil(len(symbols) / 20)
symbol_groups = [symbols[i:i+group_size] for i in range(0, len(symbols), group_size)]

for idx, symbols_group in enumerate(symbol_groups, start=1):
    print(f"Processing group {idx} of {len(symbol_groups)}...")
    process_group(symbols_group)
```

```
    print(f"Group {idx} processed successfully!")
```

.5 Real-time Stream Processing and Storage

```
def read_stock_from_kafka(env):
    deserialization_schema = JsonRowDeserializationSchema.Builder() \
        .type_info(Types.ROW([  Types.STRING(),Types.STRING(),Types.DOUBLE(),Types.DOUBLE(),Types.DOUBLE(),Types.DOUBLE(),Types.INT(),Types.DOUBLE(),Types.DOUBLE()
])) \
        .build()
    kafka_consumer = FlinkKafkaConsumer(
        topics='stock_topic',
        deserialization_schema=deserialization_schema,
        properties={'bootstrap.servers': 'localhost:9092', 'group.id': 'test_group_1'}
    )
    result = env.add_source(kafka_consumer)
    result.add_sink(JdbcSink.sink(
        "INSERT IGNORE INTO stock_price_test (`Date`, `Symbol`, `Open`, `High`,`Low`,`Close` ,`Volume`,`Dividends`,`StockSplits`) values (?,?,?,?,?,?,?,?,?)",
Types.ROW([Types.STRING(),Types.STRING(),Types.DOUBLE(),Types.DOUBLE(),Types.DOUBLE(),Types.DOUBLE(),Types.INT(),Types.DOUBLE(),Types.DOUBLE()
]),
    JdbcConnectionOptions.JdbcConnectionOptionsBuilder()
    .with_url('jdbc:mysql://localhost:3306/mydatabase?useSSL=false')
    .with_driver_name('com.mysql.jdbc.Driver')
    .with_user_name('root')
    .with_password('12345678')
    .build(),
    JdbcExecutionOptions.builder()
```

```
    .with_batch_interval_ms(5000)
    .with_batch_size(500)
    .with_max_retries(5)
    .build()
    ))
    print( "store to database")
    env.execute()
if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    env = StreamExecutionEnvironment.get_execution_environment() env.add_jars("file:///Users/zhouyuxuan/Desktop/Test/Libs/flink-sql-connector-kafka-1.17.1.jar",
                    "file:///Users/zhouyuxuan/Desktop/Test/flink-connector-jdbc-3.1.0-1.17.jar",
                    "file:///Users/zhouyuxuan/Desktop/Test/mysql-connector-java-8.0.30.jar")
    print("start reading from kafka")
    read_stock_from_kafka(env)
```

```
def json_latest_stock(symbol):
    try:
        history = yf.Ticker(symbol).history(period='1d')
    except Exception as e:
        print(f"Failed to get stock price for {symbol}: {e}")
        raise
    else:
        if not history.empty:
            history['Symbol'] = symbol
            last_column = history.columns[-1]
            history = history[[last_column] + list(history.columns[:-1])]
            history.reset_index(drop=False,inplace=True)
            history['Date']=pd.to_datetime(history['Date']).dt.date
            history['Date'] = history['Date'].astype(str)
            history['Volume'] = history['Volume'].astype(int)
```

```python
            tuples = list(history.itertuples(index=False, name=None))
            return tuples
        else:
            print(symbol, "fetched no data for stock price")

def write_stock_to_kafka(env,json_data):
    type_info = Types.ROW([
Types.STRING(),Types.STRING(),Types.DOUBLE(),Types.DOUBLE(),Types.DOUBLE(),Types.DOUBLE(),Types.INT(),Types.DOUBLE(),Types.DOUBLE()
])
    ds = env.from_collection(json_data,
        type_info=type_info)

    serialization_schema = JsonRowSerializationSchema.Builder() \
        .with_type_info(type_info) \
        .build()
    kafka_producer = FlinkKafkaProducer(
        topic='stock_topic',
        serialization_schema=serialization_schema,
        producer_config={'bootstrap.servers': 'localhost:9092', 'group.id': 'test_group'}
    )
    # note that the output type of ds must be RowTypeInfo
    ds.add_sink(kafka_producer)
    env.execute()
def job():
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    env = StreamExecutionEnvironment.get_execution_environment()
    env.add_jars("file:///Users/zhouyuxuan/Desktop/Test/Libs/flink-sql-connector-kafka-1.17.1.jar",
                    "file:///Users/zhouyuxuan/Desktop/Test/flink-connector-jdbc-3.1.0-1.17.jar",
```

```
                        "file:///Users/zhouyuxuan/Desktop/Test/mysql-connector-java-8.0.30.jar")
    for symbol in symbols:
        try:
            data = json_latest_stock(symbol)
        except Exception as e:
            print(f"Failed to get daily stock price for {symbol}: {e}")
            continue
        else:
            print("start writing to kafka")
            write_stock_to_kafka(env, data)
            print('writing done')
# fetch latest stock data (yesterday) at 00:00
schedule.every().day.at("00:00").do(job)
while True:
    schedule.run_pending()  # check whether there exists job to execute
    time.sleep(1)  # wait 1s
```

.6 User Interface

```
    @app.route('/valuation_dataset', methods=['GET'])
def get_valuation_data():
    conn = mysql.connect()
    cursor = conn.cursor()
    symbol = request.args.get('symbol', None)
    start_date = request.args.get('start_date', None)
    end_date = request.args.get('end_date', None)
    # fiscal_date_ending = request.args.get('fiscalDateEnding', None)
    interval = request.args.get('interval', None)
    output_format = request.args.get('format', 'json')  # Default to 'json', but can also be 'csv'
    query = "SELECT * FROM valuationTotal WHERE 1=1"
    if symbol:
```

```
        query += f" AND symbol='{symbol}'"
    if start_date and end_date:
        query += f" AND fiscalDateEnding BETWEEN '{start_date}' AND '{end_date}'"
    # if fiscal_date_ending:
    #     query += f" AND fiscalDateEnding='{fiscal_date_ending}'"
    if interval:
        query += f" AND `interval`='{interval}'"
    cursor.execute(query)
    data = cursor.fetchall()
    column_names = [i[0] for i in cursor.description]
    if output_format == 'json':
        result = [dict(zip(column_names, row)) for row in data]
        return jsonify(result)
    elif output_format == 'csv':
        output = StringIO()
        writer = csv.writer(output)
        writer.writerow(column_names)  # write header
        writer.writerows(data)
        output.seek(0)
        return output.getvalue(), 200, {
            'Content-Disposition': 'attachment; filename=valuation_data.csv',
            'Content-Type': 'text/csv'
        }
    else:
        return jsonify({"error": "Invalid format requested"}), 400
```