***Real-Time Financial Data Analysis***

(Real Time Big Data)

**Author**:

Abhishek Bargujar

Professor:

Francesco Corcoglioniti

**Abstract**:

This project aims to develop a real-time financial data analysis platform integrating various data sources such as Yahoo Finance to fetch, process, and visualize stock data. Utilizing advanced technologies such as Kafka, Flink, PostgreSQL, and Stream lit, the platform provides real-time insights and visualizations to assist users in making informed investment decisions. This comprehensive report outlines the application domain, data sources, technologies, system architecture, functionalities, and lessons learned throughout the project's development.

**Link to Code Repository**: [git@gitlab.inf.unibz.it:Abhishek.Bargujar/big\_data\_share\_market.git]

|  |
| --- |
| Table of Contents |
| 1. Application Domain |
| 1. Description of the Data Sources |
| 1. Technologies and Overall Architecture |
| 1. Functionalities |
| 1. Lessons Learned |

1. Application Domain:

* The application domain of this project is the financial sector, specifically focusing on investors, traders, and financial analysts who require timely and accurate financial data to make informed decisions. The financial markets are highly dynamic, with prices and trends changing rapidly. Consequently, having access to real-time data and sophisticated analytical tools is crucial for market participants to stay ahead.
* This platform aims to address the need for real-time data processing and visualization in the financial market. By providing real-time stock data, historical trends, financial metrics, and interactive visualizations, users can monitor market movements, identify trends, and make data-driven investment decisions. The platform's real-time capabilities ensure that users have the most current information at their fingertips, reducing the risk of relying on outdated data.
* The project is particularly relevant for day traders and high-frequency traders who make numerous trades within short timeframes. These traders require real-time data to capitalize on market opportunities quickly. Additionally, long-term investors and financial analysts can use the platform to track historical trends, analyse company performance, and develop investment strategies.

## 2. Description of the Data Sources

### Yahoo Finance

Yahoo Finance is a comprehensive data provider that offers a wide range of financial information, including real-time stock prices, historical data, financial statements, and market summaries. The Yahoo Finance API/Library serves as the primary data source for this project.

pip install yfinance --upgrade --no-cache-dir

# to install yfinance to get the data.

**Complexity**

* The Yahoo Finance API provides a diverse dataset encompassing various financial metrics and data types. This complexity requires robust data handling and processing capabilities to extract meaningful insights from the data.

**Size**:

* The data size varies significantly depending on the number of stocks tracked and the historical depth. Tracking hundreds of stocks with decades of historical data can result in gigabytes of data, necessitating efficient data storage and retrieval mechanisms.

**Speed**:

* Real-time data is streamed continuously, with high-frequency updates. The system must handle data ingestion and processing efficiently to ensure timely availability of real-time insights.

### Data Access

**API/Library**:

* Data from Yahoo Finance is accessed via HTTP requests, fetching JSON-formatted data. The API provides endpoints for various types of financial data, including current stock prices, historical data, and financial statements.

**Kafka Producers**:

* Real-time data is streamed into the system using Kafka producers. These producers fetch data from the Yahoo Finance API and publish it to Kafka topics, facilitating real-time data ingestion and processing.

### Additional Data Sources(Future Reference)

In addition to Yahoo Finance, other potential data sources can be integrated into the platform to enhance its capabilities. These may include:

1. **Alpha Vantage**:
   * Provides real-time and historical stock market data through a comprehensive API.
2. **IEX Cloud**:
   * Offers financial data including stock prices, market indices, and financial statements.
3. **Google Finance**:
   * Another source for real-time stock prices and market information.

These data sources can be accessed using their respective APIs, similar to the Yahoo Finance integration. By leveraging multiple data sources, the platform can provide more comprehensive and diverse financial insights.

## Technologies and Overall Architecture

A white paper with red writing on it

Description automatically generated

### Technologies

1. **Kafka**:
   * **Purpose**: Kafka is a distributed streaming platform used to build real-time data pipelines and streaming applications.
   * **Description**: Kafka is employed to ingest real-time financial, trading data from Yahoo Finance and stream it to other components in the system. It ensures reliable and scalable data transmission.
     + Here I make 4 Topics name as stock\_data (intraday data), historical data, financial data, real\_time\_data I am getting my data from here and then store in them as a dictionary and then flatten them according to need, I also correct the time frame here that is very crucial while we consume the data in flink.
   * **Libraries**: Confluent-Kafka for implementing Kafka producers in Python, enabling seamless data streaming.
     + **JSON (JavaScript Object Notation) is a file that is mainly used to store and transfer data mostly between a server & a web application. It is popularly used for representing structured data.**
2. **Flink**:
   * **Purpose**:

Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams.

* + **Description**: Apache Flink processes the streamed data in real-time, performing computations and transformations as needed. It supports complex event processing and real-time analytics.
    - **In flink I use table API and created a stream table environment where I define a input table(source table) name as stock\_data and it is reading from Kafka and also define a output table for the same and also define a function called as buy sell indicator that tell us when to buy, sell and hold and also put some jar files as kafka connector, jdbc connector.**

A screen shot of a computer code

Description automatically generated

* + **Libraries**: Apache Flink framework, with custom Flink jobs written in python for data transformation and analysis.

Pyflink.datastream for streamExecutionEnvironment: This import is used to create and configure the execution environment for data stream processing in Apache Flink, enabling the setup of the runtime context for stream applications.

Pyflink.table for streamTableEnviorment: This import is used to create and manage table environments for stream processing in Apache Flink, allowing you to execute SQL queries and transformations on streaming data.

Pathlib for path: This import is used to handle and manipulate filesystem paths in a platform-independent way, making it easier to read from or write to files and directories.

1. **PostgreSQL**:
   * **Purpose**: PostgreSQL is a powerful, open-source relational database management system.
   * **Description**: PostgreSQL stores the processed data, enabling efficient querying and retrieval. It provides robust data management capabilities and supports complex queries.
   * **Libraries**: Standard PostgreSQL setup and configuration, utilizing SQL for database interactions.
     1. I am checking SQL data through pgadmin so I make a file called servers.json that tells about password, user name , and PostgreSQL we are using.
2. **Streamlit**:
   * **Purpose**: Streamlit is an open-source app framework for creating interactive web applications.
   * **Description**: Streamlit is used to create the frontend application for data visualization and user interaction. It allows for rapid development of data-driven applications with minimal effort and here I am consuming data from two sources one is pgadmin and second one is kafka consumer. I consume data from pgadmin because of real time data and from kafka consumer I consume data for historical and financial data to implement some strategies that learn from past data.
   * **Libraries**: Streamlit library in Python, along with data visualization libraries such as matplotlib and plotly.

*The sys and os libraries access system-specific parameters and interact with the operating system. streamlit as st creates interactive web apps, while pandas as pd manipulates and analyzes data with DataFrames. plotly.graph\_objects as go makes interactive plots and charts. argparse parses command-line arguments, and datetime with dateutil.tz handles dates, times, and time zones. psycopg2 interacts with PostgreSQL databases, and streamlit\_javascript integrates JavaScript for advanced frontend features. The time module provides time-related functions, json.loads parses JSON strings, confluent\_kafka.Consumer consumes Kafka messages, and threading supports concurrent code execution.*

1. **pgAdmin**:
   * **Purpose**: pgAdmin is a management tool for PostgreSQL.
   * **Description**: pgAdmin provides an interface to manage PostgreSQL, allowing for database configuration, querying, and maintenance. It simplifies database management tasks.
   * **Libraries**: pgAdmin web interface, configured to connect with the PostgreSQL instance.

### System Architecture

The system architecture consists of several interconnected components, each playing a crucial role in the data processing pipeline. The architecture ensures seamless data flow from data ingestion to visualization.

**Components**:

1. **Kafka Producer**: Fetches real-time data from Yahoo Finance and streams it to Kafka topics.
2. **Kafka**: Serves as the messaging system, facilitating real-time data streaming between producers and consumers.
3. **Flink Processor**: Consumes data from Kafka, processes it in real-time, and sends the processed data to PostgreSQL.
4. **PostgreSQL**: Stores the processed data, enabling efficient querying and retrieval.
5. **pgAdmin**: Provides a web interface for managing the PostgreSQL database.
6. **Streamlit App**: Retrieves data from PostgreSQL and displays it on an interactive dashboard.

(part of streamlit application) provider.py defines the DataProvider class, which plays a crucial role in the application by consuming real-time stock indicators from a Kafka topic (indicators). This class is designed to run in a separate thread, ensuring continuous data retrieval. It maintains an up-to-date dataset of stock indicators, which can be accessed by the application to display real-time financial data to users. The use of thread-safe mechanisms ensures data consistency and integrity.

1. **Kafka Consumer**: Here I am consuming historical data and financial data directly from producer file and then directly using them in streamlit application.

data\_processor.py focuses on processing raw financial and historical data from Kafka topics (historical data and financial data). It sets up Kafka consumers for these topics, processes incoming messages by adding timestamps and specific processing for financial fields, and then produces the processed data to new Kafka topics (historical\_data\_processed and financial\_data\_processed). This processed data is cleaner and ready for consumption by the Streamlit app.

**Not Directly Interact**:

* data\_processor.py and streamlit\_app.py do not call each other or exchange data directly through function calls or shared variables.
* Instead, they interact indirectly through Kafka topics. data\_processor.py puts processed data onto Kafka topics, and streamlit\_app.py reads from these topics.

By using Kafka topics as intermediaries, data\_processor.py and streamlit\_app.py maintain a clean separation of concerns. This allows for modularity and scalability, as each component can be developed, tested, and scaled independently.

### Workflow

**Data Ingestion**:

1. **Step 1**: Kafka producers fetch real-time data from Yahoo Finance using the API.
2. **Step 2**: The fetched data is streamed to Kafka topics for further processing.

**Data Processing**:

3. **Step 3**: Apache Flink consumes data from Kafka topics and processes it in real-time.

4. **Step 4**: Flink performs necessary computations and transformations on the data.

5. **Step 5**: Processed data is sent to PostgreSQL for storage.

**Data Visualization**:

6. **Step 6**: The Streamlit app retrieves processed data from PostgreSQL.

7. **Step 7**: The Streamlit app displays real-time data visualizations on an interactive dashboard.

8. **Step 8**: Users interact with the dashboard to analyze financial data and gain insights.

**Database Management**:

9. **Step 9**: pgAdmin is used to manage the PostgreSQL database, including configuring connections, querying data, and maintaining the database.

## 4. Functionalities

### Kafka Producer

A screenshot of a computer

Description automatically generated

A screenshot of a computer

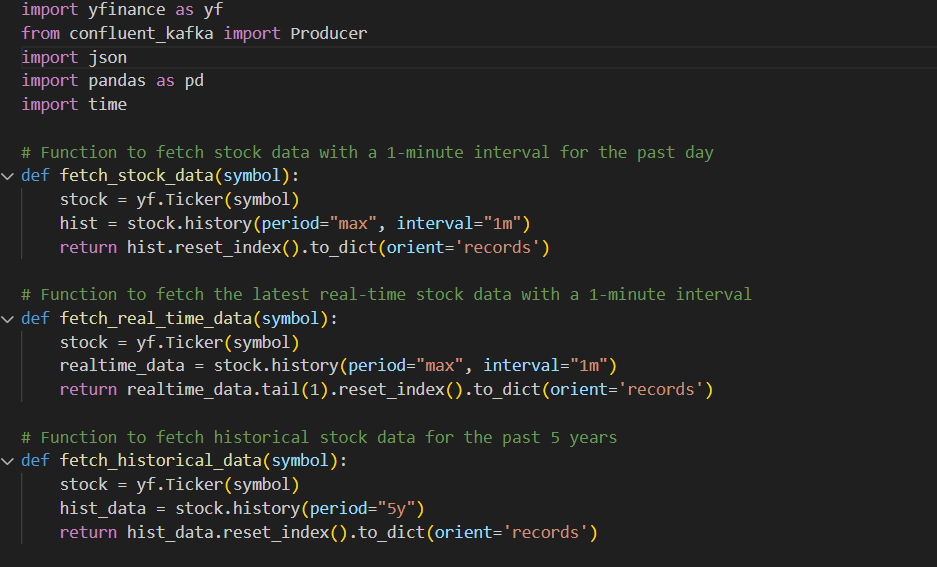
Description automatically generated

**Function**: The Kafka producer is responsible for collecting real-time financial data from Yahoo Finance and streaming it to Kafka topics. The producer fetches data using the Yahoo Finance API and publishes it to Kafka, ensuring continuous data flow.

**Implementation**:

* **Programming Language**: Python
* **Libraries**: Confluent-python for Kafka integration, yfinance for fetching data from Yahoo Finance
* **Scripts**: Custom Python scripts to handle data fetching and streaming

**Example Code**:



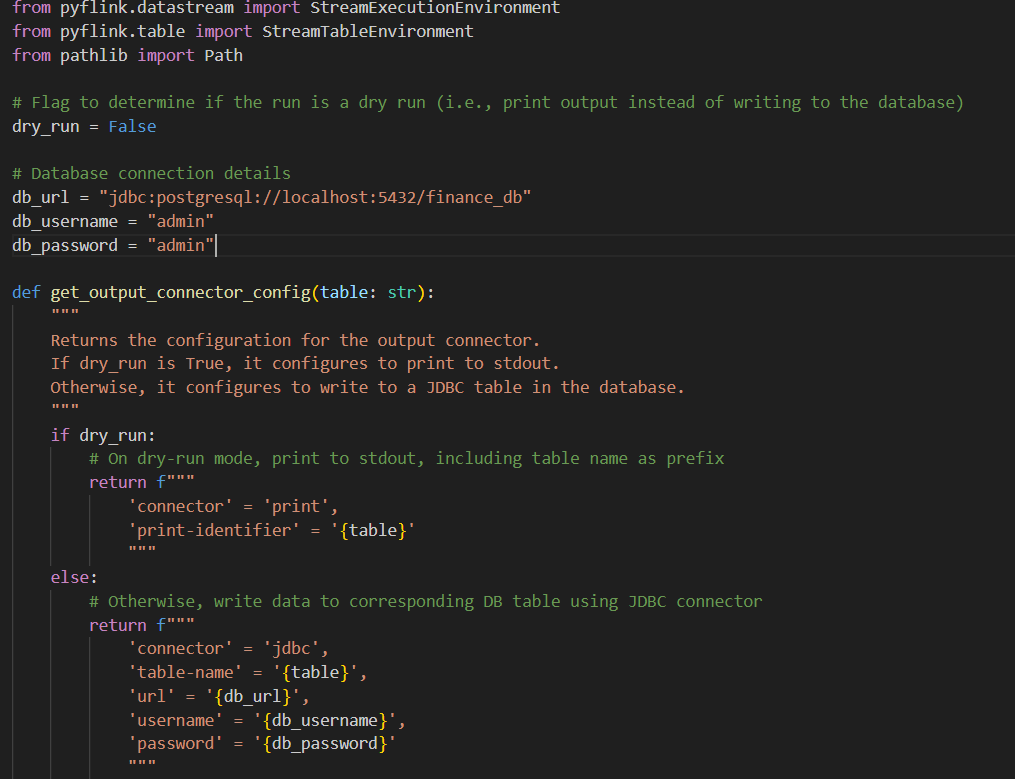
### Flink Processor

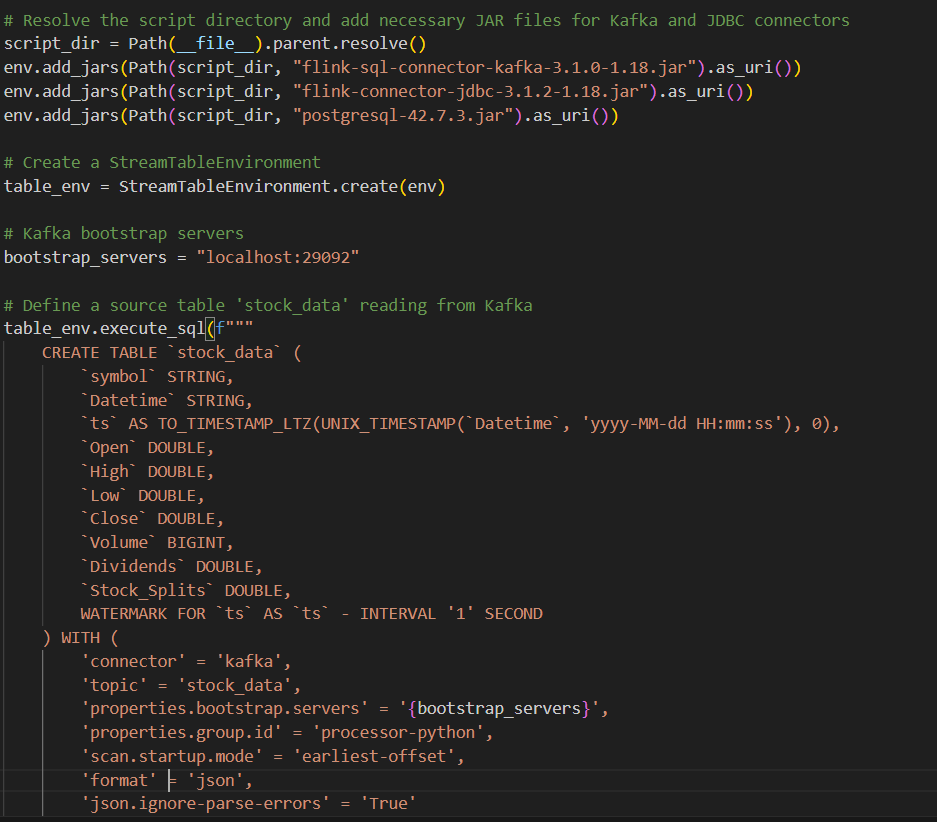
**Function**: The Flink processor consumes data from Kafka topics, processes it in real-time, and performs necessary computations. The processed data is then sent to PostgreSQL for storage.

**Implementation**:

* **Programming Language**: Python
* **Libraries**: Apache Flink for real-time data processing

**Example Code**:





A screenshot of a computer

Description automatically generated

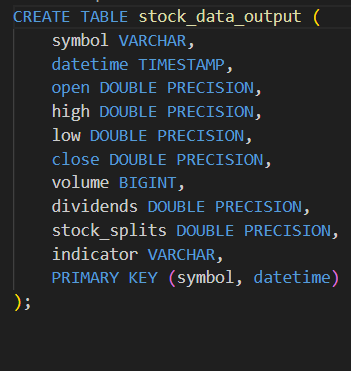
### PostgreSQL Database

**Function**: The PostgreSQL database stores processed data for persistence and future retrieval. It enables efficient querying and data management.

**Implementation**:

* **Database Management System**: PostgreSQL
* **Configuration**: Standard PostgreSQL setup with necessary schemas for storing financial data

**Example SQL Schema**:



### Streamlit Application

**Function**: The Streamlit app provides an interactive frontend for real-time data visualization. Users can interact with the dashboard to analyze financial data and gain insights.

**Implementation**:

* **Programming Language**: Python
* **Libraries**: Streamlit for web application, matplotlib and plotly for data visualization

**Example Code**:

import streamlit as st

import pandas as pd

import psycopg2

# Connect to PostgreSQL

conn = psycopg2.connect(

host="localhost",

database="financial\_data",

user="admin",

password="admin"

)

# Query data

df = pd.read\_sql\_query("SELECT \* FROM stock\_data", conn)

# Display data

st.title('Real-Time Financial Data Analysis')

st.write(df)

# Plot data

st.line\_chart(df[['timestamp', 'close']].set\_index('timestamp'))

A screenshot of a computer

Description automatically generated

### pgAdmin Interface

**Function**: pgAdmin provides an interface for managing the PostgreSQL database. It allows users to configure connections, query data, and perform database maintenance.

**Implementation**:

* **Tool**: pgAdmin

## 5. Lessons Learned

### What Worked

* **Integration**: The integration of multiple technologies (Kafka, Flink, PostgreSQL, Streamlit) worked seamlessly to create a cohesive real-time data analysis platform.
* **Real-Time Processing**: Efficient real-time data processing using Kafka and Flink enabled timely and accurate financial insights.
* **Interactive Visualization**: The use of Streamlit allowed for the creation of an interactive and user-friendly dashboard, enhancing the user experience.

### What Did Not Work

* **Flink Setup**: Encountered issues with setting up and configuring Flink, which required additional troubleshooting and resources. This highlighted the importance of thorough documentation and community support for resolving setup challenges.
* **Data Latency**: Initial latency issues with real-time data streaming were observed. These issues were resolved through optimization of data ingestion and processing pipelines.
* **Thread Issue/value mismatch:** The error Value Error: Length of values (9038) does not match length of index (9046) indicates that the processed data from Kafka has inconsistent lengths. This happens when the data you are trying to put into the Data Frame has different lengths for different columns or rows. You should ensure all columns have the same number of entries. This is for my historical and financial data mostly financial data I am facing this issue for.

### Improvements

* **Scalability**: Enhance the scalability of the system to handle larger data volumes and higher update frequencies. This could involve optimizing Kafka and Flink configurations, as well as scaling the infrastructure and this will also solve my value mismatch error and helpful to show financial data too and make some investing strategies too in upcoming future.
* **User Experience**: Further refine the Streamlit dashboard to improve user experience. This includes adding more detailed visualizations, interactive elements, and user customization options.
* **Documentation**: Improve project documentation to facilitate easier setup and maintenance. Comprehensive documentation will help new users understand the system architecture, setup procedures, and troubleshooting steps.

### Conclusion

The development and implementation of a real-time financial data analysis platform, detailing the application domain, data sources, technologies, system architecture, functionalities, and lessons learned. The project leverages robust technologies to provide timely and accurate financial insights, aiding users in making informed investment decisions.

By integrating real-time data streaming with Kafka, real-time processing with Flink, and interactive visualization with Streamlit, the platform ensures that users have access to the most current financial information. This comprehensive approach addresses the needs of various market participants, from day traders to long-term investors and financial analysts.

The project's success demonstrates the potential of combining advanced data technologies to create powerful and user-friendly financial analysis tools. Future improvements in scalability, user experience, and documentation will further enhance the platform's capabilities and usability, making it an invaluable resource for financial decision-making.