**Use Case Documentations:**

Financial analysis refers to the process of evaluating and interpreting financial data to gain insights into the financial performance, stability, and prospects of an individual or organization. It involves examining financial statements, ratios, trends, and other financial metrics to assess the financial health, profitability, liquidity, solvency, and efficiency of a business. Financial analysis helps stakeholders, including investors, creditors, managers, and analysts, allocate resources effectively, manage risks, and evaluate the overall financial viability of an entity. Financial analysis is very crucial and can often make or break organizations. It allows organizations to make informed business decides like investment opportunities, assess financial performance, and determine the feasibility of projects or initiatives. This is especially necessary in today’s volatile economic condition where you should be able to identify and assess risks associated with investments, operations, and other financial decisions. It allows companies and analysts to draw up risk management strategies and protect their own financial well-being. However, a key thing to note about the financial landscape is its constantly shifting nature that requires constant monitoring and storing of data in order to analyze it but this can only take you so far in terms of predicting future trends of the market and optimizing your decision-making process. This is where streaming analytics comes in which provides with the ability to analyse data as they are generated making predictions faster and as up to date as possible leading to more safer decisions and strategies. The following are some of the applications of streaming analytics in financial analysis:

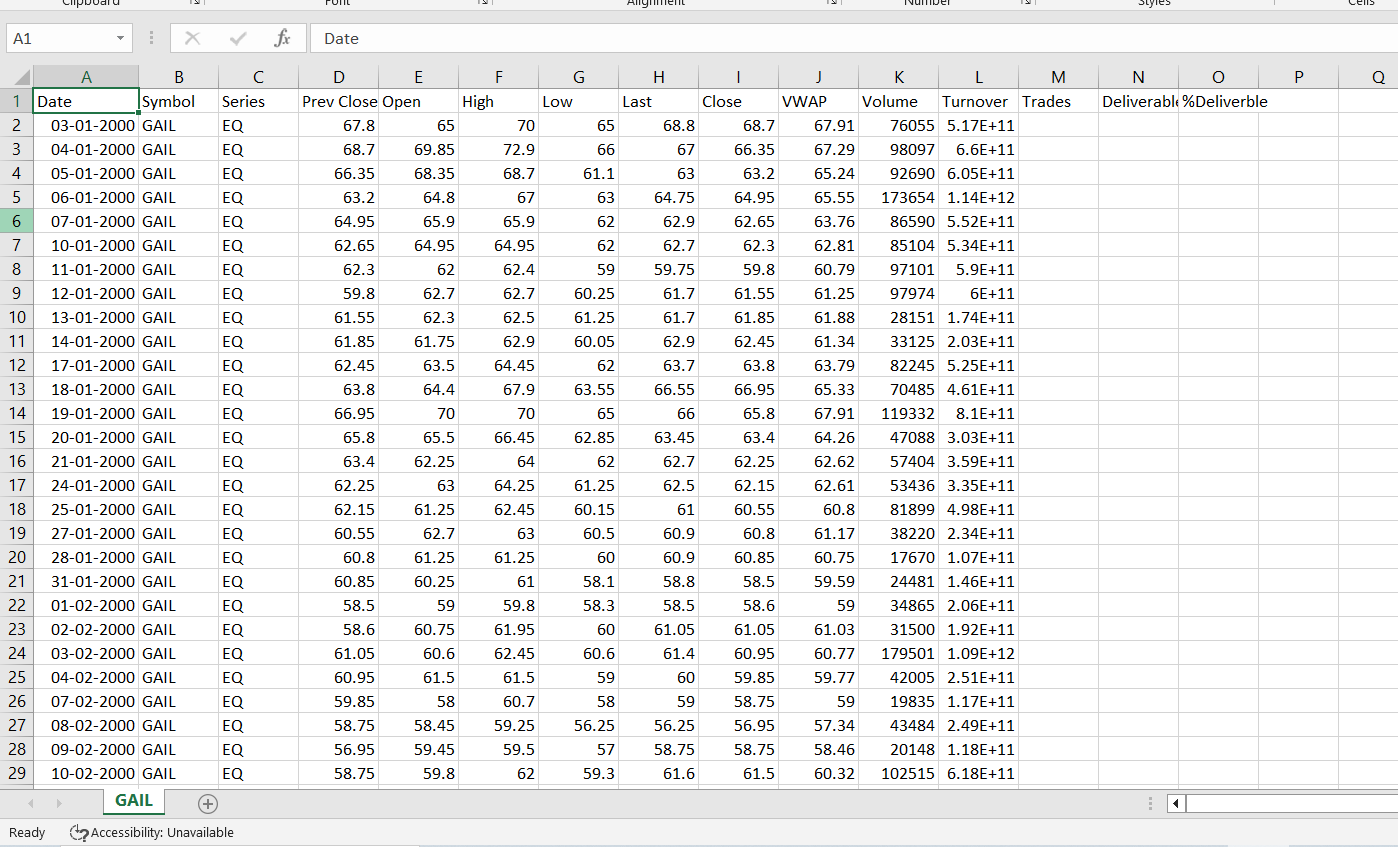
* Real-Time Market Data Analysis: Streaming analytics processes and analyzes real-time market data, including stock prices, currency rates, and market indexes. It enables traders and financial analysts to monitor market movements, detect patterns, and make informed trading decisions.
* Algorithmic Trading: Streaming analytics is crucial for algorithmic trading, where real-time data analysis and automated trading strategies are employed. It makes it possible to recognise trading signals, carry out trades, and manage risk in milliseconds or microseconds.
* Fraud Detection and Risk Management: Streaming analytics helps discover fraudulent transactions, unauthorized access, and suspicious behaviours in real-time. It applies advanced analytics and machine learning algorithms to evaluate transactional trends, detect abnormalities, and minimise risks.
* Portfolio Optimization: Streaming analytics allows real-time portfolio monitoring and optimization by analyzing market data, portfolio performance, and risk metrics. It promotes the identification of opportunities, rebalancing techniques, and risk-adjusted decision-making.
* News and Sentiment Analysis: Streaming analytics assesses market sentiment and sentiment-driven price movements in real-time by analysing news feeds, social media sentiment, and other textual data sources. It helps financial analysts gauge market sentiment, anticipate market reactions, and adjust investment strategies accordingly.
* Monitoring for Compliance and Regulatory Requirements: Using streaming analytics, financial institutions can monitor transactions and events for regulatory compliance. It provides real-time identification of suspicious activity, fraud, money laundering, and conformity to regulatory norms.
* Market Surveillance: Streaming analytics is applied for market surveillance to uncover market manipulation, insider trading, and other unlawful behaviours. It supports the ongoing real-time market integrity monitoring of trade activity by regulatory agencies and exchanges.
* High-Frequency Trading (HFT) Analysis: Streaming analytics plays a significant role in high-frequency trading, where rapid processing of market data is necessary. It facilitates the execution of complex trading algorithms, discovery of arbitrage opportunities, and latency-sensitive decision-making.

In my use case, I am using the NIFTY-50 Stock Market Data (2000 - 2021) dataset from Kaggle. The dataset consists of the price history and trading volumes of the fifty stocks in the index NIFTY 50 from NSE (National Stock Exchange) India. All datasets are at a day-level with pricing and trading values split across .cvs files for each stock along with a metadata file with some macro-information about the stocks itself. The data spans from 1st January, 2000 to 30th April, 2021.

**Dataset Link:**

[**https://www.kaggle.com/datasets/rohanrao/nifty50-stock-market-data**](https://www.kaggle.com/datasets/rohanrao/nifty50-stock-market-data)

**Dataset Screenshot:**



The data is streamed with the help of Apache Kafka which is an open source streaming analytics tool by Apache that that can handle large-scale, high-throughput data streams and provides features for real-time data processing. It is a producer-consumer architecture where we have a producer that will inject the data into the kafka broker server and a consumer that retrieves the data from the server and performs a function or analysis on the retrieved data. Apache Kafka consists of two servers the Kafka broker server and the Zookeeper server. The Kafka broker server is what connects the producer with the consumer and is responsible for the passing off the data itself as well as the metadata. The kafka broker consists of kafka topics which is a grouping of messages that is used to organize messages for production and consumption. A topic will consist of multiple partitions that can be identified using an unique offset in the partition log. The Zookeeper on the other hand is in charge of maintain the kafka broker clusters. It maintains a record of when a broker enters or leaves a cluster as well as is responsible for electing the leader broker for a cluster. Its functions include distributed configuration services, synchronization services, and a naming registry. The use case that I am working with consists of a producer python code that pushes a “.csv” file into the kafka server and a consumer python code that reads the “.csv file” from it and performs moving average on the the data as it is being streaming from the server. The moving average is done on the closing prices for two time periods, over 7 days and 30 days. The average value calculated is then plotted on a dynamically updating graph in order to track the trends of the shifting moving average.

Moving average is a statistical calculation used to analyze data points over a specific time period by smoothing out fluctuations and revealing underlying trends. It is commonly applied in time series analysis and financial analysis. The moving average is calculated by taking the average value of a set of data points within a defined window or period. The window can be a fixed number of periods or can be based on a specific time interval. With each new data point, the oldest data point is dropped from the calculation, and the newest data point is added. It help reduce noise in data, highlight trends, and provide a clearer picture of the underlying pattern. They are widely used to identify support and resistance levels, determine trend direction, generate trading signals, and assess the overall momentum of a data series.

**Advantages of a 7-Day Moving Average:**

1. Short-Term Trend Identification: A 7-day moving average provides a more sensitive indicator of short-term price trends compared to longer-term moving averages. It can capture price fluctuations and identify shorter-term patterns in the data.
2. Immediate Response to Price Changes: A 7-day moving average reacts quickly to recent price changes, allowing traders and investors to respond promptly to market movements and adjust their strategies accordingly.
3. Enhanced Timing for Trading Decisions: By analyzing a 7-day moving average, traders can better time their entry or exit points in the market. It helps identify potential price reversals or short-term price trends for more timely and strategic trading decisions.

**Advantages of a 30-Day Moving Average:**

1. Smoothing Long-Term Price Trends: A 30-day moving average provides a smoother representation of long-term price trends. It filters out short-term noise and random fluctuations, providing a clearer view of the underlying direction of the market.
2. Identification of Long-Term Support and Resistance Levels: A 30-day moving average helps identify long-term support and resistance levels. It acts as a reference point to gauge the overall market sentiment and determine key levels where buying or selling pressure may be significant.
3. Reduced Impact of Daily Volatility: As a longer-term moving average, the 30-day average is less affected by daily price volatility. It helps investors focus on broader market trends and reduces the impact of short-term market noise, offering a more stable perspective on price movements.
4. Investment Strategy Assessment: A 30-day moving average helps evaluate the performance of investment strategies over a longer period. It provides insights into the sustainability of trends and helps identify potential shifts in market sentiment that may impact long-term investment decisions.

**Guide to installing:**

1. Go to <https://kafka.apache.org/downloads> to download kafka from the official website. Its an open-source tool so it’s free to install.
2. Download the scala 2.13 binary file from the page. As of writing this, the latest version of kafka is version 3.5.0.
3. Once you hit download, a zip file will be installed in your system. You will have to unzip the file with Winrar or any other achrives extractor tool.
4. After unzipping, you will need to find the server properties file within the config folder. Open the server properties in the editor of your choice and you ll need to find the “logs.dir” which specifies the location of server log files and replace it with the location where you have stores the kafka files.

For eg:

logs.dir = /tmp/kafka-logs (default)

logs.dir = c:/kafka/kafka-logs (location of your kafka files)

1. Similiarly, open the zookeeper properties files from the same config folder. And here we change the “dataDir”.

For eg:

dataDir = /tmp/zookeeper (default)

dataDir = c:/kafka/zookeeper-data (location of your kafka files)

1. Now your kafka broker server and zookeeper server are ready. You can start the server by opening the command prompt in the directory where the kafka files are stored and use the following commands.

For zookeeper server:

.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

For kafka broker server:

.\bin\windows\kafka-server-start.bat .\config\server.properties

Note: The zookeeper server must be started before the kafka broker server.

**Issues with Kafka:**

An issue I faced after using Kafka for a while was the broker server shutting down on its own every time, I tried to start it. I tried restarting the server as well as zookeeper server multiple teams but it did not seem to work and kept shutting down. After a little searching I found out the error was because all log dirs had failed and this led to the Kafa-Broker failing. I was able to resolve this error by deleting all the kafka logs and then starting the zookeeper and kafka server again. The command for deleting all the kafka data from the server is as follows for windows:

rmdir /s /q C:\tmp\kafka-logs

(Where tmp is the location of your kafka files)

refer to : <https://stackoverflow.com/questions/51644409/kafka-broker-fails-because-all-log-dirs-have-failed>

**Code Snippet:**

**Producer-**

from kafka import KafkaProducer

import csv

# Configure Kafka producer

producer = KafkaProducer(bootstrap\_servers='localhost:9092')

# Define the topic to which the messages will be sent

topic = 'test'

# Path to the CSV file

csv\_file\_path = 'D:\GAIL.csv'

# Read the CSV file and send each row as a message to Kafka

with open(csv\_file\_path, 'r') as file:

    reader = csv.reader(file)

    header = next(reader)  # Assuming the first row is the header row

    for row in reader:

        # Convert the CSV row to a string

        csv\_row\_string = ','.join(row)

        # Produce the CSV row as a message to Kafka

        producer.send(topic, value=csv\_row\_string.encode('utf-8'))

# Flush and close the Kafka producer

producer.flush()

producer.close()

**Consumer-**

import matplotlib.pyplot as plt

from kafka import KafkaConsumer

# Configure Kafka consumer

consumer = KafkaConsumer('test', bootstrap\_servers='localhost:9092')

# Initialize lists to store closing prices and moving averages

closing\_prices = []

moving\_averages\_7days = []

moving\_averages\_30days = []

# Create a figure and axes for the plot

fig, ax = plt.subplots()

# Continuously consume messages from Kafka

for message in consumer:

    # Decode the message value from bytes to string

    csv\_row\_string = message.value.decode('utf-8')

    # Print the CSV row

    print(csv\_row\_string)

    # Extract the closing price from the CSV row

    closing\_price = float(csv\_row\_string.split(',')[7])

    # Append the closing price to the list

    closing\_prices.append(closing\_price)

    # Calculate the moving averages

    if len(closing\_prices) >= 7:

        # Shifting 7 days moving average

        moving\_average\_7days = sum(closing\_prices[-7:]) / 7

        moving\_averages\_7days.append(moving\_average\_7days)

    if len(closing\_prices) >= 30:

        # Shifting 30 days moving average

        moving\_average\_30days = sum(closing\_prices[-30:]) / 30

        moving\_averages\_30days.append(moving\_average\_30days)

    # Clear the current plot

    ax.clear()

    # Plot the closing prices, 7 days moving average, and 30 days moving average

    ax.plot(closing\_prices, label='Closing Price')

    ax.plot(moving\_averages\_7days, label='7 Days Moving Average')

    ax.plot(moving\_averages\_30days, label='30 Days Moving Average')

    # Set labels and title

    ax.set\_xlabel('Time (in Days)')

    ax.set\_ylabel('Price (in Rupees)')

    ax.set\_title('Moving Averages')

    # Add legend

    ax.legend()

    # Draw the updated plot

    plt.draw()

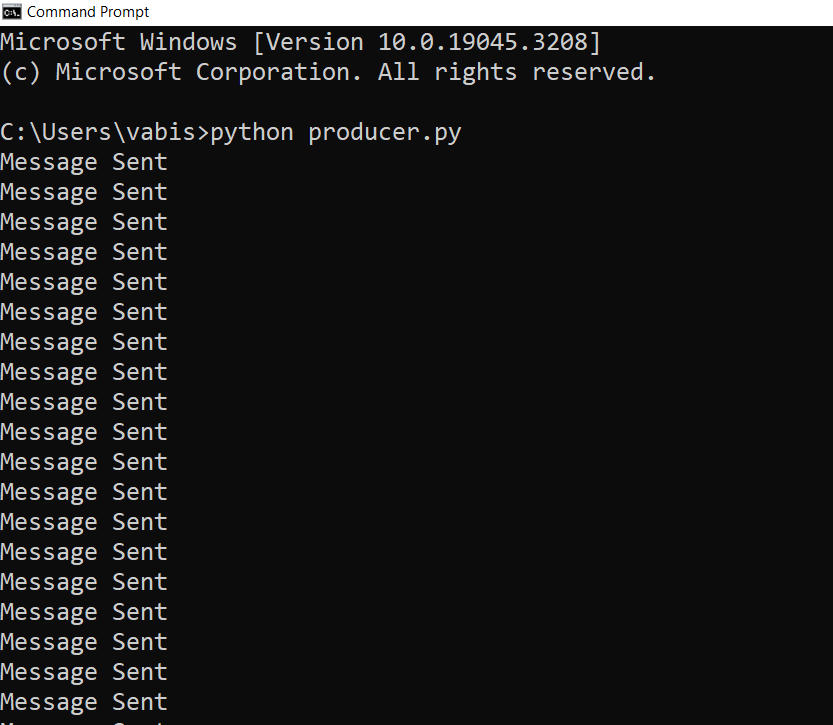
    plt.pause(0.01)

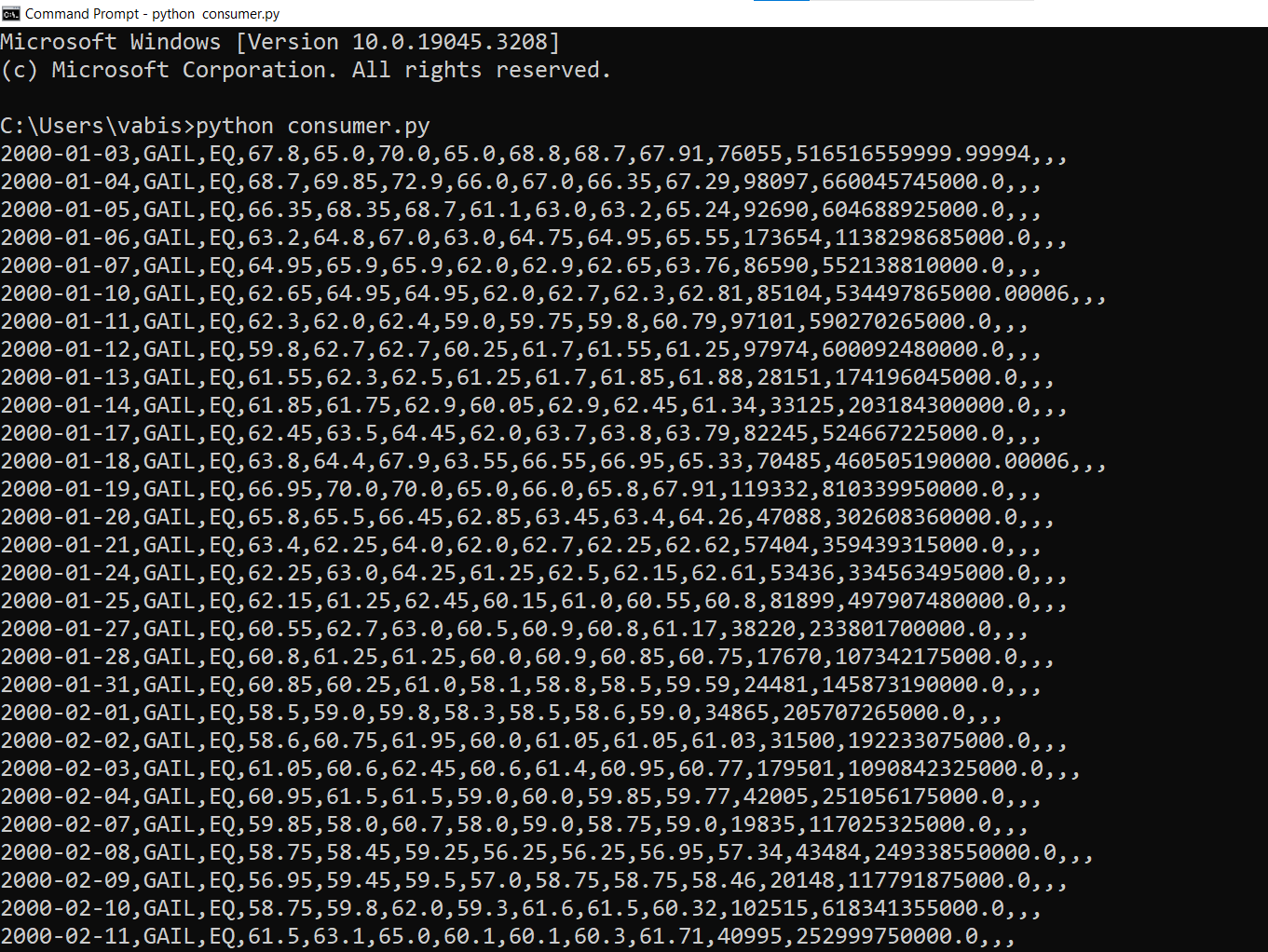
# Close the Kafka consumer

consumer.close()

**Output:**

**Producer.py-**



**Consumer.py**

Moving Average Graph:

