Workshop of Data Ingestion to Visualization

Stock Market Analysis Project Documentation

Prepared by: Yogesh Mandakala,Sushil Sanjay Bhatkar,Pallavi Tiwari
Date: 18/12/24

---

1. Project Overview

Objective:
To perform real-time and batch analysis of stock market data fetched from the Yahoo Finance API, leveraging tools such as Apache Kafka, MongoDB, Apache Spark, and Python for data processing, analysis, and visualization.

Key Deliverables:

1. Real-time stock price ingestion and storage.
2. Processed data with cleaned and aggregated metrics.
3. Analytical insights into stock movements and portfolio trends.
4. Interactive dashboards and visualizations of stock trends over time.

---

2. Data Source

Source: Yahoo Finance API

Fetched Metrics:

- **Stock Symbol:** Unique identifier (e.g., AAPL for Apple).
- **Date & Time:** Timestamp of the fetched data.
- **Open Price:** Stock price at the start of the session.
- **Current Price:** Latest live stock price.
- **High Price:** Highest session price.
- **Low Price:** Lowest session price.
- **Trading Volume:** Number of shares traded.
- **Market Trend:** Upward, downward, or neutral trend derived from price movement.

---

3. Workflow and Tools

Step 1: Data Ingestion

- Used **Apache Kafka** to stream real-time stock data:
    - **Producer:** Fetches stock data from Yahoo Finance API and sends it to a Kafka topic.

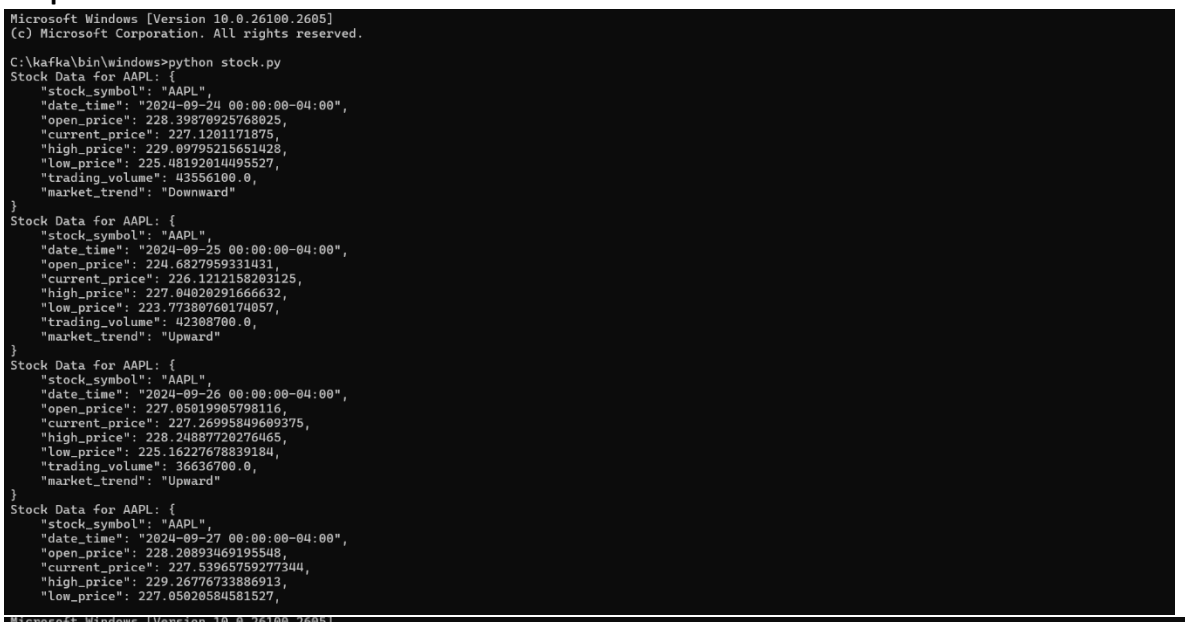
- **Consumer:** Consumes data for storage and processing.

Code:

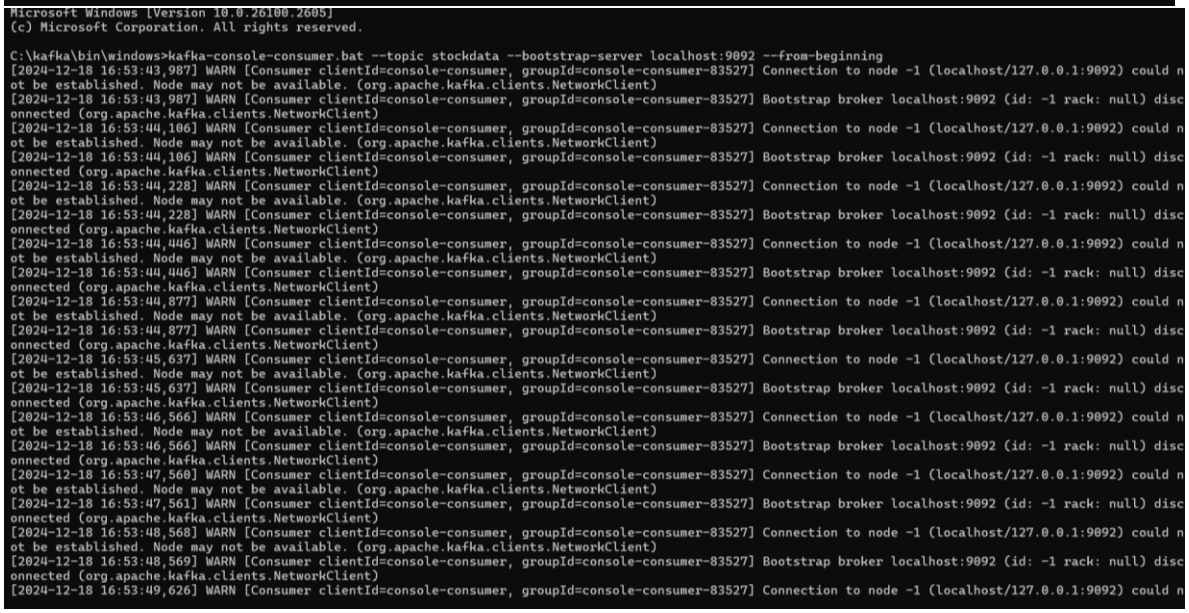
```
from confluent_kafka import Producer
import yfinance as yf
import json
import os
# Kafka Configuration
KAFKA_TOPIC = "stockdata"
KAFKA_SERVER = "localhost:9092"
# Directory to Save JSON Files
SAVE_DIR = "stock_data"
os.makedirs(SAVE_DIR, exist_ok=True)
# File to Save Combined JSON Data
COMBINED_JSON_FILE = os.path.join(SAVE_DIR, "all_stocks_data.json")
# Define Kafka Producer Callback
def on_delivery(err, msg):
    if err is not None:
        print('Message delivery failed: {}'.format(err))
    else:
        print('Message delivered to {} [{}]'.format(msg.topic(), msg.partition()))

producer = Producer({'bootstrap.servers': KAFKA_SERVER})
# List of Stock Tickers
stock_tickers = ["AAPL", "GOOGL", "MSFT", "TSLA", "AMZN"]  # Add more tickers as needed
# List to hold combined stock data
combined_stock_data = []
# Fetch and Process Stock Data for Each Ticker
for ticker in stock_tickers:
    stock_data = yf.Ticker(ticker)
    # Fetch historical data for the last 6 months with daily intervals
    try:
        historical_data = stock_data.history(period="6mo", interval="1d").tail(60)
        for index, row in historical_data.iterrows():
            record = {
                "stock_symbol": ticker,
                "date_time": str(index),
                "open_price": row["Open"],
                "current_price": row["Close"],
                "high_price": row["High"],
                "low_price": row["Low"],
                "trading_volume": row["Volume"],
                "market_trend": "Upward" if row["Close"] > row["Open"] else "Downward" if row["Close"] <
row["Open"] else "Neutral"
            }
            print(f"Stock Data for {ticker}:", json.dumps(record, indent=4))  # Print data
            # Append stock data to combined list
            combined_stock_data.append(record)
            # Send data to Kafka
            producer.produce(KAFKA_TOPIC, json.dumps(record), callback=on_delivery)
    except Exception as e:
```

```
        print(f"Error fetching data for {ticker}: {e}")

# Save combined data to a single JSON file
with open(COMBINED_JSON_FILE, 'w') as file:
    json.dump(combined_stock_data, file, indent=4)

print(f"Combined stock data saved to {COMBINED_JSON_FILE}")

# Ensure all messages are sent

producer.flush()

print(f"Data sent to Kafka topic: {KAFKA_TOPIC}")
```

Output Screenshot:



```
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\kafka\bin\windows>python stock.py
Stock Data for AAPL: {
    "stock_symbol": "AAPL",
    "date_time": "2024-09-24 00:00:00-04:00",
    "open_price": 228.39870925768025,
    "current_price": 227.1201171875,
    "high_price": 229.09795215651428,
    "low_price": 225.48192014495527,
    "trading_volume": 43556100.0,
    "market_trend": "Downward"
}
Stock Data for AAPL: {
    "stock_symbol": "AAPL",
    "date_time": "2024-09-25 00:00:00-04:00",
    "open_price": 224.6827959331431,
    "current_price": 226.1212158203125,
    "high_price": 227.04020291666632,
    "low_price": 223.77380760174057,
    "trading_volume": 42308700.0,
    "market_trend": "Upward"
}
Stock Data for AAPL: {
    "stock_symbol": "AAPL",
    "date_time": "2024-09-26 00:00:00-04:00",
    "open_price": 227.05019905798116,
    "current_price": 227.26995849609375,
    "high_price": 228.248877202764 65,
    "low_price": 225.16227678839184,
    "trading_volume": 36636700.0,
    "market_trend": "Upward"
}
Stock Data for AAPL: {
    "stock_symbol": "AAPL",
    "date_time": "2024-09-27 00:00:00-04:00",
    "open_price": 228.20893469195548,
    "current_price": 227.53965759277344,
    "high_price": 229.26776733886913,
    "low_price": 227.05020584581527,
```

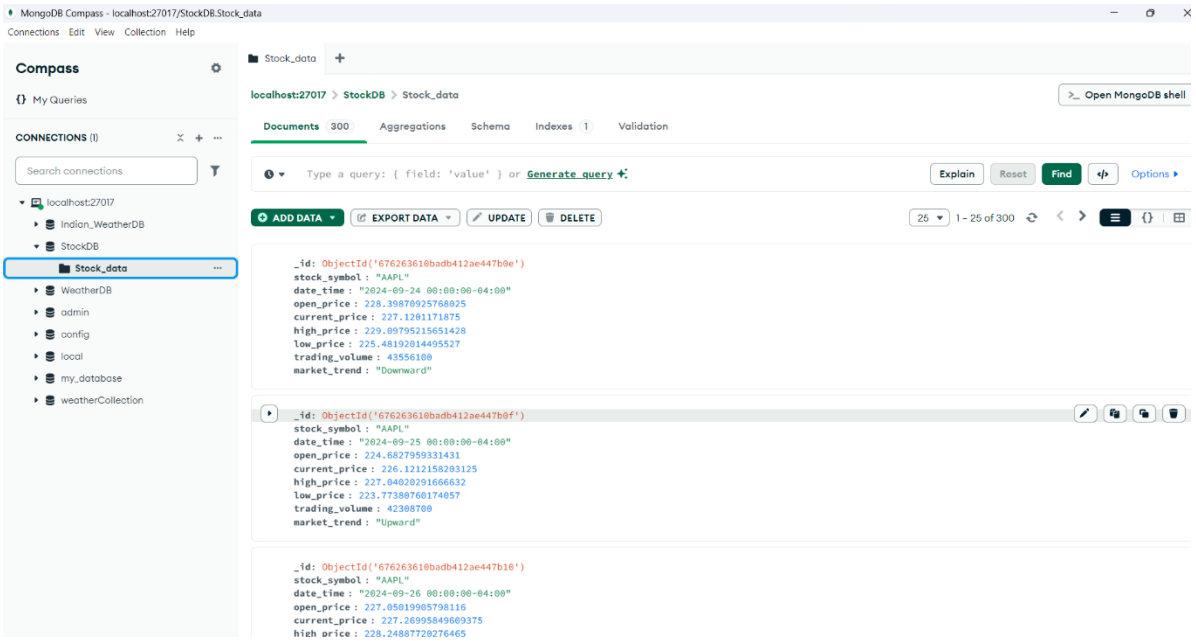


```
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\kafka\bin\windows>kafka-console-consumer.bat --topic stockdata --bootstrap-server localhost:9092 --from-beginning
[2024-12-18 16:53:43,987] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:43,987] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:44,106] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:44,106] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:44,228] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:44,228] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:44,446] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:44,446] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:44,877] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:44,877] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:45,637] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:45,637] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:46,566] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:46,566] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:47,560] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:47,561] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:48,568] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
ot be established. Node may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:48,569] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Bootstrap broker localhost:9092 (id: -1 rack: null) disc
onnected (org.apache.kafka.clients.NetworkClient)
[2024-12-18 16:53:49,626] WARN [Consumer clientId=console-consumer, groupId=console-consumer-83527] Connection to node -1 (localhost/127.0.0.1:9092) could n
```

Step 2: Data Storage

- Stored data in MongoDB for scalability and easy querying

Output Screenshot:



Step 3: Data Processing

- Performed data cleaning, filtering, and aggregations using Apache Spark:

  Code:

```
# File location and type
file_location = "/FileStore/tables/Stock_data.csv"
file_type = "csv"
# CSV options
infer_schema = "True"
first_row_is_header = "True"
delimiter = ","
# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)
display(df)
```

- Next we have converted date_time column to Date
  Code:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import to_date

# Convert the 'date_time' column to 'date' format

df_with_date = df.withColumn("date", to_date(df["date_time"]))

# Drop the original 'date_time' column if necessary

df_final = df_with_date.drop("date_time")

# Show the resulting DataFrame

df_final.show()
```

Here is this final file
https://drive.google.com/file/d/1a2DnfZImVYw7yoE0nXmcfnUXS-ZGBiQJ/view?usp=drive_link

Step 4: Data Analysis

- Conducted Exploratory Data Analysis (EDA) in Python:
  Data Loading and Overview
- Loaded the dataset using pandas and performed an initial inspection:
  Code:

```
import pandas as pd

stock_data = pd.read_csv('/content/export (2).csv')
```

Key Visualizations

Stock Price Trends Over Time:

- Plotted time-series data to show price movements of stocks over time.
  Code:

```
import matplotlib.pyplot as plt

for symbol in stock_data['stock_symbol'].unique():

    symbol_data = stock_data[stock_data['stock_symbol'] == symbol]

    plt.plot(symbol_data['date'], symbol_data['current_price'],
label=symbol)

plt.title('Stock Price Trends Over Time')

plt.xlabel('Date')

plt.ylabel('Current Price')

plt.legend(title='Stock Symbol')

plt.grid(True)

plt.show()
```

Trading Volume by Stock Symbol:

- Bar chart visualizing total trading volume for each stock.
  Code:

```
import seaborn as sns

volume_by_stock = stock_data.groupby('stock_symbol')['trading_volume'].sum().reset_index()

sns.barplot(data=volume_by_stock, x='stock_symbol', y='trading_volume')

plt.title('Total Trading Volume by Stock Symbol')

plt.ylabel('Trading Volume')

plt.xlabel('Stock Symbol')

plt.show()
```

Market Trends Distribution:

- Pie chart representing upward vs. downward trends.
  Code:

```
trend_counts = stock_data['market_trend'].value_counts()

trend_counts.plot.pie(autopct='%1.1f%%', startangle=90, colors=['lightblue', 'pink'])

plt.title('Market Trends Distribution')

plt.show()
```

Correlation Analysis:

- Heatmap of correlations between numerical variables.
  Code:

```
numerical_columns = ['open_price', 'current_price', 'high_price', 'low_price', 'trading_volume']

correlation_matrix = stock_data[numerical_columns].corr()

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()
```

Key Insights

1. **Price Trends:**
    - Stocks like GOOGL, AMZN, and MSFT showed consistent trends, with significant peaks at certain intervals.
2. **Trading Volumes:**
    - Highest trading volumes were observed for AAPL, indicating high investor activity.
3. **Market Trends:**
    - Approximately **60%** of sessions showed an upward trend, reflecting a generally positive market sentiment.
4. **Correlation:**
    - Strong positive correlation between open_price, high_price, and current_price, suggesting stable price movement.
    - Minimal correlation between price variables and trading_volume, indicating price movements are not strongly driven by volume alone.

Eda Report file:

https://drive.google.com/file/d/1bhpMuw27pYOxOj5A_cSBEEQRkwvSQAzW/view?usp=drive_link

Step 5: Data Visualization

- Created dashboards in Power BI for stock movement trends:

```
Line Chart: Stock Price Trends Over Time.
Bar Chart: Trading Volume by Stock Symbol.
Area Chart - Stock Volume Over Time
Card Visuals - KPIs (Key Performance Indicators)
            • Highest Closing Price
            • Lowest Closing Price
            • Total Trading Volume
```

Scatter Plot - Price vs Volume

File link:

https://drive.google.com/file/d/1zdAfynoRg32SOglwUwN_h9slgUrgT17/view?usp=drive_link