A
Project Report
On

"US Market Data Analysis"

Prepared by
Tushar Yadav | tushar.yadav@csu.fullerton.edu
Dhruti Dilipbhai Patel | dhrutipatel@csu.fullerton.edu

A Report Submitted to

Tseng-Ching James Shen, PhD Department of Computer Science

College of Engineering and Computer Science (ECS)

California State University, Fullerton (CSUF)

for the

Fulfillment of the Requirements for

CPSC 531 Advance Database Management

California State University, Fullerton CA - 92831
December 2023

TABLE OF CONTENTS

INTRODUCTION

Problem Statement

Analyzing Market Data from Alpha Vantage API for Investment Insights

In the dynamic landscape of financial markets, the vast and continuous flow of data presents a valuable opportunity for making data-driven investment decisions. The objective of this advanced database project is to harness the wealth of historical and real-time market data to empower investors with sophisticated tools for both real-time analysis and predictive analysis of the market.

Real-time analysis involves the scrutiny of current market conditions to facilitate immediate decision-making. This phase of the project will focus on developing mechanisms to monitor livestock prices, track market indices, and respond promptly to emerging events or news that can influence financial markets.

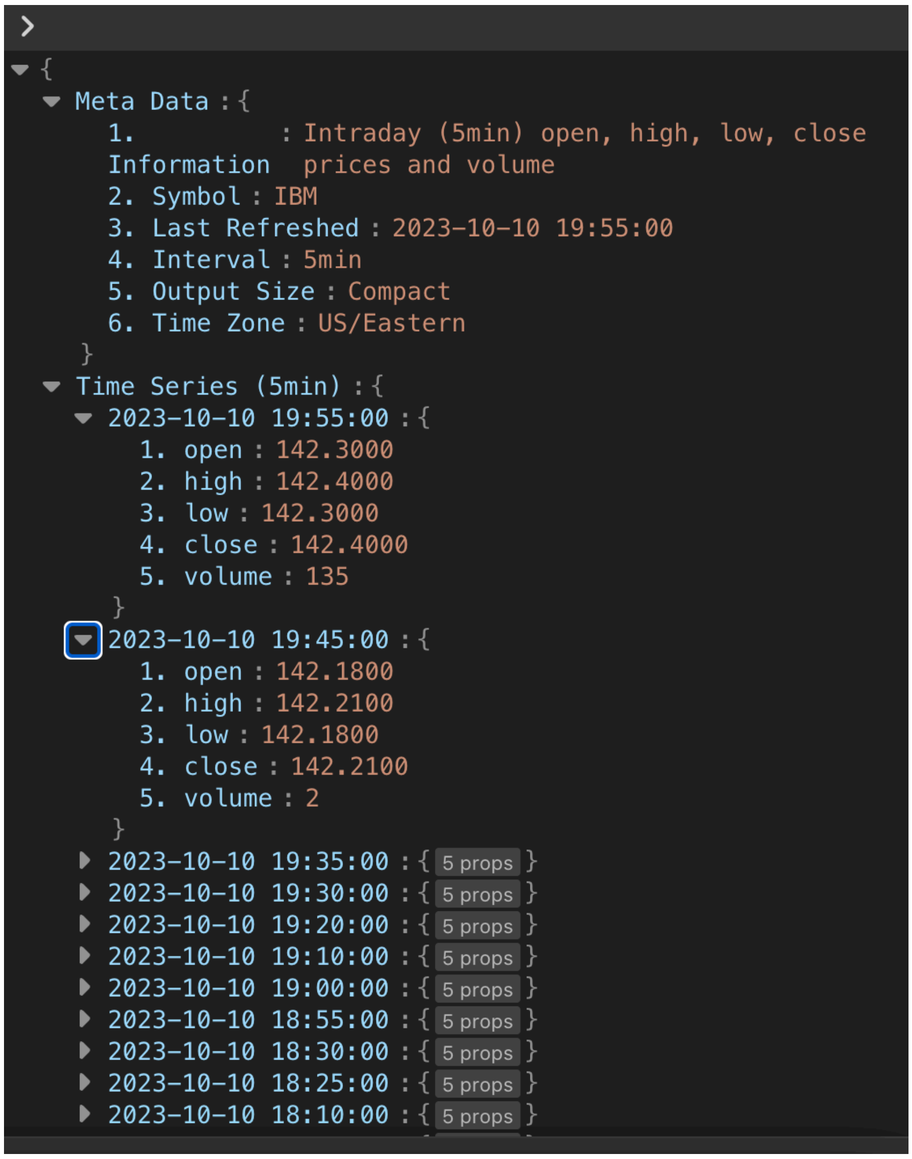
Data Description

Alpha Vantage Realtime Market Dataset is a valuable resource for investors and financial analysts seeking up-to-the-minute insights into global financial markets. Alpha Vantage, a prominent financial data provider, offers a real-time market dataset that covers a diverse range of financial instruments, including stocks, indices, and cryptocurrencies.

This dataset is particularly noteworthy for its:

1. **Real-Time Updates:**
   Alpha Vantage's Realtime Market Dataset provides users with immediate access to live market data, ensuring that investors have the latest information at their fingertips.

Sample Dataset:



```
{
  Meta Data : {
    1. Information : Intraday (5min) open, high, low, close prices and volume
    2. Symbol : IBM
    3. Last Refreshed : 2023-10-10 19:55:00
    4. Interval : 5min
    5. Output Size : Compact
    6. Time Zone : US/Eastern
  }
  Time Series (5min) : {
    2023-10-10 19:55:00 : {
      1. open : 142.3000
      2. high : 142.4000
      3. low : 142.3000
      4. close : 142.4000
      5. volume : 135
    }
    2023-10-10 19:45:00 : {
      1. open : 142.1800
      2. high : 142.2100
      3. low : 142.1800
      4. close : 142.2100
      5. volume : 2
    }
    2023-10-10 19:35:00 : { 5 props }
    2023-10-10 19:30:00 : { 5 props }
    2023-10-10 19:20:00 : { 5 props }
    2023-10-10 19:10:00 : { 5 props }
    2023-10-10 19:00:00 : { 5 props }
    2023-10-10 18:55:00 : { 5 props }
    2023-10-10 18:30:00 : { 5 props }
    2023-10-10 18:25:00 : { 5 props }
    2023-10-10 18:10:00 : { 5 props }
```

2. **Integration with Alpha Vantage API:**
   The dataset seamlessly integrates with Alpha Vantage's API, providing a streamlined and efficient means of retrieving real-time market data. This integration enhances accessibility and facilitates the incorporation of live market information into custom applications or analytical tools.

3. **Historical Data Access:**
   In addition to real-time updates, Alpha Vantage offers historical market data, allowing users to conduct in-depth analyses and back testing. This historical context enhances the dataset's utility for both short-term decision-making and long-term strategic planning.

Investors, financial analysts, and developers can leverage Alpha Vantage's Realtime Market Dataset to stay ahead in the fast-paced world of finance. Whether for real-time decision support or comprehensive market research, this dataset serves as a valuable tool for those navigating the complexities of global financial market.

Tool and Technology Used:

- Python
- Crontab
- Azure Blob Storage
- Kafka
- Plotly

FUNCTIONALITIES

1. **Automated Data Extraction:**
    - Automated extraction of financial market data at specific intervals.
    - Customizable cron jobs to manage the timing and frequency of data extraction.

2. **Data Normalization and Transformation:**
    - Python scripts to cleanse, normalize, and transform raw data into a consistent format suitable for analysis.
    - This structured data is then fed to the Kafka topic for consumption.

3. **Real-time Data Streaming:**
    - Real-time streaming capabilities via Kafka, ensuring minimal latency in data transmission.
    - Scalable infrastructure to handle high volumes of data without bottlenecking.

4. **Efficient Data Processing:**
    - Distributed data processing using Spark, enabling fast analysis of large datasets.
    - Integration of analytical libraries like Plotly and PySpark for advanced data visualization and statistical modeling.

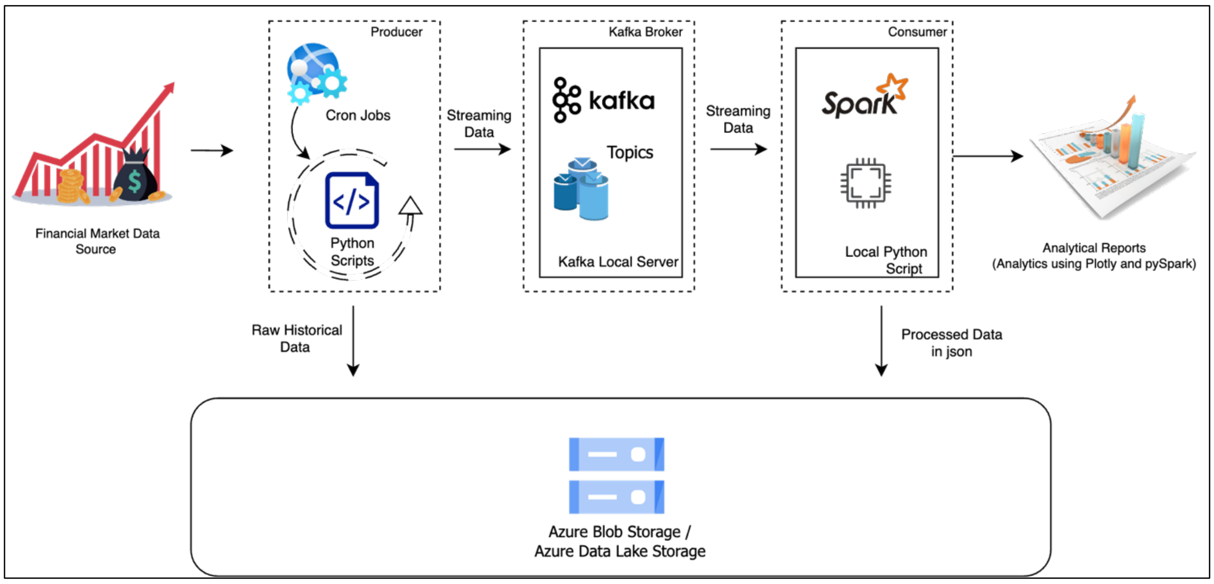
5. **Interactive Analysis:**
    - Developed Python Spark script that utilizes the Plotly Dash library to construct an interactive dashboard, showcasing all the plots for a comprehensive analysis.

6. **Data Storage:**
    - Secure and redundant storage solutions using Azure Blob Storage and Azure Data Lake Storage.
    - Structured storage of processed data in JSON format, optimizing accessibility and query efficiency for the Historical data.

ARCHITECTURE & DESIGN

High-Level View of the Data Pipeline of the Project

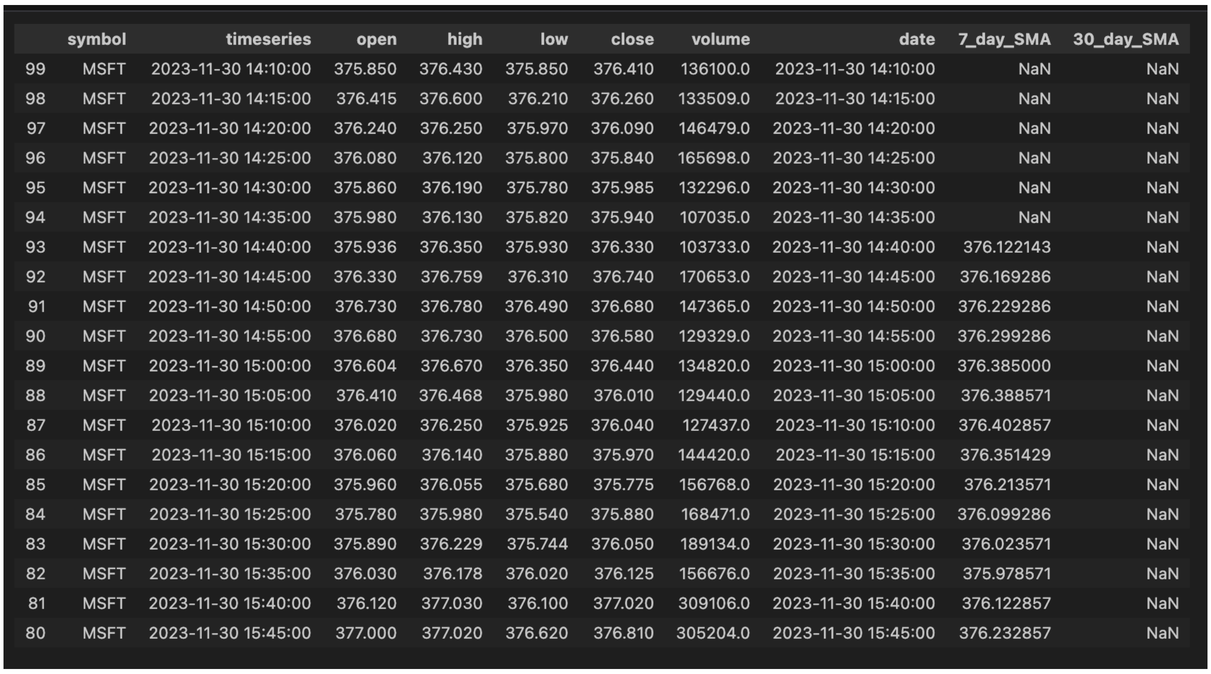
The image outlines a data pipeline flowchart, starting with Financial Market Data being processed by Cron Jobs and Python Scripts. This streaming data is then channeled into Kafka Topics hosted on a Kafka local server. Subsequently, the data is analyzed using Spark in local python script, producing analytical dashboard with Plotly and PySpark. The end products are historical processed data in JSON format, stored in Azure Blob Storage or Azure Data Lake Storage.



Methodology:

1. **Data Collection:**
   - Financial market data is sourced from Alpha Vantage API.
   - Cron jobs are scheduled to automatically trigger Python scripts, which extract the Intraday Data (5 min) at specified intervals.

2. **Data Ingestion and Processing:**
   - The Python scripts sanitize and format the raw data into a streamable structure.
   - This streaming data is then published to predefined topics on a Kafka local server, which acts as a message broker.



	symbol	timeseries	open	high	low	close	volume	date	7_day_SMA	30_day_SMA
99	MSFT	2023-11-30 14:10:00	375.850	376.430	375.850	376.410	136100.0	2023-11-30 14:10:00	NaN	NaN
98	MSFT	2023-11-30 14:15:00	376.415	376.600	376.210	376.260	133509.0	2023-11-30 14:15:00	NaN	NaN
97	MSFT	2023-11-30 14:20:00	376.240	376.250	375.970	376.090	146479.0	2023-11-30 14:20:00	NaN	NaN
96	MSFT	2023-11-30 14:25:00	376.080	376.120	375.800	375.840	165698.0	2023-11-30 14:25:00	NaN	NaN
95	MSFT	2023-11-30 14:30:00	375.860	376.190	375.780	375.985	132296.0	2023-11-30 14:30:00	NaN	NaN
94	MSFT	2023-11-30 14:35:00	375.980	376.130	375.820	375.940	107035.0	2023-11-30 14:35:00	NaN	NaN
93	MSFT	2023-11-30 14:40:00	375.936	376.350	375.930	376.330	103733.0	2023-11-30 14:40:00	376.122143	NaN
92	MSFT	2023-11-30 14:45:00	376.330	376.759	376.310	376.740	170653.0	2023-11-30 14:45:00	376.169286	NaN
91	MSFT	2023-11-30 14:50:00	376.730	376.780	376.490	376.680	147365.0	2023-11-30 14:50:00	376.229286	NaN
90	MSFT	2023-11-30 14:55:00	376.680	376.730	376.500	376.580	129329.0	2023-11-30 14:55:00	376.299286	NaN
89	MSFT	2023-11-30 15:00:00	376.604	376.670	376.350	376.440	134820.0	2023-11-30 15:00:00	376.385000	NaN
88	MSFT	2023-11-30 15:05:00	376.410	376.468	375.980	376.010	129440.0	2023-11-30 15:05:00	376.388571	NaN
87	MSFT	2023-11-30 15:10:00	376.020	376.250	375.925	376.040	127437.0	2023-11-30 15:10:00	376.402857	NaN
86	MSFT	2023-11-30 15:15:00	376.060	376.140	375.880	375.970	144420.0	2023-11-30 15:15:00	376.351429	NaN
85	MSFT	2023-11-30 15:20:00	375.960	376.055	375.680	375.775	156768.0	2023-11-30 15:20:00	376.213571	NaN
84	MSFT	2023-11-30 15:25:00	375.780	375.980	375.540	375.880	168471.0	2023-11-30 15:25:00	376.099286	NaN
83	MSFT	2023-11-30 15:30:00	375.890	376.229	375.744	376.050	189134.0	2023-11-30 15:30:00	376.023571	NaN
82	MSFT	2023-11-30 15:35:00	376.030	376.178	376.020	376.125	156676.0	2023-11-30 15:35:00	375.978571	NaN
81	MSFT	2023-11-30 15:40:00	376.120	377.030	376.100	377.020	309106.0	2023-11-30 15:40:00	376.122857	NaN
80	MSFT	2023-11-30 15:45:00	377.000	377.020	376.620	376.810	305204.0	2023-11-30 15:45:00	376.232857	NaN

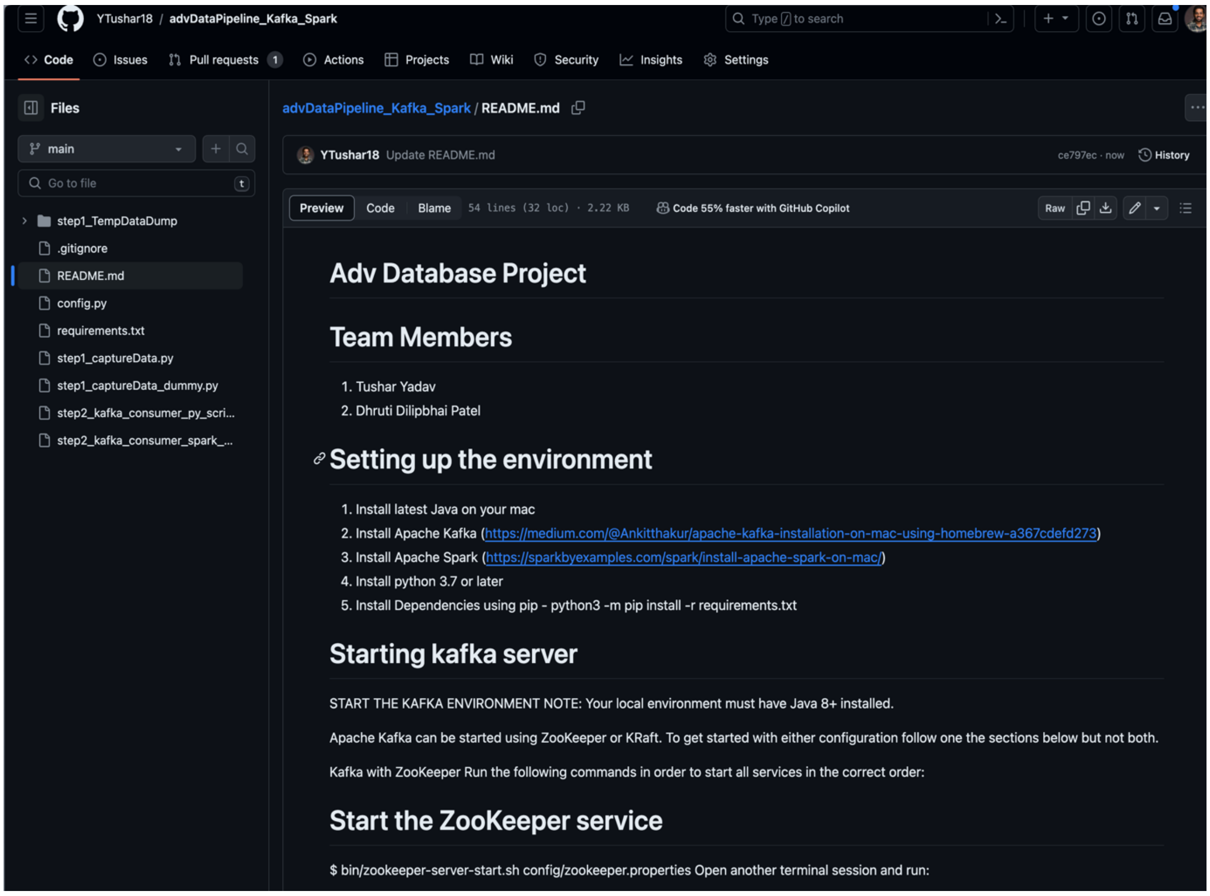
3. **Data Analysis:**
   - Local Python script with Spark sessions consume the data from Kafka topics.
   - Analytical processing is performed using PySpark for large-scale data processing and Plotly for visualization, transforming the raw data into insightful analytical reports.
   - Engineered Python Spark script to develop a dashboard using Plotly dash library.

4. **Data Storage and Management:**
   - The processed data is then serialized into JSON format for interoperability and ease of access.
   - Finally, the JSON files are stored securely in Azure Blob Storage or Azure Data Lake Storage, ensuring data durability and high availability for further use as historical data.

GITHUB LOCATION OF CODE

Project Location:

https://github.com/YTushar18/advDataPipeline_Kafka_Spark


YTushar18 / advDataPipeline_Kafka_Spark
Type / to search
Code
Issues
Pull requests 1
Actions
Projects
Wiki
Security
Insights
Settings
Files
main
Go to file
step1_TempDataDump
.gitignore
README.md
config.py
requirements.txt
step1_captureData.py
step1_captureData_dummy.py
step2_kafka_consumer_py_scri...
step2_kafka_consumer_spark_...
advDataPipeline_Kafka_Spark / README.md
YTushar18 Update README.md
ce797ec · now
History
Preview
Code
Blame
54 lines (32 loc) · 2.22 KB
Code 55% faster with GitHub Copilot
Raw

Adv Database Project

Team Members

1. Tushar Yadav
2. Dhruti Dilipbhai Patel

Setting up the environment

1. Install latest Java on your mac
2. Install Apache Kafka (https://medium.com/@Ankitthakur/apache-kafka-installation-on-mac-using-homebrew-a367cdefd273)
3. Install Apache Spark (https://sparkbyexamples.com/spark/install-apache-spark-on-mac/)
4. Install python 3.7 or later
5. Install Dependencies using pip - python3 -m pip install -r requirements.txt

Starting kafka server

START THE KAFKA ENVIRONMENT NOTE: Your local environment must have Java 8+ installed.

Apache Kafka can be started using ZooKeeper or KRaft. To get started with either configuration follow one the sections below but not both.

Kafka with ZooKeeper Run the following commands in order to start all services in the correct order:

Start the ZooKeeper service

$ bin/zookeeper-server-start.sh config/zookeeper.properties Open another terminal session and run:

DEPLOYMENT INSTRUCTIONS

The project is designed for local setup and doesn't require deployment. Currently, we have established a Python environment locally, running the Python scripts on the same machine where Kafka servers are operational. Looking ahead, our future enhancement goals involve deploying the Python scripts on Azure Functions and utilizing Azure Kafka services. Additionally, we plan to employ Azure Storage Services for data storage.

STEPS TO RUN THE APPLICATION

Setting up the environment:

1. Install the latest Java version on your mac.

2. Install Apache Kafka (*https://medium.com/@Ankitthakur/apache-kafka-installation-on-mac-using-homebrew-a367cdefd273*)

3. Install Apache Spark (https://*sparkbyexamples.com/spark/install-apache-spark-on-mac/*)

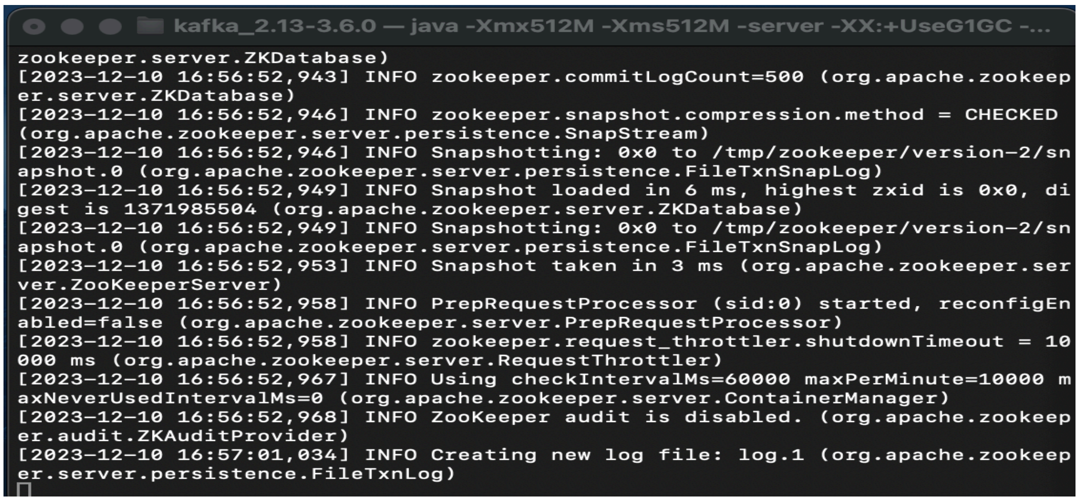
4. Install python 3.7 or later.

5. Install Dependencies using pip - python3 -m pip install -r requirements.txt.

6. Starting kafka server:
   Start The Kafka Environment Note: Your local environment must have Java 8+ installed. Apache Kafka can be started using ZooKeeper or KRaft. To get started with either configuration follow one the sections below but not both. Kafka with ZooKeeper Run the following commands to start all services in the correct order.

7. Start the ZooKeeper service:

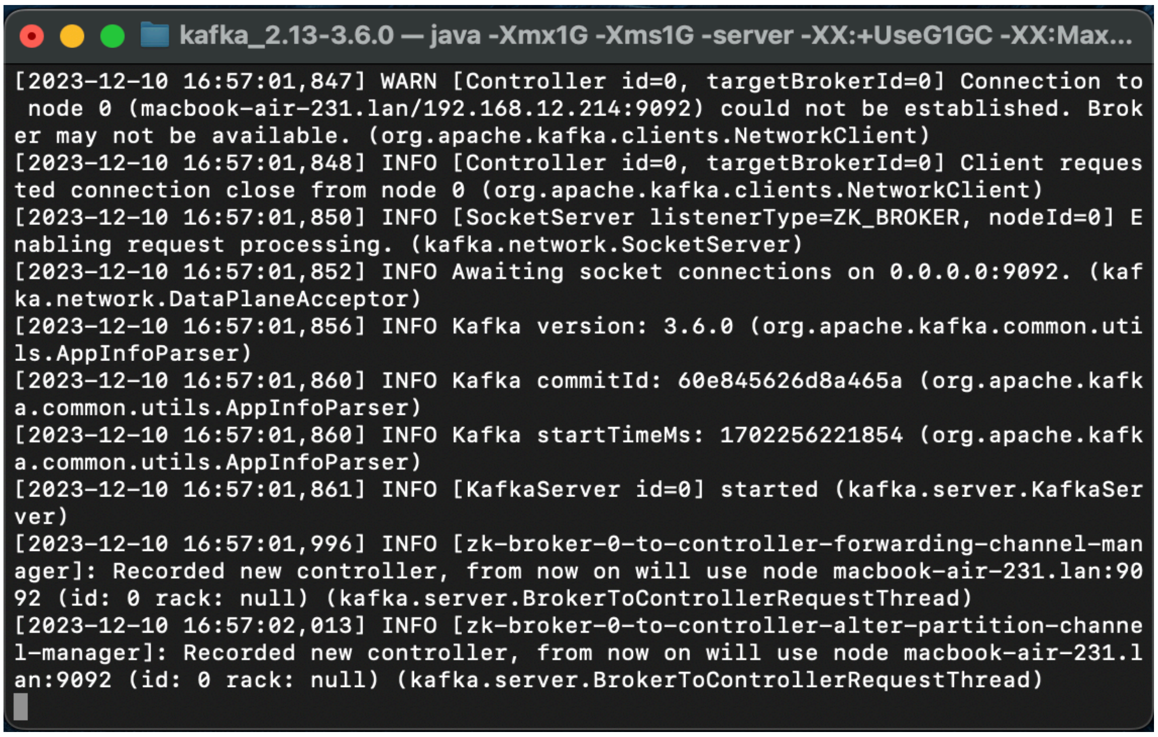
   $ bin/zookeeper-server-start.sh config/zookeeper.properties


kafka_2.13-3.6.0 — java -Xmx512M -Xms512M -server -XX:+UseG1GC -...

```
zookeeper.server.ZKDatabase)
[2023-12-10 16:56:52,943] INFO zookeeper.commitLogCount=500 (org.apache.zookeep
er.server.ZKDatabase)
[2023-12-10 16:56:52,946] INFO zookeeper.snapshot.compression.method = CHECKED
(org.apache.zookeeper.server.persistence.SnapStream)
[2023-12-10 16:56:52,946] INFO Snapshotting: 0x0 to /tmp/zookeeper/version-2/sn
apshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2023-12-10 16:56:52,949] INFO Snapshot loaded in 6 ms, highest zxid is 0x0, di
gest is 1371985504 (org.apache.zookeeper.server.ZKDatabase)
[2023-12-10 16:56:52,949] INFO Snapshotting: 0x0 to /tmp/zookeeper/version-2/sn
apshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2023-12-10 16:56:52,953] INFO Snapshot taken in 3 ms (org.apache.zookeeper.ser
ver.ZooKeeperServer)
[2023-12-10 16:56:52,958] INFO PrepRequestProcessor (sid:0) started, reconfigEn
abled=false (org.apache.zookeeper.server.PrepRequestProcessor)
[2023-12-10 16:56:52,958] INFO zookeeper.request_throttler.shutdownTimeout = 10
000 ms (org.apache.zookeeper.server.RequestThrottler)
[2023-12-10 16:56:52,967] INFO Using checkIntervalMs=60000 maxPerMinute=10000 m
axNeverUsedIntervalMs=0 (org.apache.zookeeper.server.ContainerManager)
[2023-12-10 16:56:52,968] INFO ZooKeeper audit is disabled. (org.apache.zookeep
er.audit.ZKAuditProvider)
[2023-12-10 16:57:01,034] INFO Creating new log file: log.1 (org.apache.zookeep
er.server.persistence.FileTxnLog)
```

8. Start the kafka broker service:

   $ bin/kafka-server-start.sh config/server.properties


kafka_2.13-3.6.0 — java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:Max...

```
[2023-12-10 16:57:01,847] WARN [Controller id=0, targetBrokerId=0] Connection to
 node 0 (macbook-air-231.lan/192.168.12.214:9092) could not be established. Brok
er may not be available. (org.apache.kafka.clients.NetworkClient)
[2023-12-10 16:57:01,848] INFO [Controller id=0, targetBrokerId=0] Client reques
ted connection close from node 0 (org.apache.kafka.clients.NetworkClient)
[2023-12-10 16:57:01,850] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] E
nabling request processing. (kafka.network.SocketServer)
[2023-12-10 16:57:01,852] INFO Awaiting socket connections on 0.0.0.0:9092. (kaf
ka.network.DataPlaneAcceptor)
[2023-12-10 16:57:01,856] INFO Kafka version: 3.6.0 (org.apache.kafka.common.uti
ls.AppInfoParser)
[2023-12-10 16:57:01,860] INFO Kafka commitId: 60e845626d8a465a (org.apache.kafk
a.common.utils.AppInfoParser)
[2023-12-10 16:57:01,860] INFO Kafka startTimeMs: 1702256221854 (org.apache.kafk
a.common.utils.AppInfoParser)
[2023-12-10 16:57:01,861] INFO [KafkaServer id=0] started (kafka.server.KafkaSer
ver)
[2023-12-10 16:57:01,996] INFO [zk-broker-0-to-controller-forwarding-channel-man
ager]: Recorded new controller, from now on will use node macbook-air-231.lan:90
92 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
[2023-12-10 16:57:02,013] INFO [zk-broker-0-to-controller-alter-partition-channe
l-manager]: Recorded new controller, from now on will use node macbook-air-231.l
an:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
```

Once all services have successfully launched, you will have a basic Kafka environment running and ready to use.

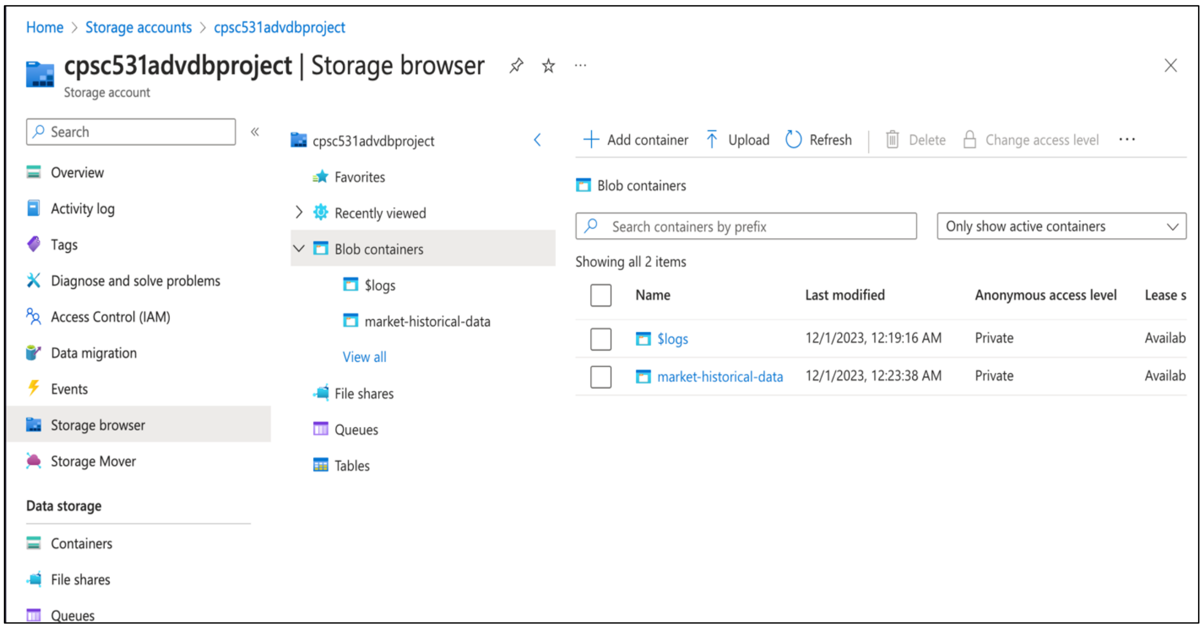
9. Create topic for you kafka streaming:
   Kafka is a distributed event streaming platform that lets you read, write, store, and process events (also called records or messages in the documentation) across many machines. Example events are payment transactions, geolocation updates from mobile phones, shipping orders, sensor measurements from IoT devices or medical equipment, and much more. These events are organized and stored in topics. Very simplified, a topic is like a folder in a filesystem, and the events are the files in that folder.

   So, before you can write your first events, you must create a topic. Open another terminal session and run:

   ```
   $ bin/kafka-topics.sh --create --topic marketdata --bootstrap-server localhost:9092
   ```

10. To setup the Azure Blob Storage Container
    a. Create Storage Account
       https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json&tabs=azure-portal

b. Create a blob for storing market historical data.


Home > Storage accounts > cpsc531advdbproject

cpsc531advdbproject | Storage browser

Storage account

	Name	Last modified	Anonymous access level	Lease s
☐	$logs	12/1/2023, 12:19:16 AM	Private	Availab
☐	market-historical-data	12/1/2023, 12:23:38 AM	Private	Availab

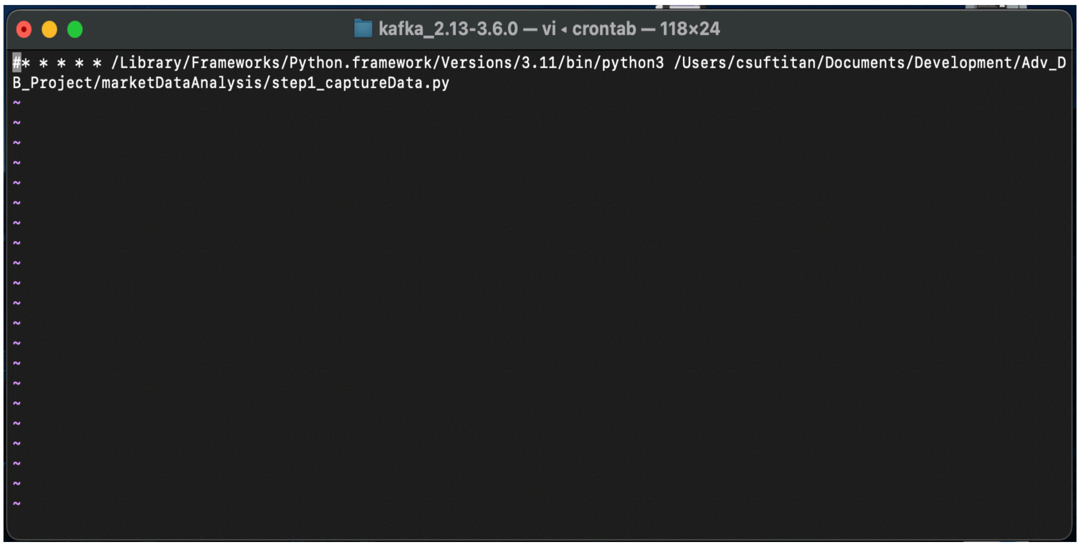
Steps To Run the Application:

Run Python Scripts:

1. Open 2 separate new terminals.

2. In one terminal we will run step1_captureData.py inside a cron job so that this script runs continuously in the background every 1 min to get data from Kafka into spark as streaming data and perform analysis.

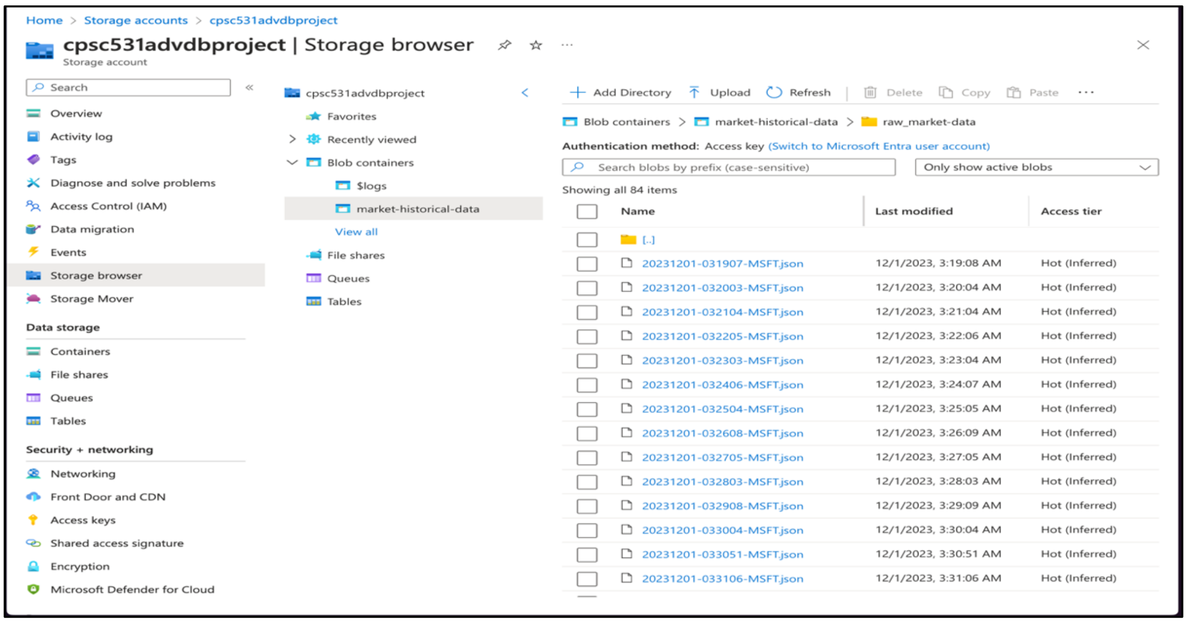
    - Crontab -e
    - Add
    - * * * * * *<path-to-python-binary> <path_to_step1_captureData.py_script>*



```
#* * * * * /Library/Frameworks/Python.framework/Versions/3.11/bin/python3 /Users/csuftitan/Documents/Development/Adv_D
B_Project/marketDataAnalysis/step1_captureData.py
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

3. In another terminal run step2_kafka_consumer_spark_script.py as a consumer application that will fetch data from the kafka topi and proceed with the analysis and will start a localhost web app with the analytical charts.

4. By this time, you should start seeing historical data being pushed onto the Azure Blob Storage.



Authentication method: Access key (Switch to Microsoft Entra user account)

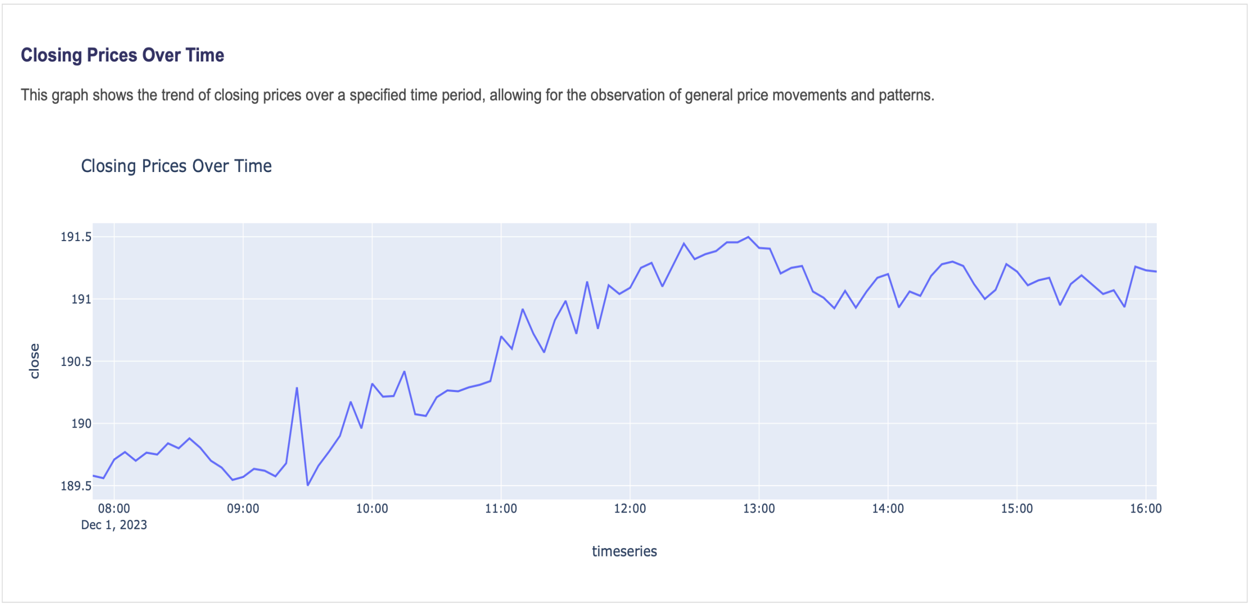
Showing all 84 items

Name	Last modified	Access tier
[..]		
20231201-031907-MSFT.json	12/1/2023, 3:19:08 AM	Hot (Inferred)
20231201-032003-MSFT.json	12/1/2023, 3:20:04 AM	Hot (Inferred)
20231201-032104-MSFT.json	12/1/2023, 3:21:04 AM	Hot (Inferred)
20231201-032205-MSFT.json	12/1/2023, 3:22:06 AM	Hot (Inferred)
20231201-032303-MSFT.json	12/1/2023, 3:23:04 AM	Hot (Inferred)
20231201-032406-MSFT.json	12/1/2023, 3:24:07 AM	Hot (Inferred)
20231201-032504-MSFT.json	12/1/2023, 3:25:05 AM	Hot (Inferred)
20231201-032608-MSFT.json	12/1/2023, 3:26:09 AM	Hot (Inferred)
20231201-032705-MSFT.json	12/1/2023, 3:27:05 AM	Hot (Inferred)
20231201-032803-MSFT.json	12/1/2023, 3:28:03 AM	Hot (Inferred)
20231201-032908-MSFT.json	12/1/2023, 3:29:09 AM	Hot (Inferred)
20231201-033004-MSFT.json	12/1/2023, 3:30:04 AM	Hot (Inferred)
20231201-033051-MSFT.json	12/1/2023, 3:30:51 AM	Hot (Inferred)
20231201-033106-MSFT.json	12/1/2023, 3:31:06 AM	Hot (Inferred)

RESULTS

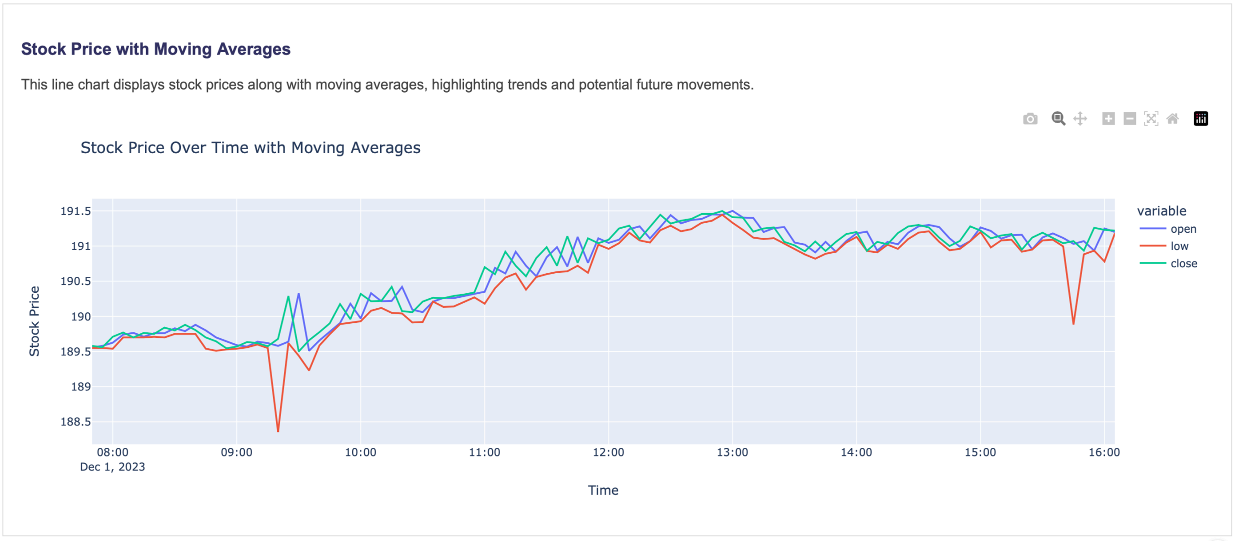
Using the Plotly Dash library, an HTML page will be generated to display a dashboard featuring various plots. Some of the plots of analysis are mentioned below:

1. This chart provides a clear and straightforward visualization of price movements over time. It's useful for seeing the general trend and closing price stability, which can help in assessing the stock's performance on a particular day.


Closing Prices Over Time

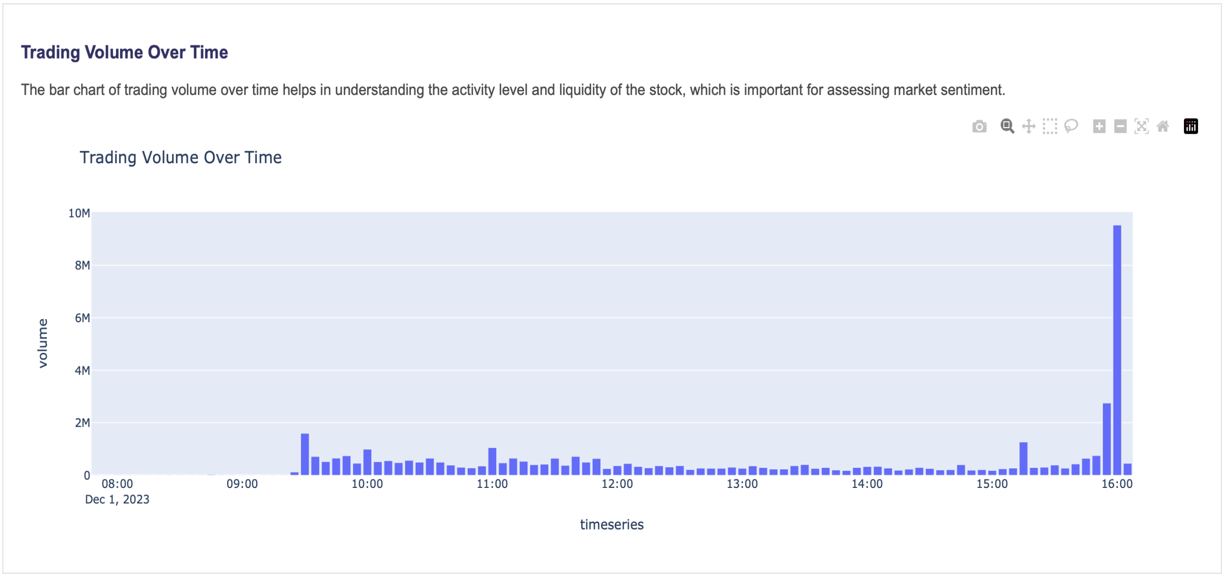
This graph shows the trend of closing prices over a specified time period, allowing for the observation of general price movements and patterns.

2. This combines the actual stock price with its moving average, smoothing out short-term fluctuations and highlighting longer-term trends or tendencies in the stock's price. It can be a tool for identifying potential support or resistance levels.


Stock Price with Moving Averages

This line chart displays stock prices along with moving averages, highlighting trends and potential future movements.

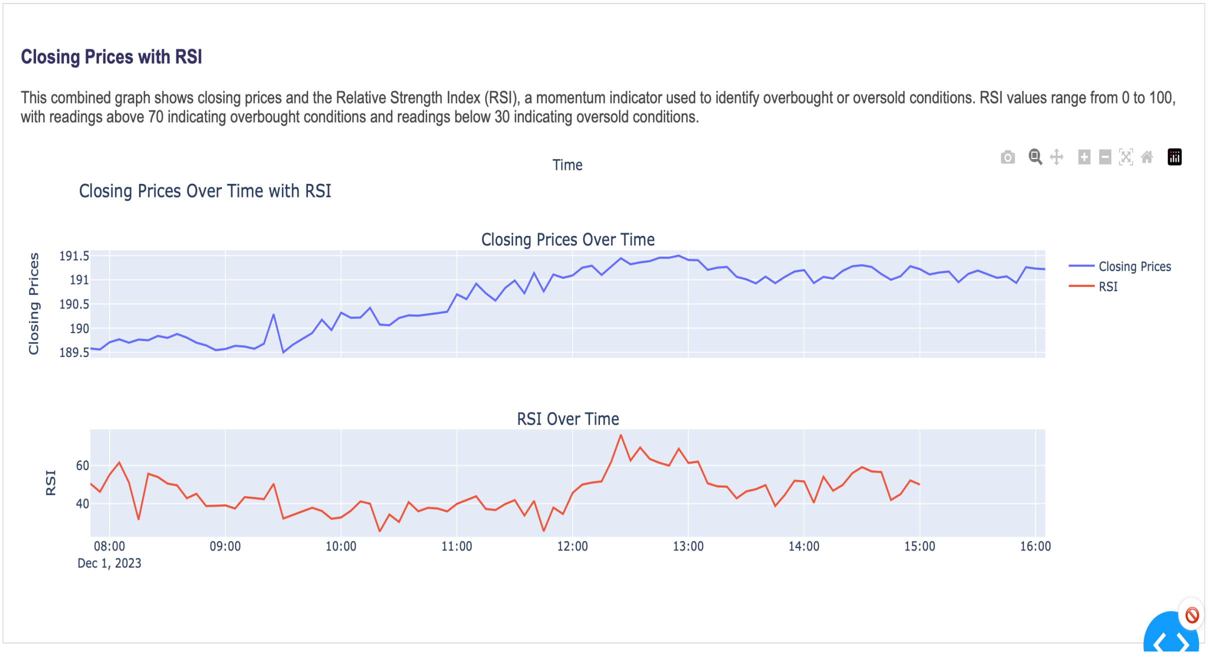
3. This visualizes the amount of trading activity over time. High volume can confirm a trend or signal a reversal if it occurs at price extremes. It can also indicate the strength of a price move; for example, a price move on high volume is often more significant than a move on low volume.



Trading Volume Over Time

The bar chart of trading volume over time helps in understanding the activity level and liquidity of the stock, which is important for assessing market sentiment.

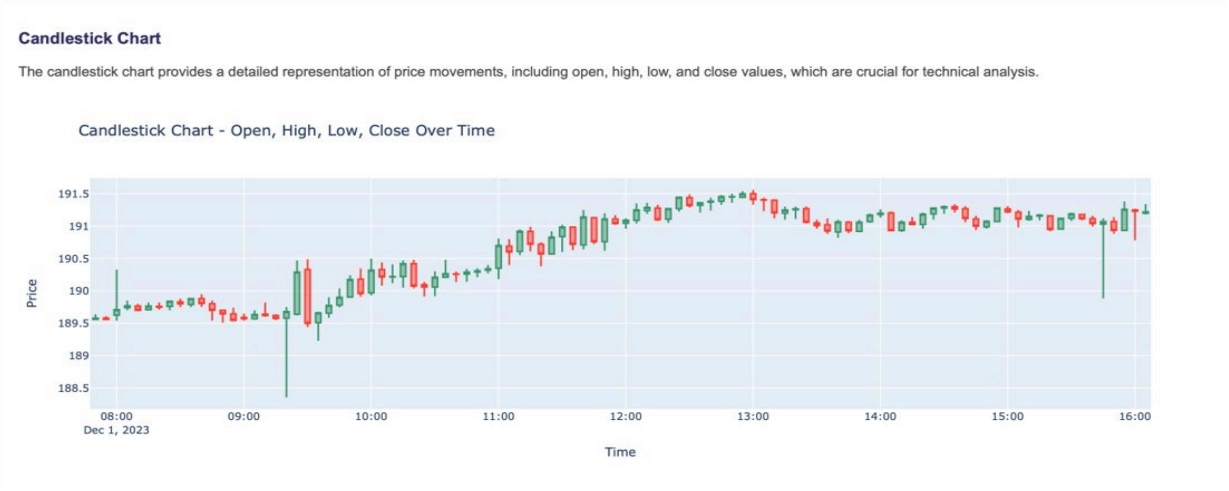
4. The RSI is a momentum oscillator that measures the speed and change of price movements. It helps in identifying overbought or oversold conditions. This can inform traders when market sentiment may be reaching an extreme and a reversal could be forthcoming.



Closing Prices with RSI

This combined graph shows closing prices and the Relative Strength Index (RSI), a momentum indicator used to identify overbought or oversold conditions. RSI values range from 0 to 100, with readings above 70 indicating overbought conditions and readings below 30 indicating oversold conditions.

5. It offers detailed information about price movements within a specific time frame, showing the open, high, low, and close prices. Traders use this to understand market sentiment and to identify patterns that may indicate future price movements.



Candlestick Chart

The candlestick chart provides a detailed representation of price movements, including open, high, low, and close values, which are crucial for technical analysis.

REFERENCES

- HTTPS://LEARN.MICROSOFT.COM/EN-US/AZURE/ARCHITECTURE/
- HTTPS://SPARK.APACHE.ORG/DOCS/LATEST/
- HTTPS://KAFKA.APACHE.ORG/USES
- HTTPS://WWW.ALPHAVANTAGE.CO/DOCUMENTATION/#