MGMT 59000BD

AI-Assisted Big Data Analytics in the Cloud

Final Project

Stock Price Prediction and Anomaly Detection

Group 2
Vineeth Pydi
Shubhankar Sharma
Roshan Raj Singh
Sai Bheeshma Ramaraju Pagilla
Bhawishya Juneja

Objective

This project aims to integrate historical stock data from Alpha Vantage with real-time stock data streamed through the Alpha Vantage API. Utilizing various Google Cloud Platform (GCP) technologies, the project will analyze stock operations for selected symbols. The primary goal is to predict stock price movements and associated investment risks for specific stocks across different time frames.

Tasks

Analytical Tasks and Machine Learning

1. **Stock Price Forecasting:** Utilizing ARIMA_PLUS model in BigQuery ML for time-series forecasting, we predict significant stock price movements based on historical and real-time data.

2. **Anomaly Detection:** Our system identifies anomalies in stock prices over the last year, utilizing daily stock prices and anomaly flags to inform investors.

3. **Data Visualization:** Interactive Data Studio dashboards are created to display stock status across different time frames and visualize forecasted prices and anomalies.

Process Overview

The project involves several stages, including data collection, cleaning, integration, and analysis, followed by machine learning for delay prediction.

Key components of our solution:

Real-time data pipeline: Cloud Pub/Sub will stream web clickstream and transactional data to Cloud Dataflow for real-time preprocessing and analysis. The processed data will be stored in BigQuery.
Batch processing: Batch jobs on Dataflow will run daily to process larger datasets for more complex modeling and analytics. Results will be persisted in BigQuery.
Machine learning: Managed AI services like Vertex AI will train models to power recommendations, search rankings, churn prediction, and other personalization use cases.
Business intelligence: Data Studio dashboards will provide interactive reporting on KPIs. Looker will enable advanced analytics for business users.
Security: Data will be encrypted in transit and at rest. IAM policies, VPC service controls, and other mechanisms will ensure data security and access control.

Data Integration and Processing

Data Sources:

Archival Data: Sourced from Alpha Vantage's API, providing historical stock data, including open, high, low, close values, and trading volumes.
Real-Time Data: Streamed through Alpha Vantage's API, leveraging Cloud Functions and Cloud Scheduler in GCP for current price updates and trading volumes.

Data Pipeline:

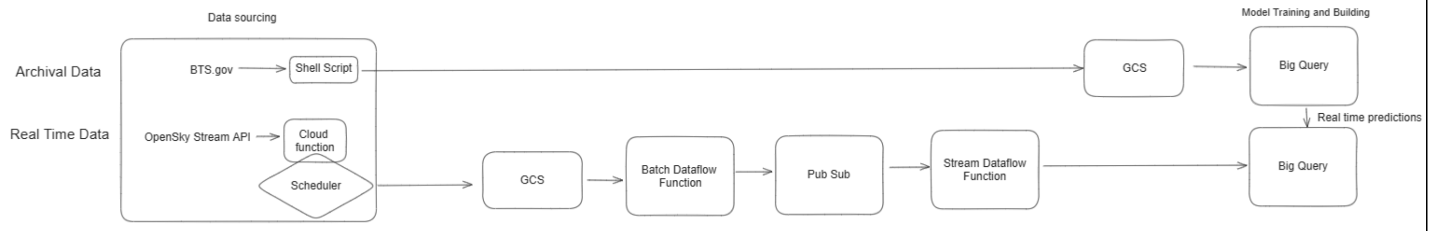
Storage: Both types of data are stored in Google Cloud Storage (GCS), forming a unified data lake.
Processing:
Batch Dataflow Function processes archival data for data transformation tasks.
Stream Dataflow Function handles real-time streaming data, subscribing to the Pub/Sub topic.

Machine Learning and Analysis:

Processed data is stored in GCS and then moved into BigQuery for further analysis. Machine Learning models are trained using BigQuery ML and Vertex AI Workbench for real-time stock price predictions and anomaly detection.

Building the Data Pipeline

1. Pipeline Overview



Data Sourcing

1. Archival Data: Data is sourced from Alpha Vantage's API which provides historical stock data. This data includes historical price points like open, high, low, close values, and trading volumes for a comprehensive list of stocks, which will be used to analyze past stock performance and identify trends over time.

2. Real-Time Data: Real-time stock data is sourced using Alpha Vantage's API through a combination of Cloud Functions and Cloud Scheduler in GCP. This setup implies that real-time data, including current price updates and trading volumes for selected stock symbols, is being ingested on a regular schedule. Cloud Functions are triggered to fetch and process this data at predetermined intervals managed by Cloud Scheduler.

Data Storage and Processing

1. Google Cloud Storage (GCS): Both archival and real-time data are stored in GCS, which acts as a unified data lake or storage system.
2. Batch Dataflow Function: The archival data in GCS is processed through a batch processing pipeline (Dataflow), which could be for cleaning, transforming, or aggregating the data.
3. Pub/Sub: The real-time data processed by the Cloud Function is published to a Pub/Sub topic, which is a messaging queue for event-driven systems. It allows for the real-time data to be consumed by downstream services.
4. Stream Dataflow Function: This function subscribes to the Pub/Sub topic to process the real-time streaming data. The processing could involve similar data transformation tasks as with the batch data but on a real-time data stream.

Model Training and Building

1. Google Cloud Storage (GCS): The processed data from both the batch and stream pipelines are stored again in GCS, potentially in a format ready for analysis and model training.

2. BigQuery: This processed data is then moved into BigQuery, a data warehouse, for further analysis and to train machine learning models. BigQuery is used for its ability to handle SQL queries over large datasets.

Real-time Stock Price Predictions

1. Real-time stock price predictions: The model trained in BigQuery ML and Vertex AI Workbench is leveraged to make real-time stock price predictions. The architecture indicates that the live data ingested and processed can be input into the machine learning model to forecast stock price movements, such as significant increases, decreases, or to identify potential market trends.

In summary, the architecture illustrates a complete data engineering and machine learning workflow, encompassing data acquisition to storage, processing, analysis, and ultimately, real-time financial analytics. The integration of both historical and live data streams facilitates a robust and dynamic approach to managing diverse data types, enabling timely and informed stock market predictions.

Fetching the Data

1. ***Archival data***

Using a shell script, we download historical data for the stocks from Yahoo Finance and store it in one of the buckets. This will be the data on which we will build and train the model.

2. ***Real time data (for deploying the model to do real-time predictions)***

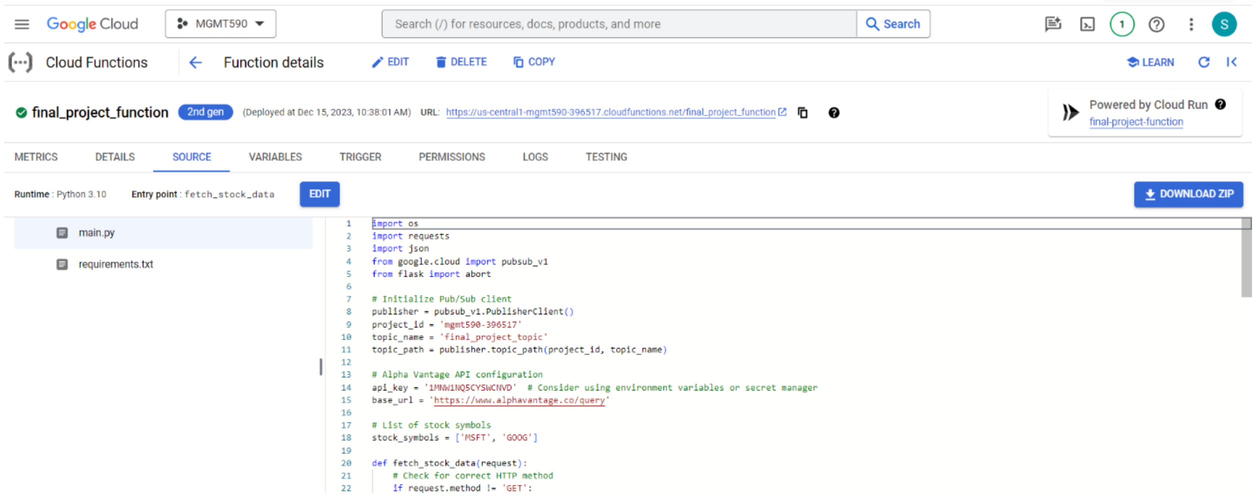
Code:

```
import os
import requests
import json
from google.cloud import pubsub_v1
from flask import abort

# Initialize Pub/Sub client
publisher = pubsub_v1.PublisherClient()
project_id = 'mgmt590-396517'
topic_name = 'final_project_topic'
topic_path = publisher.topic_path(project_id, topic_name)

# Alpha Vantage API configuration
api_key = '1MNW1NQ5CYSWCNVD'  # Consider using environment variables or secret manager
base_url = 'https://www.alphavantage.co/query'
```

```python
# List of stock symbols
stock_symbols = ['MSFT', 'GOOG']

def fetch_stock_data(request):
    # Check for correct HTTP method
    if request.method != 'GET':
        return abort(405)

    for symbol in stock_symbols:
        params = {
            'function': 'TIME_SERIES_INTRADAY',
            'symbol': symbol,
            'interval': '5min',
            'apikey': api_key,
            'datatype': 'json'
        }

        response = requests.get(base_url, params=params)
        if response.status_code == 200:
            data = response.json()
            time_series = data.get('Time Series (5min)', {})

            for date, daily_data in time_series.items():
                # Extracting the required fields
                stock_data = {
                    'Date': date,
                    'Symbol': symbol,
                    'Open': daily_data['1. open'],
                    'High': daily_data['2. high'],
                    'Low': daily_data['3. low'],
                    'Close': daily_data['4. close'],
                    'Volume': daily_data['5. volume']
                }

                # Publish to Pub/Sub
                future = publisher.publish(topic_path, json.dumps(stock_data).encode('utf-8'))
                future.result()  # Confirm the publish succeeded

        else:
            return f'Error fetching stock data for {symbol}: {response.text}', response.status_code

    return 'Data fetched and published successfully for MSFT and GOOG', 200
```

```
# The line below is for local testing and should be removed in the deployed Cloud Function
# fetch_stock_data(None)
```

A. Configure cloud function to fetch live data from API, specifying trigger point as "fetch_stock_data"



Cloud Functions — Function details

final_project_function 2nd gen (Deployed at Dec 15, 2023, 10:38:01 AM) URL: https://us-central1-mgmt590-396517.cloudfunctions.net/final_project_function

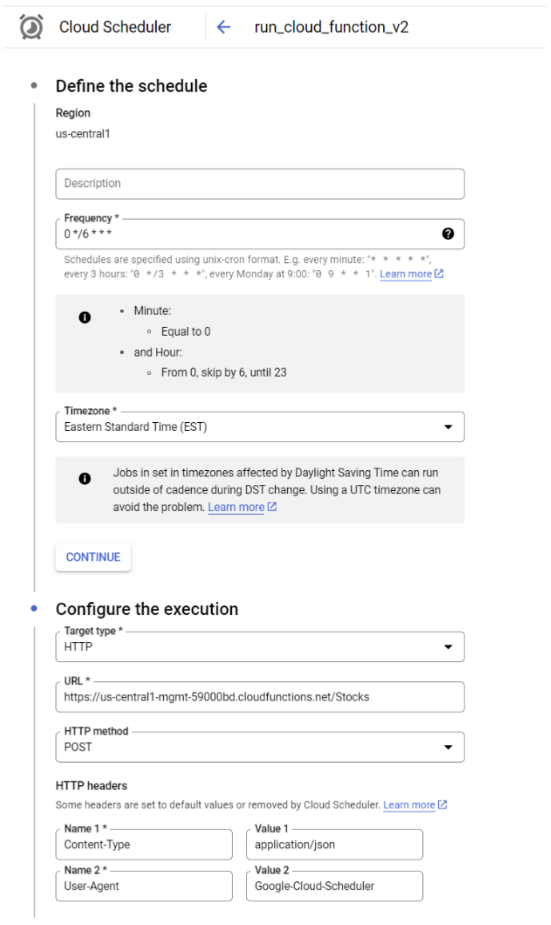
Powered by Cloud Run final-project-function

Runtime : Python 3.10 Entry point : fetch_stock_data

main.py
requirements.txt

```
import os
import requests
import json
from google.cloud import pubsub_v1
from flask import abort

# Initialize Pub/Sub client
publisher = pubsub_v1.PublisherClient()
project_id = 'mgmt590-396517'
topic_name = 'final_project_topic'
topic_path = publisher.topic_path(project_id, topic_name)

# Alpha Vantage API configuration
api_key = '1MNW1NQ5CYSWCNVD'  # Consider using environment variables or secret manager
base_url = 'https://www.alphavantage.co/query'

# List of stock symbols
stock_symbols = ['MSFT', 'GOOG']

def fetch_stock_data(request):
    # Check for correct HTTP method
    if request.method != 'GET':
```

B. Configure scheduler to run the above cloud function every 6 hours



Cloud Scheduler — run_cloud_function_v2

Define the schedule

Region
us-central1

Description

Frequency *
0 */6 * * *

Schedules are specified using unix-cron format. E.g. every minute: "* * * * *", every 3 hours: "0 */3 * * *", every Monday at 9:00: "0 9 * * 1". Learn more

- Minute:
  - Equal to 0
- and Hour:
  - From 0, skip by 6, until 23

Timezone *
Eastern Standard Time (EST)

Jobs in set in timezones affected by Daylight Saving Time can run outside of cadence during DST change. Using a UTC timezone can avoid the problem. Learn more

CONTINUE

Configure the execution

Target type *
HTTP

URL *
https://us-central1-mgmt-59000bd.cloudfunctions.net/Stocks

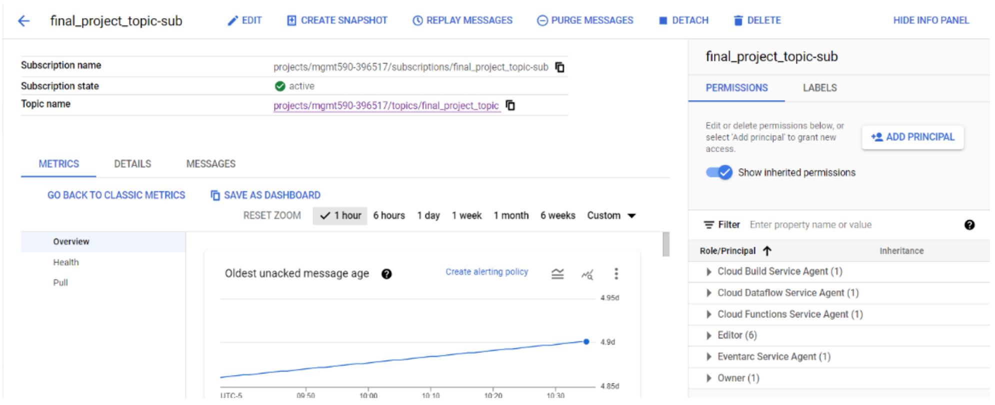
HTTP method
POST

HTTP headers

Some headers are set to default values or removed by Cloud Scheduler. Learn more

Name 1 * Content-Type	Value 1 application/json
Name 2 * User-Agent	Value 2 Google-Cloud-Scheduler

C. Create a Pub/Sub topic with a Pull subscription



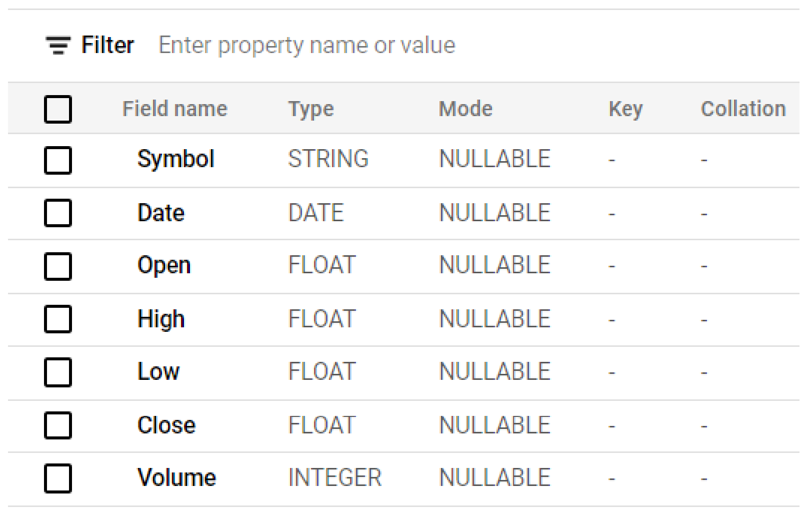
Subscription name	projects/mgmt590-396517/subscriptions/final_project_topic-sub
Subscription state	active
Topic name	projects/mgmt590-396517/topics/final_project_topic

Edit or delete permissions below, or select 'Add principal' to grant new access.

Show inherited permissions

Role/Principal	Inheritance
Cloud Build Service Agent (1)	
Cloud Dataflow Service Agent (1)	
Cloud Functions Service Agent (1)	
Editor (6)	
Eventarc Service Agent (1)	
Owner (1)	

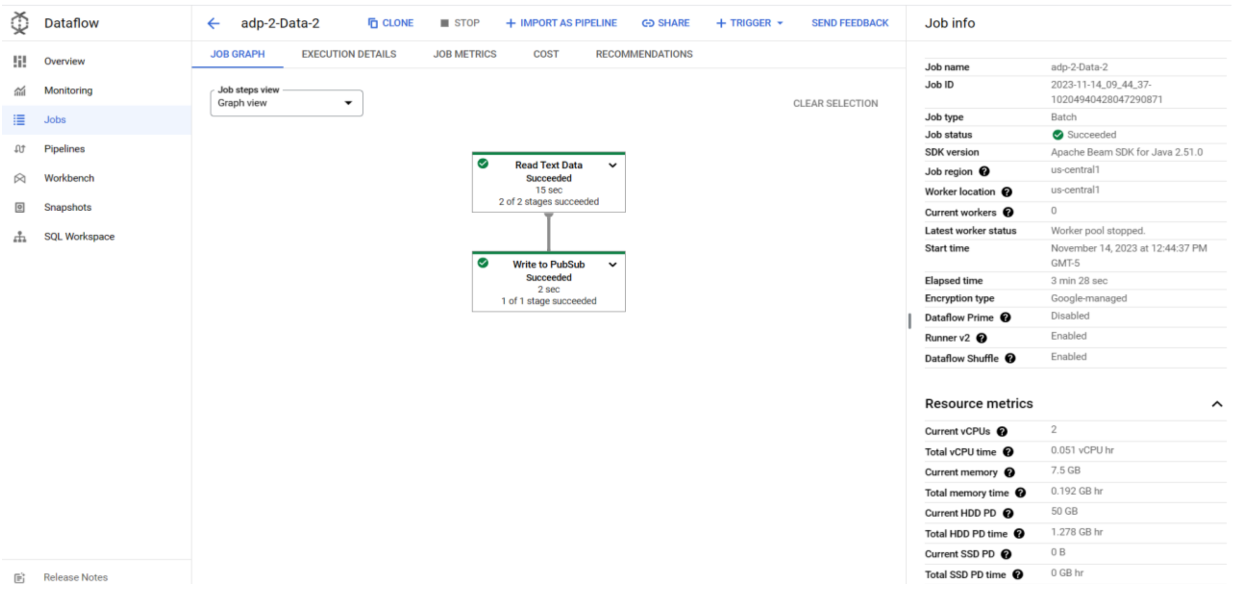
D. Create an empty table with a schema in BigQuery



Filter Enter property name or value

Field name	Type	Mode	Key	Collation
Symbol	STRING	NULLABLE	-	-
Date	DATE	NULLABLE	-	-
Open	FLOAT	NULLABLE	-	-
High	FLOAT	NULLABLE	-	-
Low	FLOAT	NULLABLE	-	-
Close	FLOAT	NULLABLE	-	-
Volume	INTEGER	NULLABLE	-	-

E. Create a batch dataflow function to move data from GCS to Pub/Sub



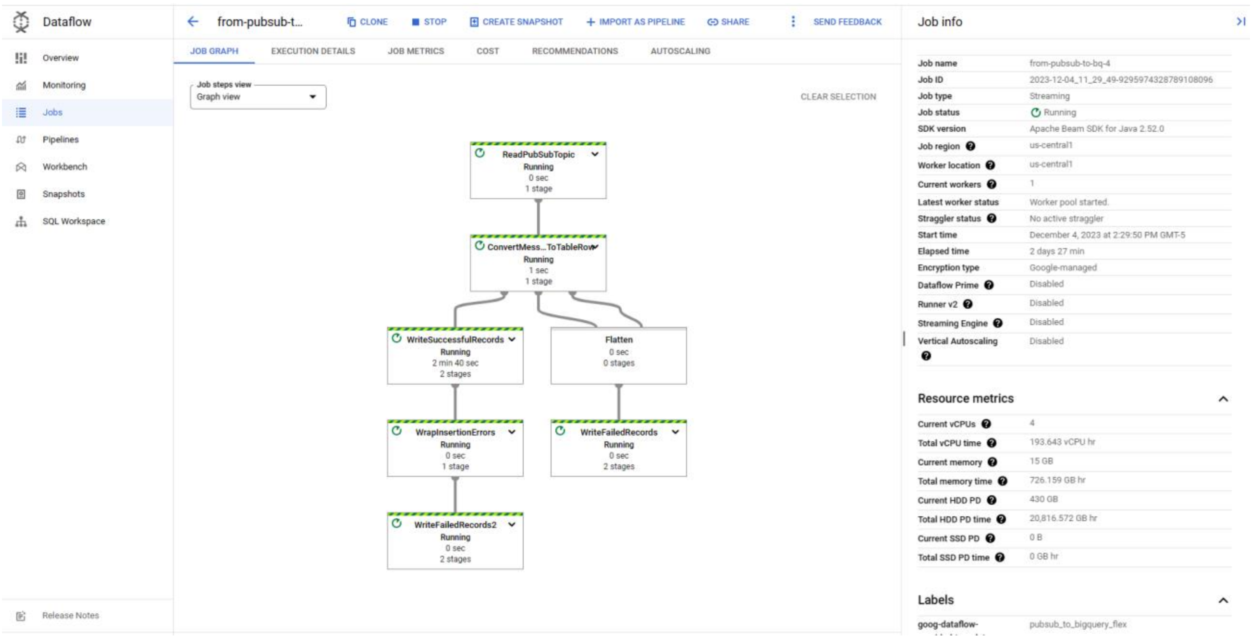
Job info

Job name	adp-2-Data-2
Job ID	2023-11-14_09_44_37-10204940428047290871
Job type	Batch
Job status	Succeeded
SDK version	Apache Beam SDK for Java 2.51.0
Job region	us-central1
Worker location	us-central1
Current workers	0
Latest worker status	Worker pool stopped.
Start time	November 14, 2023 at 12:44:37 PM GMT-5
Elapsed time	3 min 28 sec
Encryption type	Google-managed
Dataflow Prime	Disabled
Runner v2	Enabled
Dataflow Shuffle	Enabled

Resource metrics

Current vCPUs	2
Total vCPU time	0.051 vCPU hr
Current memory	7.5 GB
Total memory time	0.192 GB hr
Current HDD PD	50 GB
Total HDD PD time	1.278 GB hr
Current SSD PD	0 B
Total SSD PD time	0 GB hr

F. Create a stream dataflow function to push data from Pub/Sub to BigQuery



Job info

Job name	from-pubsub-to-bq-4
Job ID	2023-12-04_11_29_49-9295974328789108096
Job type	Streaming
Job status	Running
SDK version	Apache Beam SDK for Java 2.52.0
Job region	us-central1
Worker location	us-central1
Current workers	1
Latest worker status	Worker pool started.
Straggler status	No active straggler
Start time	December 4, 2023 at 2:29:50 PM GMT-5
Elapsed time	2 days 27 min
Encryption type	Google-managed
Dataflow Prime	Disabled
Runner v2	Disabled
Streaming Engine	Disabled
Vertical Autoscaling	Disabled

Resource metrics

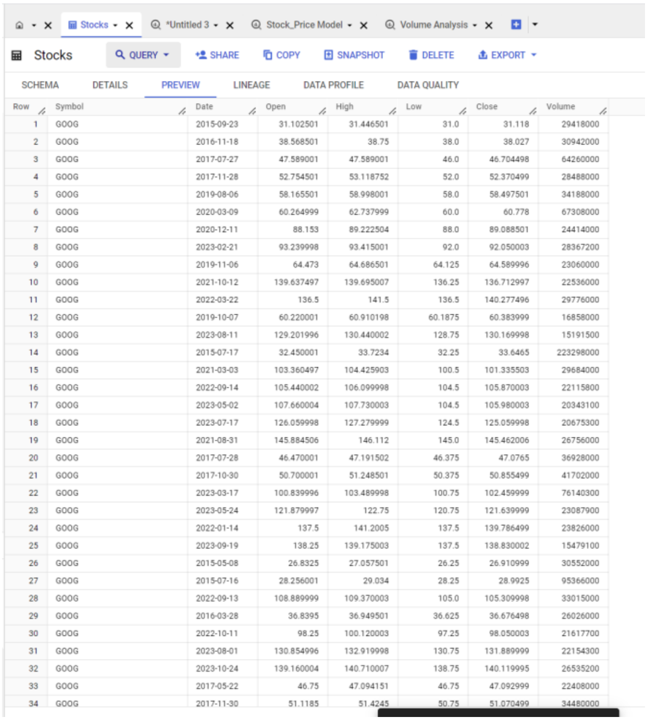
Current vCPUs	4
Total vCPU time	193.643 vCPU hr
Current memory	15 GB
Total memory time	726.159 GB hr
Current HDD PD	430 GB
Total HDD PD time	20,816.572 GB hr
Current SSD PD	0 B
Total SSD PD time	0 GB hr

Labels

goog-dataflow-	pubsub_to_bigquery_flex

G. Analyze both Archival and Real-time data on BigQuery

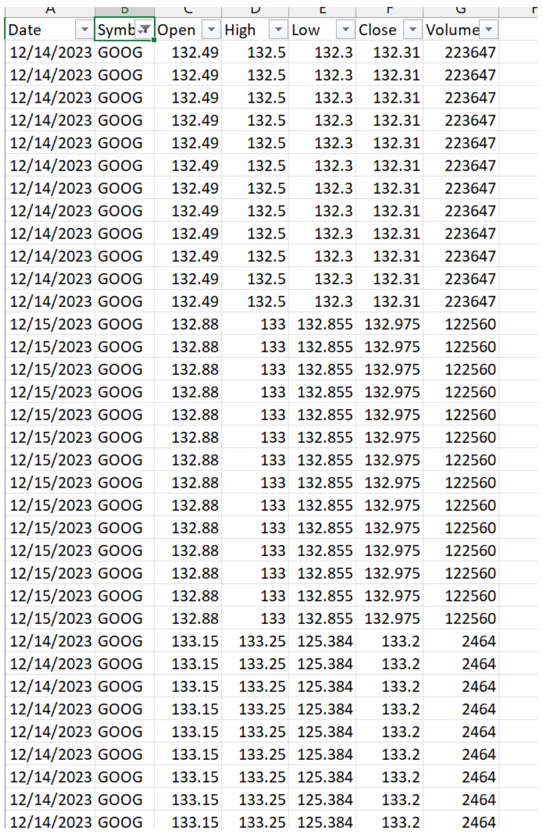
Historical Data



Stocks

Row	Symbol	Date	Open	High	Low	Close	Volume
1	GOOG	2015-09-23	31.102501	31.446501	31.0	31.118	29418000
2	GOOG	2016-11-18	38.568501	38.75	38.0	38.027	30942000
3	GOOG	2017-07-27	47.589001	47.589001	46.0	46.704498	64260000
4	GOOG	2017-11-28	52.754501	53.118752	52.0	52.370499	28488000
5	GOOG	2019-08-06	58.165501	58.998001	58.0	58.497501	34188000
6	GOOG	2020-03-09	60.264999	62.737999	60.0	60.778	67308000
7	GOOG	2020-12-11	88.153	89.222504	88.0	89.088501	24414000
8	GOOG	2023-02-21	93.239998	93.415001	92.0	92.050003	28367200
9	GOOG	2019-11-06	64.473	64.686501	64.125	64.589996	23060000
10	GOOG	2021-10-12	139.637497	139.695007	136.25	136.712997	22536000
11	GOOG	2022-03-22	136.5	141.5	136.5	140.277496	29776000
12	GOOG	2019-10-07	60.220001	60.910198	60.1875	60.383999	16858000
13	GOOG	2023-08-11	129.201996	130.440002	128.75	130.169998	15191500
14	GOOG	2015-07-17	32.450001	33.7234	32.25	33.6465	223298000
15	GOOG	2021-03-03	103.360497	104.425903	100.5	101.335503	29684000
16	GOOG	2022-09-14	105.440002	106.099998	104.5	105.870003	22115800
17	GOOG	2023-05-02	107.660004	107.730003	104.5	105.980003	20343100
18	GOOG	2023-07-17	126.059998	127.279999	124.5	125.059998	20675300
19	GOOG	2021-08-31	145.884506	146.112	145.0	145.462006	26756000
20	GOOG	2017-07-28	46.470001	47.191502	46.375	47.0765	36928000
21	GOOG	2017-10-30	50.700001	51.248501	50.375	50.855499	41702000
22	GOOG	2023-03-17	100.839996	103.489998	100.75	102.459999	76140300
23	GOOG	2023-05-24	121.879997	122.75	120.75	121.639999	23087900
24	GOOG	2022-01-14	137.5	141.2005	137.5	139.786499	23826000
25	GOOG	2023-09-19	138.25	139.175003	137.5	138.830002	15479100
26	GOOG	2015-05-08	26.8325	27.057501	26.25	26.910999	30552000
27	GOOG	2015-07-16	28.256001	29.034	28.25	28.9925	95366000
28	GOOG	2022-09-13	108.889999	109.370003	105.0	105.309998	33015000
29	GOOG	2016-03-28	36.8395	36.949501	36.625	36.676498	26026000
30	GOOG	2022-10-11	98.25	100.120003	97.25	98.050003	21617700
31	GOOG	2023-08-01	130.854996	132.919998	130.75	131.889999	22154300
32	GOOG	2023-10-24	139.160004	140.710007	138.75	140.119995	26535200
33	GOOG	2017-05-22	46.75	47.094151	46.75	47.092999	22408000
34	GOOG	2017-11-30	51.1185	51.4245	50.75	51.070499	34480000

Realtime:



Date	Symb	Open	High	Low	Close	Volume
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/14/2023	GOOG	132.49	132.5	132.3	132.31	223647
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/15/2023	GOOG	132.88	133	132.855	132.975	122560
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464
12/14/2023	GOOG	133.15	133.25	125.384	133.2	2464

<u>Technical Approach to Build the Pipeline:</u>

- **Step 1: Design batch processing architecture**
  - Identify Data Sources and Optimal Extraction Methods
    - Assess data sources such as user activity logs, product catalogs, and transaction records.
    - Establish extraction methods using Cloud Storage for unstructured data and BigQuery or Cloud SQL extracts for structured data.
  - Select Technologies for Workflow Orchestration, Processing, and Output
    - Choose Cloud Composer for workflow orchestration.
    - Select Dataflow for data processing based on Apache Beam.
    - Determine BigQuery as the output destination for processed data.
  - Design a Scalable Architecture on Google Cloud to Handle Large Volumes
    - Architect a solution using scalable services like Dataflow which auto-scales and BigQuery for large data volumes.
    - Utilize managed instance groups for scalable Compute Engine resources if necessary.
  - Document Architecture, Data Flows, and Technical Specifications

    - Use Google Docs and Diagrams within the GCP Console or tools like Lucidchart to document the architecture.

- **Step 2: Set up Cloud Composer Environment**

    - Provision Cloud Composer Workflow Orchestration Service
        - Open the GCP Console.
        - Navigate to the `Cloud Composer` section and click `Create`.
        - Use the form to configure the environment with appropriate machine types and network settings.
    - Configure Composer Environment with Required Dependencies
        - Once the environment is ready, use Cloud Shell or Cloud SDK to install dependencies with `pip` if using a custom image or specify in the `requirements.txt`.
    - Set Up Scheduling and Failure Handling Workflows
        - Define Airflow DAGs for scheduling data ingestion and processing tasks within the Cloud Composer environment.
        - Implement failure handling strategies using Airflow's built-in retry and alert mechanisms.

- **Step 3: Build ETL Data Pipelines**

    - Develop Apache Beam Pipelines for ETL Using Java/Python SDK
        - Write ETL pipeline scripts using Apache Beam's Java or Python SDK.
        - Develop transforms for standard operations – use predefined classes like `Filter`, `ParDo`, and `GroupByKey`.
    - Ingest Data from Sources like BigQuery and Cloud Storage
        - Use `ReadFromBigQuery` and `ReadFromText` (for files from Cloud Storage) transforms to ingest data into your Beam pipeline.
    - Implement Transforms for Parsing, Validating, Cleansing Data
        - Apply `ParDo` transforms to parse JSON/XML input data and perform any validation checks.
        - Apply cleaning logic using Beam's SQL transform or custom `DoFn`s.
    - Aggregate Metrics and Join Dimensional Data
        - Use Beam's `GroupByKey` or SQL transform to aggregate metrics.
        - Utilize `CoGroupByKey` for joining dimensional data from multiple PCollection sources.

- **Step 4: Develop Analytics Pipelines**

    - Build Apache Beam Pipelines for Analytics and ML Tasks
        - Add analytics logic to Beam pipelines using built-in analytics functions, statistical libraries or calls to ML APIs.

- Analyze Data Using Beam SQL, Dataframe, and ML APIs
  - Incorporate analytic transformations using `Beam SQL` or DataFrame APIs for complex analytics operations.
  - Integrate ML model predictions using direct calls to AI Platform's Prediction Service within your pipeline.
- Design Reusable Pipelines for Cohort Analysis, Funnel Analysis, Anomaly Detection
  - Abstract common analytics tasks into reusable composite transforms or templates.
- Trigger Model Training and Batch Predictions Using Vertex AI
  - Define tasks within Cloud Composer to start training jobs on Vertex AI based on aggregated data availability.

- **Step 5: Load Output to BigQuery**

Use SQL Queries to generate insights.

Building the Model

Volume Analysis:

Trading Volume Trends: Volume data can be used to confirm trends or predict potential reversals. For instance, increasing volume along with price increases can indicate a strong upward trend.

Liquidity Analysis: Volume data helps in understanding the liquidity of a stock, which is crucial for making buy or sell decisions.

Process:
Step 1: Data Aggregation

Aggregate volume data in BigQuery to analyze trading volume trends.

Step 2: Correlation Analysis

Analyze the correlation between volume and price movements using SQL in BigQuery or AI Platform Notebooks.

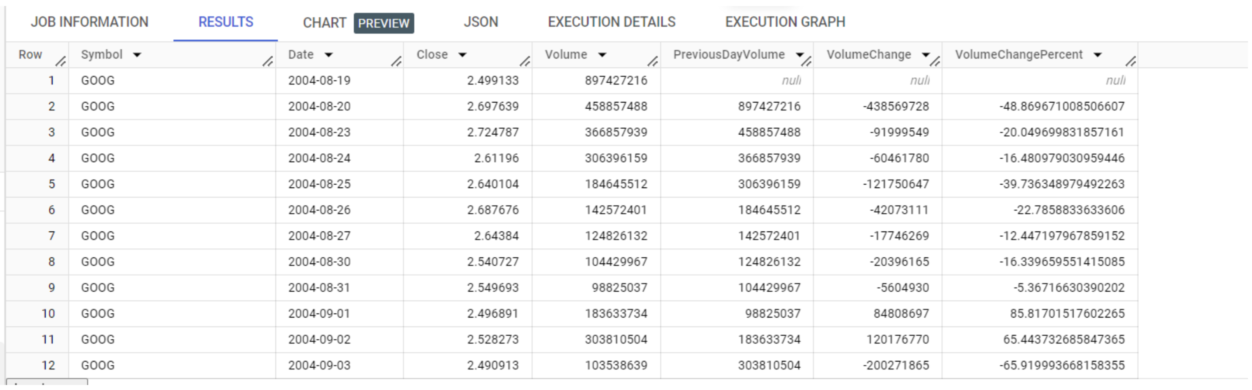
Step 3: Visualization

Use Data Studio to create visualizations that highlight volume trends and their relation to price changes.

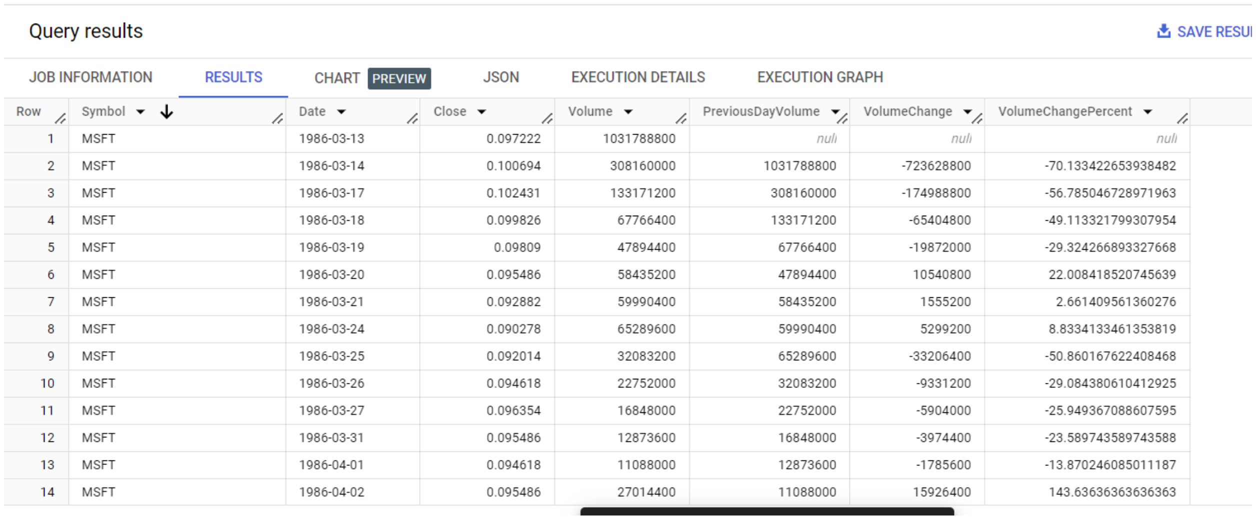
BigQuery Code:

```
WITH VolumeAnalysis AS (
 SELECT
  Symbol,
  Date,
  Close,
  Volume,
  LAG(Volume, 1) OVER (PARTITION BY Symbol ORDER BY Date) AS PreviousDayVolume,
  (Volume - LAG(Volume, 1) OVER (PARTITION BY Symbol ORDER BY Date)) AS VolumeChange,
  (Volume - LAG(Volume, 1) OVER (PARTITION BY Symbol ORDER BY Date)) / LAG(Volume, 1)
OVER (PARTITION BY Symbol ORDER BY Date) * 100 AS VolumeChangePercent
 FROM
  `mgmt-59000bd.Final_Project.Stocks`
)

SELECT
 Symbol,
 Date,
 Close,
 Volume,
 PreviousDayVolume,
 VolumeChange,
 VolumeChangePercent
FROM
 VolumeAnalysis
ORDER BY
 Symbol,
 Date;
```

Result:



Row	Symbol	Date	Close	Volume	PreviousDayVolume	VolumeChange	VolumeChangePercent
1	GOOG	2004-08-19	2.499133	897427216	null	null	null
2	GOOG	2004-08-20	2.697639	458857488	897427216	-438569728	-48.869671008506607
3	GOOG	2004-08-23	2.724787	366857939	458857488	-91999549	-20.049699831857161
4	GOOG	2004-08-24	2.61196	306396159	366857939	-60461780	-16.480979030959446
5	GOOG	2004-08-25	2.640104	184645512	306396159	-121750647	-39.736348979492263
6	GOOG	2004-08-26	2.687676	142572401	184645512	-42073111	-22.7858833633606
7	GOOG	2004-08-27	2.64384	124826132	142572401	-17746269	-12.447197967859152
8	GOOG	2004-08-30	2.540727	104429967	124826132	-20396165	-16.339659551415085
9	GOOG	2004-08-31	2.549693	98825037	104429967	-5604930	-5.36716630390202
10	GOOG	2004-09-01	2.496891	183633734	98825037	84808697	85.81701517602265
11	GOOG	2004-09-02	2.528273	303810504	183633734	120176770	65.443732685847365
12	GOOG	2004-09-03	2.490913	103538639	303810504	-200271865	-65.919993668158355



Row	Symbol	Date	Close	Volume	PreviousDayVolume	VolumeChange	VolumeChangePercent
1	MSFT	1986-03-13	0.097222	1031788800	null	null	null
2	MSFT	1986-03-14	0.100694	308160000	1031788800	-723628800	-70.133422653938482
3	MSFT	1986-03-17	0.102431	133171200	308160000	-174988800	-56.785046728971963
4	MSFT	1986-03-18	0.099826	67766400	133171200	-65404800	-49.113321799307954
5	MSFT	1986-03-19	0.09809	47894400	67766400	-19872000	-29.324266893327668
6	MSFT	1986-03-20	0.095486	58435200	47894400	10540800	22.008418520745639
7	MSFT	1986-03-21	0.092882	59990400	58435200	1555200	2.661409561360276
8	MSFT	1986-03-24	0.090278	65289600	59990400	5299200	8.8334133461353819
9	MSFT	1986-03-25	0.092014	32083200	65289600	-33206400	-50.860167622408468
10	MSFT	1986-03-26	0.094618	22752000	32083200	-9331200	-29.084380610412925
11	MSFT	1986-03-27	0.096354	16848000	22752000	-5904000	-25.949367088607595
12	MSFT	1986-03-31	0.095486	12873600	16848000	-3974400	-23.589743589743588
13	MSFT	1986-04-01	0.094618	11088000	12873600	-1785600	-13.870246085011187
14	MSFT	1986-04-02	0.095486	27014400	11088000	15926400	143.63636363636363

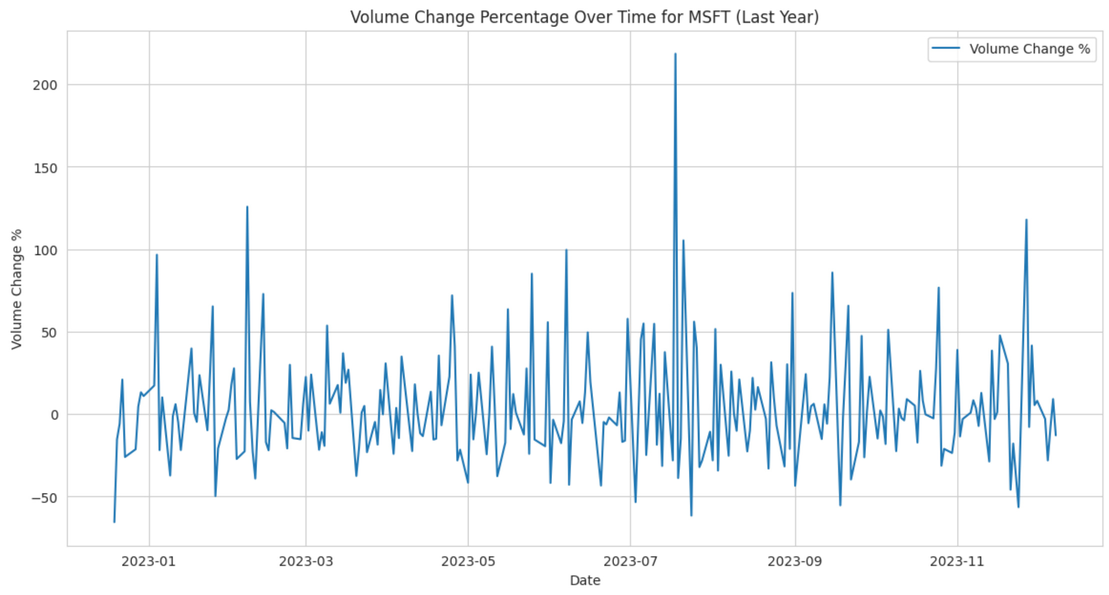
Visualizing the Result:

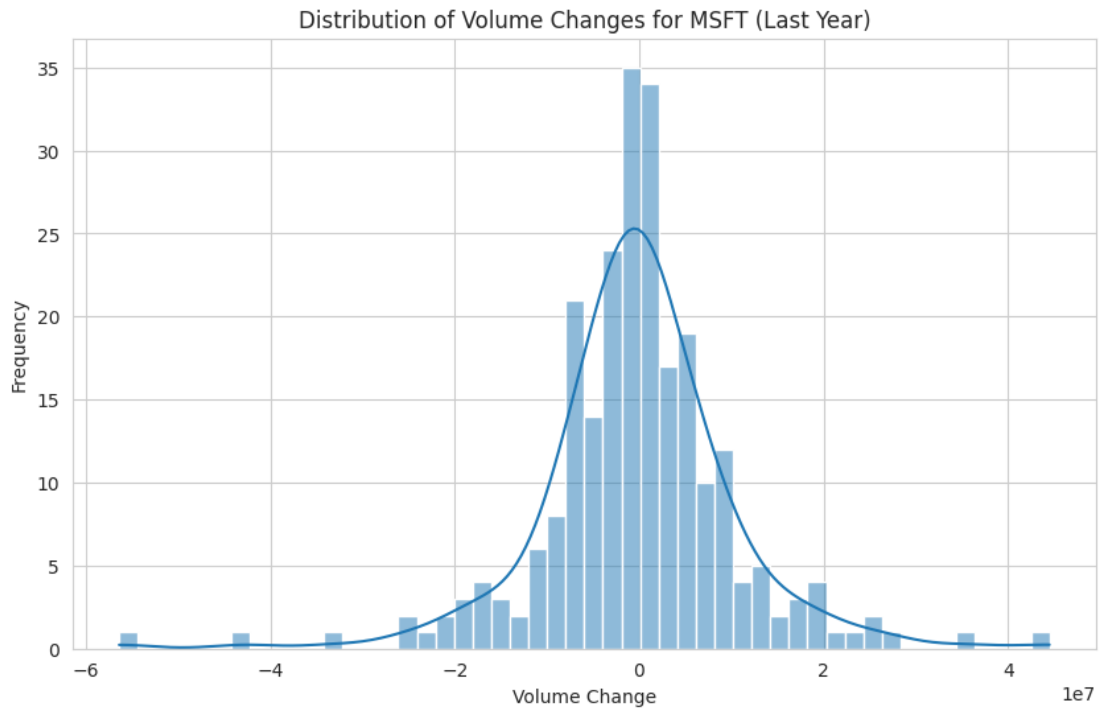
Microsoft(MSFT):

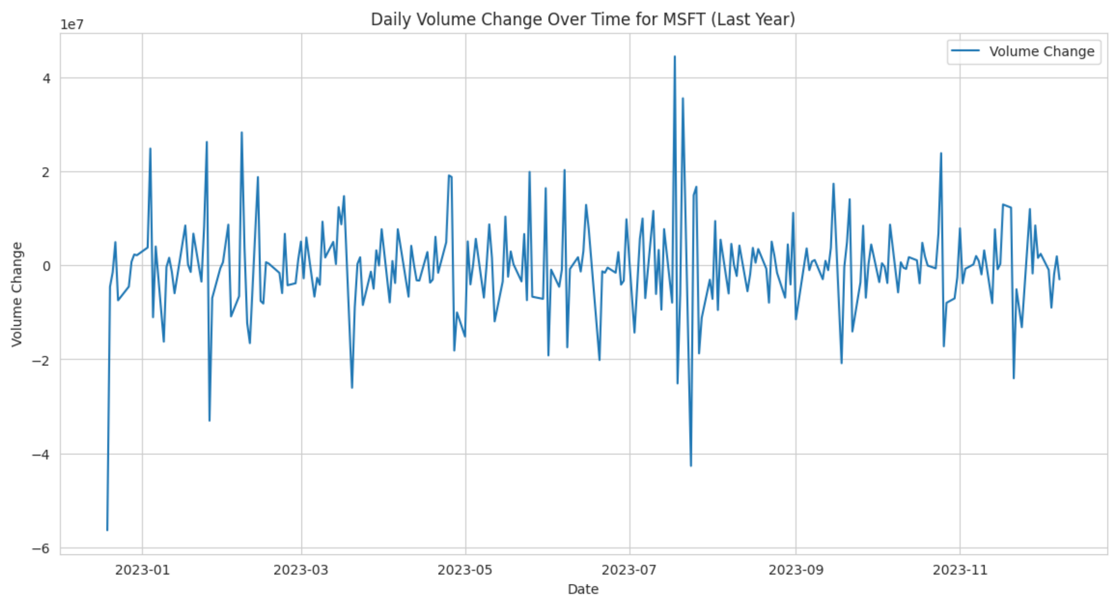
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta

# Load the data from the SQL query into a DataFrame
# Replace with the actual file path to the CSV file
df = pd.read_csv('/content/Volume Analysis.csv')

# Convert the 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Filter the DataFrame for the last year
```

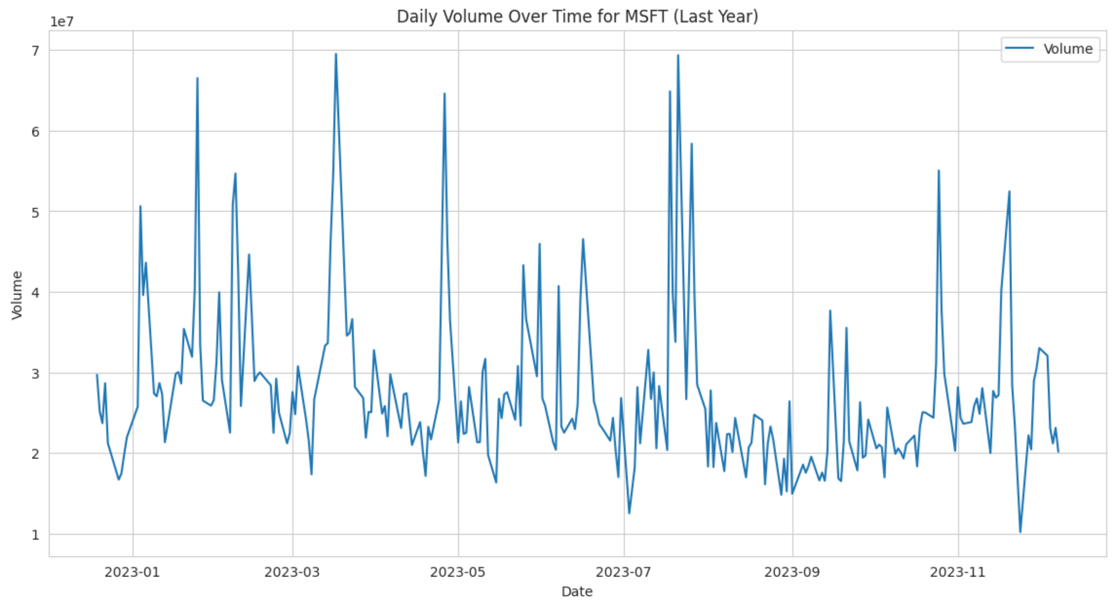
```
one_year_ago = datetime.now() - timedelta(days=365)
df_last_year = df[df['Date'] > one_year_ago]

# Ask the user for the symbol to visualize
symbol = input("Enter the stock symbol you want to visualize: ").upper().strip()

# Filter the DataFrame for the entered symbol and last year
df_symbol = df_last_year[df_last_year['Symbol'] == symbol]

if df_symbol.empty:
    print(f"No data found for symbol: {symbol} in the last year")
else:
    # Set the style for seaborn plots
    sns.set_style("whitegrid")

    # Visualization 1: Time Series Plot of Volume
    plt.figure(figsize=(14, 7))
    plt.plot(df_symbol['Date'], df_symbol['Volume'], label='Volume')
    plt.title(f'Daily Volume Over Time for {symbol} (Last Year)')
    plt.xlabel('Date')
    plt.ylabel('Volume')
    plt.legend()
    plt.show()

    # Visualization 2: Volume Change Over Time
    plt.figure(figsize=(14, 7))
    plt.plot(df_symbol['Date'], df_symbol['VolumeChange'], label='Volume Change')
    plt.title(f'Daily Volume Change Over Time for {symbol} (Last Year)')
    plt.xlabel('Date')
    plt.ylabel('Volume Change')
    plt.legend()
    plt.show()

    # Visualization 3: Histogram of Volume Changes
    plt.figure(figsize=(10, 6))
    sns.histplot(df_symbol['VolumeChange'], bins=50, kde=True)
    plt.title(f'Distribution of Volume Changes for {symbol} (Last Year)')
    plt.xlabel('Volume Change')
    plt.ylabel('Frequency')
    plt.show()

    # Visualization 4: Volume Change Percentage Over Time
    plt.figure(figsize=(14, 7))
    plt.plot(df_symbol['Date'], df_symbol['VolumeChangePercent'], label='Volume Change %')
    plt.title(f'Volume Change Percentage Over Time for {symbol} (Last Year)')
    plt.xlabel('Date')
    plt.ylabel('Volume Change %')
    plt.legend()
    plt.show()
```

```
Enter the stock symbol you want to visualize: MSFT
```









In the dynamic landscape of the stock market, trading volume serves as a critical indicator of a stock's health and investor sentiment. This section of the report delves into the volume analysis of Microsoft Corporation (MSFT) over the last year. The visualization of trading volume provides valuable insights into investor behavior, the impact of market events on trading activity, and aids in the identification of emerging trends.

Volume Change Percentage Over Time

The first visualization focuses on the percentage change in trading volume day-over-day. In the last year, MSFT exhibited several notable spikes in this metric, reflecting days with significant shifts in investor activity. Positive spikes on this graph can often be aligned with positive news releases or corporate announcements, indicating an influx of trading interest. Conversely, negative spikes may signal selling pressure or a lack of investor confidence triggered by various events. Analyzing the patterns of these percentage changes is vital for understanding investor reactions to market stimuli.

Distribution of Volume Changes
The distribution of daily volume changes for MSFT is depicted through a histogram, accompanied by a Kernel Density Estimate (KDE) to provide a smoothed probability density function. The distribution's shape highlights the typical range of volume changes and the frequency of various levels of activity. A concentrated distribution around the mean volume change suggests a stable trading environment, while a wide spread could signify periods of high volatility. This graph is instrumental in evaluating the regularity and extremity of volume changes, serving as a barometer for market stability.

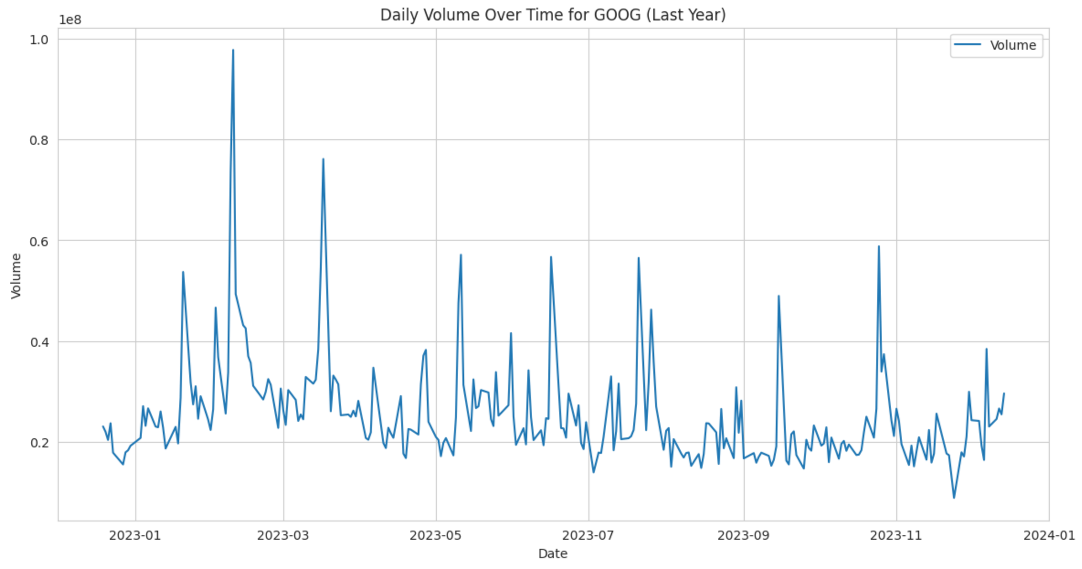
Daily Volume Change Over Time
The raw daily volume change graph presents the actual number of shares traded from one day to the next. Sharp increases or decreases are of particular interest as they may correlate with impactful market or internal company events. For instance, a significant rise in volume might coincide with MSFT's earnings reports or product launches, signaling strong market reactions. This visualization provides a concrete measure of trading volume fluctuations, offering a more tangible sense of the market's pulse.

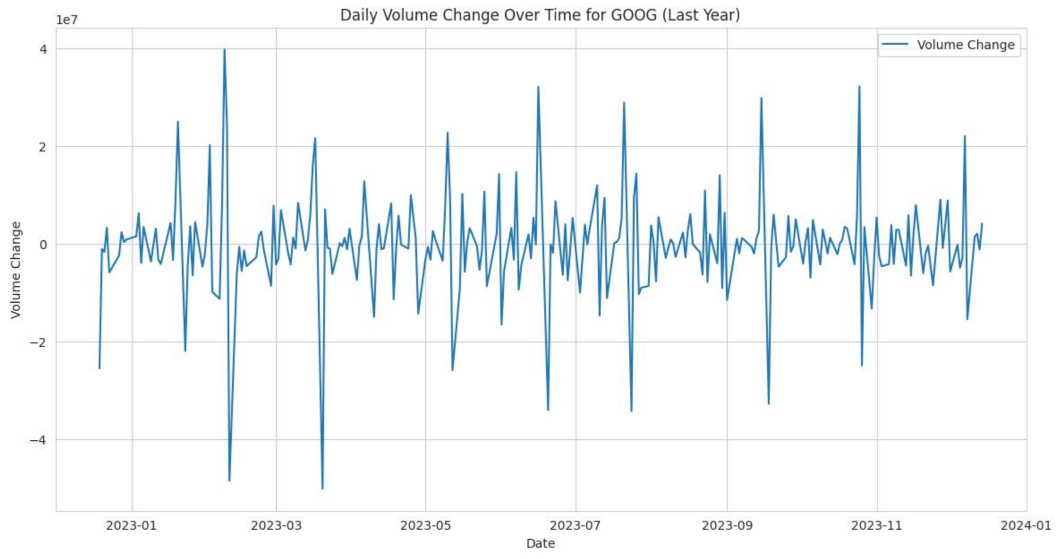
Daily Volume Over Time
Our final graph presents the daily trading volume of MSFT, reflecting the overall liquidity and trading interest in the stock. Fluctuations in daily volume may correlate with price movements; for instance, a high volume day coupled with a substantial price increase could indicate a bullish outlook among investors. This trend analysis is crucial for investors and analysts alike, as it helps in confirming the strength of price movements and could potentially be used to predict future market behavior.

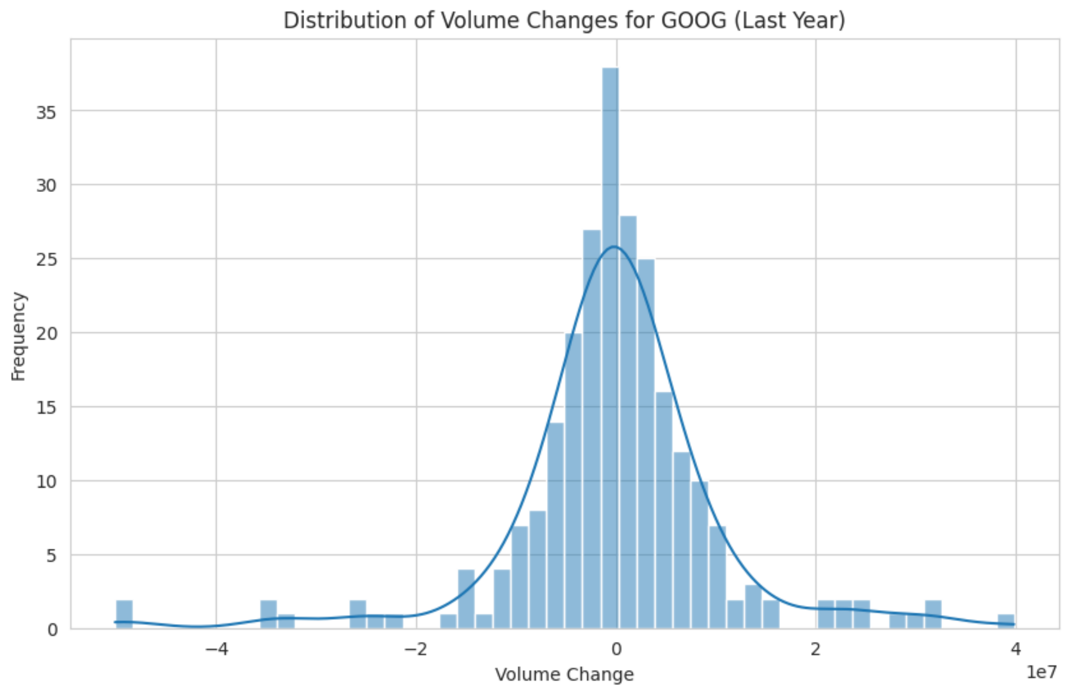
Conclusion
Through meticulous analysis of MSFT's trading volume over the past year, we've uncovered patterns and trends that provide a deeper understanding of market sentiment and investor behavior. The visualizations highlight the importance of volume as an analytic tool and underscore its utility in forming a comprehensive picture of stock activity. These insights are not only vital for individual investment decisions but also contribute to a broader market analysis, influencing trading strategies and portfolio management.

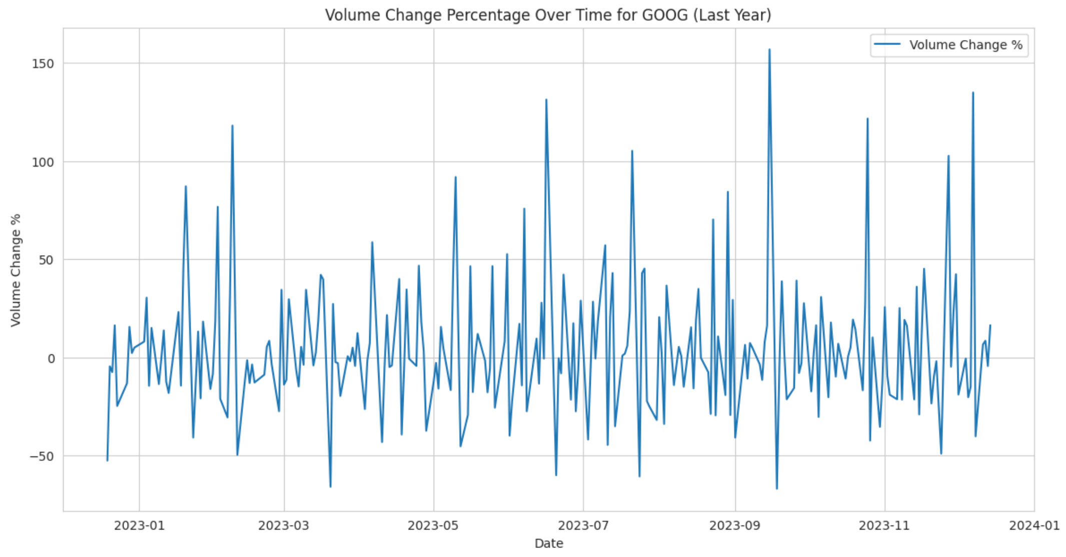
Google(GOOG):

Enter the stock symbol you want to visualize: GOOG









The analysis of trading volume provides an integral insight into the market's sentiment and the vitality of a stock. For Alphabet Inc. (GOOG), the past year's volume data offers a narrative of the stock's liquidity and investor engagement. Presented here is a series of visualizations that convey a story of trading patterns, investor response to company events, and potential market volatility.

Daily Volume Over Time

The first graph exhibits the daily trading volume for GOOG over the past year. This visualization reveals the ebb and flow of trading interest, with spikes indicating days of heightened activity. These peaks might correlate with earnings reports, product announcements, or broader market movements. Monitoring such trends is crucial as they can reflect investor enthusiasm or concern and can often precede or coincide with significant price movements.

Daily Volume Change Over Time

The second graph underscores the day-over-day changes in trading volume for GOOG. It outlines the raw numerical change in the volume of shares traded from one session to the next. Observing these fluctuations allows us to pinpoint days with unusual trading activity, which can be further analyzed in the context of news or market developments. Large surges or drops in volume change can signal shifts in investor sentiment or market dynamics.

Distribution of Volume Changes

The histogram of volume changes for GOOG provides a statistical view of how often and to what extent the trading volume has varied over the last year. The distribution shape indicates the most common ranges for volume changes and can highlight outliers or abnormal trading days. The symmetry, spread, and skewness of the distribution offer insights into the stock's trading stability and can help identify periods of irregular activity.

Volume Change Percentage Over Time

Our final chart portrays the percentage change in trading volume for GOOG, giving a relative sense of the volume fluctuations. This metric is particularly informative as it normalizes the changes against the previous day's volume, allowing for comparability across time irrespective of the absolute volume. Spikes in this graph can indicate investor reaction to news or events, with the magnitude of change serving as a gauge for the intensity of the market's response.

Conclusion

The compilation of volume-related data for GOOG over the last year demonstrates the stock's market behavior and investor interaction. The volume analysis, through these visualizations, provides a multifaceted understanding of market liquidity, investor behavior, and potential volatility. These insights are imperative for investors, traders, and analysts as they form the foundation for making informed decisions and understanding the broader market implications of trading activities.

Correlation Analysis
Exploring the correlation between volume changes and price movements.

The BigQuery code performs a correlation analysis between daily stock price changes and volume changes for each stock symbol. The CorrelationAnalysis common table expression (CTE) calculates the previous day's closing price (PreviousDayClose), the day-over-day price change (PriceChange), and the volume change (VolumeChange) for each stock symbol. The main SELECT query then uses these computed fields to determine a simple correlation indicator:

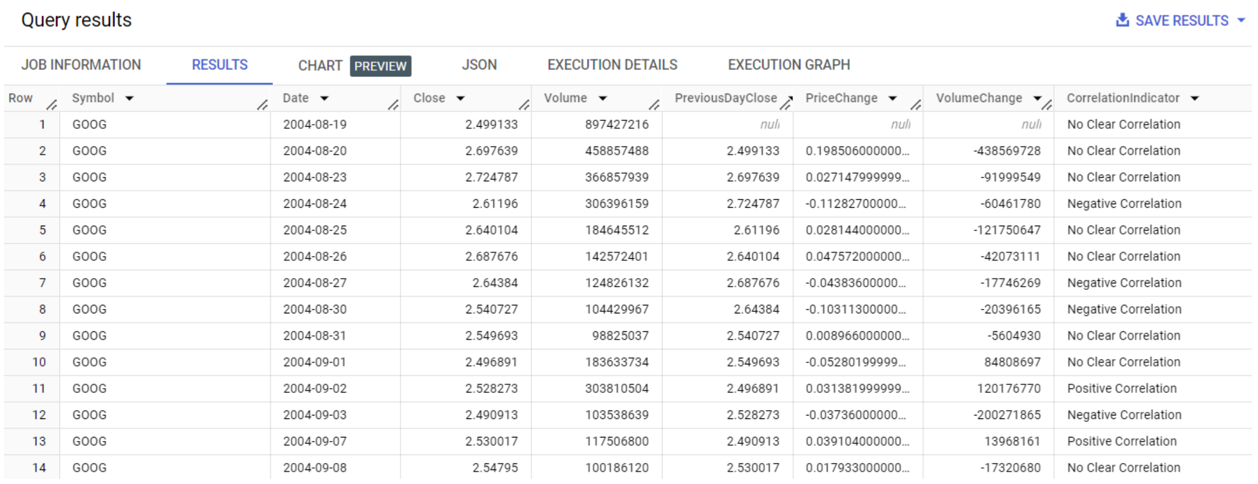
- If both the price and volume increased from the previous day, it's labeled as 'Positive Correlation'.
- If both the price and volume decreased, it's labeled as 'Negative Correlation'.
- If there's no consistent direction in the changes, it's labeled as 'No Clear Correlation'.

This kind of analysis can help identify if there is a general correlation between the stock price movement and trading volume, which could suggest that higher volumes are either driving or reinforcing the price change.

Code:
```
WITH CorrelationAnalysis AS (
 SELECT
  Symbol,
  Date,
  Close,
  Volume,
  LAG(Close, 1) OVER (PARTITION BY Symbol ORDER BY Date) AS PreviousDayClose,
  Close - LAG(Close, 1) OVER (PARTITION BY Symbol ORDER BY Date) AS PriceChange,
  Volume - LAG(Volume, 1) OVER (PARTITION BY Symbol ORDER BY Date) AS VolumeChange
 FROM
  `mgmt-59000bd.Final_Project.Stocks`
)

SELECT
 Symbol,
 Date,
 Close,
 Volume,
 PreviousDayClose,
 PriceChange,
 VolumeChange,
 CASE
  WHEN PriceChange > 0 AND VolumeChange > 0 THEN 'Positive Correlation'
  WHEN PriceChange < 0 AND VolumeChange < 0 THEN 'Negative Correlation'
```

```
    ELSE 'No Clear Correlation'
  END AS CorrelationIndicator
FROM
  CorrelationAnalysis
ORDER BY
  Symbol,
  Date;
```

Result:


Query results

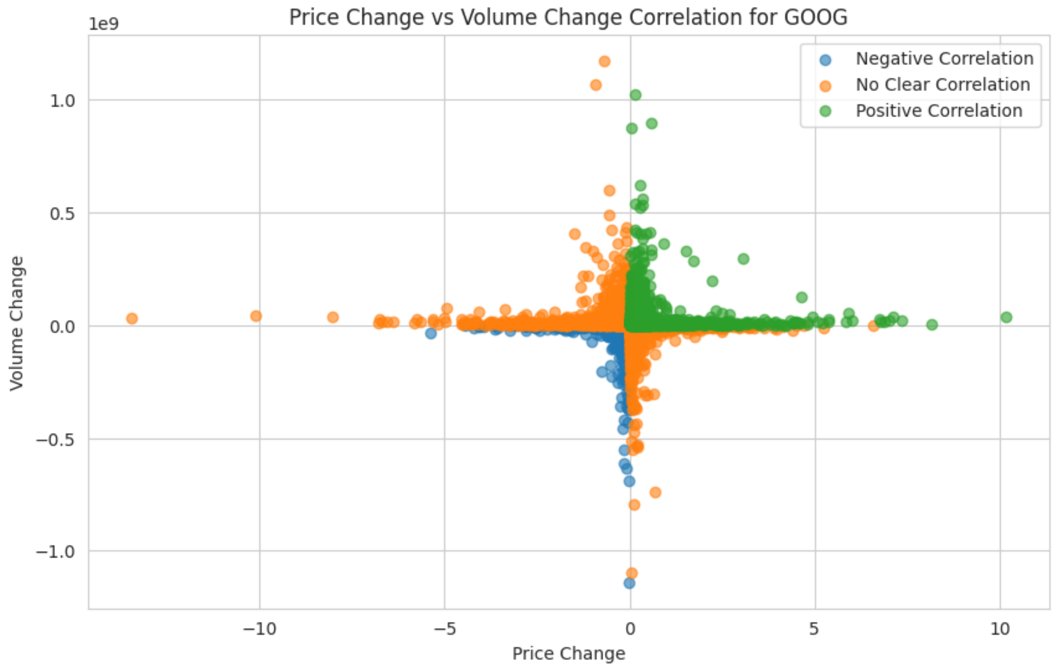
SAVE RESULTS

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	Symbol	Date	Close	Volume	PreviousDayClose	PriceChange	VolumeChange	CorrelationIndicator
1	GOOG	2004-08-19	2.499133	897427216	null	null	null	No Clear Correlation
2	GOOG	2004-08-20	2.697639	458857488	2.499133	0.198506000000...	-438569728	No Clear Correlation
3	GOOG	2004-08-23	2.724787	366857939	2.697639	0.027147999999...	-91999549	No Clear Correlation
4	GOOG	2004-08-24	2.61196	306396159	2.724787	-0.11282700000...	-60461780	Negative Correlation
5	GOOG	2004-08-25	2.640104	184645512	2.61196	0.028144000000...	-121750647	No Clear Correlation
6	GOOG	2004-08-26	2.687676	142572401	2.640104	0.047572000000...	-42073111	No Clear Correlation
7	GOOG	2004-08-27	2.64384	124826132	2.687676	-0.04383600000...	-17746269	Negative Correlation
8	GOOG	2004-08-30	2.540727	104429967	2.64384	-0.10311300000...	-20396165	Negative Correlation
9	GOOG	2004-08-31	2.549693	98825037	2.540727	0.008966000000...	-5604930	No Clear Correlation
10	GOOG	2004-09-01	2.496891	183633734	2.549693	-0.05280199999...	84808697	No Clear Correlation
11	GOOG	2004-09-02	2.528273	303810504	2.496891	0.031381999999...	120176770	Positive Correlation
12	GOOG	2004-09-03	2.490913	103538639	2.528273	-0.03736000000...	-200271865	Negative Correlation
13	GOOG	2004-09-07	2.530017	117506800	2.490913	0.039104000000...	13968161	Positive Correlation
14	GOOG	2004-09-08	2.54795	100186120	2.530017	0.017933000000...	-17320680	No Clear Correlation

Visualizing the data for Google

```python
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have the data from the SQL query in a CSV file
df = pd.read_csv('/content/Correlation Analysis.csv')

# Convert the 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Filter data for a specific stock symbol if needed
symbol = 'GOOG'  # Replace with your stock symbol
df = df[df['Symbol'] == symbol]

# Create a scatter plot of PriceChange vs VolumeChange
# Points will be colored based on the CorrelationIndicator
plt.figure(figsize=(10, 6))
for correlation, group in df.groupby('CorrelationIndicator'):
    plt.scatter(group['PriceChange'], group['VolumeChange'], label=correlation, alpha=0.6)
plt.title(f'Price Change vs Volume Change Correlation for {Symbol}')
plt.xlabel('Price Change')
plt.ylabel('Volume Change')
plt.legend()
plt.grid(True)
plt.show()
```



Analysis:

Upon examining the scatter plot, we observe a concentration of data points near the origin, indicating days where changes in both price and volume were minimal. This area of dense plotting suggests days of stable trading with no significant news or events impacting the stock. The data points extending outward from the origin reflect days with more pronounced changes in price and/or volume. Notably, there are clusters of green points in the first and third quadrants, which signify days when both price and volume moved in the same direction—either both increasing or both decreasing. This can often be interpreted as a confirmation of a bullish or bearish sentiment, respectively.
Conversely, the blue points, primarily found in the second and fourth quadrants, indicate days when the price moved inversely to volume. Such a scenario may occur when price adjustments are not backed by a consensus in trading volume, often leading to uncertainty or speculative trading.
The orange points, scattered across all four quadrants without a clear pattern, represent days where no definitive correlation between price and volume changes could be established. These

points indicate that other factors may be at play, diluting the direct relationship between price and volume changes on those days.

Implications for Market Participants

For traders and analysts, understanding the correlation between price and volume changes is crucial. A positive correlation (both price and volume increasing) may suggest strong market conviction in the price movement, potentially signaling a continuation of the trend. In contrast, a negative correlation could indicate a potential reversal or lack of confidence in the price movement's sustainability.

The lack of correlation (orange points) adds complexity, implying that decisions on those days cannot be made based solely on price-volume dynamics. These situations require a deeper analysis of market conditions, including fundamental factors or broader economic indicators.

Conclusion

The correlation analysis between price and volume changes provides an essential perspective on market dynamics. This scatter plot serves as a visual tool for identifying how often and to what extent volume changes accompany price changes. By analyzing the distribution and color coding of the data points, market participants can infer prevailing market sentiments and potentially forecast future price movements. Understanding these patterns is key to developing robust trading strategies and managing investment risks.

Daily Stock Volatility

The SQL query calculates the daily volatility for each stock symbol based on the high and low prices for the day. It also computes the daily volatility as a percentage of the closing price. Additionally, it calculates the percentage change in daily volatility from the previous trading day.

DailyVolatility: The absolute difference between the high and low prices for the day, representing the total price movement range within the day.

DailyVolatilityPercent: The daily volatility expressed as a percentage of the closing price, providing a normalized measure of the day's price movement.

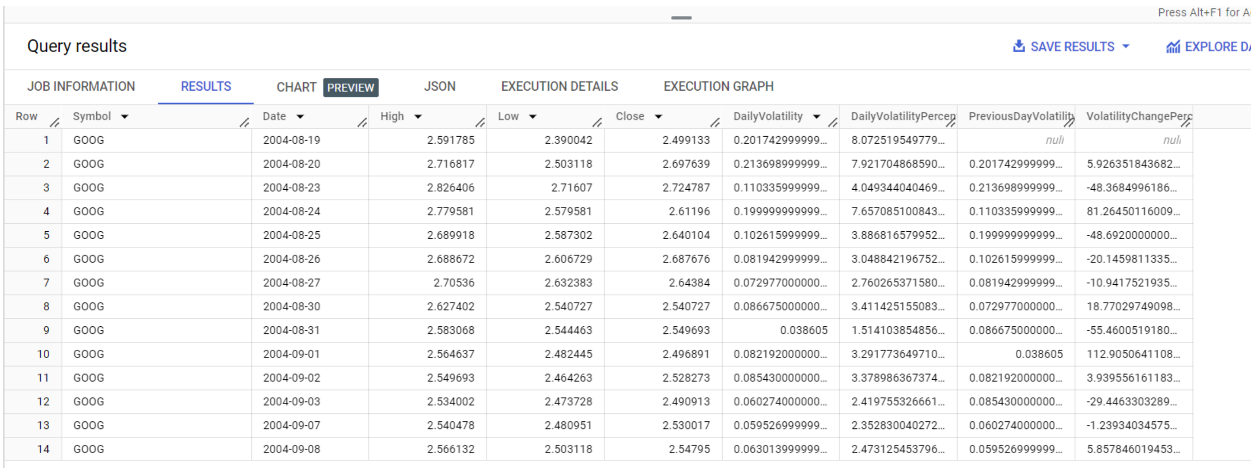
PreviousDayVolatility: The daily volatility from the previous trading day, used for comparison.

VolatilityChangePercent: The percentage change in daily volatility from the previous day, offering insight into how the day's price movement compares to the prior day.

Code

```
WITH DailyVolatility AS (
  SELECT
    Symbol,
    Date,
    High,
    Low,
    Close,
    ABS(High - Low) AS DailyVolatility,
    ABS(High - Low) / Close * 100 AS DailyVolatilityPercent
  FROM
    `mgmt-59000bd.Final_Project.Stocks`
)

SELECT
  DV.Symbol,
  DV.Date,
  DV.High,
  DV.Low,
  DV.Close,
  DV.DailyVolatility,
  DV.DailyVolatilityPercent,
  LAG(DV.DailyVolatility, 1) OVER (PARTITION BY DV.Symbol ORDER BY DV.Date) AS
PreviousDayVolatility,
  (DV.DailyVolatility - LAG(DV.DailyVolatility, 1) OVER (PARTITION BY DV.Symbol ORDER BY
DV.Date)) / LAG(DV.DailyVolatility, 1) OVER (PARTITION BY DV.Symbol ORDER BY DV.Date) *
100 AS VolatilityChangePercent
FROM
  DailyVolatility AS DV
ORDER BY
  DV.Symbol,
  DV.Date;
```

Result:

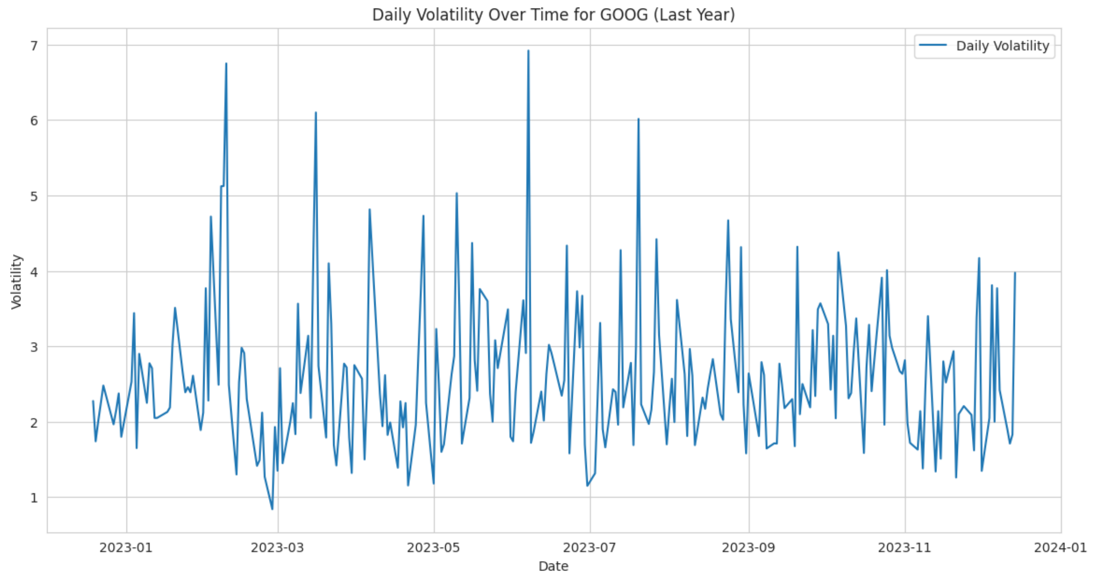


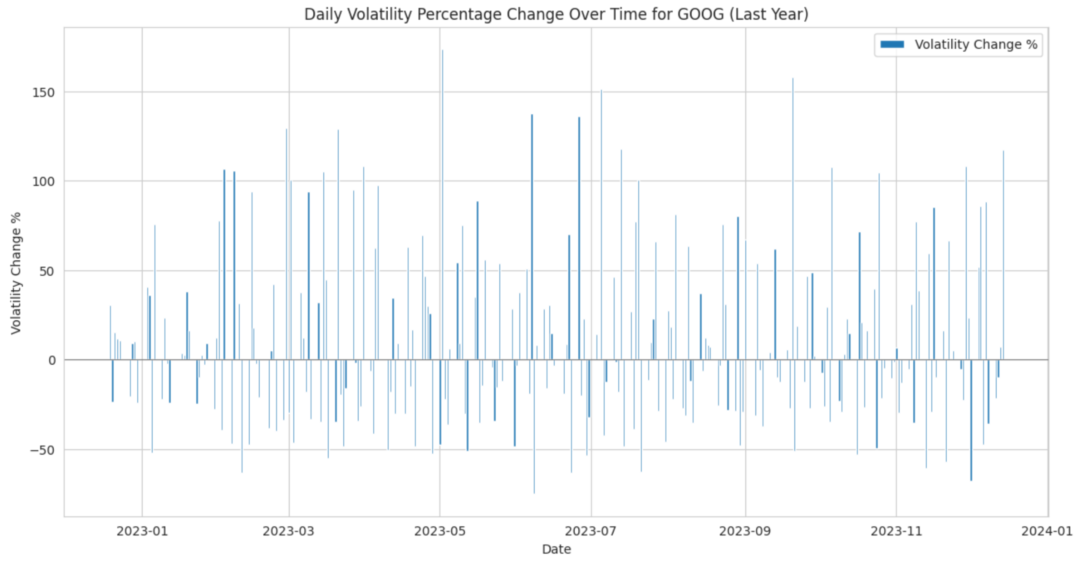
Row	Symbol	Date	High	Low	Close	DailyVolatility	DailyVolatilityPercen	PreviousDayVolatility	VolatilityChangePerc
1	GOOG	2004-08-19	2.591785	2.390042	2.499133	0.201742999999...	8.072519549779...	null	null
2	GOOG	2004-08-20	2.716817	2.503118	2.697639	0.213698999999...	7.921704868590...	0.201742999999...	5.926351843682...
3	GOOG	2004-08-23	2.826406	2.71607	2.724787	0.110335999999...	4.049344040469...	0.213698999999...	-48.3684996186...
4	GOOG	2004-08-24	2.779581	2.579581	2.61196	0.199999999999...	7.657085100843...	0.110335999999...	81.26450116009...
5	GOOG	2004-08-25	2.689918	2.587302	2.640104	0.102615999999...	3.886816579952...	0.199999999999...	-48.6920000000...
6	GOOG	2004-08-26	2.688672	2.606729	2.687676	0.081942999999...	3.048842196752...	0.102615999999...	-20.1459811335...
7	GOOG	2004-08-27	2.70536	2.632383	2.64384	0.072977000000...	2.760265371580...	0.081942999999...	-10.9417521935...
8	GOOG	2004-08-30	2.627402	2.540727	2.540727	0.086675000000...	3.411425155083...	0.072977000000...	18.77029749098...
9	GOOG	2004-08-31	2.583068	2.544463	2.549693	0.038605	1.514103854856...	0.086675000000...	-55.4600519180...
10	GOOG	2004-09-01	2.564637	2.482445	2.496891	0.082192000000...	3.291773649710...	0.038605	112.9050641108...
11	GOOG	2004-09-02	2.549693	2.464263	2.528273	0.085430000000...	3.378986367374...	0.082192000000...	3.939556161183...
12	GOOG	2004-09-03	2.534002	2.473728	2.490913	0.060274000000...	2.419755326661...	0.085430000000...	-29.4463303289...
13	GOOG	2004-09-07	2.540478	2.480951	2.530017	0.059526999999...	2.352830040272...	0.060274000000...	-1.23934034575...
14	GOOG	2004-09-08	2.566132	2.503118	2.54795	0.063013999999...	2.473125453796...	0.059526999999...	5.857846019453...

Visualizing Volatility

```
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

# Assuming you have the data from the SQL query in a CSV file
df = pd.read_csv('/content/Volatility.csv')

# Convert the 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Filter data for the last year
one_year_ago = datetime.now() - timedelta(days=365)
df_last_year = df[df['Date'] > one_year_ago]

# Filter data for a specific stock symbol if desired
symbol = 'GOOG'  # Replace with your stock symbol
df_symbol = df_last_year[df_last_year['Symbol'] == symbol]

# Plot daily volatility over time for the last year
plt.figure(figsize=(14, 7))
plt.plot(df_symbol['Date'], df_symbol['DailyVolatility'], label='Daily Volatility')
plt.title(f'Daily Volatility Over Time for {symbol} (Last Year)')
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.legend()
plt.show()
```

```
# Plot daily volatility percentage change over time for the last year
plt.figure(figsize=(14, 7))
plt.bar(df_symbol['Date'], df_symbol['VolatilityChangePercent'], label='Volatility Change %')
plt.title(f'Daily Volatility Percentage Change Over Time for {symbol} (Last Year)')
plt.xlabel('Date')
plt.ylabel('Volatility Change %')
plt.axhline(0, color='grey', linewidth=0.8)  # Add a line at y=0 for reference
plt.legend()
plt.show()
```





In financial markets, volatility is a measure of the dispersion of returns for a given security or market index, and it often signifies the risk associated with a security's changes in value. This section of the report focuses on the daily volatility of Alphabet Inc. (GOOG) over the past year, assessing both the absolute and relative measures of volatility.

Daily Volatility Over Time
The first visualization presents the daily volatility of GOOG, calculated as the absolute difference between the high and low prices within a single trading day. The plot shows fluctuations in the daily trading range for the past year. Periods with higher peaks suggest increased trading ranges, which often correspond to events or announcements that have a significant impact on investor sentiment and market activity. Conversely, lower points on the graph typically indicate days with less price movement, reflecting a more stable or certain market environment.

Daily Volatility Percentage Change Over Time
The second visualization depicts the percentage change in daily volatility compared to the previous day. This metric is particularly useful for identifying days with substantial increases or decreases in volatility from the previous trading session. Spikes in the chart can indicate sudden increases in uncertainty or market reactions to news or events, while sharp decreases may suggest a return to stability or the absorption of recent news into market prices.

Observations and Trends
Throughout the last year, we observe that volatility in GOOG's price has varied considerably, with some days experiencing wide trading ranges (as depicted by the tall spikes in the first graph) and others remaining relatively stable. The percentage change in volatility further highlights the variability from one day to the next, with some days seeing significant increases in volatility, potentially indicating market overreaction or the release of impactful news.

Conclusion
The daily volatility analysis for GOOG over the last year provides investors and market analysts with insight into the risk and uncertainty associated with the stock. Understanding these patterns can help inform better decision-making regarding entry and exit points for trades, risk management, and portfolio diversification strategies. The ability to anticipate and react to changes in volatility can be a valuable component of a comprehensive trading approach.

Model Creation

The script starts with the CREATE OR REPLACE MODEL statement, which is used to either create a new machine learning model or replace an existing model in BigQuery ML.

```
CREATE OR REPLACE MODEL `mgmt-59000bd.Final_Project.Stocks_Price_model_v3`
OPTIONS(
  MODEL_TYPE='ARIMA_PLUS',
  TIME_SERIES_TIMESTAMP_COL='Date',
  TIME_SERIES_DATA_COL='Close',
  TIME_SERIES_ID_COL='Symbol',
  HOLIDAY_REGION = 'US'
) AS
SELECT
  Date,
  Symbol,
  Close
FROM
  `mgmt-59000bd.Final_Project.Stocks`
```

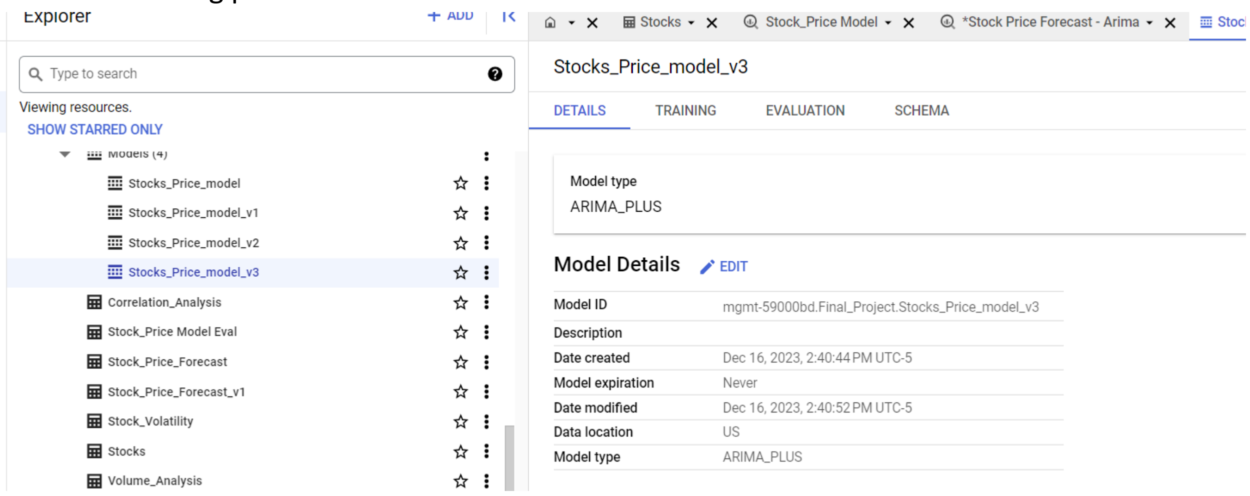
Model Options

The provided SQL script is for creating a machine learning model using Google BigQuery's ML capabilities, specifically for time series forecasting. Here's an explanation of each part of the command:

- **`CREATE OR REPLACE MODEL`:** This command tells BigQuery to create a new machine learning model or replace an existing one if it has the same name.
- **``mgmt-59000bd.Final_Project.Stocks_Price_model_v3``:** This specifies the dataset and model name where the new model will be stored, in this case, `Stocks_Price_model_v3` under the `Final_Project` dataset within the `mgmt-59000bd` project.
- **`OPTIONS`:** This clause allows you to set specific options for the machine learning model. The options used here are:
  - **`MODEL_TYPE='ARIMA_PLUS'`:** Specifies the type of model to be created. ARIMA_PLUS is an enhanced version of the ARIMA model (AutoRegressive Integrated Moving Average), which is commonly used for time series forecasting. It automatically handles seasonality and trends in the data.
  - **`TIME_SERIES_TIMESTAMP_COL='Date'`:** This option indicates which column in the data contains the timestamp for each observation. BigQuery ML will use this column to understand the order of the data.
  - **`TIME_SERIES_DATA_COL='Close'`:** Specifies the column that contains the metric you want to forecast. In this case, it's the closing price of the stock.
  - **`TIME_SERIES_ID_COL='Symbol'`:** This option tells the model that the data contains multiple time series identified by the symbol of the stock. It allows the model to differentiate between the time series for each stock symbol.
  - **`HOLIDAY_REGION = 'US'`:** This option instructs the model to consider US public holidays, which can have a significant impact on the stock market, as the market is usually closed on these days.

- `AS SELECT Date, Symbol, Close FROM 'mgmt-59000bd.Final_Project.Stocks'`: This part of the script is the query that selects the data to be used to train the model. The `FROM` clause specifies the fully-qualified table name, and the `SELECT` clause specifies the columns to be used in the model.

By running this command, BigQuery ML will create a new time series forecasting model tailored to predict future closing prices of stocks, considering each stock symbol as a separate time series. It will automatically take into account seasonality, trends, and US holidays, which could affect the trading patterns of stocks.

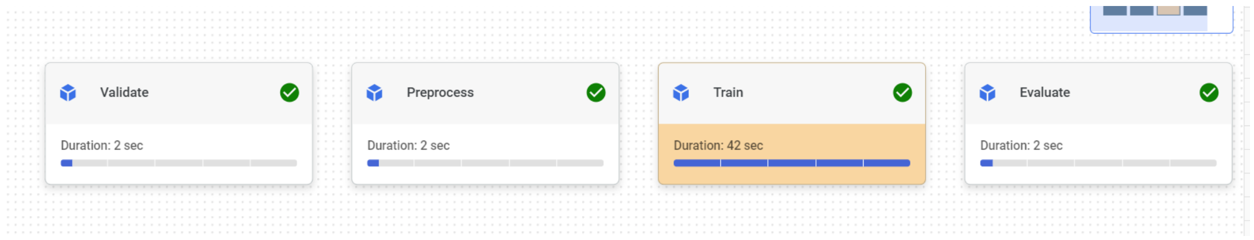


Model type
ARIMA_PLUS

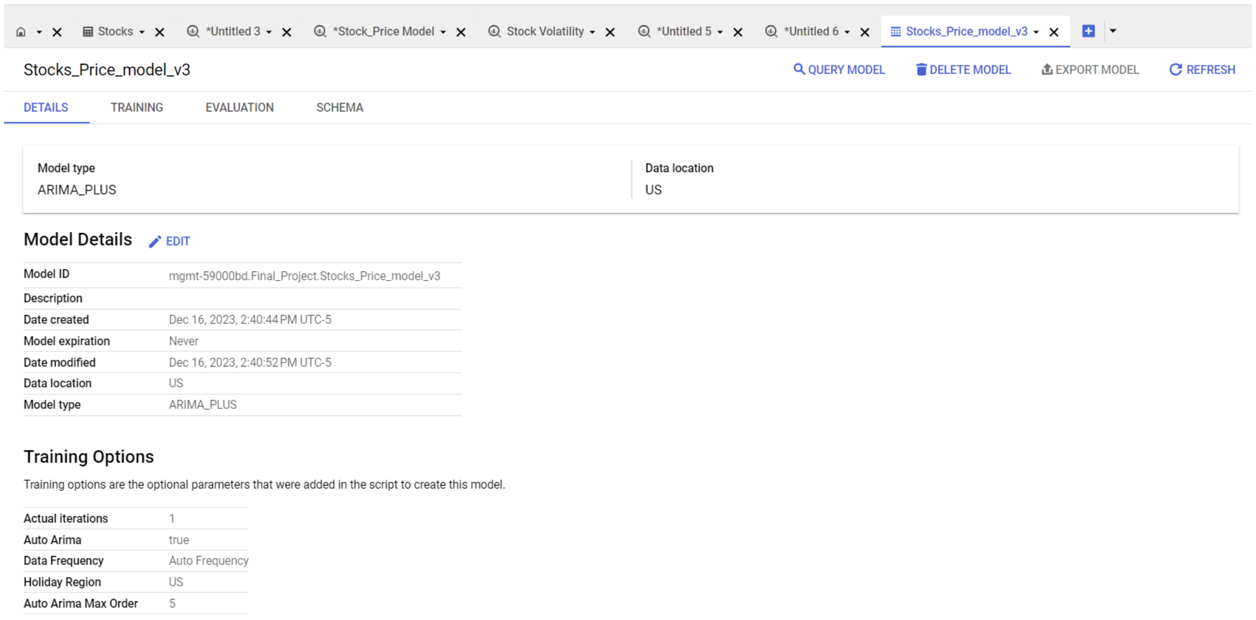
Model Details

Model ID	mgmt-59000bd.Final_Project.Stocks_Price_model_v3
Description	
Date created	Dec 16, 2023, 2:40:44 PM UTC-5
Model expiration	Never
Date modified	Dec 16, 2023, 2:40:52 PM UTC-5
Data location	US
Model type	ARIMA_PLUS

Query Visualization:



Model Metrics:



Stocks_Price_model_v3

Model type	ARIMA_PLUS
Data location	US

Model Details

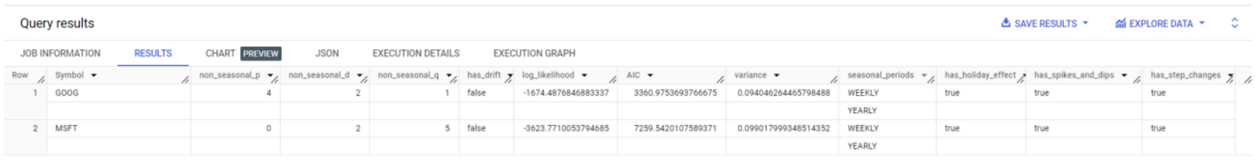
Model ID	mgmt-59000bd.Final_Project.Stocks_Price_model_v3
Description	
Date created	Dec 16, 2023, 2:40:44 PM UTC-5
Model expiration	Never
Date modified	Dec 16, 2023, 2:40:52 PM UTC-5
Data location	US
Model type	ARIMA_PLUS

Training Options

Training options are the optional parameters that were added in the script to create this model.

Actual iterations	1
Auto Arima	true
Data Frequency	Auto Frequency
Holiday Region	US
Auto Arima Max Order	5

Model Evaluation Metrics:



Query results

Row	Symbol	non_seasonal_p	non_seasonal_d	non_seasonal_q	has_drift	log_likelihood	AIC	variance	seasonal_periods	has_holiday_effect	has_spikes_and_dips	has_step_changes
1	GOOG	4	2	1	false	-1674.4876846883337	3360.9753693766675	0.094046264465798488	WEEKLY	true	true	true
									YEARLY			
2	MSFT	0	2	5	false	-3623.7710053794685	7259.5420107589371	0.099017999348514352	WEEKLY	true	true	true
									YEARLY			

The results of a time series model evaluation for two different stock symbols: GOOG (Google) and MSFT (Microsoft) are shown. The table presents several metrics and parameters that are used to understand the model's performance and its characteristics. Here is a breakdown of each column in the results:

- `Symbol`: The identifier for the stock. Each row represents the results for a different stock's time series model.
- `non_seasonal_p`: The number of non-seasonal autoregressive terms in the ARIMA model for each stock.
- `non_seasonal_d`: The number of non-seasonal differences. This value indicates how many times the data was differenced to achieve stationarity.
- `non_seasonal_q`: The number of non-seasonal moving average terms in the ARIMA model.
- `has_drift`: Whether the model includes a drift term, which is a steady trend over time.
- `log_likelihood`: A measure of the goodness of fit of the model. Higher values typically indicate a model that better fits the data.
- `AIC`: Akaike Information Criterion, a widely used measure of a statistical model. It deals with the trade-off between the goodness of fit of the model and the complexity of the model. Lower AIC values suggest a better model.

- `variance`: The variance of the model residuals. A lower variance indicates that the model's errors are smaller on average.
- `seasonal_periods`: Indicates the seasonal period detected in the data (e.g., WEEKLY, YEARLY).
- `has_holiday_effect`: Indicates whether the model has detected and adjusted for holiday effects in the data.
- `has_spikes_and_dips`: Whether the model has identified spikes and dips in the data.
- `has_step_changes`: Whether the model has identified any step changes in the data, which are sudden shifts in the level of the time series.

The results suggest that both models have detected seasonal patterns, with GOOG showing weekly seasonality and MSFT showing both weekly and yearly seasonality. Both models also account for holiday effects, spikes and dips, and step changes, which can all impact the time series of stock prices.

The differences in the `non_seasonal_p`, `non_seasonal_d`, and `non_seasonal_q` parameters between the two symbols suggest that the underlying time series patterns of GOOG and MSFT differ, requiring different ARIMA configurations to model effectively. The presence of a drift term in the MSFT model but not in the GOOG model indicates that there is a long-term trend in MSFT's stock prices which is not as pronounced in GOOG's stock prices.

The `log_likelihood` and `AIC` values are specific to each model and are used to compare different models for the same time series. Since they are from different stocks, they should not be directly compared against each other. However, within the context of a single stock's models, a higher log likelihood and a lower AIC would typically be preferred.

The `variance` of the residuals would give an idea of the noise in the model predictions; lower values are generally preferred, indicating that the model predictions are closer to the true values.

This information is crucial for analysts and data scientists when evaluating the performance of time series models for stock price forecasting. It helps in refining the models to improve predictions and understanding the behavior and characteristics of different stocks.

Forecasting Price:

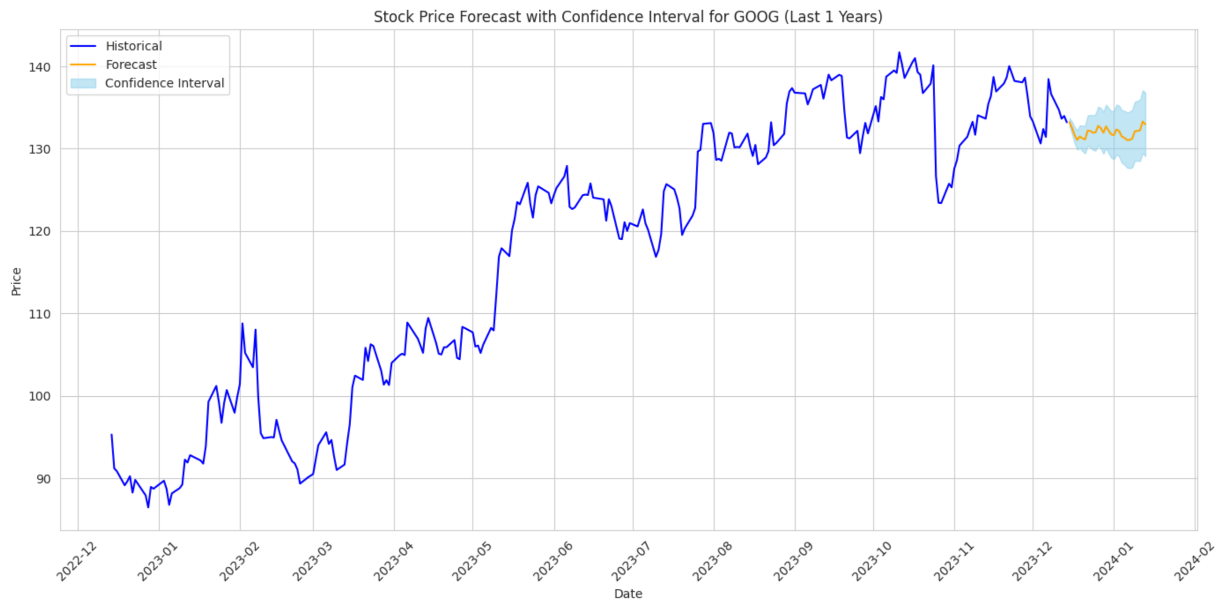
```
SELECT
  *
FROM
  ML.FORECAST(MODEL `mgmt-59000bd.Final_Project.Stocks_Price_model_v3`, STRUCT(30 AS
horizon, 0.9 AS confidence_level))
```

Visualizing the Forecasted prices

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from datetime import timedelta

# Load the forecasted and historical stock price data
forecast_df = pd.read_csv('/content/Forcasted_Price_v2.csv')
historical_df = pd.read_csv('/content/Stocks.csv')

# Convert the date columns from string to datetime objects
forecast_df['forecast_timestamp'] = pd.to_datetime(forecast_df['forecast_timestamp'])
historical_df['Date'] = pd.to_datetime(historical_df['Date'])

# Define the timeframe as past 5 years
end_date = historical_df['Date'].max()
start_date = end_date - timedelta(days= 1 * 365)

# Filter the historical data for the past 5 years
historical_data = historical_df[(historical_df['Date'] >= start_date) & (historical_df['Date'] <=
end_date)]

# Filter data for a specific stock symbol
symbol = 'MSFT'  # Replace with your stock symbol
forecast_data = forecast_df[forecast_df['Symbol'] == symbol]
historical_data = historical_data[historical_data['Symbol'] == symbol]

# Plot the historical data
plt.figure(figsize=(14, 7))
plt.plot(historical_data['Date'], historical_data['Close'], label='Historical', color='blue')

# Plot the forecasted data
plt.plot(forecast_data['forecast_timestamp'], forecast_data['forecast_value'], label='Forecast',
color='orange')
```

```
# Plot the confidence intervals
plt.fill_between(forecast_data['forecast_timestamp'],
            forecast_data['confidence_interval_lower_bound'],
            forecast_data['confidence_interval_upper_bound'],
            color='skyblue', alpha=0.5, label='Confidence Interval')

# Set plot title and labels
plt.title(f'Stock Price Forecast with Confidence Interval for {symbol} (Last 1 Years)')
plt.xlabel('Date')
plt.ylabel('Price')

# Improve the x-axis date format
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())

# Show legend
plt.legend()

# Show grid
plt.grid(True)

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```

Google:



The graph represents a stock price forecast with a confidence interval for Alphabet Inc. (GOOG) over the last year. The graph is divided into three main components:

1. Historical Data (in blue): This line shows the actual historical closing prices of GOOG stock over the last year up to the point where the forecast begins. It provides a context for the model's forecast by illustrating past performance.

2. Forecast (in orange): Starting from where the historical data ends, this line shows the predicted future closing prices for GOOG stock. The forecast extends beyond the last point of historical data, giving investors an idea of where the stock price might go based on the model's interpretation of past price movements.

3. Confidence Interval (shaded area): Surrounding the forecast, this shaded region represents the model's confidence interval—typically at a 95% level, though the exact level isn't specified in the graph. This interval shows the range within which the actual future price is expected to fall with a certain level of confidence. The width of the interval indicates the level of uncertainty in the prediction: a wider interval suggests greater uncertainty, and a narrower interval suggests more confidence in the forecasted prices.

Key observations from the graph include:

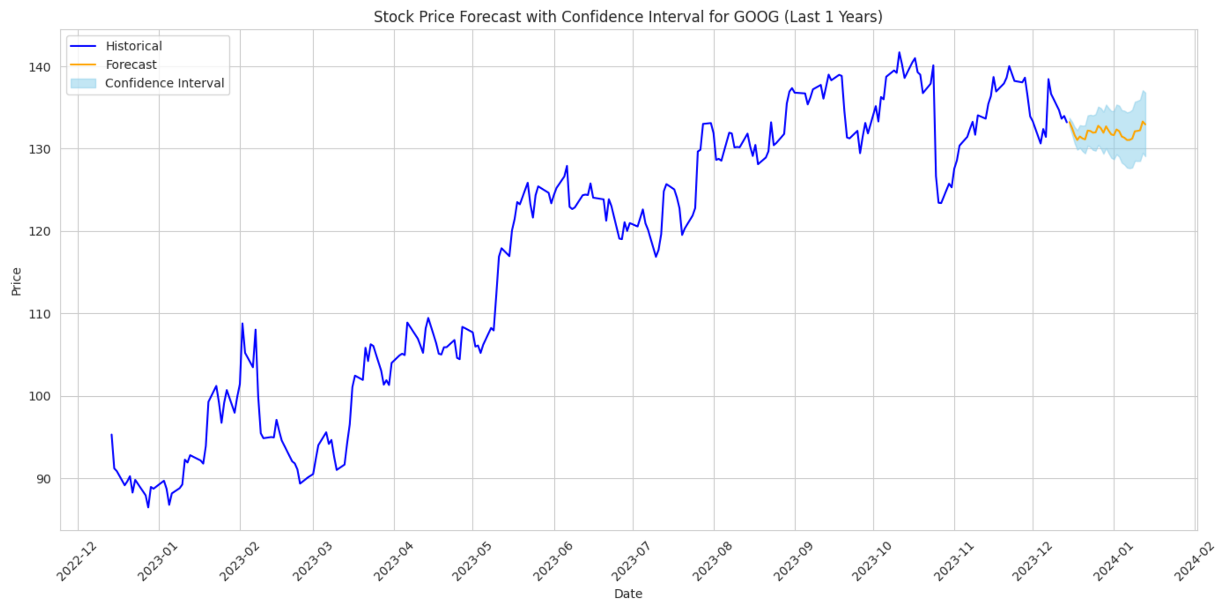
- The historical trend shows the stock's closing price movements have been volatile, with notable ups and downs throughout the year.

- The forecast suggests a relatively stable or slightly increasing trend in the closing price of GOOG stock for the upcoming period.

- The confidence interval widens as the forecast extends further into the future, indicating increasing uncertainty as the model projects further away from the last known data point.

This type of visualization is useful for investors and financial analysts as it not only provides a prediction of future stock prices but also quantifies the uncertainty associated with these predictions. It can be used as part of a broader investment strategy to assess potential risks and returns when making decisions about buying or selling stocks.

Microsoft



The provided graph illustrates the stock price forecast with a confidence interval for Microsoft Corporation (MSFT) over the last year. Here's a breakdown of the graph's components and their significance:

1. Historical Data (Blue Line): This line represents the actual historical closing prices of MSFT stock. It provides a visual representation of the stock's performance, showing how the price has fluctuated over the past year.

2. Forecast (Orange Line): Starting from the end of the historical data, the orange line indicates the predicted future closing prices for MSFT stock. The forecast is generated by a time-series model, which uses historical price data to predict future trends.

3. Confidence Interval (Shaded Area): The shaded area around the forecast line represents the confidence interval—often set at a 95% confidence level. This interval shows the range of prices within which the actual future price is expected to fall, according to the model's predictions. The interval's width suggests the degree of certainty or uncertainty associated with the forecast; a wider interval means more uncertainty, while a narrower one implies higher confidence in the prediction.

Key Insights:

- The historical trend shows a general increase in MSFT stock prices over the analyzed period, with some volatility.

- The forecasted trend suggests that the closing price of MSFT will continue in a similar pattern to the historical trend, with some fluctuations expected.

- The confidence interval widens as the forecast moves further into the future, reflecting the increasing uncertainty associated with longer-term predictions.

This graph can be used by investors and analysts to understand the potential future movements of MSFT stock prices and to make informed decisions based on the model's predictions. It also highlights the inherent uncertainty in stock price forecasting, as actual future prices may still fall outside the predicted confidence interval due to unforeseen market movements or events.

Anomaly Detection

The code is aimed at identifying anomalies in stock prices over the last year. The analysis utilizes a dataset that contains daily stock prices and a flag indicating whether each data point has been identified as an anomaly.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

# Load the anomaly data from the CSV file
file_path = '/content/Anomalies.csv'  # Replace with the path to your anomaly data file
anomalies_df = pd.read_csv(file_path)

# Convert 'Date' column to datetime format (removing timezone if present)
anomalies_df['Date'] = pd.to_datetime(anomalies_df['Date']).dt.tz_localize(None)

# Filter the DataFrame for the last year
one_year_ago = datetime.now() - timedelta(days=365)
df_last_year = anomalies_df[anomalies_df['Date'] > one_year_ago]

# Plotting the data
fig, ax = plt.subplots(figsize=(15, 7))

# Group by Symbol to plot each symbol's data in one plot
for symbol in df_last_year['Symbol'].unique():
    symbol_data = df_last_year[df_last_year['Symbol'] == symbol]
    ax.plot(symbol_data['Date'], symbol_data['Close'], label=symbol)

    # Highlight anomalies
    anomalies = symbol_data[symbol_data['is_anomaly']]
    ax.scatter(anomalies['Date'], anomalies['Close'], edgecolors='r', facecolors='none', s=100,
label=f'Anomaly in {symbol}')

ax.set_title('Stock Price Anomalies Over the Last Year')
ax.set_xlabel('Date')
ax.set_ylabel('Close Price')
ax.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Code Explanation

- **Data Loading:** The code begins by importing necessary Python libraries—pandas for data manipulation, matplotlib.pyplot for data visualization, and datetime for handling date and time objects. It then loads the dataset containing stock price information and anomaly flags from a CSV file into a pandas DataFrame.

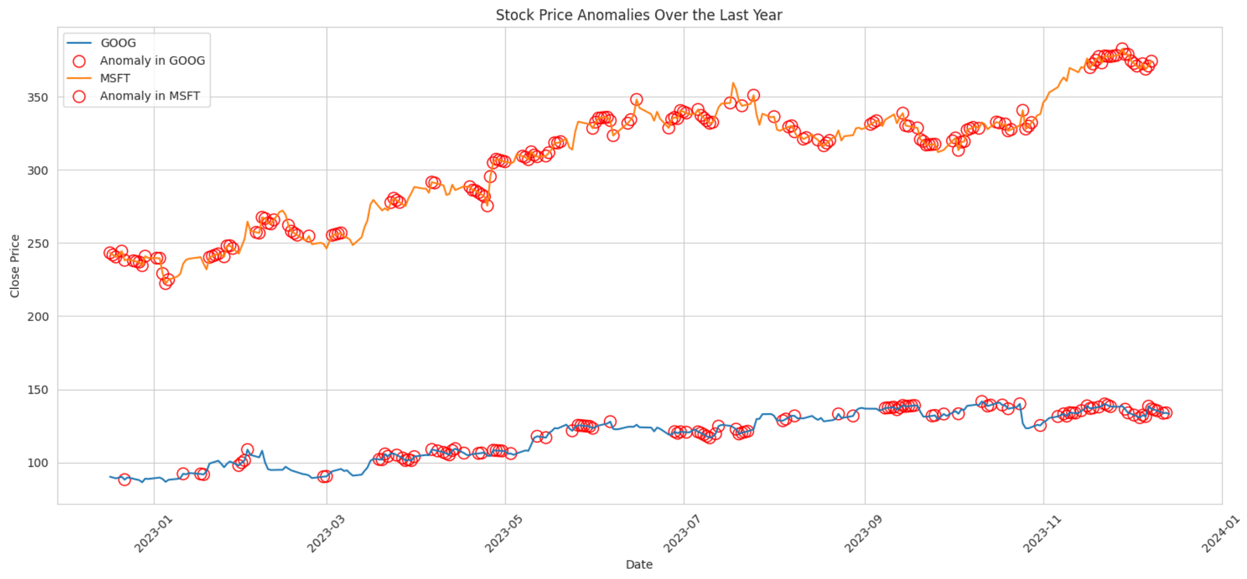
- **Date Conversion:** The 'Date' column is initially in string format, possibly containing timezone information. The code converts this column to datetime objects, standardizing it to be timezone-naive (i.e., removing any timezone information), allowing for consistent date-time comparisons.

- **Filtering Data:** A filter is applied to select only the records from the past year. This is achieved by calculating a date (one_year_ago) that is 365 days earlier than the current date and selecting rows where the 'Date' is more recent than this threshold.

- **Plotting Data:** The code proceeds to plot the closing prices for each stock symbol on a line graph. Anomalies within this one-year period are highlighted with red circles, visually indicating where unusual price movements have occurred.

- **Visualization Customization:** The plot is titled 'Stock Price Anomalies Over the Last Year', and axes are labeled appropriately. The legend distinguishes between normal price data and anomalies. X-axis labels are rotated for better readability.

Result



The graph provided visualizes stock prices for two symbols, GOOG (Google) and MSFT (Microsoft), and identifies anomalies in their price movements over the last year. Each symbol's daily closing price is plotted as a line graph with anomalies highlighted by red circles.

Analysis

GOOG (Google):

- The price of GOOG shows a general upward trend over the year, indicative of growth.
- A significant number of anomalies (red circles) are spread across the entire year. The presence of these anomalies could be indicative of unexpected news events, earnings reports, or market reactions to industry developments.
- Some clusters of anomalies suggest periods of high volatility. Investors might look into news events or company announcements on these dates for more context.

MSFT (Microsoft):

- MSFT's stock price exhibits a relatively stable trend with less volatility compared to GOOG.
- Anomalies for MSFT are also less frequent, but they appear in clusters, potentially corresponding to specific events affecting the company or its stock.
- The stable trend outside of these anomalies may appeal to investors who prefer less volatility.

Business Insights and Trading Tips

1. Volatility Analysis:

- GOOG's higher volatility may offer more opportunities for traders who capitalize on short-term price movements.
- MSFT's lower volatility might be more suitable for long-term investors seeking steady growth with fewer risks.

2. Event-Driven Trading:

- The anomalies could correspond to market reactions to specific events. Traders could look into historical news to identify patterns in market reactions and refine their event-driven trading strategies.

3. Risk Management:

- The clusters of anomalies might indicate periods of increased risk. Traders and risk managers could investigate further and potentially adjust their risk management strategies accordingly.

4. Diversification:

- The different volatility profiles of GOOG and MSFT suggest they could play different roles in a diversified investment portfolio.

5. Algorithmic Trading:

- Anomalies may be used to develop or improve algorithmic trading strategies, where the algorithm could be trained to react when similar patterns occur in the future.

6. Market Research:

- Researching the causes behind the identified anomalies could provide deeper insights into market dynamics and inform both trading and investment decisions.

7. Investor Communication:

- Companies might use such a graph to communicate with investors about stock price fluctuations and reassure them by providing context for anomalies.

It's important to note that anomaly detection is not a definitive indicator of future performance. Anomalies simply flag data points that deviate from a typical pattern, which may or may not be significant in forecasting future prices. Always corroborate anomaly data with other forms of analysis and market intelligence.

Alternative Approach to Building the Model (Vertex AI Workbench)

Code:

```
#LSTM Model code
import numpy as np
import pandas as pd
from pandas.io import gbq
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# Replace this with your Google Cloud Project ID
project_id = 'mgmt590-396517'
# Replace this with your BigQuery table
dataset_table = 'mgmt590-396517.finalprojectarchivedata.sorted_consolidated_table'

# Fetch the data from BigQuery
query = f"""
SELECT date, Symbol, open, high, low, close, volume
FROM `{dataset_table}`
ORDER BY date
"""
df = gbq.read_gbq(query, project_id=project_id)

# Preprocess the data
df['date'] = pd.to_datetime(df['date']).dt.dayofyear.astype('float32')  # Convert to day of year and cast to float
df = pd.get_dummies(df, columns=['Symbol'], dtype='float32')  # Create dummy variables for 'Symbol' as float32

#  Feature Scaling
```

```
feature_scaler = MinMaxScaler(feature_range=(0, 1))
df[['open', 'high', 'low', 'volume']] = feature_scaler.fit_transform(df[['open', 'high', 'low',
'volume']].astype('float32'))

# Scale close price separately
close_scaler = MinMaxScaler(feature_range=(0, 1))
df['close'] = close_scaler.fit_transform(df['close'].values.reshape(-1,1).astype('float32'))

# Prepare the dataset for LSTM
def create_dataset(X, y, time_step=1):
    Xs, ys = [], []
    for i in range(len(X) - time_step):
        v = X.iloc[i:(i+time_step)].values
        Xs.append(v)
        ys.append(y[i + time_step])
    return np.array(Xs), np.array(ys)

time_step = 100
features = ['date', 'open', 'high', 'low', 'volume', 'Symbol_GOOG', 'Symbol_MSFT']  # Features
including dummy variables
X = df[features]
y = df['close'].values  # Target variable

# Create the LSTM dataset
X_lstm, y_lstm = create_dataset(X, y, time_step)
X_train, X_test, y_train, y_test = train_test_split(X_lstm, y_lstm, test_size=0.2,
random_state=0)

# Reshape input to be [samples, time steps, features] and cast to float32
X_train = X_train.reshape(X_train.shape[0], time_step, X_train.shape[2]).astype('float32')
X_test = X_test.reshape(X_test.shape[0], time_step, X_test.shape[2]).astype('float32')

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(time_step, len(features))))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=25))
model.add(Dense(units=1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

```
# Predict and inverse transform the results
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
# Inverse transform the predictions
train_predict = close_scaler.inverse_transform(train_predict)
test_predict = close_scaler.inverse_transform(test_predict)

# Plot the results
plt.figure(figsize=(12,6))
train_index = df.index[time_step:len(train_predict)+time_step]
test_index =
df.index[len(train_predict)+(time_step*2):len(train_predict)+(time_step*2)+len(test_predict)]

# Inverse transform the original close price
actual_train = close_scaler.inverse_transform(y_train.reshape(-1,1))
actual_test = close_scaler.inverse_transform(y_test.reshape(-1,1))

plt.plot(train_index, actual_train, label='Train Data')
plt.plot(test_index, actual_test, label='Actual Close Price')
plt.plot(test_index, test_predict, label='Predictions')

plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Stock Price Prediction with LSTM')
plt.legend()
plt.show()
```

Created Model

my_lstm_model
1.keras

This code snippet outlines the process of building a Long Short-Term Memory (LSTM) neural network to forecast stock prices using historical stock data fetched from a Google BigQuery table. Here's a step-by-step explanation of the code:

Setup and Data Retrieval

1. Import necessary Python libraries for data manipulation, machine learning, and visualization.
2. Define `project_id` and `dataset_table` variables with the Google Cloud Project ID and the BigQuery dataset table, respectively.
3. Use a SQL query to fetch stock data ordered by date from the specified BigQuery table.

4. Read the query results into a pandas DataFrame using the `read_gbq` function.

Data Preprocessing
5. Convert the 'date' column to a numerical format representing the day of the year and cast it to a float32 data type.
6. Use one-hot encoding to convert categorical 'Symbol' values into dummy/indicator variables, which are necessary for the model to process non-numeric data.
7. Scale the 'open', 'high', 'low', and 'volume' features using MinMaxScaler to normalize the data within a range of [0, 1].
8. Scale the 'close' prices separately, as this will be the target variable for prediction.

Dataset Preparation for LSTM
9. Define a function `create_dataset` that creates sequences of the historical data with the corresponding target stock price. This is necessary for training the LSTM, which requires input data in sequences to capture temporal dependencies.
10. Set the `time_step` variable, which specifies how many days of data the model should look at to make a prediction.
11. Prepare the input features (`X`) and the target variable (`y`), which is the 'close' price.
12. Use the `create_dataset` function to transform the data into a format suitable for the LSTM, resulting in `X_lstm` and `y_lstm`.
13. Split the data into training and testing sets using `train_test_split`.

LSTM Model Building
14. Reshape the input data to be [samples, time steps, features], which is required for LSTM models in TensorFlow/Keras.
15. Initialize a Sequential model and add LSTM layers. LSTM layers are configured to have 50 units each, with the first returning sequences and the second not, to connect to a Dense layer afterward.
16. Add Dense layers with 25 units and a final output layer with 1 unit corresponding to the predicted 'close' price.
17. Compile the LSTM model using the 'adam' optimizer and mean squared error loss function.

Model Training
18. Fit the model to the training data for a specified number of epochs and batch size.

Prediction and Visualization
19. Predict the 'close' prices for the training and testing datasets using the trained model.
20. Inverse the scaling transformation to return the predicted prices to their original scale.

21. Plot the training data, actual 'close' prices, and predicted prices to visualize the model's performance.
22. Label the axes, title the plot, and add a legend.

The final visualization will show how well the LSTM model's predictions align with the actual stock prices. This can help assess the model's performance and potentially guide investment decisions. It's important to note that stock price prediction is inherently uncertain, and such models should be used with caution.

Challenges and Learnings

1. Data Integration Complexity: One of the primary challenges was integrating various data sources with differing formats and frequencies. This required meticulous planning and execution to ensure seamless data flow and synchronization.
2. Real-Time Data Processing: Processing real-time data presented unique challenges, particularly in ensuring data accuracy and minimizing latency. We learned the importance of efficient pipeline architecture and the judicious use of resources.
3. Machine Learning Model Optimization: Developing and tuning machine learning models for stock price prediction and anomaly detection was a complex task. We learned the intricacies of model selection, feature engineering, and the crucial role of data quality in model performance.
4. Scalability and Performance: Ensuring that our system could scale effectively while maintaining high performance was challenging. This taught us valuable lessons in designing scalable architectures and the importance of regular performance assessments.
5. User Interface and Experience Design: Creating intuitive and insightful Data Studio dashboards was challenging. We learned about the importance of user-centered design and iterative development based on user feedback.
6. Security and Compliance: Ensuring data security and compliance with financial regulations was a priority. This experience underscored the importance of robust security measures and staying updated with compliance requirements.
7. Team Collaboration and Project Management: Coordinating a multifaceted project with various components required effective teamwork and project management. This experience honed our skills in communication, collaboration, and agile project management methodologies.

These challenges provided valuable learning opportunities, contributing to our professional growth and enhancing our ability to tackle complex projects in the future.

Project Enhancements Scope

1. **Advanced Visualization:** We will integrate dynamic and interactive visualization techniques in Data Studio to improve user engagement and data interpretability.
2. **Diverse Data Integration:** Future iterations will include a variety of financial indicators and external factors to provide a more comprehensive analysis.

3. **Scalability and Performance:** Continuous assessment and optimization of data pipelines and machine learning models will be conducted to handle larger datasets efficiently.
4. **Client-Centered Approach:** All aspects of our analysis will be aligned closely with potential client needs, focusing on investment strategies and market trends.
5. **Robust Testing and Feedback**: We will implement a rigorous testing protocol and actively seek community feedback to refine and improve our project.

Conclusion and Future Directions

Our project, leveraging the robust capabilities of Google Cloud Platform, represents a significant stride in the realm of financial analytics. By intricately weaving together BigQuery, Cloud Pub/Sub, Dataflow, and Data Studio with Vertex AI, we've created a system that not only predicts stock price movements with enhanced accuracy but also identifies anomalies in real-time data, a critical tool for investors in today's volatile market.

The potential of our project extends beyond its current capabilities. Future enhancements will focus on integrating a wider array of data sources, including economic indicators and market sentiment analyses. This will provide a more holistic view of the factors influencing stock prices. Furthermore, we plan to explore the integration of more advanced machine learning models and deep learning techniques to further refine our predictions and anomaly detection capabilities.

Our commitment to scalability and adaptability will ensure that our system remains effective and efficient as it handles increasingly larger datasets. This will be particularly important as the financial market evolves and the volume of data available for analysis continues to grow.

We also emphasize a client-centered approach, ensuring that our analyses and predictions align with the real-world needs and strategies of investors and financial analysts. This approach will be continuously refined through active engagement with our user base and the incorporation of their feedback into our system.

In summary, this project is not just a culmination of our current learnings but also a foundation for ongoing exploration and innovation in the field of financial data analytics.