Analyzing Layoff Data and Stock Performance

A SQL-Graph Hybrid Model

Team3:
Qinwei Wu 002191775
Kanimozhi Velusamy 002809300
Jiani Lyu 002299940
Cheril Yogi 002790646

Agenda

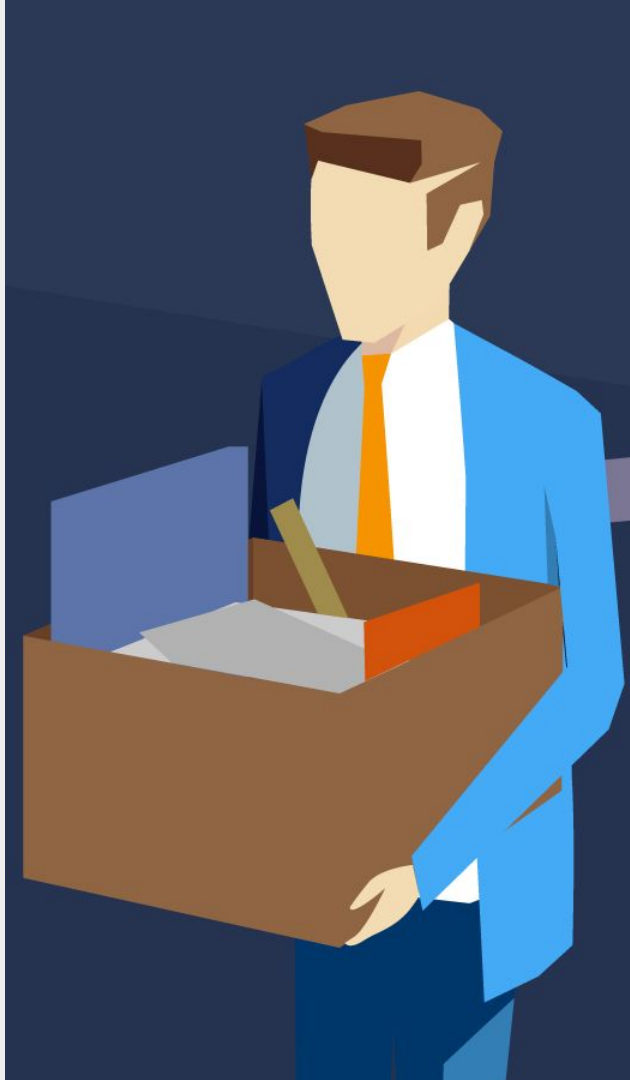
Introduction

Data Architecture

ER Diagram and Data Source

Graph Implementation

Reports and Analysis



Recent Events…

Bloomberg

Live TV Markets Economics Industries Tech Politics Businessweek Opinion More

Technology

Apple Cut at Least 600 Workers When Car, Scree

By Mark Gurman

April 4, 2024 at 7:52 PM EDT

Gift this article

Save

In this Article

The New York

Amazon Is Said to Pl… Thousands of Employees

The job cuts of approximately 10,000, which would start as soon as this week, would focus on the company's devices organization, retail division and human resources.

yahoo/finance

Facebook parent Meta to lay off 10,000 more workers amid 'Year of Efficiency' push

Alexandra Garfinkle and Daniel Howley

Updated Tue, Mar 14, 2023 • 3 min read

232

AP WORLD U.S. ELECTION 2024

The Masters Sydney ma…

BUSINESS

Unilever to cut 7,500 jobs and spin off its ice cream business, which includes Ben & Jerry's

Bloomberg

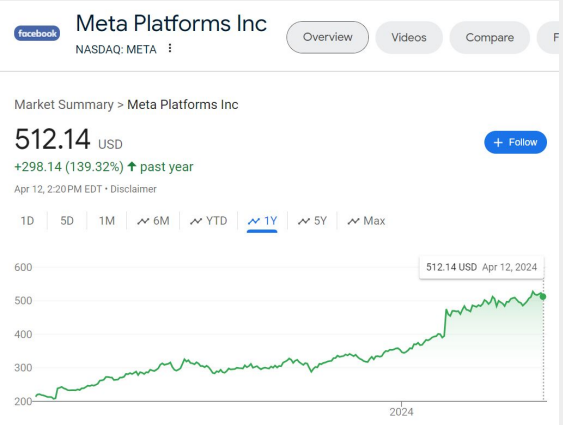
Live TV Markets Economics Industries Tech Politics Businessweek Opinion More

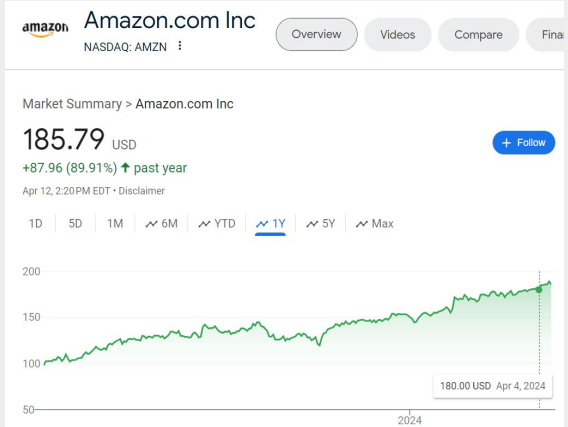
Technology

Google to Cut 12,000 Jobs in 6… Reduction of Global Workforce

- Company is latest tech giant to significantly reduce headcount
- Severance packages include vacation, bonus pay and health care

Recent Events…





Data Architecture: Our Model

Relational Model - Layoff:

Structured, easy to navigate with companies and events,

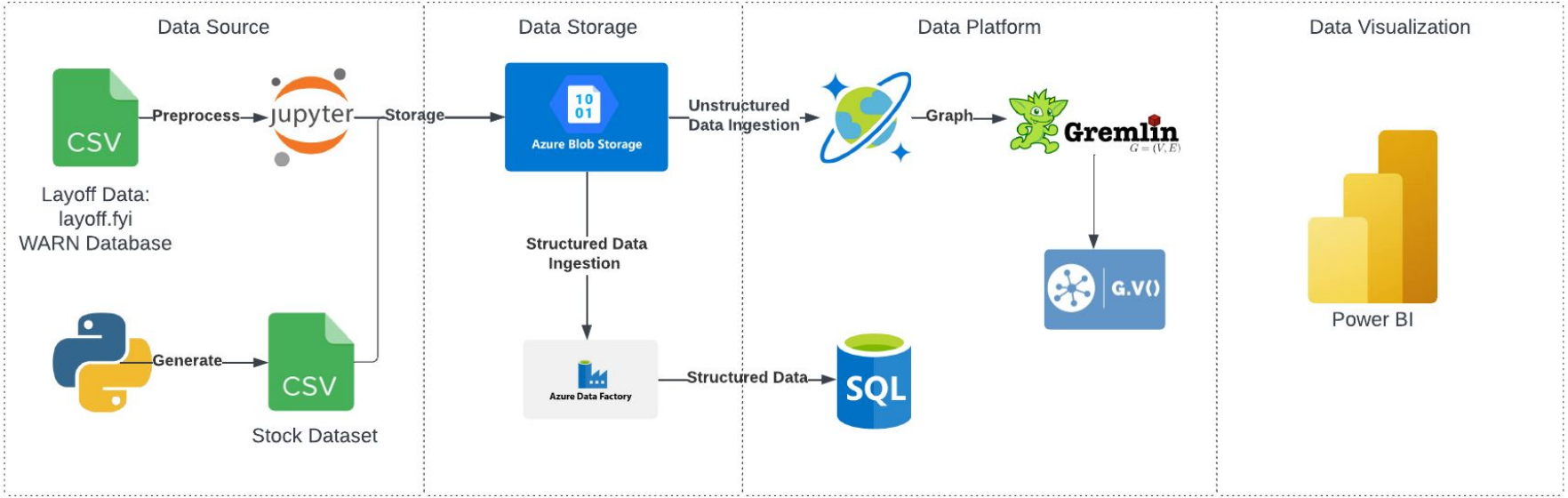
Document Model - Stock:

Unstructured, horizontally scalable, fast read/write

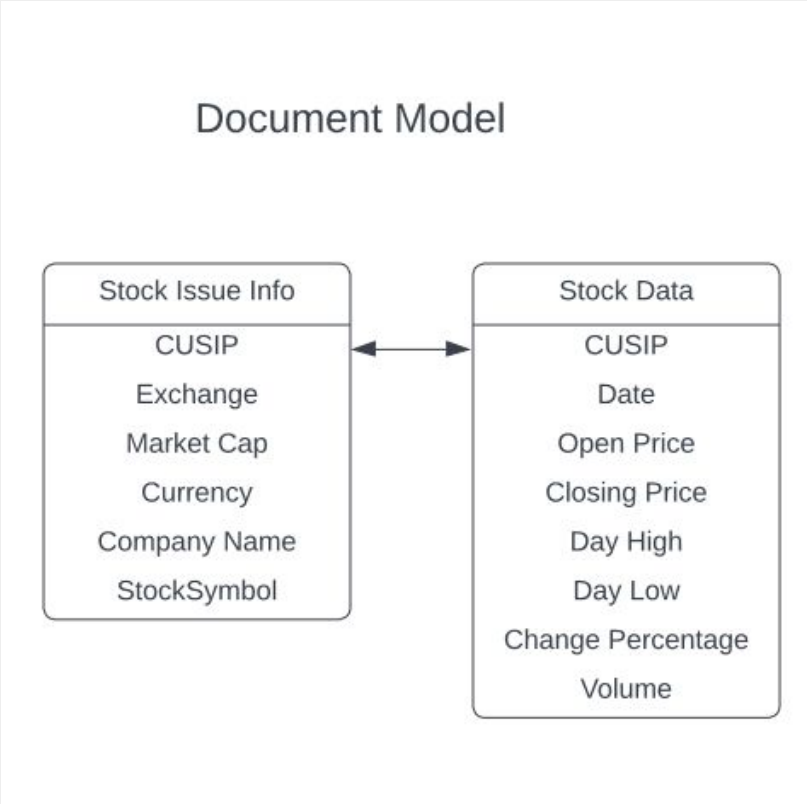
Graph Model:

Unstructured, dynamic and interconnected relationships

Data Architecture: Diagram



ERD: Document



Data Source: Stock Company Overview

- We arbitrarily chose 30 public companies that cover a various of different backgrounds.
- We manually scouted those public information from online data sources such as Bloomberg, Yahoo Finance, etc.
- We will not be refreshing those stock information.

	A	B	C	D	E	F
1	Company	Ticker	Exchange	CUSIP	Currency	Market Cap
2	Alphabet Inc Class C	GOOG	NASDAQ	02079K107	USD	1,884.6B
3	Amazon.com Inc	AMZN	NASDAQ	23135106	USD	1,873.7B
4	Buzzfeed Inc	BZFD	NASDAQ	12430A102	USD	68.0M
5	Chegg Inc	CHGG	NASDAQ	163092109	USD	779.3M
6	Cisco Systems Inc	CSCO	NASDAQ	17275R102	USD	202,094.9M
7	DoorDash Inc	DASH	NASDAQ	25809K105	USD	55,639.5M
8	Dropbox Inc	DBX	NASDAQ	26210C104	USD	8,358.8M
9	eBay Inc	EBAY	NASDAQ	278642103	USD	27,340.0M
10	Enphase Energy Inc	ENPH	NASDAQ	29355A107	USD	16,424.2M
11	Etsy Inc	ETSY	NASDAQ	29786A106	USD	8,142.8M
12	Lemonade Inc	LMND	NYSE	52567D107	USD	1,152.4M
13	Lucid Group Inc	LCID	NASDAQ	549498103	USD	6,555.1M
14	LYFT Inc	LYFT	NASDAQ	55087P104	USD	7,736.3M
15	Meta Platforms Inc	META	NASDAQ	30303M102	USD	1,237.9B
16	Micron Technology Inc	MU	NASDAQ	595112103	USD	130,547.6M
17	Microsoft Corp	MSFT	NASDAQ	594918104	USD	3,126.1B
18	Netflix Inc	NFLX	NASDAQ	64110L106	USD	262,827.9M
19	Opendoor Technologies Inc	OPEN	NASDAQ	683712103	USD	2,059.3M
20	PayPal Holdings Inc	PYPL	NASDAQ	70450Y103	USD	71,796.0M
21	Reddit Inc	RDDT	NYSE	75734B100	USD	8,004.3M
22	SecureWorks Corp	SCWX	NASDAQ	81374A105	USD	593.3M
23	Thoughtworks Holding Inc	TWKS	NASDAQ	88546E105	USD	816.5M
24	Tripadvisor Inc	TRIP	NASDAQ	896945201	USD	3,832.2M
25	TrueCar Inc	TRUE	NASDAQ	89785L107	USD	309.1M
26	Uber Technologies Inc	UBER	NYSE	90353T100	USD	160,258.1M
27	Unity Software Inc	U	NYSE	91332U101	USD	10,304.7M
28	Vroom Inc	VRM	NASDAQ	92918V208	USD	24.1M
29	Wayfair Inc	W	NYSE	94419L101	USD	8,170.0M
30	Workday Inc	WDAY	NASDAQ	98138H101	USD	72,006.0M
31	Zoom Video Communications Inc	ZM	NASDAQ	98980L101	USD	20,105.2M

Data Source: Stock Data

- Due to the widely available properties of stock data, we were able to fetch this data from several sources.
- After experimenting, we are using a Python script to pull data from Yahoo Finance daily.
- We are analyzing data from 2022/1/1 to current datetime.

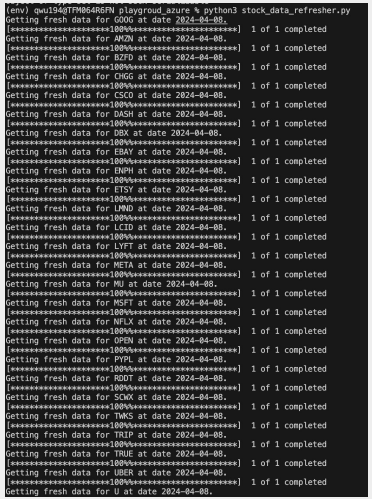
	Date	Open	High	Low	Close	Volume	Percent Change
2	1/3/2022 16:00:0	144.48	145.55	143.5	145.07	1261225	0.41%
3	1/4/2022 16:00:0	145.55	146.61	143.82	144.42	1146389	-0.78%
4	1/5/2022 16:00:0	144.18	144.3	137.52	137.65	2482076	-4.53%
5	1/6/2022 16:00:0	137.5	139.69	136.76	137.55	1452452	0.04%
6	1/7/2022 16:00:0	137.91	138.25	135.79	137	970412	-0.66%
7	1/10/2022 16:00	135.1	138.64	133.14	138.57	1704784	2.57%
8	1/11/2022 16:00	138.18	140.33	136.81	140.02	1175062	1.33%
9	1/12/2022 16:00	141.55	142.81	141.11	141.65	1182079	0.07%
10	1/13/2022 16:00	141.84	143.19	138.91	139.13	1328254	-1.91%
11	1/14/2022 16:00	137.5	141.2	137.5	139.79	1191296	1.67%
12	1/18/2022 16:00	136.6	137.39	135.62	136.29	1370098	-0.23%
13	1/19/2022 16:00	136.94	138.4	135.5	135.65	1039764	-0.94%
14	1/20/2022 16:00	136.51	137.91	133.14	133.51	1096528	-2.20%
15	1/21/2022 16:00	133.01	134.76	130	130.09	2095961	-2.20%
16	1/24/2022 16:00	126.03	130.78	124.64	130.37	2764602	3.44%
17	1/25/2022 16:00	128.44	129.34	126.38	126.74	1800430	-1.32%
18	1/26/2022 16:00	130.59	132.81	127.15	129.24	1981544	-1.03%
19	1/27/2022 16:00	131.36	132.61	128.95	129.12	1514251	-1.71%
20	1/28/2022 16:00	130	133.37	128.69	133.29	1525878	2.53%
21	1/31/2022 16:00	134.2	135.84	132.27	135.7	1702778	1.12%
22	2/1/2022 16:00:0	137.84	138.2	134.57	137.88	2560160	0.03%
23	2/2/2022 16:00:0	151.86	152.1	145.56	148.04	4487538	-2.52%
24	2/3/2022 16:00:0	145.29	149.12	142.21	142.65	2846507	-1.82%
25	2/4/2022 16:00:0	143.02	144.54	139.82	143.02	2461220	0.00%
26	2/7/2022 16:00:0	143.71	143.85	138.7	138.94	2230537	-3.32%
27	2/8/2022 16:00:0	138.99	139.84	136.87	139.21	1712752	0.16%
28	2/9/2022 16:00:0	140.85	142.18	140.38	141.45	1431358	0.43%
29	2/10/2022 16:00	139.5	141.43	138.05	138.6	1650885	-0.65%
30	2/11/2022 16:00	138.75	139.28	133.29	134.13	1940440	-3.33%
31	2/14/2022 16:00	133.37	136.17	133.3	135.3	1339640	1.45%
32	2/15/2022 16:00	137.47	137.9	135.54	136.43	1328878	-0.76%
33	2/16/2022 16:00	136.43	137.95	134.82	137.49	1280483	0.78%
34	2/17/2022 16:00	136.15	136.84	132.2	132.31	1548406	-2.82%
35	2/18/2022 16:00	133.04	133.82	130.31	130.47	1592930	-1.93%
36	2/22/2022 16:00	129.99	131.9	127.74	129.4	1945329	-0.45%
37	2/23/2022 16:00	131.08	131.75	127.5	127.59	1321564	-2.66%
38	2/24/2022 16:00	125	133.04	124.76	132.67	2158254	6.14%

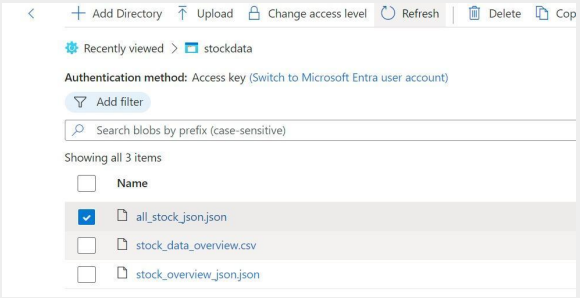
Data Source: Stock Data (Cont'd)

```python
def get_company_stock_data(ticker: str, start_date: str, end_date: str) -> str:
    """
        Ticker: str, i.e. GOOG, AMZN
        start_date: str, "2020-01-01"
        end_date:str , today

        output: file path
    """
    print(f"Getting fresh data for {ticker} at date {end_date}.")
    stock_df = yf.download(ticker, start=start_date, end=end_date)
    # Calculate pct change
    stock_df[CHANGE_PCT_NAME] = (
        stock_df[[stock_df.columns[0], stock_df.columns[3]]].pct_change(axis=1)[stock_df.columns[3]] * 100
    )
    # Groom through df
    stock_df = stock_df.drop(stock_df.columns[4], axis=1)
    stock_df['date'] = pd.to_datetime(stock_df.index).strftime('%Y-%m-%d')
    stock_df['date'] = stock_df['date'].astype(str)
    stock_df['ticker'] = ticker
    stock_df.to_csv(f"/Users/qwu194/Documents/playground/playgroud_azure/test_file/{ticker}.csv", index=False)
    # json_output = stock_df.to_json(orient='records')
    # return json_output
```

Data Source: Stock Data Refresh

- Scheduled to run at Mon-Fri at 10 pm.
- Creates and uploads a json file of all stocks from yesterday to current date.






```python
def main():
    """
        Main data refresher
    """
    current_date = date.today()
    ongoing_json = []
    _get_list_of_tickers()
    for ticker in _get_list_of_tickers():
        ticker_json = get_company_stock_data(
            ticker=ticker,
            start_date=STOCK_START_DATE,
            end_date=current_date,
        )
        ongoing_json.append(
            {ticker: ticker_json}
        )

    _write_json_to_file(STOCK_OUTPUT_LOC, ongoing_json)
    file_uploader(
        upload_file_path=STOCK_OUTPUT_LOC,
        container_name=STOCK_CONTAINER_NAME,
        blob_name=f"all_stock_json.json",
    )
```

Data Source: Cosmos_Implementer

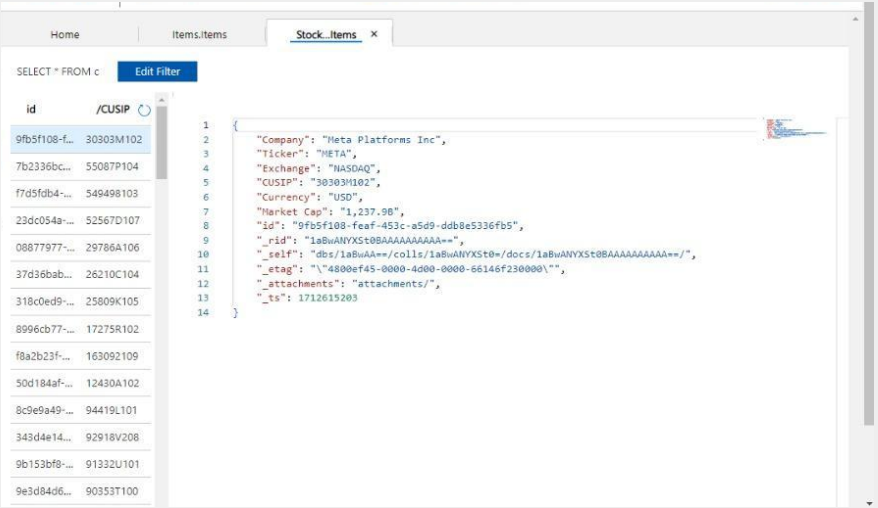
- Scheduled to run at Mon-Fri at 10:30 pm.
- The implementor will download the newest data refreshed by the stock data, and upsert into the cosmos db.

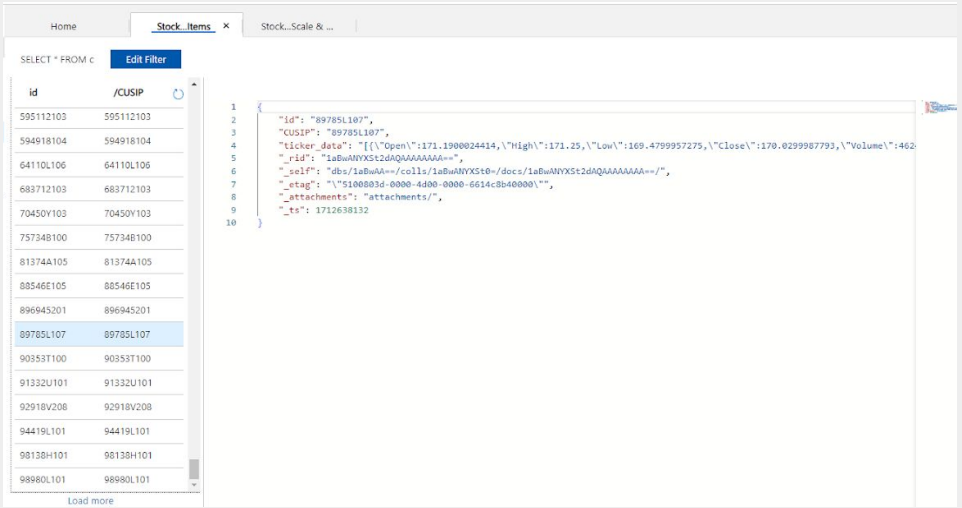
```
Document with Ticker: TWKS upserted successfully.
Document with Ticker: TRIP upserted successfully.
Document with Ticker: TRUE upserted successfully.
Document with Ticker: UBER upserted successfully.
Document with Ticker: U upserted successfully.
Document with Ticker: VRM upserted successfully.
Document with Ticker: W upserted successfully.
Document with Ticker: WDAY upserted successfully.
Document with Ticker: ZM upserted successfully.
```

```python
def file_downloader(container_name: str, blob_name: str, download_file_path: str):
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=blob_name)
    os.makedirs(os.path.dirname(download_file_path), exist_ok=True)
    with open(download_file_path, mode="wb") as download_file:
        download_file.write(blob_client.download_blob().readall())
    print(f"Complete downloading")

stock_issue_info_blob_name = 'stock_overview_json.json'
stock_data_blob_name = 'all_stock_json.json'
stock_issue_info_download_path = r'C:\Users\teyal\Downloads\stock_overview_json.json'
stock_data_download_path = r'C:\Users\teyal\Downloads\all_stock_json.json'

file_downloader('stockdata', stock_issue_info_blob_name, stock_issue_info_download_path)
file_downloader('stockdata', stock_data_blob_name, stock_data_download_path)
```

```python
def upsert_documents(documents, container, ticker_to_cusip):
    for doc in documents:
        ticker = list(doc.keys())[0]
        cusip = ticker_to_cusip.get(ticker)

        if not cusip:
            print(f"No CUSIP found for document with Ticker: {ticker}")
            continue


        transformed_doc = {
            'id': cusip,
            'CUSIP': cusip,
            'ticker_data': doc[ticker]
        }
        try:
            container.upsert_item(transformed_doc)
            print(f"Document with Ticker: {ticker} upserted successfully.")
        except exceptions.CosmosHttpResponseError as e:
            print(f"Failed to upsert document: {e}")
```

Data Source: Cosmos_Implementer (Cont'd)





Data Source: Data Refresh Scheduling

```python
def file_downloader(download_file_path: str, container_name: str, blob_name: str):
    """
        download_file_path: str, local file path
        container_name: str, container name i.e. stockdata, layoffdata
        blob_name: str, file in container name
    """
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=blob_name)

    with open(file=download_file_path, mode="wb") as download_file:
        download_file.write(blob_client.download_blob().readall())

    print(f"File at {download_file_path} has finished downloading")
```

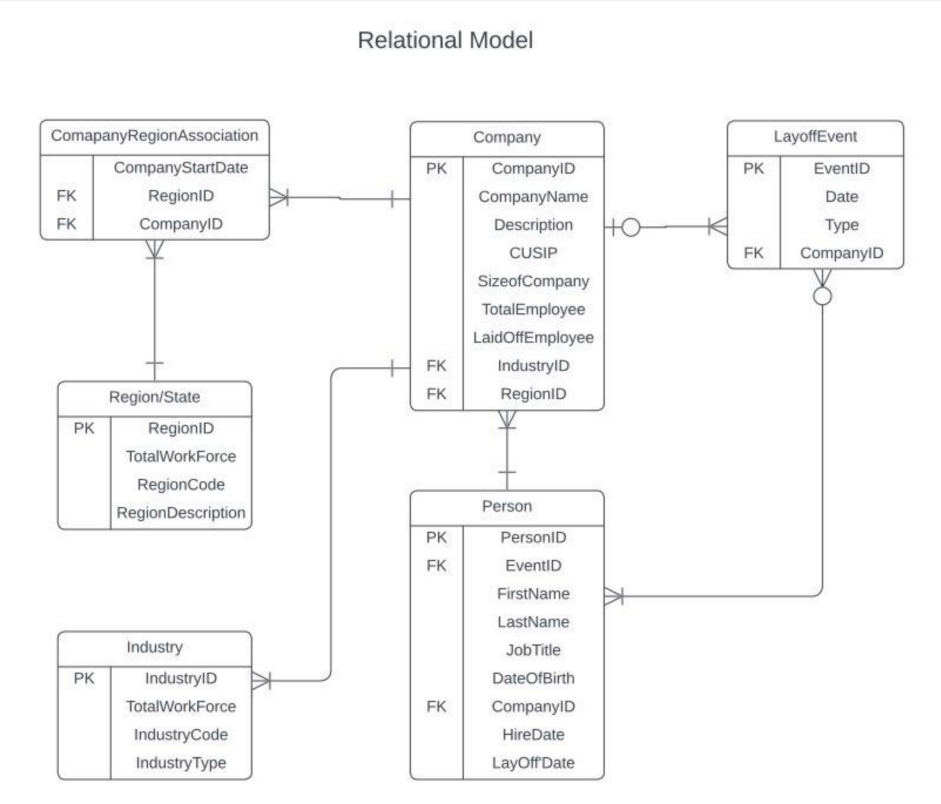
```python
def file_uploader(upload_file_path: str, container_name: str, blob_name: str):
    """
        upload_file_path: str, local file path
        container_name: str, container name i.e. stockdata, layoffdata
        blob_name: str, file to be named in container
    """
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=blob_name)

    with open(file=upload_file_path, mode="rb") as data:
        blob_client.upload_blob(data, overwrite=True)

    print(f"File at {upload_file_path} has finished uploading")
```

Data Source: Data Refresh Scheduling (Cont'd)

- We used Crontab with cron expression to schedule data refreshes every night, Monday to Friday.
- We scheduled the job by running ***crontab -e***
- Crontab:
  - A linux daemon that runs continuously in the background. It starts when the system boots, and looks for a file named .crontab in all directories. Places commands into the event list with time specifics. Examine the tasks, sleeps, and executes.
  - It is easy to set up, and efficient on system memories with no excessive overheads.

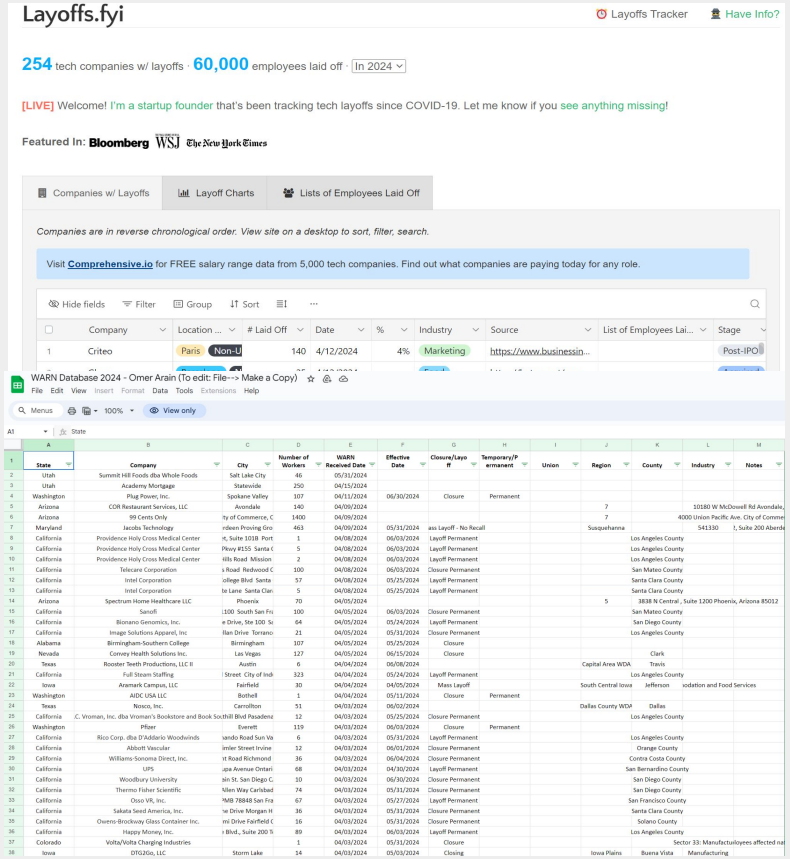
```
00 22 * * 1-5 python3 /Users/qwu194/Documents/playground/playgroud_azure/stock_data_refresher.py
30 22 * * 1-5 python3 /Users/qwu194/Documents/playground/playgroud_azure/layoff_data_refresher.py
30 22 * * 1-5 python3 /Users/qwu194/Documents/playground/playgroud_azure/cosmos_implementer.py
```

ERD: Relational



Data Source: Layoff Data

- We were able to source layoff data from two different locations:
  - Layoffs.fyi
  - WARN Database
- We used Jupyter notebooks to transform and pick out the data for the company list.
- We schedule job to fetch new data from WARN Database.



Data Source: Layoff Data (Cont'd)

```
df = pd.read_excel(r'/Users/kanimozhivelusamy/Downloads/tech_layoffs.xlsx')
df.head()

df['Date_layoffs'] = pd.to_datetime(df['Date_layoffs'])
df_filtered = df[(df['Date_layoffs'].dt.year >= 2022) & (df['Date_layoffs'].dt.year <= 2024) & (df['Country'] == 'US
#print(df_filtered)
df_filtered.to_csv('df_2022_to_2024_layoffs_USA_filtered.csv', index=False)


layoff_df = pd.read_csv(r'/Users/kanimozhivelusamy/Downloads/df_2022_to_2024_layoffs_USA_filtered.csv')
company_names_df = pd.read_excel(r'/Users/kanimozhivelusamy/Downloads/Company_2024.xlsx')
company_names_list = company_names_df['Company'].tolist()
filtered_layoff_df = layoff_df[layoff_df['Company'].isin(company_names_list)]
filtered_layoff_df.to_csv('filtered_layoff_details_1.csv', index=False)
```

```
industry_data_modified = industry_data.rename(columns={'IndustryID': 'IndustryID', 'Description': 'IndustryType', 'T
#print(industry_data_modified)
industry_combined = industry_data_modified.groupby('IndustryType', as_index=False).agg({'IndustryID': 'first', 'Tota
#print(industry_combined)
row_num = len(industry_combined)
industry_code = ['IC' + str(np.random.randint(1, 1000)).zfill(5) for _ in range(row_num)]
industry_map = dict(zip(industry_combined['IndustryType'], industry_code))
industry_combined['IndustryCode'] = industry_combined['IndustryType'].map(industry_map)
industry_combined.to_csv('industry_table.csv', index=False)
```

CompanyID	CompanyName	Description	SizeOfCompany	TotalEmployee	Location	IndustryID	RegionID
6	Enphase Energy Inc	Energy	3150	3500	San Francisco Bay Area	I434	R1993
21	Etsy Inc	Retail	1820	2045	New York City	I395	R1046
56	Unity Software Inc	Other	6360	6625	San Francisco Bay Area	I078	R1993
91	Tripadvisor Inc	Travel	3000	3125	Boston	I522	R1084
311	SecureWorks Corp	Security	1700	2000	Atlanta	I890	R1428
328	Thoughtworks Holding Inc	Other	11001	11580	Chicago	I711	R1974
504	Uber Technologies Inc	Transportation	19877	20077	San Francisco Bay Area	I424	R1993
530	TrueCar Inc	Transportation	323	425	Los Angeles	I674	R1995
540	Chegg Inc	Education	1920	2000	San Francisco Bay Area	I804	R1993

IndustryType	IndustryID	TotalWorkForce	IndustryCode
Consumer	I455	275415	IC00067
Education	I804	2000	IC00238
Energy	I434	3500	IC00698
Finance	I835	30071	IC00893
Food	I317	20833	IC00114
HR	I958	17500	IC00693
Hardware	I813	48000	IC00455
Infrastructure	I498	82000	IC00600
Media	I243	11200	IC00070
Other	I078	242497	IC00316
Real Estate	I633	2545	IC00742
Retail	I395	423945	IC00588
Security	I890	2000	IC00841
Transportation	I424	32938	IC00665
Travel	I522	3125	IC00232

Data Source: Layoff Data (Cont'd)

```
region_data = df[['RegionID', 'Location', 'TotalEmployee']]
#print(region_data)
row_num = len(region_combined)
region_code = ['RC' + str(np.random.randint(1, 100)).zfill(2) for _ in range(row_num)]
region_map = dict(zip(region_combined['RegionID'], region_code))
region_combined['RegionCode'] = region_combined['RegionID'].map(region_map)
region_combined.to_csv('region_table.csv', index=False)
```

```
company_table = pd.read_csv(r'/Users/kanimozhivelusamy/Downloads/DAMG 7275/ProjectP3/company_table.csv')
company_year = pd.read_csv(r'/Users/kanimozhivelusamy/Downloads/DAMG 7275/ProjectP3/company_year.csv')
merged_data = pd.merge(company_table, company_year, on='CompanyName')
# Print the merged data
#print(merged_data)
company_region = merged_data.rename(columns={'RegionID': 'RegionID', 'CompanyID': 'CompanyID', 'FoundedIn': 'StartYe
#print(company_region)
company_region_table = company_region[['RegionID', 'CompanyID', 'StartYear']]
company_region_table.to_csv('company_region_table.csv', index=False)
```

```
layoff_events_data = []
for company_id, layoff_date in layoff_dates.items():
    event_data = {
        "EventID": fake.unique.random_int(min=1000, max=9999),
        "Date": layoff_date,
        "Type": layoff_types[company_id],
        "CompanyID": company_id
    }
    layoff_events_data.append(event_data)

df_layoff_events = pd.DataFrame(layoff_events_data)
print(df_layoff_events.head())
csv_file_path = 'C:/Users/teyal/Downloads/layoff_table.csv'
df_layoff_events.to_csv(csv_file_path, index=False)
```

RegionID	RegionDescription	TotalWorkForce	RegionCode
R1046	New York City	5836	RC46
R1084	Boston	20525	RC01
R1415	Seattle	592000	RC58
R1428	Atlanta	2000	RC93
R1926	Boise	48000	RC93
R1974	Chicago	11580	RC33
R1993	San Francisco Bay Area	504703	RC09
R1995	Los Angeles	425	RC43

RegionID	CompanyID	StartYear
R1993	6	2006
R1046	21	2005
R1993	56	2004
R1084	91	2000
R1428	311	1999

EventID	Date	Type	CompanyID
1354	2023-12-18	Restructuring	6
8872	2023-12-13	Downsizing	21
7921	2023-11-28	Cost Reduction	56
2004	2023-11-12	Automation	91
5058	2023-08-14	Restructuring	311
5604	2023-06-21	Cost Reduction	504
5541	2023-06-14	Outsourcing	530
1283	2023-06-12	Downsizing	540

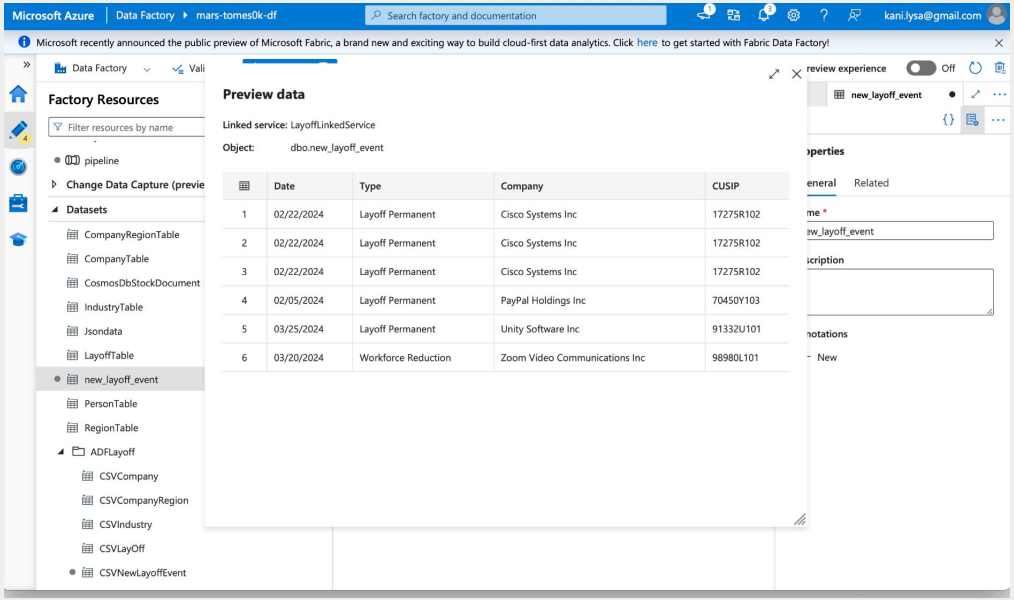
Data Source: Layoff Data Refresh

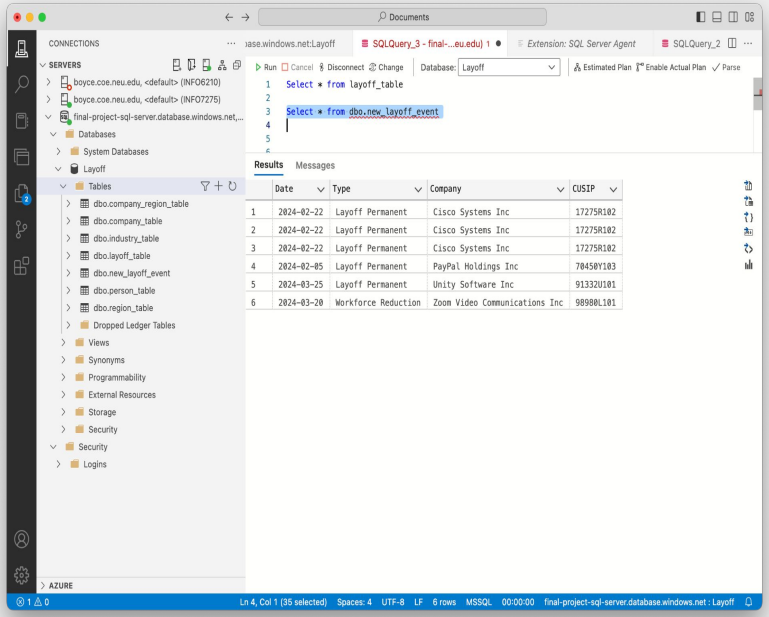
- We pull daily from Google Sheet API for the WARN database.
- We then manipulate data using pandas to filter out newest data that matches our company list.
- We generate “new_layoff_event” file and use the file_uploader function to upload to blob storage.

```python
def layoff_data_parser(current_date:str = date.today()) -> str:
    print(f"Refreshing layoff data")

    yesterday = pd.Timestamp(current_date - timedelta(days =1))

    layoff_df = pd.read_csv(NEWEST_LAYOFF_DATA)

    layoff_df = layoff_df[layoff_df["Effective Date"].notna() & layoff_df["Company"].notna()]

    layoff_df['Date'] = pd.to_datetime(layoff_df['WARN Received Date'], format='%Y-%m-%d', errors="coerce")
    layoff_df['Type'] = layoff_df['Closure/Layoff']

    layoff_df = layoff_df[layoff_df.Date > yesterday]

    new_layoffs = pd.DataFrame(columns=NEW_LAYOFF_COLUMN)

    comp_dict = get_company_names()
    for company in comp_dict.keys():
        data_parser = layoff_df[layoff_df['Company'].str.contains(company)]
        if not data_parser.empty:
            new_df = data_parser[['Company', 'Date', 'Type']].copy()
            new_df['Company'] = comp_dict[company][0]
            new_df['CUSIP'] = comp_dict[company][1]
            new_layoffs = pd.concat([new_layoffs, new_df], ignore_index=True)

    new_layoffs.to_csv(NEWEST_LAYOFF_DATA_PARSED, index=False)
```

Date	Type	Company	CUSIP
2/22/2024	Layoff Permanent	Cisco Systems Inc	17275R102
2/22/2024	Layoff Permanent	Cisco Systems Inc	17275R102
2/22/2024	Layoff Permanent	Cisco Systems Inc	17275R102
2/5/2024	Layoff Permanent	PayPal Holdings Inc	70450Y103
3/25/2024	Layoff Permanent	Unity Software Inc	91332U101
3/20/2024	Workforce Reduction	Zoom Video Communications	98980L101

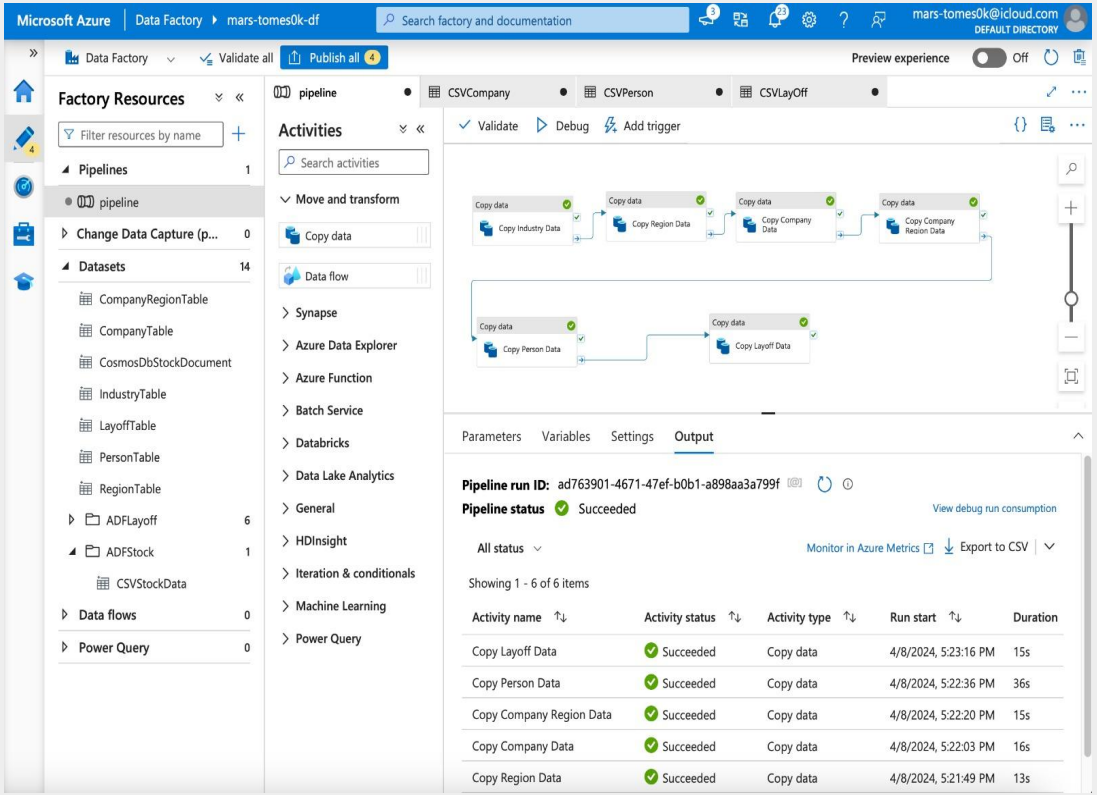
Data Source: Layoff Data Refresh (Cont'd)



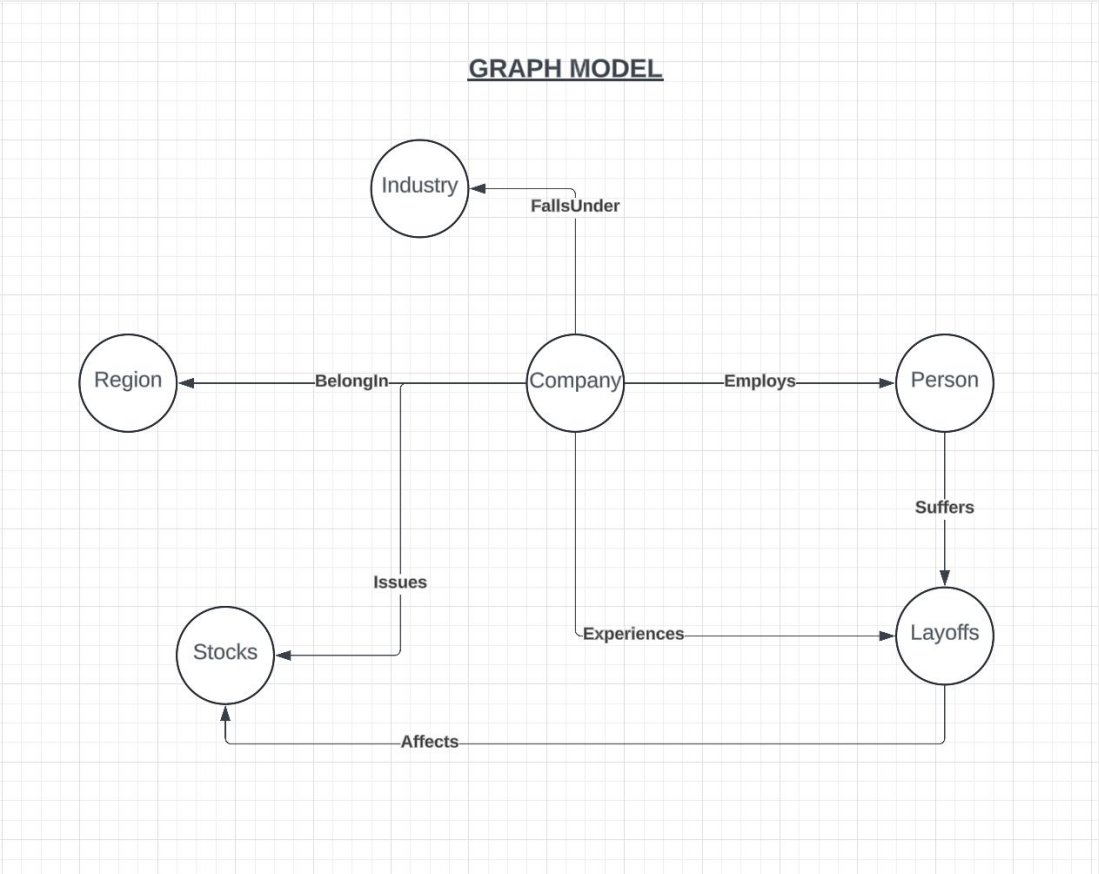


Data Ingestion

- Azure Data Factory serves as a pivotal component for data ingestion within our infrastructure.
- SQL data residing in the blob storage undergoes seamless ingestion via designated pipelines, ensuring efficient transfer to their respective databases.
- Scheduled data updates are orchestrated through the pipeline, facilitating the flow of data to designated tables based on predefined schedules.



ERD: Graph


GRAPH MODEL
Industry
FallsUnder
Region
BelongIn
Company
Employs
Person
Suffers
Issues
Stocks
Experiences
Layoffs
Affects

Graph Implementation

- Utilizing Cosmos DB Gremlin API to construct a graphical representation to corporate all the relationships.
- Data pertaining to the graph is stored in Azure Blob container named “graphdata”.
- Python employed to establish connection with Azure Blob and to transfer data to Cosmos DB.

```python
import azure.cosmos.cosmos_client as cosmos_client
from azure.storage.blob import BlobServiceClient
from gremlin_python.driver import serializer
from dotenv import load_dotenv
import os

# Loading the azure cosmos db variables from .env file
load_dotenv()

# Connection to Azure Cosmos DB
def get_cosmos_db_client():
    ENDPOINT = os.getenv('ENDPOINT')
    PRIMARY_KEY = os.getenv('PRIMARY_KEY')
    DATABASE = os.getenv('DATABASE')
    GRAPH = os.getenv('GRAPH')
    username = f"/dbs/{DATABASE}/colls/{GRAPH}"
    message_serializer = serializer.GraphSONSerializersV2d0()
    return cosmos_client.Client(ENDPOINT, username=username, password=PRIMARY_KEY, message_serializer=message_serializer)

# Connection to Azure Blob Storage
def get_blob_service_client():
    connection_string = os.getenv('CONNECTION_STRING')
    return BlobServiceClient.from_connection_string(connection_string)

# Function to read data from Azure Blob Storage
def read_blob_data(container_name, blob_name):
    blob_service_client = get_blob_service_client()
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=blob_name)
    blob_data = blob_client.download_blob()
    return blob_data.readall().decode('utf-8')
```

Vertices

```python
# Function for creating vertex
def create_vertex(client, label, properties):
    try:
        # Gremlin query
        query = f"g.addV('{label}'){''.join(f'.property(\'{prop}\', {repr(value)})' for prop, value in properties.items())}"

        # Printing the constructed query for inspection
        print("Constructed query:", query)

        # Executing the query
        callback = client.submitAsync(query)
        results = callback.result()
        for result in results:
            print(result)
    except GremlinServerError as e:
        print(f"Error executing query: {e}")

# Function for vertex worker : to process each row in the dataframe
def vertex_worker(df, g_client, v_label, v_keys):
    for row in df:
        properties = {key: row[key] for key in v_keys}
        create_vertex(g_client, v_label, properties)

# Function to insert vertices to client
def insert_vertices_to_client(df, g_client, v_label, v_keys):
    try:
        for row in df:
            properties = {key: row[key] for key in v_keys}
            create_vertex(g_client, v_label, properties)
    except Exception as e:
        print(f"Error inserting vertices: {e}")
```

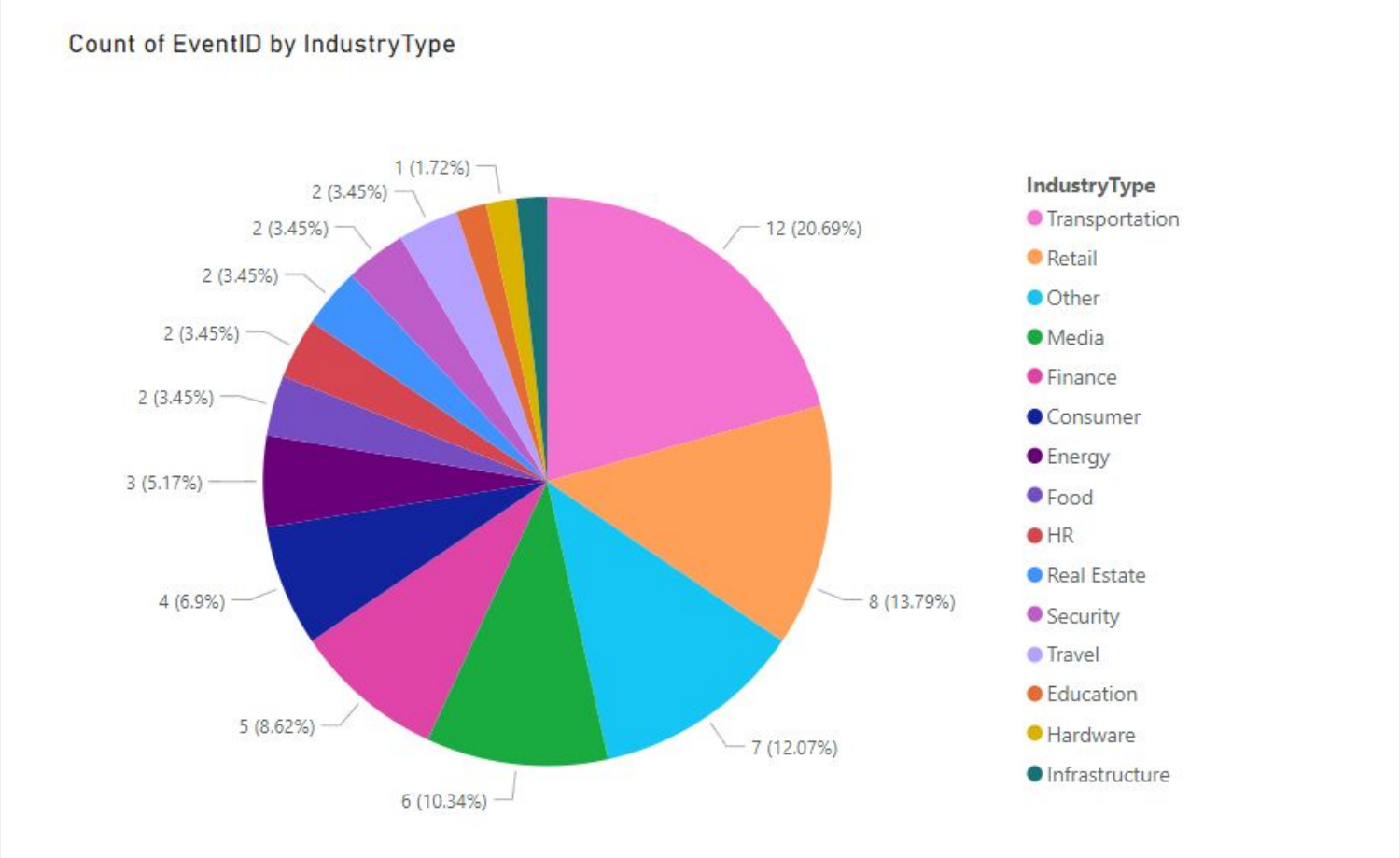
```python
# Dataframes
vertex_dfs = [
    {
        "df": company_df,
        "v_label": "company",
        "v_keys": ["company_id", "company_name", "ids"]
    },
    {
        "df": person_df,
        "v_label": "person",
        "v_keys": ["person_id", "person_name", "ids"]
    },
    {
        "df": stock_df,
        "v_label": "stock",
        "v_keys": ["CUSIP", "market_cap", "ids"]
    },
    {
        "df": layoff_df,
        "v_label": "layoff",
        "v_keys": ["layoff_id", "layoff_type", "ids"]
    },
    {
        "df": region_df,
        "v_label": "region",
        "v_keys": ["region_id", "region_name", "ids"]
    },
    {
        "df": industry_df,
        "v_label": "industry",
        "v_keys": ["industry_id", "industry_type", "ids"]
    }
]

# Inserting vertices using threading
threads = []
# Iterate over vertex_dfs and create vertices
for vertex_info in vertex_dfs:
    df = vertex_info["df"]
    v_label = vertex_info["v_label"]
    v_keys = vertex_info["v_keys"]
    insert_vertices_to_client(df, gremlin_client, v_label, v_keys)

# Waiting for all threads to complete
for t in threads:
    t.join()
```

Edges

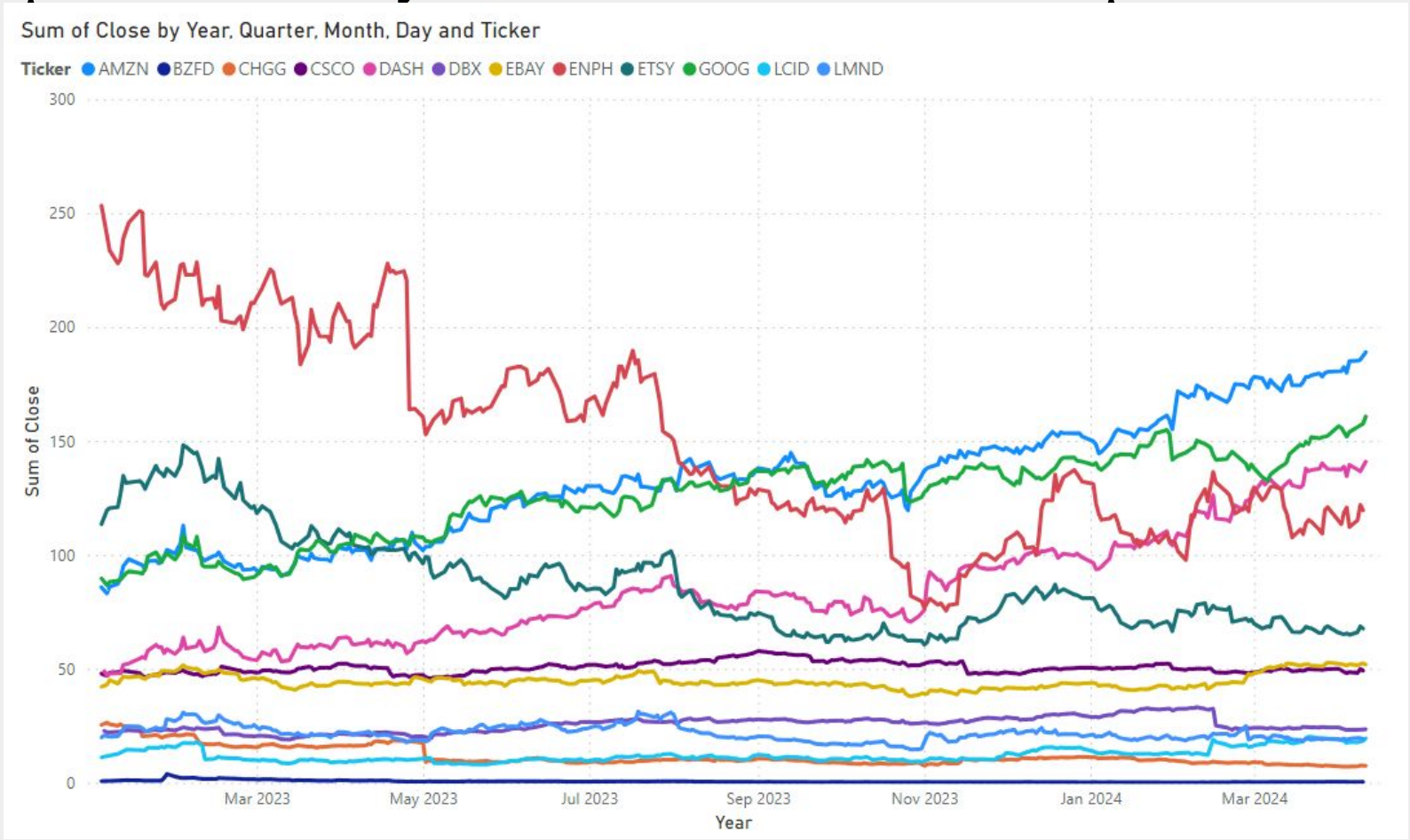
```python
# Initialize Gremlin client
gremlin_client = client.Client(
    ENDPOINT,
    'g',
    username=f'/dbs/{DATABASE}/colls/{GRAPH}',
    password=PRIMARY_KEY,
    message_serializer=serializer.GraphSONSerializersV2d0()
)

def create_gremlin_query(df, from_v_key, to_v_key, e_label):
    queries = []
    for index, row in df.iterrows():
        source = row[from_v_key]
        dest = row[to_v_key]
        query = f"g.V('{source}').addE('{e_label}').to(g.V('{dest}'))"
        print("Constructed edge query:", query)  # Print the constructed query
        queries.append(query)
    return queries


# Inside the insert_edges function
def insert_edges(client, all_edges):
    for idx, query in enumerate(all_edges):
        if idx % 800 == 0:
            time.sleep(3)
        try:
            client.submit(query)
        except protocol.GremlinServerError as e:
            print("Gremlin Server Error:", e.status_code)
        except Exception as e:
            print("An unexpected error occurred:", e)

# Inside the edge_worker function
def edge_worker(df, g_client, from_v_key, to_v_key):
    insert_edges(g_client, create_gremlin_query(df, edge_info["e_label"],from_v_key, to_v_key))
```

```
    {
        "df": FallsUnder_df,
        "e_label": "FallsUnder",
        "from_v_key": "company_id",
        "to_v_key": "industry_id"
    },
    {
        "df": Issues_df,
        "e_label": "Issues",
        "from_v_key": "company_id",
        "to_v_key": "CUSIP"
    },
    {
        "df": Suffers_df,
        "e_label": "Suffers",
        "from_v_key": "person_id",
        "to_v_key": "layoff_id"
    },
    {
        "df": Affects_df,
        "e_label": "Affects",
        "from_v_key": "layoff_id",
        "to_v_key": "CUSIP"
    },
    {
        "df": Experiences_df,
        "e_label": "Experiences",
        "from_v_key": "company_id",
        "to_v_key": "layoff_id"
    }
]

threads = []
# Inside the main section where threads are created and started
for edge_info in edge_dfs:
    t = threading.Thread(target=edge_worker, args=(edge_info["df"], gremlin_client, edge_info["from_v_key"], edge_info["to_v_key"]))
    threads.append(t)
    t.start()
```

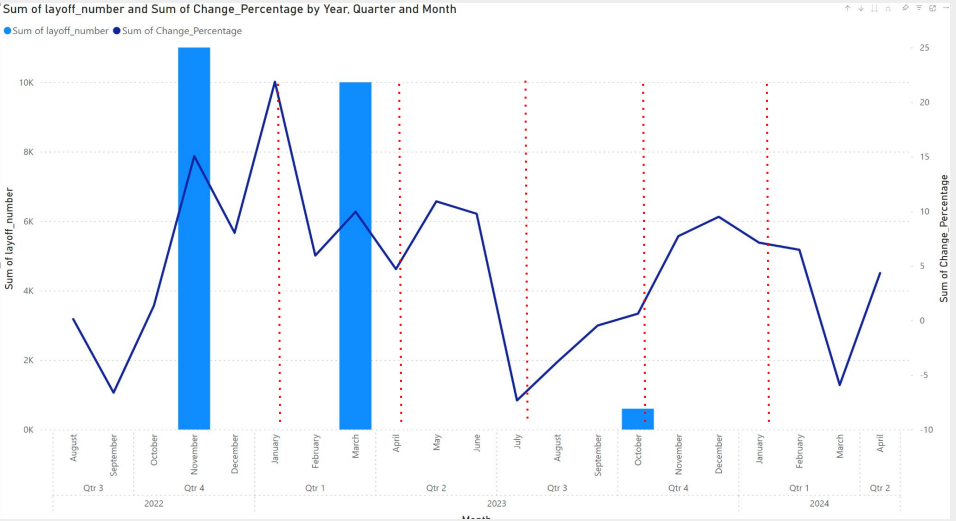
Reports and Analysis: Layoff Event Analysis(Industry Analysis)

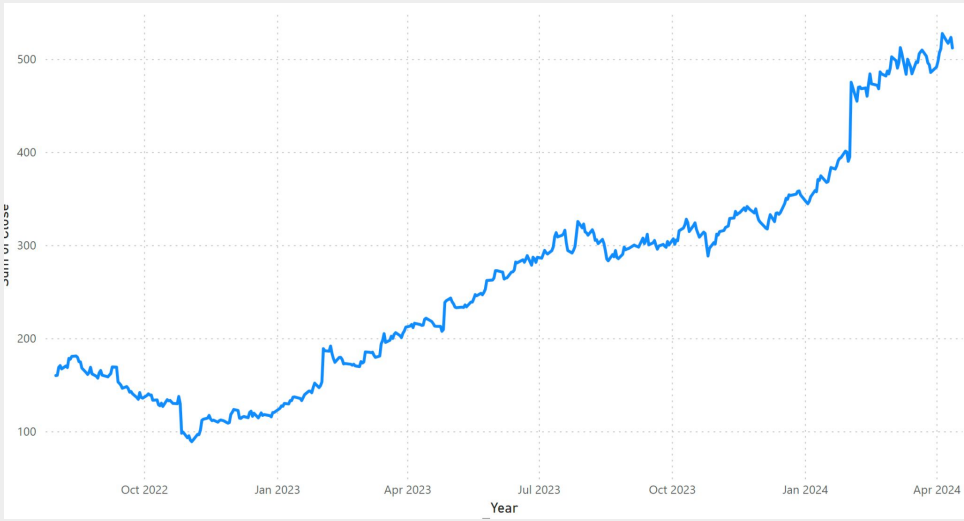


Reports and Analysis: Stock Performance Report

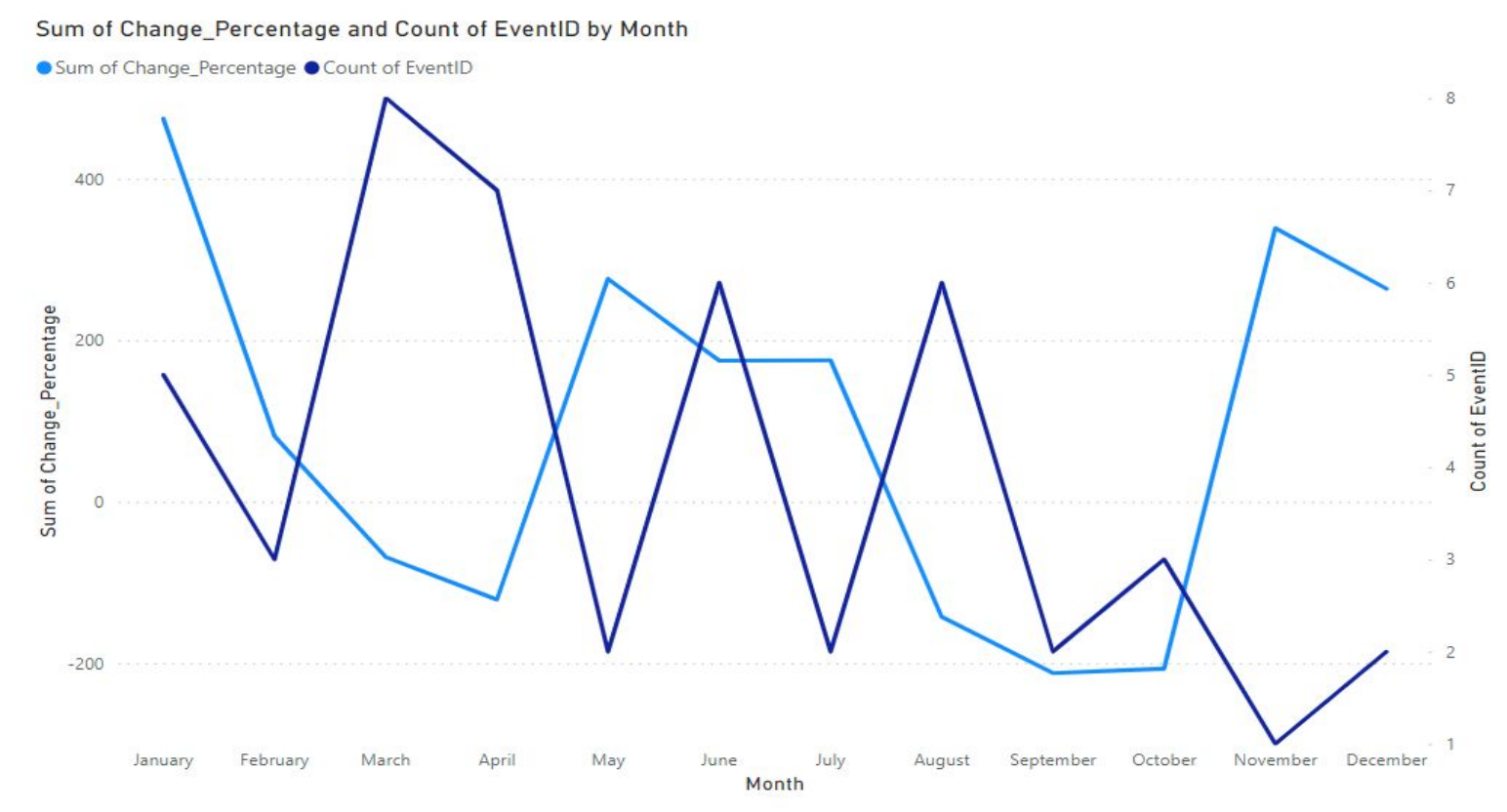


Reports and Analysis: Meta Layoff





Reports and Analysis:
How Stock Performance Impact Layoff Event



Conclusion

- In conclusion, the relationship between stock performance and layoffs is intricate and influenced by a combination of company-specific actions, industry trends, and broader economic conditions. It’s crucial for companies to approach layoffs as a strategic decision with consideration to both its immediate financial implications and its longer-term impact on the company's growth and reputation. Additionally, industry-specific insights must be taken into account when preparing for potential future layoffs, ensuring that strategies are responsive to both market and sector-specific pressures.

Thanks! It’s time to Questions?