SI 206
Final Project Report
Nick Lo, Trevor Hogland, Marina Zhai - 3 Tickets

1. Our primary goal for this project was to analyze the economic success of the U.S. by understanding the historical data of 5 top stocks (Nvidia, Apple, Amazon, Microsoft, and Meta), GDP, and Sentiment rating (insights related to articles). We wanted to see if either sentiment score or economic index would be a better indicator of the performance of stocks. We planned to work with:
    a. Alpha Vantage
    a. EconDB
    b. Stock Data

We planned to gather information such as weekly stock prices (open, high, low, close prices, and volume), sentiment score, and economic index.

2. The APIs and website we ended up using are:
    a. Alpha Vantage
    b. EconDB
    c. Polygon

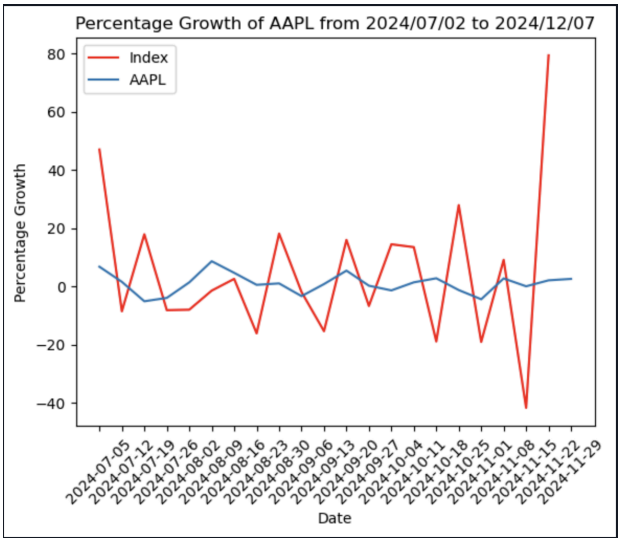
We changed the Stock Data API because there were constraints on gathering historical data, which was important for our final visualizations. We ended up gathering weekly stock prices for 5 key stocks (NVDA, APPL, AMZN, MSFT, and META), open, high, low, & close prices, and volume. We also gathered news article sentiment scores, article details (title, published date, description), and weekly economic index. We concluded that sentiment scores are a better indicator of economic performance in the U.S. as positive sentiment scores align with or precede (more closely) with spikes in stock growth, suggesting media coverage may influence market performance. Where sentiment and stock performance diverge indicate other factors also play a significant role in influencing stock prices

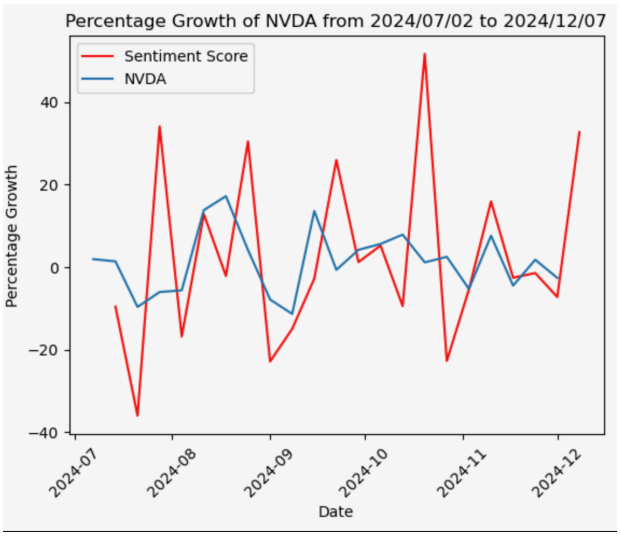
3. We faced various challenges throughout our project, particularly with extracting the correct amount of information (limiting to 25 per push, 100 in the database), gathering enough historical data, and organizing our database.

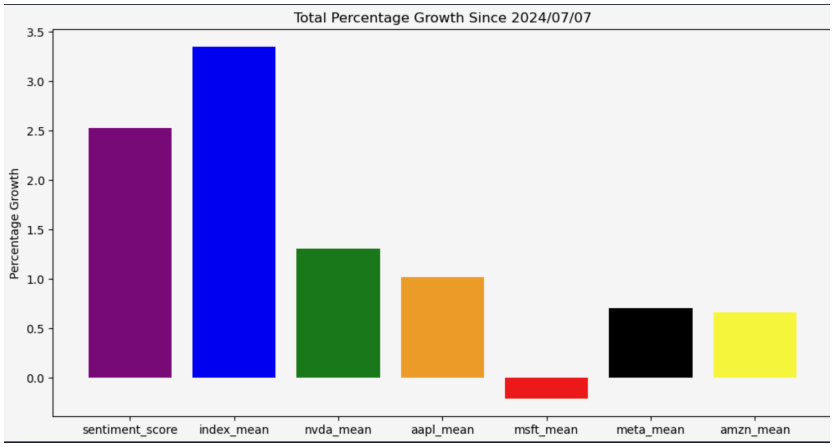
4.

	,stock_id	date	open	high	low	close	volume	index_value	ticker
0	1	2024-11-29	141.99	142.05	131.8	138.25	903463597	1.79	NVDA
1	1	2024-11-22	139.5	152.89	137.15	141.95	1396925283	3.21	NVDA
2	1	2024-11-15	148.68	149.65	140.08	141.98	1017459795	1.87	NVDA
3	1	2024-11-08	137.21	149.77	135.57	147.63	973098624	2.04	NVDA
4	1	2024-11-01	143.0	143.14	132.1106	135.4	987765940	1.65	NVDA
5	1	2024-10-25	138.13	144.42	137.46	141.54	1154273138	2.11	NVDA
6	1	2024-10-18	136.47	140.89	128.74	138.0	1357584514	1.71	NVDA
7	1	2024-10-11	124.99	135.78	124.95	134.8	1290685141	1.94	NVDA
8	1	2024-10-04	118.31	125.04	115.14	124.92	1272577548	2.22	NVDA
9	1	2024-09-27	116.55	127.665	114.86	121.4	1419480253	2.07	NVDA
10	1	2024-09-20	116.79	119.66	113.22	116.0	1466985987	2.4	NVDA
11	1	2024-09-13	104.88	120.79	103.69	119.1	1589076945	2.03	NVDA
12	1	2024-09-06	116.01	116.21	100.95	102.83	1566022704	1.99	NVDA
13	1	2024-08-30	129.57	131.26	116.71	119.37	1869975248	2.35	NVDA
14	1	2024-08-23	124.28	130.75	123.1	129.37	1575723972	1.97	NVDA
15	1	2024-08-16	106.32	125.0	106.26	124.58	1598129482	2.02	NVDA
16	1	2024-08-09	92.06	108.8	90.69	104.75	2056049067	1.99	NVDA
17	1	2024-08-02	113.69	120.16	101.37	107.27	2213649314	1.83	NVDA
18	1	2024-07-26	120.35	124.69	106.3	113.06	1513222902	1.68	NVDA
19	1	2024-07-19	130.56	131.39	116.56	117.93	1351385198	1.98	NVDA
20	1	2024-07-12	127.49	136.15	127.04	129.24	1401139984	1.81	NVDA
21	1	2024-07-05	123.47	128.85	118.83	125.83	933185147	2.66	NVDA
22	1	2024-06-28	123.24	128.12	118.04	123.54	1832912673	2.37	NVDA
23	1	2024-06-21	132.99	140.76	124.3	126.57	1756092587	2.16	NVDA
24	2	2024-11-29	231.46	237.81	229.74	237.33	198118837	1.79	AAPL
25	2	2024-11-22	225.25	230.7199	225.17	229.87	196343939	3.21	AAPL
26	2	2024-11-15	225.0	228.87	221.5	225.0	223817755	1.87	AAPL
27	2	2024-11-08	220.99	228.66	219.71	226.96	208083442	2.04	AAPL
28	2	2024-11-01	233.32	234.73	220.27	222.91	248222115	1.65	AAPL

```
joined_query = "SELECT wp.*, we.index_value, s.ticker FROM WeeklyPrices wp JOIN WeeklyEconomicIndex we ON wp.date = we.date join Stocks s on wp.stock_id =
weekly_prices_query = "SELECT * FROM WeeklyPrices"
sentiment_query = "SELECT * FROM SentimentData"
weekly_economic_index = "SELECT * FROM WeeklyEconomicIndex"
```

5. 





6.

a. Access the folder named Trevor and run trevor.py 5 times fetching all the data and inserting it into the database. Make sure to change the database name to reflect the database you wish to create or insert this data into. Do the same for nick_main.py in Nick and marina.py in Marina.
b. After running the files and getting the data into the database, check that the database contains four tables: one for stocks, one for the Weekly Economic Index, one for Weekly Stock data, and one for stock sentiment.
c. Run the code in visualizations.ipynb in the FINAL folder to see calculations and visualizations. You can modify the input to the functions to adjust the visualizations and compare the index and sentiment across our five stocks.

7.

nick_main.py in Nick

Function	Input	Output
get_date_value	url: the url string where I used BeautifulSoup to webscrape data	A list of tuples containing the date and the economic index to be used in store_data
create_database	db_name: a string to create database if not exist by calling the sqlite3 server	The connection and the cursor
store_data	cur: the cursor conn: the connection data: the data in list form from the output of get_data_value	Returns none

trevor.py in Trevor

Function	Input	Output
fetch_data_from_api	url - The API request url	The JSON response or the data from the API
create_database	db_name - The name of the database to create	Conn and cursor - the connection to the SQlite database and the object to execute future queries
insert_25_rows	cursor - cursor to execute queries conn - connection to the SQlite database stock_id - id of stock the function is inserting data - data from the API batch_size = 25 - Amount of data to insert with one run	True (data is inserted) or False (data is not inserted)
insert_remaining_rows	cursor - cursor to execute queries conn - connection to the SQlite database stock_id - id of stock the function is inserting data - data from the API	None

check_if_125_rows	cursor - cursor to execute SQL queries	True (if rows = 125) or False (any other case)

marina.py in Marina

Function	Input	Output
get_stock_id	Stock ticker symbol as a string	An integer representing the stock_id corresponding to the ticker
fetch_data_from_api	Ticker as a string- the stock ticker for which news is being fetched and the limit- (default to 25) for the maximum number of news articles to fetch	A json object containing the response data from Polygon API
create_database(db_name)	db_name as a string which is the SQlite database to create/connect to	Conn and cursor- connection and cursor object to SQlite database and executing SQlite queries
sentiment_mapping	A string representing sentiment (“positive”, “neutral”,”negative”)	An integer corresponding to sentiment (1 for positive, 0 for neutral, and -1 for negative)
insert_25_new_rows	Cursor - SQlite cursor for database operations Conn - SQlite connection object Stock_id - the id of the stock (int) Articles - (list) of dictionaries representing the articles fetched from the api	True if new rows are inserted and false if no new rows are inserted (25 already exist)
Insert_remaining_rows	Input is the same as insert_25_new_rows	None

8.

Date	Issue Description	Location of Resource	Result (Did it solve the issue?)
11/26	We are forced to switch topics as our registration for TicketMaster and SeatGeek did not go through, thus unable to	TicketMaster API and SeatGeek API	We switched topics to stocks, where there are a lot more resources that we can use

	proceed with our initial topic.		
12/1	Unable to find an index that suits our need	https://fred.stlouisfed.org/data/WEI	Finally found the data table on the website in which we are able to scrape from
12/2	We changed from stock_data api to polygon api as we were unable to get enough historical information needed for our calculations for visualizations	Stock Data Polygon	We were able to successfully correct our code to fit the needs and extract the sentiment score from polygon api
12/2	We were limited by the polygon api for how many requests we could make but we needed more data after inserting 100 rows of data	time.sleep usage	Following the documentation for how many requests we could make, we implemented time.sleep(60)