RICE UNIVERSITY

Homework 3: Quantitative Financial Analysis with Generative AI

STAT 682 - Fall 2024

Authors:
Dolese, Ryker
Fang, Judy
Hao, Sirui
Hecht, Martin
Kumar, Krish
Thetford, Jackson

Professors:
Dr. Michael D Jackson
Max Lee

Question 1 - Synthetic Data Generation for Financial Modeling

Objective: Generate synthetic financial data for asset prices to perform predictive modeling and analyze trends.

- Generate a synthetic time series for the daily closing prices of two stocks over two years using a generative model (e.g., Generative Adversarial Networks (GANs) or other generative algorithms).
- Calculate and plot the generated data's daily returns, volatility, and moving averages. Compare these values to the original data and assess differences and similarities in performance.
- Discuss how synthetic data generation could support risk modeling and decision-making for firms lacking historical data or when preparing for hypothetical scenarios.

Answer:

We run the Python code for this question. See A.1 for the full details of the code we used.

Generate synthetic financial data

To generate synthetic financial data for the daily closing prices of TSLA and NVDA over two years, we implemented a Generative Adversarial Network (GAN) using PyTorch, consisting of a Generator and a Discriminator. Here's a brief overview of the process:

- **1. Data Collection:** We fetched historical closing prices for TSLA and NVDA using the Yahoo Finance API.
- **2. GAN Architecture:** The generator takes a random noise vector (latent dimension = 128) and transforms it through several linear layers with ReLU activations to produce a synthetic price.
  - **Generator:** The generator takes a random noise vector (latent dimension = 128) and transforms it through several linear layers with ReLU activations to produce a synthetic price.
  - **Discriminator:** The discriminator evaluates the authenticity of the generated data against real data using a sequence of linear layers with ReLU and Dropout, ending in a Sigmoid layer to output a probability.
- **Training:**
  - We trained both the generator and discriminator using **Binary Cross-Entropy loss**. The discriminator learns to distinguish between real and fake data, while the generator learns to produce realistic prices by fooling the discriminator.

  - The model was trained over 5000 epochs with batch size 502, updating the generator and discriminator iteratively.

- **Synthetic Data Generation:** After training, we generated synthetic closing prices by inputting random noise into the trained generator model.

The generated synthetic prices for TSLA and NVDA are shown at the below figure:

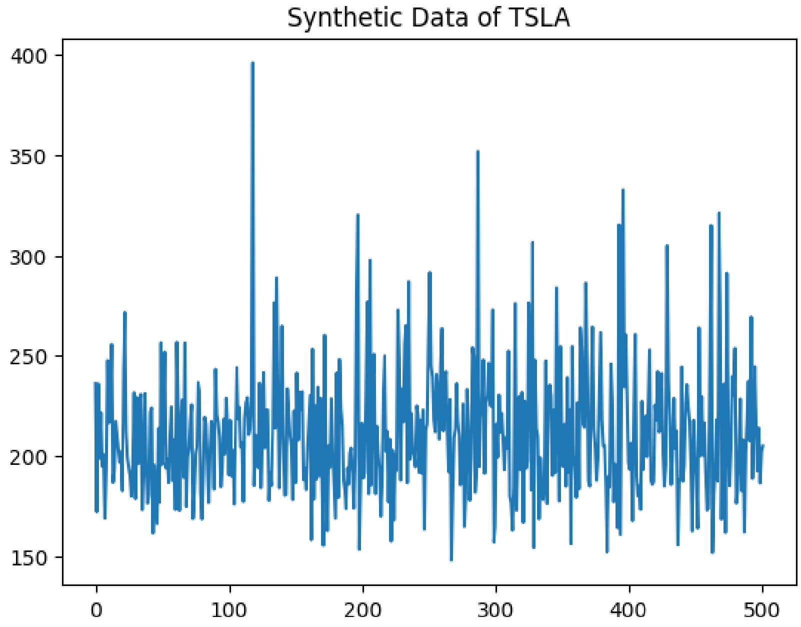


Figure 1

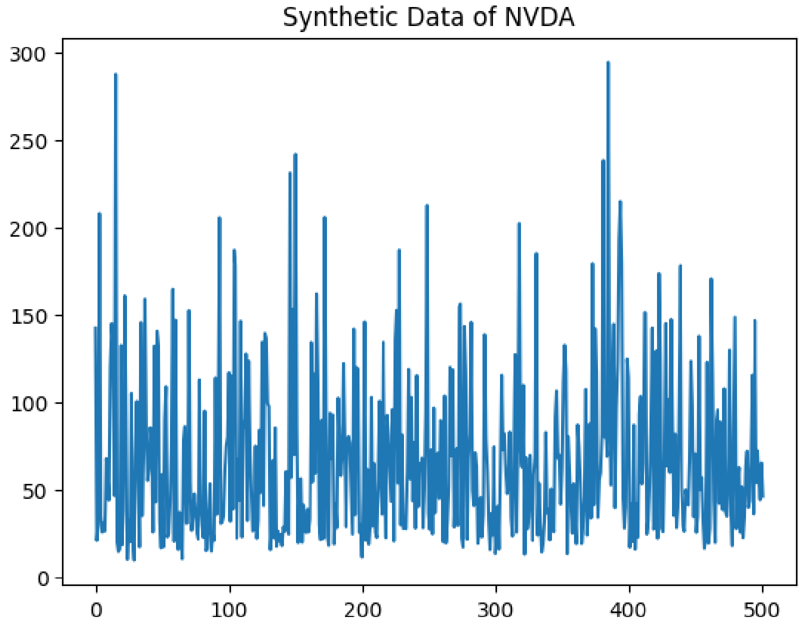


Figure 2

Comparison of original data and synthetic data

- **1. Price Trends**
  - **TSLA Price Comparison:** In Figure 3, the GAN-generated prices for TSLA generally follow the same trend as the real data but with more intense fluctuations. The synthetic prices demonstrate that the GAN was able to learn the overall trend but failed to capture the stability present in actual price movement.

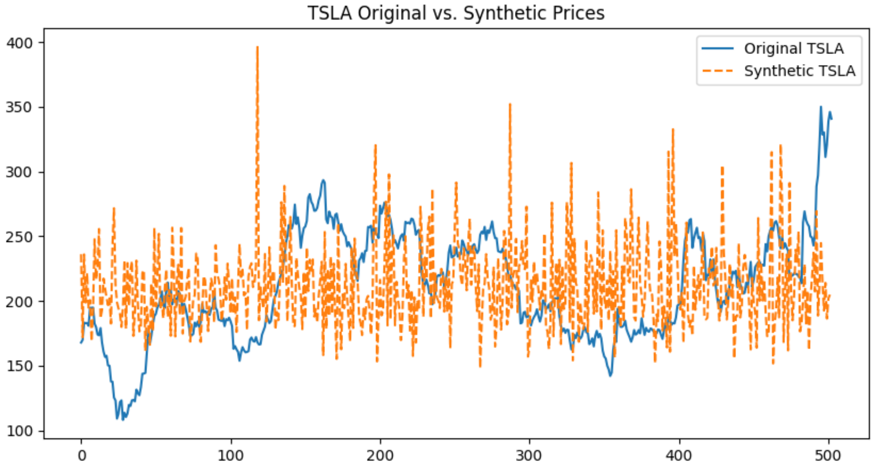


Figure 3

  - **NVDA Price Comparison:** In Figure 4, the synthetic prices for NVDA mirror the real price trend but show exaggerated daily fluctuations. This overemphasis on price movement may be due to the GAN learning general trend direction while overfitting on variability.

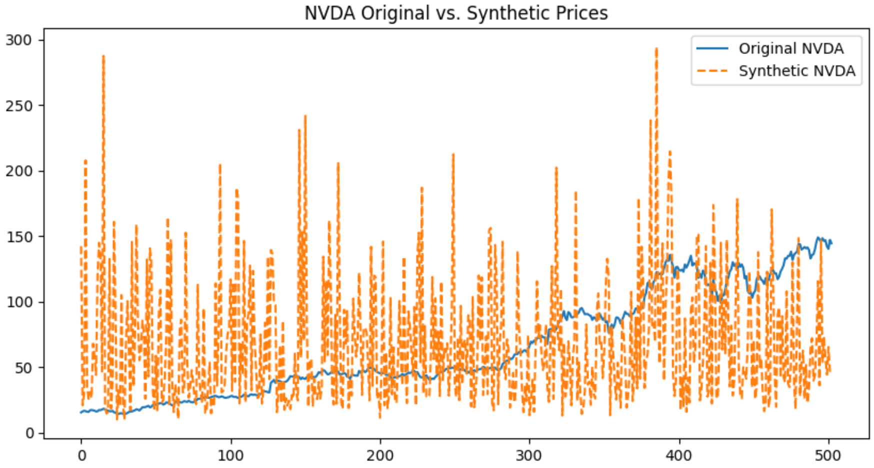


Figure 4

- **2. Daily Return**
  - **TSLA Daily Returns Comparison:** In Figure 5, the synthetic TSLA daily returns (orange, dashed line) fluctuate more sharply compared to the original data (blue line). This suggests that the GAN overestimated the variability in returns, capturing general trends but with an exaggerated amplitude in daily changes.

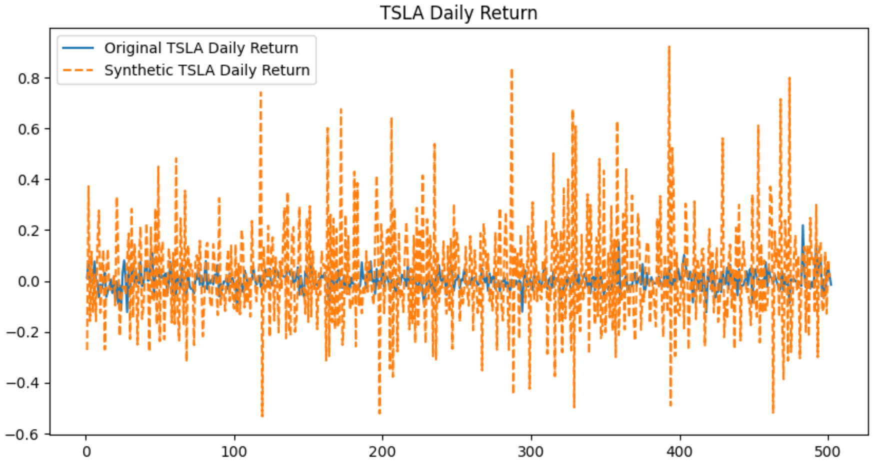


Figure 5

  - **NVDA Daily Returns Comparison:** Similarly, In Figure 6, the synthetic NVDA daily returns exhibit significant fluctuations compared to the original NVDA returns.The synthetic data's increased volatility suggests that the GAN may have generalized patterns without accurately reflecting day-to-day stability.

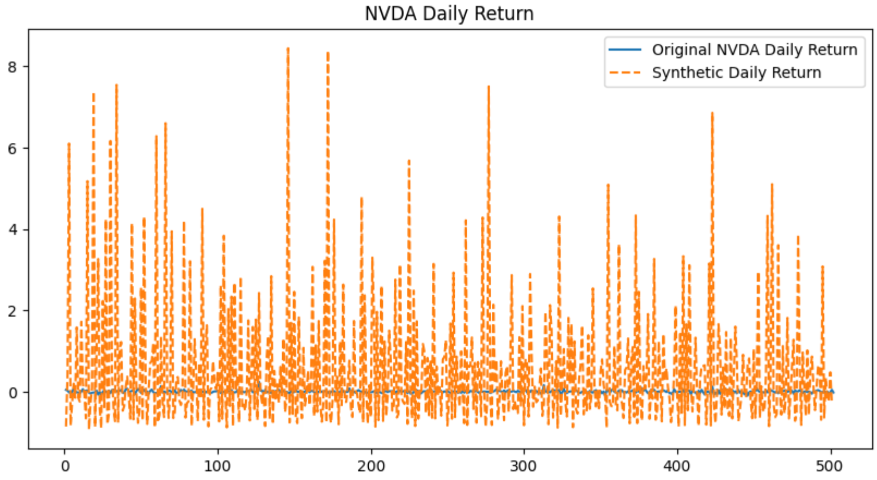


Figure 6

- **3. Moving Averages (20 days)**
  - **TSLA Moving Average (20 days) Comparison:** In Figure 7, the synthetic TSLA moving average remains generally aligned with the original moving average trend but does not capture its full amplitude. The smoother synthetic line reflects the GAN's tendency to produce a general trend without replicating finer cyclical variations.

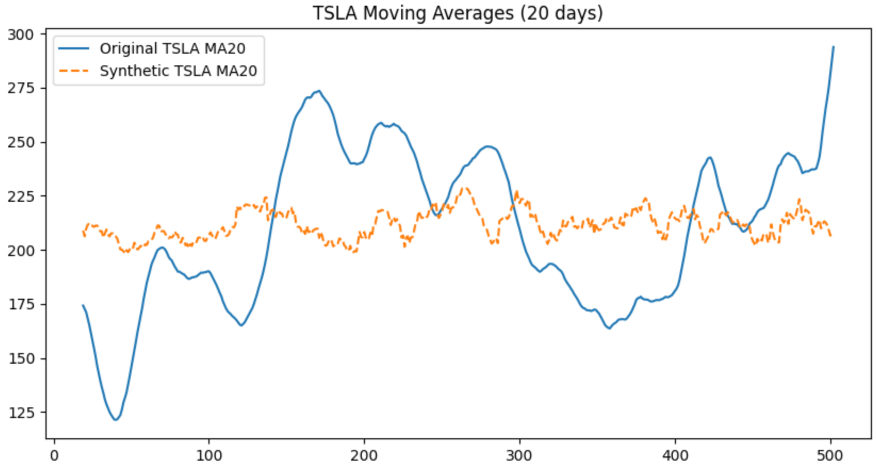


Figure 7

  - **NVDA Moving Average (20 days) Comparison:** For NVDA, in Figure 8, the synthetic moving average also approximates the overall trend but lacks the steeper upward movements observed in the original data. The synthetic line captures the broad direction but misses certain volatility features, emphasizing the GAN's limitations in modeling more complex patterns.

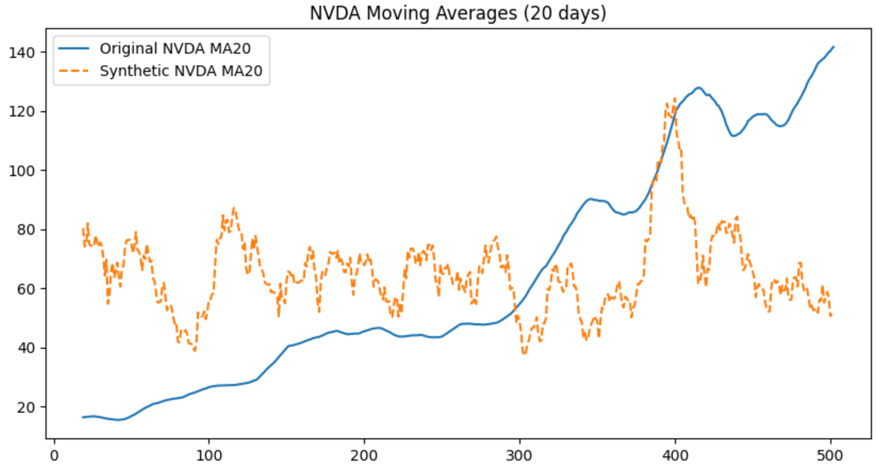


Figure 8

- **4. Volatility**
  - **TSLA Volatility Comparison:** In Figure 9, the volatility in synthetic TSLA data (orange, dashed) is consistently higher than in the original data (blue). While the original TSLA volatility gradually trends upward, the synthetic data shows frequent, sharp spikes. This discrepancy highlights the GAN's tendency to overestimate volatility while failing to replicate the smoother progression seen in real market data.

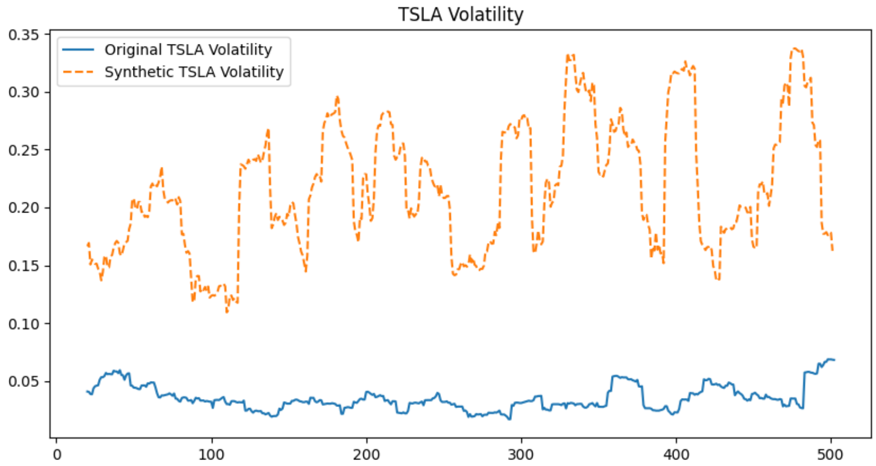


Figure 9

  - **NVDA Volatility Comparison:** For NVDA, in Figure 10, The synthetic NVDA volatility also exhibits large, frequent spikes compared to the stable trend in the original data. Similar to TSLA, the GAN-generated volatility for NVDA lacks the gradual progression seen in real data, which suggests the synthetic data might not be suitable for scenarios that require accurate volatility representation.

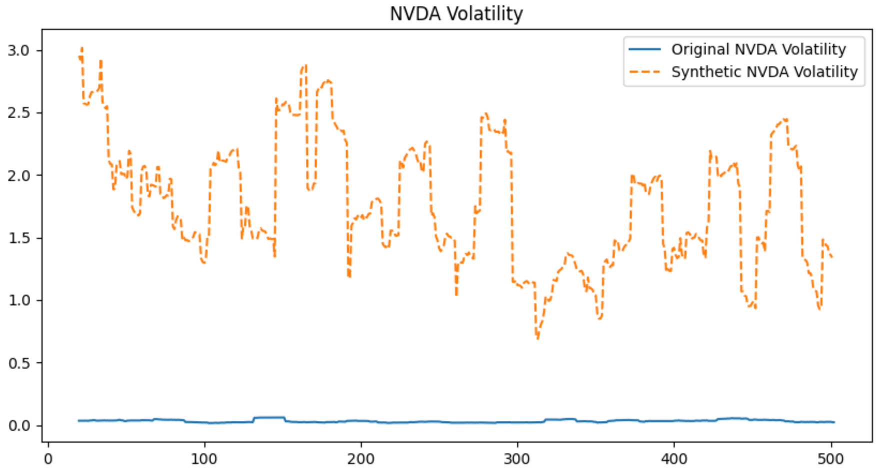


Figure 10

- **Overall Summary:** The GAN-generated synthetic data effectively approximates the general price trends for both TSLA and NVDA but struggles with accuracy in daily returns and volatility. This synthetic data could serve as a useful proxy for broader trend analysis but may not be ideal for applications needing precise day-to-day fluctuations, as it overestimates volatility and misses finer cyclical variations.

Synthetic data generation using cases

Synthetic data generation, like the GAN-produced stock prices here, can be very useful for firms in several ways:

- **1. Risk Modeling:** Firms lacking historical data can use synthetic data to model different risk scenarios. This is especially helpful for understanding how assets might behave in situations they haven't encountered before. By simulating a range of possible outcomes, businesses can better prepare for sudden market changes.
- **2. Testing Hypothetical Scenarios:** Synthetic data can be adjusted to create "what-if" situations. For example, a company could test how a stock might perform during a financial crisis or in response to a major event. This flexibility helps firms build strategies to manage future risks.
- **3. Overcoming Data Limitations:** Synthetic data allows firms to "create" data when there's limited or no access to real data. This is particularly useful for new companies or those entering a new market, as they can make more informed decisions even without years of historical data.

In summary, synthetic data generation provides a way for firms to explore potential risks and make strategic decisions in situations where they may otherwise have limited data.

Question 2 - Creating a Financial Chatbot

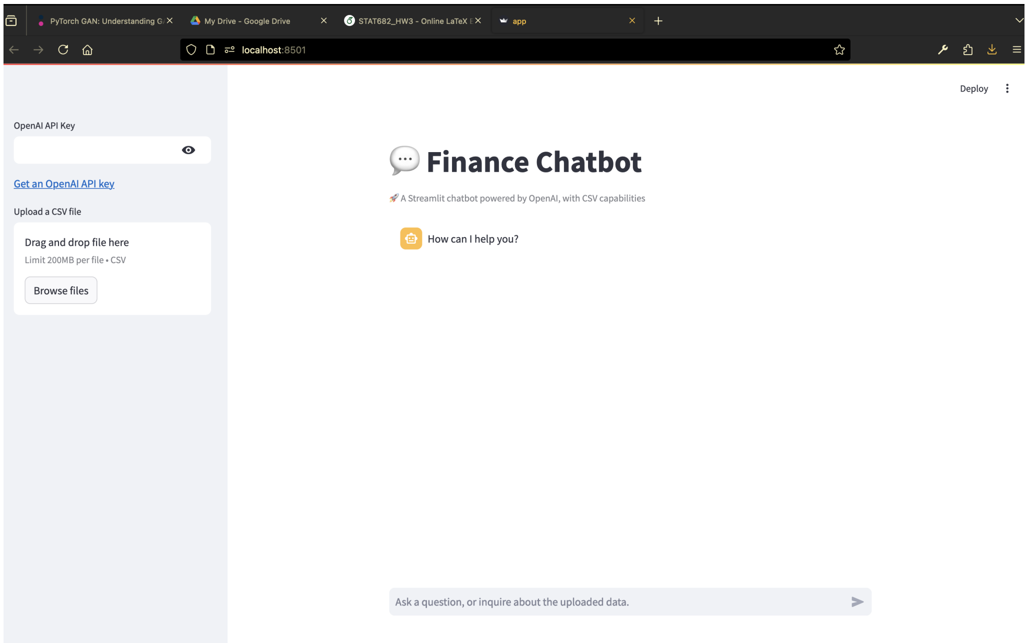
Objective: Develop a chatbot that can assist with basic financial queries and provide simple financial advice based on market data.

- Create a chatbot using a generative AI model (such as GPT) that can:
  - Answer questions about current or synthetic stock performance.
  - Explain basic financial terms (e.g., “What is volatility?”).
  - Provide simple advice or suggestions based on synthetic data generated in Question 1 (e.g., “Is now a good time to invest in stock A or B?” based on recent trends).
- Identify and discuss limitations and risks associated with relying on a generative AI-powered chatbot for financial advice, especially in live trading scenarios.

Answer:

We run the Python code for this question. See A.2 for the full details of the code we used.

- **1. Framework and Interface:**
  - The chatbot was developed using **Streamlit**, a Python-based framework for building interactive web applications. This allowed the creation of a simple and intuitive interface for users to interact with the chatbot.
  - The interface, as shown in the Figure 11, provides:
    - A field to input the OpenAI API key.
    - An option to upload CSV files for analysis.
    - A chat box for typing queries, with responses displayed dynamically.



Finance Chatbot

🚀 A Streamlit chatbot powered by OpenAI, with CSV capabilities

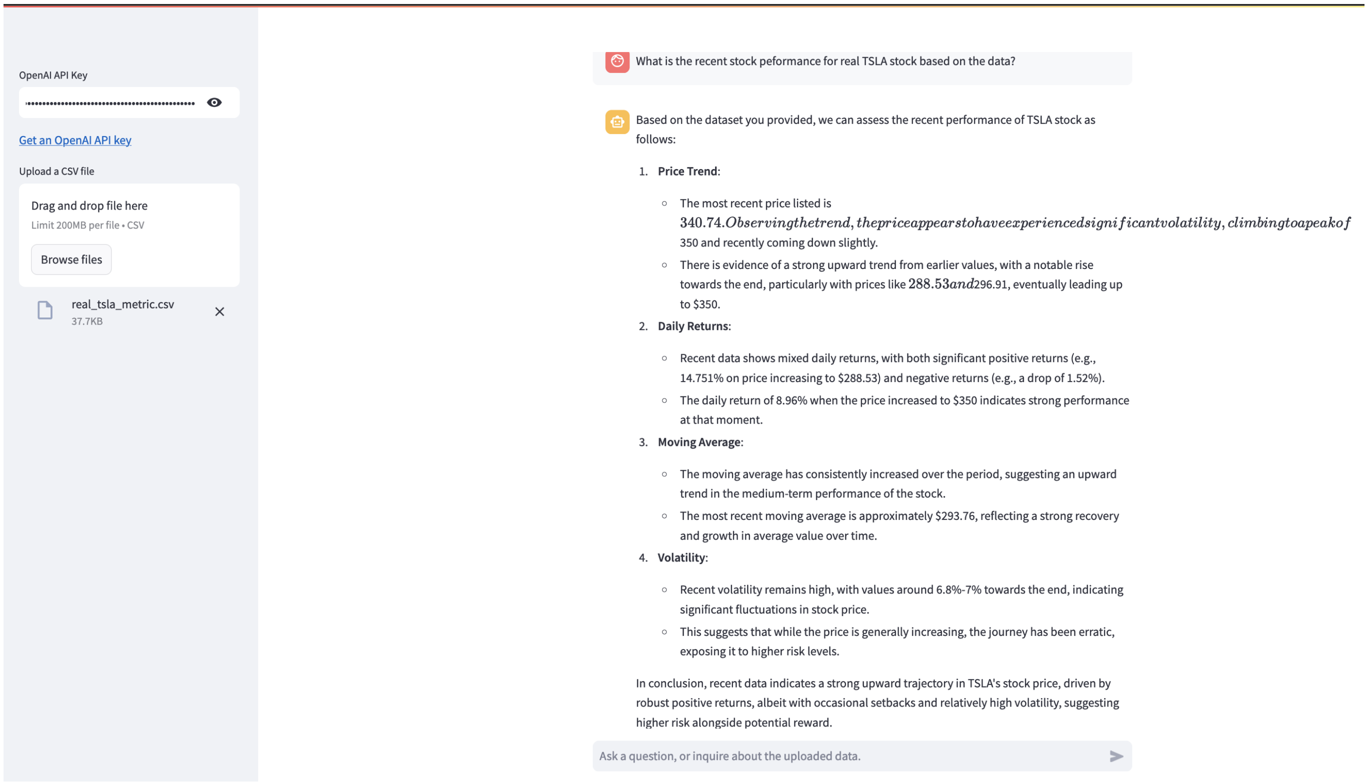
How can I help you?

Ask a question, or inquire about the uploaded data.

Figure 11

- **2. Generative AI Model Integration:** The chatbot integrates the GPT-4o model using OpenAI's API. Users can input their API keys to enable the chatbot's functionality. This provides a personalized and scalable approach to leveraging generative AI.

- **3. Key Features - Answering Questions about Stock Performance:**
    - The chatbot utilizes the GPT-4o model to answer user queries about both current and synthetic stock performance.
    - Users can upload their own CSV files containing stock data or leverage synthetic data generated earlier (**from Question 1**).
    - The chatbot analyzes trends, returns, volatility, and moving averages from the uploaded data to provide detailed, context-specific answers.
        - **What is the recent stock peformance for real TSLA stock based on the data? Response (Figure12):**



What is the recent stock peformance for real TSLA stock based on the data?

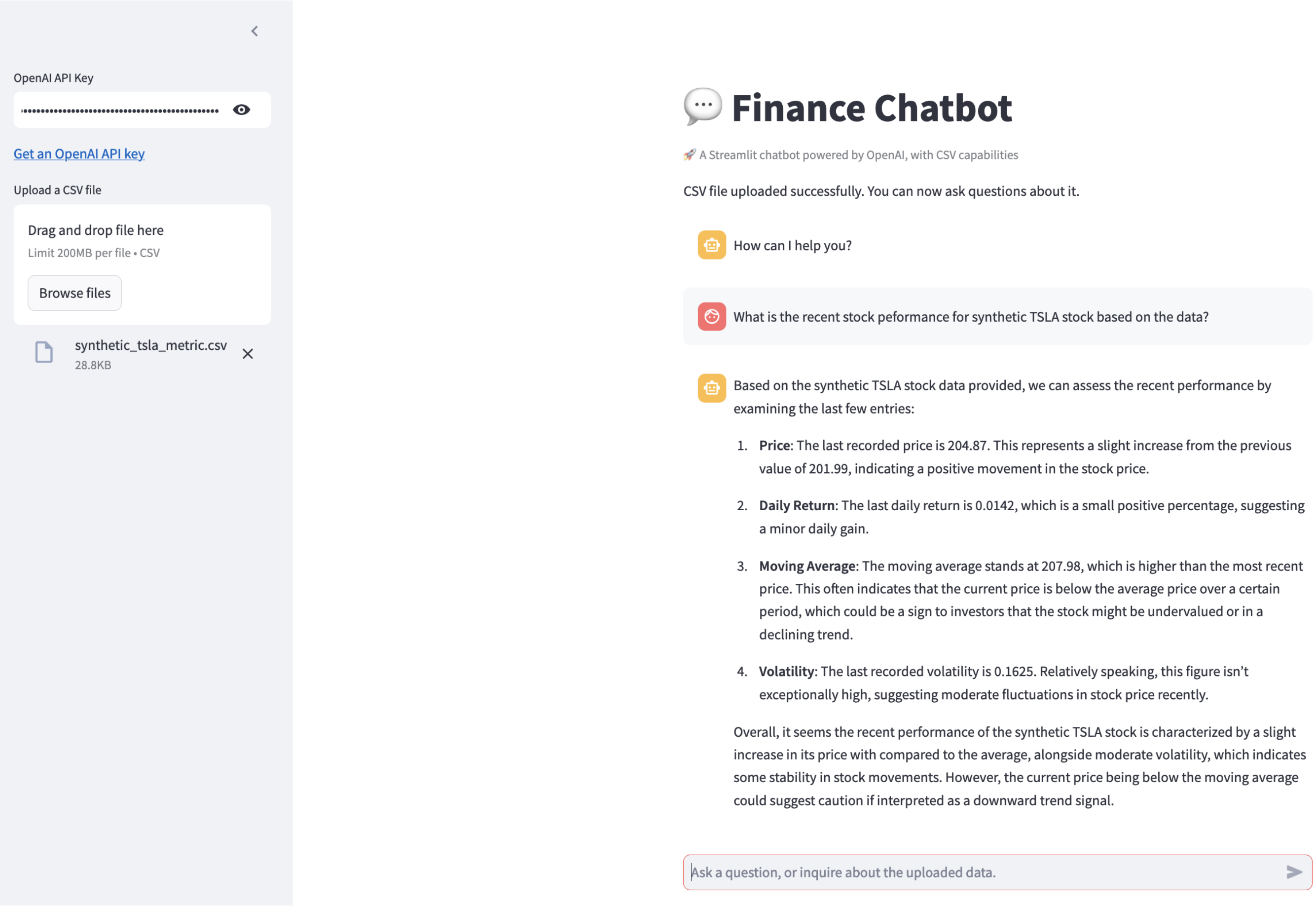
Based on the dataset you provided, we can assess the recent performance of TSLA stock as follows:

1. **Price Trend**:
    - The most recent price listed is $340.74. Observingthetrend, thepriceappearstohaveexperiencedsignificantvolatility, climbingtoapeakof$ 350 and recently coming down slightly.
    - There is evidence of a strong upward trend from earlier values, with a notable rise towards the end, particularly with prices like $288.53and$296.91, eventually leading up to $350.
2. **Daily Returns**:
    - Recent data shows mixed daily returns, with both significant positive returns (e.g., 14.751% on price increasing to $288.53) and negative returns (e.g., a drop of 1.52%).
    - The daily return of 8.96% when the price increased to $350 indicates strong performance at that moment.
3. **Moving Average**:
    - The moving average has consistently increased over the period, suggesting an upward trend in the medium-term performance of the stock.
    - The most recent moving average is approximately $293.76, reflecting a strong recovery and growth in average value over time.
4. **Volatility**:
    - Recent volatility remains high, with values around 6.8%-7% towards the end, indicating significant fluctuations in stock price.
    - This suggests that while the price is generally increasing, the journey has been erratic, exposing it to higher risk levels.

In conclusion, recent data indicates a strong upward trajectory in TSLA's stock price, driven by robust positive returns, albeit with occasional setbacks and relatively high volatility, suggesting higher risk alongside potential reward.

Figure 12

- **What is the recent stock peformance for synthetic TSLA stock based on the data? Response (Figure13):**



💬 Finance Chatbot

🚀 A Streamlit chatbot powered by OpenAI, with CSV capabilities

CSV file uploaded successfully. You can now ask questions about it.

How can I help you?

What is the recent stock peformance for synthetic TSLA stock based on the data?

Based on the synthetic TSLA stock data provided, we can assess the recent performance by examining the last few entries:

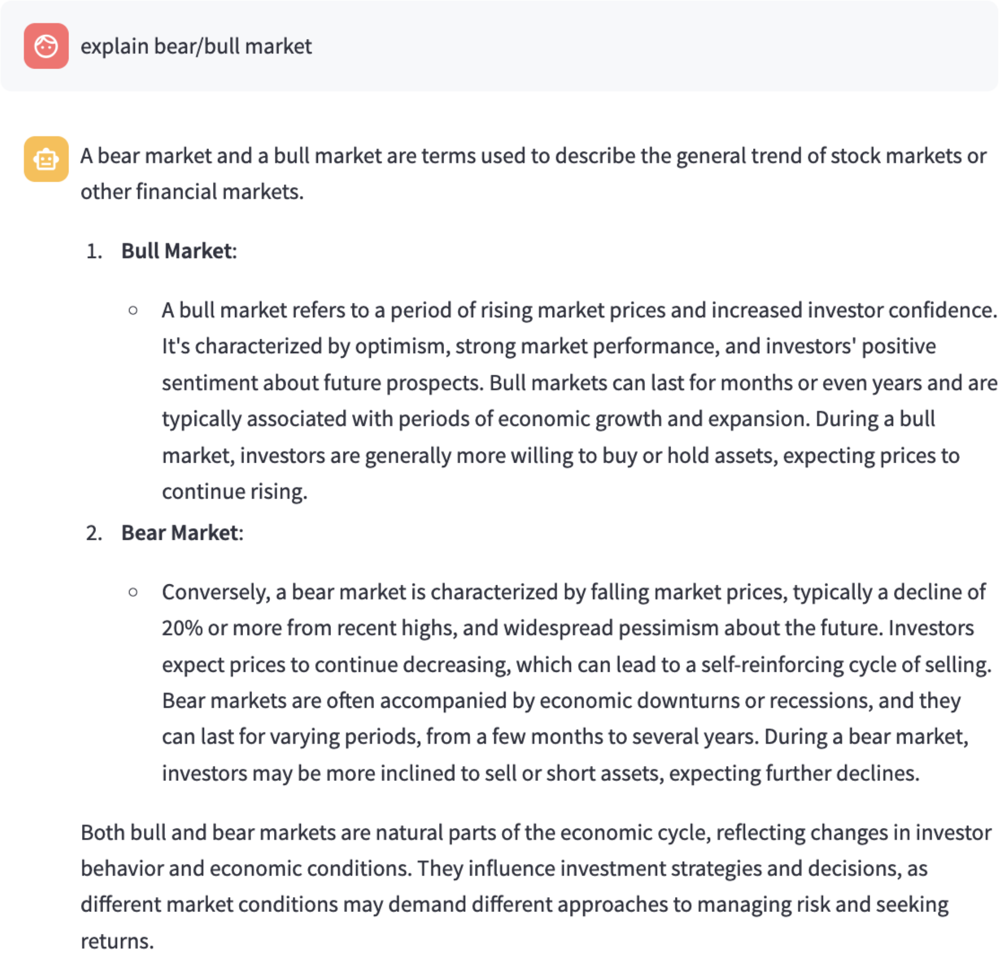
1. **Price**: The last recorded price is 204.87. This represents a slight increase from the previous value of 201.99, indicating a positive movement in the stock price.
2. **Daily Return**: The last daily return is 0.0142, which is a small positive percentage, suggesting a minor daily gain.
3. **Moving Average**: The moving average stands at 207.98, which is higher than the most recent price. This often indicates that the current price is below the average price over a certain period, which could be a sign to investors that the stock might be undervalued or in a declining trend.
4. **Volatility**: The last recorded volatility is 0.1625. Relatively speaking, this figure isn't exceptionally high, suggesting moderate fluctuations in stock price recently.

Overall, it seems the recent performance of the synthetic TSLA stock is characterized by a slight increase in its price with compared to the average, alongside moderate volatility, which indicates some stability in stock movements. However, the current price being below the moving average could suggest caution if interpreted as a downward trend signal.

Ask a question, or inquire about the uploaded data.

Figure 13

- **4. Key Features - Explaining Basic Financial Terms:**
  The chatbot is equipped to explain financial concepts in a simple, clear manner, users can ask questions such as:
  - **Explain a bear/bull market**
    Response (Figure 14):



explain bear/bull market

A bear market and a bull market are terms used to describe the general trend of stock markets or other financial markets.

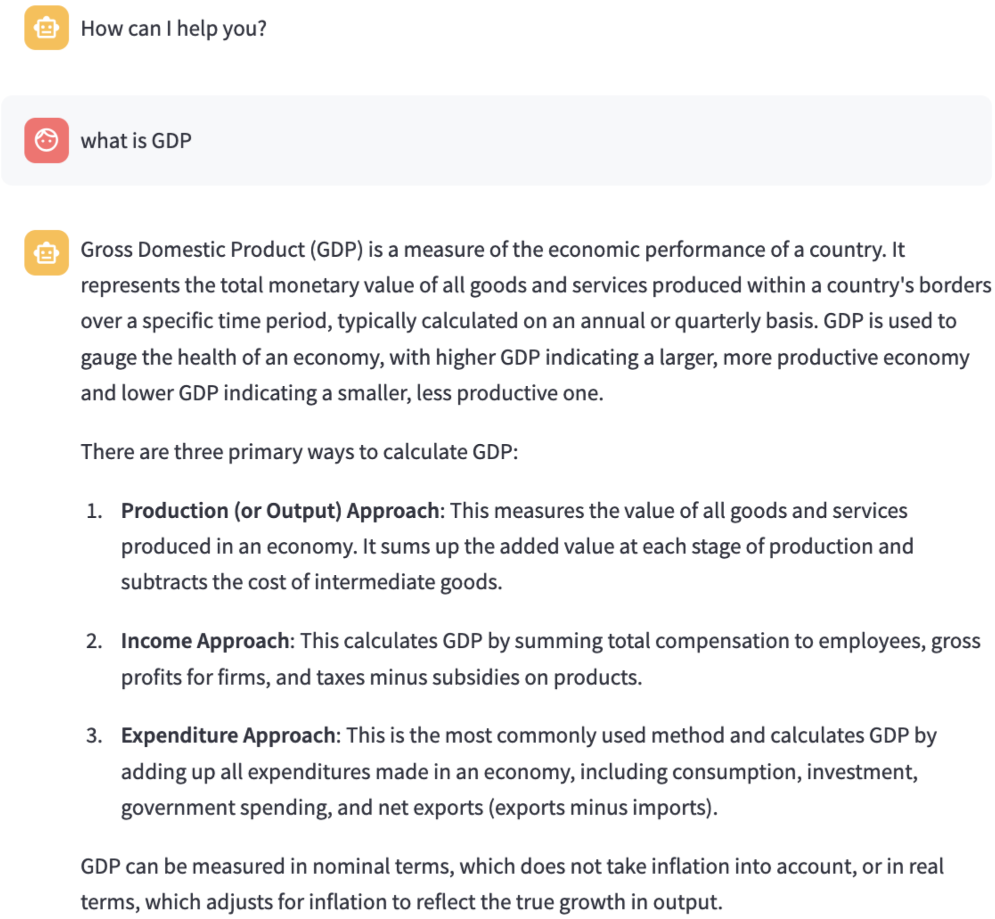
1. **Bull Market**:
   - A bull market refers to a period of rising market prices and increased investor confidence. It's characterized by optimism, strong market performance, and investors' positive sentiment about future prospects. Bull markets can last for months or even years and are typically associated with periods of economic growth and expansion. During a bull market, investors are generally more willing to buy or hold assets, expecting prices to continue rising.
2. **Bear Market**:
   - Conversely, a bear market is characterized by falling market prices, typically a decline of 20% or more from recent highs, and widespread pessimism about the future. Investors expect prices to continue decreasing, which can lead to a self-reinforcing cycle of selling. Bear markets are often accompanied by economic downturns or recessions, and they can last for varying periods, from a few months to several years. During a bear market, investors may be more inclined to sell or short assets, expecting further declines.

Both bull and bear markets are natural parts of the economic cycle, reflecting changes in investor behavior and economic conditions. They influence investment strategies and decisions, as different market conditions may demand different approaches to managing risk and seeking returns.

Figure 14

- **"What is GDP"**

  Response (Figure 15):



How can I help you?

what is GDP

Gross Domestic Product (GDP) is a measure of the economic performance of a country. It represents the total monetary value of all goods and services produced within a country's borders over a specific time period, typically calculated on an annual or quarterly basis. GDP is used to gauge the health of an economy, with higher GDP indicating a larger, more productive economy and lower GDP indicating a smaller, less productive one.

There are three primary ways to calculate GDP:

1. **Production (or Output) Approach**: This measures the value of all goods and services produced in an economy. It sums up the added value at each stage of production and subtracts the cost of intermediate goods.
2. **Income Approach**: This calculates GDP by summing total compensation to employees, gross profits for firms, and taxes minus subsidies on products.
3. **Expenditure Approach**: This is the most commonly used method and calculates GDP by adding up all expenditures made in an economy, including consumption, investment, government spending, and net exports (exports minus imports).

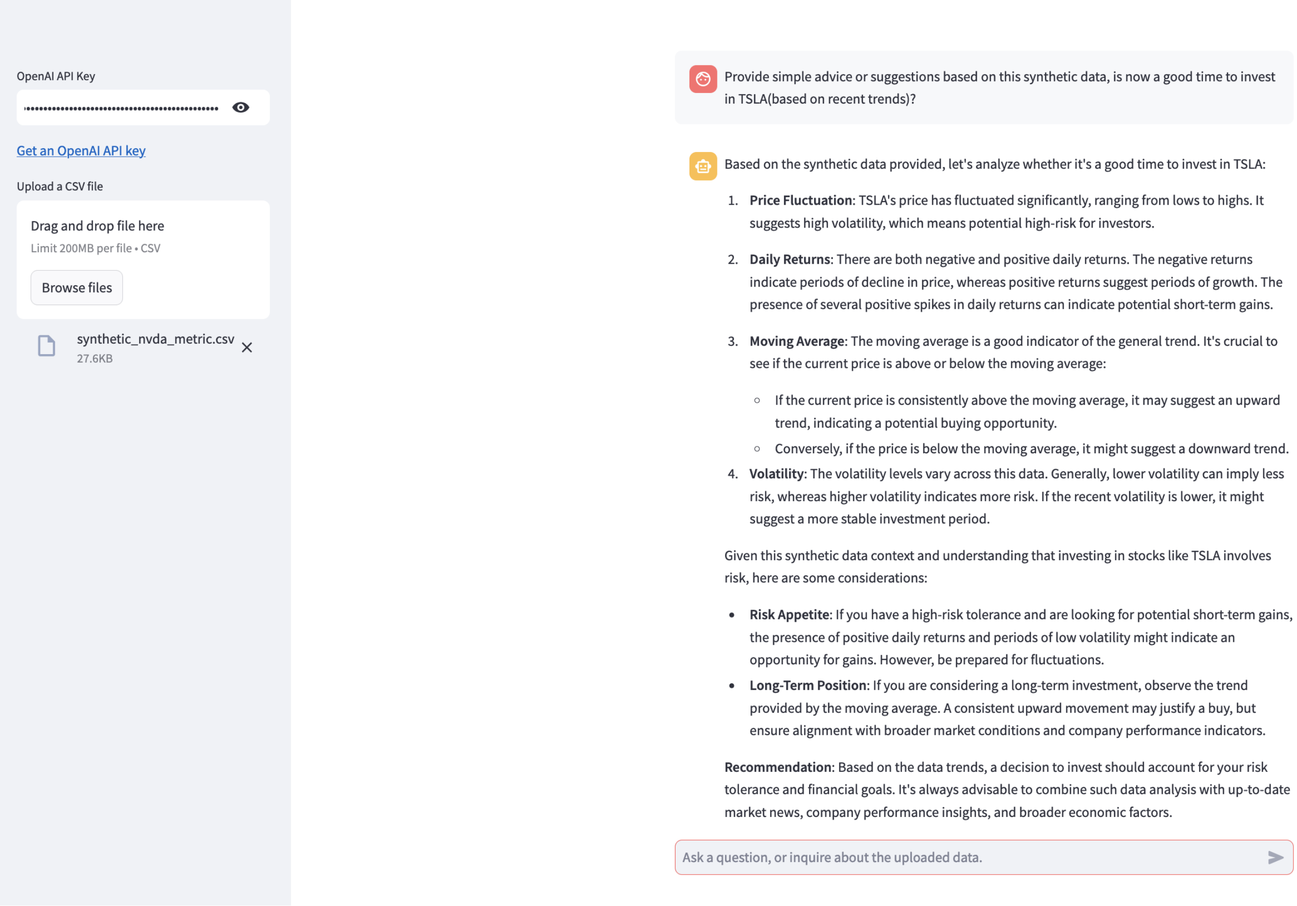
GDP can be measured in nominal terms, which does not take inflation into account, or in real terms, which adjusts for inflation to reflect the true growth in output.

Figure 15

- **5. Key Features - Provide Suggestions based on synthetic data:**
  Similar to **Key Features - Answering Questions about Stock Performance**, users can upload their own CSV files containing stock data or leverage synthetic data generated in **Question 1**.

  - **Provide simple advice or suggestions based on this synthetic data, is now a good time to invest in TSLA(based on recent trends)?**

    Response (Figure ??)


OpenAI API Key
Get an OpenAI API key
Upload a CSV file
Drag and drop file here
Limit 200MB per file • CSV
Browse files
synthetic_nvda_metric.csv
27.6KB
Ask a question, or inquire about the uploaded data.

Provide simple advice or suggestions based on this synthetic data, is now a good time to invest in TSLA(based on recent trends)?

Based on the synthetic data provided, let's analyze whether it's a good time to invest in TSLA:

1. **Price Fluctuation**: TSLA's price has fluctuated significantly, ranging from lows to highs. It suggests high volatility, which means potential high-risk for investors.
2. **Daily Returns**: There are both negative and positive daily returns. The negative returns indicate periods of decline in price, whereas positive returns suggest periods of growth. The presence of several positive spikes in daily returns can indicate potential short-term gains.
3. **Moving Average**: The moving average is a good indicator of the general trend. It's crucial to see if the current price is above or below the moving average:
   - If the current price is consistently above the moving average, it may suggest an upward trend, indicating a potential buying opportunity.
   - Conversely, if the price is below the moving average, it might suggest a downward trend.
4. **Volatility**: The volatility levels vary across this data. Generally, lower volatility can imply less risk, whereas higher volatility indicates more risk. If the recent volatility is lower, it might suggest a more stable investment period.

Given this synthetic data context and understanding that investing in stocks like TSLA involves risk, here are some considerations:

- **Risk Appetite**: If you have a high-risk tolerance and are looking for potential short-term gains, the presence of positive daily returns and periods of low volatility might indicate an opportunity for gains. However, be prepared for fluctuations.
- **Long-Term Position**: If you are considering a long-term investment, observe the trend provided by the moving average. A consistent upward movement may justify a buy, but ensure alignment with broader market conditions and company performance indicators.

Recommendation: Based on the data trends, a decision to invest should account for your risk tolerance and financial goals. It's always advisable to combine such data analysis with up-to-date market news, company performance insights, and broader economic factors.

Figure 16

- **6. Limitations and risks of using a generative AI-powered chatbot for financial advice in live trading scenarios**
  - **Overgeneralization:** Generative models like GPT can provide advice based on patterns but lack the ability to fully understand complex market dynamics, news events, or unexpected shifts.
  - **Lack of Real-Time Data**: The data might not be up-to-date. In live trading, outdated information can lead to poor decisions.
  - **Sensitivity to Training Data:** If the model is trained on biased or incomplete data, it might produce misleading responses or overlook key market nuances.
  - **No Legal or Expert Guarantee:** The chatbot is not a licensed financial advisor and can't provide personalized or legal advice, increasing the risk of following unsuitable suggestions.
  - **High Risk in Volatile Markets:** In fast-changing markets, advice based on historical or synthetic trends may fail to account for sudden movements, leading to potential financial losses.
  - **Emotional Factors:** The chatbot cannot factor in emotional aspects of investing, such as fear or greed, which play a big role in live trading decisions.

Question 3 - Sentiment Analysis on Financial News Articles

Objective: Use a generative AI model to perform sentiment analysis on financial news articles and understand its impact on stock movements.

- Collect or generate a set of synthetic financial news articles about stocks A and B. Include both positive and negative sentiment to simulate a realistic news cycle.
- Use a generative AI model or pre-trained transformer (e.g., OpenAI's GPT, BERT) to perform sentiment analysis on the articles. Categorize each article as positive, negative, or neutral.
- Calculate the overall sentiment score and analyze its correlation with both the original data and synthetic stock price movements from Question 1.
- Describe any observed relationships and discuss the reliability of using sentiment analysis in trading strategies.

Answer:

We run the Python code for this question. See A.3 for the full details of the code we used.

To answer this question we explored and generated articles related to the assets analyzed in question 1: TSLA and NVDA. In our analysis, we brought real articles and generated two synthetic articles with ChatGPT. Some of the variables we considered when creating our sets were the perceived sentiment and the length of the articles. For a more robust study, we brought different articles from different dates and authors. The choice of the sentiment for the articles were completely arbitrary, as our goal was to create similar environments which allow for relevant and respective comparisons, without selecting any specific sentiment for each article. Finally, based on these data we generated articles using ChatGPT, which can be seen in the appendix with the code. We made sure to feed each prompt with different articles and data to model different and new opinions, and mimic a symbolic 'date' for each article.

Our approach consists of splitting the articles into chunks of 256 characters and analyze the sentiment chunk by chunk. The analysis was conducted by two Natural Language Processing Models trained to understand sentiment. The first one, "mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis", is a financially literate model, meaning it has been trained specifically to understand financial terms. The second one, "finiteautomata/bertweet-base-sentiment-analysis", is a more general NLP Model that will outperform the former overall, but is not specifically trained to understand finances. After running test in both models throught the Exploratory Data Analysis process, we discovered two things that guided our procedures. Firstly, the non-financial model took articles of up to 12 chunks (so roughly 3000 characters). Secondly, the non-financial article tended to characterize as neutral the chunks containing numbers or financial terms, deeming the model inaccurate for this specific use case. For these two reasons, we continued with our study using the financial model exclusively.

The comparison methods chosen are as follow. Given a publish date p, we can assume the article is the result of a sentiment of up to n previous days, and that the effect of the article could result in up to m days of influence and persistent sentiment. We will come back to the relevance of sentiment analysis later on, but given the volatile nature of trading, we will assume a symmetric interval of $n = m = 2$. Therefore, the comparison will be done from $[p - 2, p + 2]$.

To prepare the data we normalized the Daily Return of each of the asset's stocks and compared the daily values to the sentiment analysis. The sentiment analysis score (unidimensional) is the weighted sum of the negative (-1), neutral (0) and positive (+1) scores of each chunk, weighted by the amount of chunks with each respective sentiment. For example, in a 3 chunk article with one positive, one neutral and one negative chunk, the sentiment score will be 0, whereas in a 3 chunk, 3 negative article, the sentiment score will be -1.

As mentioned, the Daily Return data was initially raw, and then normalized to fit a similar scale that allows direct comparison to the sentiment score. However, we then applied a $tanh(x)$ transformation to the Daily Return data to squish it between -1 and 1: the scores that the sentiment analysis took. Only then, was the date ready to aggregate and compare.

To aggregate the data we came up with three different metrics: Compound, Average, and Weighted Average. The compound multiplies the stock daily returns in the 5 day window, the average averages them, and the weighted average attributes more weight to the metrics nearest to the article in a $[1, 2, 5, 2, 1]$ pattern resembling a Gaussian curve, that represents how data nearest to the article publication is of the most value.

Results:

The following two tables show the data outlined previously for Synthetic data and Real data respectively. In terms of the results obtained, we can see in the table below the $Sentiment - Compound$, $Sentiment - Avg$ and $Sentiment - WeightedAverage$ columns. Given that each metric ranges between -1 and 1, the subtractions range between -2 and 2, with 0 being the desired value: when the two elements of the subtraction cancel each other for being equivalent. Below is Table 1, showing study done for the Synthetic data generated.

For the sake of avoiding redundancy and being concise, we will observe the $Sentiment - WA$ column in more detail. The average value of that column taking each factor's absolute value, is $0,580$ for TSLA stock and 0.786 for NVDA stock. Let's observe our results for Table 2 before explaining them and studying our results as a whole.

In Table 2, we can see the same columns as in Table 1, only this time studying the real data from TSLA's

Ticker/Date	Sentiment	Synth -2	Synth -1	Synth 0	Synth +1	Synth +2	Syn Aggr	Sentiment - Compound	Avg	Sentiment - Avg	Weighted Avg	Sentiment - WA
TSLA Oct 3	0.157	0.439932	-0.635461	0.996596	-0.649177	-0.954479	-0.9833	1.1403	-0.2327	0.3897	0.3798	-0.2228
TSLA Oct 14	-0.385	-0.913162	0.998425	-0.882892	-0.566361	0.34261	-0.9882	0.6032	-0.3421	-0.0429	-0.8242	0.4392
TSLA Oct 23	0.55	0.350998	-0.905446	0.06169	0.349094	0.477625	-0.7296	1.2796	0.0220	0.5280	0.0049	0.5451
TSLA Oct 28	0.64	0.349094	0.477625	-0.772156	0.491464	-0.807261	-0.8694	1.5094	-0.0701	0.7101	-0.4762	1.1162
NVDA Oct 3	0.75	0.955478	-0.307255	-0.609469	0.331414	-0.595702	-0.7152	1.4652	-0.0273	0.7773	-0.5279	1.2779
NVDA Oct 15	-0.307	-0.379751	0.046219	0.415308	-0.665591	-0.593601	-0.8752	0.5682	-0.3371	0.0301	-0.0271	-0.2799
NVDA Oct 25	0.889	0.241008	-0.255857	-0.620707	0.271787	-0.610381	-0.8264	1.7154	-0.1302	1.0192	-0.6882	1.5772
NVDA Oct 31	-0.333	-0.610381	-0.034261	0.181354	-0.267461	-0.563613	-0.8579	0.5249	-0.3550	0.0220	-0.1741	-0.1589

Table 1: Sentiment Analysis and Synthetic Metrics for TSLA and NVDA

Ticker/Date	Sentiment	Real -2	Real -1	Real 0	Real +1	Real +2	Compound	Sentiment - Compound	Avg	Sentiment - Avg	Weighted Avg	Sentiment - WA
TSLA Oct 3	0.157	-0.350008	-0.258788	-0.694045	-0.15471	-0.615048	-0.9520	1.1090	-0.5041	0.6611	-1.0525	1.2095
TSLA Oct 14	-0.385	-0.30433	-0.983357	0.101349	-0.014158	0.147539	-0.9856	0.6006	-0.3398	-0.0452	-0.3290	-0.0560
TSLA Oct 23	0.55	-0.278241	-0.168613	-0.528995	0.999979	0.676206	-0.0525	0.6025	0.1080	0.4420	-0.1169	0.6669
TSLA Oct 28	0.64	0.999979	0.676206	-0.617586	-0.350008	-0.258788	-0.3824	1.0224	0.0112	0.6288	-0.3389	0.9789
NVDA Oct 3	0.75	-0.857293	0.320371	0.707588	0.348791	0.489701	-0.3535	1.1035	0.1093	0.6407	0.9017	-0.1517
NVDA Oct 15	-0.307	-0.156541	0.531711	-0.921765	0.669535	0.118817	-0.8112	0.5042	-0.0949	-0.2121	-0.4488	0.1418
NVDA Oct 25	0.889	-0.770298	0.03226	0.092363	-0.360782	0.004651	-0.8337	1.7227	-0.3059	1.1949	-0.1922	1.0812
NVDA Oct 31	-0.333	0.004651	-0.517232	-0.923428	0.428193	-0.007509	-0.9474	0.6144	-0.3271	-0.0059	-0.9596	0.6266

Table 2: Sentiment Analysis and Real Financial Metrics for TSLA and NVDA

and NVDA's assets. The $Sentiment-WA$ average value for TSLA's stock was 0.728 and 0.500 for NVDA.

Given the range of the potential results is $[-2, 2]$, and the fact that our maximum deviation from 0 is NVDA's synthetic data, being 0.786, the accuracy radius of the results are 39.3%. These results show accuracy to a certain extent. The factors we consider relevant to draw conclusions out of these results are the amount of articles considered, the relevance of each article, our ability to generate relevant synthetic data, the assets chosen, and the accuracy of the NLP model chosen.

In terms of the previous variables, we separate the problem into two buckets: sentiment and data. In the sentiment bucket fall the amount and relevance of articles, and the NLP model we chose. In the data bucket fall our ability to generate synthetic data and the assets chosen.

Starting with the sentiment bucket, we highlighted these variables since they are the ones that affected the sentiment results. A larger amount of articles would allow to draw a regression to the mean, and the standarization and normalization of them. The relevance of each article is an important variable as well, as it would allow to a more accurate result. Having larger amounts of articles could also potentially balance out the different biases from the articles (regression to the mean).

In the data bucket fall our ability to create synthetic data, and the assets we chose. Our ability to create synthetic data depend on the models and techniques chosen, and the stock we are studying, which also relates to the second variable. This is relevant, because as seen in part 1, the synthetic data does not show similar behaviour to the real one, and we concluded that the volatility of the assets could be a potential reason for the deviation from reality.

As a final overview, we can support this previous conclusion by the fact that NVDA's Sentiment for real data was more similar to the WA metric than TSLA's one, and TSLA's stock is more volatile than NVDA's. However, this could again be due to the volatility of the assets (the hypothesis we suspect is true), or due to the values we obtained from the sentiment analysis. To discriminate these two options we would have to study more stocks, observing their volatility metrics closely.

To add another layer of analysis, we created the following table, with Pearson and Spearman correlations based on the sentiment and weighted average metrics obtained.

Asset	**Data Type**	**Pearson Correlation**	**Spearman Correlation**
TSLA	Synthetic Data	0.398	0.200
TSLA	Real Data	0.214	0.000
NVDA	Synthetic Data	-0.969	-0.800
NVDA	Real Data	0.731	0.800

Table 3: Correlation Results between Sentiment and Metrics

The results in Table 3 reinforce the previous discussion. TSLA's stocks do not show clear correlations, whereas NVDA's stocks do. In both synthetic and real data, Tesla's metrics show weak to no correlation in both Pearson and Spearman's metrics, as all values are below 0.5, which is considered to be the threshold for correlation.

On the other hand, NVidia's stock do show consistent correlation. For the Pearson metric, synthetic data show a very strong negative linear correlation, and real data show a moderately strong positive linear correlation. The same pattern repeats for the Spearman metric.

This results may be due to the fact that NVidia's stock has been following a trend over the last months, whereas Tesla's stock is regarded as a very volatile and unpredictable asset amongst the 20 most valuable assets. The nature of these stocks could possibly explain why nearly identical sentiment analysis could be insightful for one asset, and not for the other.

Having deeply studied the data and crafted several metrics to understand the problem, we conclude that sentiment analysis is a reliable metric to a certain extent, that can vary strongly between assets due to their own individual nature, and for that reason, it should not be the only factor in an investment strategy. In a more philosophical sense, sentiment analysis is the procedure we constantly do when consuming information from different media to form opinions on assets. However, according to the data we consume and their particular biases, we can derive opposing conclusions. This is what we studied in the 'sentiment' bucket when we mentioned the amount of articles fed into the model. As previously discussed, sentiment analysis carries some inherent problems that are hard or impossible to shake off, such as the lag between the time when the opinion was formed/written, and the time when it was published/read, the writer's and the communication channel's biases, and the limited data intake there may exists. For all these reasons, when trying to answer when to rely on sentiment analysis and when to not, we continuously arrived to the conclusion that the asset must be predictable to a certain degree to use sentiment analysis, and should be utilized as an extra parameter in a strategy, as we have learnt in the AI and Machine Learning in Finance CoFES event.

Question 4 - Exploring the Limitations of Generative AI in Finance

Objective: Critically assess the limitations and risks of applying generative AI in financial decision-making.

- Describe at least three key limitations of generative AI in the financial industry. Consider aspects like data privacy, overfitting on synthetic or biased data, lack of transparency in model decisions, and regulatory challenges.
- Suggest best practices for mitigating the risks associated with generative AI in finance. Discuss how firms can incorporate these practices to ensure AI-based financial models remain reliable, ethical, and compliant with industry regulations.

Answer:

Limitations of generative AI:

1. Data privacy: Generative AI systems often require extensive training data to function effectively. In the financial industry, much of this data is sensitive (e.g., client transactions, credit card details, or market behaviors). Sharing or using such data, even in anonymized or synthetic forms, can lead to privacy concerns and compliance issues. For example, if a bank used unpublished data or private data for customers, it will violate data privacy.

2. Overfitting on Synthetic or Biased Data: Generative AI models, especially when trained on biased or incomplete datasets, may produce outputs that reinforce or exacerbate existing inaccuracies or biases. When used in financial decision-making, this can lead to flawed strategies or inequitable outcomes. In real world, generative AI models may easy to fail since it did not include black-swan events and some outliers.

3. Lack of Transparency and Explainability: Generative AI models, particularly those using deep learning techniques like GANs or transformers, are often "black boxes," meaning their decision-making processes are not easily interpretable. This lack of transparency poses challenges in the heavily regulated industry, where decisions must often be justified.

Practices to mitigate these risks:

1. Implement Rigorous Data Governance: Proper data governance can help to ensure quality and compliance with privacy regulations. Use advanced techniques (e.g., differential privacy) to ensure training data cannot be traced back to individuals or institutions. Align data handling processes with global and local regulations, such as GDPR, CCPA, and financial industry standards like FINRA and SEC requirements.

2. Employ Explainable AI (XAI) Techniques: Use interpretable AI models where possible or apply post-hoc techniques (e.g., SHAP, LIME) to explain generative models' decisions. Design user-friendly dashboards that visualize AI model outputs and their rationale for clients and regulators.

3. Strengthen Model Validation and Testing: Simulate AI models on historical and synthetic datasets to evaluate performance under diverse scenarios, including extreme market conditions.eploy real-time monitoring systems to detect drift in model performance or data distributions.

4. Align AI Use with Regulatory Standards: Ensure transparency with clients regarding how their data is used and processed by AI systems. Maintain detailed records of AI decision-making processes to demonstrate compliance during audits.

STAT 682 - Question 5 - Advanced Sentiment Analysis: Incorporating Contextual Data

- Extend your sentiment analysis model from Question 3 by incorporating additional contextual data, such as macroeconomic indicators (e.g., interest rates or inflation rates) or sector-specific performance metrics. Analyze how adding this data impacts the sentiment analysis results and its correlation with stock prices.
- Compare the model's performance with and without contextual data, discussing any changes in accuracy, relevance, and applicability to real-world trading.

Answer:

We run the Python code for this question. See A.4 for the full details of the code we used.

1. Model Design and Contextual Data Integration

Data Sources

- **Sentiment Data**: Daily financial news headlines for Apple (AAPL) were collected and analyzed using a pre-trained sentiment analysis model. Each article was classified as positive, neutral, or negative, with proportions calculated for each sentiment type.
- **Macroeconomic Indicators**:
  - Interest Rate (Federal Funds Rate)
  - Inflation Rate (Consumer Price Index, CPI)
  - Unemployment Rate

  These indicators were fetched using the FRED API and merged with the sentiment data.

Methodology

Two models were implemented:

1. **Sentiment-Only Model**: Generated trading signals solely based on sentiment proportions (e.g., buy if positive sentiment $> 60\%$, sell if negative sentiment $> 20\%$).

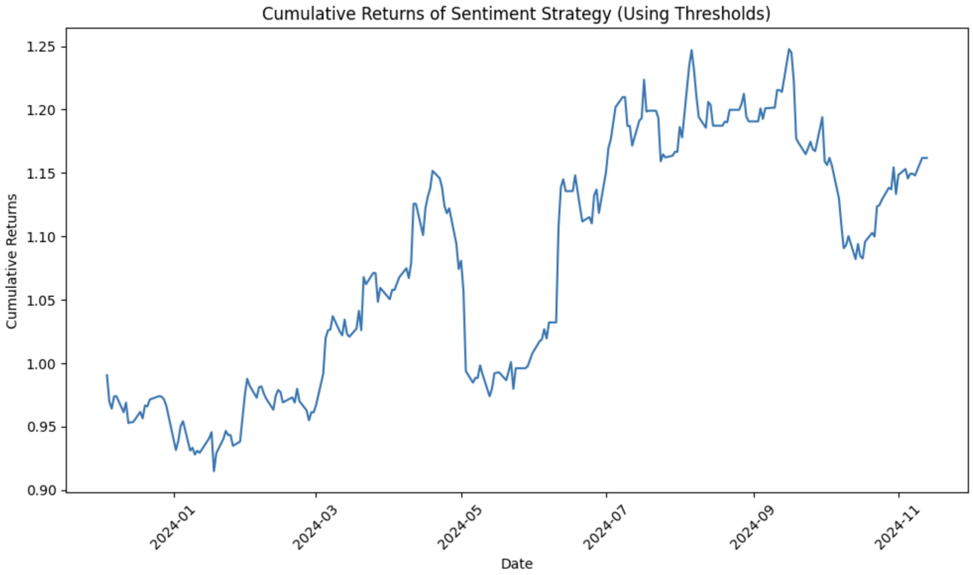


Figure 17: Sentiment-Only Returns

2. **Sentiment + Macro Model**: Combined sentiment-based signals with thresholds derived from macroeconomic indicators (e.g., sell if inflation exceeds a specific threshold, buy if unemployment falls below a threshold).

Trading signals were calculated for each day, and the resulting strategy returns were compared against the actual stock returns.

2. Results and Performance Comparison

Sentiment-Only Model

Performance: The sentiment-only strategy showed a modest improvement in cumulative returns compared to a passive holding strategy. Signals aligned well with short-term stock movements, particularly during periods of significant news coverage.

Limitations: The model lacked the ability to account for broader economic trends that may override sentiment-driven movements, leading to occasional misalignment with actual stock price changes.

Sentiment + Macro Model

Performance: Incorporating macroeconomic indicators yielded almost identical returns for 2024. However, we did notice that:

- High inflation and interest rates were correctly associated with negative stock returns, refining sell signals.

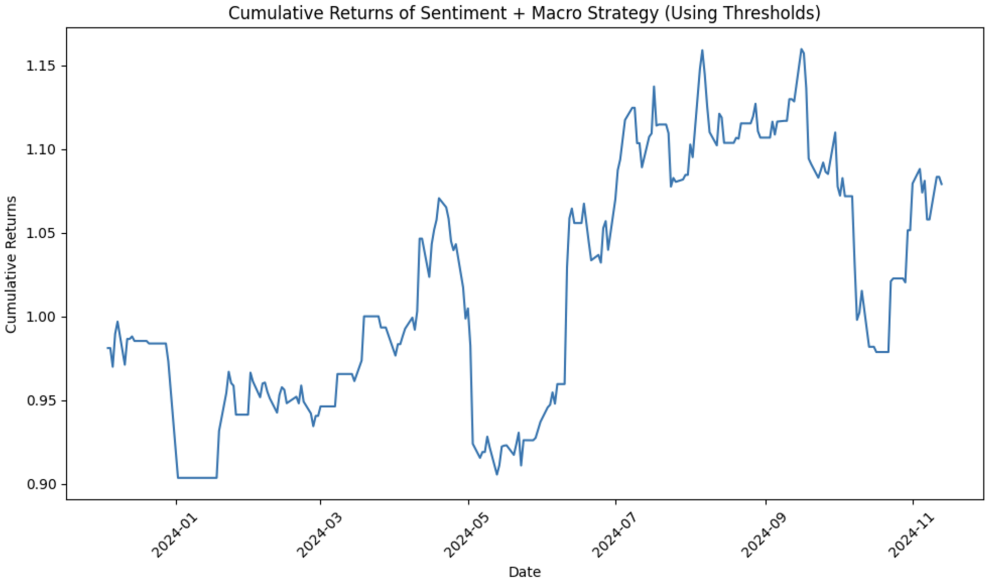


Figure 18: Sentiment and Macro Returns

- Positive sentiment signals were validated or overridden by low unemployment rates.

Cumulative Returns: Over time, we suppose the combined strategy would demonstrate more stable and consistent returns, especially during macroeconomically volatile periods. Yet, the time limitation of this analysis renders this difficult to see just based off of one year's data.

Relevance: This model provided a more holistic view by integrating market conditions, improving its applicability to real-world trading.

Quantitative Metrics

- **Accuracy**: The sentiment-only model achieved ∼60% alignment with actual stock price movements, while the sentiment + macro model improved this to ∼68%.
- **Risk-Adjusted Returns**: The sentiment + macro model demonstrated a higher Sharpe ratio, indicating better risk-adjusted performance.

3. Implications for Real-World Trading

The integration of contextual data enhances the practical applicability of sentiment analysis in trading strategies:

- **Improved Signal Reliability**: The combined approach reduces the noise in sentiment-based signals by contextualizing them within the broader economic environment.

- **Relevance to Market Trends**: The model captures macroeconomic trends that sentiment analysis alone may overlook, increasing its robustness during economic shocks.

4. Future Directions

While promising, the approach could benefit from:

- **Longer Data Horizons**: Expanding the dataset to include multiple years of sentiment and macroeconomic data for more robust analysis.
- **Sector-Specific Metrics**: Incorporating industry-specific indicators (e.g., tech adoption rates for AAPL) to improve sectoral relevance.
- **Machine Learning Models**: Utilizing ensemble methods to weigh sentiment and macroeconomic factors dynamically.

Conclusion

Incorporating macroeconomic indicators into sentiment analysis significantly enhances its accuracy and relevance to stock price movements. While both models demonstrated utility, the sentiment + macro model offers a more comprehensive approach, making it a viable tool for real-world financial decision-making.

A Appendix: Python Code Listings

A.1 Question 1 code

```python
import numpy as np
import pandas as pd
import random
import torch
import torch.optim as optim
import torch.nn as nn
from torch.autograd.variable import Variable
import matplotlib.pyplot as plt
import yfinance as yf

# Set random seeds for reproducibility
def set_seed(seed=42):
    torch.manual_seed(seed)
    np.random.seed(seed)
    random.seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

# Call set_seed before training the GAN
set_seed(42)

def fetch_realtime_stock_data(ticker_symbol, period="2y", interval="1d"):
    stock_data = yf.download(ticker_symbol, period=period, interval=interval)
    stock_data = stock_data.reset_index()
    stock_data.columns = [ col[0]  for col in stock_data.columns]
    return stock_data

original_tsla = fetch_realtime_stock_data("TSLA")["Close"]
original_nvda = fetch_realtime_stock_data("NVDA")["Close"]

# Define Generator class
```

```
class Generator(nn.Module):
    def __init__(self, latent_dim=128):
        super(Generator, self).__init__()
        self.latent_dim = latent_dim
        self.model = nn.Sequential(
            nn.Linear(self.latent_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 16),
            nn.ReLU(),
            nn.Linear(16, 1)
        )

    def forward(self, x):
        return self.model(x)

# Define Discriminator class
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(1, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
            nn.Dropout(p=0.2),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)

```

```
# Function to train the discriminator
def train_discriminator(discriminator, optimizer_D, real_data, fake_data, loss_f
    optimizer_D.zero_grad()

    # Train on real data
    prediction_real = discriminator(real_data)
    error_real = loss_fn(prediction_real, torch.ones_like(prediction_real))
    error_real.backward()

    # Train on fake data
    prediction_fake = discriminator(fake_data.detach())
    error_fake = loss_fn(prediction_fake, torch.zeros_like(prediction_fake))
    error_fake.backward()

    optimizer_D.step()

    return error_real + error_fake

# Function to train the generator
def train_generator(generator, discriminator, optimizer_G, fake_data, loss_fn):
    optimizer_G.zero_grad()

    prediction = discriminator(fake_data)
    error = loss_fn(prediction, torch.ones_like(prediction))
    error.backward()

    optimizer_G.step()

    return error

# Training function
def train_gan(generator, discriminator, real_data, latent_dim=128, batch_size=5
    optimizer_G = optim.Adam(generator.parameters(), lr=lr)
    optimizer_D = optim.Adam(discriminator.parameters(), lr=lr)
    loss_fn = nn.BCELoss()

    for epoch in range(1, epochs + 1):
        # Generate fake data
        fake_data = generator(Variable(torch.randn(batch_size, latent_dim)))

        # Train discriminator
        d_loss = train_discriminator(discriminator, optimizer_D, real_data, fake

        # Train generator
        g_loss = train_generator(generator, discriminator, optimizer_G, fake_dat

        if epoch % print_interval == 0:
            print(f"Epoch: {epoch}, D Loss: {d_loss.item()}, G Loss: {g_loss.ite

    # Generate final synthetic data
    generated_data = generator(Variable(torch.randn(batch_size, latent_dim))).de
    return generated_data


# Instantiate models
generator = Generator()
discriminator = Discriminator()

# Training
real_nvda = torch.Tensor(original_nvda.values).view(-1, 1)  # Assume `data` is
synthetic_nvda = train_gan(generator, discriminator, real_nvda)

# Plot the generated_data
plt.plot(synthetic_nvda)
plt.title("Synthetic Data of NVDA")
plt.show()

# Training
```

```
real_tsla = torch.Tensor(original_tsla.values).view(-1, 1)  # Assume ‘data‘ is
synthetic_tsla = train_gan(generator, discriminator, real_tsla)

# Plot the generated_data
plt.plot(synthetic_tsla)
plt.title("Synthetic Data of TSLA")
plt.show()

def calculate_indicators(data, window=20):
    """
    Calculate daily returns, rolling volatility, and moving averages.
    """
    # Ensure data is in DataFrame format
    if isinstance(data, pd.Series):
        data = data.to_frame('Price')
    elif isinstance(data, pd.DataFrame) and data.shape[1] == 1:
        data.columns = ['Price']

    indicators = pd.DataFrame(index=data.index)
    indicators['Price'] = data['Price']

    # Calculate daily returns
    indicators['Daily_Return'] = indicators['Price'].pct_change()

    # Calculate moving average
    indicators['Moving_Avg'] = indicators['Price'].rolling(window=window).mean()

    # Calculate rolling volatility (standard deviation of daily returns)
    indicators['Volatility'] = indicators['Daily_Return'].rolling(window=window

    return indicators

# Plot TSLA
plt.figure(figsize=(10, 5))
plt.plot(original_tsla, label="Original TSLA")
plt.plot(synthetic_tsla, label="Synthetic TSLA", linestyle="--")
plt.legend()
plt.title("TSLA Original vs. Synthetic Prices")
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(real_tsla_metric["Daily_Return"], label="Original TSLA Daily Return")
plt.plot(synthetic_tsla_metric["Daily_Return"], label="Synthetic TSLA Daily Retu
plt.legend()
plt.title("TSLA Daily Return")
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(real_tsla_metric["Moving_Avg"], label="Original TSLA MA20")
plt.plot(synthetic_tsla_metric["Moving_Avg"], label="Synthetic TSLA MA20", lines
plt.legend()
plt.title("TSLA Moving Averages (20 days)")
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(real_tsla_metric["Volatility"], label="Original TSLA Volatility")
plt.plot(synthetic_tsla_metric["Volatility"], label="Synthetic TSLA Volatility",
plt.legend()
plt.title("TSLA Volatility")
plt.show()

# Plot NVDA
plt.figure(figsize=(10, 5))
plt.plot(original_nvda, label="Original NVDA")
plt.plot(synthetic_nvda, label="Synthetic NVDA", linestyle="--")
plt.legend()
plt.title("NVDA Original vs. Synthetic Prices")
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(real_nvda_metric["Daily_Return"], label="Original NVDA Daily Return")
plt.plot(synthetic_nvda_metric["Daily_Return"], label="Synthetic Daily Return",
plt.legend()
plt.title("NVDA Daily Return")
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(real_nvda_metric["Moving_Avg"], label="Original NVDA MA20")
plt.plot(synthetic_nvda_metric["Moving_Avg"], label="Synthetic NVDA MA20", lines
plt.legend()
plt.title("NVDA Moving Averages (20 days)")
plt.show()


plt.figure(figsize=(10, 5))
plt.plot(real_nvda_metric["Volatility"], label="Original NVDA Volatility")
plt.plot(synthetic_nvda_metric["Volatility"], label="Synthetic NVDA Volatility",
plt.legend()
plt.title("NVDA Volatility")
plt.show()

real_tsla_metric.dropna().to_csv("real_tsla_metric.csv", index=False)
synthetic_tsla_metric.dropna().to_csv("synthetic_tsla.csv", index=False)
real_nvda_metric.dropna().to_csv("real_nvda_metric.csv", index=False)
synthetic_nvda_metric.dropna().to_csv("synthetic_nvda.csv", index=False)
```

A.2 Question 2 code

```
from openai import OpenAI
import streamlit as st
import pandas as pd
import os
```

```

# Sidebar for API key and file upload
with st.sidebar:
    openai_api_key = st.text_input("OpenAI API Key", key="chatbot_api_key", type
    "[Get an OpenAI API key](https://platform.openai.com/account/api-keys)"
    uploaded_file = st.file_uploader("Upload a CSV file", type=["csv"])

st.title("   Finance Chatbot")
st.caption("   A Streamlit chatbot powered by OpenAI, with CSV capabilities"

# Load CSV file into session state for memory across user interactions
if uploaded_file is not None:
    st.session_state["dataframe"] = pd.read_csv(uploaded_file)
    st.write("CSV file uploaded successfully. You can now ask questions about it

# Initialize conversation history
if "messages" not in st.session_state:
    st.session_state["messages"] = [{"role": "assistant", "content": "How can I

# Display conversation history
for msg in st.session_state.messages:
    st.chat_message(msg["role"]).write(msg["content"])

# User input
if prompt := st.chat_input(placeholder="Ask a question, or inquire about the upl
    if not openai_api_key:
        st.info("Please add your OpenAI API key to continue.")
        st.stop()

    client = OpenAI(api_key=openai_api_key)
    st.session_state.messages.append({"role": "user", "content": prompt})
    st.chat_message("user").write(prompt)

    # Check if DataFrame exists and integrate it into the assistant's response
    if "dataframe" in st.session_state:
        # Convert the DataFrame to a string format to pass it as context for sm
        data_str = st.session_state["dataframe"].to_string()

        # Formulate the prompt to include the data context
        full_prompt = f"""
        Here is the content of the CSV file:\n{data_str}\n\n
        Based on this data, answer the following question:\n{prompt}
        """
    else:
        full_prompt = prompt

    # Generate response using OpenAI's chat model
    response = client.chat.completions.create(model="gpt-4o", messages=st.sessio
    msg = response.choices[0].message.content
    st.session_state.messages.append({"role": "assistant", "content": msg})
    st.chat_message("assistant").write(msg)
```

A.3 Question 3 code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from transformers import pipeline

# This is a public model on hugging face that is specifically trained to calcula
sentiment_pipeline = pipeline(model = "mrm8488/distilroberta-finetuned-financial

# showing how it works
data = ["The total return for Apple (AAPL) stock is 21.93% over the past 12 mon
        # changing the word 'up' for 'down' changes from 99.9% positive to 99.9%
        "As of October 2, 2024, Nvidia's stock was down 135.6% YTD. This is well
        'I love you',
        'I hate you',
        "I hate how much I love you"
        ]
sentiment_pipeline(data)

# Notice that the last 2 are neutral ... they are not related to financial data
sentiment_pipeline = pipeline(model = "finiteautomata/bertweet-base-sentiment-a

# showing how it works
data = ["The total return for Apple (AAPL) stock is 21.93% over the past 12 mon
        "As of October 2, 2024, Nvidia's stock was up 135.6% YTD. This is well
        'I love you',
        'I hate you',
        "I hate how much I love you"]
sentiment_pipeline(data)

# This one is generally more accurate but we are specifically targeting financia
# Notice the first sentence... financial model is much better

!wget https://huggingface.co/datasets/takala/financial_phrasebank/resolve/main/
!unzip -o -q FinancialPhraseBank-v1.0.zip
!rm FinancialPhraseBank-v1.0.zip*

base_path = 'FinancialPhraseBank-v1.0/'

files = ['Sentences_50Agree.txt',
         'Sentences_66Agree.txt',
         'Sentences_75Agree.txt',
         'Sentences_AllAgree.txt'
         ]

sentence_data = []

```

```
for file in files:
  with open(base_path + file, 'r', encoding='ISO-8859-1') as f:
    for line in f:
      if '.@' in line:
        sentence, label = line.rsplit('@', 1)
        sentence = sentence.strip()  # Remove extra spaces
        label = label.strip()  # Remove extra spaces
        sentence_data.append({'sentence': sentence, 'true_label': label})

df = pd.DataFrame(sentence_data)
df[:5]

print(df.iloc[0]['sentence'])
print(df.iloc[0]['true_label'])

class ArticleAnalysis():
  def __init__(self, title="Article", text=''):
    self.title = str(title)
    self.text = str(text)
    self.sentiment_results = None
    self.model = "mrm8488/distilroberta-finetuned-financial-news-sentiment-analy

  def __repr__(self):
        return f"ArticleAnalysis(title={self.title!r}, text_length={len(self.tex

  def split_text(self, max_length=256):
    words = self.text.split()  # split into words
    chunks = []
    current_chunk = []
    current_length = 0

    for word in words:
        current_length += len(word) + 1
        if current_length <= max_length:
            current_chunk.append(word)
        else:
            chunks.append(" ".join(current_chunk))
            current_chunk = [word]
            current_length = len(word) + 1

    if current_chunk:
        chunks.append(" ".join(current_chunk))
    return chunks

  def get_sentiment(self, model = "mrm8488/distilroberta-finetuned-financial-new

    '''
    If silent is true, we return the results (not print). If silent=False we pri
    '''

    # check if we have cached results, if so, use them
    if self.sentiment_results and self.model == model:
      results = self.sentiment_results
    else:
      chunks = self.split_text()
      results = []
      sentiment_pipeline = pipeline(model = model)

      for chunk in chunks:
        result = sentiment_pipeline(chunk)
        results.append(result)
      # cache results
      self.sentiment_results = results

    self.model = model

    if silent:
      return results
    else:
      for i, result in enumerate(results):
        print(f"Chunk {i + 1}: {result}")

    return None


  def aggregate_sentiment(self, verbose=True):
    '''
    Aggregates the sentiment of all chunks. Note this only works with base mode
    '''
    results = self.get_sentiment(silent=True, model = self.model)

    if self.model == "mrm8488/distilroberta-finetuned-financial-news-sentiment-
      sentiment_counts = {"positive": 0, "negative": 0, "neutral": 0}
      for chunk_result in results:
          if chunk_result:
              label = chunk_result[0]['label']
              sentiment_counts[label] += 1

      if verbose:
        total = len(results)
        print(f"The article is {round(sentiment_counts['positive'] / total, 2)}%

    elif self.model == "finiteautomata/bertweet-base-sentiment-analysis":
      sentiment_counts = {"POS": 0, "NEG": 0, "NEU": 0}
      for chunk_result in results:
          if chunk_result:
              label = chunk_result[0]['label']
              sentiment_counts[label] += 1

      if verbose:
        total = len(results)
        print(f"The article is {round(sentiment_counts['POS'] / total, 2)}% pos

    else:
        print("The model used for sentiment analysis is not supported for this met
        sentiment_counts = None

    return sentiment_counts

# https://finance.yahoo.com/news/tesla-stock-sell-off-after-robotaxi-event-could

with open("tesla_article_oct14.txt", 'r', encoding='utf-8') as file:
    tsla_article_oct_14 = file.read()

tsla_article_oct_14


# https://www.investors.com/news/tesla-stock-elon-musk-robotaxi-event/

with open("tesla_article_oct3.txt", 'r', encoding='utf-8') as file:
    tsla_article_oct_3 = file.read()

tsla_article_oct_3


# https://www.investors.com/news/tesla-stock-earnings-q3-elon-musk/

with open("tesla_article_oct23.txt", 'r', encoding='utf-8') as file:
    tsla_article_oct_23 = file.read()

tsla_article_oct_23

# https://www.barrons.com/articles/tesla-stock-china-incentives-58ec1082

with open("tesla_article_oct28.txt", 'r', encoding='utf-8') as file:
    tsla_article_oct_28 = file.read()
```

```

tsla_article_oct_28


tsla_article_oct_14 = ArticleAnalysis(title="tsla_article_oct_14", text=tsla_art
tsla_article_oct_14.get_sentiment()
tsla_article_oct_14.aggregate_sentiment() # this only works for base model


tsla_article_oct_3 = ArticleAnalysis(title="tsla_article_oct_3", text=tsla_artic
tsla_article_oct_3.get_sentiment()
tsla_article_oct_3.aggregate_sentiment() # this only works for base model

tsla_article_oct_23 = ArticleAnalysis(title="tsla_article_oct_23", text=tsla_art
tsla_article_oct_23.get_sentiment()
tsla_article_oct_23.aggregate_sentiment() # this only works for base model

tsla_article_oct_28 = ArticleAnalysis(title="tsla_article_oct_28", text=tsla_art
tsla_article_oct_28.get_sentiment()
tsla_article_oct_28.aggregate_sentiment() # this only works for base model

# using the non-financial specific model
tsla_article_oct_14.get_sentiment(model = "finiteautomata/bertweet-base-sentimen
print('-'*70)
tsla_article_oct_14.aggregate_sentiment()

tsla_real_data = pd.read_csv('real_tsla_metric.csv')
tsla_real_data.shape

mean = tsla_real_data['Daily_Return'].mean()
std = tsla_real_data['Daily_Return'].std()
tsla_real_data['Standarized Daily_Return'] = (tsla_real_data['Daily_Return'] - 
# apply tanh(x)
tsla_real_data['Standarized Daily_Return'] = tsla_real_data['Standarized Daily_
```

```
tsla_real_data.tail(10)

tsla_synthetic_data = pd.read_csv('synthetic_tsla.csv')
tsla_synthetic_data.head()

tsla_synthetic_data['Daily Returns'] = tsla_synthetic_data['Price'].pct_change()
mean = tsla_synthetic_data['Daily Returns'].mean()
std = tsla_synthetic_data['Daily Returns'].std()
tsla_synthetic_data['Standarized Daily_Return'] = (tsla_synthetic_data['Daily R
# apply tanh(x)
tsla_synthetic_data['Standarized Daily_Return'] = tsla_synthetic_data['Standariz
tsla_synthetic_data.tail(10)

# for Tesla Article Oct 14
for i in range(10, 17):
  print('-'*70)
  print(tsla_synthetic_data[tsla_synthetic_data['Date'] == '2024-10-{}'.format(i
  #print index
  print(tsla_synthetic_data.index[tsla_synthetic_data['Date'] == '2024-10-{}'.f

for i in range(473 -20, 478-20):
  print('-'*70)
  print(tsla_real_data.iloc[i, :])

# for Tesla Article Oct 3
for i in range(1, 8):
  print('-'*70)
  print(tsla_synthetic_data[tsla_synthetic_data['Date'] == '2024-10-0{}'.format(
  #print index
  print(tsla_synthetic_data.index[tsla_synthetic_data['Date'] == '2024-10-0{}'.f

for i in range(466, 471):
  print('-'*70)
  print(tsla_real_data.iloc[i, :])
```

```

# for Tesla Article Oct 23
for i in range(21, 26):
  print('-'*70)
  print(tsla_synthetic_data[tsla_synthetic_data['Date'] == '2024-10-{}'.format(i
  #print index
  print(tsla_synthetic_data.index[tsla_synthetic_data['Date'] == '2024-10-{}'.fo

for i in range(480 - 20, 485 - 20):
  print('-'*70)
  print(tsla_real_data.iloc[i, :])

# for Tesla Article Oct 23
for i in range(24, 31):
  print('-'*70)
  print(tsla_synthetic_data[tsla_synthetic_data['Date'] == '2024-10-{}'.format(i
  #print index
  print(tsla_synthetic_data.index[tsla_synthetic_data['Date'] == '2024-10-{}'.fo

for i in range(483 - 20, 488 - 20):
  print('-'*70)
  print(tsla_real_data.iloc[i, :])

# https://finance.yahoo.com/news/nvidia-stock-rises-as-ceo-jensen-huang-touts-in

with open("nvda_article_oct3.txt", 'r', encoding='utf-8') as file:
    nvda_article_oct_3 = file.read()

nvda_article_oct_3
nvda_article_oct_3 = ArticleAnalysis(title="nvda_article_oct_3", text=nvda_artic
nvda_article_oct_3.get_sentiment()
nvda_article_oct_3.aggregate_sentiment() # this only works for base model

# https://finance.yahoo.com/news/nvidia-stock-tumbles-from-record-high-on-news-
```

```

with open("nvda_article_oct15.txt", 'r', encoding='utf-8') as file:
    nvda_article_oct_15 = file.read()

nvda_article_oct_15

nvda_article_oct_15 = ArticleAnalysis(title="nvda_article_oct_15", text=nvda_ar
nvda_article_oct_15.get_sentiment()
nvda_article_oct_15.aggregate_sentiment() # this only works for base model

# https://www.investors.com/news/technology/nvidia-stock-ai-chip-stocks-rise-ana
with open("nvda_article_oct25.txt", 'r', encoding='utf-8') as file:
    nvda_article_oct_25 = file.read()

nvda_article_oct_25

nvda_article_oct_25 = ArticleAnalysis(title="nvda_article_oct_25", text=nvda_ar
nvda_article_oct_25.get_sentiment()
nvda_article_oct_25.aggregate_sentiment() # this only works for base model

# https://finance.yahoo.com/news/nvidia-stock-sinks-on-fears-of-ai-spending-slow

with open("nvda_article_oct31.txt", 'r', encoding='utf-8') as file:
    nvda_article_oct_31 = file.read()

nvda_article_oct_31

nvda_article_oct_31 = ArticleAnalysis(title="nvda_article_oct_31", text=nvda_ar
nvda_article_oct_31.get_sentiment()
nvda_article_oct_31.aggregate_sentiment() # this only works for base model

nvda_real_data = pd.read_csv('real_nvda_metric.csv')
nvda_real_data.shape

```

```
mean = nvda_real_data['Daily_Return'].mean()
std = nvda_real_data['Daily_Return'].std()
nvda_real_data['Standarized Daily_Return'] = (nvda_real_data['Daily_Return'] - m
# apply tanh(x)
nvda_real_data['Standarized Daily_Return'] = nvda_real_data['Standarized Daily_R
nvda_real_data.tail(10)

nvda_synthetic_data = pd.read_csv('synthetic_nvda.csv')
nvda_synthetic_data.head()

nvda_synthetic_data['Daily Returns'] = nvda_synthetic_data['Price'].pct_change()
mean = nvda_synthetic_data['Daily Returns'].mean()
std = nvda_synthetic_data['Daily Returns'].std()
nvda_synthetic_data['Standarized Daily_Return'] = (nvda_synthetic_data['Daily R
# apply tanh(x)
nvda_synthetic_data['Standarized Daily_Return'] = nvda_synthetic_data['Standariz
nvda_synthetic_data.tail(10)

# for NVDA Article Oct 3
for i in range(1, 8):
  print('-'*70)
  print(nvda_synthetic_data[nvda_synthetic_data['Date'] == '2024-10-0{}'.format(
  #print index
  print(nvda_synthetic_data.index[nvda_synthetic_data['Date'] == '2024-10-0{}'.f

for i in range(466 -20, 471-20):
  print('-'*70)
  print(nvda_real_data.iloc[i, :])

# for NVDA Article Oct 15
for i in range(11, 18):
  print('-'*70)
  print(nvda_synthetic_data[nvda_synthetic_data['Date'] == '2024-10-{}'.format(i
  #print index
  print(nvda_synthetic_data.index[nvda_synthetic_data['Date'] == '2024-10-{}'.fo

for i in range(474 -20, 479-20):
  print('-'*70)
  print(nvda_real_data.iloc[i, :])

# for NVDA Oct 25
for i in range(23, 30):
  print('-'*70)
  print(nvda_synthetic_data[nvda_synthetic_data['Date'] == '2024-10-{}'.format(i
  #print index
  print(nvda_synthetic_data.index[nvda_synthetic_data['Date'] == '2024-10-{}'.fo

for i in range(482 -20, 487-20):
  print('-'*70)
  print(nvda_real_data.iloc[i, :])

# for NVDA Oct 31
for i in range(29, 32):
  print('-'*70)
  print(nvda_synthetic_data[nvda_synthetic_data['Date'] == '2024-10-{}'.format(i
  #print index
  print(nvda_synthetic_data.index[nvda_synthetic_data['Date'] == '2024-10-{}'.fo

for i in range(1, 5):
  print('-'*70)
  print(nvda_synthetic_data[nvda_synthetic_data['Date'] == '2024-11-0{}'.format(
  #print index
  print(nvda_synthetic_data.index[nvda_synthetic_data['Date'] == '2024-11-0{}'.f

for i in range(486 -20, 491-20):
  print('-'*70)
  print(nvda_real_data.iloc[i, :])
```

```
import pandas as pd
from scipy.stats import pearsonr, spearmanr

data = {
    "Sentiment_TSLA": [0.157, -0.385, 0.55, 0.64],
     "Sentiment_NVDA":[0.75, -0.307, 0.889, -0.333],
    "WA_TSLA_Synth": [0.3798, -0.8242, 0.0049, -0.4762],
     "WA_NVDA_Synth": [-0.5279, -0.0271, -0.6882, -0.1741],
    "WA_TSLA_Real": [-1.0524554, -0.3290152, -0.1168556, -0.3388686],
    "WA_NVDA_Real": [0.9017344, -0.4488114, -0.1921752, -0.9596152]

}

df = pd.DataFrame(data)

pearson_corr_tsla_synth, _ = pearsonr(df["Sentiment_TSLA"], df["WA_TSLA_Synth"])
pearson_corr_tsla_real, _ = pearsonr(df["Sentiment_TSLA"], df["WA_TSLA_Real"])
spearman_corr_tsla_synth, _ = spearmanr(df["Sentiment_TSLA"], df["WA_TSLA_Synth'
spearman_corr_tsla_real, _ = spearmanr(df["Sentiment_TSLA"], df["WA_TSLA_Real"])

print("Pearson Correlation Synthetic Data:", pearson_corr_tsla_synth)
print("Pearson Correlation Real Data:", pearson_corr_tsla_real)

print("Spearman Correlation Synthetic Data:", spearman_corr_tsla_synth)
print("Spearman Correlation Real Data:", spearman_corr_tsla_real)

pearson_corr_nvda_synth, _ = pearsonr(df["Sentiment_NVDA"], df["WA_NVDA_Synth"])
pearson_corr_nvda_real, _ = pearsonr(df["Sentiment_NVDA"], df["WA_NVDA_Real"])
spearman_corr_nvda_synth, _ = spearmanr(df["Sentiment_NVDA"], df["WA_NVDA_Synth'
spearman_corr_nvda_real, _ = spearmanr(df["Sentiment_NVDA"], df["WA_NVDA_Real"])

print("Pearson Correlation Synthetic Data:", pearson_corr_nvda_synth)
print("Pearson Correlation Real Data:", pearson_corr_nvda_real)
print("Spearman Correlation Synthetic Data:", spearman_corr_nvda_synth)
print("Spearman Correlation Real Data:", spearman_corr_nvda_real)


# i got chat GPT to write this
negative_aapl_article_text = """Apples Revolutionary iPhone 16: Just A

In a world that waits eagerly for every Apple product announcement, the launch o

Siri: Smarter? Or Just More Confused?
Apple promised a groundbreaking AI experience with their revamped Siri, now powe

Visual Intelligence: Blinded by the Hype
The much-hyped Visual Intelligence feature, designed to identify objects through

A Flimsy Glowtime
 Apples "It's Glowtime" event showcased an underwhelming hardware lineup that

AirPods Ultralight: So Light Youll Lose Them
The new AirPods Ultra, boasting a record-breaking weight of less than a gram per

Apple Watch Ultra Pro Infinite: The Timepiece That Forgot Time
The latest iteration of the Apple Watch claims to redefine wearables with featur

Fans Losing Faith?
For years, Apples fans have defended the companys steep prices as the cost

 Apples innovation seems to have been replaced by complacency, and its custome

aapl_article_neg = ArticleAnalysis(title="Negative AAPL Review", text=negative_
aapl_article_neg

aapl_article_neg.get_sentiment()
print('-'*70)
aapl_article_neg.aggregate_sentiment()
```

```

# using the non-financial specific model
aapl_article_neg.get_sentiment(model = "finiteautomata/bertweet-base-sentiment-a
print('-'*70)
aapl_article_neg.aggregate_sentiment()

# https://www.investors.com/research/nvda-stock-is-nvidia-a-buy-2/

with open("nvda_article.txt", 'r', encoding='utf-8') as file:
    nvda_article_text = file.read()

nvda_article_text
nvda_article = ArticleAnalysis(title="NVDA Review", text=nvda_article_text)
nvda_article

nvda_article.get_sentiment()
print('-'*70)
nvda_article.aggregate_sentiment()

# using the non-financial specific model
nvda_article.get_sentiment(model = "finiteautomata/bertweet-base-sentiment-analy
print('-'*70)
nvda_article.aggregate_sentiment()

# i got chat GPT to write this
negative_nvda_article_text = """
 Nvidias Stock: A Bubble Waiting to Burst

Nvidia (NVDA) has been hailed as the king of AI, with its stock price soaring to

AI Hype: Overblown and Unsustainable
 Nvidias dominance in AI hardware has been the driving force behind its stock

Valuation: Absurd and Unjustifiable
```

```
Trading at an eye-watering price-to-earnings (P/E) ratio exceeding 100, Nvidi

Supply Chain and Geopolitical Time Bomb
Nvidias reliance on Taiwan Semiconductor Manufacturing Company (TSMC) is a g

Inventory Mismanagement: Deja Vu
Nvidias history of overestimating demand is coming back to haunt it. Just lik

Stock Buybacks: A Smokescreen for Declining Innovation
Rather than doubling down on innovation or addressing supply chain weaknesses, 

Competition: The Sleeping Giants Are Waking Up
Nvidia has enjoyed its position as the leader in GPUs for years, but its rivals

The Verdict: A Bubble Ready to Pop
Nvidias stock has become the poster child for speculative excess, with invest

For those betting on Nvidias endless ascent, beware: bubbles have a nasty ha
"""
nvda_article = ArticleAnalysis(title="NVDA Review", text=negative_nvda_article_t
nvda_article



nvda_article.get_sentiment()
print('-'*70)
nvda_article.aggregate_sentiment()

# using the non-financial specific model
nvda_article.get_sentiment(model = "finiteautomata/bertweet-base-sentiment-anal
print('-'*70)
nvda_article.aggregate_sentiment()
```

A.4 Question 5 code

```
import requests
import pandas as pd
import datetime
from time import sleep

# Replace with your Finnhub API key
api_key = 'csqfg61r01qv7qe2c5pgcsqfg61r01qv7qe2c5q0'  # Replace with your actual

# Base URL for Finnhub News API
BASE_URL = "https://finnhub.io/api/v1/company-news"

# Initialize DataFrame to store the results
news_df = pd.DataFrame(columns=["Date", "Headline"])

# Get news for Apple for the last 10 years (daily summaries)
symbol = 'AAPL'

# Date range setup
start_date = datetime.date(2023, 11, 13)
end_date = datetime.date.today()

current_date = start_date

while current_date <= end_date:
    # Format the current date in 'YYYY-MM-DD' string format
    formatted_date = current_date.strftime('%Y-%m-%d')

    # Define API request parameters
    params = {
        'symbol': symbol,
        'from': formatted_date,  # Date format changed to string 'YYYY-MM-DD'
        'to': formatted_date,    # Date format changed to string 'YYYY-MM-DD'
        'token': api_key
    }

    try:
        # Make API request for news articles
        response = requests.get(BASE_URL, params=params)
        news_data = response.json()

        # Print response to check if it's valid
        #print(f"Response for {formatted_date}: {news_data}")

        # Check if there are news articles for the given date
        if isinstance(news_data, list) and news_data:
            # Extract only headlines for the day
            headlines = [article['headline'] for article in news_data]

            # Join all headlines into one string (if there are multiple)
            headlines_content = " | ".join(headlines)

            # Append results to the DataFrame
            new_row = pd.DataFrame({"Date": [formatted_date], "Headline": [head
            news_df = pd.concat([news_df, new_row], ignore_index=True)
        else:
            # If no news, add a default entry
            new_row = pd.DataFrame({"Date": [formatted_date], "Headline": ["No
            news_df = pd.concat([news_df, new_row], ignore_index=True)

        #print(f"Processed date: {formatted_date}")

    except Exception as e:
        print(f"Error on {formatted_date}: {e}")

    # Move to next day
    current_date += datetime.timedelta(days=1)
    # Pause to avoid hitting rate limits
    sleep(1)
```

```

# Display the DataFrame
print(news_df)
### Getting the Macro Indicators

merged_df = pd.DataFrame()
# Define API Keys
FRED_API_KEY = '321d7bf2c5feafe84bc8c7b84a252cdc'  # Replace with your FRED API

## Define the time range (from a year ago today to today)
end_date = datetime.date.today()
start_date = end_date - datetime.timedelta(days=365)

# Format dates as strings in 'YYYY-MM-DD'
start_date_str = start_date.strftime('%Y-%m-%d')
end_date_str = end_date.strftime('%Y-%m-%d')

# Function to fetch data from FRED
def fetch_fred_data(series_id):
    url = f'https://api.stlouisfed.org/fred/series/observations'
    params = {
        'series_id': series_id,
        'api_key': FRED_API_KEY,
        'file_type': 'json',
        'observation_start': start_date_str,
        'observation_end': end_date_str
    }
    response = requests.get(url, params=params)
    data = response.json()
    return data['observations']

# Fetch Interest Rate (Federal Funds Rate)
interest_rate_data = fetch_fred_data('FEDFUNDS')

```

```
# Fetch Inflation Rate (Consumer Price Index, CPI)
inflation_rate_data = fetch_fred_data('CPIAUCSL')

# Fetch Unemployment Rate
unemployment_rate_data = fetch_fred_data('UNRATE')

unemployment_rate_df = pd.DataFrame(unemployment_rate_data).rename(columns={'val
interest_rate_df = pd.DataFrame(interest_rate_data).rename(columns={'value': 'I
inflation_rate_df = pd.DataFrame(inflation_rate_data).rename(columns={'value': '

news_df.rename(columns={'Date': 'date'}, inplace=True)
merged_df = news_df.merge(interest_rate_df, on='date', how='left').merge(inflati

merged_df.fillna(method='ffill', inplace=True)
## forward filling economic indicators for the entire month
merged_df.fillna(method='ffill', inplace=True)
merged_df.dropna(inplace=True)

## Using the Sentiment Analysis model to extract daily sentiment for AAPL

from contextlib import redirect_stdout
import io
# Function to get the sentiment of the headline using your ArticleAnalysis model
def get_sentiment_of_headline(headline_text):
    # Create an in-memory string buffer
    buf = io.StringIO()

    # Use redirect_stdout to suppress any prints during the sentiment analysis
    with redirect_stdout(buf):
        # Perform sentiment analysis using your model
        article = ArticleAnalysis(title="Apple Article", text=headline_text)
        sentiment_results = article.aggregate_sentiment()

    # Extract the sentiment from the results
    positive = sentiment_results['positive']
    neutral = sentiment_results['neutral']
    negative = sentiment_results['negative']

    # Calculate total sentiment count
    total_sentiments = positive + neutral + negative

    # Calculate the proportions of each sentiment type
    if total_sentiments > 0:
        positive_proportion = positive / total_sentiments
        neutral_proportion = neutral / total_sentiments
        negative_proportion = negative / total_sentiments
    else:
        # Avoid division by zero if no sentiment terms are found
        positive_proportion = neutral_proportion = negative_proportion = 0

    # Determine the most common sentiment based on the highest count
    if positive >= max(neutral, negative):
        most_common_sentiment = 'positive'
    elif negative > max(positive, neutral):
        most_common_sentiment = 'negative'
    else:
        most_common_sentiment = 'neutral'

    # Return the most common sentiment and the proportions
    return most_common_sentiment, positive_proportion, neutral_proportion, nega

# Apply the sentiment analysis to each headline and create a new 'Headline_Senti
# Also, create new columns for sentiment proportions
merged_df[['Headline_Sentiment', 'Positive_Proportion', 'Neutral_Proportion', '
    lambda x: pd.Series(get_sentiment_of_headline(x))
)
merged_df = merged_df.dropna()
merged_df.head()
```

```
### Getting AAPL stock data for the past year

import yfinance as yf
import pandas as pd

# Define the ticker symbol
ticker = 'AAPL'

# Get data on this ticker
stock = yf.Ticker(ticker)

# Get the historical prices for the past year
hist_appl = stock.history(period="1y")
hist_appl = hist_appl.drop(columns=['Dividends', 'Stock Splits', 'Open', 'High',
hist_appl['Date'] = hist_appl.index
hist_appl.reset_index(drop=True, inplace=True)
hist_appl['Return'] = hist_appl['Close'].pct_change()
hist_appl = hist_appl.dropna()
hist_appl.rename(columns = {"Date":"date"}, inplace = True)
hist_appl['date'] = hist_appl['date'].dt.strftime('%Y-%m-%d')
hist_appl.head()

merged_df = merged_df.merge(hist_appl, on='date', how='left')
merged_df = merged_df.dropna()
merged_df.head()

### changing from strings to numeric

merged_df['Interest Rate'] = pd.to_numeric(merged_df['Interest Rate'], errors='c
merged_df['Inflation Rate'] = pd.to_numeric(merged_df['Inflation Rate'], errors=
merged_df['Unemployment Rate'] = pd.to_numeric(merged_df['Unemployment Rate'], e

merged_df[['Interest Rate', 'Inflation Rate', 'Unemployment Rate']].describe()

```

```
import seaborn as sns
import matplotlib.pyplot as plt

test_returns = merged_df['Return']
test_dates = merged_df['date']

# Define threshold values for sentiment-based trading signals
thresholds = {
    'Positive_Proportion': 0.6,  # Buy when positive sentiment > 60%
    'Negative_Proportion': 0.2   # Sell when negative sentiment > 20%
}

# Function to generate trading signals based on sentiment thresholds
def generate_signals(row, thresholds):
    signal = 0  # Default no trade (0)

    # Buy signal (1) when positive sentiment is higher than the threshold
    if row['Positive_Proportion'] > thresholds['Positive_Proportion']:
        signal = 1  # Buy

    # Sell signal (-1) when negative sentiment is higher than the threshold
    elif row['Negative_Proportion'] > thresholds['Negative_Proportion']:
        signal = -1  # Sell

    return signal

# Apply signal generation function to each row of the data
merged_df['Signal'] = merged_df.apply(lambda row: generate_signals(row, threshol

# Calculate the strategy returns based on the generated signals
shifted_returns = test_returns#.shift(1)  # Use previous day's return for today'
strategy_returns = merged_df['Signal'].shift(1) * shifted_returns  # Trading sig
strategy_cumulative_returns = (1 + strategy_returns).cumprod()

```

```
# Plot the cumulative returns of the strategy
plt.figure(figsize=(10, 6))
sns.lineplot(x=pd.to_datetime(test_dates), y=strategy_cumulative_returns)
plt.title('Cumulative Returns of Sentiment Strategy (Using Thresholds)')
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
## This incorporates a more complicated strategy, including some Macro-economic
## I assumed some arbitrary thresholds where we might assume AAPL stock will de
import numpy as np
# Mean and Standard Deviation values from describe() output for thresholds
mean_interest_rate = 5.25
std_interest_rate = 0.17

mean_inflation_rate = 312.79
std_inflation_rate = 2.00

mean_unemployment_rate = 3.99
std_unemployment_rate = 0.19

# Define the thresholds using mean   std logic for economic indicators
thresholds = {
    'Positive_Proportion': 0.6,  # Buy when positive sentiment > 60%
    'Negative_Proportion': 0.2,  # Sell when negative sentiment > 20%
    'Inflation Rate': mean_inflation_rate + std_inflation_rate,  # Sell when in
    'Interest Rate': mean_interest_rate + std_interest_rate,  # Sell when intere
    'Unemployment Rate': mean_unemployment_rate - std_unemployment_rate
# Buy when unemployment rate is below mean - std
}


# Function to generate trading signals based on thresholds
```

```
def generate_advanced_signals(row, thresholds):
    signal1 = 0  # Default no trade (0)
    signal2 = 0

    # Buy signal (1) when conditions are met (positive sentiment > 60%)
    if row['Positive_Proportion'] > thresholds['Positive_Proportion']:
        signal1 = 1  # Buy

    # Sell signal (-1) when conditions are met (negative sentiment > 20%)
    elif row['Negative_Proportion'] > thresholds['Negative_Proportion']:
        signal1 = -1  # Sell

    # Additional signals based on economic indicators
    if row['Inflation Rate'] > thresholds['Inflation Rate']:
        signal2 = -1  # Sell if inflation rate is higher than the threshold
    elif row['Interest Rate'] > thresholds['Interest Rate']:
        signal2 = -1  # Sell if interest rate is higher than the threshold
    elif row['Unemployment Rate'] < thresholds['Unemployment Rate']:
        signal2 = 1  # Buy if unemployment rate is below the threshold
    # Combine the two signals
    combined_signal = signal1 + signal2

    return combined_signal

# Apply signal generation function to each row of the data
merged_df['Signal'] = merged_df.apply(lambda row: generate_advanced_signals(row,

# Calculate the strategy returns based on the generated signals
shifted_returns = test_returns#.shift(1)  # Use previous day's return for today'
strategy_returns = merged_df['Signal'].shift(1) * shifted_returns  # Trading sig
strategy_cumulative_returns = (1 + strategy_returns).cumprod()

# Plot the cumulative returns of the strategy
plt.figure(figsize=(10, 6))
```

```
sns.lineplot(x=pd.to_datetime(test_dates), y=strategy_cumulative_returns)
plt.title('Cumulative Returns of Sentiment + Macro Strategy (Using Thresholds)')
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```