# Stock Sentiment Analysis Using Machine Learning Techniques
# AKSHAT SHAW - 22118006

**Finance Club Open Project Summer 2024**
GitHub

## 1 Introduction

This project involves analyzing the sentiment of news headlines related to specific stocks and simulating trading strategies based on these sentiments. The goal is to determine whether sentiment analysis can provide actionable trading signals to generate profitable trades. The report outlines the process of data collection, sentiment analysis, trading signal generation, trade simulation, and performance evaluation.

## 2 Data Collection

### 2.1 Stock Data

Stock price data is fetched using the `yfinance` library. The data includes daily closing prices for the selected stock over a specified period. The `yfinance` library provides a convenient way to download historical stock prices.

```
    import yfinance as yf
    def fetch_stock_data(ticker):
    print("Fetching stock prices...")
    stock_data = yf.download(ticker, period="max")
    stock_data = stock_data[['Close']]
    return stock_data
```

### 2.2 News Data

News headlines are scraped from the Business Insider website. The headlines are relevant to the stock being analyzed and provide the necessary context for sentiment analysis. API from New york Times can also be used but it has limitation for only limited period of time. Here we are using requests and BeautifulSoup library to scrape the web-pages.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
from tqdm.notebook import tqdm

def fetch_news(ticker):
    columns = ['datetime', 'ticker', 'source', 'headline']
    df = pd.DataFrame(columns=columns)
    ticker = str(ticker).lower()
    print("Fetching news headlines...")
    for page in tqdm(range(1, 200)):
        url = f'https://markets.businessinsider.com/news/{ticker}-stock?p={page}'
        response = requests.get(url)
        html = response.text
        soup = BeautifulSoup(html, 'lxml')
        articles = soup.find_all('div', class_='latest-news__story')
        for article in articles:
            datetime = article.find('time', class_='latest-news__date').get('datetime'
                                                )
            title = article.find('a', class_='news-link').text
            source = article.find('span', class_='latest-news__source').text
            df = pd.concat([pd.DataFrame([[datetime, ticker, source, title]], columns=
                                                df.columns), df], ignore_index=
                                                True)
    df['datetime'] = pd.to_datetime(df['datetime'])
    df['date'] = df['datetime'].dt.date
    df['time'] = df['datetime'].dt.time
    df.drop(columns=['datetime'], inplace=True)
    return df
```

## 3 Sentiment Analysis

Sentiment Analysis for the news headlines falls under the domain of Natural Language Processing (NLP). Several methods can be employed, such as converting text to embeddings using Word2vec, TF-IDF, One-Hot Encoding, etc. These embeddings can then be used to train a supervised model based on the embedding and price movement of the stock.
Other method that I have used here is to use a pre-trained model and library that can used to find out the Sentiment positive, negative or neutral which then trigger the Trade algorithm.
Also we can use BERT based pre-trained models based on transformer architecture which is great for such classification tasks.
As we are moving toward the GEN-AI era, we can also impliment fine-tuning Large Language Model (LLM), which is again a very great use of GEN-AI for builiding trading algorithms.

### 3.1 Word2Vec

Here is the implementation of Word2vec embedding using pre-trained vectors, through which we are converting text to vector representation to train the model for sentiment analysis after which we can implement the trading algorithm. Also we could have used the Tf-Idf and other methods for the embedding.

```
# using word2vec embedding model from google news data
import gensim.downloader
glove_vectors = gensim.downloader.load('word2vec-google-news-300')
# getting word vectors and taking mean of it.
def get_average_word2vec(tokens, model):
valid_tokens = [token for token in tokens if token in model]
if not valid_tokens:
    return np.zeros(model.vector_size)
vectors = [model[token] for token in valid_tokens]
return np.mean(vectors, axis=0)
#apply the function to the 'tokens' column to get the embeddings
news['embedding']=news['text'].apply(lambda x: get_average_word2vec(x, glove_vectors))
#convert embeddings to separate columns
embedding_df = pd.DataFrame(news['embedding'].to_list(), index=news.index)
embedding_df.columns = [f'embedding_{i}' for i in range(embedding_df.shape[1])]
```

### 3.2 VaderSentiment Library

The `vaderSentiment` library is used for sentiment analysis. This library provides a compound score representing the overall sentiment of a headline.

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer

nltk.download('vader_lexicon')
analyzer = SentimentIntensityAnalyzer()

def calculate_sentiment_scores(headlines_df):
    headlines_df['sentiment'] = headlines_df['headline'].apply(lambda x: analyzer.
                                                polarity_scores(x)['compound'])
    headlines_df['date'] = pd.to_datetime(headlines_df['date'])
    return headlines_df
```

### 3.3 BERT-based pre-trained model

The following code uses the BERT based RoBERTa model which is pre-trained on the twitter data for sentiment analysis.

```
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax
```

```
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)

def polarity_scores_roberta(example):
    encoded_text = tokenizer(example, return_tensors='pt')
    output = model(**encoded_text)
    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
    scores_dict = {
        'roberta_neg' : scores[0],
        'roberta_neu' : scores[1],
        'roberta_pos' : scores[2]
    }
return scores_dict
```

### 3.4 Fine-Tuning LLM

A LLM can be fine-tuned on the sentiment dataset publicly available to generate responses based on the News Headline for furture tasks. However use for this project I have implimented fine-tuning in a sepearte file.

## 4 Trading Signal Generation

Sentiment scores are aggregated by date, and trading signals are generated based on the aggregated sentiment scores. The sentiment score threshold can be tuned accordingly to obtain a better result.

```
def aggregate_sentiment_scores(headlines_df):
    sentiment_summary = headlines_df.groupby('date')['sentiment'].mean()
    return sentiment_summary

def generate_trading_signals(sentiment_summary):
    signals = sentiment_summary.apply(lambda x: 1 if x > 0.2 else (-1 if x < -0.2 else
                                                 0))
    return signals
```

## 5 Trade Simulation

Trades are simulated based on the generated trading signals. A simple strategy of buying and selling based on the signals is implemented. This is the main Trading Logic that triggers the BUY/SELL options for a stock based on the current sentiment and SELL on if there is profit.

```
def simulate_trades(stock_data, trading_signals,intial_capital):
    portfolio = []
    position = 0
    buy_price = 0
    quantity =0
    for date, price in (stock_data['Close'].items()):
        if date in trading_signals.index:
            signal = trading_signals.loc[date]
            if signal == 1 and position == 0:  # Buy signal
                position = 1
                buy_price = price
                quantity = intial_capital/price
                intial_capital = intial_capital%price
                portfolio.append({"date": date, "type": "buy", "price": buy_price, "
                                                          capital":intial_capital})
            elif signal == -1 and position == 1 and (price>buy_price) :  # Sell signal
                position = 0
                sell_price = price
                profit = (sell_price - buy_price)*quantity
                intial_capital = intial_capital+ quantity*sell_price
                portfolio.append({"date": date, "type": "sell", "price": sell_price,"
                                                    capital":intial_capital, "
                                                    profit": profit})
    return pd.DataFrame(portfolio)
```

## 6 Performance Metrics

### 6.1 Portfolio Metrics

The number of trades, win percentage, and total profit are calculated to evaluate the performance of the trading strategy.

```
def calculate_portfolio_metrics(portfolio):
    total_trades = len(portfolio) // 2
    wins = portfolio[portfolio['type'] == 'sell']['profit'] > 0
    win_percentage = wins.mean() * 100
    total_profit = portfolio[portfolio['type'] == 'sell']['profit'].sum()
    return total_trades, win_percentage, total_profit
```

### 6.2 Sharpe Ratio and Maximum Drawdown

The Sharpe Ratio and Maximum Drawdown are calculated to provide additional insights into the risk-adjusted performance and downside risk of the trading strategy.

```
def calculate_sharpe_ratio(portfolio, risk_free_rate=0.01):
    daily_returns = portfolio[portfolio['type'] == 'sell']['profit']
    excess_returns = daily_returns - risk_free_rate
    sharpe_ratio = excess_returns.mean() / excess_returns.std()
    return sharpe_ratio

def calculate_max_drawdown(portfolio):
    portfolio['cumulative_profit'] = portfolio['profit'].cumsum()
    cumulative_max = portfolio['cumulative_profit'].cummax()
    drawdown = portfolio['cumulative_profit'] - cumulative_max
    max_drawdown = drawdown.min()
    return max_drawdown
```

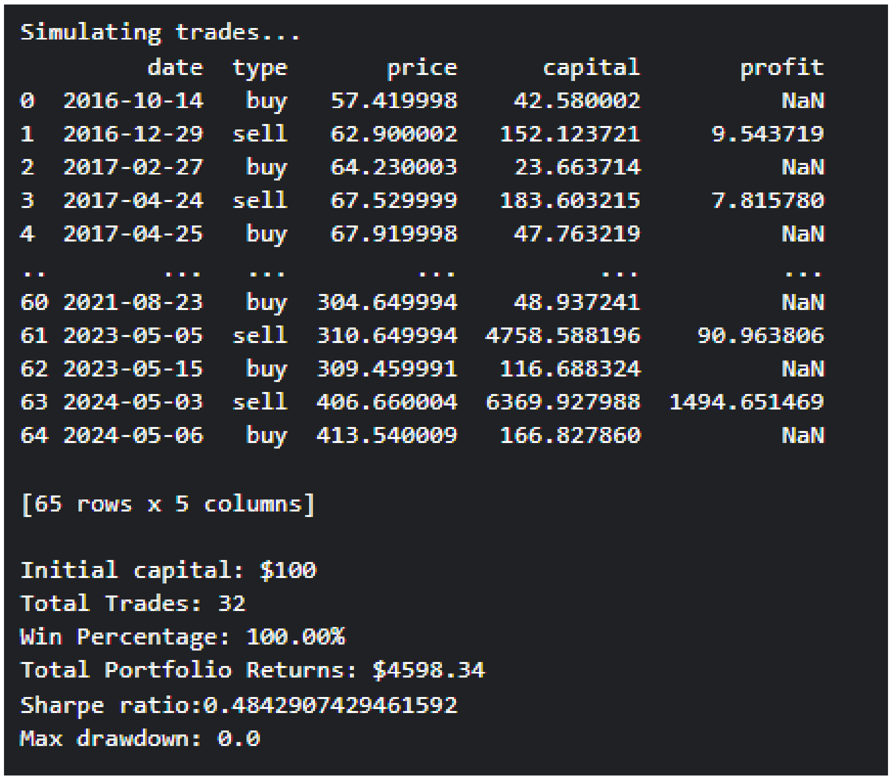
## 7 Visualization

The `plotly` library is used to create an interactive plot of the stock price with buy and sell signals.

```
import plotly.graph_objects as go

def plot_signals(stock_data, portfolio):
    start_date = portfolio['date'].min()
    end_date = portfolio['date'].max()
    # Filter stock data to the date range available in the portfolio
    trimmed_stock_data = stock_data[(stock_data.index >= start_date) & (stock_data.
                                            index <= end_date)]

    buy_signals = portfolio[portfolio['type'] == 'buy']
    sell_signals = portfolio[portfolio['type'] == 'sell']
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=trimmed_stock_data.index,
        y=trimmed_stock_data['Close'],
        mode='lines',
        name='Stock Price',
        line=dict(color='blue')
    ))
    #buy signals
    fig.add_trace(go.Scatter(
        x=buy_signals['date'],
        y=buy_signals['price'],
```



```
Simulating trades...
          date  type       price      capital        profit
0   2016-10-14   buy   57.419998    42.580002           NaN
1   2016-12-29  sell   62.900002   152.123721      9.543719
2   2017-02-27   buy   64.230003    23.663714           NaN
3   2017-04-24  sell   67.529999   183.603215      7.815780
4   2017-04-25   buy   67.919998    47.763219           NaN
..         ...   ...         ...          ...           ...
60  2021-08-23   buy  304.649994    48.937241           NaN
61  2023-05-05  sell  310.649994  4758.588196     90.963806
62  2023-05-15   buy  309.459991   116.688324           NaN
63  2024-05-03  sell  406.660004  6369.927988   1494.651469
64  2024-05-06   buy  413.540009   166.827860           NaN

[65 rows x 5 columns]

Initial capital: $100
Total Trades: 32
Win Percentage: 100.00%
Total Portfolio Returns: $4598.34
Sharpe ratio:0.4842907429461592
Max drawdown: 0.0
```

Figure 1: Portfolio for Microsoft stocks with initial capital of 100 dollor.

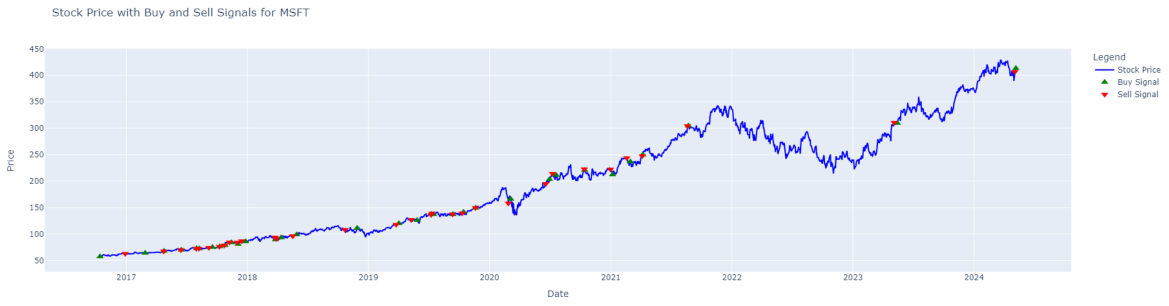


Figure 2: BUY/SELL signals based on the sentiment

```
        mode='markers',
        name='Buy Signal',
        marker=dict(symbol='triangle-up', color='green', size=10)
    ))
    #sell signals
    fig.add_trace(go.Scatter(
        x=sell_signals['date'],
        y=sell_signals['price'],
        mode='markers',
        name='Sell Signal',
        marker=dict(symbol='triangle-down', color='red', size=10)
    ))
    fig.update_layout(
        title='Stock Price with Buy and Sell Signals',
        xaxis_title='Date',
        yaxis_title='Price',
        legend_title='Legend',
        hovermode='x'
    )
    fig.show()
```

## 8 Conclusion

The project demonstrates that sentiment analysis of news headlines can be used to generate trading signals. A trading algorithm thus can be designed to respond to the trading signals generated by the sentiment analysis. Thus in this project i have demonstrated the use of machine learning methods applied in the financial domain for the buying and selling of stocks and portfolio management. The use of NLP techniques for the sentiment analysis of the news headlines, thus generating trading signals is sucssefully impliment in the code snippets given above.

## 9 Future Work

Future improvements could include:

- Using more sophisticated sentiment analysis techniques, such as advanced transformer models and fine-tuning large language models.
- Incorporating additional data sources for sentiment analysis, including social media, financial reports, and expert opinions.
- Testing the strategy on different stocks and time periods to assess its robustness and generalizability.
- Implementing more advanced trading strategies and risk management techniques, such as algorithmic trading, portfolio diversification, and dynamic stop-loss orders.
- Enhancing the visualization of trading signals and portfolio performance with more interactive and informative charts.
- By addressing these areas, the project can be expanded to provide a more comprehensive and effective trading strategy based on sentiment analysis.