Appendix A

Extra Information

A.1 Final Generated Graphs

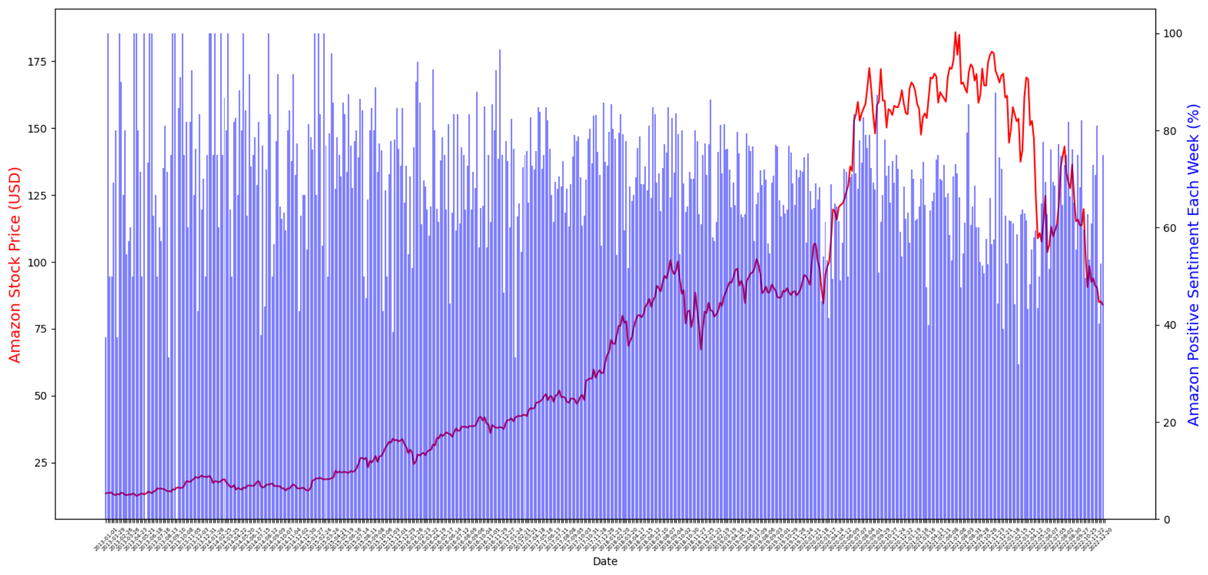

Figure A.1: AMZN price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

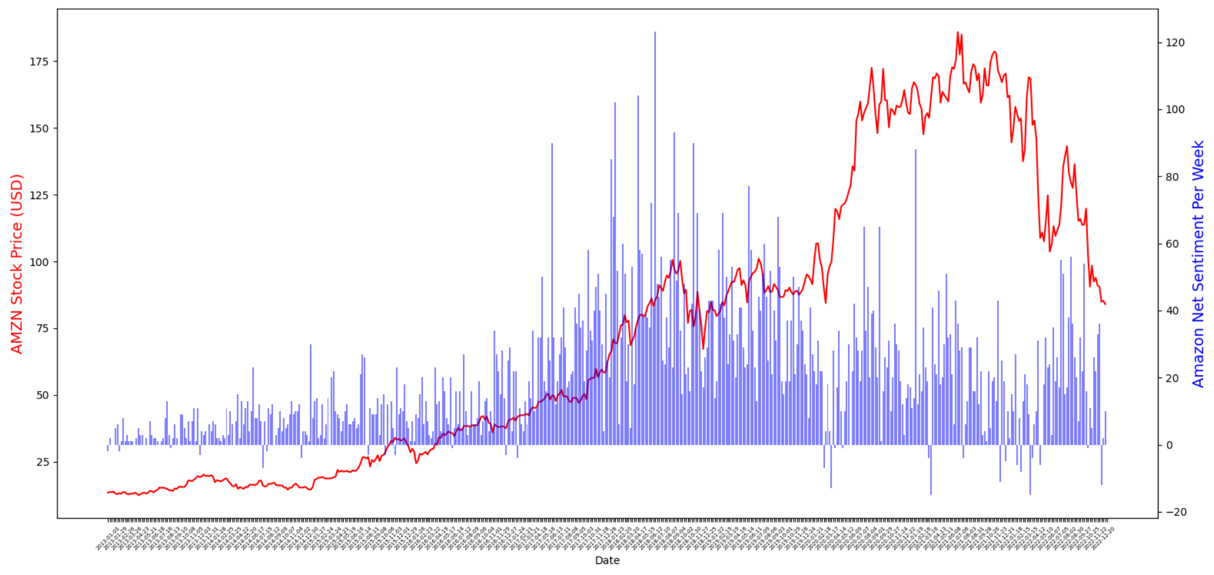

Figure A.2: AMZN price against net sentiment for that week, with periods leading up to sharp drops highlighted.

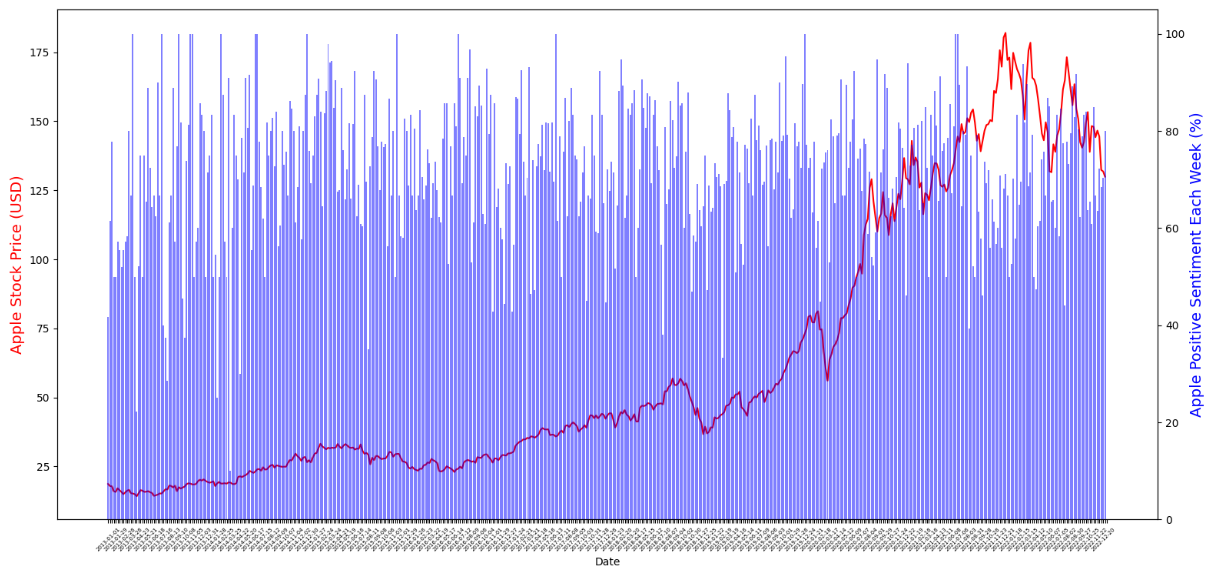

Figure A.3: APPL price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

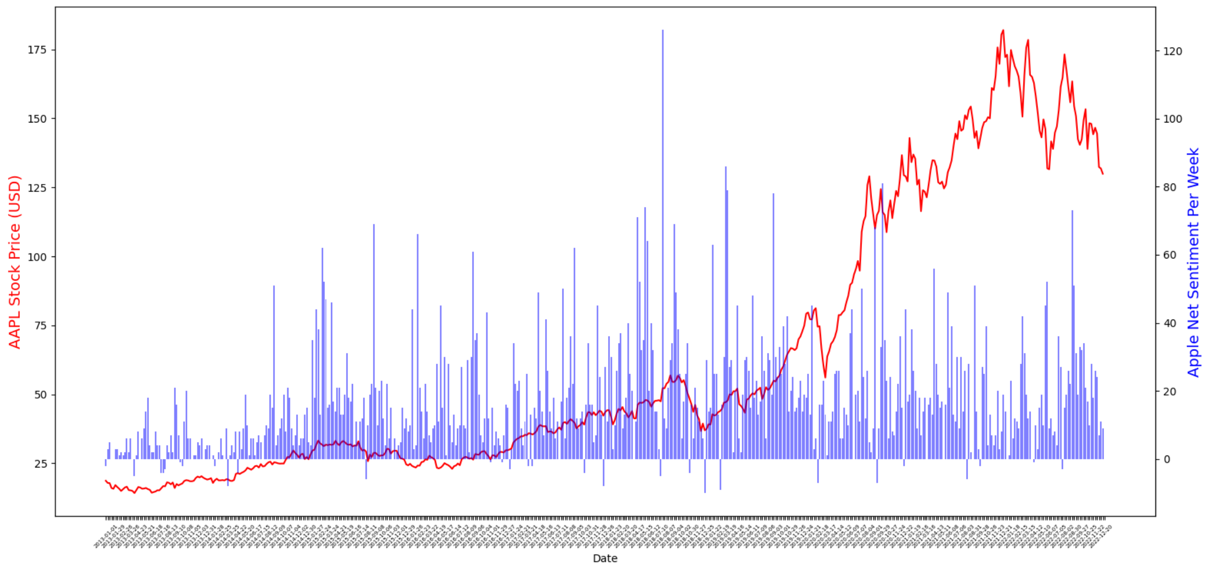

Figure A.4: APPL price against net sentiment for that week, with periods leading up to sharp drops highlighted.

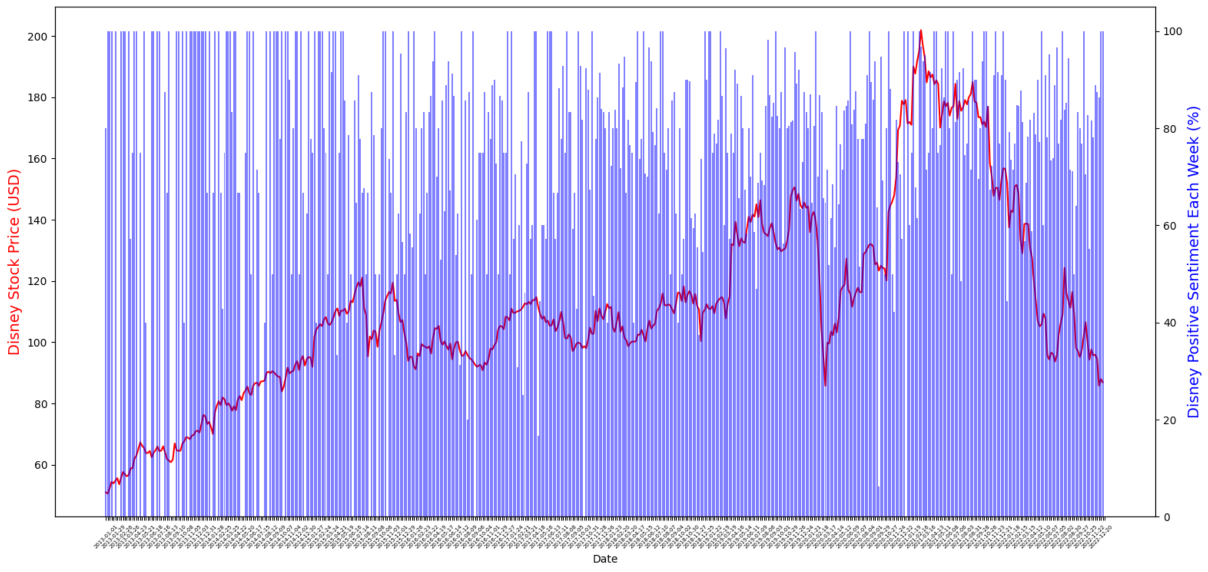

Figure A.5: DIS price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

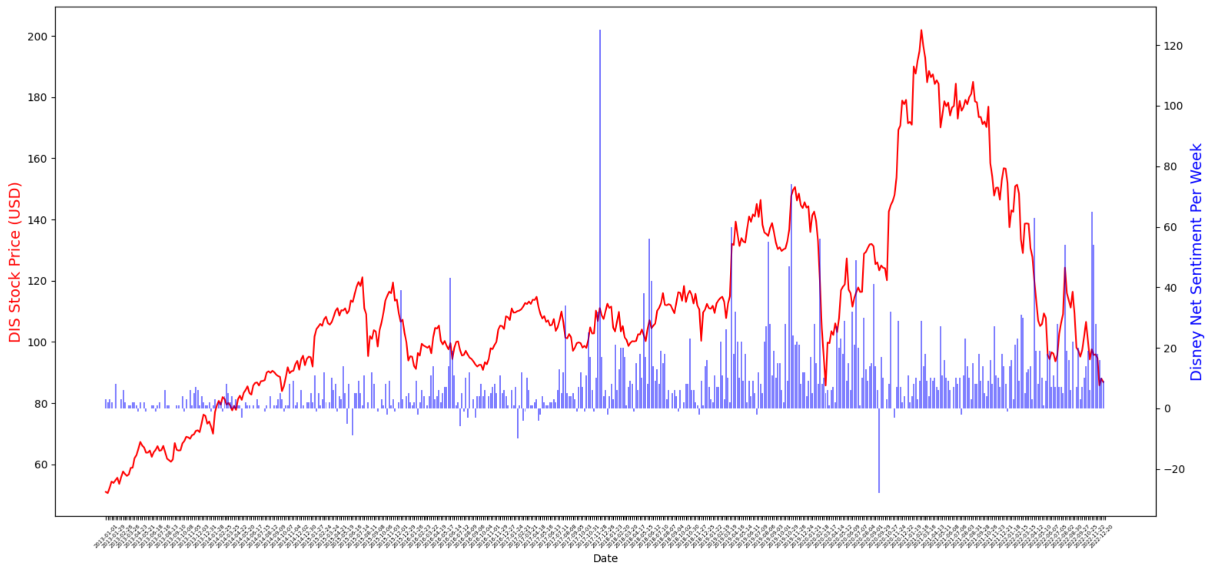

Figure A.6: DIS price against net sentiment for that week, with periods leading up to sharp drops highlighted.

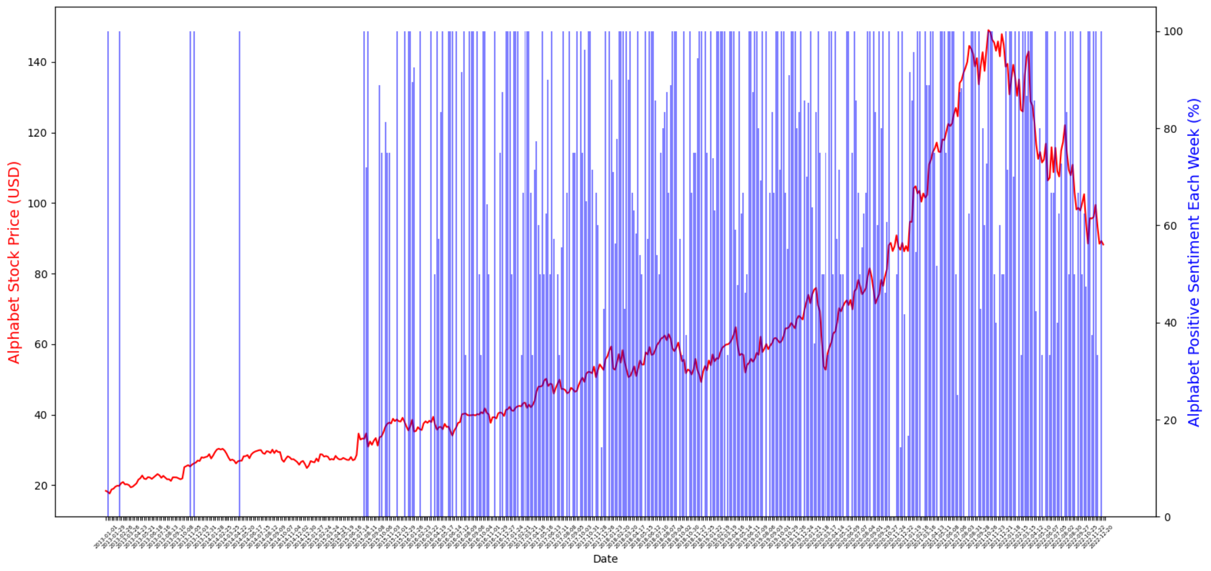

Figure A.7: GOOGL price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

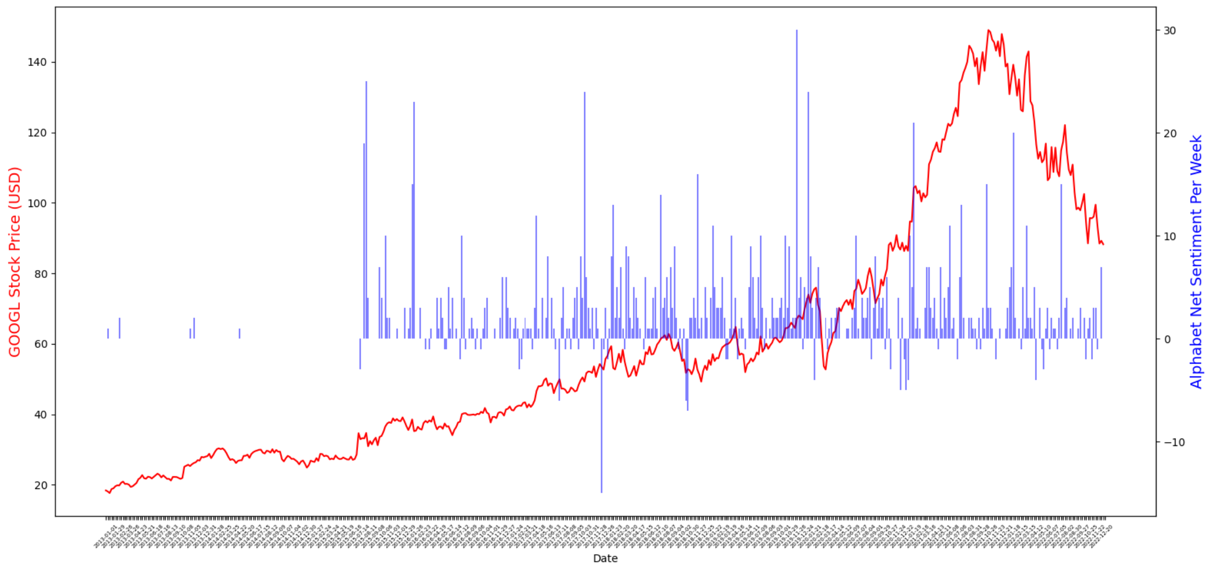

Figure A.8: GOOGL price against net sentiment for that week, with periods leading up to sharp drops highlighted.

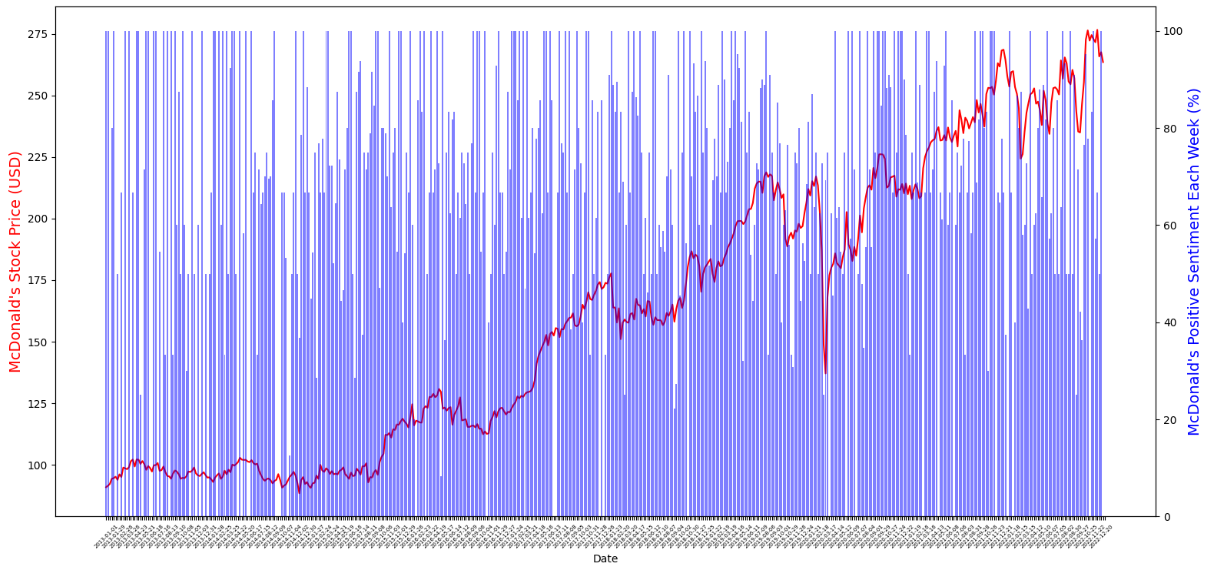

Figure A.9: MCD price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

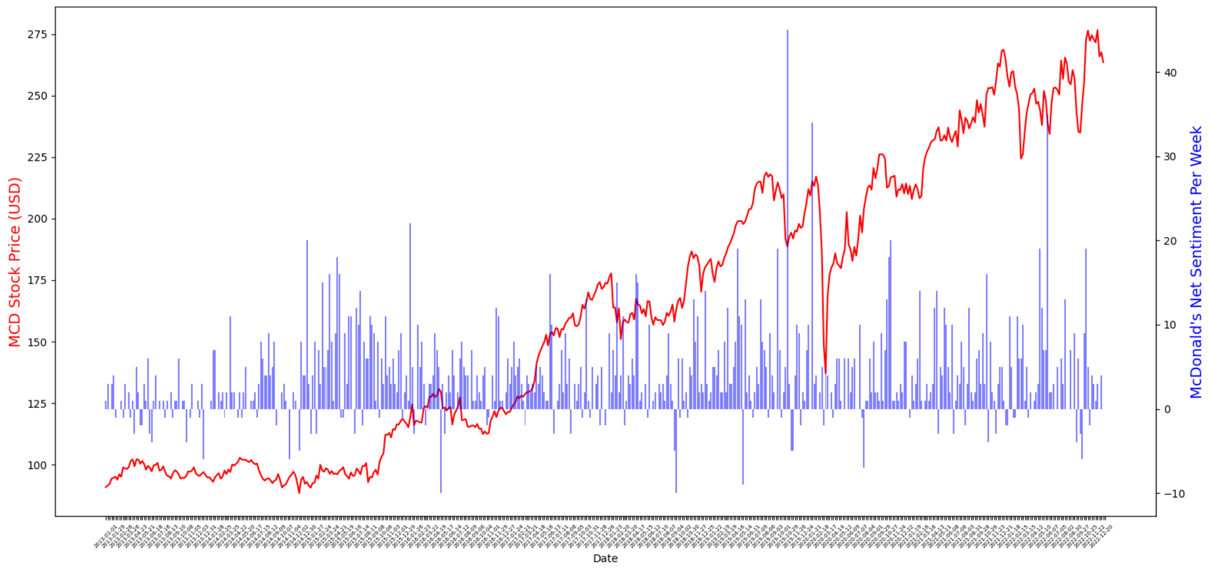

Figure A.10: MCD price against net sentiment for that week, with periods leading up to sharp drops highlighted.

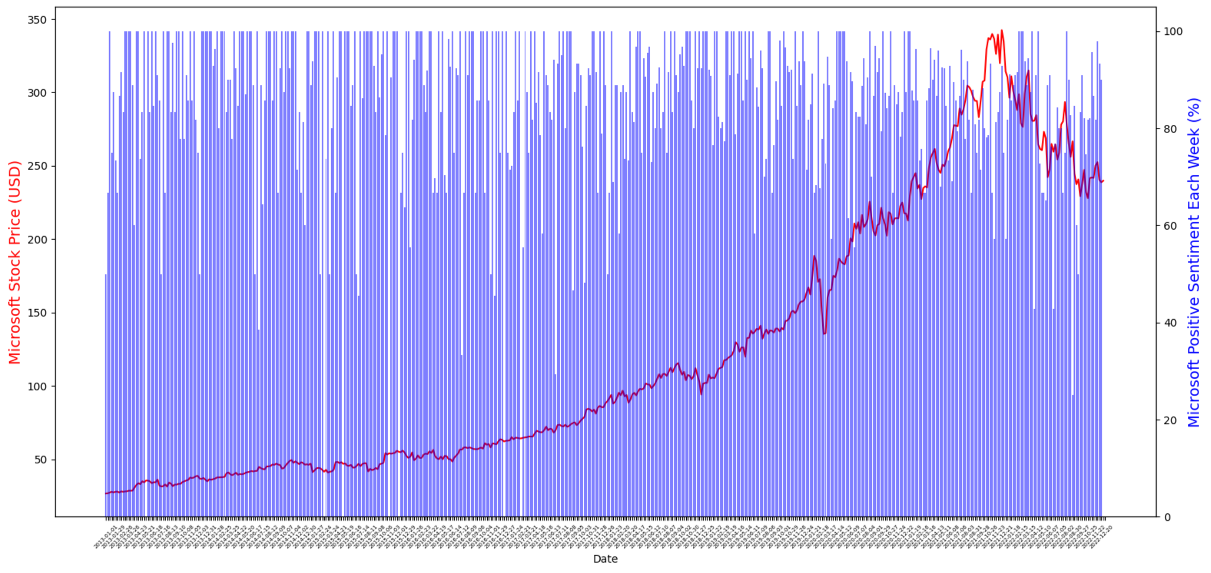

Figure A.11: MSFT price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

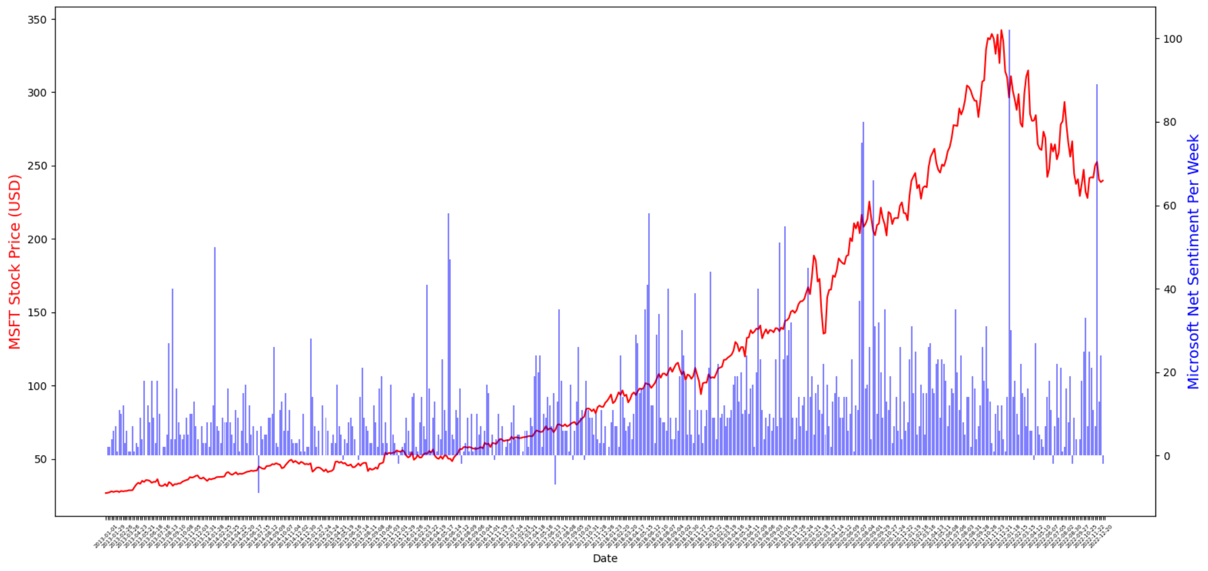

Figure A.12: MSFT price against net sentiment for that week, with periods leading up to sharp drops highlighted.

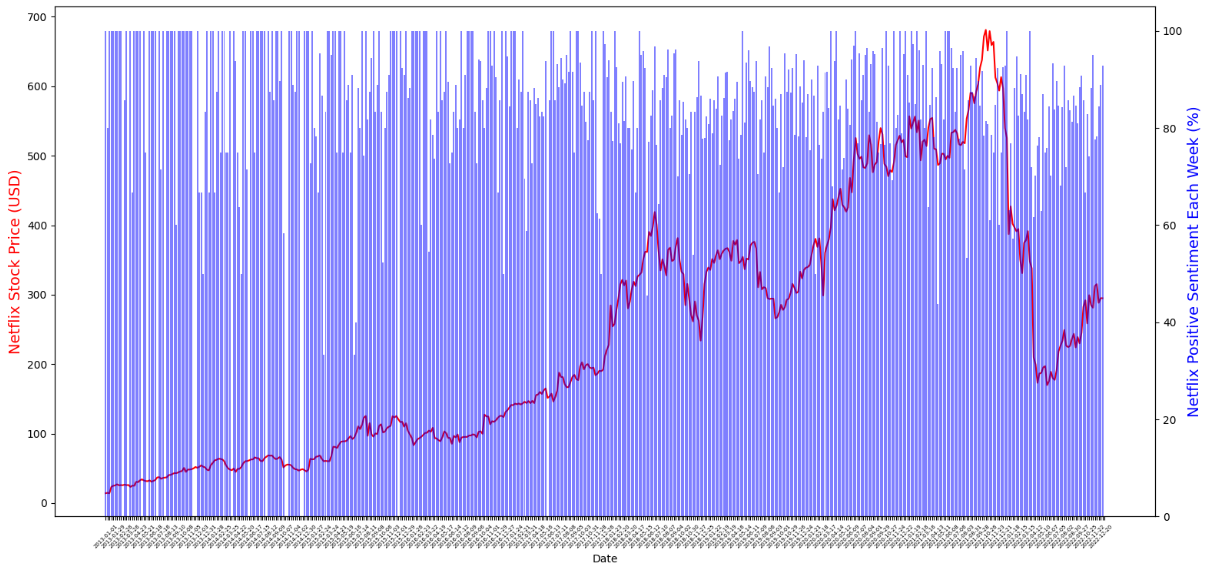

Figure A.13: NFLX price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

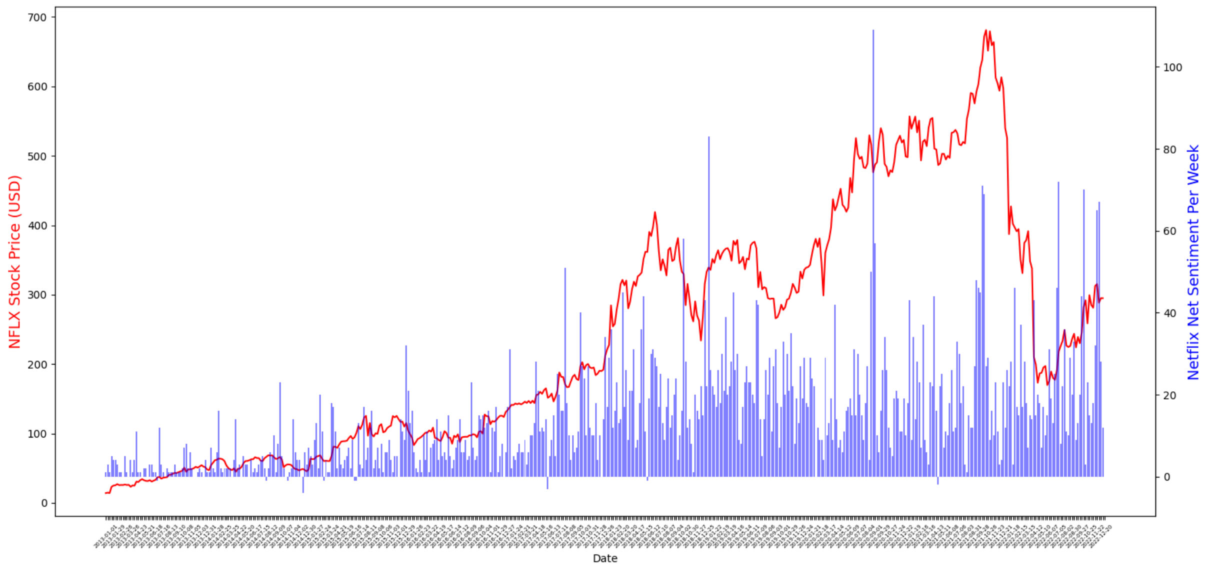

Figure A.14: NFLX price against net sentiment for that week, with periods leading up to sharp drops highlighted.

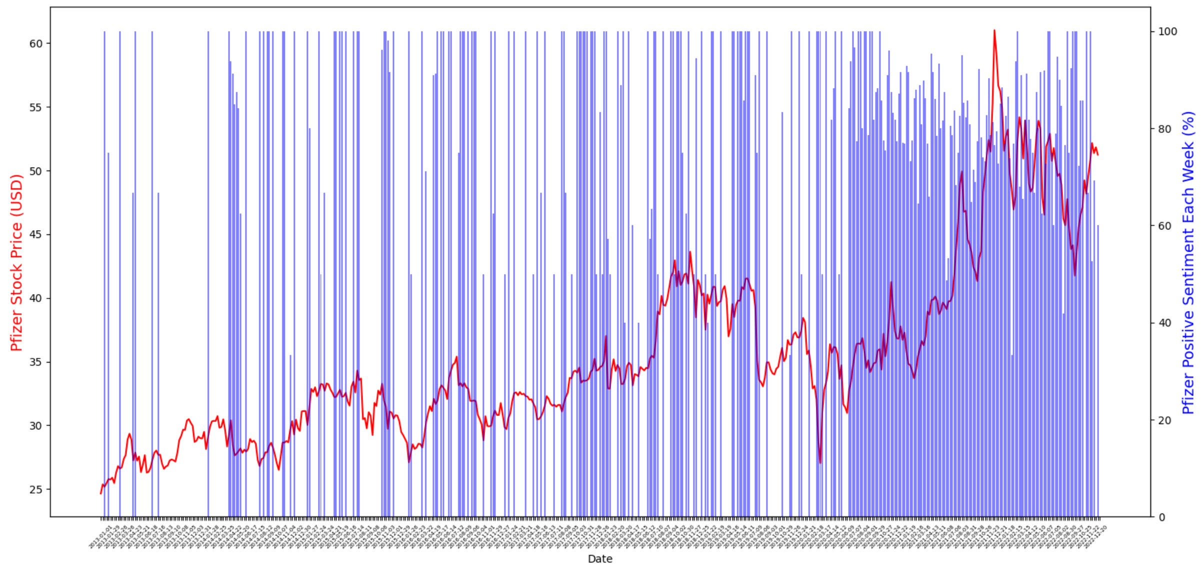

Figure A.15: PFE price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

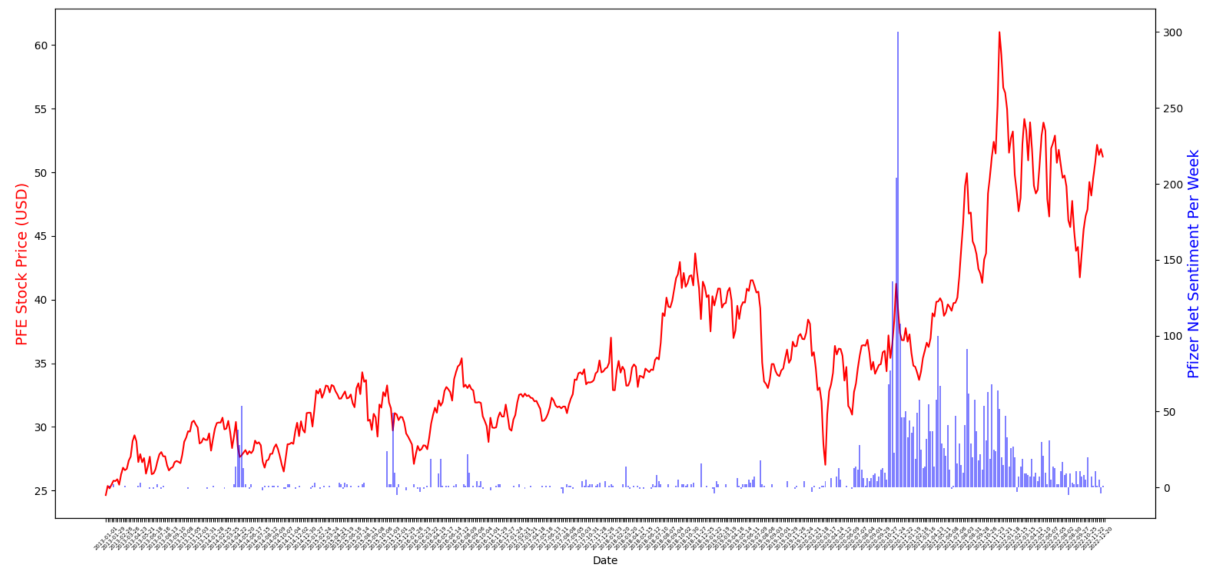

Figure A.16: PFE price against net sentiment for that week, with periods leading up to sharp drops highlighted.

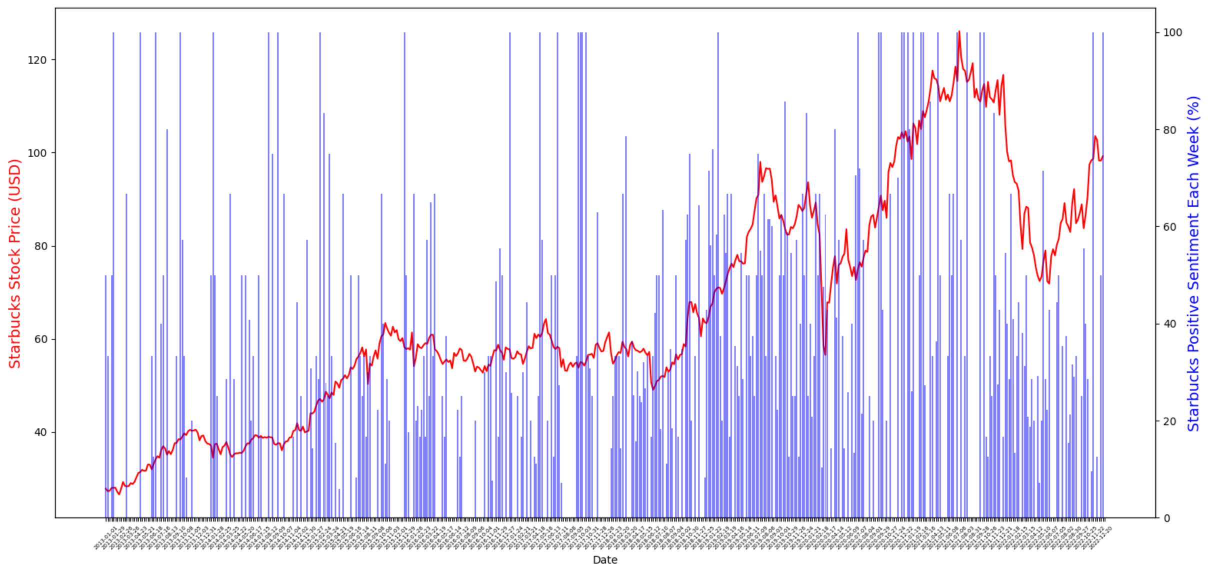

Figure A.17: SBUX price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

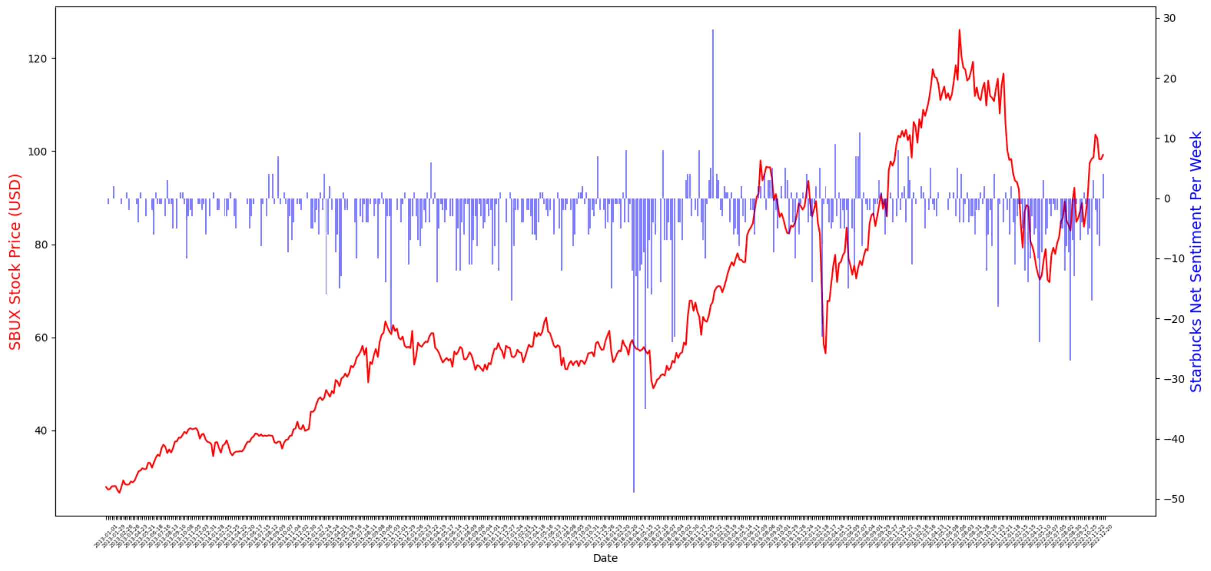

Figure A.18: SBUX price against net sentiment for that week, with periods leading up to sharp drops highlighted.

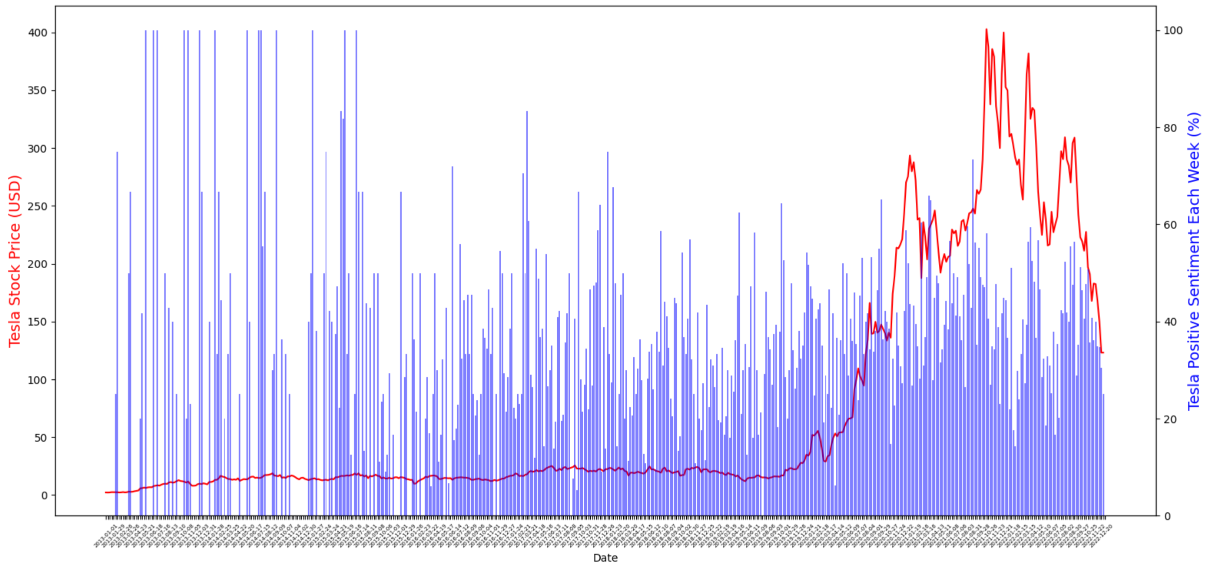

Figure A.19: TSLA price against positive percentage sentiment for that week, with periods leading up to sharp drops highlighted.

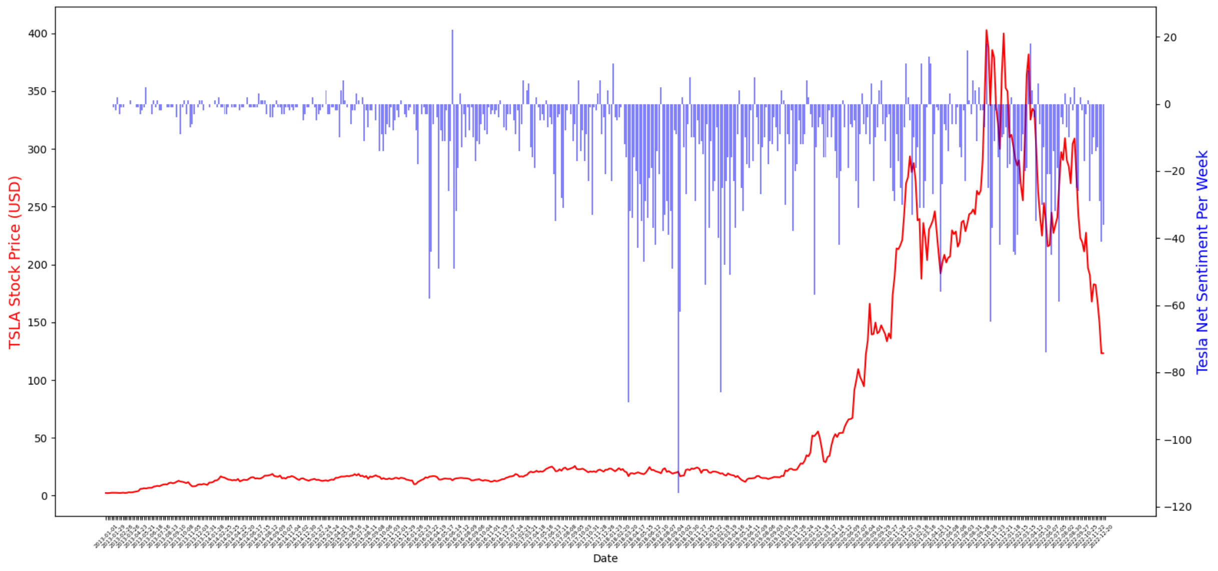

Figure A.20: TSLA price against net sentiment for that week, with periods leading up to sharp drops highlighted.

Appendix B

User Guide

B.1 Instructions

The twitter_sentiments_source folder consists of a single Python file and a collection of CSV files, each containing the stock prices of a company each week from the 1st January 2013 until the 1st January 2023 or the scraped tweets obtained by running the twitter_scrape function. In order to prevent the long wait times that come with scraping 10 years of tweets, the scraped tweets have been provided, and the code can be run straight away. These files are used by the sentiment_analysis.py file in order to graph the correlation between sentiment online towards a company and the stock price that week.

Requirements: Python - version 3 or later

Steps:

1. Navigate to the local repository using the command line.
2. Install pip using command: sudo apt install python3-pip
3. Install Natural Language ToolKit library using command: pip install nltk
4. Download stopwords from nltk using command: python3 -c ”import nltk; nltk.download(’all’)”
5. Install MatPlotLib using command: pip install matplotlib
6. Install pandas using command: pip install pandas
7. Install SNScrape using command: pip install snscrape
8. Run the following command: python3 sentiment_analysis.py

9. When asked to scrape Twitter, if changes have been made to the list of stocks then enter ”Y”, if not enter ”N”.

10. If an error occurs, this may be due to ongoing changes to Twitter’s API, in which case run the following commands: pip install –upgrade pip pip install –upgrade snscrape python3 sentiment_analysis.py

This will use the all_training_data.csv file to train the Naive Bayes classifier to analyse the sentiments of tweets, then analyse all the tweets for each company and generate graphs showing stock prices each week and corresponding sentiments.

Appendix C

Source Code

```python
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tag import pos_tag
from nltk.tokenize import word_tokenize
from nltk import classify, NaiveBayesClassifier
import re, string, random
import matplotlib.pyplot as plt
import csv
import pandas as pd
import datetime
from dateutil.relativedelta import relativedelta
from datetime import datetime
import snscrape.modules.twitter as sntwitter
import pandas as pd

# Scrape Twitter for tweets from financial news accounts related to selected companies
def twitter_scrape():
    # Create list of accounts to search
    accounts_list = ["CNBC","FT", "Reuters", "CNN", "TheEconomist", "BBCWorld","BBCNews", "BBCBreaking", "nytimes", "WSJ", "washingtonpost", "AP", "guardian", "SkyNews", "TelegraphNew
    stocks_list = ["Meta", "Netflix", "Tesla", "Amazon", "Pfizer", "Microsoft", "Disney", "Alphabet", "McDonald's", "Starbucks"]
    #Iterate across accounts listed
    for stock in stocks_list:
        print("Scraping tweets for " + stock + "...")
        tweets_list = []
        for account in accounts_list:
            # Create empty list to store scraped tweets
            # Search tweets since 2013 from the specified account whose text contains the keyword
            for i,tweet in enumerate(sntwitter.TwitterSearchScraper(stock + ' since:2013-01-01 until:2023-01-01 from:' + account).get_items()):
                if i>20000:
                    break
                # Disallow replies and retweets
                if "@" and "RT" not in tweet.rawContent:
                    # Add relevant data to list
                    tweets_list.append([tweet.user.username, tweet.date, tweet.rawContent])
        # Create dataframe from the tweets_list above
        tweets_df = pd.DataFrame(tweets_list, columns=['Username', 'Datetime', 'Text'])
        tweets_df.dropna(inplace = True)
        # Convert dataframe to CSV file with custom name
        tweets_df.to_csv(stock +'_all_tweets.csv')
        print("Scraping complete for " + stock + ".")


# Remove hyperlinks, punctuation and special characters from tokens
# Convert remaining tokens to a normal form (losing becomes lose, profits becomes profit)
# Based on the Digital Oceans implementation of the NLTK library, altered for simplicity and clarity

def remove_noise(news_tokens, stopwords = ()):
    own_cleaned_tokens = []
    for token, tag in pos_tag(news_tokens):
        token = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+#]|[!*\(\),]|''(?:%[0-9a-fA-F][0-9a-fA-F]))+''(@[A-Za-z0-9_]+)','', token)
        if tag.startswith("NN"):
            word_type = 'n'
        elif tag.startswith('VB'):
            word_type = 'v'
        else:
            word_type = 'a'
        token = WordNetLemmatizer().lemmatize(token, word_type)
        if len(token) > 0 and token not in string.punctuation and token.lower() not in stopwords:
            own_cleaned_tokens.append(token.lower())
    return own_cleaned_tokens

# Create dictionary of tokens for each headline

def get_headlines_for_model(cleaned_tokens_list):
    for news_tokens in cleaned_tokens_list:
        yield dict([token, True] for token in news_tokens)

# Scrape data from CSV file and separate by sentiment

run_scrape = input("Run Twitter scrape (Necessary only if changes have been made to the code) Y/N: ")
if run_scrape.lower() == 'y':
    run_scrape = input("This action will overwrite existing records and may take up to 30 minutes to complete. Is this completely necessary: Y/N: ")
    if run_scrape.lower() == 'y':
        twitter_scrape()

positive_headline_tokens = []
negative_headline_tokens = []
df = pd.read_csv('all_training_data.csv', encoding = "ISO-8859-1")
df.reset_index()
for index, row in df.iterrows():
    if (row[0] == "positive"):
        positive_headline_tokens.append(row[1].split())
    if (row[0] == "negative"):
        negative_headline_tokens.append(row[1].split())
own_positive_cleaned_tokens_list = []
own_negative_cleaned_tokens_list = []
stopwords = stopwords.words('english')
for tokens in positive_headline_tokens:
    own_positive_cleaned_tokens_list.append(remove_noise(tokens, stopwords))
for tokens in negative_headline_tokens:
    own_negative_cleaned_tokens_list.append(remove_noise(tokens, stopwords))
ready_positive_dataset = [(headline, "Positive")
                         for headline in get_headlines_for_model(own_positive_cleaned_tokens_list)]
ready_negative_dataset = [(headline, "Negative")
                     for headline in get_headlines_for_model(own_negative_cleaned_tokens_list)]

# Combine positive and negative sentiment data and randomly split
# Create model with training data then test

dataset = ready_positive_dataset + ready_negative_dataset
print("Dataset Length:", len(dataset))
random.shuffle(dataset)
train_data = dataset[:4100]
test_data = dataset[4100:]
classifier = NaiveBayesClassifier.train(train_data)
print("Accuracy:", classify.accuracy(classifier, test_data))
print(classifier.show_most_informative_features(25))

# Separate tweets into months and creates pairs of values
# Each month has a corresponding percentage of tweets that are positive and a net sentiment

def headline_analysis(csv_file, total_headlines_predicted, weekly_headlines_count, weekly_positive_headlines_count, weekly_negative_headlines_count):
    with open(csv_file) as file_obj:
    # Create reader object by passing the file object to reader method
            reader_obj = csv.reader(file_obj)
            for row in reader_obj:
                if "h" not in row[0]:
                    if(row[2] > current_week_str and row [2] < next_week_str):
                        weekly_headlines_count += 1
                        total_headlines_predicted += 1
                        custom_tokens = remove_noise(word_tokenize(row[3]))
                        if (classifier.classify(dict([token, True] for token in custom_tokens))== "Positive"):
                            weekly_positive_headlines_count += 1
                        if (classifier.classify(dict([token, True] for token in custom_tokens))== "Negative"):
                            weekly_negative_headlines_count += 1
    return (total_headlines_predicted, weekly_headlines_count, weekly_positive_headlines_count, weekly_negative_headlines_count)

# Fills empty dictionaries with pairs, key being the date and value being the positive sentiment percentage or net sentiment

def data_map(total_headlines_predicted, analysis_output, weekly_percentage_sentiment_price_pairs, weekly_net_sentiment_price_pairs):
    total_headlines_predicted = analysis_output [0]
    if (analysis_output[1] == 0):
        weekly_percentage_sentiment_price_pairs.update({current_week_str:0})
        weekly_net_sentiment_price_pairs.update({current_week_str:0})
    else:
        positive_percentage = (analysis_output[2] / analysis_output[1])*100
        net_sentiment = analysis_output[2] - analysis_output[3]
        weekly_percentage_sentiment_price_pairs.update({current_week_str:positive_percentage})
        weekly_net_sentiment_price_pairs.update({current_week_str:net_sentiment})
    return total_headlines_predicted

# Create two empty dictionaries for each company

tesla_percentage_pairs = {}
tesla_net_pairs = {}

amazon_percentage_pairs = {}
amazon_net_pairs = {}

netflix_percentage_pairs = {}
netflix_net_pairs = {}

apple_percentage_pairs = {}
apple_net_pairs = {}

disney_percentage_pairs = {}
disney_net_pairs = {}

alphabet_percentage_pairs = {}
alphabet_net_pairs = {}

pfizer_percentage_pairs = {}
pfizer_net_pairs = {}

microsoft_percentage_pairs = {}
microsoft_net_pairs = {}

mcdonalds_percentage_pairs = {}
mcdonalds_net_pairs = {}

starbucks_percentage_pairs = {}
starbucks_net_pairs = {}

# Iterate by week, starting from January 1st 2013, calling the headline_analysis function and using the results to call the data_map function

current_week = datetime(2013,1,1)
current_week_str = current_week.strftime("%Y-%m-%d")
total_headlines_predicted = 0
while(current_week_str < "2023-01-01"):
    next_week = current_week + relativedelta(weeks=1)
    current_week_str = current_week.strftime("%Y-%m-%d")
    next_week_str = next_week.strftime("%Y-%m-%d")
    print("Analysing tweets for week beginning " + current_week_str + "...")
    analysis_output = headline_analysis('Apple_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, apple_percentage_pairs, apple_net_pairs)
    analysis_output = headline_analysis('Tesla_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, tesla_percentage_pairs, tesla_net_pairs)
    analysis_output = headline_analysis('Amazon_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, amazon_percentage_pairs, amazon_net_pairs)
    analysis_output = headline_analysis('Netflix_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, netflix_percentage_pairs, netflix_net_pairs)
    analysis_output = headline_analysis('Disney_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, disney_percentage_pairs, disney_net_pairs)
    analysis_output = headline_analysis('Alphabet_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, alphabet_percentage_pairs, alphabet_net_pairs)
    analysis_output = headline_analysis('Pfizer_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, pfizer_percentage_pairs, pfizer_net_pairs)
    analysis_output = headline_analysis('Microsoft_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, microsoft_percentage_pairs, microsoft_net_pairs)
    analysis_output = headline_analysis('McDonald\'s_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, mcdonalds_percentage_pairs, mcdonalds_net_pairs)
    analysis_output = headline_analysis('Starbucks_all_tweets.csv', total_headlines_predicted, 0, 0, 0)
    total_headlines_predicted = data_map(total_headlines_predicted, analysis_output, starbucks_percentage_pairs, starbucks_net_pairs)
    print("Analysing tweets for week beginning " + current_week_str + " completed!")
    current_week = current_week + relativedelta(weeks=1)
print("Total Headlines Predicted:", total_headlines_predicted)

# Automatically generate graphs based on the company data given as a parameter

def auto_graph(weekly_percentage_pairs, weekly_net_pairs, company_name, stock_name):
    graph_name1 = stock_name + "_positive"
    graph_name2 = stock_name + "_net"
    x1 = []
    y1 = []
    with open(stock_name + '_weekly.csv') as file_obj:
        next(file_obj)
        # Create reader object by passing the file object to reader method
        reader_obj = csv.reader(file_obj)
        for row in reader_obj:
            if (row[0] >= "2013-01-01"):
                x1.append(row[0])
                y1.append(float(row[4]))
    x2 = []
    y2 = []
    for k, v in weekly_percentage_pairs.items():
        x2.append(k)
        y2.append(v)

    # Create figure and axis objects with subplots()

    fig,ax=plt.subplots()

    # Make a plot for the stock price line chart
    ax.plot(x1, y1, color = 'r', label = "Stock Price")
    ax.set_xlabel("Date")
    ax.set_ylabel(company_name + " Stock Price (USD)", color = "r", fontsize = 14)

    # Make a plot with different y-axis using second axis object for the sentiment bar chart

    ax2=ax.twinx()
    ax2.bar(x2, y2, color = 'b', label = "Sentiment", alpha = 0.5)
    ax2.set_ylabel(company_name + " Positive Sentiment Each Week (%)",color="b",fontsize=14)
    ax.tick_params(axis = "x", rotation = 45, labelsize = 5)
    temp = ax.xaxis.get_ticklabels()
    temp = list(set(temp) - set(temp[::4]))
    for label in temp:
        label.set_visible(False)
    plt.savefig(graph_name1)
    plt.show()

    x3 = []
    y3 = []
    for k, v in weekly_net_pairs.items():
        x3.append(k)
        y3.append(v)

    fig,ax=plt.subplots()
    ax.plot(x1, y1, color = 'r', label = "Stock Price")
    ax.set_xlabel("Date")
    ax.set_ylabel(stock_name + " Stock Price (USD)", color = "r", fontsize = 14)

    ax2=ax.twinx()
    ax2.bar(x3, y3, color = 'b', label = "Sentiment", alpha = 0.5)
    ax2.set_ylabel(company_name + " Net Sentiment Per Week",color="b",fontsize=14)
    ax.tick_params(axis = "x", rotation = 45, labelsize = 5)
    temp = ax.xaxis.get_ticklabels()
    temp = list(set(temp) - set(temp[::4]))
    for label in temp:
        label.set_visible(False)
    plt.savefig(graph_name2)
    plt.show()

# Call auto_graph function for each company being analysed

auto_graph(apple_percentage_pairs, apple_net_pairs, "Apple", "AAPL")
auto_graph(tesla_percentage_pairs, tesla_net_pairs, "Tesla", "TSLA")
auto_graph(netflix_percentage_pairs, netflix_net_pairs, "Netflix", "NFLX")
auto_graph(amazon_percentage_pairs, amazon_net_pairs, "Amazon", "AMZN")
auto_graph(disney_percentage_pairs, disney_net_pairs, "Disney", "DIS")
auto_graph(alphabet_percentage_pairs, alphabet_net_pairs, "Alphabet", "GOOGL")
auto_graph(pfizer_percentage_pairs, pfizer_net_pairs, "Pfizer", "PFE")
auto_graph(microsoft_percentage_pairs, microsoft_net_pairs, "Microsoft", "MSFT")
auto_graph(mcdonalds_percentage_pairs, mcdonalds_net_pairs, "McDonald\'s", "MCD")
auto_graph(starbucks_percentage_pairs, starbucks_net_pairs, "Starbucks", "SBUX")
```