University of Hertfordshire UH

Utility Of Twitter Tweet Sentiment In The Predictability Of Stock Prices

University of Hertfordshire

James Allen (19066364)
Data Science MSc Candidate

September 2022

Abstract

This paper examines the relevance and utility of using Twitter Tweet sentiment to predict price changes of stocks. There have been several high-profile examples of influential individuals and groups using social media to affect the price of stocks.

Drawing on various methods, this study examined the relationships between the sentiment and volume of Tweets and sought to identify correlations should they exist. This study concluded that there there is no significant advantage to using Tweet sentiment above that of standard regression models.

I have made numerous suggestions for how changes could improve the study, such as filtering for spikes in the dataset, including likes as a parameter, and significantly widening the pool of Tweets drawn upon. Due to time constraints, implementing these suggestions was beyond the scope of this study.

Acknowledgments

To my long-suffering wife Katy, whose support has been unwavering, I couldn't have done it without you. Thank you for everything.

My children, Oliver and Chloe. You were the basis of my inspiration to start this journey, and my motivation to see it through. Dad loves you x

I would like to thank my supervisor Dr Ralf Napiwotzki for his continued support throughout this project.

Finally, I would also like to thank the wider course team within School of Physics, Engineering and Computer Science at the University of Hertfordshire. In particular Dr James Geach, Dr Ashley Spindler, Dr Carolyn Devereaux and Dr Vidas Regelskis. Each of whom delivered some of the most interesting and inspiring learning experiences I have had the pleasure to receive.

Project Declaration

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire (UH).

It is my own work except where indicated in the report.

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on the university website, provided I, as the source, is acknowledged.

Contents

Chapter 1

Introduction

In recent years social media has exerted increasing influence over different areas of asset classes. Examples are so-called 'meme stocks' such as Gamestop (GME) and Robinhood (HOOD), which saw significant gains following their popularity within the Reddit community r/wallstreetbets Wallace (2021) Tchir (2021). Additionally, a single tweet from an influential figure can hugely impact assets. An example would be the impact Elon Musk continues to have on the cryptocurrency, Dogecoin Goodwin (2020). With this project, I intend to explore whether there is any utility to using Tweet sentiment and volume to predict stock price changes.

1.1 Literature Review

Analysis within asset price prediction is well-established and incorporates a range of disciplines such as technical analysis, quantitative analysis and behavioural economics. Much of this analysis uses directly related data such as asset prices at a given time, company information and trade volumes. These data enable the creation of models such as moving averages and Sharpe ratios to ascertain an asset's position in the broader market and how that relates to its current price Fabozzi & Drake (2009). With that said, alternative data, such as that available through news sites and social media, has begun to be used more frequently, presenting technical, moral and legislative challenges to its use. Hansen & Borch (2022)

Since they were introduced to the discipline in 2014, language tasks have been performed using Recurrent Neural Network, sequence to sequence models. This model takes a word vector as an input and a 'hidden state' from the previous step in a relatively simple state. This hidden step acts to contextualise the output that accompanies it (Figure 1.1). Joshi (2019)

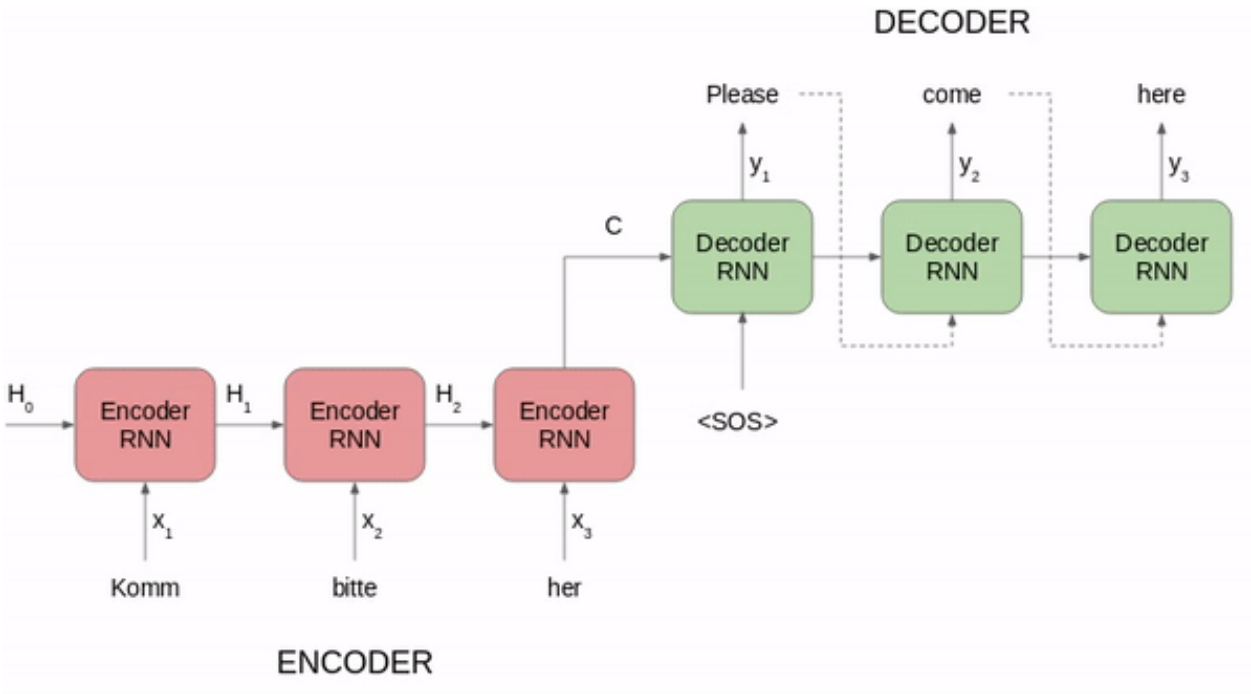


Figure 1.1: Simple Sequential Model

In 2017 a new self-attention Transformer Model was suggested that improved on the existing models in terms of both outcomes and training costs. Vaswani et al. (2017) This represents a significant leap forward for Natural Language Processing in a relatively short period of time. Subsequently, transformer models have formed the basis of some of the most advanced pre-training frameworks, such as Bidirectional Encoder Representations from Transformers (BERT) developed by Google. Jacob Devlin & Ming-Wei Chang (2018)

There have also been significant leaps regarding our access to computing power and natural language processing (NLP) frameworks Nambiar & Poess (2011). For example, it is possible to access multiple processors and significant RAM for a relatively low cost through services such as Google Cloud, Amazon Web Services or Microsoft Azure. These services have enabled the ubiquitous application of NLP frameworks Berleant (2015). This enhanced access to comparatively high computing power presents new opportunities to acquire and derive meaning from publicly available text at a previously unimaginable scale. In turn, this has the potential to alter how we approach stock price analysis from the perspective of both quantitative analysis and behavioural economics Stitchbury (2020).

1.2 Objectives and Method

I use publicly available resources to explore trends in Twitter sentiment towards asset classes. Furthermore, I seek to ascertain whether Twitter sentiment has any utility in predicting asset performance over a fixed period from January 2015 - December 2020. This project will be completed in a number of distinct parts (Figure 1.2).

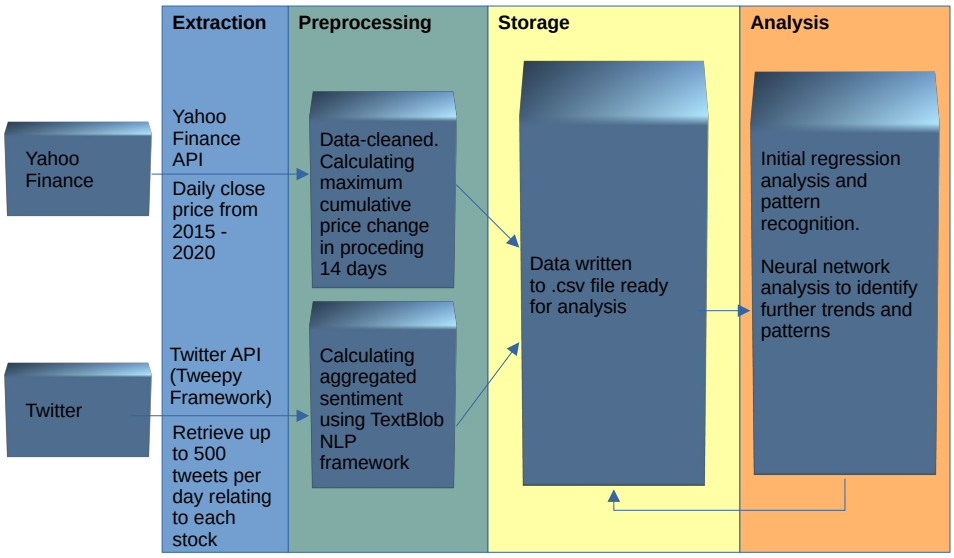


Figure 1.2: Project Process

The first part of this project involves acquiring, processing and building the data set. I have used the Yahoo Finance API to acquire financial data from December 2014 to January 2021. The Yahoo Finance API takes in data from a broader range than the target range to enable the calculation of the required projected price changes (https://openbase.com/python/yfinance/documentation). I also used the Twitter developer API and Tweepy extension to collect tweet content and volume relating to the individual stocks used in the study (https://developer.twitter.com). I used the Natural Language Processing framework TextBlob (https://textblob.readthedocs.io/en/dev/) to process the acquired Tweets and perform sentiment analysis. I then aggregated this sentiment for each day within the target range. In total, just over 5,400,000 Tweets from January 2015 to December 2020 were collated and analysed as part of this project. Since the part of the project took many days, I built a small home server from a Raspberry pi and deployed the code as a Docker container.

The second part of this project relates to the analysis of the data. I used various data analysis methods, including graphing and visualisations, regression techniques, and deep learning Neural Networks. I used graphing and visualisations to provide a cognitive understanding, at a glance, of the datasets and how the various features relate to each other. I used regression techniques to explore uniformity across the data and to take into account the momentum of stocks, whereby, for instance, if a stock has performed well for two weeks, it

has an increased likelihood of performing well for a third. Finally, I utilised a deep learning neural network with multiple dense layers. The purpose was to explore whether the additional complexity offered by these methods would identify patterns and relationships not picked up by more simplistic methods. I performed this analysis using the Python language, Keras and TensorFlow.

1.3 Chapter Overview

Chapter 1: Introduction

An overview of the project, including background information and inspiration for following this line of enquiry. This section also includes a Literature Review section of pertinent works relating to this project and an Objectives and Method section that details the processes used to gather the data and perform the analysis.

Chapter 2: Data Collection

An in-depth breakdown of resources and methods used to gather the data used in the project. This section includes details of applications made for enhanced data access.

Chapter 3: Data Preprocessing

Detailed information on the preprocessing techniques used in the project. This includes specific information on the additional features created and NLP frameworks to extract sentiment data.

Chapter 4: Analysis

This section covers the analysis of the data, including regression, correlation and neural network processes.

Chapter 5: Conclusions and Future Work

This final section covers the final conclusions for the project. I also cover the limitations I found in the project, areas I would have liked to cover further, and suggestions for different approaches to the project for the future.

Chapter 2

Data Collection

2.1 Yahoo Finance

First established in 1997, Yahoo Finance is the umbrella name for a range of media properties relating to businesses, finance, and stock markets. It provides content in various formats, such as videos and written news articles. Furthermore, Yahoo Finance partners with other websites from where it also posts content.

Yahoo Finance also provides up-to-date stock price information covering 79 exchanges from 50 countries and just under 10,000 quotes relating to crypto assets. In addition to its up-to-date information, Yahoo Finance also provides historical stock price data and company information.

Yahoo Finance's data is free and can be accessed directly through the website, https://uk.finance.yahoo.com/ via API or with the yfinance Python tool https://pypi.org/project/yfinance/. Registration with the Yahoo site is optional and is free at the point of use. While a premium option is available, it is not necessary to purchase this to access the information required for this project.

Yahoo Finance operates as a standalone company under Apollo fundsUnknown (2021), a subsidiary of the hedge fund Apollo Global Management, Inc. Verizon also retains a 10% stake in Yahoo.

2.1.1 The Yahoo Finance API

Yahoo Finance holds an extensive repository of financial information. I used the Python library yfinance to access the Yahoo Finance data for this study. yfinance is free, open-source and relatively straightforward to use. As such, it is a popular choice for accessing financial information, receiving 384,000 downloads per month, according to PyPi. Aroussi (2020)

The script for scraping the relevant financial data was written in Python and can be viewed in Appendix A.1. To ensure I had as much data as possible to draw upon, I chose to gather all closing price data for the S&P 500 from 25th December 2014 to 1st February 2021.

To do this, I first downloaded a full list of the companies featured from the Standard & Poors page on the S&P 500 (https://www.spglobal.com/spdji/en/indices/equity/sp-500/#data). I then had to write a script to clean and prepare the data, removing

whitespace from the text within the columns and making sure everything was formatted appropriately. Using this cleaned data, the script then built a list of all the ticker symbols in the S&P 500. Finally, the script iterated over that list of symbols and built the final data set ready for filtering and preprocessing before analysis. See appendix A.1 for source code.

In total, the final data set contained three columns (Date, Ticker and Close) and just over 750,000 rows containing daily closes for all stocks.

Date	Ticker	Close
2015-01-02	MMM	131.340057373047
2015-01-02	AOS	25.0512580871582
2015-01-02	ABT	38.9610404968262
2015-01-02	ABBV	47.901779174804 7
2015-01-02	ABMD	37.310001373291

Figure 2.1: Stock Price Data Sample

2.2 Twitter

Twitter is a social networking site where users post short messages known as "Tweets". Users can also "follow" accounts they find interesting and with which they wish to engage. Once a user has followed an account, any Tweets posted by that account will appear on the user's Home Feed. The Home Feed is what the primary stream users will see when they log in or open the app.

A Tweet has a maximum length of 280 characters, including spaces. Provided an account is not set to private, once a Tweet is sent, it can be viewed publicly by anyone on the internet.

Users can "like" Tweets by clicking a heart icon near the Tweets. Generally speaking, the more likes a Tweet has, the more people agree with it. However, this metric is not absolute as it will often be skewed, where more people will view tweets by accounts with large followings. Users can also "retweet" Tweets which means they share a Tweet to their profile, making it visible to their followers.

As of 2019, Twitter had a user count of approximately 330 million Dixon (2022). It is considered one of the leading social media networks worldwide and popular for personal and marketing use.

Twitter is a publicly traded company on the New York Stock Exchange. Its share distribution is 60% institutional investors, with Vanguard Group with the largest holding, 17% insider ownership, and the public owns the rest Business (2022).

2.2.1 Twitter Developer Platform

The Twitter Developer Platform (https://developer.twitter.com/) is the central area where one can access a range of Twitter tools, including advertising management, web integration and API access.

For this project, I focused explicitly on tools for accessing the API to request historical Tweets and Tweet volumes using Python. Before one can access Twitter data and make requests through the API, one must set up a Twitter developer account and apply for appropriate access privileges. There are three levels of non-commercial access available with a Twitter developer account Essential, Elevated and Academic:

Essential

This level allows users to request up to 500,000 tweets per month from the past seven days. With Essential access, the filter applied to requests is limited to 512 characters and five rules, and up to 25 requests every 15 minutes can be made. Essential access is quick and easy to set up, with almost immediate approval.

Elevated

Elevated access has similar limitations to Essential access in that users can only request tweets from the past seven days. It differs in that users can make up to 2 million tweets per month and make up to 50 requests per 15 minutes.

To apply for elevated access, one will need to make a written application through the developer portal. In my experience, this application and admission are dealt with relatively quickly.

Academic

Academic access offers the highest form of non-commercial access to the Twitter API. With Academic access, users can retrieve up to 10 million Tweets per month, using 1024 characters in the queries and 100 rules. Furthermore, academic access removes the limitations on how far back historical requests can go. This means requests can be made for Tweets dating back to Twitter's launch in 2006.

In the application, one has to add specific details of the project, the nature of the information to be retrieved and where the results of the research will be shared. One will also be expected to provide proof of academic credentials and the course for which one's research is performed.

2.2.2 Twitter Developer Application

For this project, I needed access to historical Tweet data going back as far as 2014. As a result, the only appropriate level of access that would satisfy these requirements was the Academic access level.

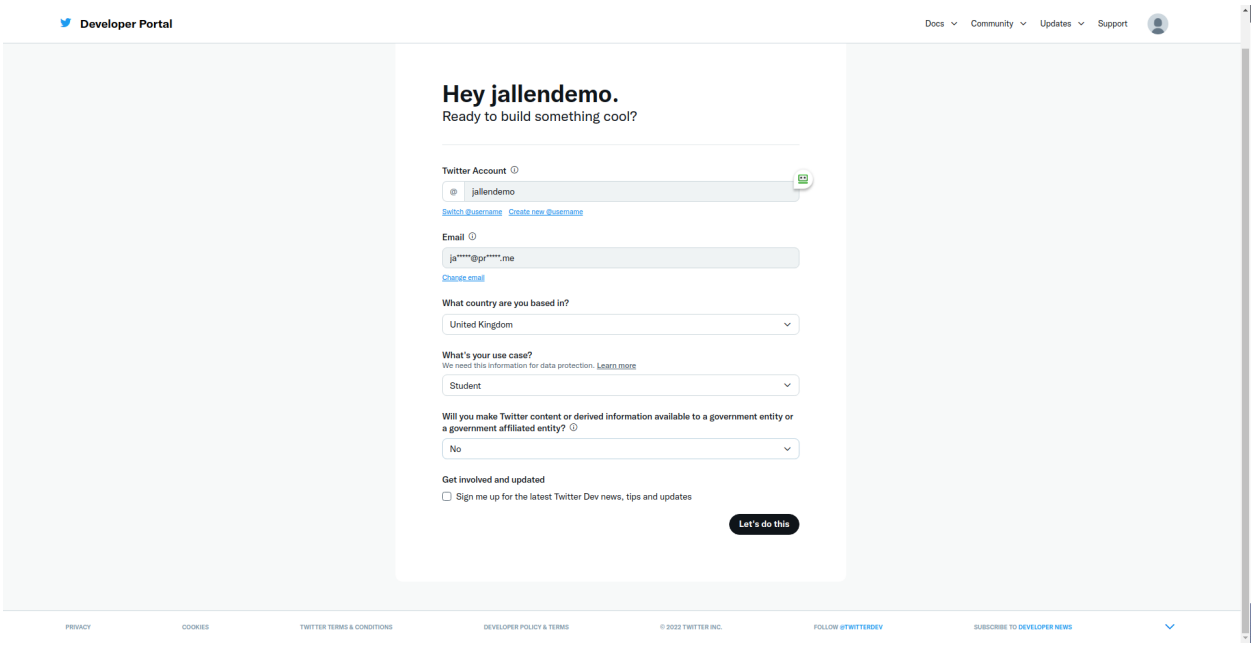


Figure 2.2: Twitter Developer Application 1

Before one can make an application to the developer platform, one must first register a Twitter account with a verified email address and phone number. Once this is complete, go to https://developer.twitter.com/ to begin one's application.

One will first be asked to confirm basic details such as the associated Twitter account, email and country where one is based (See figure 3.1). One will be asked to verify one use case, such as commercial, bot, and others. Given the nature of the project, I selected student. Once this part of the process is complete, one will automatically have access to the Essential level of API access.

The next stage is to make the application for Academic access. This involves providing as much information as possible on one's academic project. In my application, I tried to be as specific as possible regarding every aspect of the project, including packages, coding languages, and an overview of the process my code would go through to process the data.

I also tried to be as specific as possible as to the type of information I would be producing and the data I would be storing. For the purposes of this project, I would be processing the tweets in sequence, taking an aggregate sentiment and daily volume and then discarding the actual Tweets and usernames. Only ultimately holding a value for each day.

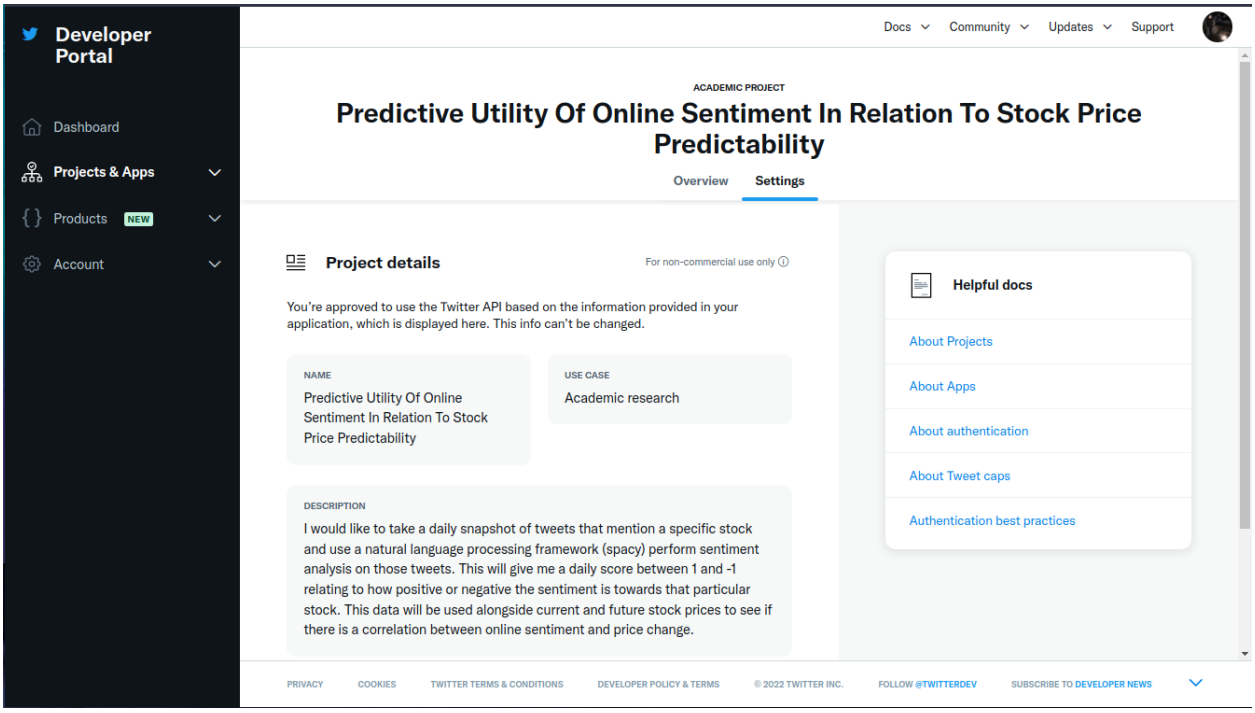


Figure 2.3: Twitter Developer Application 2

Finally, to support one's application, one will need to provide proof of one's academic credentials and the course one is on. Twitter has some preferred methods, such as Google workspace or a link to one academic profile on the universities website. Unfortunately, this wasn't available when I made the application. This meant my first application was denied, and it took a little back and forth with the approval team to work around the issue. In the end, I provided them with my LinkedIn profile and a screenshot of my student page confirming the details of the course on which I'm enrolled.

My application took roughly 48hrs to complete. Once that was done, I had access to the full Twitter API (Figure 3.2).

2.2.3 Tweepy

Tweepy (https://www.tweepy.org/) is an open-source Python library used to retrieve data from, and interact with, the Twitter API. Tweepy is intuitive and accessible and as a result, has just over 9,000 stars on GitHub.

Tweepy has a wide range of functionality. For instance, one can create, post, like and retweet Tweets. one can also retrieve user information such as region, follower count, following count and age of profile. This list of the functionality of Tweepy is by no means exhaustive as there are many actions one can perform with the library, depending on one's level of access acquired at the application stage.

For the purposes of this project, I was primarily concerned with acquiring bulk downloads of Tweets from the target time period.

Chapter 3

Data Preprocessing

3.1 Preprocessing (Yahoo Finance)

Preprocessing for the Yahoo Finance data took a number of steps, the first of which was to calculate the forward-looking price change for each of the target stocks over a range of time-frames between 7 and 28 days. The purpose of this was to provide a useful label for further analysis. Essentially for each line of stock for a given day's trading, there would be additional lines showing the actual cumulative price change in the upcoming 7, 14 and 28 days. I initialised two functions to support this work. The first takes a data source, stock ticker, start date and period as inputs. It then returns the largest price change from the price at the start date across that period. The second function then converts that absolute price change to a percentage. Once these functions were initialised, I ran them for the target periods of 7, 14 and 28 days and appended the resulting columns to the existing data.

Date	Ticker	Close	7 Day % Change	14 Day Price Change	14 Day % Change	21 Day % Change	28 Day % Change
2014-12-26	GOOG	26.6283912658691	-1.726	-1.888	-7.089	-4.859	1.109
2014-12-26	AMZN	15.4545001983643	0.954	-0.608	-3.934	-5.937	1.068
2014-12-26	META	80.7799987792969	-2.884	-3.04	-3.763	-6.932	-3.652
2014-12-26	MSFT	41.995662689209	-2.339	-0.605	-1.441	-3.425	-1.462
2014-12-26	NFLX	48.5785713195801	2.614	-1.537	-3.164	-0.797	28.646
2014-12-29	GOOG	26.4438972473145	-3.104	-1.884	-7.124	-4.195	1.814
2014-12-29	AMZN	15.6020002365112	-3.157	-1.031	-6.611	-6.826	-0.763
2014-12-29	META	80.0199966430664	-3.537	-3.3	-4.124	-6.048	-3.149
2014-12-29	MSFT	41.6185150146484	-2.36	-0.746	-1.791	-2.55	-0.927
2014-12-29	NFLX	48.8471412658691	-3.144	-3.3	-6.756	-1.342	30.6
2014-12-30	GOOG	26.4483852386475	-5.366	-1.707	-6.455	-4.434	-2.223
2014-12-30	AMZN	15.5150003433228	-4.837	-0.778	-5.015	-6.723	-1.144
2014-12-30	META	79.2200012207031	-3.875	-2.77	-3.497	-3.762	-4.342

Figure 3.1: Sample Data

3.2 Preprocessing (Twitter Data)

3.2.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a branch of Machine Learning and Artificial Intelligence that seeks to enable computers to understand human language, so that information

and insights can be extracted. In its purest form, NLP is multi-disciplinary in that it draws from a wide range of areas. These areas include computer science, artificial intelligence and linguistics.

There are also many distinct areas that exist within the discipline of NLP. These areas include speech recognition, speech segmentation, named entity recognition and sentiment analysis. For this project's purposes, we are primarily concerned with sentiment analysis as it pertains to tweets mentioning specific stocks.

3.2.2 Sentiment Analysis

Sentiment analysis is an area of natural language processing that is concerned with providing insight into unstructured text data that is written in a syntax similar to the syntax of human communication. Essentially sentiment analysis seeks to 'understand' the text and then provide insight as to whether the text indicates positive or negative sentiment. This indication can take different forms, the most common of which is graded sentiment, whereby the sentiment can be graded as very positive, positive, neutral, negative or very negative. This can also be represented as a continuous value. In this project, we will be using continuous values between -1 and 1 as provided by the TextBlob NLP framework to indicate sentiment where -1 indicates negative sentiment, 1 indicates positive sentiment, and 0 indicates neutral.

3.2.3 The Textblob Framework

TextBlob is an open-source, accessible and intuitive Python Library that is used for processing textual data. Text blob is built upon and acts as a scaled-down version of the widely used Natural Language Toolkit (NLTK). There are various use cases for Textblob, including noun phrase extraction, part of speech tagging, tokenization, word inflexion, lemmatization, WordNet integration, and others.

Using TextBlob first involves using a simple command to create a TextBlob object. Once this object is created, TextBlob can return sentiment analysis on the TextBlob that includes a polarity between -1 and 1, and a subjectivity score between 0 and 1. The polarity score is used to identify how positive or negative the text is, and the subjectivity score gives an idea of how subjective the content of the text used is. Below is an example of the outputs from some test data, the full code of which can be seen in Appendix A.3.

```
'I love $MSFT, climbing higher and higher' - Sentiment polarity =
    0.3333333333333333 - Sentiment subjectivity = 0.5333333333333333
'$META dropping like a stone' - Sentiment polarity = 0.0 - Sentiment
    subjectivity = 0.0
'$AMZN is on fire right now!' - Sentiment polarity = 0.3571428571428571 -
    Sentiment subjectivity = 0.5357142857142857
'People thought that $GOOG paid way too much for YouTube and look how that
    turned out.' - Sentiment polarity = 0.2 - Sentiment subjectivity = 0.2
'In case if all existing customers convert to ad customers then they will
    receive 27$/month ( from ads + customer price) it is huge free cash flow
```

```
    increase for $NFLX' - Sentiment polarity = 0.4 - Sentiment subjectivity =
    0.8500000000000001
'CBDC getting closer to reality (and with $AMZN in the race) >> #ECB Taps
    Amazon, Four Others to Pitch Digital Euro Prototype ' - Sentiment polarity =
    0.0 - Sentiment subjectivity = 0.0
```

3.2.4 Twitter Pre-processing Workflow

The pre-processing workflow was written in such a way as to iterate over each calendar day within the target date range. Since the original stock price didn't include weekend, and since we would still want to consider weekend sentiment data, I wrote the code to fill in the weekend data as it went. The missing close price data would be filled in with that of the preceding days, there for Saturdays and Sundays would contain Fridays data, but with accurate sentiment info for those days. I included it at this stage as it was more computationally efficient than splitting them out separately.

The code for this task runs in multiple stages. A simplified flowchart for the algorithm can be seen in figure 3.3, and the full code can be viewed in-app. The first stage is to initialise the security criteria so the API can access the Twitter API through the Twitter Dev portal.

I then created a function that would take a search term, start date and end date as inputs and return up to 500 Tweets for that day. Next, I created another function that takes a list of Tweets, and iterates over them, performing sentiment analysis for each one and returning the mean sentiment for the list of Tweets provided. Finally, I built a third function that takes in a search term, date from and date to and returns the volume of Tweets for that day.

Next, I initialised the required variables. This involved creating a blank data frame, setting a start date, setting an end date, setting the day increment, cursor date (which is an indicator of the code's position), and finally, a list of stock query strings and their associated stock ticker.

To execute the code, I wrote a while loop that would continue to run until the specified end date was reached Figure 3.3.

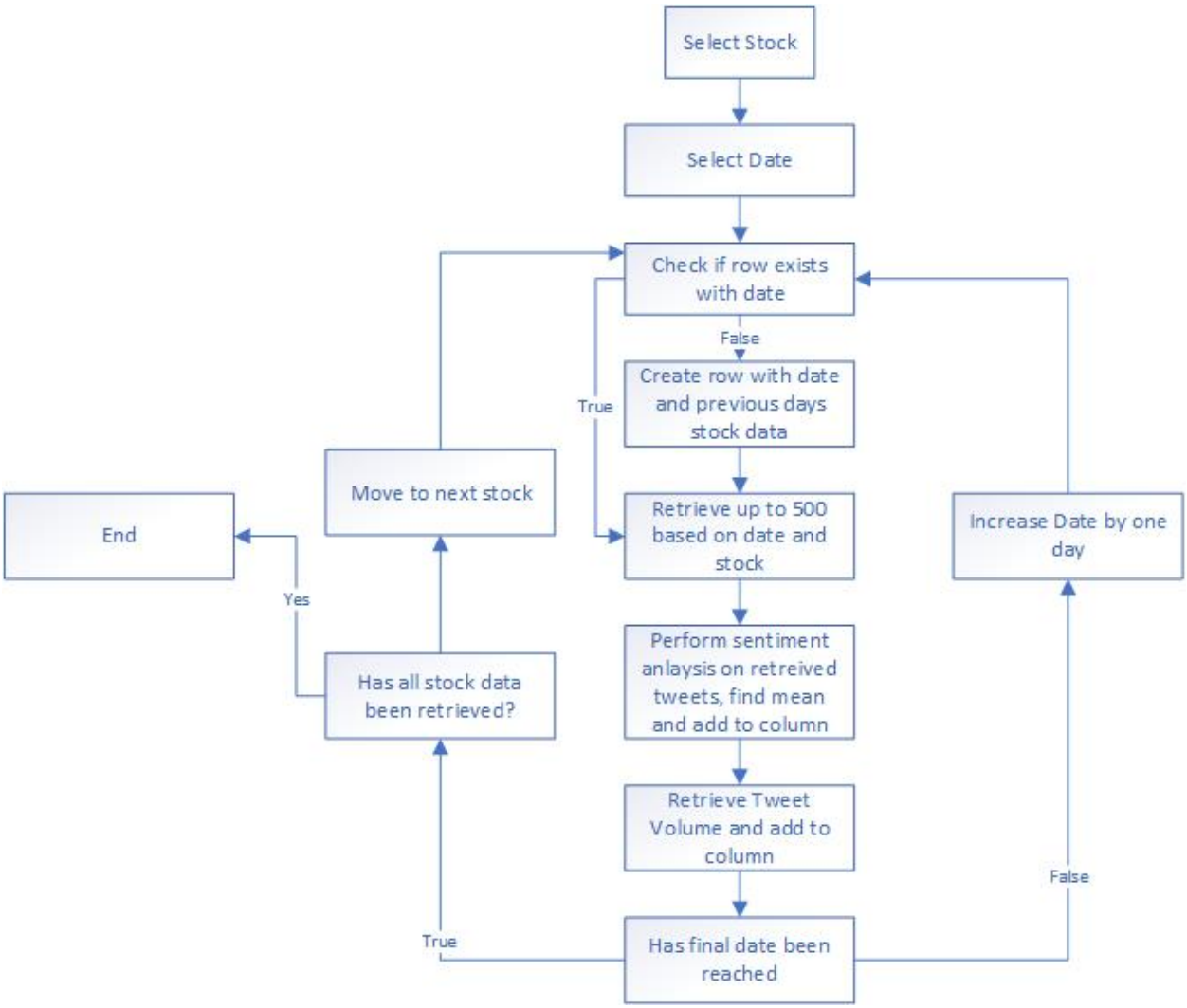


Figure 3.2: Twitter Data Workflow. For code see Appendix A.4

Within this while loop, the cursor date was formatted to an appropriate string that could be used by the Tweepy library. Next, a 'to date' was also created and formatted. With the date set and formatted, the script selects the first stock in the list and builds the search query to be used by Tweepy. Next, the script creates a date to be checked from the cursor date and checks to see if there is a row with that date. If there isn't, the script creates a row with the previous day's data.

Now the stock price data is available, regardless of whether or not it is a weekend date, the code starts gathering the Twitter data. First, it runs the function to return the average tweet sentiment for a day, with an except clause in case there are no Tweets for that day. If there aren't, the script will set the sentiment to 0, signifying neutral sentiment. Next, the script uses the function to retrieve the Tweet volume for the day and add that to the row with the rest of the data.

Once this is completed, it writes the data to an output CSV file and moves on to the next stock repeating the process. Once all dates and stocks have been collated and saved to

the output CSV, the script is complete.

3.2.5 Twitter Processing Result

Following the processing of the Twitter sentiment and volume data, I was able to create plots of each of the stocks together with the price changes and an indication of the beginning of 2020 to indicate when the coronavirus pandemic became a factor in prices in figure 3.4 you can see a sample of this data as it relates to META.

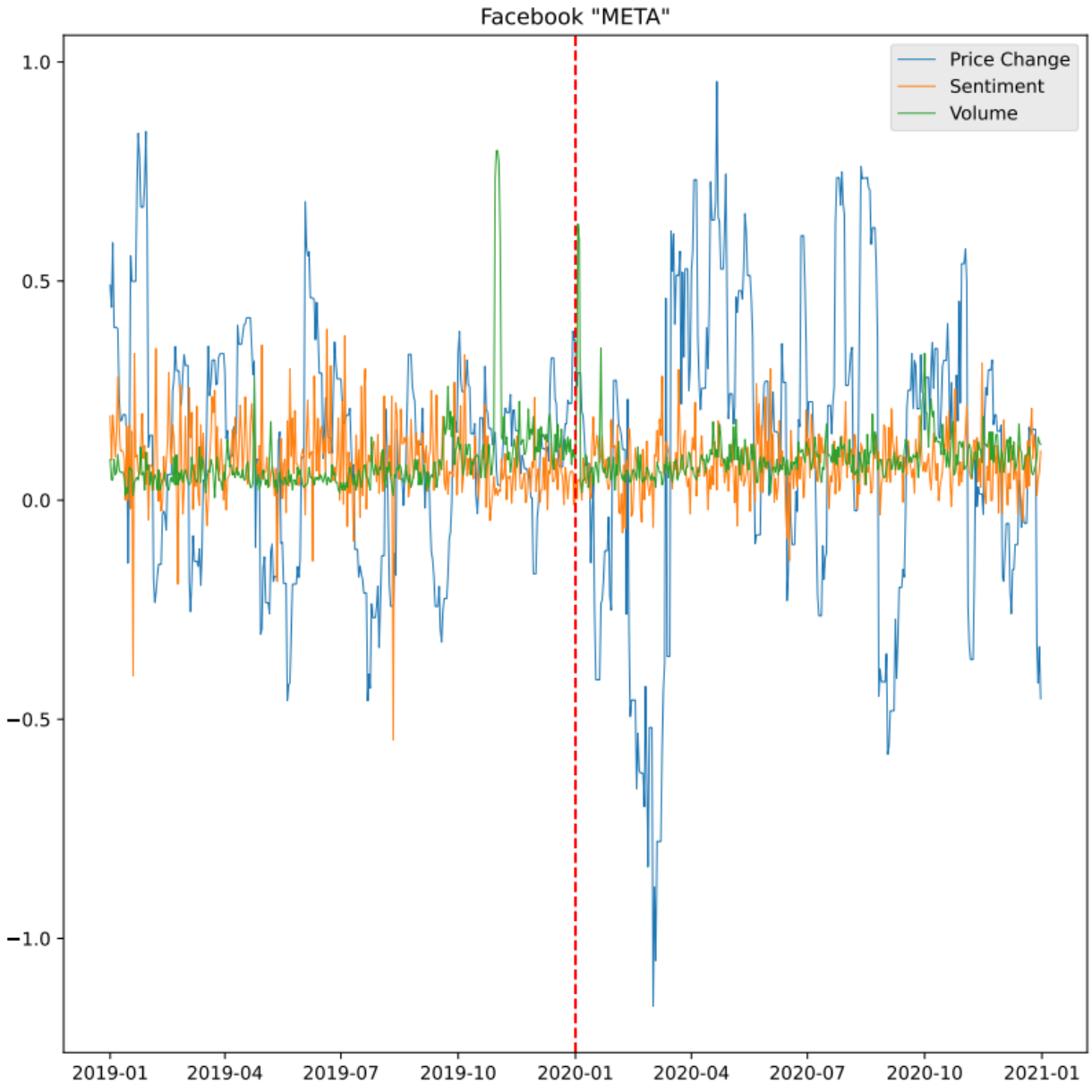


Figure 3.3: META Price Changes, Sentiment and Volume

Some interesting points to note. You can see a clear spike in Twitter activity (Green) for

Facebook / Meta in October 2019. This was due to Twitter banning political advertising campaigns, thus putting pressure on Facebook Wong (2019). Furthermore, figure 3.4 shows a clear fall in the stock price in early 2020 as the effects of the Coronavirus pandemic begin to show. This is reflected across all target stocks (See appendix A.5).

The spikes in the visualisation show that the data is fairly 'noisy'. To smooth out the data, I ran a series of scripts that would take 7, 14 and 28-day rolling averages for all of the data and plot it accordingly.

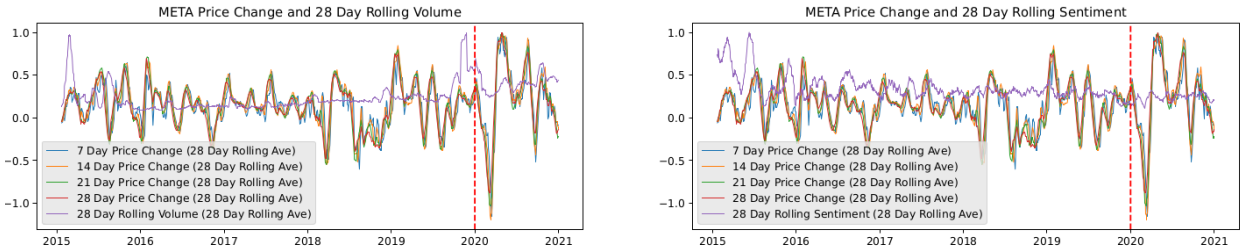


Figure 3.4: META Data, 28 Day Roling Averages

An example of the smoothed 28 days rolling data is shown in figure 3.5. Further examples for all stocks can be found in appendix A.6.

Chapter 4

Analysis

4.1 Correlation Matrix

A Correlation Matrix is a relatively simple table showing correlations between different data sets' features. I calculated the correlation using the Pandas framework and Pearson Coefficient which is the default setting for coefficients in Pandas.

Essentially the correlation matrix is where the correlation for each feature is measured against every other feature. This can then be represented either numerically or in visualisations such as heat maps. The equation to calculate a coefficient can be seen in figure 4.1

$$\rho_{X,Y} = \frac{\mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y]}{\sqrt{\mathbb{E}[X^2] - (\mathbb{E}[X])^2}\,\sqrt{\mathbb{E}[Y^2] - (\mathbb{E}[Y])^2}}.$$

Figure 4.1: Correlation Coefficient Equation

4.1.1 Correlation Matrix (Results)

The result of the correlation matrix can be seen in figure 4.2. As shown by the key, the lighter colours signify a higher correlation and the darker colours signify a lower correlation.

Much of the heatmap indications are as expected since many of the features are directly related. Examples of these are the absolute price and percentage changes against rolling average price changes.

An area of interest in the correlation is the week-on-week correlation between stock prices. This suggests that a price movement on a given week is an indicator of movements on the subsequent following weeks. A further explanation of this could be momentum theory Du (2012) which is an observation that the trajectory of a stock price can be predicted by recent price changes. A stock that has risen has a significant probability of continuing to rise. See Appendix A.7 for Correlation Matrix Code.

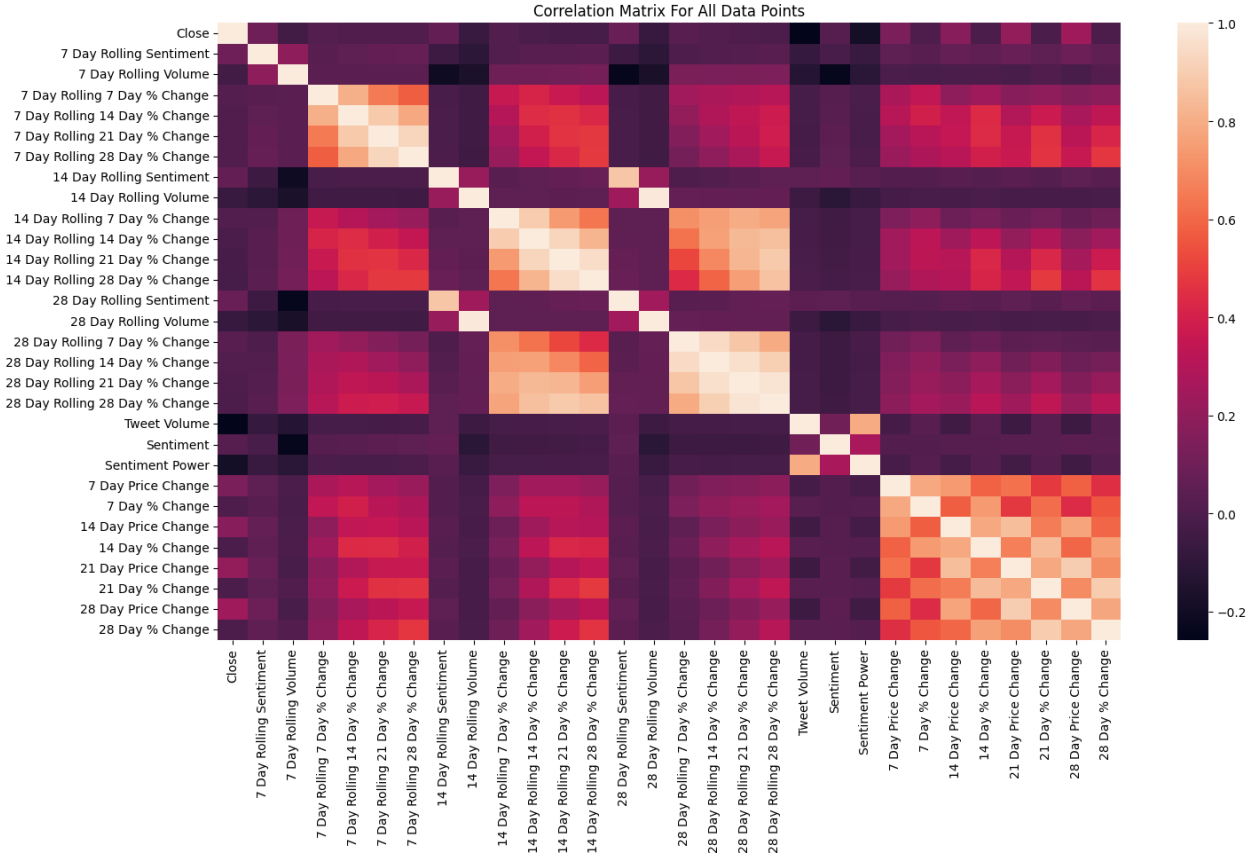


Figure 4.2: Correlation Matrix

4.2 Regression

In statistics, regression is a method that seeks to identify relationships between one dependent variable, often referred to as the Y variable or label, and other variables, often referred to as X or features. Regression is an incredibly powerful tool used in many industries and is the bedrock of statistical analysis.

The most common form of regression is least squared linear regression. The goal of least-squared linear regression is to find the minimum squared distance between the model line and the actual results. This is often thought of as error minimisation between a linear prediction line and the actual results.

4.2.1 Regression (Results)

For this project, I ran a number of regression models that incorporated support vector regression with various kernels as available within the python sklearn toolkit. The highest accuracy achieved was with Ordinary Least Squared (OLS) regression. The results of this can be seen below. Results were similar across all stocks, achieving an accuracy of around 80%.

The full code for the regression calculations can be found in appendix A.8, and the full results in appendix A.9.

```
MSFT Linear Regression accuracy: 0.8057802526355372
NFLX Linear Regression accuracy: 0.8103591502429877
META Linear Regression accuracy: 0.8164783534982967
GOOG Linear Regression accuracy: 0.8036488263096435
AMZN Linear Regression accuracy: 0.7847801667524315
```

4.3 Artificial Neural Network

Artificial Neural Networks are systems of computing inspired by the brain found in living animals. Artificial Neural Networks are constructed using multiple connected nodes. The connection of these computing nodes mimics that of the biological brain by simulating something akin to neurons. At initialisation, these nodes have an associated value which acts as a weight. A subset of the data known as the training data is fed into the network through the input layer and processed through the nodes. As the data is moves through the network, the weights of the nodes are updated towards minimising the error rate when compared against another subset of the data called the test set.

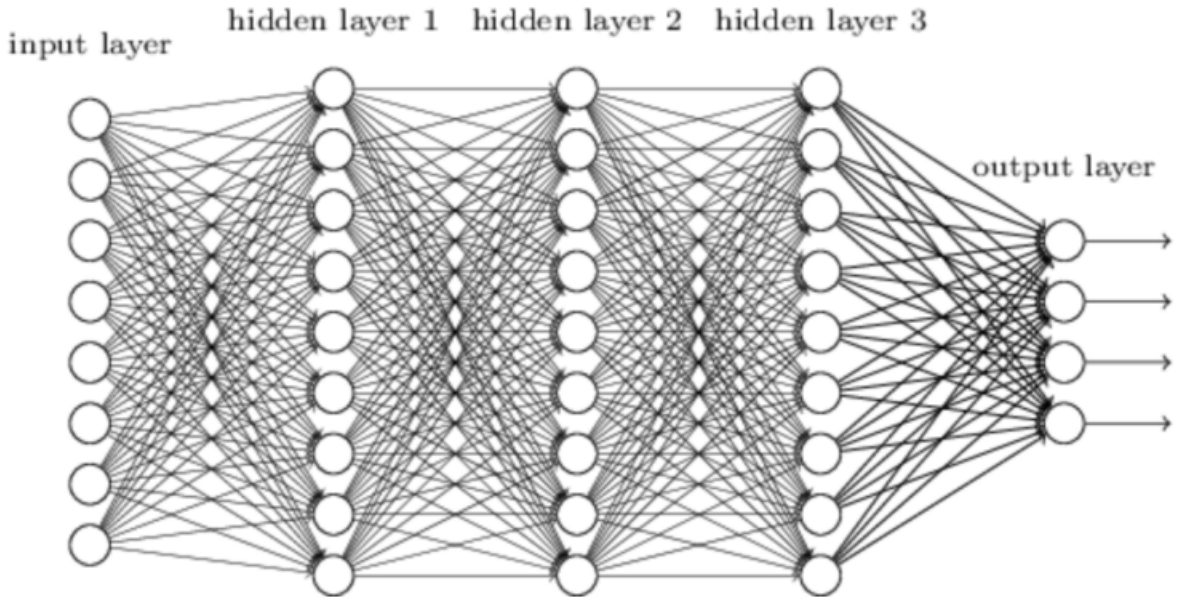


Figure 4.3: Deep Learning Neural Network

4.3.1 Neural Network (Results)

Given the time constraints of the project and significant commitments relating to the acquisition and pre-processing of the data, I was limited in the scope of neural networks I was able to apply to the data set. In total, I ran the Neural Network for 1,000 epochs. As seen in figure 4.4 the loss showed an error percentage of just under 50% this is essentially what you would expect from a random result. This suggests that the model is consistently over-training and presents no predictive utility for this data set. The full code can be seen in Appendix A.10.

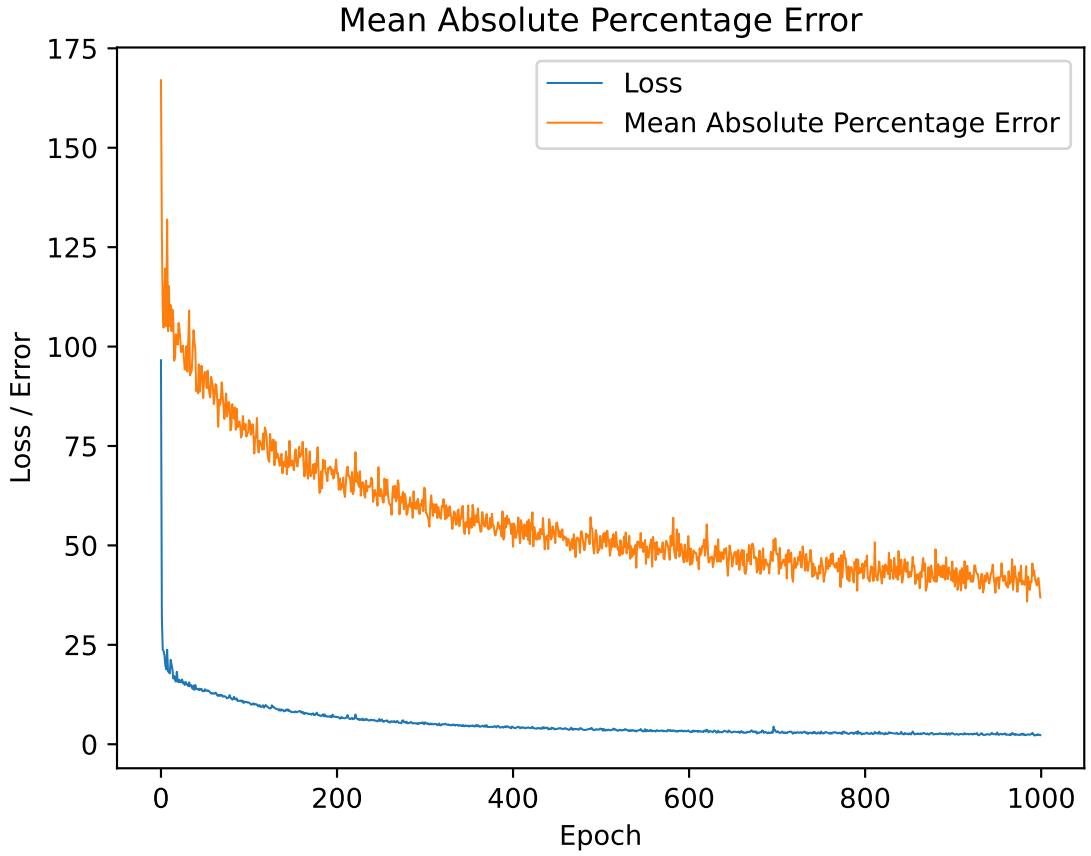


Figure 4.4: Mean Absolute Percentage Error

Chapter 5

Conclusions and Future Work

Following the research project, I can conclude that, in the format tested in this project, there is no benefit to using Twitter sentiment to predict stock prices. Simple regression models achieve an accuracy of approximately 80%, and the correlation matrix clearly showed correlations between week-on-week movements. Conversely, the Neural Network analysis quickly became over-trained on the data set. I would surmise that the complexity of a Neural Network is perhaps overkill for features related in such a way as those found in this data set.

If I were to undertake this project again or were to have more time available I would like to explore a number of avenues. The first would be to include a wider range of data thus increasing the spectrum of information to be taken in through analysis. This could include a much broader of range of stocks with more variations in their characteristics such as volatility.

I would also like to include a larger range of sources of data. For instance sentiment, data can be gathered from blogs, news sites and other social media channels. It would also be possible to consider as features the sources and authors of the data used. Furthermore, there is a significant amount of company information available such as board membership and ownership. Gathering information at this scale would require changes to the infrastructure of the project. As a result, it would become as much a Data Engineering project as well as a Data Science project however would be very interesting to explore.

finally, it would be interesting to approach the project from a slightly different angle and specifically target spikes in sentiment activity instead of specific stocks, and analyse that against price movements. This would be of interest as I found when extracting the Twitter data that there are protracted periods of minimal activity, therefore limiting what analysis could be performed. As with my previous point, this could be performed across a range of mediums.

Bibliography

Aroussi, R. (2020), 'yfinance: Download market data from Yahoo! Finance API'.
URL: *https://github.com/ranaroussi/yfinance*

Berleant, D. (2015), 'Trends in the cost of computing'. Section: AI Timelines.
URL: *https://aiimpacts.org/trends-in-the-cost-of-computing/*

Business, C. (2022), 'TWTR - Twitter Inc Shareholders - CNNMoney.com'.
URL: *https://money.cnn.com/quote/shareholders/shareholders.html?symb=TWTRsubView=institutior*

Dixon, S. (2022), 'Twitter MAU worldwide 2019'.
URL: *https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/*

Du, D. (2012), 'Momentum and behavioral finance', *Managerial Finance* **38**(4), 364–379. Publisher: Emerald Group Publishing Limited.
URL: *https://doi.org/10.1108/03074351211207527*

Fabozzi, F. J. & Drake, P. P. (2009), *Finance: Financial Markets, Business Finance, and Asset Management*, John Wiley & Sons, Hoboken. OCLC: 437111256.

Goodwin, J. (2020), 'Elon Musk tweeted about a bitcoin rival. It soared 20% | CNN Business'.
URL: *https://www.cnn.com/2020/12/20/investing/elon-musk-bitcoin-dogecoin/index.html*

Hansen, K. B. & Borch, C. (2022), 'Alternative data and sentiment analysis: Prospecting non-standard data in machine learning-driven finance', *Big Data & Society* **9**(1), 20539517211070701. Publisher: SAGE Publications Ltd.
URL: *https://doi.org/10.1177/20539517211070701*

Jacob Devlin & Ming-Wei Chang (2018), 'Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing'.
URL: *http://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html*

Joshi, P. (2019), 'Transformers In NLP | State-Of-The-Art-Models'.
URL: *https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/*

Nambiar, R. & Poess, M. (2011), Transaction Performance vs. Moore's Law: A Trend Analysis, *in* 'Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)', Vol. 6417, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 110–120.
URL: *https://go.exlibris.link/XkVmlyz3*

Stitchbury, J. (2020), 'Four ways to apply NLP in financial services'. Section: AI & Digitalization.
URL: *https://www.refinitiv.com/perspectives/ai-digitalization/four-ways-to-apply-nlp-in-financial-services/*

Tchir, P. (2021), 'What Do We 'Know' About Robinhood & WallStreetBets'. Section: Markets.
URL: *https://www.forbes.com/sites/petertchir/2021/01/30/what-do-we-know-about-robinhood–wallstreetbets/*

Unknown (2021), 'Apollo Funds Complete Acquisition of Yahoo'.
URL: *https://www.apollo.com/media/press-releases/2021/09-01-2021-161530593*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Gomez, A. N. & Polosukhin, I. (2017), 'Attention Is All You Need'. arXiv:1706.03762 [cs].
URL: *http://arxiv.org/abs/1706.03762*

Wallace, K. (2021), 'What the Heck Is Going on With GameStop?'. Section: Commentary.
URL: *https://www.morningstar.com/articles/1019249/what-the-heck-is-going-on-with-gamestop*

Wong, J. C. (2019), 'Twitter to ban all political advertising, raising pressure on Facebook', *The Guardian* .
URL: *https://www.theguardian.com/technology/2019/oct/30/twitter-ban-political-advertising-us-election*

Appendix A

Appendix

A.1 yfinance Data Mining (Code)

```
from os import truncate
from requests import head
import yfinance as yf
import pandas as pd

# Import s&p 500 data, strip whitespace from left and right of column names and
# remove spaces
sp_const_data = pd.read_csv("Data/Inputs/SP_500_Company_List.csv", header = 0)
sp_const_data.columns = sp_const_data.columns.str.lstrip()
sp_const_data.columns = sp_const_data.columns.str.rstrip()
sp_const_data.columns = sp_const_data.columns.str.replace(' ','_')

sp_const_data['Symbol'] = sp_const_data['Symbol'].str.rstrip()
symbols = sp_const_data['Symbol']

sp_stock_list = pd.DataFrame()

for stock in symbols:
    asset_temp = yf.Ticker(stock)
    asset_hist = asset_temp.history(start="2014-12-25", end="2021-02-1")
    asset_hist = pd.DataFrame(asset_hist['Close'])
    asset_hist['Ticker'] = stock
    asset_hist = asset_hist[['Ticker', 'Close']]
    sp_stock_list = pd.concat([sp_stock_list, asset_hist])

sp_stock_list.to_csv("Data/1_full_sp_stock_list.csv", index=False)
```

A.2 yfinance Preprocessing (Code)

```
import numpy as np
import pandas as pd
import datetime


# READ CSV, FILTER OUT PRE-DESIGNATED STOCKS AND ASSIGN TO 'filtered_stocks'
full_stock = pd.read_csv("Data/1_full_sp_stock_list.csv",
                     parse_dates=True)

target_stocks = ["MSFT", "META", "GOOG", "AMZN", "NFLX"]
filtered_stocks = pd.DataFrame(full_stock[full_stock.Ticker.isin(target_stocks)])


filtered_stocks.to_csv("Data/2_filtered_and_prepped_data.csv",
               index=False)
filtered_stocks = pd.read_csv("2_Data/filtered_and_prepped_data.csv",
                     parse_dates=True)



# FUNCTION TO FIND THE PRICE CHANGE OVER A GIVEN TIMEFRAME
def abs_price_change(stock_list, ticker, start_date, period):
    """Takes a stock list as a dataframe ticker symbol, a given date,
    and a time period and returns the greatest change over that period
    """
    start = datetime.date.fromisoformat(start_date)
    increment = datetime.timedelta(days=period)
    filt = (filtered_stocks['Date'] >= str(start))\
           & (filtered_stocks['Date'] <= str(start + increment))\
           & (filtered_stocks['Ticker'] == ticker)
    period = filtered_stocks.loc[filt]
    change = period['Close'] - period.iloc[0, 2] # Subtracts initial value from
        column
    largest_change = 0
    for i in change:
        if abs(i) > largest_change:
            largest_change = i
    return largest_change


# RUNS PERCENTAGE PRICE CHANGE FOR GIVEN INPUTS
def find_percent_change(initial_value, change):
    percent_change = ((((initial_value + change) / initial_value) - 1)) * 100
    return percent_change


# Run 'abs_price_change' for each line in stock list
```

```
# Run 'find_percent_change' for each line in stock list
filtered_stocks["7 Day Price Change"] = np.nan
filtered_stocks["7 Day % Change"] = np.nan
for index, row in filtered_stocks.iterrows():
    period = 7
    start_date = row['Date']
    ticker = row['Ticker']
    price_change = abs_price_change(filtered_stocks,
                                    ticker,
                                    start_date,
                                    period)
    filtered_stocks.iloc[index, 3] = round(price_change, 3)
    close_price = row['Close']
    percent_change = find_percent_change(close_price, price_change)
    filtered_stocks.iloc[index, 4] = round(percent_change, 3)


# Run 'abs_price_change' for each line in stock list
# Run 'find_percent_change' for each line in stock list
filtered_stocks["14 Day Price Change"] = np.nan
filtered_stocks["14 Day % Change"] = np.nan
for index, row in filtered_stocks.iterrows():
    period = 14
    start_date = row['Date']
    ticker = row['Ticker']
    price_change = abs_price_change(filtered_stocks,
                                    ticker,
                                    start_date,
                                    period)
    filtered_stocks.iloc[index, 5] = round(price_change, 3)
    close_price = row['Close']
    percent_change = find_percent_change(close_price, price_change)
    filtered_stocks.iloc[index, 6] = round(percent_change, 3)


filtered_stocks["21 Day Price Change"] = np.nan
filtered_stocks["21 Day % Change"] = np.nan
for index, row in filtered_stocks.iterrows():
    period = 21
    start_date = row['Date']
    ticker = row['Ticker']
    price_change = abs_price_change(filtered_stocks,
                                    ticker,
                                    start_date,
                                    period)
    filtered_stocks.iloc[index, 7] = round(price_change, 3)
    close_price = row['Close']
```

```
    percent_change = find_percent_change(close_price, price_change)
    filtered_stocks.iloc[index, 8] = round(percent_change, 3)


filtered_stocks["28 Day Price Change"] = np.nan
filtered_stocks["28 Day % Change"] = np.nan
for index, row in filtered_stocks.iterrows():
    period = 28
    start_date = row['Date']
    ticker = row['Ticker']
    price_change = abs_price_change(stock_list,
                                    ticker,
                                    start_date,
                                    period)
    filtered_stocks.iloc[index, 9] = round(price_change, 3)
    close_price = row['Close']
    percent_change = find_percent_change(close_price, price_change)
    filtered_stocks.iloc[index, 10] = round(percent_change, 3)


filtered_stocks.to_csv("Data/2_filtered_and_prepped_data.csv",
                index=False)
```

A.3 TextBlob Example

```
from textblob import TextBlob

text1 = TextBlob("I love $MSFT, climbing higher and higher")
text2 = TextBlob("$META dropping like a stone")
text3 = TextBlob("$AMZN is on fire right now!")
text4 = TextBlob("People thought that $GOOG paid way too much for YouTube and
    look how that turned out.")
text5 = TextBlob("In case if all existing customers convert to ad customers then
    they will receive 27$/month ( from ads + customer price) it is huge free cash
    flow increase for $NFLX")
text6 = TextBlob("CBDC getting closer to reality (and with $AMZN in the race) >>
    #ECB Taps Amazon, Four Others to Pitch Digital Euro Prototype ")

print(f"'{text1}' - Sentiment polarity = {text1.sentiment.polarity} - Sentiment
    subjectivity = {text1.sentiment.subjectivity}")
print(f"'{text2}' - Sentiment polarity = {text2.sentiment.polarity} - Sentiment
    subjectivity = {text2.sentiment.subjectivity}")
print(f"'{text3}' - Sentiment polarity = {text3.sentiment.polarity} - Sentiment
    subjectivity = {text3.sentiment.subjectivity}")
```

```
print(f"'{text4}' - Sentiment polarity = {text4.sentiment.polarity} - Sentiment
    subjectivity = {text4.sentiment.subjectivity}")
print(f"'{text5}' - Sentiment polarity = {text5.sentiment.polarity} - Sentiment
    subjectivity = {text5.sentiment.subjectivity}")
print(f"'{text6}' - Sentiment polarity = {text6.sentiment.polarity} - Sentiment
    subjectivity = {text6.sentiment.subjectivity}")
```

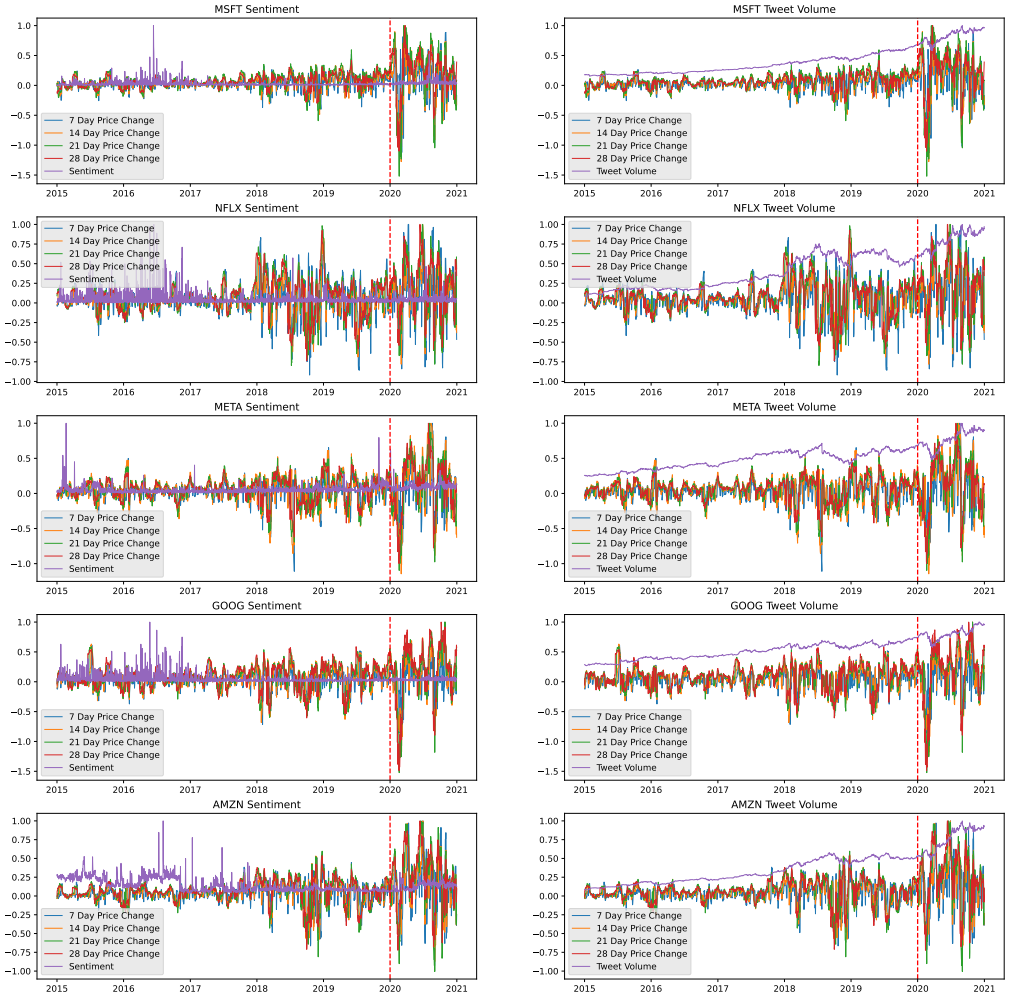
A.4 Twitter Scrape Full Code

```
import tweepy
import pandas as pd
import datetime
import time
import numpy as np

from textblob import TextBlob

__author__ = "James Allen"
__date__ = "2022-06-28"
__maintainer__ = "James Allen"
__website__ = "MeetJamesAllen.com"
__email__ = "james@meetjamesallen.com"

## Gain access to Twitter Developer account
keys = pd.read_csv('Data/Inputs/Twitter_Keys.csv', index_col=False, header=None)
API_Key = keys.iloc[0,1]
API_Key_Secret = keys.iloc[1,1]
Bearer_Token = keys.iloc[2,1]
Access_Token = keys.iloc[3,1]
Access_Token_Secret = keys.iloc[4,1]

client = tweepy.Client(Bearer_Token, wait_on_rate_limit = True) #
    wait_on_wait_limit pauses if Twitter limit reached


def retrieve_days_tweet(search_term, from_date, to_date):
    """Retrieves 500 tweets from a given day
    """
    tweet_list = []
    tweets = client.search_all_tweets(query = search_term,start_time =
        from_date,end_time = to_date,max_results = 500)
    for tweet in tweets.data:
        tweet_list.append(tweet.text)
    tweet_list_final = pd.DataFrame(tweet_list)
    print(tweet_list_final)
    return tweet_list_final


def get_sentiment(tweet_list):
    sentiment_list = pd.Series(dtype=float)
    for pos in range(len(tweet_list)):
        tweet = TextBlob(str(tweet_list.iloc[pos]))
        tweet_sentiment = pd.Series(tweet.polarity)
```

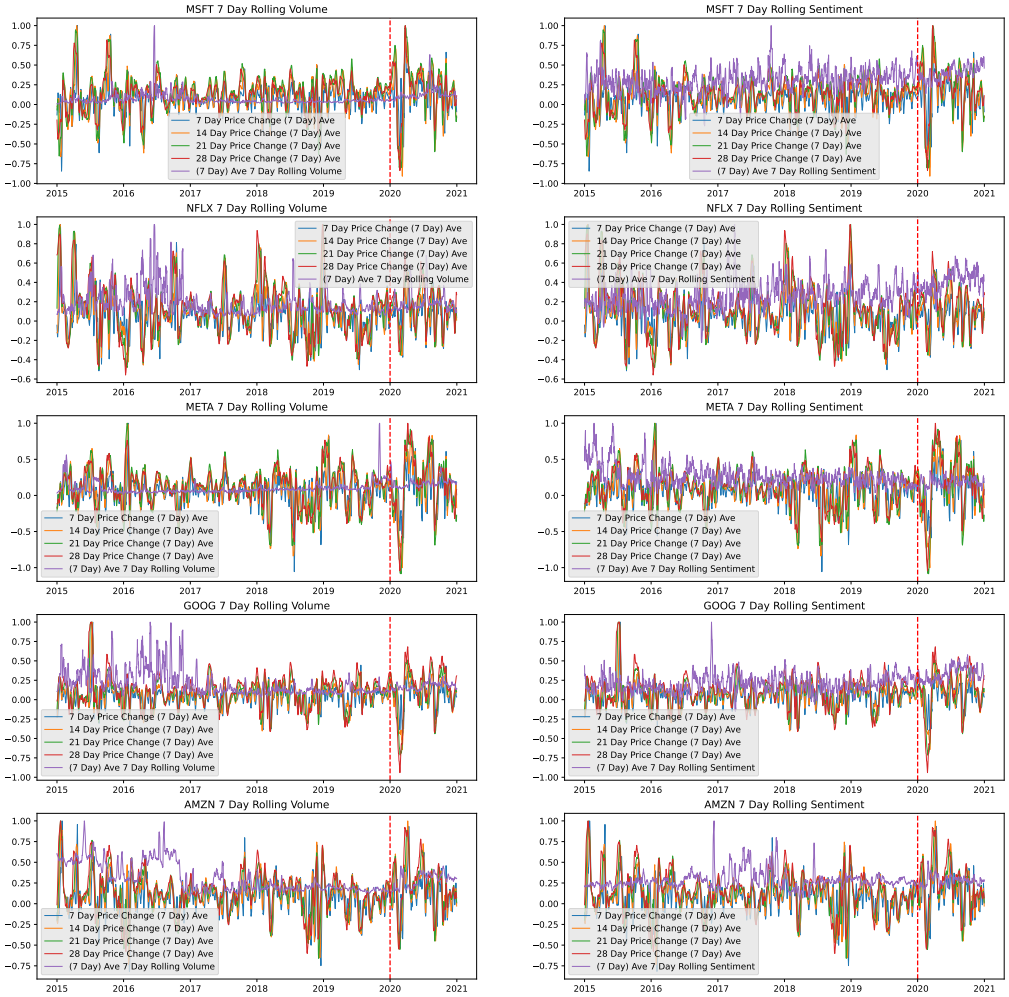
```
        sentiment_list = pd.concat([sentiment_list, tweet_sentiment])
    return sentiment_list.mean()

def count_tweets(search_term, from_date, to_date):
    return client.get_all_tweets_count(query = search_term,
                                       start_time=from_date,
                                       end_time=to_date,
                                       granularity='day')


data_with_tweets = pd.DataFrame() # New DataFrame with all onfo
start_date = datetime.date.fromisoformat('2020-07-14')
end_date = datetime.date.fromisoformat('2020-12-31')
increment = datetime.timedelta(days=1)
cursor_date = start_date
stocks_terms = [['$MSFT OR MSFT', 'MSFT'],
                ['$META OR META', 'META'],
                ['$GOOG OR GOOG','GOOG'],
                ['$AMZN OR AMZN', 'AMZN'],
                ['$NFLX OR NFLX', 'NFLX']]

stock_list = pd.read_csv('Data/2_filtered_and_prepped_data.csv')

final_stock_list = pd.DataFrame()

while cursor_date <= end_date:

    cursor_date_fmt = cursor_date.strftime('%Y-%m-%dT00:00:00Z')
    to_date = cursor_date + increment
    to_date_fmt = to_date.strftime('%Y-%m-%dT00:00:00Z')

    for term in stocks_terms:

        search_term = f"({term[0]}) -is:retweet lang:en (Buy OR Sell)"

        check_date = cursor_date.strftime('%Y-%m-%d')
        stock_row = stock_list.loc[(stock_list['Date'] == check_date) &
            (stock_list['Ticker'] == term[1])]
        if stock_row.empty:
            previous_day = cursor_date - increment
            previous_day = previous_day.strftime('%Y-%m-%d')
            weekend_row =
                pd.DataFrame(final_stock_list.loc[(final_stock_list['Date'] ==
                previous_day) & (final_stock_list['Ticker'] == term[1])])
            weekend_row['Date'] = check_date

            tweet_count = count_tweets(search_term, cursor_date_fmt, to_date_fmt)
```

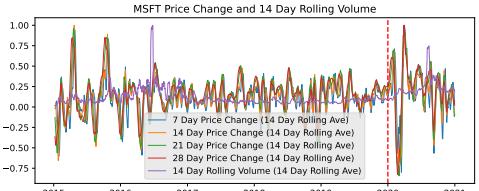
```
            count = tweet_count.data[0]['tweet_count']
            weekend_row['Tweet Volume'] = count
            time.sleep(1)

            try:
                tweets = retrieve_days_tweet(search_term, cursor_date_fmt,
                    to_date_fmt)
                twitter_sentiment = get_sentiment(tweets)
                weekend_row['Sentiment'] = twitter_sentiment
                final_stock_list = pd.concat([final_stock_list, weekend_row])
            except:
                weekend_row['Sentiment'] = 0
                final_stock_list = pd.concat([final_stock_list, weekend_row])

        else:
            stock_row = pd.DataFrame(stock_row)
            tweet_count = count_tweets(search_term, cursor_date_fmt, to_date_fmt)
            count = tweet_count.data[0]['tweet_count']
            stock_row['Tweet Volume'] = count
            time.sleep(1)
            try:
                tweets = retrieve_days_tweet(search_term, cursor_date_fmt,
                    to_date_fmt)
                twitter_sentiment = get_sentiment(tweets)
                stock_row['Sentiment'] = twitter_sentiment
                final_stock_list = pd.concat([final_stock_list, stock_row])
            except:
                stock_row['Sentiment'] = 0
                final_stock_list = pd.concat([final_stock_list, stock_row])


        time.sleep(1)
        final_stock_list.to_csv('Data/3_analysis_ready_with_sentiment.csv',
            index=False)
        final_stock_list = pd.read_csv('Data/3_analysis_ready_with_sentiment.csv')
        final_stock_list['Date'] = final_stock_list['Date'].astype(str)

    cursor_date = cursor_date + increment
```

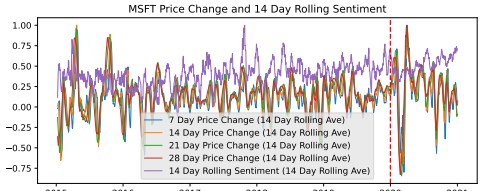
A.5 Price, Sentiment and Tweet Volume Data(All)

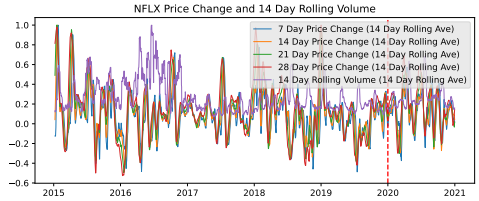


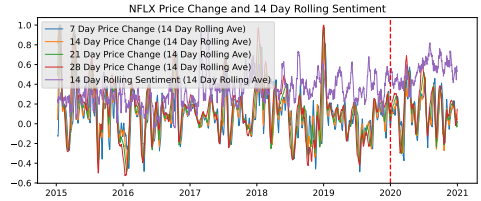
A.6 Rolling Sentiment (7, 14 and 28 Day)

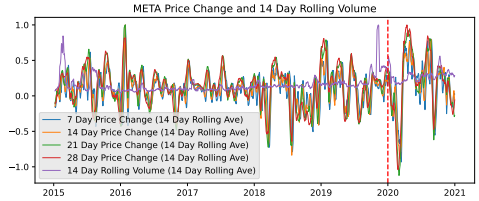


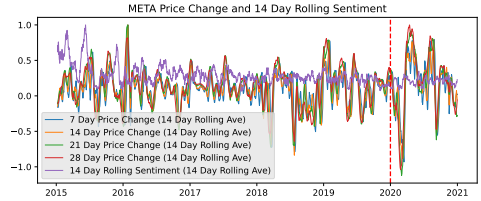

MSFT Price Change and 14 Day Rolling Volume
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Volume (14 Day Rolling Ave)

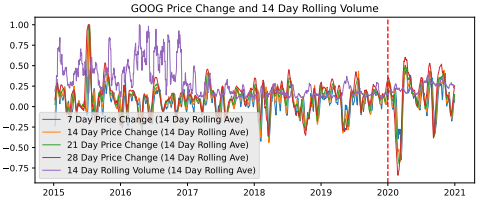

MSFT Price Change and 14 Day Rolling Sentiment
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Sentiment (14 Day Rolling Ave)

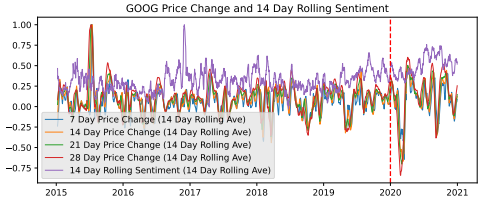

NFLX Price Change and 14 Day Rolling Volume
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Volume (14 Day Rolling Ave)

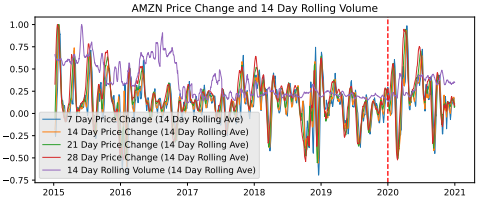

NFLX Price Change and 14 Day Rolling Sentiment
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Sentiment (14 Day Rolling Ave)

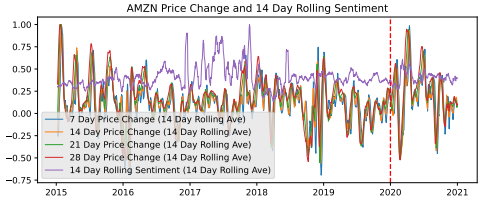

META Price Change and 14 Day Rolling Volume
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Volume (14 Day Rolling Ave)

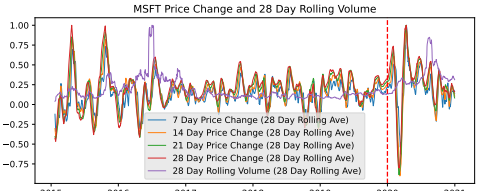

META Price Change and 14 Day Rolling Sentiment
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Sentiment (14 Day Rolling Ave)

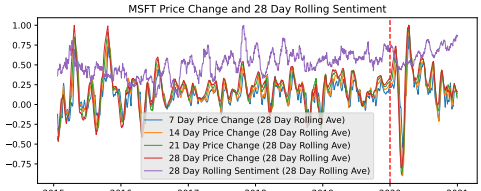

GOOG Price Change and 14 Day Rolling Volume
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Volume (14 Day Rolling Ave)

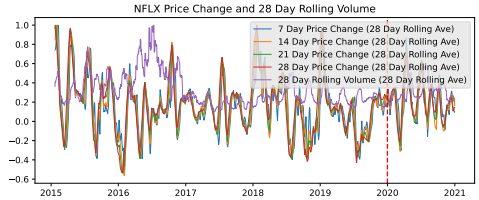

GOOG Price Change and 14 Day Rolling Sentiment
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Sentiment (14 Day Rolling Ave)

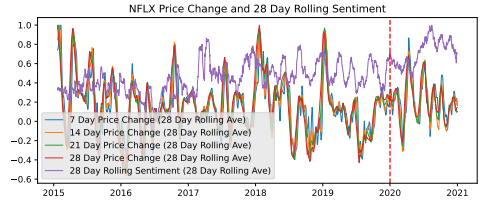

AMZN Price Change and 14 Day Rolling Volume
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Volume (14 Day Rolling Ave)

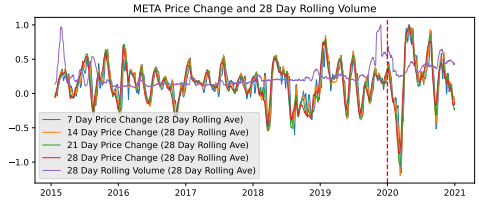

AMZN Price Change and 14 Day Rolling Sentiment
7 Day Price Change (14 Day Rolling Ave)
14 Day Price Change (14 Day Rolling Ave)
21 Day Price Change (14 Day Rolling Ave)
28 Day Price Change (14 Day Rolling Ave)
14 Day Rolling Sentiment (14 Day Rolling Ave)

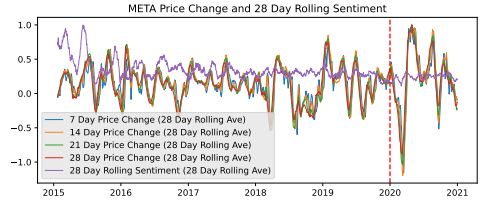

MSFT Price Change and 28 Day Rolling Volume
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Volume (28 Day Rolling Ave)

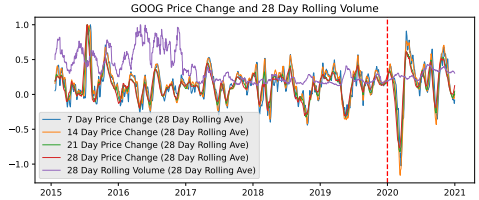

MSFT Price Change and 28 Day Rolling Sentiment
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Sentiment (28 Day Rolling Ave)

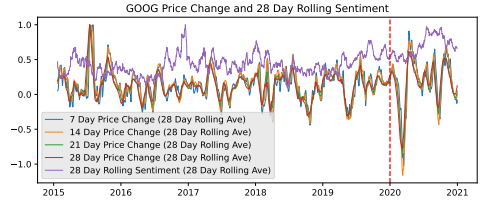

NFLX Price Change and 28 Day Rolling Volume
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Volume (28 Day Rolling Ave)

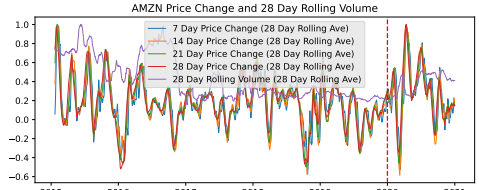

NFLX Price Change and 28 Day Rolling Sentiment
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Sentiment (28 Day Rolling Ave)

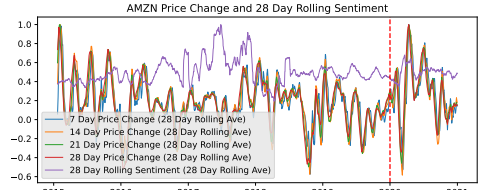

META Price Change and 28 Day Rolling Volume
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Volume (28 Day Rolling Ave)


META Price Change and 28 Day Rolling Sentiment
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Sentiment (28 Day Rolling Ave)


GOOG Price Change and 28 Day Rolling Volume
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Volume (28 Day Rolling Ave)


GOOG Price Change and 28 Day Rolling Sentiment
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Sentiment (28 Day Rolling Ave)


AMZN Price Change and 28 Day Rolling Volume
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Volume (28 Day Rolling Ave)


AMZN Price Change and 28 Day Rolling Sentiment
7 Day Price Change (28 Day Rolling Ave)
14 Day Price Change (28 Day Rolling Ave)
21 Day Price Change (28 Day Rolling Ave)
28 Day Price Change (28 Day Rolling Ave)
28 Day Rolling Sentiment (28 Day Rolling Ave)

A.7 Correlation Matrix Code

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

data = pd.read_csv('data/Stock_Data_Full_Final.csv')


data_values = data[['Close', '7 Day Rolling Sentiment', '7 Day Rolling Volume',
                    '7 Day Rolling 7 Day % Change', '7 Day Rolling 14 Day %
                       Change',
                    '7 Day Rolling 21 Day % Change', '7 Day Rolling 28 Day %
                       Change',
                    '14 Day Rolling Sentiment', '14 Day Rolling Volume',
                    '14 Day Rolling 7 Day % Change', '14 Day Rolling 14 Day %
                       Change',
                    '14 Day Rolling 21 Day % Change', '14 Day Rolling 28 Day %
                       Change',
                    '28 Day Rolling Sentiment', '28 Day Rolling Volume',
                    '28 Day Rolling 7 Day % Change', '28 Day Rolling 14 Day %
                       Change',
                    '28 Day Rolling 21 Day % Change', '28 Day Rolling 28 Day %
                       Change',
                    'Tweet Volume', 'Sentiment', 'Sentiment Power',
                    '7 Day Price Change', '7 Day % Change', '14 Day Price Change',
                    '14 Day % Change', '21 Day Price Change', '21 Day % Change',
                    '28 Day Price Change', '28 Day % Change']]

corr_matrix = data_values.corr()
sn.heatmap(corr_matrix, xticklabels=True, yticklabels=True)
plt.title('Correlation Matrix For All Data Points')
plt.show()
```

A.8 Regression Code

```
import pandas as pd
import numpy as np
from sklearn import preprocessing, svm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

data = pd.read_csv('data/Stock_Data_Full_Final.csv')
```

```
data_values = data[['Ticker', 'Close', '7 Day Rolling Sentiment', '7 Day Rolling
    Volume',
                    '7 Day Rolling 7 Day % Change', '7 Day Rolling 14 Day %
                        Change',
                    '7 Day Rolling 21 Day % Change', '14 Day Rolling Sentiment',
                        '14 Day Rolling Volume',
                    '14 Day Rolling 7 Day % Change', '14 Day Rolling 14 Day %
                        Change',
                    '14 Day Rolling 21 Day % Change', '28 Day Rolling Sentiment',
                        '28 Day Rolling Volume',
                    '28 Day Rolling 7 Day % Change', '28 Day Rolling 14 Day %
                        Change',
                    '28 Day Rolling 21 Day % Change', 'Tweet Volume', 'Sentiment',
                        'Sentiment Power',
                    '7 Day Price Change', '7 Day % Change', '14 Day Price Change',
                    '14 Day % Change', '21 Day Price Change', '21 Day % Change',
                    '28 Day % Change']]


tickers = ['MSFT', 'NFLX', 'META', 'GOOG', 'AMZN']

for ticker in tickers:
    forecast_check = 60 # Number of days to attempt forecasting
    data_values = data[['Ticker', 'Close', '7 Day Rolling Sentiment', '7 Day
        Rolling Volume',
                    '7 Day Rolling 7 Day % Change', '7 Day Rolling 14 Day %
                        Change',
                    '7 Day Rolling 21 Day % Change', '14 Day Rolling Sentiment',
                        '14 Day Rolling Volume',
                    '14 Day Rolling 7 Day % Change', '14 Day Rolling 14 Day %
                        Change',
                    '14 Day Rolling 21 Day % Change', '28 Day Rolling Sentiment',
                        '28 Day Rolling Volume',
                    '28 Day Rolling 7 Day % Change', '28 Day Rolling 14 Day %
                        Change',
                    '28 Day Rolling 21 Day % Change', 'Tweet Volume', 'Sentiment',
                        'Sentiment Power',
                    '7 Day Price Change', '7 Day % Change', '14 Day Price Change',
                    '14 Day % Change', '21 Day Price Change', '21 Day % Change',
                    '28 Day % Change']]

    data_mask = data_values['Ticker'] == ticker
    data_filtered = pd.DataFrame(data_values[data_mask])
    data_filtered.reset_index(inplace=True)
    data_filtered.drop('Ticker', axis=1, inplace=True)
```

```
X = np.array(data_filtered.drop(['28 Day % Change'],1))
X_forecast = X[-forecast_check:] # Returns real values for X which we can
    visualise
X = preprocessing.scale(X)
y = np.array(data_filtered['28 Day % Change'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)


# Linear Regression
lin_reg_clf = LinearRegression() # Set linear regression classifier
lin_reg_clf.fit(X_train, y_train)
lin_reg_acc = lin_reg_clf.score(X_test, y_test) # Find linear regression
    accuracy
print(f'{ticker} Linear Regression accuracy: {lin_reg_acc}')


# Support Vector Regression (Linear kernel)
svl_reg_clf = svm.SVR(kernel='linear')
svl_reg_clf.fit(X_train, y_train)
svl_reg_acc = svl_reg_clf.score(X_test, y_test) # Find linear regression
    accuracy
print(f'{ticker} Support Vector (Linear) accuracy: {svl_reg_acc}')


# Support Vector Regression (Radial Basis Function Kernal)
svrbf_reg_clf = svm.SVR(kernel='rbf')
svrbf_reg_clf.fit(X_train, y_train)
svrbf_reg_acc = svrbf_reg_clf.score(X_test, y_test) # Find linear regression
    accuracy
print(f'{ticker} Support Vector (Radial Basis Function) accuracy:
    {svrbf_reg_acc}')


# Support Vector Regression (Polynomial kernel)
svp_reg_clf = svm.SVR(kernel='poly')
svp_reg_clf.fit(X_train, y_train)
svp_reg_acc = svp_reg_clf.score(X_test, y_test) # Find linear regression
    accuracy
print(f'{ticker} Support Vector (Polynomial) accuracy: {svp_reg_acc}')


# Support Vector Regression (Sigmoid kernel)
svs_reg_clf = svm.SVR(kernel='sigmoid')
svs_reg_clf.fit(X_train, y_train)
svs_reg_acc = svs_reg_clf.score(X_test, y_test) # Find linear regression
    accuracy
print(f'{ticker} Support Vector (Sigmoid) accuracy: {svs_reg_acc}')
```

A.9 Regression Results

```
MSFT Linear Regression accuracy: 0.8057802526355372
MSFT Support Vector (Linear) accuracy: 0.7963724798712932
MSFT Support Vector (Radial Basis Function) accuracy: 0.6408735094390159
MSFT Support Vector (Polynomial) accuracy: 0.43201264145408225
MSFT Support Vector (Sigmoid) accuracy: -8.11380818697977

NFLX Linear Regression accuracy: 0.8103591502429877
NFLX Support Vector (Linear) accuracy: 0.797703766468536
NFLX Support Vector (Radial Basis Function) accuracy: 0.6540319693143426
NFLX Support Vector (Polynomial) accuracy: 0.4947504899034143
NFLX Support Vector (Sigmoid) accuracy: -0.1732768140880243

META Linear Regression accuracy: 0.8164783534982967
META Support Vector (Linear) accuracy: 0.8169297544647828
META Support Vector (Radial Basis Function) accuracy: 0.7184078976737989
META Support Vector (Polynomial) accuracy: 0.6453622026655645
META Support Vector (Sigmoid) accuracy: -2.143287371860095

GOOG Linear Regression accuracy: 0.8036488263096435
GOOG Support Vector (Linear) accuracy: 0.7949518626202864
GOOG Support Vector (Radial Basis Function) accuracy: 0.6215375121974136
GOOG Support Vector (Polynomial) accuracy: 0.626661151346368
GOOG Support Vector (Sigmoid) accuracy: -5.116554100610496

AMZN Linear Regression accuracy: 0.7847801667524315
AMZN Support Vector (Linear) accuracy: 0.7817194077904265
AMZN Support Vector (Radial Basis Function) accuracy: 0.695417758774072
AMZN Support Vector (Polynomial) accuracy: 0.5144202841825164
AMZN Support Vector (Sigmoid) accuracy: -2.390877156740232
```

A.10 Neural Network Code

```
from pyexpat import XML_PARAM_ENTITY_PARSING_ALWAYS
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import losses
from keras import metrics
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from sklearn import preprocessing
```

```
data_full = pd.read_csv('Data_Analysis/code/data/Data_Prepped_For_NN.csv')
data_full = data_full.sample(frac=1, random_state=15)
data_full.reset_index(inplace=True, drop=True)



training_rows = round(data_full.shape[0] * 0.8) # Set training rows to 80% of
    total
test_rows = data_full.shape[0] - training_rows
features = data_full[['Close', '7 Day Rolling Sentiment', '7 Day Rolling
    Volume', '7 Day Rolling 7 Day % Change',
            '7 Day Rolling 14 Day % Change', '7 Day Rolling 21 Day % Change', '14
                Day Rolling Sentiment',
            '14 Day Rolling Volume', '14 Day Rolling 7 Day % Change', '14 Day
                Rolling 14 Day % Change',
            '14 Day Rolling 21 Day % Change', '28 Day Rolling Sentiment', '28 Day
                Rolling Volume',
            '28 Day Rolling 7 Day % Change', '28 Day Rolling 14 Day % Change',
                '28 Day Rolling 21 Day % Change',
            'Tweet Volume', 'Sentiment', 'Sentiment Power', '7 Day Price Change',
                '7 Day % Change',
            '14 Day Price Change', '14 Day % Change', '21 Day Price Change', '21
                Day % Change']]
label = data_full['28 Day % Change']


X_train = features.iloc[:training_rows]
# X_train = preprocessing.normalize(X_train)

Y_train = label.iloc[:training_rows]
X_test = features.iloc[-test_rows:]
Y_test = label.iloc[-test_rows:]


print(f"x_train {X_train.shape}")
print(f"y_train {Y_train.shape}")
print(f"x_test {X_test.shape}")
print(f"y_test {Y_test.shape}")


model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_dim=25))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(1))
```

```
# Show model summary
model.summary()


# Loss function
loss_fn = tf.keras.losses.MeanSquaredError(reduction="auto",
    name="mean_squared_error")


# Compiler
model.compile(optimizer='Adam',
              loss=loss_fn,
              metrics=['mape'])

history = model.fit(X_train, Y_train, epochs=1000, batch_size=64)


# Plot losses and accuracy
plt.plot(history.history['loss'], linewidth=0.75, label='Loss')
plt.plot(history.history['mape'], linewidth=0.75, label='Mean Absolute
    Percentage Error')
plt.ylabel('Loss / Error')
plt.xlabel('Epoch')
plt.title('Mean Absolute Percentage Error')
plt.legend()
plt.savefig('Data_Analysis/code/data/DNN_Mean_Absolute_Percentage_Error.pdf')
plt.show()
```