FE 511: Introduction to Bloomberg & Thomson Reuters

Stock Market Prediction based on social media Sentiment Analysis

By Nishit Rao - 10434156

Prof. Agathe Sadeghi

TABLE OF CONTENTS

Introduction

Social media in today's date is no less than a weapon. It can be used to make friends or go to war. Due to the lack of instant verification, everything posted on social media, whether reliable or not, is considered as news and becomes widespread almost instantly. Consistent, informative, and insightful posting about your business is key to bringing you long-term, stable results in a volatile market. Some social media platforms have evolved into online trading clubs for gathering trade ideas, swapping tips, and inflating stocks while some have been hounded by hate clubs and trollers who can bring down the stock market by tweeting falsified information. This iterates the fact that the keyboard is most definitely mightier than the sword when Twitter's 280-character limit can cause share prices to rocket or plummet in a matter of hours.

We take a deep dive into one such case. It's the case of consumers tweeting reviews about a particular movie or tv show that they watched and how it affects the stock of the studio or company that produced this content. Here, we talk about Netflix. It is the second largest entertainment/media company by market capitalization as of February 2022. In the 2010s, Netflix was the top-performing stock in the S&P 500stock market index, with a total return of 3,693%. As of January 2023, Netflix had over 230 million subscribers worldwide, including 74.3 million in the United States and Canada; 76.7 million in Europe, the Middle East and Africa, 41.7 million in Latin America and 38 million in the Asia-Pacific region. The Netflix service is available worldwide aside from Mainland China, Syria, North Korea, and Russia.

Let's focus on one of their first hit tv shows after the pandemic hit which was Squid game. Squid Game is a South Korean survival drama series created by Hwang Dong-hyuk for Netflix. The series revolves around a secret contest where 456 players, all of whom are in deep financial hardship, risk their lives to play a series of deadly children's games for the chance to win a ₩45.6 billion (US$35 million, €33 million, or £29 million as of broadcast) reward. Squid Game was released worldwide on September 17, 2021, to critical acclaim and international attention. It is Netflix's most-watched series, becoming the top-viewed program in 94 countries and attracting more than 142 million member households and amassing 1.65 billion viewing hours during its first four weeks from launch.

Problem Statement

In an age of growing dependency on social media to get our daily news or information about or recommendation for OTT content we consume, does sentiments expressed on social media about a product affect the stock market od the company that produced it? Can tweets by influential people send market into a frenzy? If so, can we predict it to keep the damage to a minimum or capitalize on it for maximum growth?

Objective

- Data Collection : Collect a dataset of tweets from Twitter related to the show or movie being streamed on Netflix whose impact we want to check on the stock market. We can use the Twitter API to gather these tweets.

- Data Pre-processing: Filter out these tweets based on keywords and hashtags such that only relevant information is a part of the dataset. We can retain the username, date, location, and the text of the tweet. For deeper analysis we can retort to retaining likes, retweets, and check for verified status of the user.

- Sentiment Analysis : Every review tweeted about the movie or show is either a positive, negative, or neutral sentiment towards the content. We first define a list of keywords that are considered positive, and negative. This will help us determine the polarity of the tweet.

- Stock Market Data Collection: We need to collect historical stock market data for the past 3 years from Bloomberg terminal or WRDS. This will feature the daily open and close price and the volume and share price for Netflix over the course of 3 years.

- Feature Engineering: This involves combining the tweet data and the stock market data to create relevant features for the prediction model. Features like an average sentiment score based on the average polarity of tweets in a day and the stock price of the previous day.

- Prediction Model: Build a regression model such as regression or classification model that predicts the stock prices of Netflix based on the features created in the previous step.

- Model Evaluation: Evaluate the model's performance using appropriate metrics such as mean absolute error (MAE), root mean squared error (RMSE) and R-squared value.

- Deployment: Once the model is trained and tested, you can deploy it to a web application or integrate it with other tools to make real-time stock market predictions based on the tweets.

Dataset

Historical stock data of Netflix was extracted from WRDS. This data contains daily stock price for about 3 years from the year March 2020 to December 2022. During this period, Netflix saw a big boom in stock prices due to its hit tv shows like Squid Game, Bridgerton, Stranger Things, You, Wednesday and movies like Red Notice, Extraction, The Gray Man, Birdbox and Knives Out among others.

The selected stock markets' price data are collected from WRDS for the chosen period in csv file format. The downloaded data files have seven features—Date, Open, High, Low, Close, Volume, and Adjusted Close—which on a specific date, show the stock traded day, stock open price, stock maximum trading price, stock lowest trading price, stock closing price, number of shares traded, and closing price of a stock when dividends are paid to investors, respectively. In this project, only the Date, open price and Close price are used. Data was only collected on days where the stock market was open.

Data from twitter can be extracted by using Twitter.api by creating an account on the developer portal. You will require keys and tokens to get the ID for consumer keys and access keys.

In [4]:
```
pip install tweepy
```

```
Requirement already satisfied: tweepy in ./opt/anaconda3/lib/python3.9/site-packages (4.14.0)
Requirement already satisfied: requests<3,>=2.27.0 in ./opt/anaconda3/lib/python3.9/site-packages (from tweepy) (2.28.1)
Requirement already satisfied: requests-oauthlib<2,>=1.2.0 in ./opt/anaconda3/lib/python3.9/site-packages (from tweepy) (1.3.1)
Requirement already satisfied: oauthlib<4,>=3.2.0 in ./opt/anaconda3/lib/python3.9/site-packages (from tweepy) (3.2.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in ./opt/anaconda3/lib/python3.9/site-packages (from requests<3,>=2.27.0->tweepy) (1.26.11)
Requirement already satisfied: idna<4,>=2.5 in ./opt/anaconda3/lib/python3.9/site-packages (from requests<3,>=2.27.0->tweepy) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in ./opt/anaconda3/lib/python3.9/site-packages (from requests<3,>=2.27.0->tweepy) (2022.9.24)
Requirement already satisfied: charset-normalizer<3,>=2 in ./opt/anaconda3/lib/python3.9/site-packages (from requests<3,>=2.27.0->tweepy) (2.0.4)
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
import tweepy
from textblob import TextBlob
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from datetime import date, timedelta
```

In [6]:
```
consumer_key = "rLHemhvMIwtz9FW3NxLWcMjB2"
consumer_secret = 'BiMylD9lsSNTU2b67xJMTx1vtKIHmWA03hcCJ9hdFVm4eHJbLp'
access_key = '873052019992403968-IkKL1rzHzZHjrO32HdOyzM1bwhpPmHQ'
access_secret = 'delwIGSJ0zZhN7q3atYsmgyBB9U701PTOMciIGj17gVkn'
```

In [7]:
```
# Twitter authentication
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)

# Creating an API object
api = tweepy.API(auth)
```

Once we have the key and tokens, we can extract the tweets from Twitter.api depending on our requirement such as tweets from a particular user, date, location, or time. Here, we will be extracting tweets related hashtag 'Squid Game' from dates 03-01-2020 to 12-31-2022.

In [8]:
```
# Function to extract tweets
# Define search query and date range for tweet collection
query = '#SquidGame'
since_date = '2020-01-03'
until_date = '2022-31-12'

# Collect tweet data
tweet_data = []
for tweet in tweepy.Cursor(api.search_tweets, q=query, lang='en', since_id=since_date, until=until_date).items():
    tweet_data.append({
        'text': tweet.text,
        'created_at': tweet.created_at
    })

# 200 tweets to be extracted
number_of_tweets=2000
tweets = api.user_timeline(screen_name=username)

# Empty Array
tmp=[]

# create array of tweet information: username,
# tweet id, date/time, text
squidgame_tweets_csv = [tweet.text for tweet in tweets] # CSV file created
for j in squidgame_tweets_csv:

 # appending tweets to the empty array tmp
    tmp.append(j)

 # Printing the tweets
    print(tmp)
```

Loading the Dataset containing squid game tweets taken from Twitter which were stored in squidgame_tweets.csv

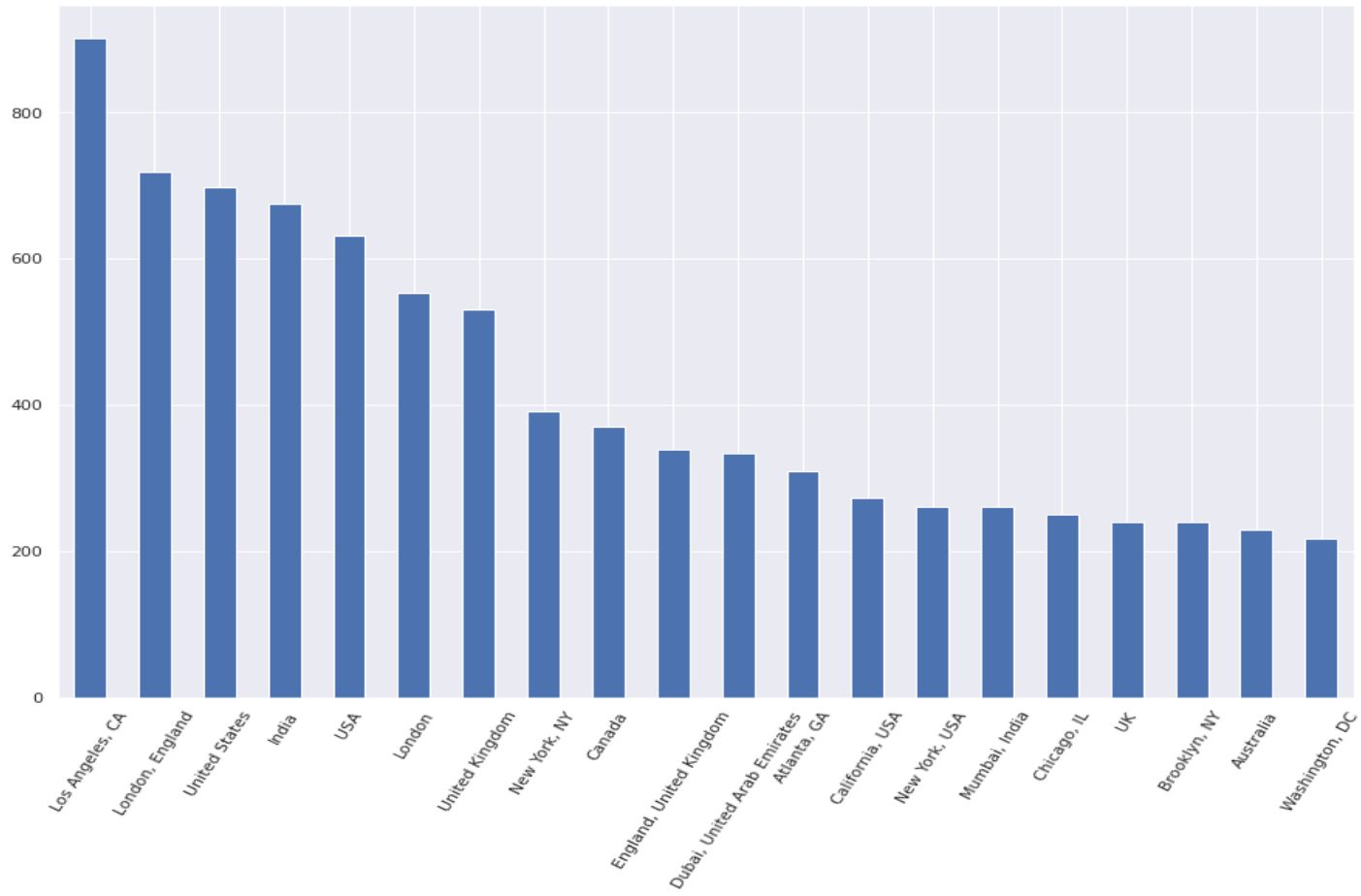
In [8]:
```
squidgame_tweets_df = pd.read_csv('/Users/nishitrao/Downloads/squidgame_tweets.csv',
                                  usecols=['date', 'user_location', 'text'])
squidgame_tweets_df
```

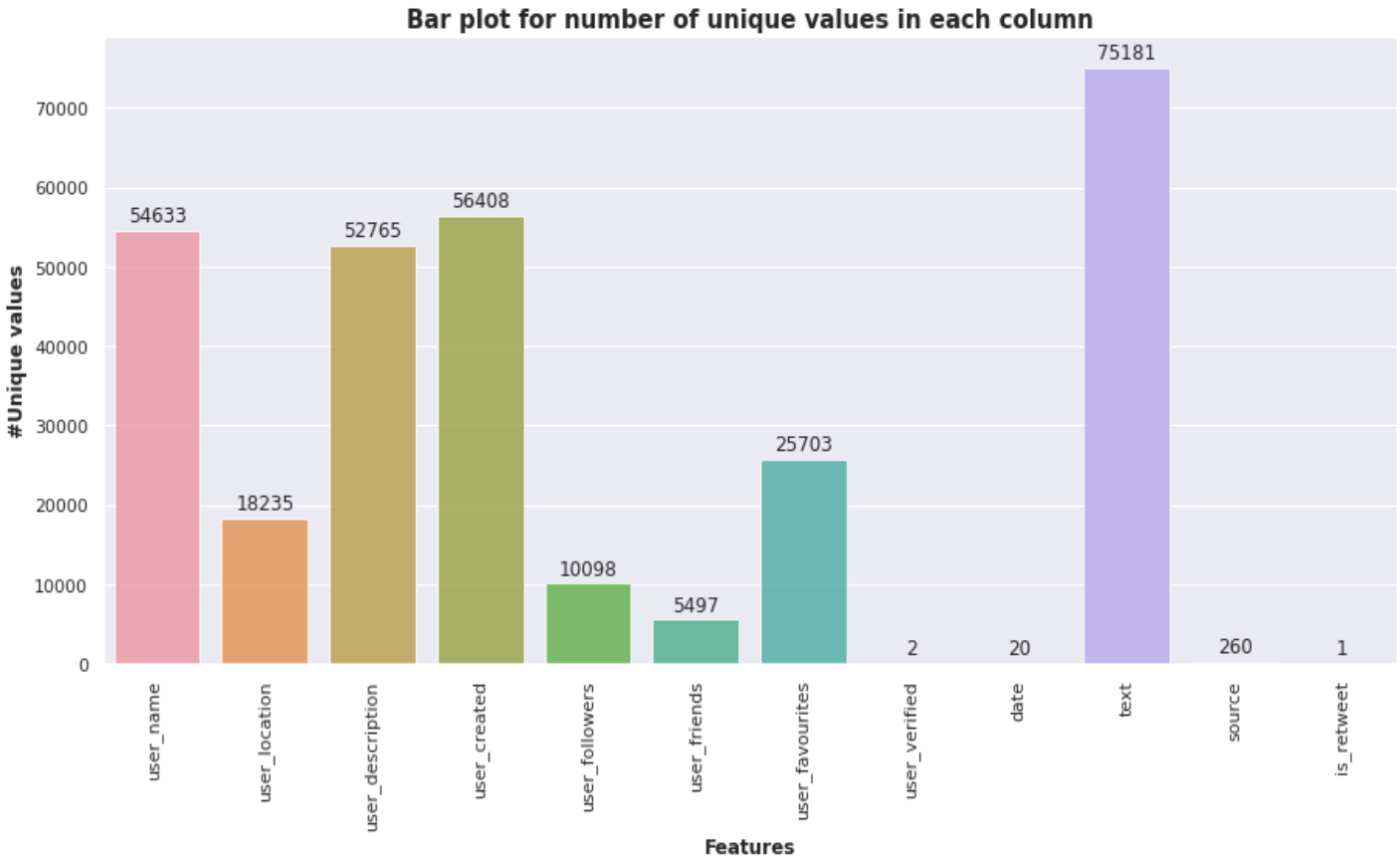
Out[8]:

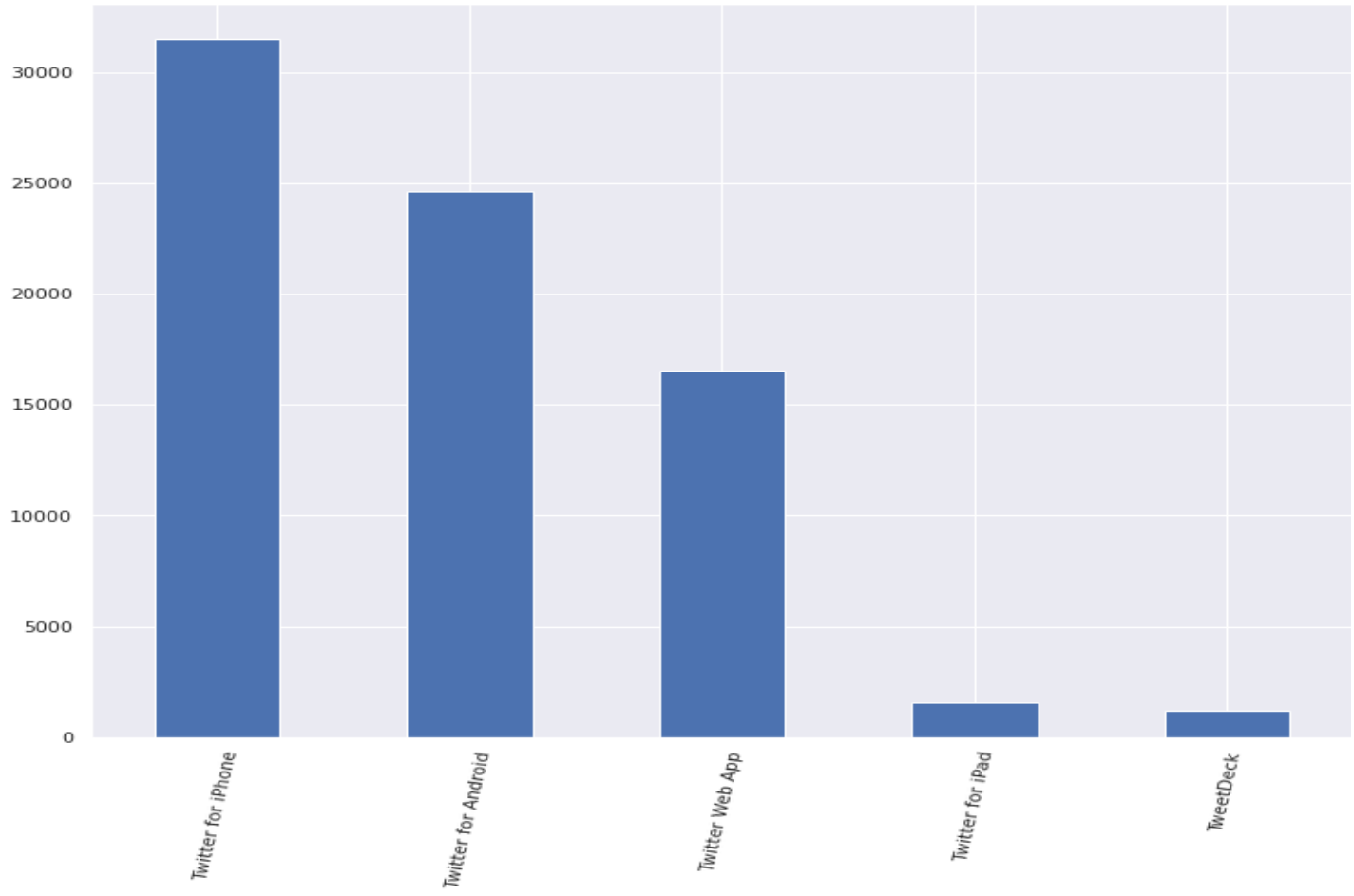
	user_location	date	text
0	Barcelona, Cataluña	2021-10-06 12:05:38+00:00	When life hits and the same time poverty strikes you\nGong Yoo : Lets play a game \n#SquidGame #...
1	New York, NY	2021-10-06 12:05:22+00:00	That marble episode of #SquidGame ruined me. 😭😭😭
2	California	2021-10-06 12:05:22+00:00	#Squidgame time
3	Dallas, TX	2021-10-06 12:05:04+00:00	//Blood on 1st slide\nI'm joining the squidgame thing, I'm already dead by sugar honeycomb ofc\n...
4	France	2021-10-06 12:05:00+00:00	The two first games, players were killed by the mask guys ; the bloody night and the third game,...
...	...	...	...
80014	USA	2021-10-28 13:46:26+00:00	Yes Yes Yes\n\n@ArianaGrande\n#Squidgame\n https://t.co/BQgwd91ODK #Outerwear #Product #Neck #Sl...
80015	Chicago, IL	2021-10-28 13:45:38+00:00	SQUID GAME was reviewed on RevAAA #squidgame \n\nReview Anything Anyone Anywhere https://t.co/zj...
80016	Karachi Pakistan	2021-10-28 13:45:00+00:00	Back & Forth Between ‘Squid Game’ Creator & Lebron James Have Gotten The Attention Of Ne...
80017	Halloweentown 🎃	2021-10-28 13:44:57+00:00	So what sort of games do you think they'll play when they inevitably make #SquidGame 2? #squidga...
80018	London	2021-10-28 13:44:38+00:00	@venancio_taylor @tracklist cala a boca swifter \n\n#gfvip #tzvip #rosmello #prelemi #gregorelli...

80019 rows × 3 columns

Data Visualization




Bar plot for number of unique values in each column
#Unique values
Features
0
10000
20000
30000
40000
50000
60000
70000
54633
18235
52765
56408
10098
5497
25703
2
20
75181
260
1
user_name
user_location
user_description
user_created
user_followers
user_friends
user_favourites
user_verified
date
text
source
is_retweet


0
5000
10000
15000
20000
25000
30000
Twitter for iPhone
Twitter for Android
Twitter Web App
Twitter for iPad
TweetDeck

Data visualization gives us a lot of important information about the dataset at first glance. The data tells us that Twitter from iPhone produced most tweets, LA in California, USA had the highest number of users tweet about Squid Game. Through data visualization, we also realized that there is a lot of data which is null or not appropriate for feeding to a ML model. It gives us insight into features that are required to build the best possible ML prediction model. Feature extraction can be done by pre-processing the dataset where we filter out the best features. From the above, we understand the most effective features are user location, username, date, and tweet.

Data Pre-processing

Each of the tweets will be preprocessed by running a function with the following guidelines. The function will then transform the data. This preprocessing process involves Lemmatization, Removing Keywords, and Removing Short words. These steps were added as they better allow for the data to be preprocessed for sentimental analysis. On visualization we understood that the dataset had a lot of null values, hence we need to pre-process the data and make it legitimate for the ML model to analyze.

The pre-processing guidelines are as follows:

1) Lower Casing: Each text is converted to lowercase.

2) Replacing URLs: Links starting with “Http” or “https”, or “www” are replaced by “URL”.

3) Replacing Emojis: Replace emojis by using a pre-defined dictionary containing emojis along with their meaning. (e.g.: “:)” to “EMOJI smile”)

4) We are removing Non-Alphabets: Replacing characters except Digits and Alphabets with space.

5) Removing Consecutive letters: 3 or more consecutive letters are replaced by two letters. (e.g.: “Heyyyy” to “Hey”)

6) Removing Short words: Words with a length of less than two are eliminated.

7) Removing Stop words: Stop words are the English words that do not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. (e.g.: “the”, “he”, “have”)

8) Lemmatizing: Lemmatization is the process of converting a word to its base form. (e.g., “Great” to “Good”)

The idea is to convert all the data into textual format which can be understood by our ML model for determining its polarity as positive, negative, or neutral and giving it a score accordingly.

Once we are done with this step, we move on towards sentiment analysis.

In [11]:
```
# Pre-process tweet data
def preprocess_tweet(tweet_text):
    # Remove URLs, hashtags, and usernames
    tweet_text = ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)",
                                 " ", tweet_text).split())

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(tweet_text)
    filtered_tweet = [word for word in word_tokens if word.lower() not in stop_words]


    # Perform stemming or lemmatization
    lemmatizer = WordNetLemmatizer()
    stemmed_tweet = [lemmatizer.lemmatize(word) for word in filtered_tweet]
    return ' '.join(stemmed_tweet)
```

Sentiment Analysis and Feature Engineering

Twitter sentiment analysis analyzes the sentiment or emotion of tweets. It uses natural language processing and machine learning algorithms to classify tweets automatically as positive, negative, or neutral based on their content. It can be done for individual tweets, or a larger dataset related to a particular topic or event. It can be used for crisis management in most companies and to understand their audience better.

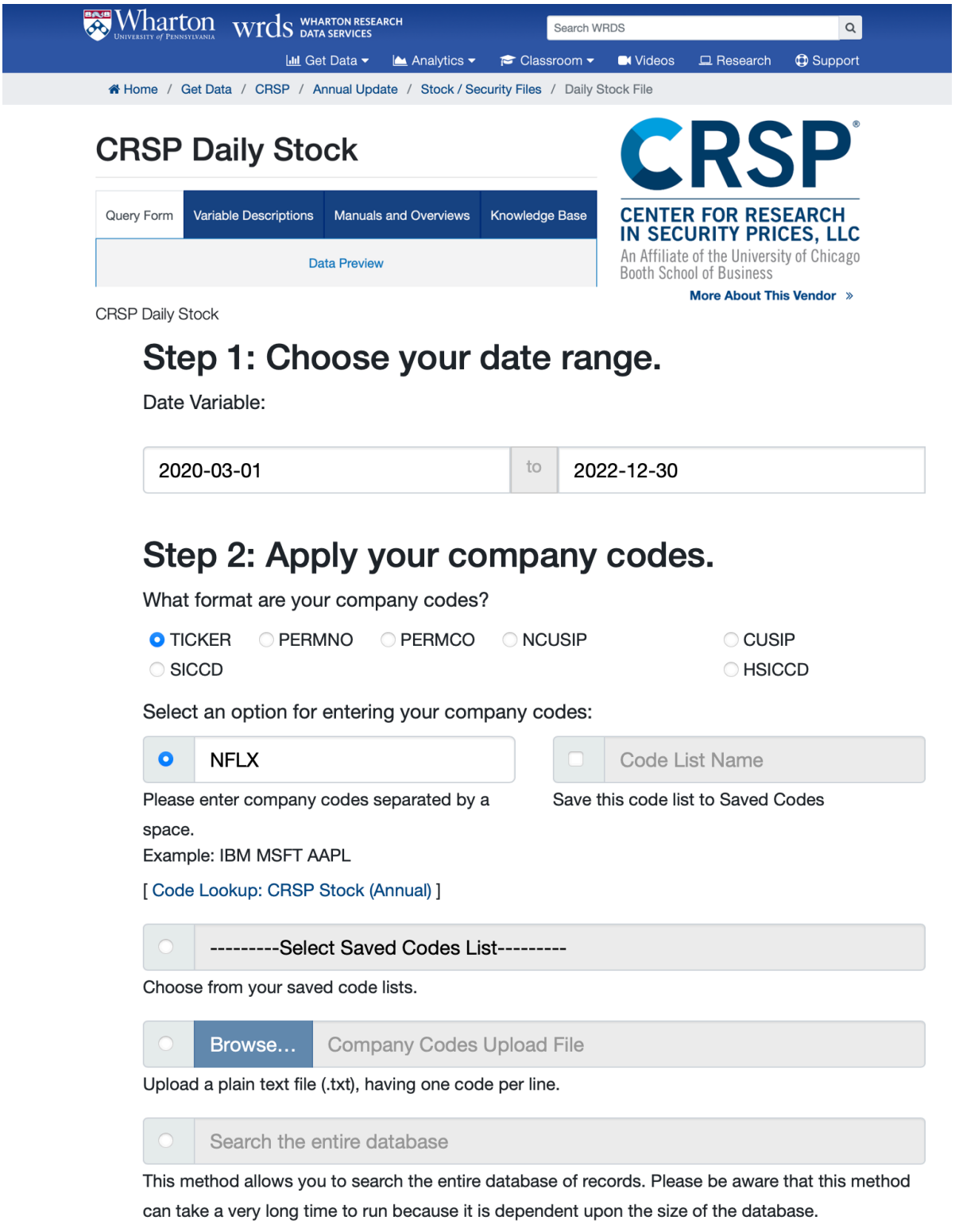
In [13]:
```
# Perform sentiment analysis
def get_tweet_sentiment(tweet_text):
    analysis = TextBlob(preprocess_tweet(tweet_text))
    return analysis.sentiment.polarity


sentiment = SentimentIntensityAnalyzer()
df['positive'] = [sentiment.polarity_scores(a)['pos'] for a in df['text']]
df['negative'] = [sentiment.polarity_scores(a)['neg'] for a in df['text']]
df['neutral'] = [sentiment.polarity_scores(a)['neu'] for a in df['text']]


df = df[['text', 'positive', 'negative', 'neutral']]
df.head()
```

	text	positive	negative	neutral
0	life hit time poverti strike yougong yoo let ...	0.173	0.108	0.719
1	marbl episod squidgam ruin 😭😭😭	0.000	0.487	0.513
2	squidgam time	0.000	0.000	1.000
3	blood slideim join squidgam thing im alreadi ...	0.142	0.277	0.581
4	two first game player kill mask guy bloodi ni...	0.000	0.461	0.539

After getting the compound score for each tweet, we grouped the dates together and averaged the compound scores for that day. As you can see, the average compound scores each day is a number ranging from -1 to 1. With the negative scores representing an overall negative sentiment on Twitter for that day and vice-versa. The resulting DF should look similar to this:

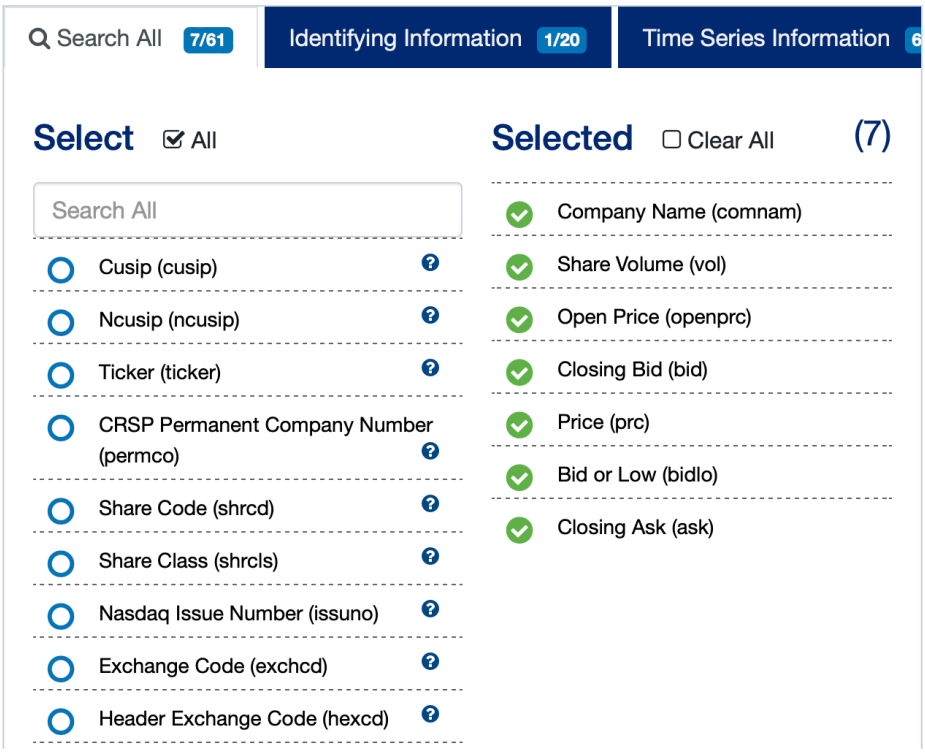
	date	sentiment
0	2021-08-20	0.104813
1	2021-08-21	0.178620
2	2021-08-22	0.152940
3	2021-08-23	0.266215
4	2021-08-24	0.202516
5	2021-08-25	0.194975
6	2021-08-26	0.038705
7	2021-08-27	0.178311
8	2021-08-28	0.188372
9	2021-08-29	0.208930
10	2021-08-30	-0.025710

Extraction of Historical Stock Market Data of Netflix:



Step 3: Choose query variables.

How does this work?



Step 4: Select query output.

How does this work?

Select the desired format of the output file. For large data requests, select a compression type to expedite downloads. If you enter your email address, you will receive an email that contains a URL to the output file when the data request is finished processing.

Output Format

- (•) comma-delimited text (*.csv)
- ( ) Excel spreadsheet (*.xlsx)
- ( ) tab-delimited text (*.txt)
- ( ) HTML table (*.htm)
- ( ) SAS Windows_64 dataset (*.sas7bdat)
- ( ) STATA file (*.dta)

Compression Type

- (•) Uncompressed
- ( ) zip (*.zip)
- ( ) gzip (*.gz)

Date Format

- (•) YYYY-MM-DD. (e.g. 1984-07-25)
- ( ) MM/DD/YYYY. (e.g. 07/25/1984)
- ( ) DD/MM/YYYY. (e.g. 25/07/1984)

E-Mail Address *(Optional)*

7@stevens.edu Edit Preferences

Custom Field *(Optional)*

Custom Field

Save This Query *(Optional)*

[ ] Saved Query Name

Notes on this Query *(Optional)*

Saved Query Notes

Submit Form

Netflix stock market data

```
In [5]: import pandas as pd
        from matplotlib import pyplot as plt
```

```
In [3]: #Read the csv file
        df = pd.read_csv('/Users/nishitrao/Downloads/NFLX.csv')
        df
```

Out[3]:

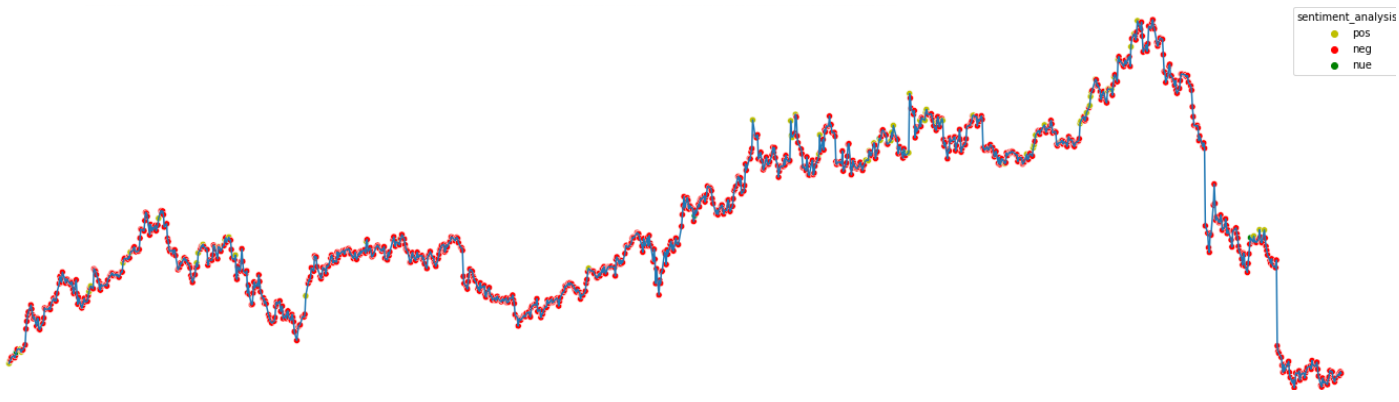
	Date	Closing Price	Volume	Open Price	Closing Price 1 Day Ago	High Price	Low Price
0	10-04-2023	338.99	2650954	335.27	339.33	339.88	333.3600
1	06-04-2023	339.33	4660542	339.34	342.35	340.48	332.6300
2	05-04-2023	342.35	4205545	345.30	346.75	345.43	336.2526
3	04-04-2023	346.75	3298072	348.49	348.28	349.80	343.9500
4	03-04-2023	348.28	4413650	341.83	345.48	348.58	340.4001
...	...	...	...	...	...	...	...
764	26-03-2020	362.99	7235016	344.00	342.39	363.84	341.7300
765	25-03-2020	342.39	8767171	361.02	357.32	362.00	339.1700
766	24-03-2020	357.32	11638683	369.99	360.27	372.93	353.0301
767	23-03-2020	360.27	13449378	347.89	332.83	366.11	340.8838
768	20-03-2020	332.83	10853846	342.31	332.03	350.49	332.0000

769 rows × 7 columns

Understanding Time-series

We will track the data set over a time period of 3 years and how the stocks fluctuate.

```
In [12]: import seaborn as sns
         plt.figure(figsize=(25,7));
         sns.lineplot(x=df["date"],y=df["Adj Close"])
         df['sentiment_analysis']=df['P_mean']
         df['sentiment_analysis']=df['sentiment_analysis'].apply(lambda x: 'pos' if x>0 else 'neu' if x==0 else 'neg')
         sns.scatterplot(x=df["date"],y=df['Adj Close'],hue=df['sentiment_analysis'],palette=['y','r','g'])
         plt.xticks(rotation=45);
         plt.title("Stock market of Netfilx from Mar-2020 to Dec-2022",fontsize=16);
```



Prediction Model

ARIMA Model

In [17]:
```
!pip install pmdarima==1.8.1
```

```
Collecting pmdarima==1.8.1
  Downloading pmdarima-1.8.1-cp39-cp39-macosx_10_15_x86_64.whl (601 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 601.6/601.6 kB 12.8 MB/s eta 0:00:00a 0:00:01
Requirement already satisfied: urllib3 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdarima==1.8.1) (1.26.1
1)
Requirement already satisfied: scipy>=1.3.2 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdarima==1.8.1) (1.
9.1)
Requirement already satisfied: scikit-learn>=0.22 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdarima==1.8.
1) (1.0.2)
Requirement already satisfied: pandas>=0.19 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdarima==1.8.1) (1.
4.4)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdar
ima==1.8.1) (63.4.1)
Requirement already satisfied: numpy>=1.17.3 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdarima==1.8.1)
(1.23.5)
Requirement already satisfied: joblib>=0.11 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdarima==1.8.1) (1.
1.0)
Requirement already satisfied: Cython!=0.29.18,>=0.29 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdarima==
1.8.1) (0.29.32)
Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in ./opt/anaconda3/lib/python3.9/site-packages (from pmdari
ma==1.8.1) (0.13.2)
Requirement already satisfied: python-dateutil>=2.8.1 in ./opt/anaconda3/lib/python3.9/site-packages (from pandas>=0.
19->pmdarima==1.8.1) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./opt/anaconda3/lib/python3.9/site-packages (from pandas>=0.19->pmdari
ma==1.8.1) (2022.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./opt/anaconda3/lib/python3.9/site-packages (from scikit-learn
>=0.22->pmdarima==1.8.1) (2.2.0)
Requirement already satisfied: packaging>=21.3 in ./opt/anaconda3/lib/python3.9/site-packages (from statsmodels!=0.1
2.0,>=0.11->pmdarima==1.8.1) (21.3)
Requirement already satisfied: patsy>=0.5.2 in ./opt/anaconda3/lib/python3.9/site-packages (from statsmodels!=0.12.0,
>=0.11->pmdarima==1.8.1) (0.5.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in ./opt/anaconda3/lib/python3.9/site-packages (from packagin
g>=21.3->statsmodels!=0.12.0,>=0.11->pmdarima==1.8.1) (3.0.9)
Requirement already satisfied: six in ./opt/anaconda3/lib/python3.9/site-packages (from patsy>=0.5.2->statsmodels!=0.
12.0,>=0.11->pmdarima==1.8.1) (1.16.0)
Installing collected packages: pmdarima
Successfully installed pmdarima-1.8.1
```

Train test data for ARIMA

In [18]:
```
df_arima=df_for_training.copy()
# df_arima['Volume']=df['Volume']
n_past_for_arima=1
adj_close_feature=df_arima['Adj Close']
open_feature=df_arima['Open']
df_arima['Adj Close']=df_arima['Adj Close'].shift(-n_past_for_arima)
df_arima['Open']=df_arima['Open'].shift(-n_past_for_arima)
df_arima.dropna(inplace=True)
df_arima['Adj Close_feature']=adj_close_feature.iloc[:-1]
df_arima['open_feature_feature']=open_feature.iloc[:-1]
# df_arima=df_arima.reindex(columns=['open_lag','High', 'Low', 'Close', 'adj_close_lag','P_mean','Adj Close','Open',])
X_arima=df_arima.drop(['Adj Close','Open'],axis=1)
y_arima=df_arima[['Open','Adj Close']]
X_train_arima_twitter, X_test_arima_twitter, y_train_arima, y_test_arima=train_test_split(X_arima, y_arima, test_size=0

X_train_arima_without_twitter, X_test_arima_without_twitter= X_train_arima_twitter.drop('P_mean', axis=1), X_test_arima
X_train_arima_without_twitter
```

	High	Low	Close	Volume	Adj Close_feature	open_feature_feature
date						
2021-08-09	522.669983	517.989990	519.969971	1367800.0	515.840027	520.000000
2021-08-10	520.789978	512.969971	515.840027	1960500.0	512.400024	517.000000
2021-08-11	519.570007	509.769989	512.400024	1673900.0	510.720001	511.859985
2021-08-12	513.000000	507.200012	510.720001	1685700.0	515.919983	512.640015
2021-08-13	521.440002	511.510010	515.919983	2177700.0	517.919983	515.239990

897 rows × 6 columns

In [19]:
```
X_train_arima_twitter
```

	High	Low	Close	Volume	P_mean	Adj Close_feature	open_feature_feature
date							
2021-08-09	522.669983	517.989990	519.969971	1367800.0	-0.051282	515.840027	520.000000
2021-08-10	520.789978	512.969971	515.840027	1960500.0	-0.085427	512.400024	517.000000
2021-08-11	519.570007	509.769989	512.400024	1673900.0	-0.058824	510.720001	511.859985
2021-08-12	513.000000	507.200012	510.720001	1685700.0	-0.023041	515.919983	512.640015
2021-08-13	521.440002	511.510010	515.919983	2177700.0	-0.082237	517.919983	515.239990

897 rows × 7 columns

Building an ARIMA model for open price over the last 3 years.

In [20]:
```
from pmdarima import auto_arima
arima_model_for_open_without_twitter = auto_arima(y_train_arima['Open'], exogenous=X_train_arima_without_twitter,
                                                  start_p=2, d=None, start_q=3, max_p=5, max_d=3, max_q=5, D=None, star
                                                  seasonal=True,
                                                  error_action='ignore',
                                                  suppress_warnings=True,
                                                  stepwise=True
                                                  )

arima_model_for_open_twitter = auto_arima(y_train_arima['Open'], exogenous=X_train_arima_twitter,
                                          start_p=2, d=None, start_q=3, max_p=5, max_d=3, max_q=5, D=None, start_Q=1, m
                                          seasonal=True,
                                          error_action='ignore',
                                          suppress_warnings=True,
                                          stepwise=True
                                          )
arima_model_for_open_without_twitter.summary()
arima_model_for_open_twitter.summary()
```

SARIMAX Results

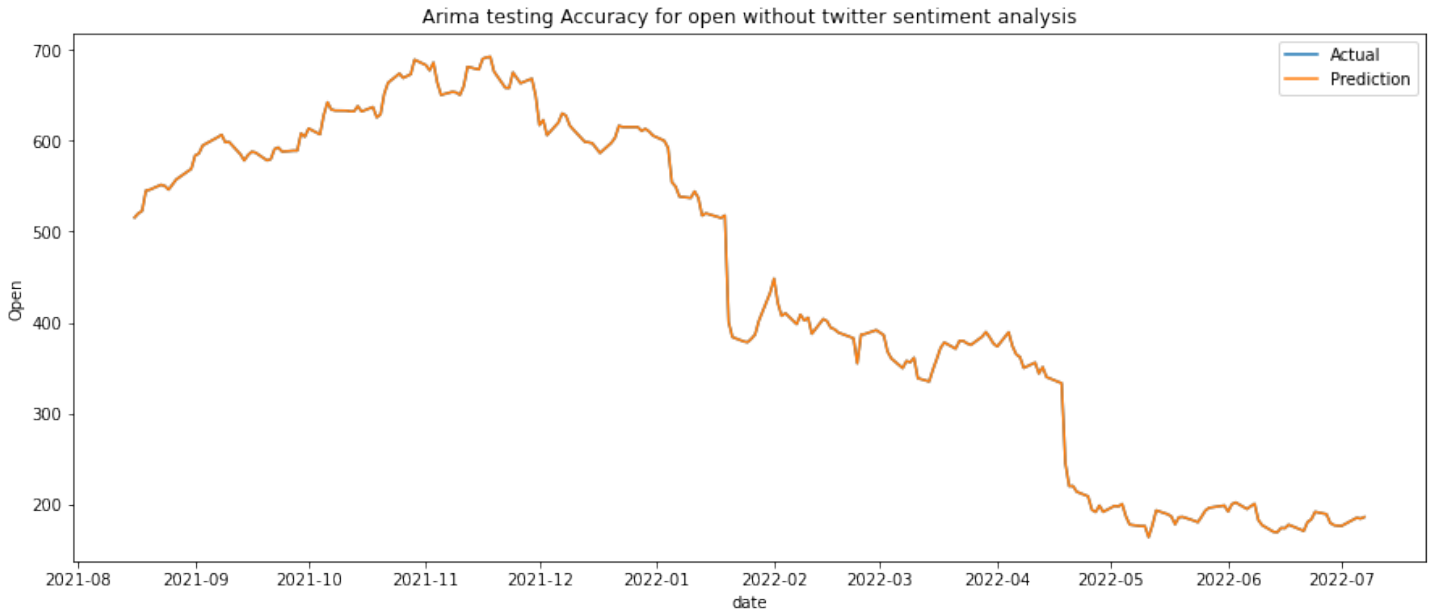
Dep. Variable:	y	No. Observations:	897
Model:	SARIMAX(0, 1, 0)	Log Likelihood	9481.618
Date:	Mon, 29 Aug 2022	AIC	-18947.237
Time:	09:43:56	BIC	-18908.853
Sample:	0	HQIC	-18932.571
	- 897		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
High	-7.286e-16	0.000	-2.5e-12	1.000	-0.001	0.001
Low	1.2e-15	0.001	9.7e-13	1.000	-0.002	0.002
Close	2.776e-17	0.001	2.99e-14	1.000	-0.002	0.002
Volume	-8.703e-17	9.49e-10	-9.17e-08	1.000	-1.86e-09	1.86e-09
P_mean	2.17e-14	1.58e-06	1.38e-08	1.000	-3.09e-06	3.09e-06
Adj Close_feature	1.336e-16	0.001	1.74e-13	1.000	-0.002	0.002
open_feature_feature	1.0000	0.001	1411.750	0.000	0.999	1.001
sigma2	1e-10	6.81e-11	1.468	0.142	-3.35e-11	2.34e-10

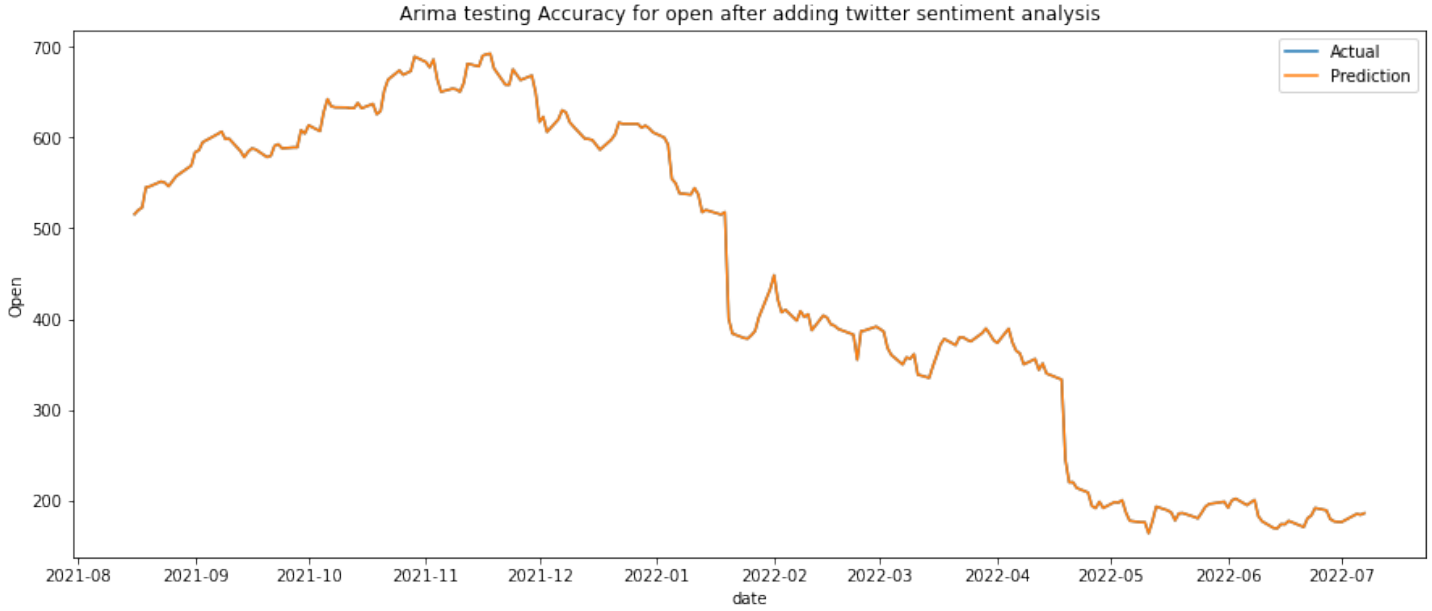
Ljung-Box (L1) (Q):	48.88	Jarque-Bera (JB):	12453.04
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.42	Skew:	0.10
Prob(H) (two-sided):	0.00	Kurtosis:	21.26

Finding mean squared error by computing testing and training accuracy for open price.

In [16]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
test_prediction_without_twitter= arima_model_for_open_without_twitter.predict(n_periods=len(X_test_arima_without_twitte
test_prediction_twitter= arima_model_for_open_twitter.predict(n_periods=len(X_test_arima_twitter), exogenous=X_test_ari
plt.figure(figsize=(15,6))
sns.lineplot(x=y_test_arima['Open'].index, y=y_test_arima['Open'], label='Actual')
sns.lineplot(x=y_test_arima['Open'].index, y=test_prediction_without_twitter, label='Prediction')
plt.title('Arima testing Accuracy for open without twitter sentiment analysis')
plt.show()
e=mean_squared_error(test_prediction_without_twitter, y_test_arima['Open'])
print(f'Testing mean square error for open feature without twitter sentiment analysis {e}')

plt.figure(figsize=(15,6))
sns.lineplot(x=y_test_arima['Open'].index, y=y_test_arima['Open'], label='Actual')
sns.lineplot(x=y_test_arima['Open'].index, y=test_prediction_twitter, label='Prediction')
plt.title('Arima testing Accuracy for open after adding twitter sentiment analysis')
plt.show()
e=mean_squared_error(test_prediction_twitter, y_test_arima['Open'])
print(f'Testing mean square error for open feature with twitter sentiment analysis {e}')
```



Testing mean square error for open feature without twitter sentiment analysis 2.94 49940298976172e-18



Testing mean square error for open feature with twitter sentiment analysis 1.14708 25568056457e-18

Building an ARIMA model for adjusted close price over the last 3 years.

In [21]:
```
from pmdarima import auto_arima
arima_model_for_Adj_Close_without_twitter = auto_arima(y_train_arima['Adj Close'], exogenous=X_train_arima_without_twit
                                                       start_p=2, d=None, start_q=3, max_p=5, max_d=3, max_q=5, D=None, star
                                                       seasonal=True,
                                                       error_action='ignore',
                                                       suppress_warnings=True,
                                                       stepwise=True
                                                       )

arima_model_for_Adj_Close_twitter = auto_arima(y_train_arima['Adj Close'], exogenous=X_train_arima_twitter,
                                               start_p=2, d=None, start_q=3, max_p=5, max_d=3, max_q=5, D=None, start_Q=1, m
                                               seasonal=True,
                                               error_action='ignore',
                                               suppress_warnings=True,
                                               stepwise=True
                                               )
arima_model_for_Adj_Close_without_twitter.summary()
arima_model_for_Adj_Close_twitter.summary()
```

SARIMAX Results

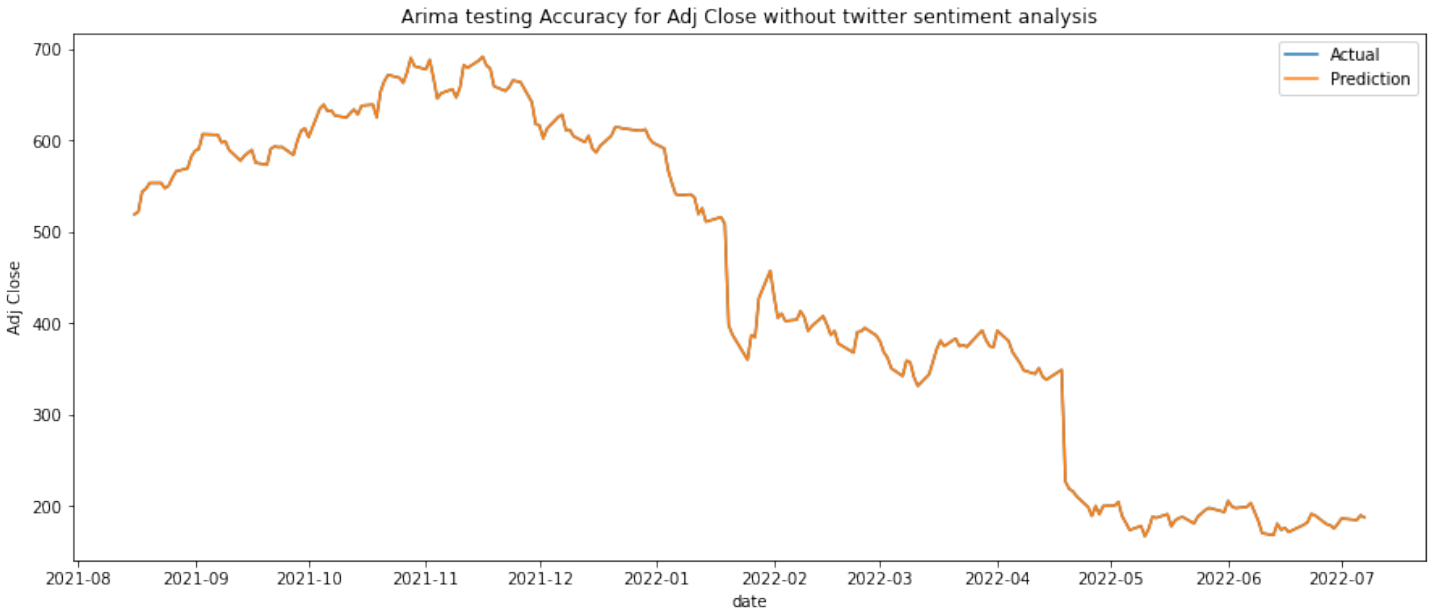
Dep. Variable:	y	No. Observations:	897
Model:	SARIMAX(0, 1, 0)	Log Likelihood	9481.618
Date:	Mon, 29 Aug 2022	AIC	-18947.237
Time:	09:44:07	BIC	-18908.853
Sample:	0	HQIC	-18932.571
	- 897		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
High	4.788e-16	6.27e-16	0.764	0.445	-7.5e-16	1.71e-15
Low	-6.939e-18	3.45e-15	-0.002	0.998	-6.77e-15	6.76e-15
Close	-2.88e-16	3.78e-16	-0.762	0.446	-1.03e-15	4.52e-16
Volume	8.506e-18	5.26e-09	1.62e-09	1.000	-1.03e-08	1.03e-08
P_mean	4.219e-14	5.34e-20	7.9e+05	0.000	4.22e-14	4.22e-14
Adj Close_feature	1.0000	3.12e-17	3.21e+16	0.000	1.000	1.000
open_feature_feature	-3.747e-16	2.4e-15	-0.156	0.876	-5.07e-15	4.32e-15
sigma2	1e-10	6.79e-11	1.474	0.141	-3.3e-11	2.33e-10

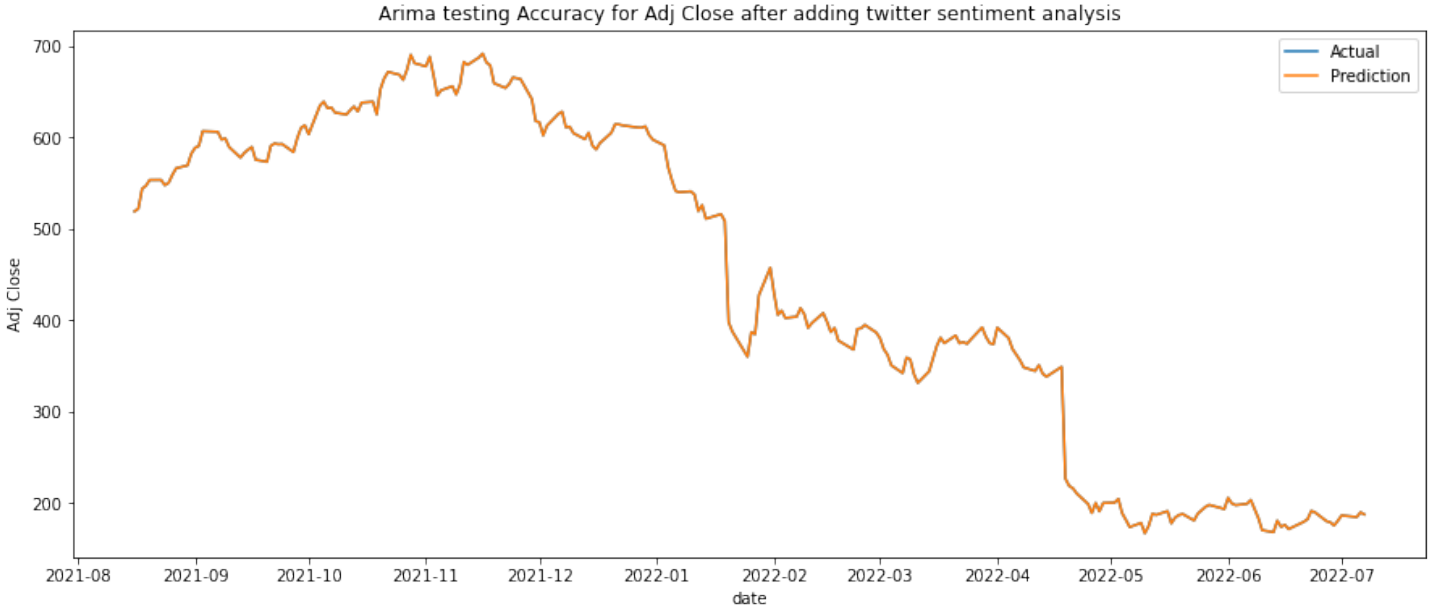
Ljung-Box (L1) (Q):	48.85	Jarque-Bera (JB):	12430.42
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.42	Skew:	-0.10
Prob(H) (two-sided):	0.00	Kurtosis:	21.25

Finding mean squared error by computing testing and training accuracy for adjusted close price.

```
In [22]: from sklearn.metrics import mean_squared_error,mean_absolute_error
train_prediction_without_twitter= arima_model_for_Adj_Close_without_twitter.predict(n_periods=len(X_train_arima_without
train_prediction_twitter= arima_model_for_Adj_Close_twitter.predict(n_periods=len(X_train_arima_twitter), exogenous=X_t
plt.figure(figsize=(15,6))
sns.lineplot(x=y_train_arima['Adj Close'].index, y=y_train_arima['Adj Close'], label='Actual')
sns.lineplot(x=y_train_arima['Adj Close'].index, y=train_prediction_without_twitter, label='Prediction')
plt.title('Arima training Accuracy for Adj Close without twitter sentiment analysis')
plt.show()
e=mean_squared_error(train_prediction_without_twitter, y_train_arima['Adj Close'])
print(f'Training mean absolute error for Adj Close feature without twitter {e}')

plt.figure(figsize=(15,6))
sns.lineplot(x=y_train_arima['Adj Close'].index, y=y_train_arima['Adj Close'], label='Actual')
sns.lineplot(x=y_train_arima['Adj Close'].index, y=train_prediction_twitter, label='Prediction')
plt.title('Arima training Accuracy for Adj Close after adding twitter sentiment analysis')
plt.show()
e=mean_squared_error(train_prediction_twitter, y_train_arima['Adj Close'])
print(f'Training mean absolute error for Adj Close feature with twitter {e}')
```



Testing absolute mean square error for Adj Close feature without twitter sentiment analysis 1.0145387129955273e-20



Testing absolute mean square error for Adj Close feature with twitter sentiment an alysis 1.0938434385844854e-20

In [24]:
```
test_prediction_twitter
```

```
array([518.909973, 521.869995, 543.710022, 546.880005, 553.330017,
       553.409973, 547.580017, 550.119995, 558.919983, 566.179993,
       569.190002, 582.070007, 588.549988, 590.530029, 606.710022,
       606.049988, 597.539978, 598.719971, 589.289978, 577.76001 ,
       582.869995, 586.5     , 589.349976, 575.429993, 573.140015,
       590.650024, 593.26001 , 592.390015, 592.640015, 583.849976,
       599.059998, 610.340027, 613.150024, 603.349976, 634.809998,
       639.099976, 631.849976, 632.659973, 627.039978, 624.940002,
       629.76001 , 633.799988, 628.289978, 637.969971, 639.      ,
       625.140015, 653.159973, 664.780029, 671.659973, 668.52002 ,
       662.919983, 674.049988, 690.309998, 681.169983, 677.719971,
       688.289978, 668.400024, 645.719971, 651.450012, 655.98999 ,
       646.909973, 657.580017, 682.609985, 679.330017, 687.400024,
       691.690002, 682.02002 , 678.799988, 659.200012, 654.059998,
       658.289978, 665.640015, 663.840027, 641.900024, 617.77002 ,
       616.469971, 602.130005, 612.690002, 625.580017, 628.080017,
       611.      , 611.659973, 604.559998, 597.98999 , 605.039978,
       591.059998, 586.72998 , 593.73999 , 604.919983, 614.23999 ,
       614.090027, 613.119995, 610.710022, 610.539978, 612.090027,
       602.440002, 597.369995, 591.150024, 567.52002 , 553.289978,
       541.059998, 539.849976, 540.840027, 537.219971, 519.200012,
       525.690002, 510.799988, 515.859985, 508.25    , 397.5     ,
       387.149994, 366.420013, 359.700012, 386.700012, 384.359985,
       427.140015, 457.130005, 429.480011, 405.600006, 410.170013,
       402.100006, 403.529999, 412.890015, 406.269989, 391.309998,
       396.570007, 407.459991, 398.079987, 386.670013, 391.290009,
       602.440002, 597.369995, 591.150024, 567.52002 , 553.289978,
       541.059998, 539.849976, 540.840027, 537.219971, 519.200012,
       525.690002, 510.799988, 515.859985, 508.25    , 397.5     ,
       387.149994, 366.420013, 359.700012, 386.700012, 384.359985,
       427.140015, 457.130005, 429.480011, 405.600006, 410.170013,
       402.100006, 403.529999, 412.890015, 406.269989, 391.309998,
       396.570007, 407.459991, 398.079987, 386.670013, 391.290009,
       377.380005, 367.459991, 390.029999, 390.799988, 394.519989,
       386.23999 , 380.029999, 368.070007, 361.730011, 350.26001 ,
       341.76001 , 358.790009, 356.769989, 340.320007, 331.01001 ,
       343.75    , 357.529999, 371.399994, 380.600006, 374.589996,
       382.920013, 374.48999 , 375.709991, 373.850006, 378.51001 ,
       391.820007, 381.470001, 374.589996, 373.470001, 391.5     ,
       380.149994, 368.350006, 362.149994, 355.880005, 348.      ,
       344.100006, 350.429993, 341.130005, 337.859985, 348.609985,
       226.190002, 218.220001, 215.520004, 209.910004, 198.399994,
       188.539993, 199.520004, 190.360001, 199.460007, 199.869995,
       204.009995, 188.320007, 180.970001, 173.100006, 177.660004,
       166.369995, 174.309998, 187.639999, 186.509995, 190.559998,
       177.190002, 183.479996, 186.350006, 187.440002, 180.339996,
       187.830002, 191.399994, 195.190002, 197.440002, 192.910004,
       205.089996, 198.979996, 197.139999, 198.610001, 202.830002,
       192.770004, 182.940002, 169.690002, 167.539993, 180.110001,
       173.350006, 175.509995, 170.910004, 178.889999, 181.710007,
       190.850006, 189.139999, 179.600006, 178.360001, 174.869995,
```

Model Evaluation

Randomly picking a date to validate our ARIMA model and check its accuracy

In [25]:
```
x_forecast=X_arima[X_arima.index =='2022-01-01']
x_forecast


x_forecast=X_arima[X_arima.index =='2022-01-01']
open_prediction = arima_model_for_open_twitter.predict(n_periods=len(x_forcast), exogenous=x_forcast)
adj_close_prediction = arima_model_for_Adj_Close_twitter.predict(n_periods=len(x_forcast),exogenous=x_forcast)#shape =
# open_prediction=float(open_prediction)
# adj_close_prediction=float(adj_close_prediction)
# Convert timestamp to date
forecast_dates = []
for time_i in predict_period_dates:
    forecast_dates.append(time_i.date())

print(f'Date = {forecast_dates[-2]}, Prediction open {open_prediction[0]}')
print(f'Date = {forecast_dates[-2]}, Prediction Adjusted close {adj_close_prediction[0]}')
```

High	Low	Close	Volume	P_mean	Adj Close_feature	open_feature_feature
190.210007	183.5	189.270004	6334500.0	-0.055427	186.979996	186.020004

Predicted open price for 2022-01-01 is 186.020004 and Predicted close price is 186.979996

The actual open price on 2022-01-01 was 184.27 and actual adjusted close price was 189.27

From this we can determine the accuracy of our model is pretty good.

Conclusion and Future Work

We can conclude that stock market does get affected by the tweets on social media. Twitter impacts the stock market and stock of the companies about which the tweet is. This could be understood using our prediction model. LSTM and ARIMA were the two prediction models we used to determine the testing and validation accuracy. we compared the actual stock price with the predicted stock price for both models on opening and closing day with and without the Twitter Sentiment Analysis. We realize that ARIMA gave us better results which were more accurate and closer to the actual stock price over that period of time.

In the future we can also test the same model for a different set of tweets such as for different shows that we previously mentioned like Bridgerton, Wednesday, You, Stranger Things etc. by changing the hashtag in the query. We can also test this for different streaming websites such as Amazon Prime, HBO, Disney, Hulu, Apple TV and for their shows and how these streaming websites have affected the parent company's stock prices. This model can also be used to determine the value of streaming for parent companies such as Amazon, Apple, Disney to determine how well their streaming division is doing or if they need to sell their streaming or collaborate with a different streaming company like the recent merger of Paramount streaming with Showtime and the acquisition of Discovery+ by HBO.

Future work regarding this study would include using the models on different stock markets across the world. Furthermore, using a data range of more than one year may provide more accurate results. Additionally, analyzing the models in different economic situations such as booms or recession may allow us to better see the productivity of the models. Besides, the use of a neural network for classifying the sentimental analysis tweets may offer better results.