Twitter Sentiements and Stocks

Oscar Eli Vasquez

August 3, 2020

1 Twitter Sentiments and Stock Performance

The purpose of this project was to explore social media posts from Twitter and attempt to determine if there was a correlation between the sentiments behind select Twitter user posts and select stock index performance. The Alpha Vantage API was used to gather stock information, and the Twitter API was used to gather user tweets that were then analyzed for positive/ negative sentiments using VADER and TextBlob analyzers.

As of 2020 Alpha Vantage has lost access to several US indices, specifically the ones chosen for this study (NDX, DJI, and SPX) which were chosen because they covered a variety of industries. For this demo, 4 tech stocks were chosen MSFT, AAPL, TSLA, and AMZN.

[1]:
```
# Dependencies
import os
import json
import pandas as pd
import requests as req
import config
from datetime import datetime, timedelta
from dateutil.parser import parse
```

[2]:
```
# This will call on the Alpha Vantage API key that has been stored as an␣
 ↪environmental variable
api_key = os.getenv('ALPHAVANTAGE_API_KEY')

# A url is used to get access to the information
url = "https://www.alphavantage.co/query?function=TIME_SERIES_Daily&symbol="
```

[3]:
```
# Define the list of stocks you want to analyze
stock_list = ['MSFT','AAPL','TSLA',"AMZN"]

# Initialize empty list that will hold stock information
stock_list_details = []

# Loop through list that will extract data in JSON format summarizing a days␣
 ↪open, high, low, close, and trading volume
for stock in stock_list:
    data = {"function":"TIME_SERIES_DAILY",
```

```
            "symbol":stock,
            "outputsize":"compact",
            "datatype":"json",
            "apikey":"5H4MYYEY30E01CYT"
           }
    response = req.get(url, data).json()
    stock_list_details.append(response)

# stock_list_details[0]
```

[4]:
```
# Tranform data into Pandas dataframes, using dates as columns
msft_df = pd.DataFrame(stock_list_details[0]['Time Series (Daily)'])
aapl_df = pd.DataFrame(stock_list_details[1]['Time Series (Daily)'])
tsla_df = pd.DataFrame(stock_list_details[2]['Time Series (Daily)'])
amzn_df = pd.DataFrame(stock_list_details[3]['Time Series (Daily)'])

# Transpose the dataframes to list the headers as the aforementioned open, high,␣
 ↪low, close, volume for each day (date)
msft_transpose = msft_df.transpose()
aapl_transpose = aapl_df.transpose()
tsla_transpose = tsla_df.transpose()
amzn_transpose = amzn_df.transpose()

# List first 5 rows of dataframe to check for accuracy
msft_transpose.head()
```

[4]:

	1. open	2. high	3. low	4. close	5. volume
2020-07-31	204.4000	205.1000	199.0100	205.0100	51247969
2020-07-30	201.0000	204.4600	199.5700	203.9000	25079596
2020-07-29	202.5000	204.6500	202.0100	204.0600	19632602
2020-07-28	203.6100	204.7000	201.7400	202.0200	23251388
2020-07-27	201.4700	203.9700	200.8600	203.8500	30160867

[5]:
```
# Reset index to add numbered indices and rename the date column
msft_reset = pd.DataFrame(msft_transpose.reset_index(drop=False)).
 ↪rename(columns={'index':'date'})
aapl_reset = pd.DataFrame(aapl_transpose.reset_index(drop=False)).
 ↪rename(columns={'index':'date'})
tsla_reset = pd.DataFrame(tsla_transpose.reset_index(drop=False)).
 ↪rename(columns={'index':'date'})
amzn_reset = pd.DataFrame(amzn_transpose.reset_index(drop=False)).
 ↪rename(columns={'index':'date'})
```

[6]:
```
# Set conditions that will limit data to July 1, 2020 onward...this only␣
 ↪produces a boolean result where this is true or false for each row
msft_filterByDate = msft_reset.date > '2020-06-30'
aapl_filterByDate = aapl_reset.date > '2020-06-30'
```

```
tsla_filterByDate = tsla_reset.date > '2020-06-30'
amzn_filterByDate = amzn_reset.date > '2020-06-30'

# Update the data in the dataframes to include data for the desired dates only
msft_final =  msft_reset[msft_filterByDate]
aapl_final =  aapl_reset[aapl_filterByDate]
tsla_final =  tsla_reset[tsla_filterByDate]
amzn_final =  amzn_reset[amzn_filterByDate]
```

[7]:
```
# Filter data further to only include trade volume and daily closing price, this␣
 ↪was done to simplify analysis of the data
# This limited analysis to whether positive/negative sentiments led to higher/
 ↪lower trade volume and a gain/ drop in price
msft_final = msft_final.filter(['date','5. volume','4. close']).
 ↪rename(columns={'5. volume':'MSFT_volume','4. close':'MSFT_closing_price'})
aapl_final = aapl_final.filter(['date','5. volume','4. close']).
 ↪rename(columns={'5. volume':'AAPL_volume','4. close':'AAPL_closing_price'})
tsla_final = tsla_final.filter(['date','5. volume','4. close']).
 ↪rename(columns={'5. volume':'TSLA_volume','4. close':'TSLA_closing_price'})
amzn_final = amzn_final.filter(['date','5. volume','4. close']).
 ↪rename(columns={'5. volume':'AMZN_volume','4. close':'AMZN_closing_price'})

msft_final.head()
```

[7]:

	date	MSFT_volume	MSFT_closing_price
0	2020-07-31	51247969	205.0100
1	2020-07-30	25079596	203.9000
2	2020-07-29	19632602	204.0600
3	2020-07-28	23251388	202.0200
4	2020-07-27	30160867	203.8500

[8]:
```
# Create a list of the dataframes
df_list = [msft_final,aapl_final,tsla_final,amzn_final]

# Output the dataframes to CSVs
path = "Stock_Outputs_Update/"
for i in range(len(df_list)):
    df_list[i].to_csv(path + stock_list[i] + ".csv",index = False)
```

1.1 Twitter Sentiments

As mentioned previously, Twitter sentiments were gathered from various users using the Twitter API. These weren't just any users however, news outlets were the chosen demographic for this study. These included The Wall Street Journal, The Financial Times, The New York Times, Barron's, Investor's Business Daily, Reuters, Wired, Bloomberg, and USA Today. By focusing on more investor-centric news sources, the hypothesis was that bad news would result in an overall drop in trade. This hypothesis was also applied to sources that weren't market focused.

[9]:
```
# Import dependencies
import numpy as np
import tweepy
import time
from dateutil.parser import parse


# Import and initialize sentiment analyzers
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()
from textblob import TextBlob
```

[10]:
```
# Setup tweepy API authentication keeping API key secure using a config.py file
auth = tweepy.OAuthHandler(config.api_key, config.api_secret)
auth.set_access_token(config.access_token, config.access_token_secret)
api = tweepy.API(auth, wait_on_rate_limit=True)
target_users = ["@WSJ", "@FT", "@nyTimes", "@barronsonline", "@IBDinvestors",␣
 ↪"@reuters", "@wired", "@business", "@USAToday"]
```

[11]:
```
# Looking back N days
N=2

# Get datetime format for the last day in July to acquire tweets for the entire␣
 ↪month of July
days_ago = datetime.now() - timedelta(days=N)
days = str(days_ago)
dt = parse(days)
converted_time = dt.strftime('%Y-%m-%d')
converted_time
```

[11]:
```
'2020-07-31'
```

[12]:
```
for item in target_users:

    # Lists to hold sentiments, resets for each item in target_users
    compound_list_textblob = []
    compound_list_vader = []
    tweet_times = []
    text_list = []

    for tweet in tweepy.Cursor(api.user_timeline,item, since=converted_time).
 ↪items():

        # Acquire tweet contexts
        tweet_text = json.dumps(tweet._json, indent=3)
        tweet = json.loads(tweet_text)
```

```
        day = parse(str(tweet["created_at"])).strftime('%Y-%m-%d')

        # Run textblob and vader analysis on each tweet
        results_textblob = TextBlob(tweet["text"])
        results_vader = analyzer.polarity_scores(tweet["text"])


        # Get the aggregate score for vader and textblob
        compound_textblob = results_textblob.sentiment.polarity
        compound_vader = results_vader["compound"]


        # Add each value to the appropriate list
        compound_list_textblob.append(compound_textblob)
        compound_list_vader.append(compound_vader)
        tweet_times.append(day)

        # Create dictionary to later convert into dataframe
        sentiment = {"TextBlob": compound_list_textblob,
                     "Vader": compound_list_vader,
                     "date": tweet_times}

    # Create the dataframe with column manipulation to group by day and␣
 ↪calculate mean
    sent_df = pd.DataFrame.from_dict(sentiment).groupby(['date']).mean().
 ↪reset_index()
    sent_df = sent_df.set_index('date')

    # Export to csv file for each item in target_list
    path = "Sentiment_Analyzer_Update/"
    sent_df.to_csv(path + item + ".csv", encoding= 'utf-8', float_format='%.3f')
```

1.2 Merging the files

As you may have seen in the code. The data was output in CSV format. These CSVs were merged here, grouping the stock data with each news outlet sentiment result by date.

[13]:
```
# Set path to twitter sentiment CSVs
sent_path = "Sentiment_Analyzer_Update/"

# Initialize empty list to be filled
sent_csv_list = []

# Fill list of the sentiment analysis CSVs by appending filenames in sent_path␣
 ↪folder
for file in os.listdir(sent_path):
    sent_csv_list.append(sent_path+file)
```

```
sent_csv_list
```

[13]:
```
['Sentiment_Analyzer_Update/@IBDinvestors.csv',
 'Sentiment_Analyzer_Update/@business.csv',
 'Sentiment_Analyzer_Update/@FoxBusiness.csv',
 'Sentiment_Analyzer_Update/@FT.csv',
 'Sentiment_Analyzer_Update/@WSJ.csv',
 'Sentiment_Analyzer_Update/@nyTimes.csv',
 'Sentiment_Analyzer_Update/@reuters.csv',
 'Sentiment_Analyzer_Update/@USAToday.csv',
 'Sentiment_Analyzer_Update/@barronsonline.csv',
 'Sentiment_Analyzer_Update/@wired.csv']
```

[14]:
```
# This loop will create a merged CSV output for each news source
for file in sent_csv_list:

    # Read in the Indices data/reset values in the for loop
    msft_path = os.path.join('Stock_Outputs_Update/MSFT.csv')
    msft_df = pd.read_csv(msft_path)
    msft_df['date'] = pd.to_datetime(msft_df['date'])
    msft_df['MSFT_closing_price'] = msft_df['MSFT_closing_price'].map('{:.2f}'.
 ↪format)

    aapl_path = os.path.join('Stock_Outputs_Update/AAPL.csv')
    aapl_df = pd.read_csv(aapl_path)
    aapl_df['date'] = pd.to_datetime(aapl_df['date'])
    aapl_df['AAPL_closing_price'] = aapl_df['AAPL_closing_price'].map('{:.2f}'.
 ↪format)

    tsla_path = os.path.join('Stock_Outputs_Update/TSLA.csv')
    tsla_df = pd.read_csv(tsla_path)
    tsla_df['date'] = pd.to_datetime(tsla_df['date'])
    tsla_df['TSLA_closing_price'] = tsla_df['TSLA_closing_price'].map('{:.2f}'.
 ↪format)

    amzn_path = os.path.join('Stock_Outputs_Update/AMZN.csv')
    amzn_df = pd.read_csv(amzn_path)
    amzn_df['date'] = pd.to_datetime(amzn_df['date'])
    amzn_df['AMZN_closing_price'] = amzn_df['AMZN_closing_price'].map('{:.2f}'.
 ↪format)

    all_df = ""

    # Create df for each file
    df = pd.read_csv(file)
    df['date'] = pd.to_datetime(df['date'])
    df['TextBlob'] = df['TextBlob'].map('{:.3f}'.format)
```

```
    df['Vader'] = df['Vader'].map('{:.3f}'.format)



    # Since the df is sorted by days, we can call the last 30 rows for the last␣
↪30 days and reassign df
    df = df.tail(30)

    # Merge stock date
    msft_df = pd.merge(df,msft_df, how='outer', on='date').
↪reset_index(drop=True).fillna(0).sort_values(by='date')
    msft_df = msft_df.reset_index(drop=True)

    aapl_df = pd.merge(msft_df,aapl_df, how='outer', on='date').
↪reset_index(drop=True).fillna(0).sort_values(by='date')
    aapl_df = aapl_df.reset_index(drop=True)

    tsla_df = pd.merge(aapl_df,tsla_df, how='outer', on='date').
↪reset_index(drop=True).fillna(0).sort_values(by='date')
    tsla_df = tsla_df.reset_index(drop=True)
    amzn_df = pd.merge(tsla_df,amzn_df, how='outer', on='date').
↪reset_index(drop=True).fillna(0).sort_values(by='date')

    # Final merge to combine all data
    all_df = amzn_df.reset_index(drop=True)

    end_path = "Merged_Update/"


    all_df.to_csv(end_path +file[26:-4]+"_all.csv",index=False)
```

1.3 Plotting

Below is an example of the plots used for high level analysis of the data. The line plot depicts closing price change for each day, and the scatter plot shows the sentiment of the day by color, as well as trading volume by marker size.

[15]:
```
# Import dependencies
import matplotlib.pyplot as plt
```

[16]:
```
# Set path to merged CSVs
merged_path = "Merged_Update/"

# Initialize empty list to be populated with the merged CSV filenames
merged_list = []
```

```
# Populate list
for file in os.listdir(merged_path):
    merged_list.append(merged_path+file)
```

[20]:
```
# Initialize empty list to be populated by dataframes
list_of_dfs = []

# Populate list
for file in merged_list:

    df = pd.read_csv(file, index_col = False)
    df = df.loc[(df!=0).all(axis=1)]
    list_of_dfs.append(df)

# Call on the 7th item in the list. This calls on the Wired dataframe
list_of_dfs[7].head()
```

[20]:

	date	TextBlob	Vader	MSFT_volume	MSFT_closing_price	AAPL_volume	\
3	2020-07-06	0.091	0.122	32033747.0	210.70	29745936.0	
4	2020-07-07	0.141	0.159	33600749.0	208.25	28206999.0	
5	2020-07-08	0.105	0.082	33602226.0	212.83	29274479.0	
6	2020-07-09	0.153	0.043	33122280.0	214.32	31420357.0	
7	2020-07-10	0.075	0.117	26178663.0	213.67	22564330.0	

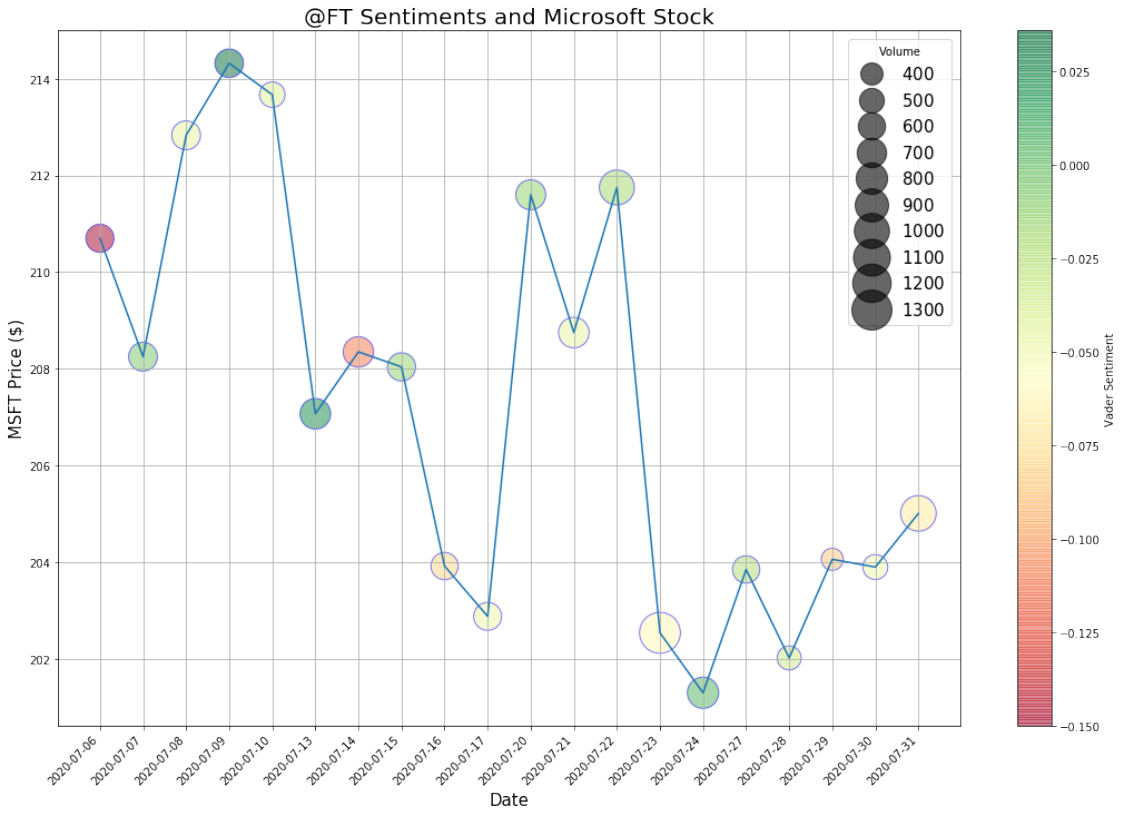
	AAPL_closing_price	TSLA_volume	TSLA_closing_price	AMZN_volume	\
3	373.85	20570284.0	1371.58	6880340.0	
4	372.69	21493301.0	1389.86	5257688.0	
5	381.37	16330188.0	1365.88	5037856.0	
6	383.01	11718292.0	1394.28	6388971.0	
7	383.68	23346353.0	1544.65	5486005.0	

	AMZN_closing_price
3	3057.04
4	3000.12
5	3081.11
6	3182.63
7	3200.00

[19]:
```
# This can be automated to output various graphs
# Plotting simplified by using i to iterate through different Twitter users
i=1

x = list_of_dfs[i].date
vader = list_of_dfs[i]['Vader'].values
textblob = list_of_dfs[i]['TextBlob'].values

# Choose stock to display
```

```
y = list_of_dfs[i]['MSFT_closing_price'].values
volume = (list_of_dfs[i]['MSFT_volume'].values)/50000

# Define plot and subplot
fig, ax = plt.subplots(figsize = (15,10))

# Plot a scatter and line on graph
# Markers have size dependent on trade volume and color dependent on Vader␣
 ↪sentiment results (can be changed to TextBlob)
scatter = ax.scatter(x, y, s=volume, c=vader, cmap='RdYlGn', alpha=0.5, marker =␣
 ↪'o',edgecolors='b')
ax.plot(x, y)

# Label axes
ax.set_xticklabels(labels=x,rotation = (45),ha='right')
ax.set_xlabel('Date', size = 15)
ax.set_ylabel('MSFT Price ($)', size = 15)
ax.grid(True)


# Add legend and colorbar to label volume and sentiment data range
handles, labels = scatter.legend_elements(prop="sizes", alpha=0.6)
legend = ax.legend(handles,labels,loc = 'upper right', title = 'Volume', prop =␣
 ↪{'size':15})
plt.colorbar(scatter,label = 'Vader Sentiment')

fig.tight_layout()

ax.set_title(merged_list[i][14:-8] + " Sentiments and Microsoft Stock", size =␣
 ↪20)
```

[19]: Text(0.5, 1, '@FT Sentiments and Microsoft Stock')



1.4 Conclusions (Lessons Learned)

For this study Twitter sentiments from 9 news sources were used to correlate with 4 tech stock trends. Immediate observations showed that positive sentiments for a day often led to higher trade volumes, but no definite correlations could be made between closing price and the day's sentiment. Studying the stocks also provided more ideas and questions on how to analyze the data. The immediate ideas that come to mind is investigating whether high volume trading days corresponding to negative sentiments are due to people buying/selling more shares (I would hypothesize selling), looking at the difference between opening and closing price for each day and see how that corresponds with the day's sentiments, and as this study uses about 30 days worth of data, produce a smoothed curve overtime to get a better gauge of the overall trend for the month. In regards to the sentiment analysis, it was observed that the VADER analyzer outputs more negative sentiments than TextBlob. Another interesting quirk found in the data was that often times, news sources of varying opinions about certain situations often had opposite sentiments produced by the analyzers.