The impact of Trump’s Twitter on the Wall Street

Y.Yang
2019.10

Abstract

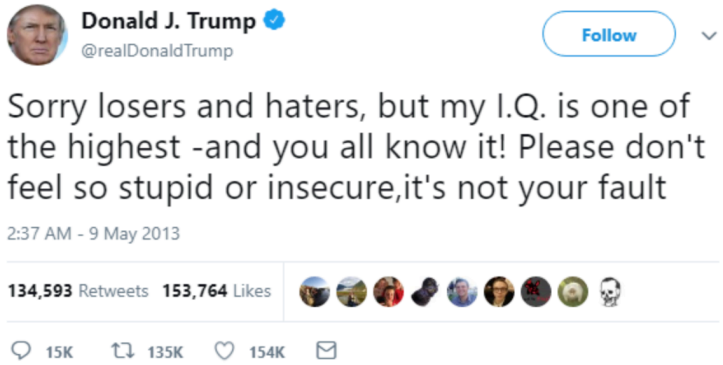
In this project, I built a Twitter Api and used it to extract total 3200 Donald Trump's recent tweets. I downloaded Sentiment140 (1,600,000 labeled tweets) as the training data and then built a sentiment classifier (Naive Bayes Classifier). After applying this built model on the retrieved tweets I successfully labeled those tweets. Finally I analyzed the emotions of the president and tried to find the impact of his tweets on the stock market, which I referred to the data of NYSE Composite.

In conclusion, the correlation is weak but when Trump tweeted more than 20 which reveal a negative sense, the stock market is likely to vary with the tweets number accordingly.

Motivation

Recent years, President Trump tweets using the sensitive words such as “China,” “Wall”, “democrats,” and “great” were the biggest market movers.

In this project, I will be studying how strong is the power of these tweets and how do they shape the moves of investors in the Wall Street. I believe the fallouts are likely to be reflected on the NYSE Composite, as by far, The New York Stock Exchange is the world's largest stock exchange by market capitalization.



Datasets

Dataset of NYA: The historical data of the NYSE Composite
https://finance.yahoo.com/quote/%5Enya/history/

Sentiment140: Training sets for Twitter Sentiment Analysis (1,600,000 labeled tweets)
http://help.sentiment140.com/for-students

Trump's tweets: import Tweepy and use Cursor way for Twitter data mining
http://docs.tweepy.org/en/v3.8.0/cursor_tutorial.html

Data Preparation and Cleaning

A. Data preparation for NYA:

Add a new column calculating the daily percentage change % of the NYSE Composite:

(Close - Open)/Open *100%

B. Pre-Processing for dataset Sentiment140 and the extracted Twitter text messages:

We cannot get the sentiment from punctuation. Therefore, punctuation does not matter to Sentiment Analysis. Also, tweet components like images, videos, URLs, usernames, emojis, etc. do not contribute to the polarity (whether it is positive or negative) of the tweet. We should remove them.

Research Questions

Question 1: Which words are the most common in Trump’s tweets?

Question 2: How many times did Trump tweet a day?

Question 3: What's Trump's mood on Tweet today?

Question 4: How did trump's tweets shape the Stock Market?

Condition 1: if his tweets revealed a very positive sense
Condition 2: if Trump tweeted provocative words like “democrats” and “china”
Condition 3: if Trump tweeted more than 20 pos/neg tweets on a single day

Methods of Model Building for Sentiment Analysis

Step 1: Download labeled tweets as the training dataset
Step 2: Pre-process Tweets (Tokenize text messages and remove tweet components like punctuation, images, videos, URLs, usernames, emojis, etc. as they do not contribute to the polarity.)
Step 3: Build features for machine learning
Step 4: Build a sentiment classifier (Naive Bayes Classifier) fitting the training dataset
Step 5: Validate the model

Methods of Twitter Data Mining

Step 1: Authenticate a Python Application with Twitter using Tweepy, which provides easy access to the Twitter API
Step 2: Fetch the recent 3200 tweets of @realDonaldTrump (Extract Tweets by the Cursor way, which handles all the pagination loop behind the scenes)
Step 3: Pre-process Tweets and building features for machine learning
Step 4: Apply the built sentiment classifier to predict the emotions of those retrieved tweets
Step 5: Analyze the labeled data

Codes for Key Processes

Pre-Processing text messages of Tweets:

```
import re
from nltk.tokenize import TweetTokenizer # TweetTokenizer does not split the contraction into two parts

def preprocess(corpus_list):
    for t in corpus_list:
        t['text'] = t['text'].lower() # convert text to lower-case
        t['text'] = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', '', t['text']) # remove URLs
        t['text'] = re.sub('@[^\s]+', '', t['text']) # remove usernames
        t['text'] = re.sub(r'#([^\s]+)', '', t['text'])  # remove the '#' in #hashtag
        t['text'] = TweetTokenizer().tokenize(t['text']) #tokenize the modified text and remove repeated characters
    return corpus_list
```

Build features for machine learning:

```
def build_bag_of_words_features_filtered(statuses):
    corpus = []
    for t in statuses:
        t['text'] = {word:1 for word in t['text'] if not word in useless_words}
        if len(t['text'])>0: # remove empty text
            corpus.append(t)
    return corpus
```

Twitter data mining:

```
statuses = []

for page in tw.Cursor(api.user_timeline, screen_name='realDonaldTrump', tweet_mode="extended", count=200).pages(16):
    for status in page:
        statuses.append({'Date': status.created_at.strftime('%Y-%m-%d'), 'text': status.full_text})
```

Finding 1

Fig.1 WordCloud of the 100 most common words Trump tweeted

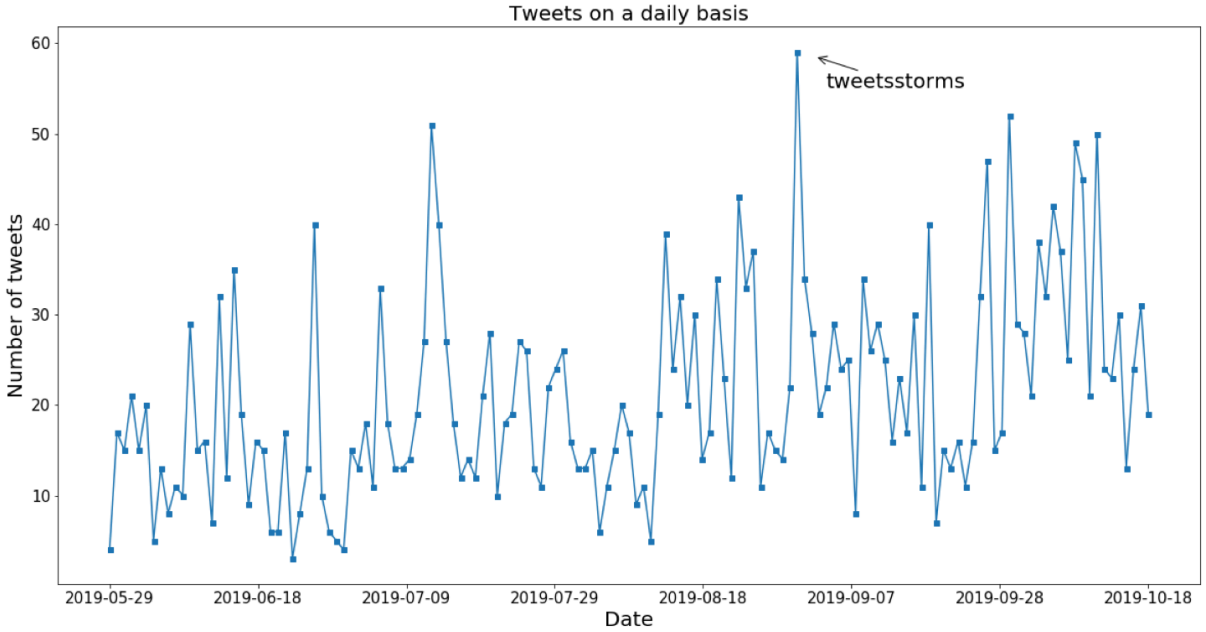


Fig.2 Number of Trump's Tweets on a daily basis

He likes tweeting about "republican" and "democrat" stuff, which meet our intuitive sense.

The president's market-moving tweeting "ballooned" in frequency in August 2019, when the China trade war and the Federal Reserve were top of the mind.

Finding 2

I used the built sentiment_classifier to predict the emotions of those retrieved tweets. And in order to analyze the correlation between emotions and stock price changes numerically, I labeled each positive tweet 1 and negative one -1; 0 for neither positive nor negative. Here it comes the distribution of his mood revealed by the tweets:

count (days)	143
mean	21
std	11.44
min	3
25%	13
50%	18
75%	28
max	59

Table 1 Statistic data of daily tweet numbers

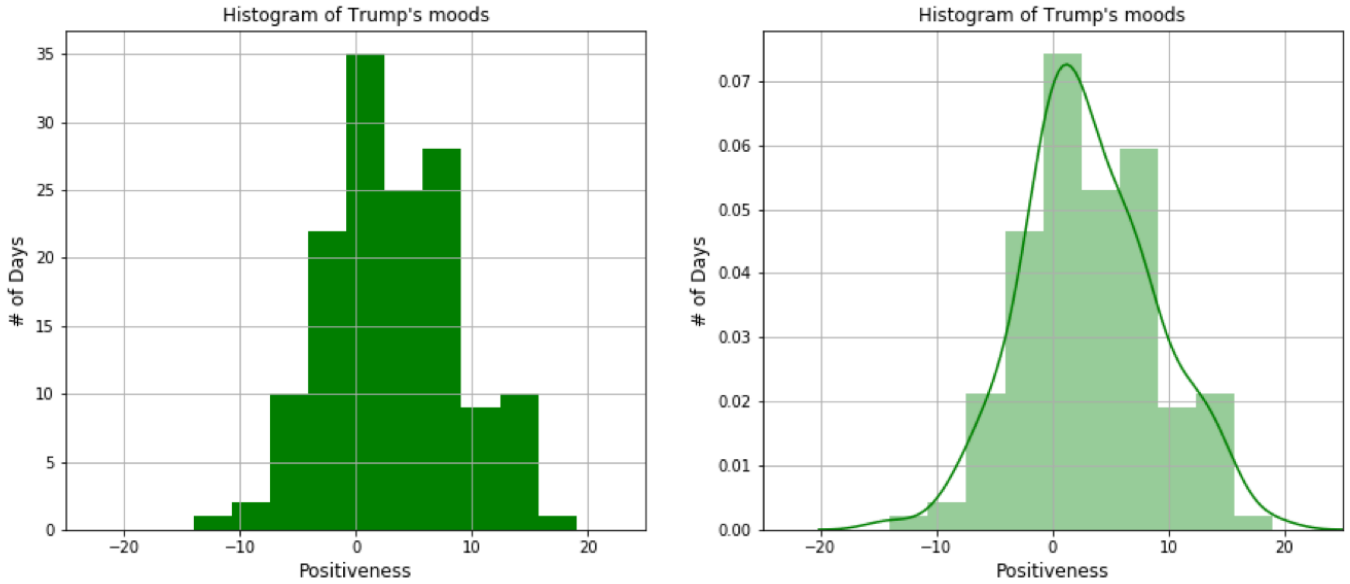


Fig.3 Distribution of Trump's mood

It can be seen as a normal distribution with the fact that 'neutral' (positiveness=0) is the most common case. Also, we can conclude Trump is likely to tweet messages positive instead of negative given probability.

Finding 3-1

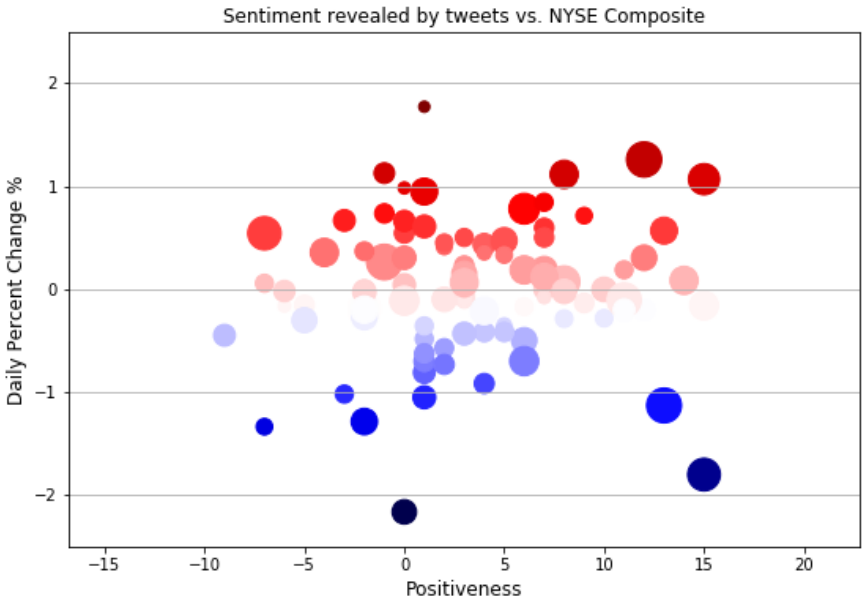


Fig.4 Scatter Plot: Sentiments of Trump and NYSE changes (Size: number of tweets)

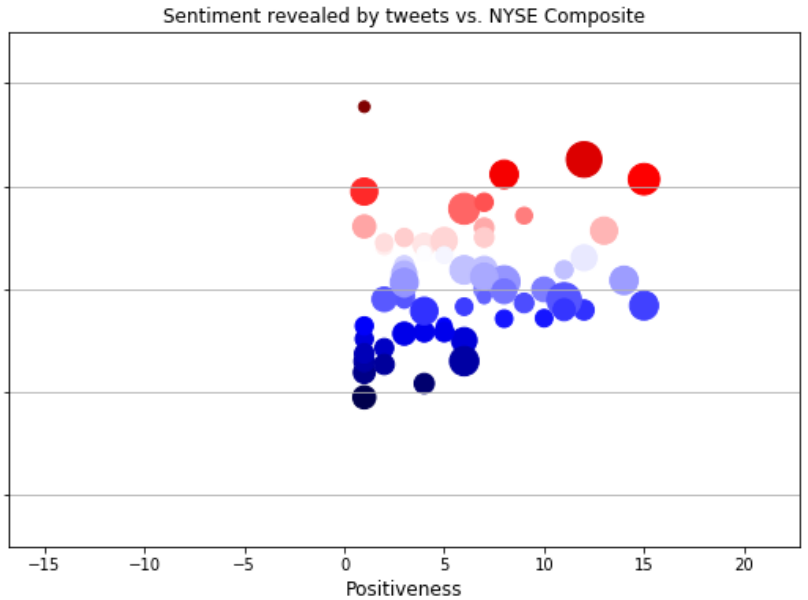


Fig.5 When Trump's tweets sent positive signals, outliers withdrawn (Positiveness > 10)

	All Tweets	Positive
Correlation	0.09	0.27

Finding 3-2

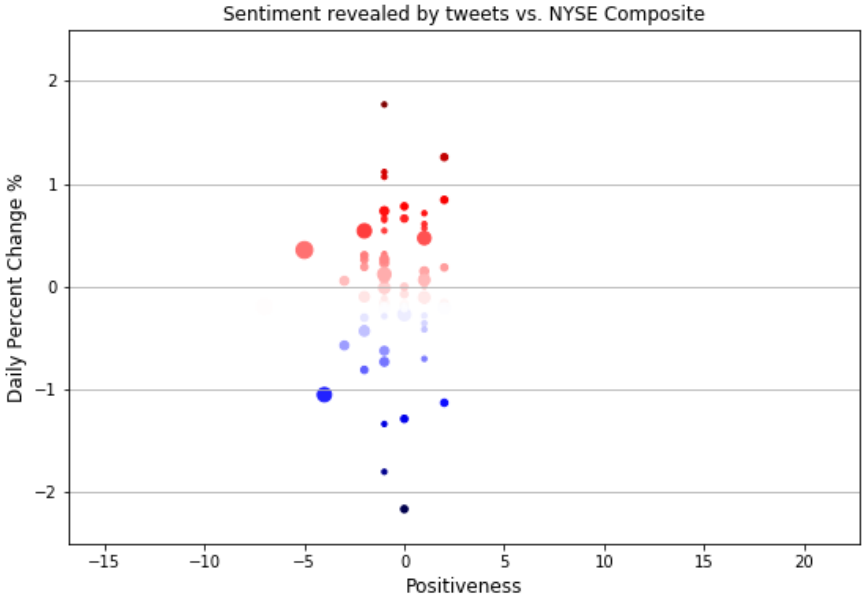


Fig.6 when Trump tweeted ‘democrats’

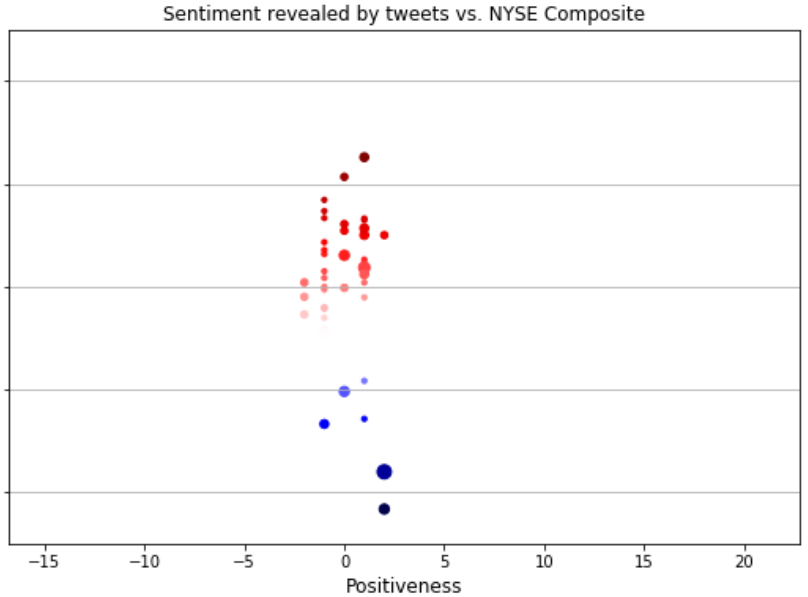


Fig.7 when Trump tweeted ‘china’

This results show Trump was more interested in 'democrats' stuff than 'china'. He indeed took aim at Democrats on a range of issues. Additionally, when he tweeted subjects of “democrats”, his political rivalries, the stock market reacted in a wide range, while the NYSE was more likely to rise in most cases when the president gave opinions about China.

Finding 4

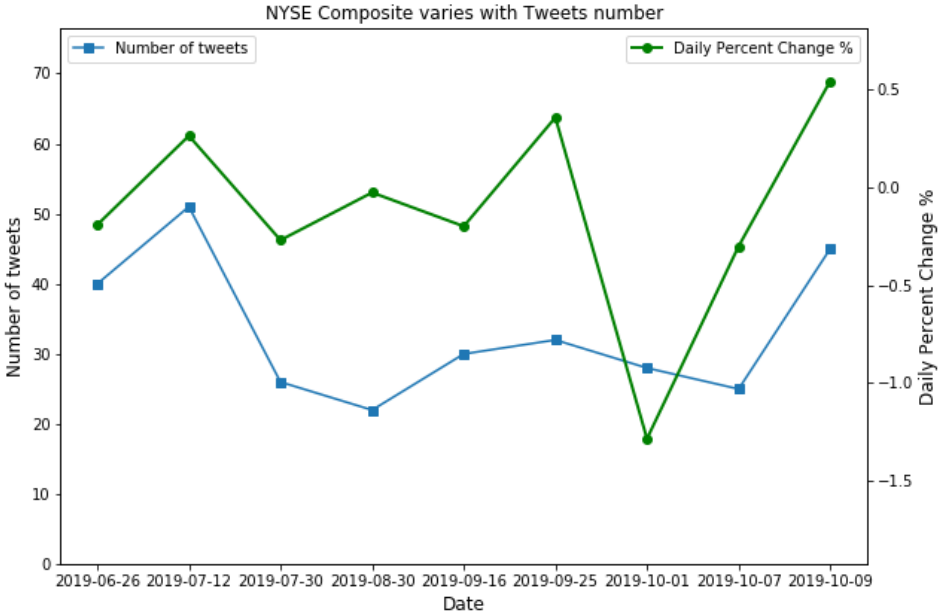


Fig.8 when Trump tweeted more than 20 showing negative on a single day

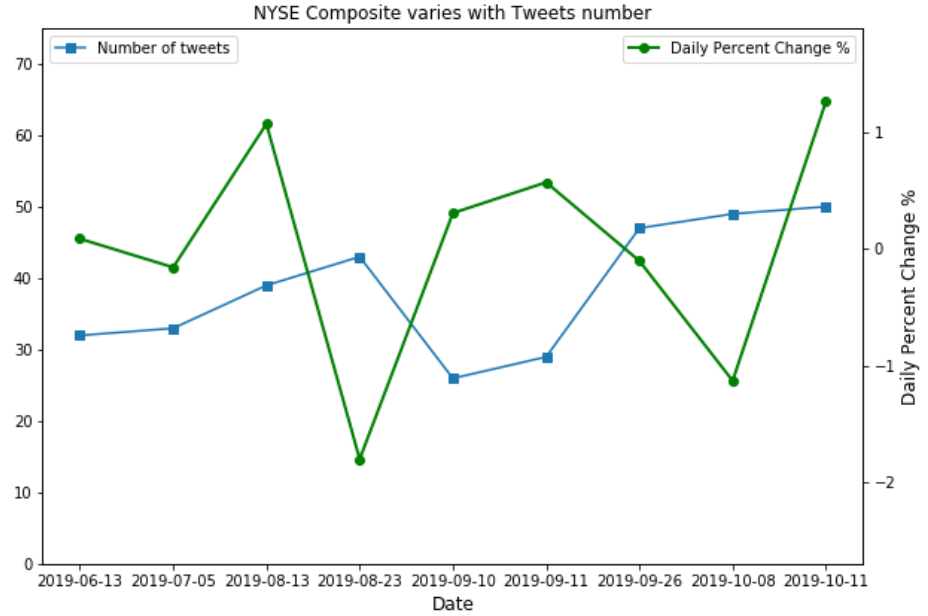


Fig.9 when Trump tweeted more than 20 showing positive on a single day

	> 20, neg	> 20, pos
Correlation	0.51	-0.21

Fig. 8 shows there is a strong linear relationship between the two variables - on days with more than 20 Trump tweets generally showing a negative sense, the stock market varied with the tweets number accordingly.

Conclusions

Donald Trump, a politician, not surprisingly, prefers to tweet using key words “democrat” and “republican”. He tweeted average 21 tweets per day across the recent 143 days and there was a “tweetsstorms” with total 59 tweets releasing on a single day in August. Trump’s overall attitude revealed by his tweets is “neutral” given the normal distribution of his emotions. We can also conclude Trump is likely to tweet something positive instead of negative considering probability.

Unfortunately, the correlation between the emotion of Trump’s tweets and the change of the NYSE composite is not as strong as I expected ahead of this investigation, even only considering positive cases with outliers removed.

For specific issues, Trump was more interested in 'democrats’ issues than 'china’. The NYSE was likely to rise in most cases when the president gave opinions about China. One another important finding is the stock market varied with the tweets number accordingly on days with more than 20 Trump tweets showing a negative mood. This strong linear relationship has been verified.

Limitations

In this project, I studied how Trump's tweets shape the confidence of investors reflecting on the NYSE composite. The resulted correlation may not be applicable for the other stock markets. For specific cases, the relationship could be stronger. One may study the trend of local Chinese markets while analyzing the cases of his tweeting "China", for example. In addition, although I fetched text messages of tweets and removed components like punctuation, images and videos were not retrieved for sentiment analysis. They also contribute to emotions. Moreover, the accuracy of the built sentiment classifier is around 75%, although pretty good for such a simple model considering the estimated accuracy for a person is about 80%, I cannot secure all the sentiment judgements on Trump's tweets are correct. Pretty sure the model of NLP can be improved.

References

Creating The Twitter Sentiment Analysis Program in Python with Naive Bayes Classification:
https://towardsdatascience.com/creating-the-twitter-sentiment-analysis-program-in-python-with-naive-bayes-classification-672e5589a7ed

Analyze Sentiments Using Twitter Data and Tweepy in Python:
https://www.earthdatascience.org/courses/earth-analytics-python/using-apis-natural-language-processing-twitter/analyze-tweet-sentiments-in-python/

Extracting Twitter Data, Pre-Processing and Sentiment Analysis using Python 3.0:
https://towardsdatascience.com/extracting-twitter-data-pre-processing-and-sentiment-analysis-using-python-3-0-7192bd8b47cf

Project: The impact of Trump's Twitter on the Wall Street

Recent years, the President Trump tweets using the sensitive words such as "China," "Wall", "democrats," and "great" were the biggest market movers.
In this project, I will study how strong is the power of the tweets and how do they shape the moves of investors, which are believed to be reflected on the NYSE Composite, as by far The New York Stock Exchange located at Wall Street is the world's largest stock exchange by market capitalization.

Part 1: Extracting Twitter Data, Pre-Processing and Sentiment Analysis Model Building

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# !pwd
```

The historical data of the NYSE Composite can be downloaded from:
https://finance.yahoo.com/quote/%5Enya/history/ (https://finance.yahoo.com/quote/%5Enya/history/)
First, let's take a look at it.

Data Exploration

In [3]:
```
df_NYA = pd.read_csv('NYA_2019.csv')
```

Insert a new column showing the daily change of the NYSE Composite:

In [4]:
```
df_NYA['Daily Percent Change %'] = (df_NYA.Close-df_NYA.Open)/df_NYA.Open*100
```

In [5]:
```
df_NYA.head()
```

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume	Daily Percent Change %
0	2018-12-31	11338.240234	11378.190430	11270.629883	11374.389648	11374.389648	3442870000	0.318827
1	2019-01-02	11238.769531	11407.799805	11204.280273	11383.530273	11383.530273	3733160000	1.288048
2	2019-01-03	11343.790039	11343.790039	11169.459961	11190.440430	11190.440430	3822860000	-1.351838
3	2019-01-04	11323.730469	11559.019531	11323.730469	11533.339844	11533.339844	4213410000	1.851063
4	2019-01-07	11536.049805	11678.969727	11504.769531	11605.959961	11605.959961	4104710000	0.606015

"Volume" refers to the number of shares traded in a given time period.

In [6]:
```
df_NYA.Date.min(), df_NYA.Date.max()
```

Out[6]:
```
('2018-12-31', '2019-10-18')
```

So the downloaded dataset reflects the changes within the year of 2019.

Authenticating a Python Application with Twitter

In [7]:
```
import pickle
import os
import json
```

The functions that the OS module: provides allows you to interface with the underlying operating system that Python is running on – Windows, Mac or Linux.
You can find important information about your location or about the process.

The Pickle module: is serializing and de-serializing a Python object structure.

In [8]:
```
if not os.path.exists('secret_twitter_credentials_final.pkl'):
    Twitter={}
    Twitter['Consumer Key'] = [redacted]
    Twitter['Consumer Secret'] = [redacted]
    Twitter['Access Token'] = [redacted]
    Twitter['Access Token Secret'] = [redacted]
    with open('secret_twitter_credentials.pkl','wb') as f:
        pickle.dump(Twitter, f)
else:
    Twitter=pickle.load(open('secret_twitter_credentials_final.pkl','rb'))
```

Authenticate a Python Application with Twitter using Tweepy, which provides easy access to the Twitter API:

In [9]:
```
import tweepy as tw

auth = tw.OAuthHandler(Twitter['Consumer Key'], Twitter['Consumer Secret'])
auth.set_access_token(Twitter['Access Token'], Twitter['Access Token Secret'])
api = tw.API(auth, wait_on_rate_limit=True)

print(api)
```

```
<tweepy.api.API object at 0x1213846d0>
```

Pre-processing

In [10]:
```
import nltk
import csv
```

First, I need some downloadable Training sets for the Twitter Sentiment Analysis.
Here I will be using the Sentiment140 for students, which started as a class project from Stanford University.
This training dataset of Twitter was automatically created using Twitter Api and was manually assigned labels for each tweet as either “Positive”, “Negative” or “Neutral”.
Source: http://help.sentiment140.com/for-students (http://help.sentiment140.com/for-students)

The data is a CSV with emoticons removed. Data file format has 6 fields:
0 - the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
1 - the id of the tweet (2087)
2 - the date of the tweet (Sat May 16 23:58:44 UTC 2009)
3 - the query (lyx). If there is no query, then this value is NO_QUERY.
4 - the user that tweeted (robotickilldozr)
5 - the text of the tweet (Lyx is cool)

Only the texts and the polarities are required for building the model. I will be collecting them to a list named corpus:

In [11]:
```
corpus = []

with open( [redacted]training_noemoticon.csv','r', encoding='ISO-8859-1') as csvfile:
    lineReader = csv.reader(csvfile, delimiter=',', quotechar="\"")
    for row in lineReader:
        #print (row)
        corpus.append({'text':row[5], 'polarity':row[0]})

        #{'polarity':row[0], 'id':row[1], 'date':row[2], 'user':row[4], 'text':row[5]}
```

Tips: I used 'r' instead of 'rb' to read the file.
'rb' will open the file in binary mode - now the data is read and written in the form of bytes objects. This mode should be used for all files that don’t contain text.
https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files (https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files)

UTF-8 is a multibyte encoding that can represent any Unicode character. ISO 8859-1 is a single-byte encoding that can represent the first 256 Unicode characters. Both encode ASCII exactly the same way.

In [12]:
```
corpus[:2] # test
```

Out[12]:
```
[{'text': "@switchfoot http://twitpic.com/2y1zl (http://twitpic.com/2y1zl) - Awww, that's a bummer.  You shoulda got David Carr of Third Day
to do it. ;D",
  'polarity': '0'},
 {'text': "is upset that he can't update his Facebook by texting it... and might cry as a result  School today also. Blah!",
  'polarity': '0'}]
```

In [13]:
```
print (type(corpus[0]['text']), len(corpus))
```

```
<class 'str'> 1600000
```

So total I have 1,600,000 labeled tweets, which will be used as the training dataset. This is exactly what I wanted.

Remove tweet components like images, videos, URLs, usernames, emojis, etc. which do not contribute to the polarity of the tweet and use the `word_tokenize` function to properly tokenize the text:

In [14]:
```
import re
from nltk.tokenize import TweetTokenizer # TweetTokenizer does not split the contraction into two parts

def preprocess(corpus_list):
    for t in corpus_list:
        t['text'] = t['text'].lower() # convert text to lower-case
        t['text'] = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', '', t['text']) # remove URLs
        t['text'] = re.sub('@[^\s]+', '', t['text']) # remove usernames
        t['text'] = re.sub(r'#([^\s]+)', '', t['text'])  # remove the '#' in #hashtag
        t['text'] = TweetTokenizer().tokenize(t['text']) #tokenize the modified text and remove repeated characters
    return corpus_list
```

re.sub():
replace a string that matches a regular expression instead of perfect match. It will specify a regular expression pattern in the first argument, a new string in the second argument, and a string to be processed in the third argument.
https://docs.python.org/3/library/re.html (https://docs.python.org/3/library/re.html)

String literals may optionally be prefixed with a letter 'r' or 'R'; such strings are called raw strings: backslashes are not handled in any special way in a string literal prefixed with 'r'.
Example: r"\n" is a two-character string containing '' and 'n'.

In [15]:
```
corpus = preprocess(corpus)
```

I will build features of, in addition, the Python **`string.punctuation`** list and **the English stopwords** to filter out those words that would not help in the classification.
And meanwhile I will build the bag-of-words model, which assumed that each word is a feature that can either be `positive` or `negative` used by a classifier.
If a word is missing, that would be assigning `False` .

In [16]:
```
import string

others = ['rt',''','…','...','“','”']
useless_words = nltk.corpus.stopwords.words("english") + list(string.punctuation) + others
```

In [17]:
```
string.punctuation
```

Out[17]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [18]:
```
def build_bag_of_words_features_filtered(statuses):
    corpus = []
    for t in statuses:
        t['text'] = {word:1 for word in t['text'] if not word in useless_words}
        if len(t['text'])>0: # remove empty text
            corpus.append(t)
    return corpus
```

In [19]:
```
corpus = build_bag_of_words_features_filtered(corpus)
len (corpus)
```

Out[19]:
```
1592508
```

In [20]:
```
corpus[:2] #test: filtered tokens
```

Out[20]:
```
[{'text': {'awww': 1,
   "that's": 1,
   'bummer': 1,
   'shoulda': 1,
   'got': 1,
   'david': 1,
   'carr': 1,
   'third': 1,
   'day': 1,
   ';d': 1},
  'polarity': '0'},
 {'text': {'upset': 1,
   "can't": 1,
   'update': 1,
   'facebook': 1,
   'texting': 1,
   'might': 1,
   'cry': 1,
   'result': 1,
   'school': 1,
   'today': 1,
   'also': 1,
   'blah': 1},
  'polarity': '0'}]
```

Then divide the data into three by the polarity numbers:

In [21]:
```
words_neg = [i['text'] for i in corpus if i['polarity']=='0']
words_neu = [i['text'] for i in corpus if i['polarity']=='2']
words_pos = [i['text'] for i in corpus if i['polarity']=='4']
```

In [22]:
```
len(words_neg), len(words_neu), len(words_pos)
```

Out[22]:
```
(796355, 0, 796153)
```

It seems there is no "neutral" data in the dataset for the further analysis. It does not matter, we can still build a model.

Build the features for machine learning

The format (example): [({ "here":1, "some":1, "words":1 }, "pos"), ({ "another":1, "tweet":1}, "pos")...]

In [23]:
```
neg_features = [(i, 'neg') for i in words_neg]
neu_features = [(i, 'neu') for i in words_neu]
pos_features = [(i, 'pos') for i in words_pos]

neg_features[:2] # test
```

Out[23]:
```
[({'awww': 1,
   "that's": 1,
   'bummer': 1,
   'shoulda': 1,
   'got': 1,
   'david': 1,
   'carr': 1,
   'third': 1,
   'day': 1,
   ';d': 1},
  'neg'),
 ({'upset': 1,
   "can't": 1,
   'update': 1,
   'facebook': 1,
   'texting': 1,
   'might': 1,
   'cry': 1,
   'result': 1,
   'school': 1,
   'today': 1,
   'also': 1,
   'blah': 1},
  'neg')]
```

Building a sentimental classifier

In [24]:
```
from nltk.classify import NaiveBayesClassifier
```

In [25]:
```
split = 56000 # Training:Test = 7:3
sentiment_classifier = NaiveBayesClassifier.train(pos_features[:split]+neg_features[:split])
```

In [26]:
```
nltk.classify.util.accuracy(sentiment_classifier, pos_features[:split]+neg_features[:split])*100
```

Out[26]:
```
83.87410714285714
```

In [27]:
```
nltk.classify.util.accuracy(sentiment_classifier, pos_features[split:]+neg_features[split:])*100
```

Out[27]:
```
74.07038665106842
```

Accuracy here is around 75%, pretty good for such a simple model considering that the estimated accuracy for a person is about 80%.
I can also print the most informative features for review:

In [28]:
```
sentiment_classifier.show_most_informative_features()
```

```
Most Informative Features
                     sux = 1                neg : pos    =     32.3 : 1.0
                migraine = 1                neg : pos    =     28.3 : 1.0
                   boooo = 1                neg : pos    =     27.7 : 1.0
                coughing = 1                neg : pos    =     25.7 : 1.0
                  lonely = 1                neg : pos    =     23.4 : 1.0
                  gutted = 1                neg : pos    =     21.7 : 1.0
                 noooooo = 1                neg : pos    =     21.0 : 1.0
              bamboozle = 1                 neg : pos    =     21.0 : 1.0
                 4hours = 1                 pos : neg    =     19.7 : 1.0
                    ache = 1                neg : pos    =     18.5 : 1.0
```

In [29]:
```
xtest={'nooooooo':1,'lonely':1,'sux':1,'boooo':1} # simple test on the model
```

In [30]:
```
y_predicted = sentiment_classifier.classify(xtest)
y_predicted
```

Out[30]:
```
'neg'
```

Save the model:

In [31]:
```
with open("saved_model.pkl", "wb") as f:   #Pickling
      pickle.dump(sentiment_classifier, f)
```

Mining Twitter Data for Sentiment Analysis

api.user_timeline: by default will return the 20 most recent statuses posted from the user specified.
And this method can only return up to 3,200 of a user's most recent Tweets. Native retweets of other statuses by the user is included in this total, regardless of whether include_rts is set to false when requesting this resource.

Read more: https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline (https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline)

In [32]:
```
#Example:
#for status in tw.Cursor(api.user_timeline, screen_name='realDonaldTrump', tweet_mode="extended").pages(1):
    #print(len (status))
#--> 20
```

Cursor way: handles all the pagination loop behind the scenes.
Read more: http://docs.tweepy.org/en/v3.8.0/cursor_tutorial.html (http://docs.tweepy.org/en/v3.8.0/cursor_tutorial.html)

I also refered to: https://stackoverflow.com/questions/46734636/tweepy-api-user-timeline-count-limited-to-200 (https://stackoverflow.com/questions/46734636/tweepy-api-user-timeline-count-limited-to-200)
https://stackoverflow.com/questions/11351711/getting-a-users-entire-twitter-timeline-with-tweepy (https://stackoverflow.com/questions/11351711/getting-a-users-entire-twitter-timeline-with-tweepy)

Tips:
if use "Get" method for mining, there are two initial buckets available for **GET requests**: 15 calls every 15 minutes, and 180 calls every 15 minutes. Source:
https://developer.twitter.com/en/docs/basics/rate-limiting (https://developer.twitter.com/en/docs/basics/rate-limiting)

One should sleep the execution for 900/180 seconds in order to abide by the request limit like:
status = twitter_api.GetStatus(tweet["tweet_id"])
time.sleep(sleep_time)

In [35]:
```
statuses = []

for page in tw.Cursor(api.user_timeline, screen_name='realDonaldTrump', tweet_mode="extended", count=200).pages(16):
    for status in page:
        statuses.append({'Date': status.created_at.strftime('%Y-%m-%d'), 'text': status.full_text})
```

strftime(format): to create a string representing the time under the control of an explicit format string.
Source: https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior (https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior)

In [36]:
```
assert len(statuses) > 3000
```

In [37]:
```
len(statuses), statuses[-1], statuses[0]
```

Out[37]:
```
(3195,
 {'Date': '2019-05-29',
  'text': '...If Alabama does not elect a Republican to the Senate in 2020, many of the incredible gains that we have made during my Presidency may be lost, including our Pro-Life victories. Roy Moore cannot win, and the consequences will be devastating....Judges and Supreme Court Justices!'},
 {'Date': '2019-10-19', 'text': '#StopTheCoup'})
```

In [38]:
```
len (statuses[0]['text']) # original length of the first fetched text
```

Out[38]:
```
12
```

So we got total around 3200 tweets, the maximum number we can get using the cursor way, of President Trump dating back to 25th, May, 2019.

```
Apply the functions preprocess( ) and build_bag_of_words_features_filtered( ) again on the text we just collected to filter out the
punctuation, the English stopwords, etc:
```

In [39]:
```
statuses = preprocess(statuses)
len (statuses) # words count of the first fetched text
```

Out[39]:
```
3195
```

In [40]:
```
statuses = build_bag_of_words_features_filtered(statuses)
len (statuses) # words count of the first fetched text
```

Out[40]:
```
3013
```

In [41]:
```
statuses[0] # an example of filtered statuses
```

Out[41]:
```
{'Date': '2019-10-19',
 'text': {'one': 1,
  'pro-trump': 1,
  'demonstrator': 1,
  'began': 1,
  'hounding': 1,
  'impeachment': 1,
  'inquiry': 1}}
```

In [42]:
```
with open("statuses.txt", "wb") as fp:   #Pickling
      pickle.dump(statuses, fp)
```

on days with more than 35 Trump tweets, the stock market has fallen a bit, while on days with fewer than five Trump tweets, it's gone up.

Project: The impact of Trump's Twitter on the Wall Street

Part 2: Statistical Sentiment-Analysis for Trump's tweets ¶

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
df_NYA = pd.read_csv('NYA_2019.csv')
df_NYA['Daily Percent Change %'] = (df_NYA.Close-df_NYA.Open)/df_NYA.Open*100
```

Loading the collected statuses of Trump's tweets and the built model of NLP:

In [3]:
```
import pickle

with open("saved_model.pkl", "rb") as f:   #Unpickling
      sentiment_classifier = pickle.load(f)

with open("statuses.txt", "rb") as fp:
      statuses = pickle.load(fp)
```

The "text" in the "statuses" has already been processed by preprocess( ) and build_bag_of_words_features_filtered( ) which filter out the punctuation, the English stopwords, etc.

In [4]:
```
len(statuses), statuses[0] # test the loaded file
```

Out[4]:
```
(3013,
 {'Date': '2019-10-19',
  'text': {'one': 1,
   'pro-trump': 1,
   'demonstrator': 1,
   'began': 1,
   'hounding': 1,
   'impeachment': 1,
   'inquiry': 1}})
```

Remove the data of today since it is not over, the number of tweets may rise later:

In [5]:
```
from datetime import date
today = date.today()

def remove_today(L):
    ans = []
    for status in L:
        if status['Date'] != today.strftime("2019-10-19"): #2019-10-19 or today: %Y-%m-%d
            ans.append(status)
    return ans
```

In [6]:
```
statuses = remove_today(statuses)
```

In [7]:
```
len(statuses), statuses[:2]
```

Out[7]:
```
(3003,
 [{'Date': '2019-10-18',
   'text': {'republicans': 1,
    'must': 1,
    'stick': 1,
    'together': 1,
    'fight': 1}},
  {'Date': '2019-10-18',
   'text': {'think': 1,
    'many': 1,
    'lives': 1,
    'saved': 1,
    'syria': 1,
    'turkey': 1,
    'getting': 1,
    'ceasefire': 1,
    'yesterday': 1,
    'thousands': 1,
    'maybe': 1}}])
```

The shape looks good.

FREQUENT WORDS

Frequently we want to know which words are the most common in the tweets since we are looking for some patterns.
First, I will be collecting all the words from the statuses:

In [8]:
```
full_words = []
for status in statuses:
    full_words.extend(status['text'].keys())
len (full_words), full_words[:10]
```

Out[8]:
```
(40977,
 ['republicans',
  'must',
  'stick',
  'together',
  'fight',
  'think',
  'many',
  'lives',
  'saved',
  'syria'])
```

In [9]:
```
from collections import Counter
Counter(full_words).most_common(10)
```

Out[9]:
```
[('great', 489),
 ('president', 395),
 ('democrats', 252),
 ('thank', 245),
 ('people', 243),
 ('trump', 231),
 ('news', 215),
 ('country', 206),
 ('new', 191),
 ('big', 171)]
```

Use the built-in library Counter to get 100 most commonly used words in the tweets and generate a wordcloud, a cloud filled with lots of words in different sizes, which represent the frequency or the importance of each word:

In [10]:
```
from wordcloud import WordCloud

most_common_df_extend = pd.DataFrame(Counter(full_words).most_common(100), columns=['Word', 'Frequency'])

wordcloud = WordCloud(max_font_size=80, max_words=100, colormap="tab10", width=500, height=500,
                      background_color="white").generate(' '.join(most_common_df_extend['Word']))
# the image may be stretched into the figure and the default width and height makes a blurry image

fig = plt.figure()
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.figure(figsize=(10, 10)) # the size in jupyter

fig.savefig('/Users/yang/Desktop/graph_cloud.png')
plt.show()
```


```
<Figure size 720x720 with 0 Axes>
```

More examples of WordCloud:
https://amueller.github.io/word_cloud/auto_examples/index.html#example-gallery (https://amueller.github.io/word_cloud/auto_examples/index.html#example-gallery)

Next let's apply the sentiment_classifier to predict the emotions of those retrieved tweets.
In order to analyze the correlation between emotions and stock price changes numerically, I labeled each positive tweet 1 and negative one -1; 0 for neither positive nor negative.

In [11]:
```
for t in statuses:
    t['Predicted'] = sentiment_classifier.classify(t['text'])

    if sentiment_classifier.classify(t['text'])=='pos':
        t['Positiveness'] = 1
    elif sentiment_classifier.classify(t['text'])=='neg':
        t['Positiveness'] = -1
    else:
        t['Positiveness'] = 0
statuses[0] #sample test
```

Out[11]:
```
{'Date': '2019-10-18',
 'text': {'republicans': 1, 'must': 1, 'stick': 1, 'together': 1, 'fight': 1},
 'Predicted': 'neg',
 'Positiveness': -1}
```

Here I would like to build some interesting sub-datasets of Trump's tweets, filtering by a couple of "sensitive" words.

In [12]:
```
def searchkeyword(word, sta):
    ans = []
    for i in sta:
        if word in i['text']:
            ans.append(i)
    return ans
```

In [13]:
```
statuses_1 = searchkeyword('democrats', statuses)
statuses_2 = searchkeyword('china', statuses)
```

Load the data of statuses and make it a DataFrame. Let's call it df_Trump.

In [14]:
```
df_Trump = pd.DataFrame.from_dict(statuses).dropna()
df_Trump_1 = pd.DataFrame.from_dict(statuses_1)
df_Trump_2 = pd.DataFrame.from_dict(statuses_2)
df_Trump.head()
```

Out[14]:

	Date	text	Predicted	Positiveness
0	2019-10-18	{'republicans': 1, 'must': 1, 'stick': 1, 'tog...	neg	-1
1	2019-10-18	{'think': 1, 'many': 1, 'lives': 1, 'saved': 1...	pos	1
2	2019-10-18	{'also': 1, 'friend': 1, 'time': 1, 'pleased':...	pos	1
3	2019-10-18	{'want': 1, 'thank': 1, 'secretary': 1, 'energ...	pos	1
4	2019-10-18	{'believe': 1, 'important': 1, 'work': 1, 'cou...	neg	-1

In [15]:
```
len(df_Trump), len(df_Trump_1), len(df_Trump_2)
```

Out[15]:
```
(3003, 252, 110)
```

This result shows Trump was more interested in 'democrats' stuff than 'china'. He indeed took aim at Democrats on a range of issues.

How many times did Trump tweet a day?

Let's have a look at how many tweets he made on a daily basis using visualization tools.
First, I count the values of each date (how many tweets per day) and sort it by time:

In [16]:
```
time_data = df_Trump['Date'].value_counts().sort_index(ascending=True)
time_data[-5:]
```

Out[16]:
```
2019-10-14    30
2019-10-15    13
2019-10-16    24
2019-10-17    31
2019-10-18    19
Name: Date, dtype: int64
```

In [17]:
```
time_data.describe()
```

Out[17]:
```
count    143.000000
mean      21.000000
std       11.442446
min        3.000000
25%       13.000000
50%       18.000000
75%       28.000000
max       59.000000
Name: Date, dtype: float64
```

He tweeted average 21 tweets per day.

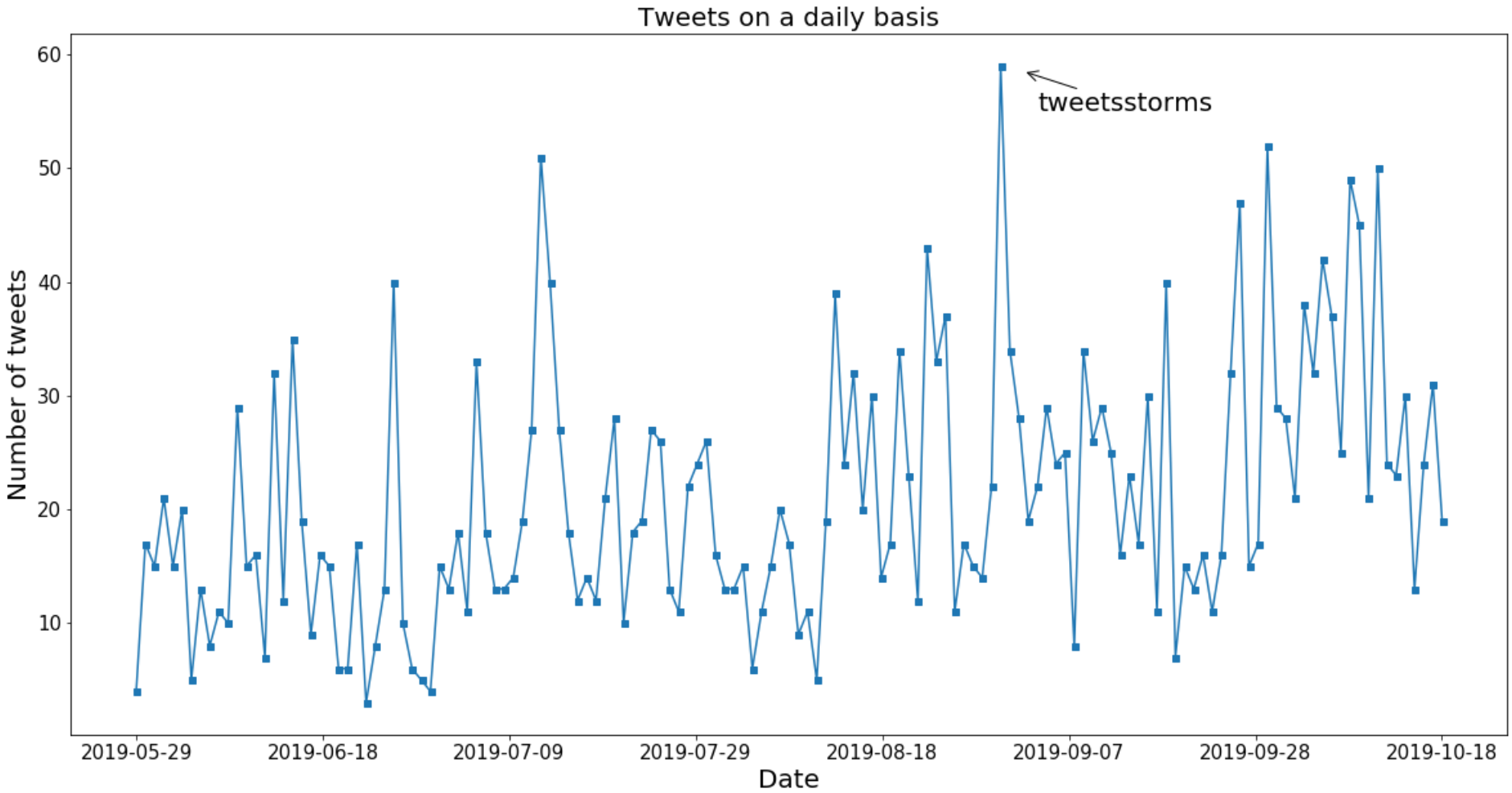
In [18]:
```
df_Trump['Date'].value_counts().sort_index(ascending=True).idxmax()
```

Out[18]:
```
'2019-08-31'
```

There was a major "tweetsstorms" on 31th, Aug. The Commander-in-Tweet naturally focused on the preparations in Florida for Hurricane Dorian, praising the crews that are preparing the emergency response should the storm strike populated areas.

In [19]:

```
fig = plt.figure(figsize=(20,10))
plt.plot(time_data, marker='s', markersize=5)

# Label the axes
plt.xlabel('Date', fontsize=20)
plt.ylabel('Number of tweets', fontsize=20)
plt.xticks(np.arange(0, len(time_data)+1, len(time_data)/7.05))
plt.tick_params(labelsize=15)

plt.title('Tweets on a daily basis', fontsize=20)

plt.annotate("tweetsstorms", fontsize=20,
             xy=(96.5, 58.5), xycoords='data',
             xytext=(98, 55), textcoords='data',
             arrowprops=dict(arrowstyle="->",
                             connectionstyle="arc3"),
               )

fig.savefig('/Users/yang/Desktop/graph.png')
plt.show()
```



The president’s market-moving tweeting “ballooned” in frequency in August 2019, when the China trade war and the Federal Reserve were top of the mind.

What's Trump's mood on Tweet today?
- Distribution

Next step I will analyze the correlation between President Trump's tweets and the changes of NYSE Composite they might cause.
First I calculate the "postiveness" of those tweets:

In [20]:
```
def Counter_df(df):
    df_copy = df.copy()
    df = df.groupby('Date', as_index=False)['Positiveness'].sum() # calculate the "summarized" sentiment of each day
    df['Count'] = df_copy['Date'].value_counts().sort_index(ascending=True).values # count the "Date" values

    return df
```

In [21]:
```
df_Trump = Counter_df(df_Trump)
df_Trump_1 = Counter_df(df_Trump_1)
df_Trump_2 = Counter_df(df_Trump_2)
df_Trump.head()
```

Out[21]:

	Date	Positiveness	Count
0	2019-05-29	0	4
1	2019-05-30	3	17
2	2019-05-31	9	15
3	2019-06-01	13	21
4	2019-06-02	-1	15

In [22]:
```
Sentiment_Range = [df_Trump.Positiveness.min(), df_Trump.Positiveness.max()]
Sentiment_Range
```

Out[22]:
```
[-14, 19]
```

So the positiveness changed dramatically ranging from -14 to the greatest 19.
Have a quick look at those sub-datasets:

In [23]:
```
df_Trump_1.head()
```

Out[23]:

	Date	Positiveness	Count
0	2019-05-30	0	2
1	2019-05-31	1	1
2	2019-06-02	-2	4
3	2019-06-04	-1	1
4	2019-06-05	-3	3

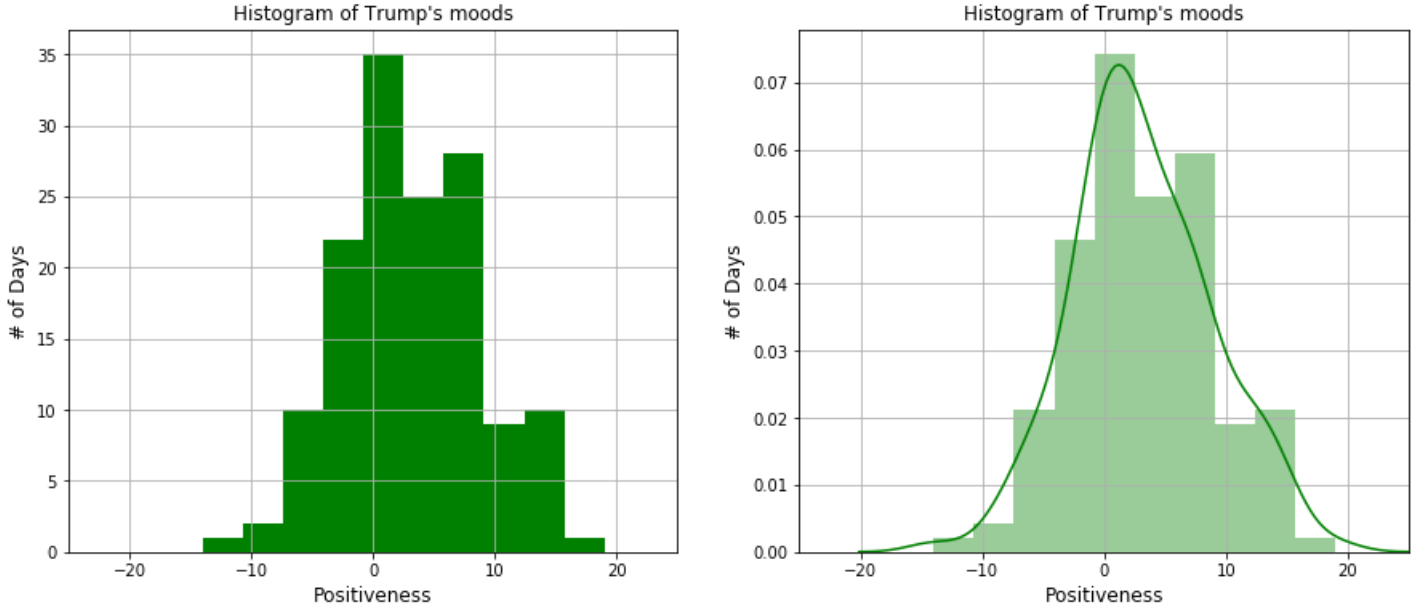
In [24]:
```
df_Trump_2.head()
```

Out[24]:

	Date	Positiveness	Count
0	2019-06-01	-1	1
1	2019-06-03	-1	1
2	2019-06-07	-1	1
3	2019-06-11	-1	1
4	2019-06-13	-1	1

Before we dive into the visualization part, let's have a look at the distribution of Trump's emotions across the year:

In [25]:
```
import seaborn as sns

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15,6))

ax[0].hist(df_Trump['Positiveness'], 10, facecolor='green')
ax[1] = sns.distplot(df_Trump['Positiveness'], bins=10, color='green')
# sns enables a default plot with a kernel density estimate

for a in ax:
    a.set_xlabel('Positiveness', fontsize=12)
    a.set_xlim(-25, 25)
    a.set_ylabel('# of Days', fontsize=12)
    a.set_title('Histogram of Trump\'s moods', fontsize=12)
    a.grid(True)

fig.savefig('/Users/yang/Desktop/graph00.png')
plt.show()
```



Use seaborn package to plot a density curve:
https://seaborn.pydata.org/generated/seaborn.distplot.html#seaborn.distplot (https://seaborn.pydata.org/generated/seaborn.distplot.html#seaborn.distplot)

It can be seen as a normal distribution with 'neutral' (positiveness=0) is the most common case.

How did trump's tweets shape the Stock Market?
- Sentiment revealed by tweets vs. NYSE Composite

Sentiment revealed by tweets vs. NYSE Composite

Join the two tables by Date - Trump's tweets and economy indexes:

In [26]:
```
joined = df_Trump.merge(df_NYA, on='Date', how='inner')
print (len(joined))
joined.tail()
```

```
101
```

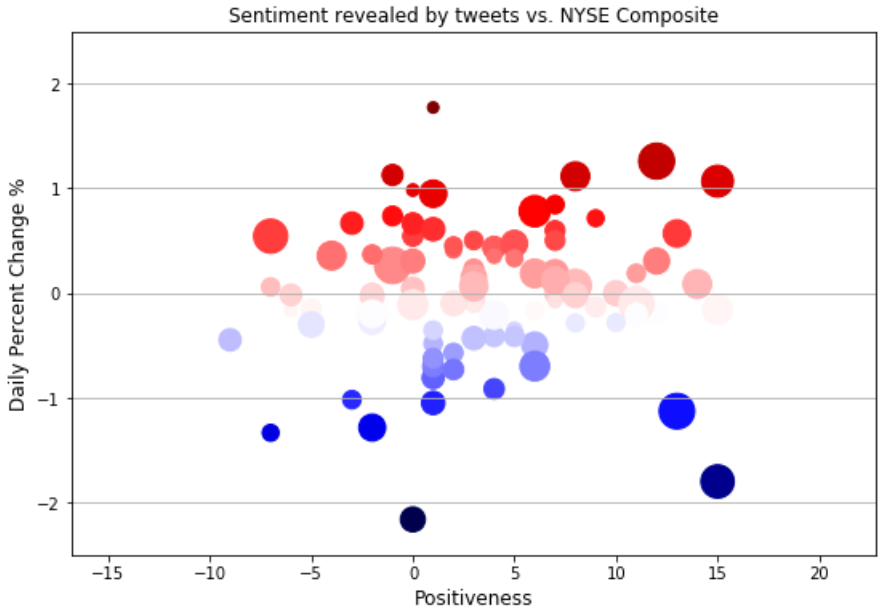
Out[26]:

	Date	Positiveness	Count	Open	High	Low	Close	Adj Close	Volume	Daily Percent Change %
96	2019-10-14	4	30	12923.480469	12923.480469	12883.959961	12896.219727	12896.219727	2557020000	-0.210940
97	2019-10-15	7	13	12897.030273	13044.830078	12897.030273	13006.040039	13006.040039	3340740000	0.845232
98	2019-10-16	8	24	12997.459961	13028.200195	12978.730469	12994.889648	12994.889648	3222570000	-0.019776
99	2019-10-17	3	31	13030.509766	13078.120117	13021.400391	13039.230469	13039.230469	3115960000	0.066925
100	2019-10-18	11	19	13032.469727	13044.080078	12972.839844	13006.639648	13006.639648	3264290000	-0.198198

Define a Scatter function for visualization:

In [27]:
```
def Scatter_Joined(df):
    fig, axis = plt.subplots(figsize=(9,6))
# Grid lines, Xticks, Xlabel, Ylabel

    axis.yaxis.grid(True)
    axis.set_title('Sentiment revealed by tweets vs. NYSE Composite',fontsize=12)
    axis.set_xlabel('Positiveness',fontsize=12)
    axis.set_xlim(1.2*Sentiment_Range[0], 1.2*Sentiment_Range[1])
    axis.set_ylabel('Daily Percent Change %',fontsize=12)
    axis.set_ylim(-2.5, 2.5)

    X = df['Positiveness']
    Y = df['Daily Percent Change %']
    area = df['Count']
    axis.scatter(X, Y, c=Y, s=10*area, cmap=plt.cm.seismic)

    fig.savefig('/Users/yang/Desktop/graph11.png')
    plt.show()
```

In [28]:
```
Scatter_Joined(joined)
```



The correlation coefficient is a measure of linear association between two variables. Values of the correlation coefficient are always between -1 and +1:

In [29]:
```
np.corrcoef(joined['Positiveness'], joined['Daily Percent Change %'])
```

Out[29]:
```
array([[1.        , 0.08996809],
       [0.08996809, 1.        ]])
```

There are apparent two outliers of the market's reaction towards the extraordinary mood Trump showed:

In [30]:
```
joined[(joined['Positiveness'] > 10) & (joined['Daily Percent Change %'] < -1.0)]
```

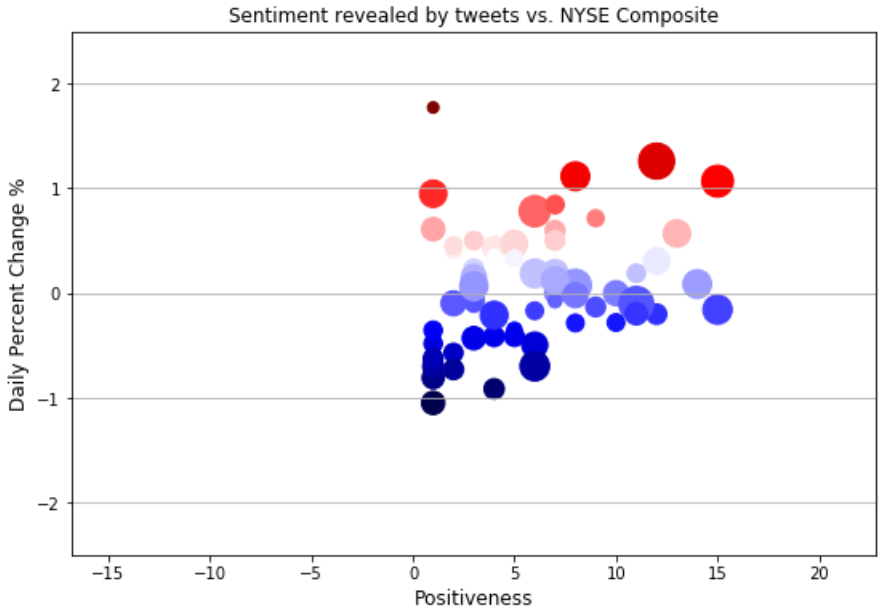
Out[30]:

	Date	Positiveness	Count	Open	High	Low	Close	Adj Close	Volume	Daily Percent Change %
61	2019-08-23	15	43	12643.940430	12710.650391	12370.070313	12416.450195	12416.450195	3937300000	-1.799204
92	2019-10-08	13	49	12734.549805	12734.549805	12589.809570	12590.910156	12590.910156	3356450000	-1.127952

What happeneded on those days? For example, On 23rd, Aug, China announced new tariffs Friday on another $75 billion worth of American goods. Trump has repeatedly blamed the Fed for concerns about a slowing U.S. economy and argued his trade war with China.

Let's remove the outliers and see the correlation of the president's tweets having a very positive sense and the stock market:

In [31]:
```
joined_out = joined.drop(index = [61, 92])
joined_filtered_1 = joined_out[joined_out.Positiveness>0]
Scatter_Joined(joined_filtered_1)
```



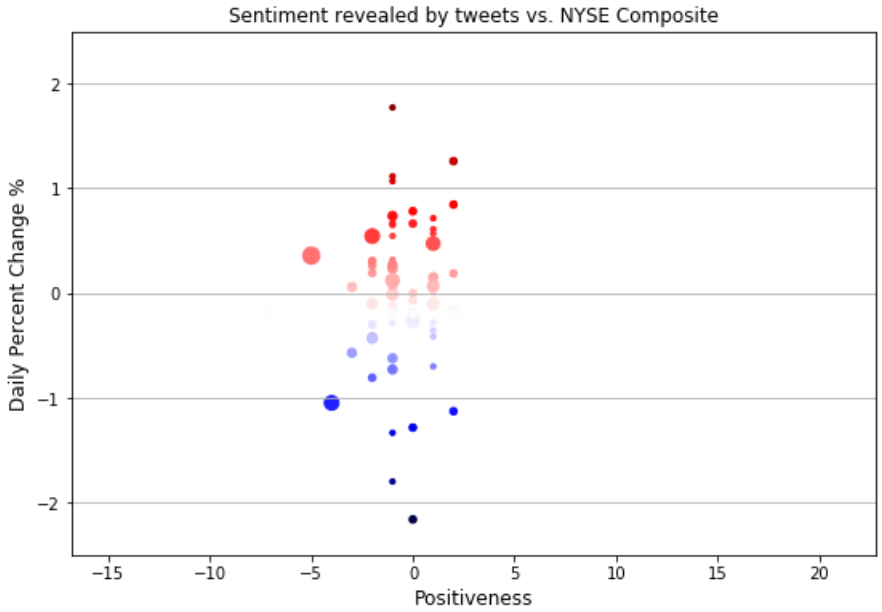
In [32]:
```
np.corrcoef(joined_filtered_1['Positiveness'], joined_filtered_1['Daily Percent Change %'])
```

Out[32]:
```
array([[1.        , 0.26775861],
       [0.26775861, 1.        ]])
```

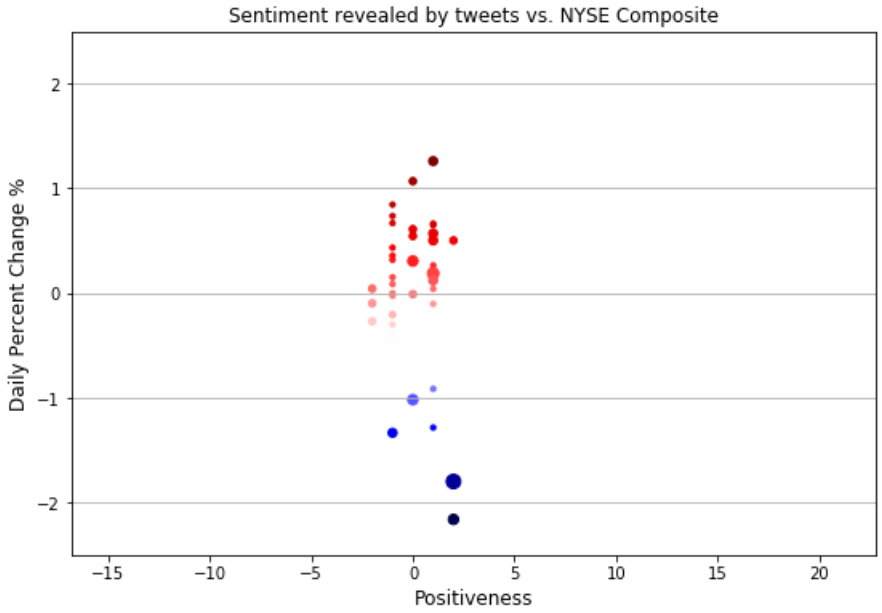
The potential correlation becomes more solid after withdrawing a few outliers.

In [33]:
```
joined_1 = df_Trump_1.merge(df_NYA, on='Date', how='inner')
joined_2 = df_Trump_2.merge(df_NYA, on='Date', how='inner')
```

In [34]:
```
Scatter_Joined(joined_1) #'democrats'
```



In [35]:
```
Scatter_Joined(joined_2) #'china'
```



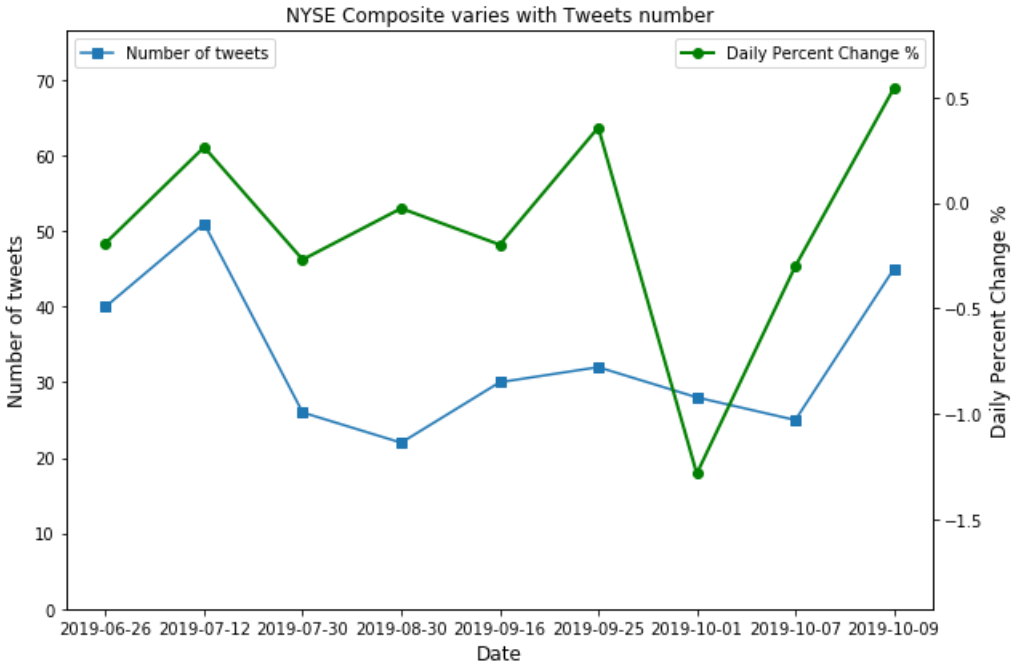
NYSE Composite varies with Tweets number

In [36]:
```
def Line_Joined(df):
    fig, ax1 = plt.subplots(figsize=(9, 6))

    ax1.plot(df.Date, df['Count'], marker='s', label = 'Number of tweets')
    ax1.set_xlabel('Date', fontsize=12)
    ax1.set_xticks(np.arange(0, len(df.Date)+1, len(df.Date)/7.05))
    ax1.set_ylabel('Number of tweets', fontsize=12)
    ax1.set_ylim(0, 1.5*df['Count'].max())
    ax1.set_title('NYSE Composite varies with Tweets number', fontsize=12)
    ax1.legend(loc = 'upper left', fontsize='medium', fancybox=True)

    ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
    ax2.set_ylabel('Daily Percent Change %', fontsize=12)  # again on Y
    ax2.set_ylim(1.5*df['Daily Percent Change %'].min(), 1.5*df['Daily Percent Change %'].max())
    ax2.plot(df.Date, df['Daily Percent Change %'], 'g', marker='o', linewidth=2, label = 'Daily Percent Change %')
    #ax2.tick_params(axis='y', labelcolor=color)
    ax2.legend(loc = 'upper right', fontsize='medium', fancybox=True)
    fig.tight_layout() # adjust subplot to fit the figure area
    fig.savefig('/Users/yang/Desktop/graph22.png')
    return plt.show()
```

Plots with different scales:
https://matplotlib.org/gallery/api/two_scales.html (https://matplotlib.org/gallery/api/two_scales.html)

In [37]:
```
joined_filtered_1 = joined[(joined['Count']>20) & (joined['Positiveness']<0)]
```

In [38]:
```
Line_Joined(joined_filtered_1)
```



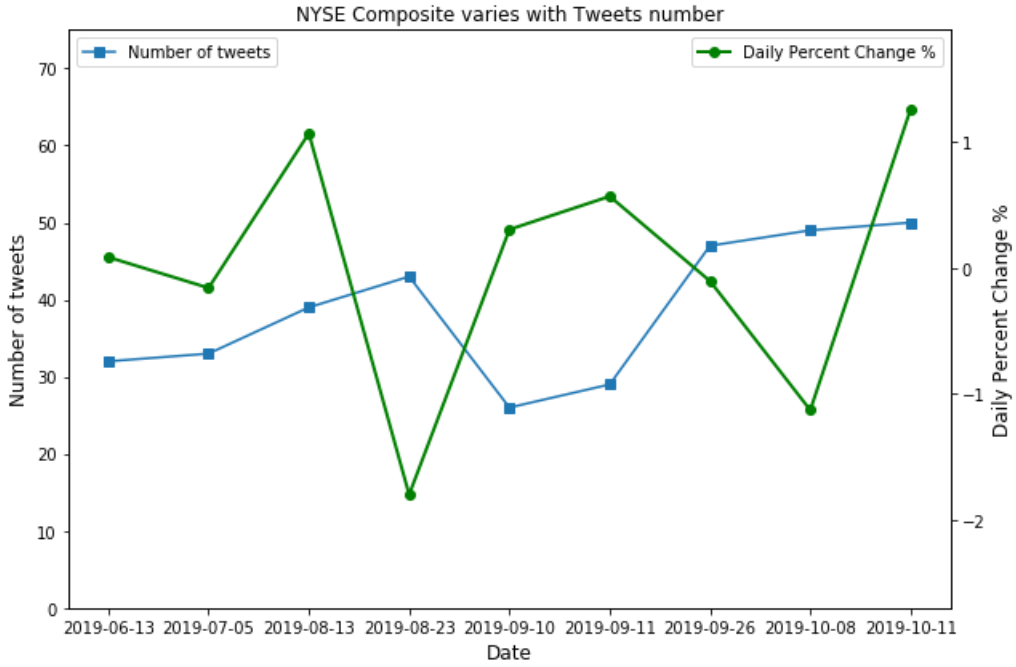
In [39]:
```
np.corrcoef(joined_filtered_1['Count'], joined_filtered_1['Daily Percent Change %'])
```

Out[39]:
```
array([[1.        , 0.50995762],
       [0.50995762, 1.        ]])
```

It shows there is a strong linear relationship between the two variables. On days with more than 20 Trump tweets generally showing a negative sense, the stock market varied with the number accordingly.

In [40]:
```
joined_filtered_2 = joined[(joined['Count']>20) & (joined['Positiveness']>10)]
```

In [41]:
```
Line_Joined(joined_filtered_2)
```



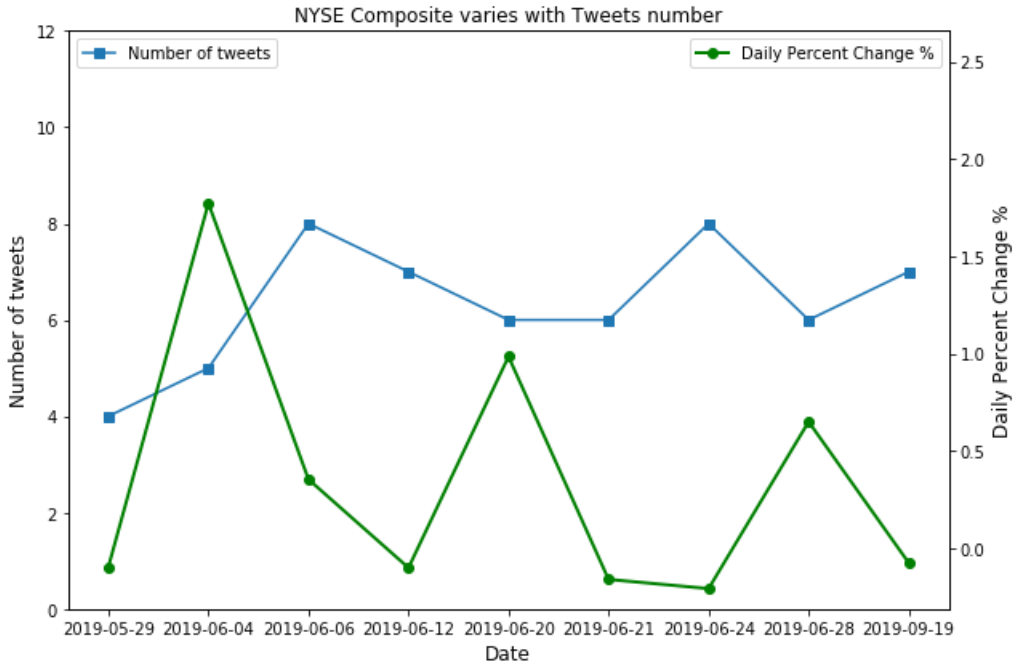
In [42]:
```
np.corrcoef(joined_filtered_2['Count'], joined_filtered_2['Daily Percent Change %'])
```

Out[42]:
```
array([[ 1.        , -0.20763074],
       [-0.20763074,  1.        ]])
```

The relationship is weaker compared to the negative situation. His positive tweets become more "neutral" for investors to make decisions.

In [43]:
```
joined_filtered_3 = joined[(joined['Count']<9)]
```

In [44]:
```
Line_Joined(joined_filtered_3)
```



On days with fewer than 9 Trump tweets, the NYSE Composite fluctuated.

In [ ]: