Sentiment Trading Project

Ethan, Amanda, Josephine, Quratul, Sanjit

Project objective:

Use NLP techniques to label tweets with their sentiment, and use these labels as trading signals

Data Collection and Initial Analysis

Collecting Data

Method

Library: twint & yfinance

- twint: non-rate-limited library for scraping tweets
- Yfinance: Yahoo Finance's API
  - note: highest frequency remains daily data, which was a potential roadblock
- command line output →

```
# scrape from yfinance
yfin = yf.Ticker(ticker_name)
yfin_historical = yfin.history(start=start_date, end=end_date, interval="1d")
yfin_historical.to_csv(price_filename)

# scrape from Twitter
c = twint.Config()
c.Since = start_date
c.Until = end_date
c.Search = keywords
# c.Limit = num_input
c.Store_json = True
c.Output = tweet_json_filename
twint.run.Search(c)
```

```
1155527688989347840 2019-07-28 18:17:24 +0000 <BlockFxGold> For Bitcoin Pric
e &amp;amp; Working, Go to Chinese Central Bank Website! -  https://t.co/iWwuW35
CWn  https://t.co/TlOs2BUtWF
1155527657183801354 2019-07-28 18:17:16 +0000 <WillEgoist> You have to love
potentially earning millions of #dollars with #limitedeffort #picoin #crypto
 the #cash of the future. I am not going to miss the #bitcoin train again
1155527642189160448 2019-07-28 18:17:13 +0000 <Sinoneon16> @elad_network #EL
ADNetwork is an excellent project with a unique idea. The team behind #ELADN
etwork is perfect. They will take this project to the top level. So don't he
sitate to participate here. #realestate #blockchain #property #bitcoin
```

Twint Output

id	created_at	date	time	datetime	username	tweet	replies_count	retweets_count	likes_count	hashtags	cashtags	?	!
198473676476416	2020-11-29 23:57:40 GMT	2020-11-29	23:57:40	2020-11-29 23:57:40	tslato500	ValueAnalyst1 Gary why did the stock split cr...	1.0	0.0	1.0	[]	[]	0	0
197797894598664	2020-11-29 23:54:59 GMT	2020-11-29	23:54:59	2020-11-29 23:54:59	tigersharktrade	Thinking about playing some midlate December c...	2.0	1.0	4.0	[tesla, tsla]	[TSLA]	1	0
196495273816066	2020-11-29 23:49:48 GMT	2020-11-29	23:49:48	2020-11-29 23:49:48	sonalgupta0297	Dow Jones SampP 500 and Nasdaq advanced on a w...	0.0	0.0	0.0	[]	[]	0	0
194046517497857	2020-11-29 23:40:04 GMT	2020-11-29	23:40:04	2020-11-29 23:40:04	401ktimer	It doesnt matter if believes that Tesla stock...	0.0	0.0	1.0	[tesla]	[TSLAQ, TSLA]	0	0
192832551690246	2020-11-29 23:35:15 GMT	2020-11-29	23:35:15	2020-11-29 23:35:15	brown_te	BusinessFamous 90 year SampP 500 chart Subst...	0.0	0.0	1.0	[]	[]	0	0

YFinance OHLC Average

```
# pdf is the dataframe with the price data

# find OHLC average of price data
pdf['avg']=(pdf['Open']+pdf['High']+pdf['Low']+pdf['Close'])/4
pdf = pdf[['Date','avg']]

# create price_df

pdf['date']=pdf['Date'].apply(lambda x: datetime.strptime(x,'%Y-%m-%d'))
pdf = pdf.set_index('date').asfreq('1D')
pdf['avg']=pdf['avg'].interpolate(method='linear')
```

date	Date	avg
2020-09-30	2020-09-30	3359.5600
2020-10-01	2020-10-01	3381.3100
2020-10-02	2020-10-02	3345.0375

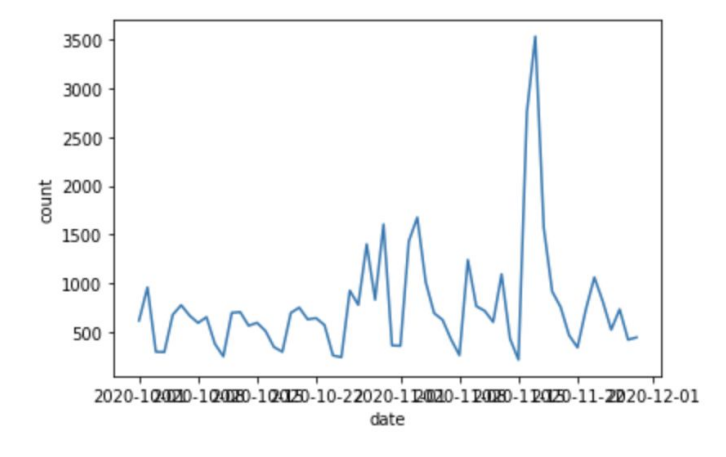
Yfinance Data

Using OHLC Average:

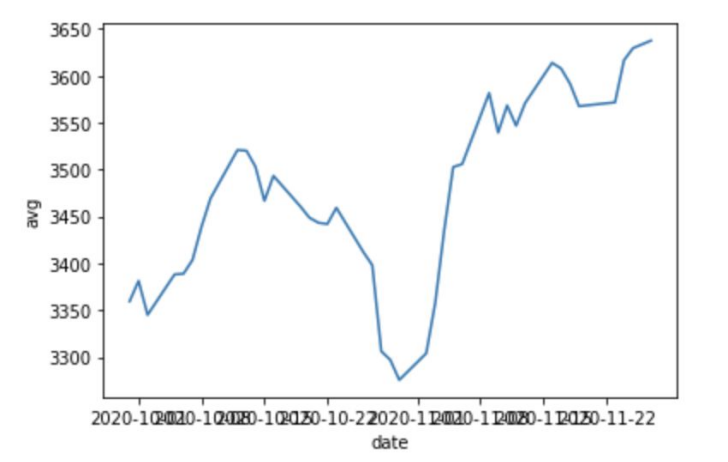
- OHLC Average is the average of Open, High, Low and Close Data
- Linear interpolation
  - No data on weekends
- Final dataframe for price data as shown

Plotting Tweets and Stock Price Over Time

Tweets over Time



OHLC avg. over Time



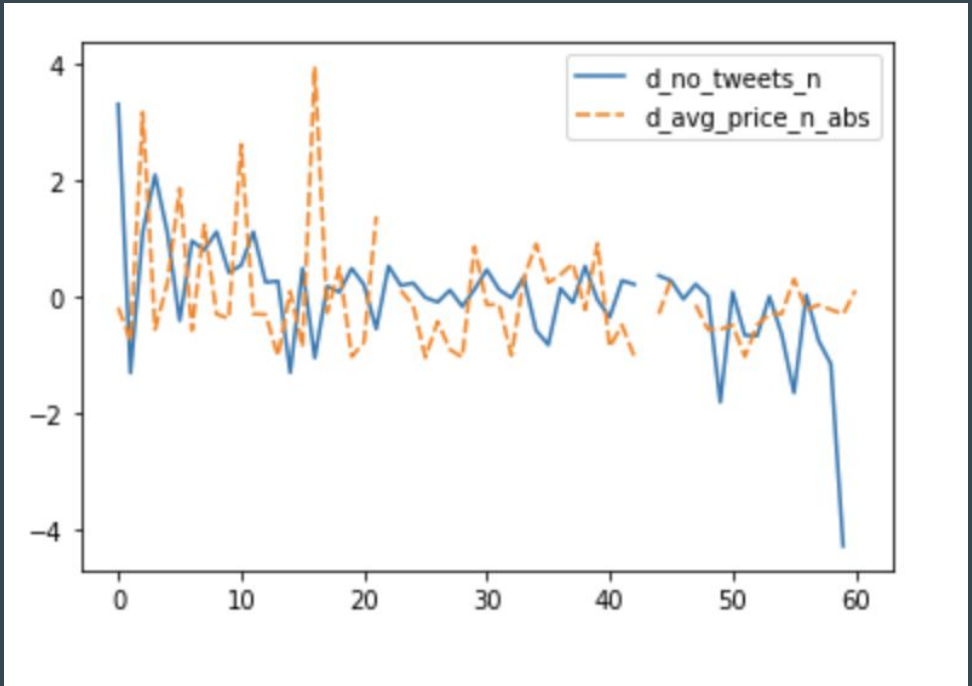
Analysing Data

```
# stopwords for the wordcloud
stopwords = set(STOPWORDS)
stopwords.update(['SampP'])

# maximum number of words, etc.
wordcloud = WordCloud(max_words=100,background_color="white", width=200,height=200,stopwords=stopwords).generate(txt)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

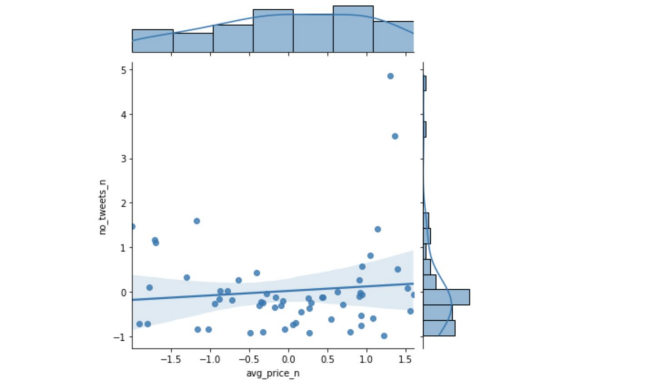
Wordcloud

- Input stopwords (where relevant)
- Generates a wordcloud of the most common words that appear within the tweets

Normalised Change in No. of Tweets and Change in Price



- Number of tweets generally not a good indicator of change in price
- Small change in number of tweets → large change in price



Reducing Spam for better Sentiment Analysis Outcomes

Filter by X Cashtags

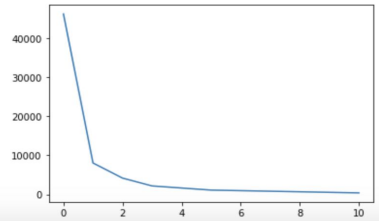
```
df['n_cashtags']=df['cashtags'].apply(lambda x: len(x))
```

```
# graph of number of hashtags

x_cashtags = []
cashtags=[0, 1, 2, 3, 5, 10]

for i in cashtags:
    iscashtagval = (df['n_cashtags']>=i)
    x_cashtags.append(len(df.loc[iscashtagval].copy()))

x_cashtags
```

```
[46253, 8044, 4173, 2166, 1101, 388]
```

```
sns.lineplot(x=cashtags, y=x_cashtags, markers=True)
```

```
<AxesSubplot:>
```



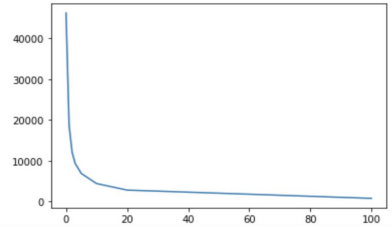
Filtering by Number of Likes

```
# graph of number of tweets

x_likes = []
likes=[0, 1, 2, 3, 5, 10, 20, 100]

for i in likes:
    islikeval = (df['likes_count']>=i)
    x_likes.append(len(df.loc[islikeval].copy()))

x_likes
```

```
[46253, 18657, 12024, 9332, 6881, 4411, 2807, 787]
```

```
sns.lineplot(x=likes, y=x_likes, markers=True)
```

```
<AxesSubplot:>
```



Filter by X Hashtags

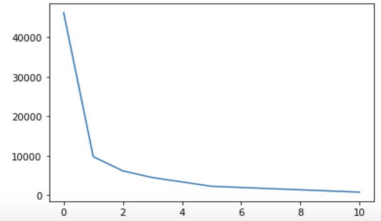
```
df['n_hashtags']=df['hashtags'].apply(lambda x: len(x))
```

```
# graph of number of hashtags

x_hashtags = []
hashtags=[0, 1, 2, 3, 5, 10]

for i in hashtags:
    ishashtagval = (df['n_hashtags']>=i)
    x_hashtags.append(len(df.loc[ishashtagval].copy()))

x_hashtags
```

```
[46253, 9769, 6170, 4476, 2264, 780]
```

```
sns.lineplot(x=hashtags, y=x_hashtags, markers=True)
```

```
<AxesSubplot:>
```



- aiming to reduce spam caused by ads, tweets with > Cashtag

Text cleaning

Text Cleaning

```
#Doesn't work with tsla.csv, since this contains fewer columns
#Designed to work with tweet_^GSPC_20201001_20201130.csv and similar files

#c is the column name of the column containing tweets

def mainCleaner(df, c):
  pre_len = len(df)
  df = df[df['language'].isin(['en'])]

  post_len = len(df)
  print('length before: {}, length after keeping only english tweets: {}'.format(pre_len, post_len))

  # strings to lists (where applicable)
  pre_type = type(df['cashtags'].loc[1])
  to_list_col = ['mentions','urls','photos','hashtags','cashtags']

  for col in to_list_col:
      df[col] = df[col].apply(lambda x: ast.literal_eval(x))

  post_type = type(df['cashtags'].loc[1])
  print('type before: {}, type after conversion: {}'.format(pre_type, post_type))

  # count punctuation '!', '?'
  df['?'] = df[c].apply(lambda x: x.count('?'))
  df['!'] = df[c].apply(lambda x: x.count('!'))
```

c = column in df containing the tweets

Defined a function called main Cleaner:

- Filters non English tweets
- Converts strings to lists
- Tallies the frequency of punctuation marks

Removing the noise

```
# remove URLs, mentions, cashtags, punctuation
def clean_tweet(i):
  t = df[c][i]

  # remove URLS
  for u in df['urls'][i]:
      t = t.replace(u, "")

  # remove mentions
  for m in df['mentions'][i]:
      m = '@' + m
      t = t.replace(m, "")

  # remove cashtags
  for k in df['cashtags'][i]:
      k = '$' + k
      t = t.replace(k, "")

  # reassign
  df[c][i] = t

  for i in df.index:
      clean_tweet(i)


# remove punctuation
df[c] = df[c].apply(lambda x: re.sub(r'[^\w\s]', '', x))

# remove urls from tweets
df[c] = df[c].apply(lambda x: re.sub("(#[A-Za-z0-9]+)|"
  + "(\\$[A-Za-z]+)|(@[A-Za-z0-9]+)|http[^ ]+|/\r?\n|\r/}", "", x))
```

Defined a function called clean_tweet:

- Takes input i
- From the dataframe:
  - Removes urls
  - Removes mentions &
  - Removes cashtags
- From the tweets column:
  - Removes punctuation &
  - Removes urls

Stop words, lemmatization

```
# convert time to datetime
df['datetime'] = df['date'] + " " + df['time']
df['datetime'] = df['datetime'].apply(lambda dt:(datetime.strptime(dt,'%Y-%m-%d %H:%M:%S')))

print("unique timezones: {}".format(df['timezone'].nunique()))
print("no need to change to timezone aware object")

# keep only relevant columns
df = df[['datetime','tweet','replies_count','retweets_count','likes_count','?','!']]
#df.dtypes

# remove short words, length <=3
df[c] = df[c].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))

def lemmatizer(text):
  lem_text = [WordNetLemmatizer().lemmatize(i) for i in text]
  return lem_text

df[c] = lemmatizer(df[c])

#stemming
# separate into individual words, ie tokenization
tokenized_tweet = df[c].apply(lambda x: x.split())
stemmer = PorterStemmer()
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x])
tokenized_tweet = tokenized_tweet.reset_index(drop=True)
for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i])
df[c] = tokenized_tweet
'''

# drop NaN tweets
df = df[df[c].notna()]
df = df.reset_index(drop = True)

return df
```

- Removed stop words, characterised by short length <=3

- Initially used stemming
- Later used lemmatization for more accuracy
- Examples of stemming:
  - Observed -> observ
  - Hold - > hold
  - Held -> held
  - Continue - > continu
- Examples of Lemmatization:
  - Observed -> observe
  - Hold -> hold
  - Held -> hold
  - Continue -> continue

Refining the data

```
#Restricting to tweets with at least one like and one retweet
#the dataset is too big right now
#This may improve tweet quality as well

df_clean = df_clean[df_clean['likes_count'] > 0]
df_clean = df_clean[df_clean['retweets_count'] > 0]
```

```
df_clean
```

The cleaned data frame looks like...

	datetime	tweet	replies_count	retweets_count	likes_count	?	!
1	2020-11-29 23:54:59	Thinking about playing some midlate December c...	2.0	1.0	4.0	1	0
8	2020-11-29 23:23:52	Apparently even after everything thats happene...	3.0	1.0	11.0	0	0
11	2020-11-29 23:23:04	sectors have SCTR rank above SampP itself Indu...	0.0	4.0	23.0	0	0
21	2020-11-29 23:05:04	SampP Nasdaq Composite Russell 2000 MSCI World...	2.0	1.0	6.0	0	0
22	2020-11-29 23:02:01	Earnings Trade Idea 550C bidask 11001190 Close...	4.0	8.0	51.0	0	0
...	...	...	...	...	...	...	...
47347	2020-10-01 04:25:42	dawson_Stock Looking back performance SampP In...	1.0	1.0	1.0	0	0
47375	2020-10-01 02:54:46	Best performing energy companies SampP Cabot C...	0.0	1.0	1.0	0	0
47376	2020-10-01 02:46:08	2020 Allen Crowe Race Purse 1st6000 2nd4000 3r...	0.0	3.0	16.0	0	0
47393	2020-10-01 01:45:38	Since 1988 when consumer confidence been above...	1.0	2.0	11.0	0	0
47397	2020-10-01 01:04:24	September Portfolio Update SampP Portfolio 522...	7.0	8.0	87.0	0	3

8201 rows × 7 columns

- Eliminated tweets with zero likes and retweets
- To remove:
  - spam
  - adverts
  - low quality data

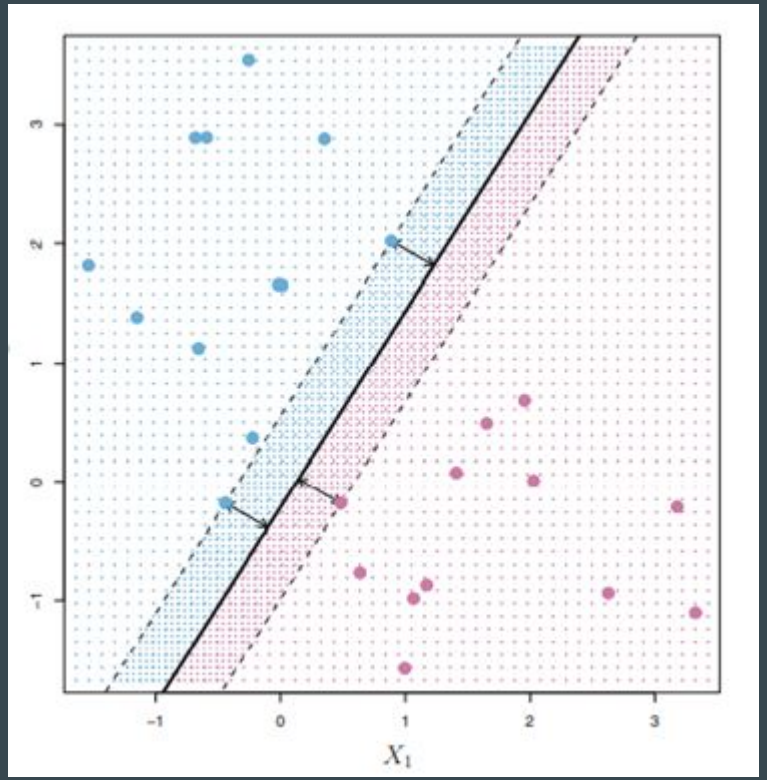
Support Vector Machines

Motivation

- Classification task with 3 labels: Bullish, Bearish, Neutral
  - SVMs can be extended to multiple classes using one-versus-one or one-versus-all classification
- High-dimensional dataset: many word vectors, few training observations!
  - SVMs work well with sparse, high-dimensional datasets, as it only needs a few observations (support vectors) to fit

Maximal Margin Classifier

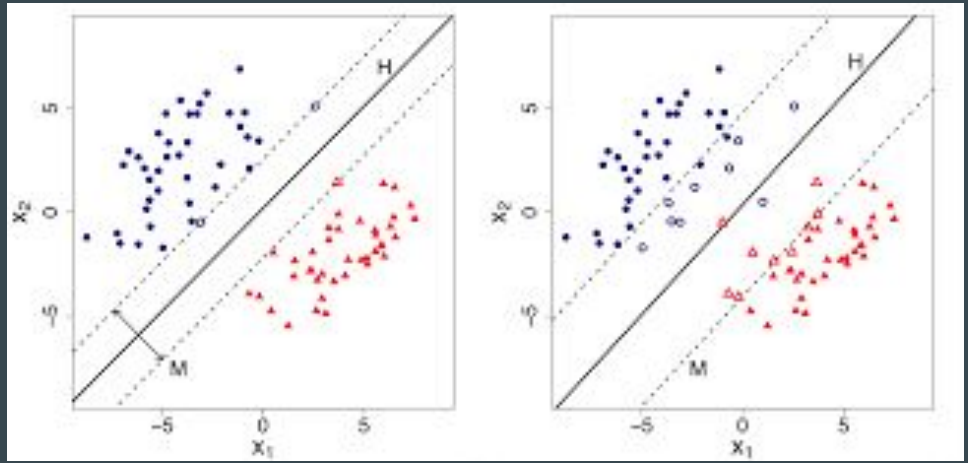
- Use a hyperplane to separate two classes of observations
- If our data *can be perfectly separated*, then the maximal margin classifier is the one that has maximum perpendicular distance from each training observation
    - It represents the mid-line of the largest “slab” we can fit in between two classes pink and blue.
- Support vectors are the observations that lie on the dotted lines (the margin)
    - In our optimization problem, they have the largest impact on the position of the hyperplane



What if our data can’t be perfectly separated?

Support Vector Classifier

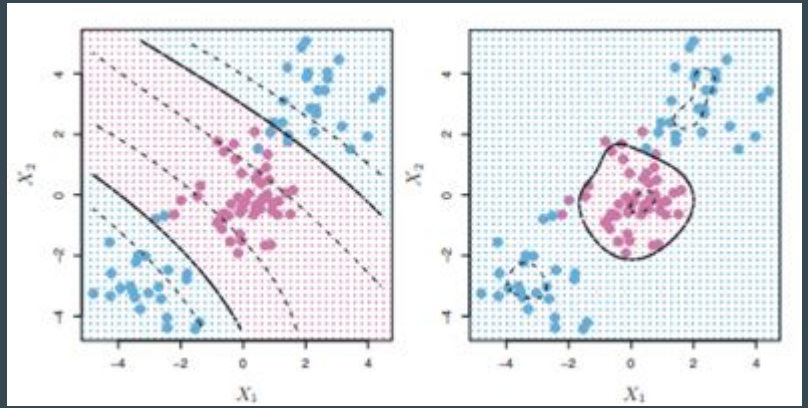
- We allow some leeway when fitting the classifier, letting some points fall into the margin
  - We have a “soft margin”
  - Trained in a way that some training observations may end up inside the margin, or even on the wrong side of the hyperplane.
- This prevents the classifier from being overly sensitive (overfitting)



What if the decision boundary is non-linear?

Kernels & Support Vector Machines

- To fit a support vector classifier, we only need the pairwise inner products of the SUPPORT VECTORS
- To fit a non-linear decision boundary, we can replace inner product with a generalization - we call this a Kernel
    - K just needs to be a function that quantifies how similar two vectors are.
    - We can have polynomial kernels or radial kernels



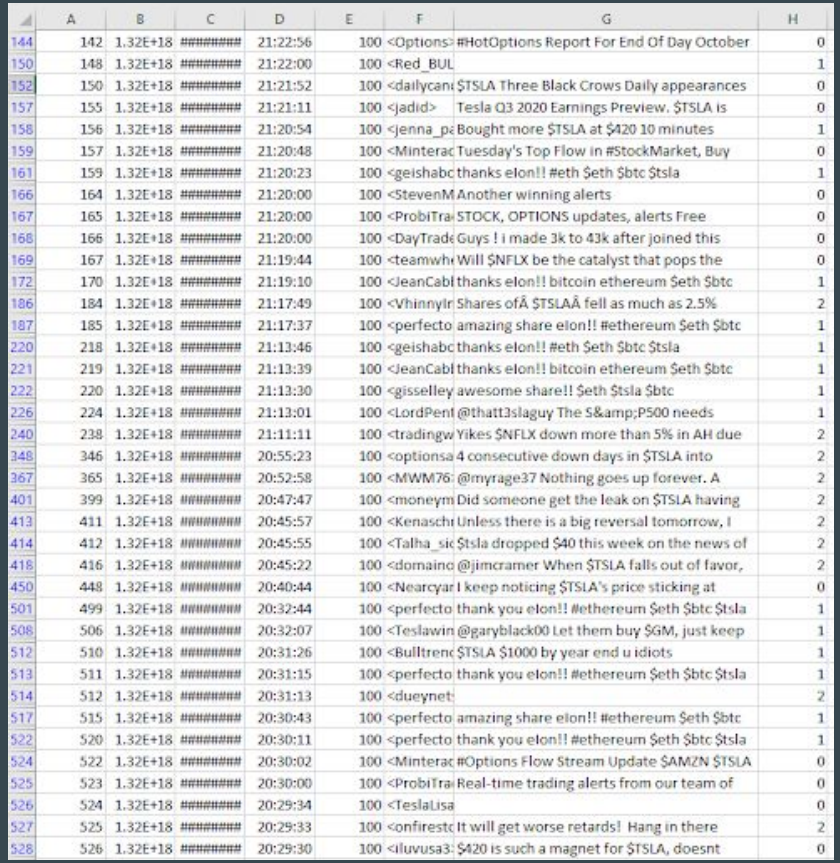
Tweet Labelling

SVMs are a supervised learning algorithm, so we have to supply it with a labelled dataset

Labelled a total of ~200 tweets (quite small)

This was taking quite long so we tried using a stock market sentiment dataset from kaggle that was scraped from twitter

https://www.kaggle.com/yash612/stockmarket-sentiment-dataset



	A	B	C	D	E	F	G	H
144	142	1.32E+18	########	21:22:56	100	<Options>	#HotOptions Report For End Of Day October	0
150	148	1.32E+18	########	21:22:00	100	<Red_BUL		1
152	150	1.32E+18	########	21:21:52	100	<dailycan	$TSLA Three Black Crows Daily appearances	0
157	155	1.32E+18	########	21:21:11	100	<jadid>	Tesla Q3 2020 Earnings Preview. $TSLA is	0
158	156	1.32E+18	########	21:20:54	100	<jenna_pa	Bought more $TSLA at $420 10 minutes	1
159	157	1.32E+18	########	21:20:48	100	<Minterac	Tuesday's Top Flow in #StockMarket, Buy	0
161	159	1.32E+18	########	21:20:23	100	<geishabc	thanks elon!! #eth $eth $btc $tsla	1
166	164	1.32E+18	########	21:20:00	100	<StevenM	Another winning alerts	0
167	165	1.32E+18	########	21:20:00	100	<ProbiTra	STOCK, OPTIONS updates, alerts Free	0
168	166	1.32E+18	########	21:20:00	100	<DayTrade	Guys ! i made 3k to 43k after joined this	0
169	167	1.32E+18	########	21:19:44	100	<teamwh	Will $NFLX be the catalyst that pops the	0
172	170	1.32E+18	########	21:19:10	100	<JeanCabl	thanks elon!! bitcoin ethereum $eth $btc	1
186	184	1.32E+18	########	21:17:49	100	<VhinnyIr	Shares ofÃ $TSLAÃ fell as much as 2.5%	2
187	185	1.32E+18	########	21:17:37	100	<perfecto	amazing share elon!! #ethereum $eth $btc	1
220	218	1.32E+18	########	21:13:46	100	<geishabc	thanks elon!! #eth $eth $btc $tsla	1
221	219	1.32E+18	########	21:13:39	100	<JeanCabl	thanks elon!! bitcoin ethereum $eth $btc	1
222	220	1.32E+18	########	21:13:30	100	<gisselley	awesome share!! $eth $tsla $btc	1
226	224	1.32E+18	########	21:13:01	100	<LordPent	@thatt3slaguy The S&P500 needs	1
240	238	1.32E+18	########	21:11:11	100	<tradingw	Yikes $NFLX down more than 5% in AH due	2
348	346	1.32E+18	########	20:55:23	100	<optionsa	4 consecutive down days in $TSLA into	2
367	365	1.32E+18	########	20:52:58	100	<MWM76:	@myrage37 Nothing goes up forever. A	2
401	399	1.32E+18	########	20:47:47	100	<moneym	Did someone get the leak on $TSLA having	2
413	411	1.32E+18	########	20:45:57	100	<Kenaschi	Unless there is a big reversal tomorrow, I	2
414	412	1.32E+18	########	20:45:55	100	<Talha_sic	$tsla dropped $40 this week on the news of	2
418	416	1.32E+18	########	20:45:22	100	<domainc	@jimcramer When $TSLA falls out of favor,	2
450	448	1.32E+18	########	20:40:44	100	<Nearcyar	I keep noticing $TSLA's price sticking at	0
501	499	1.32E+18	########	20:32:44	100	<perfecto	thank you elon!! #ethereum $eth $btc $tsla	1
508	506	1.32E+18	########	20:32:07	100	<Teslawin	@garyblack00 Let them buy $GM, just keep	1
512	510	1.32E+18	########	20:31:26	100	<Bulltrenc	$TSLA $1000 by year end u idiots	1
513	511	1.32E+18	########	20:31:15	100	<perfecto	thank you elon!! #ethereum $eth $btc $tsla	1
514	512	1.32E+18	########	20:31:13	100	<dueynet		2
517	515	1.32E+18	########	20:30:43	100	<perfecto	amazing share elon!! #ethereum $eth $btc	1
522	520	1.32E+18	########	20:30:11	100	<perfecto	thank you elon!! #ethereum $eth $btc $tsla	1
524	522	1.32E+18	########	20:30:02	100	<Minterac	#Options Flow Stream Update $AMZN $TSLA	0
525	523	1.32E+18	########	20:30:00	100	<ProbiTra	Real-time trading alerts from our team of	0
526	524	1.32E+18	########	20:29:34	100	<TeslaLisa		0
527	525	1.32E+18	########	20:29:33	100	<onfirestc	It will get worse retards! Hang in there	2
528	526	1.32E+18	########	20:29:30	100	<iluvusa3	$420 is such a magnet for $TSLA, doesnt	0

Tokenization, cleaning

Tokenization etc.

```
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import LabelEncoder
from collections import defaultdict
from nltk.corpus import wordnet as wn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import model_selection, naive_bayes, svm
from sklearn.metrics import accuracy_score
```

```
Corpus=dfCleaned
```

```
import string
#change all text to lower case
Corpus['tweet'] = [entry.lower() for entry in Corpus['tweet']]
Corpus['tweet'] = [re.sub(r'[^\w\s]', '', s) for s in Corpus['tweet']]
```

```
Corpus
```

	date	time	tweet	label
0	20/10/2020	21:42:42	any thoughts on for tomorrow er\n	0.0
2	20/10/2020	21:42:32	thanks eion bitcoin ethereum	1.0
3	20/10/2020	21:42:32	amazing share eion	1.0
4	20/10/2020	21:42:31	awesome share	1.0
5	20/10/2020	21:42:26	thanks eion	1.0
...	...	...	...	...
867	20/10/2020	19:26:10	most profitable ...	0.0
868	20/10/2020	19:26:00	choose this professional trading chatroom ...	0.0
880	20/10/2020	19:22:00	for best alerts daily join us discord ...	0.0
885	20/10/2020	19:21:17	why you should buy tesla etfs ahead of q3 earn...	0.0
888	20/10/2020	19:20:38	most ...	0.0

Training-test split, word vectorization

```
[14]: # Training test split of 70:30
trainX, testX, trainY, testY = model_selection.train_test_split(Corpus['tweet'],Corpus['label'],test_size=0.3)

[15]: encoder = LabelEncoder()
trainY = encoder.fit_transform(trainY)
testY = encoder.fit_transform(testY)

[16]: # Vectorize the words (max 5000 unique words)
# Use a TF-IDF method, (Term frequency and Inverse Document Frequency)
tfidfVect = TfidfVectorizer(max_features=5000)
tfidfVect.fit(Corpus['tweet'])
trainXTfidf = tfidfVect.transform(trainX)
testXTfidf = tfidfVect.transform(testX)

[17]: print(tfidfVect.vocabulary_) # Vocab from the corpus

{'any': 58, 'thoughts': 431, 'on': 309, 'for': 181, 'tomorrow': 438, 'er': 158, 'thanks': 423, 'eion': 150, 'bitcoin': 80, 'ethereum': 162, 'amazing': 53, 'share': 378, 'awesome': 70, 'd
eliver': 131, 'feature': 174, 'that': 424, 'lets': 259, 'you': 488, 'summon': 410, 'your': 489, 'car': 97, 'from': 189, 'anywhere': 62, 'connected': 116, 'by': 92, 'land': 248, 'in': 23
1, 'spite': 401, 'of': 307, 'shortseller': 383, 'fud': 191, 'force': 182, 'majeure': 276, 'earnings': 148, 'after': 48, 'hours': 221, 'feels': 175, 'like': 260, 'elon': 153, 'is': 237,
'about': 45, 'to': 434, 'roast': 363, 'the': 425, 'shorts': 382, 'holding': 219, 'calls': 95, 'through': 432, 'join': 241, 'most': 292, 'profits': 340, 'trading': 440, 'alerts': 50, 'cha
troomgt': 105, 'gm': 198, 'ford': 183, 'and': 55, 'auto': 68, 'parts': 320, 'suppliers': 411, 'led': 256, 'industrial': 234, 'stocks': 406, 'higher': 214, 'optimism': 313, 'electric': 15
2, 'vehicles': 456, 'but': 88, 'tesla': 419, 'slid': 393, 'tues': 442, '68': 39, 'tenneco': 418, '29': 24, 'visteon': 459, '28': 23, '20': 15, 'lear': 255, 'borgwarner': 81, '16': 11, 'd
ana': 126, '13': 10, 'aptiv': 63, '08': 6, '21': 18, 'sampp': 369, '500': 32, '05': 4, 'profitable': 339, 'group': 204, 'great': 203, 'opportunity': 312, 'netflix': 297, 'dumping': 145,
'its': 240, 'call': 94, 'eps': 157, 'estimate': 159, '213': 19, 'vs': 460, 'actual': 47, '174': 13, 'revenue': 361, '638': 36, '644': 37, 'surprise': 413, '18': 14, 'largest': 250, 'cha
t': 103, 'similar': 389, 'bought': 83, 'at': 67, 'out': 318, 'sent': 376, 'more': 291, 'stonks': 407, 'either': 151, 'going': 201, 'break': 84, 'market': 279, 'or': 315, 'rocket': 364,
'lol': 265, 'closed': 110, 'above': 46, 'first': 177, 'resistance': 356, 'line': 263, 'can': 96, 'get': 194, 'ugly': 447, 'other': 316, 'indicators': 233, 'showing': 385, 'it': 239, 'b
e': 73, 'fake': 171, 'anyway': 61, 'will': 478, 'interesting': 235, 'general': 193, 'motors': 293, 'stock': 405, 'soars': 396, 'ahead': 49, 'hummer': 224, 'ev': 164, 'reveal': 360, 'vi
a': 458, 'bosch': 82, 'expects': 170, 'around': 64, 'onethird': 310, 'all': 51, 'newly': 299, 'registered': 354, 'worldwide': 481, 'purely': 342, '2030': 17, 'twothirds': 446, 'new': 29
8, 'still': 403, 'powered': 329, 'combustion': 111, 'engine': 155, 'many': 278, 'them': 426, 'as': 65, 'hybrids': 225, 'waiting': 462, 'pre': 330, 'earning': 147, 'tweet': 445, 'give': 1
96, 'us': 453, 'bulls': 87, 'sign': 387, 'hope': 220, 'snapchat': 394, 'destroyed': 132, 'up': 450, 'over': 319, '17': 12, 'expected': 169, '005': 1, '001': 0, '547mm': 35, '678mm': 38,
'huge': 223, 'inc': 232, 'set': 377, 'announce': 56, 'q3': 344, '2020': 16, 'wednesday': 469, 'oct': 305, '21st': 20, 'markets': 280, 'close': 109, 'consensus': 118, 'forecast': 184, 'qu
arter': 346, '020': 3, 'reported': 355, 'same': 368, 'last': 251, 'year': 487, 'was': 465, '016': 2, 'n_lyme': 44, 'than': 421, 'anything': 60, 'me': 285, 'shows': 386, 'value': 455, 'p
latform': 324, 'so': 395, 'much': 294, 'gvoke': 206, 'nothing': 302, 'calculated': 93, 'mc': 284, '240m': 21, 'forward': 187, 'looking': 269, 'anyone': 59, 'does': 138, 'only': 311, 'cou
nt': 120, 'with': 480, 'likes': 262, 'todays': 436, 'gains': 192, '52': 34, '30': 25, 'discord': 135, 'watchlist': 466, 'today': 435, 'realtime': 350, 'our': 317, 'team': 416, 'seasone
d': 372, 'professionals': 338, 'best': 76, 'room': 365, 'global': 197, 'community': 113, 'check': 106, 'china': 107, 'buyers': 90, 'mic': 286, 'm3': 274, 'complain': 115, 're': 348, 'qua
lity': 345, 'problems': 336, 'amp': 54, 'technology': 417, 'lag': 247, 'usmade': 454, 'cut': 124, 'prices': 334, '7x': 42, 'ytd': 490, 'slashed': 392, 'warranty': 464, 'coverage': 122,
'7day': 41, 'return': 359, 'policy': 326, 'see': 373, 'how': 222, 'flies': 178, 'europe': 163, 'competes': 114, 'etc': 160, 'we': 468, 'run': 367, 'preview': 332, 'scheduled': 371, 'resu
lts': 357, 'october': 306, '060': 5, '686': 40, 'yy': 491, '829b': 43, '316': 26, '420': 28, '10': 7, 'minutes': 288, 'before': 75, 'pretty': 331, 'happy': 208, 'who': 476, 'knows': 246,
'what': 472, 'bring': 85, 'historically': 215, 'speaking': 400, 'may': 283, 'drop': 142, 'which': 474, 'case': 98, 'perhaps': 322, 'should': 384, 'have': 209, 'waited': 461, 'tuesdays':
443, 'top': 439, 'flow': 179, 'buy': 89, 'sell': 374, 'another': 57, 'winning': 479, 'options': 314, 'updates': 452, 'free': 188, 'chatroom': 104, 'dont': 140, 'forget': 186, 'take': 41
5, 'trial': 441, 'guys': 205, 'made': 275, '3k': 27, '43k': 30, 'joined': 242, 'this': 430, 'catalyst': 99, 'pops': 328, 'bubble': 86, 'sampp500': 370, 'needs': 296, 'way': 467, 'consecu
tive': 117, 'down': 141, 'days': 128, 'into': 236, 'goes': 200, 'forever': 185, 'massive': 281, 'rug': 366, 'pull': 341, 'long': 266, 'due': 143, 'chart': 102, 'sure': 412, 'look': 268,
'coming': 112, 'id': 227, 'very': 457, 'cautious': 100, 'if': 229, 'were': 471, '250': 22, 'soon': 399, 'imo': 230, 'did': 134, 'someone': 397, 'leak': 254, 'having': 210, 'shitty': 380,
'wtf': 486, 'unless': 449, 'there': 427, 'big': 78, 'reversal': 362, 'wouldnt': 485, 'want': 463, 'here': 212, 'when': 473, 'falls': 172, 'favor': 173, 'whimper': 475, 'bang': 71, 'kee
p': 244, 'noticing': 303, 'price': 333, 'sticking': 402, '42069': 29, 'significantly': 388, 'often': 308, 'efficient': 149, 'hypothesis': 226, 'would': 484, 'suggest': 409, 'thank': 422,
'let': 258, 'just': 243, 'buying': 91, 'quietly': 347, 'sit': 391, 'corner': 119, 'laughing': 252, '1000': 9, 'end': 154, 'idiots': 228, 'since': 390, 'stream': 408, 'update': 451, 'lma
o': 264, 'worse': 483, 'retards': 358, 'hang': 207, 'dumb': 144, 'fucks': 190, 'fools': 180, 'short': 381, 'red': 352, 'devils': 133, 'laughinga': 253, 'away': 69, 'bank580': 72, 'do': 1
37, 'lose': 270, 'money': 290, 'day': 127, 'no': 300, 'matter': 282, 'time': 433, 'put': 343, 'daytrading': 129, 'isnt': 238, 'everyone': 166, 'swing': 414, 'might': 287, 'better': 77,
'heres': 213, 'expect': 167, 'teslas': 420, 'told': 437, 'earlier': 146, 'stimulus': 404, 'really': 349, 'understand': 448, 'why': 477, 'know': 245, 'something': 398, 'pricing': 335, 'lo
vely': 272, 'always': 52, 'gotta': 202, 'hits': 217, 'love': 271, 'hold': 218, 'enough': 156, 'likely': 261, 'go': 199, 'they': 428, 'now': 304, 'few': 176, 'beautiful': 74, 'pop': 327,
```

Naive Bayes & SVM Performance

[19]:
```
# fit the training dataset on the NB classifier
nbModel = naive_bayes.MultinomialNB()
nbModel.fit(trainXTfidf,trainY)
# predict the labels on validation dataset
predictionsNB = nbModel.predict(testXTfidf)
# Use accuracy_score function to get the accuracy
print("Naive Bayes Accuracy Score -> ",accuracy_score(predictionsNB, testY)*100)
```

```
Naive Bayes Accuracy Score ->  51.06382978723404
```

[20]:
```
# Classifier - Algorithm - SVM
# fit the training dataset on the classifier
svmModel = svm.SVC(C=1.0, kernel='linear', gamma='auto')
svmModel.fit(trainXTfidf,trainY)
# predict the labels on validation dataset
predictionsSVM = svmModel.predict(testXTfidf)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score -> ",accuracy_score(predictionsSVM, testY)*100)
```

```
SVM Accuracy Score ->  72.3404255319149
```

Problems

The current dataset is way too skewed! (Not enough bearish tweets by virtue of the time frame we chose)

Because we couldn't scrape any more tweets (twitter blocked twint at this point), we tried using a dataset on kaggle as the training set, with the tsla tweets as the test set.

Unfortunately, the results weren't too good - which shows the importance of making sure you have good data from the beginning.

```
# fit the training dataset on the NB classifier
nbModel = naive_bayes.MultinomialNB()
nbModel.fit(trainXTfidf,trainY)
# predict the labels on validation dataset
predictionsNB = nbModel.predict(testXTfidf)
# Use accuracy_score function to get the accuracy
print("Naive Bayes Accuracy Score -> ",accuracy_score(predictionsNB, testY)*100)

# Classifier - Algorithm - SVM
# fit the training dataset on the classifier
svmModel = svm.SVC(C=1.0, kernel='linear', gamma='auto')
svmModel.fit(trainXTfidf,trainY)
# predict the labels on validation dataset
predictionsSVM = svmModel.predict(testXTfidf)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score -> ",accuracy_score(predictionsSVM, testY)*100)
```

```
Naive Bayes Accuracy Score ->  29.87012987012987
SVM Accuracy Score ->  25.97402597402597
```

Universal Language Model Fine-tuning for Text Classification (ULM-FiT)

- ULM-FiT is a transfer learning technique
- Language model pre-trained on WIKITEXT-103 corpus
- Model Fine-tuned, re-trained on part of Stock Market Sentiment Dataset
- Classifier fine tuning

BERT

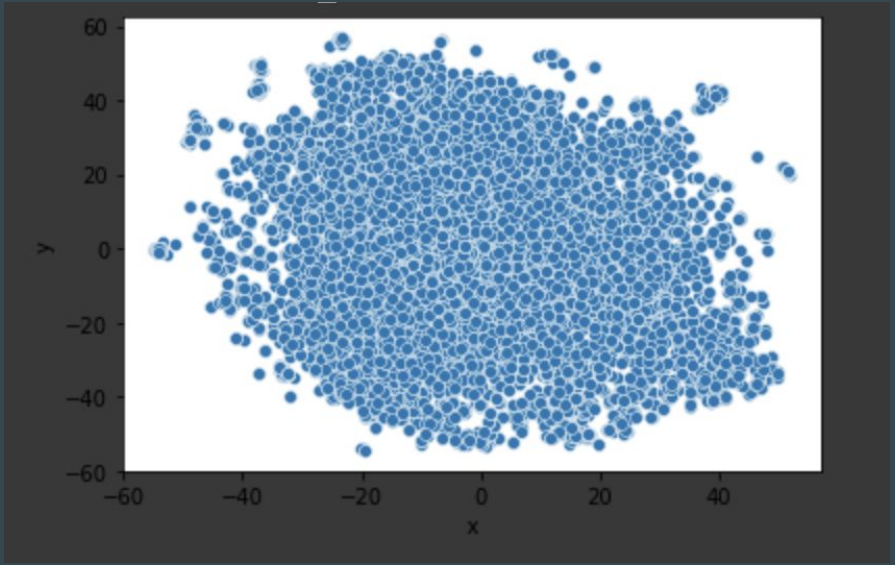
	‘Colour’	‘Animal’	‘Length’
‘Red carpet’	0.9	-0.8	-0.5
‘The cat is fat’	-0.8	1.0	-0.1

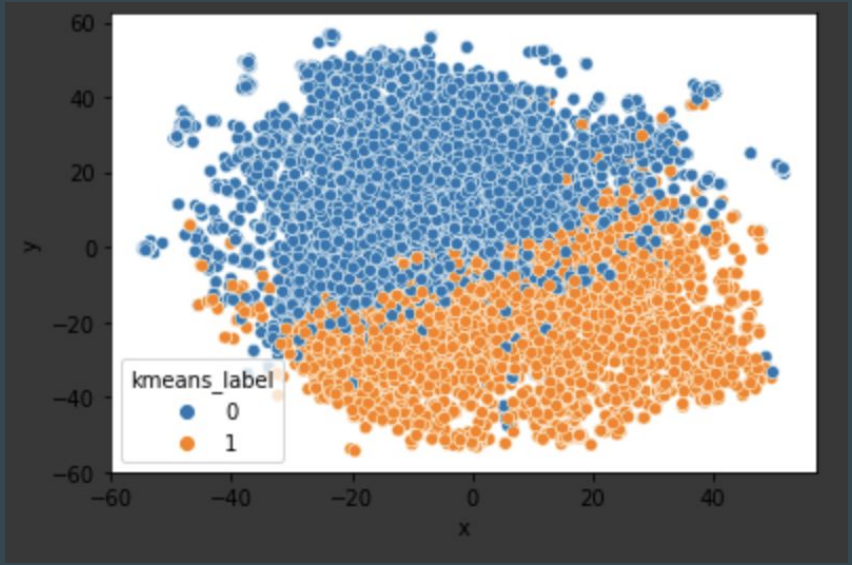
We tried to use BERT in two ways:

- Used a fine-tuned model (tuned on a labelled dataset) called FinBERT to generate labels.
- Used a small BERT model to vectorise tweets, then fit a K-means model to these vectors.

K-means

Visualising 512-dimensional vectors using TSNE





BERT

```
[ ]  #Testing accuracy of FinBERT labels using Stock_Market_Sentiment_Dataset.csv
     print(df_test_clean['Sum'].value_counts()[0], " out of ", df_test_clean.shape[0], " labels don't match")
     print("Accuracy = ", 1 - (df_test_clean['Sum'].value_counts()[0] / df_test_clean.shape[0]))
```

```
175  out of  791  labels don't match
Accuracy =  0.7787610619469026
```

- For the K-means model, we need to decide what the Kmeans labels mean, and it is not always obvious how to.
- If we picked the ‘right’ label, accuracy on Stock_Market_Sentiment_Dataset.csv (the test dataset) was 64%. If not, accuracy was 36% (we had two clusters).
- The model seems to have picked up on the skew of the dataset.

Generating signals

```
datetime
2020-11-29 23:54:59     0
2020-11-29 23:23:52     0
2020-11-29 23:23:04     1
2020-11-29 23:05:04     0
2020-11-29 23:02:01     0
                       ..
2020-10-01 04:25:42     1
2020-10-01 02:54:46     1
2020-10-01 02:46:08     0
2020-10-01 01:45:38     1
2020-10-01 01:04:24     0
```

```
df_clean = df_clean.resample('1D').sum()
```

```
df_clean['FinBERT_label']
```

```
datetime
2020-10-01     19
2020-10-02    -50
2020-10-03      2
2020-10-04     -2
2020-10-05     23
2020-10-06    -14
2020-10-07      1
2020-10-08     22
2020-10-09     32
2020-10-10      9
2020-10-11      6
2020-10-12     28
2020-10-13     12
```

Backtesting

- Testing a basic strategy on historical data on two months of S&P 500 daily price data.

```
[150] class strat1(Strategy):
        signal_strength = 5

        def init(self):
            #Create indicator
            #An indicator is just an array of values, but one that is revealed
            #gradually in Strategy.next() much like Strategy.data is.
            #It accepts a function and its arguments as arguments.
            self.signal = self.I(lambda x: x, self.data.Signal)

        def next(self):
            #If signal is more than signal_strength, close any existing positions and buy
            if self.signal > self.signal_strength:
                self.position.close()
                self.buy()

            #Elif signal is less than minus signal_strength, close any existing positions and sell
            elif self.signal < -self.signal_strength:
                self.position.close()
                self.sell()
```

Results (FinBERT)

- We didn't make any money, but at least we didn't lose much.

```
Start                     2020-10-01 00:00:00
End                       2020-11-27 00:00:00
Duration                    57 days 00:00:00
Exposure Time [%]                   95.122
Equity Final [$]                  9896.52
Equity Peak [$]                     10000
Return [%]                        -1.03483
Buy & Hold Return [%]              7.61802
Return (Ann.) [%]                 -6.19348
Volatility (Ann.) [%]              12.1878
Sharpe Ratio                             0
Sortino Ratio                            0
Calmar Ratio                             0
Max. Drawdown [%]                 -5.50738
Avg. Drawdown [%]                 -5.50738
Max. Drawdown Duration      56 days 00:00:00
Avg. Drawdown Duration      56 days 00:00:00
# Trades                                31
Win Rate [%]                       48.3871
Best Trade [%]                      2.1231
Worst Trade [%]                   -2.42404
Avg. Trade [%]                  -0.0476232
Max. Trade Duration          7 days 00:00:00
Avg. Trade Duration          2 days 00:00:00
Profit Factor                      0.922574
Expectancy [%]                  -0.0402203
SQN                              -0.186762
_strategy                            strat1
_equity_curve                            ...
_trades                     Size  EntryB...
dtype: object
```

Results (Unsupervised BERT)

- The model didn't lose money! But return is still very far off buy and hold, and still close to the risk-free rate.

```
Start                     2020-10-01 00:00:00
End                       2020-11-27 00:00:00
Duration                    57 days 00:00:00
Exposure Time [%]                   95.122
Equity Final [$]                   10187.9
Equity Peak [$]                    10251.7
Return [%]                         1.87927
Buy & Hold Return [%]              7.61802
Return (Ann.) [%]                  12.1239
Volatility (Ann.) [%]              14.8243
Sharpe Ratio                      0.817837
Sortino Ratio                      1.25442
Calmar Ratio                       1.72471
Max. Drawdown [%]                 -7.02951
Avg. Drawdown [%]                 -4.05549
Max. Drawdown Duration      46 days 00:00:00
Avg. Drawdown Duration      25 days 00:00:00
# Trades                                39
Win Rate [%]                       51.2821
Best Trade [%]                      2.1231
Worst Trade [%]                   -2.14964
Avg. Trade [%]                   0.0712696
Max. Trade Duration          3 days 00:00:00
Avg. Trade Duration          2 days 00:00:00
Profit Factor                      1.19371
Expectancy [%]                   0.0770154
SQN                                0.433122
_strategy                             strat1
_equity_curve                             ...
_trades                      Size  EntryB...
dtype: object
```

Results (ULM-FiT)

- Results are similar to to the previous models. The Sharpe for all of them is below 1, which is not great.
- We need more fine-grained price data, with more variance, and most importantly over a longer period to make any conclusions for any of the models.

Start	2020-10-01 00:00:00
End	2020-11-27 00:00:00
Duration	57 days 00:00:00
Exposure Time [%]	92.6829
Equity Final [$]	10174.9
Equity Peak [$]	10267.4
Return [%]	1.74873
Buy & Hold Return [%]	7.61802
Return (Ann.) [%]	11.2438
Volatility (Ann.) [%]	14.6357
Sharpe Ratio	0.768246
Sortino Ratio	1.33914
Calmar Ratio	2.81988
Max. Drawdown [%]	-3.98733
Avg. Drawdown [%]	-1.9574
Max. Drawdown Duration	30 days 00:00:00
Avg. Drawdown Duration	12 days 00:00:00
# Trades	34
Win Rate [%]	47.0588
Best Trade [%]	3.50888
Worst Trade [%]	-1.95262
Avg. Trade [%]	0.108908
Max. Trade Duration	4 days 00:00:00
Avg. Trade Duration	2 days 00:00:00
Profit Factor	1.27529
Expectancy [%]	0.116317
SQN	0.369037
_strategy	strat1
_equity_curve	...
_trades	Size EntryB...
dtype: object	