Assignment 2, exercise 1

In [279…

```
#Importing of packages
import nltk
from nltk import word_tokenize
import re
import pandas as pd
from collections import Counter
from nltk.corpus import stopwords
import numpy as np
import math
from wordcloud import WordCloud
english_words = set(nltk.corpus.words.words())
stopwords=stopwords.words('english')
```

In [2]:

```
#Importing the dataset
df = pd.read_csv('/Users/espensivertsen/Downloads/wallstreet_subs.csv')
```

Creating the tags

Creation and implementation of the 'textsubmission' column

In [3]:

```
#Creation and implementation of the 'textsubmission' column
x=df['title']
y=df['selftext']
x_1=[]
y_1=[]
for i in range(len(df)):
    x_1.append(x[i])
    y_1.append(y[i])
list1=[]
for i in range(len(df)):
    list1.append(x_1[i]+' '+y_1[i])

df['textsubmission']=list1
```

Finding every word starting with '$'

In [4]:

```
b=str(list(df['textsubmission']))
reg_ex=('\$\w+')

c=re.findall(reg_ex,b)
```

Removing the '$'

In [5]:

```
c = [e[1:] for e in c]
```

Removing every element containing a number

In [6]:

```
test = [x for x in c if not any(x1.isdigit() for x1 in x)]
```

Making everything uppercase

In [7]:
```
main_list=[x.upper() for x in test]
```

Finding the top 15 tags

In [318…
```
word_counter = {}
for word in main_list:
    if word in word_counter:
        word_counter[word] += 1
    else:
         word_counter[word] = 1

popular_words = sorted(word_counter, key = word_counter.get, reverse = True)
top_15 = popular_words[:15]
top_15
```

Out[318…
```
['SPY',
 'TSLA',
 'SPCE',
 'PLTR',
 'MSFT',
 'ROPE',
 'AAPL',
 'AMZN',
 'NIO',
 'ZM',
 'AMD',
 'BABA',
 'GME',
 'DIS',
 'BA']
```

Exercise 1: Tokenizing and cleaning

Making the "selftext" column a list

In [9]:
```
liste_selftext=[]
for i in range(len(df['selftext'])):
    liste_selftext.append(df['selftext'][i])
```

Removing URLS

In [10]:
```
liste_selftext_1=[]
for i in range(len(liste_selftext)):
    liste_selftext_1.append(re.sub(r'http\S+','',str(liste_selftext[i])))
```

Removing numbers

In [11]:
```
def remove(list):
    pattern = '[0-9]'
    list = [re.sub(pattern, '', i) for i in list]
    return list

liste_selftext_2 = remove(liste_selftext_1)
```

Making everything lowercase

In [12]:
```
liste_selftext_3=[x.lower() for x in liste_selftext_2]
```

Removing stop words

In [116…
```
def remove_stopwords(data):
    output_array=[]
    for sentence in data:
        temp_list=[]
        for word in sentence.split():
            if word not in stopwords:
                temp_list.append(word)
        output_array.append(' '.join(temp_list))
    return output_array

liste_selftext_4=remove_stopwords(liste_selftext_3)
```

Tokenizing the list and removing punctuation

In [14]:
```
tokenizer = nltk.RegexpTokenizer(r"\w+")
liste_token=[]
for i in range(len(df['selftext'])):
    liste_token.append(tokenizer.tokenize(str(liste_selftext_4[i])))
```

remove stopwords that were "hidden" in punctuation

In [15]:
```
liste_token2=[]
for i in range(len(liste_token)):
    test=[]
    for j in liste_token[i]:
        if j not in stopwords:
            test.append(j)
    liste_token2.append(test)
```

removing duplicates within the same sublist

In [16]:
```
list_tokens = []
for i in range(len(liste_token2)):
    a=[]
    for elem in liste_token2[i]:
        if elem not in a:
            a.append(elem)
    list_tokens.append(a)
```

Adding the tokens column to the df

In [17]:
```
df['tokens']=list_tokens
```

Exercise 2: Adding the stocks to the dataframe

Making the tags lower case so that they match the tokens (which we made all lower case)

In [118…
```
top_15_lower=[x.lower() for x in top_15]
```

Creating a list of the stocks within the tokens list (when there is no tag the list is empty)

In [19]:
```
list_stocks=[]
for i in range(len(list_tokens)):
    append_list=[]
    for j in range(len(top_15_lower)):
            if top_15_lower[j] in list_tokens[i]:
                append_list.append(top_15_lower[j])
    list_stocks.append(append_list)
```

replacing all the empty items with 'Other'

In [20]:
```
for i in range(len(list_stocks)):
    if list_stocks[i]==[]:
        list_stocks[i]= ['Other']
```

Adding the list of stocks to the dataframe

In [21]:
```
df['stocks']=list_stocks
```

Exercise 3: Creation of large documents for each stock

Creating a list for every words for each tag + 'Other'

In [117…
```
words_ba=[]
for i in range(len(list_stocks)):
    if 'ba' in list_stocks[i]:
        words_ba.append(list_tokens[i])

words_spy=[]
for i in range(len(list_stocks)):
    if 'spy' in list_stocks[i]:
        words_spy.append(list_tokens[i])

words_tsla=[]
for i in range(len(list_stocks)):
    if 'tsla' in list_stocks[i]:
          words_tsla.append(list_tokens[i])

words_spce=[]
for i in range(len(list_stocks)):
    if 'spce' in list_stocks[i]:
        words_spce.append(list_tokens[i])

words_pltr=[]
for i in range(len(list_stocks)):
    if 'pltr' in list_stocks[i]:
        words_pltr.append(list_tokens[i])

words_msft=[]
for i in range(len(list_stocks)):
    if 'msft' in list_stocks[i]:
        words_msft.append(list_tokens[i])

words_rope=[]
for i in range(len(list_stocks)):
    if 'rope' in list_stocks[i]:
        words_rope.append(list_tokens[i])

words_aapl=[]
for i in range(len(list_stocks)):
```

```
    if 'aapl' in list_stocks[i]:
        words_aapl.append(list_tokens[i])

words_amzn=[]
for i in range(len(list_stocks)):
    if 'amzn' in list_stocks[i]:
        words_amzn.append(list_tokens[i])

words_nio=[]
for i in range(len(list_stocks)):
    if 'nio' in list_stocks[i]:
        words_nio.append(list_tokens[i])

words_zm=[]
for i in range(len(list_stocks)):
    if 'zm' in list_stocks[i]:
        words_zm.append(list_tokens[i])

words_amd=[]
for i in range(len(list_stocks)):
    if 'amd' in list_stocks[i]:
        words_amd.append(list_tokens[i])

words_baba=[]
for i in range(len(list_stocks)):
    if 'baba' in list_stocks[i]:
        words_baba.append(list_tokens[i])

words_gme=[]
for i in range(len(list_stocks)):
    if 'gme' in list_stocks[i]:
        words_gme.append(list_tokens[i])

words_dis=[]
for i in range(len(list_stocks)):
    if 'dis' in list_stocks[i]:
        words_dis.append(list_tokens[i])

words_Other=[]
for i in range(len(list_stocks)):
    if 'Other' in list_stocks[i]:
        words_Other.append(list_tokens[i])
```

Removing non english words from the tokens and flattening the list for the 5 chosen tags (couldnt remove non-english words when cleaning of the tokens because tags such as 'tsla', 'gme' would get removed then)

In [25]:

```
words_gme1=[]
for i in range(len(words_gme)):
    test2=[]
    for j in words_gme[i]:
        if j in english_words:
            test2.append(j)
    words_gme1.append(test2)
words_gme_flat = [x for l in words_gme1 for x in l]

words_spy1=[]
for i in range(len(words_spy)):
    test3=[]
    for j in words_spy[i]:
        if j in english_words:
            test3.append(j)
```

```
    words_spy1.append(test3)
words_spy_flat = [x for l in words_spy1 for x in l]

words_tsla1=[]
for i in range(len(words_tsla)):
    test4=[]
    for j in words_tsla[i]:
        if j in english_words:
            test4.append(j)
    words_tsla1.append(test4)
words_tsla_flat = [x for l in words_tsla1 for x in l]

words_spce1=[]
for i in range(len(words_spce)):
    test5=[]
    for j in words_spce[i]:
        if j in english_words:
            test5.append(j)
    words_spce1.append(test5)
words_spce_flat = [x for l in words_spce1 for x in l]

words_amzn1=[]
for i in range(len(words_amzn)):
    test6=[]
    for j in words_amzn[i]:
        if j in english_words:
            test6.append(j)
    words_amzn1.append(test6)
words_amzn_flat = [x for l in words_amzn1 for x in l]
```

Exercise 4: TF and IDF

Choosing the stocks: 'gme', 'spy', 'tsla', 'spce' and 'amzn' and finding the 5 most common terms from the token list and how many times they occur for each chosen stock

In [214…

```
#gme
words_to_count_gme = (word for word in words_gme_flat)
top_gme=Counter(words_to_count_gme).most_common(len(words_gme_flat))

#spy
words_to_count_spy = (word for word in words_spy_flat)
top_spy=Counter(words_to_count_spy).most_common(len(words_spy_flat))

#tsla
words_to_count_tsla = (word for word in words_tsla_flat)
top_tsla=Counter(words_to_count_tsla).most_common(len(words_tsla_flat))

#spce
words_to_count_spce = (word for word in words_spce_flat)
top_spce=Counter(words_to_count_spce).most_common(len(words_spce_flat))

#amzn
words_to_count_amzn = (word for word in words_amzn_flat)
top_amzn=Counter(words_to_count_amzn).most_common(len(words_amzn_flat))
```

In [215…

```
print(top_gme[0:5])
print(top_spy[0:5])
print(top_tsla[0:5])
print(top_spce[0:5])
print(top_amzn[0:5])
```

```
[('buy', 409), ('short', 393), ('going', 388), ('like', 387), ('get', 378)]
[('spy', 7038), ('market', 2447), ('like', 2242), ('p', 2129), ('going', 193
9)]
[('like', 1249), ('buy', 1094), ('get', 1032), ('market', 1020), ('stock', 101
7)]
[('like', 391), ('go', 364), ('stock', 335), ('buy', 333), ('get', 331)]
[('like', 343), ('market', 299), ('going', 265), ('time', 262), ('one', 260)]
```

Describe similarities and differences between the stocks: We can see from the results that a lot of the same words go again and again such ass 'buy', 'going', 'like', 'stock', 'market' which makes since this subreddit is about the stock market. How the 'p' appeared am I not sure of, the 'p' is not in the stopwords list, maybe its an abbreviation for something like prize or something? Other than that all the result are quite alike, and expectable.

Why aren't the TFs not necessarily a good description of the stocks? TF is how many times a word appears in a text. When discussing on reddit a lot of words not necessarily used to desribe the stock may be used. Even though we've done some cleaning steps there will be words left. It's alot of talk about 'buying', 'going', the 'market' etc which will occur just because the stocks are talked about which isnt really describing the stock.

Calculating the TF (number the given word occurs / total words in the document) for the top 5 tokens for the 5 chosen stocks.

In [237…

```
#gme
tf_gme=[]
for i in range(len(top_gme)):
    tf_gme.append((top_gme[i][0], top_gme[i][1]/len(words_gme_flat)))

#spy
tf_spy=[]
for i in range(len(top_spy)):
    tf_spy.append((top_spy[i][0], top_spy[i][1]/len(words_spy_flat)))

#tsla
tf_tsla=[]
for i in range(len(top_tsla)):
    tf_tsla.append((top_tsla[i][0], top_tsla[i][1]/len(words_tsla_flat)))

#spce
tf_spce=[]
for i in range(len(top_spce)):
    tf_spce.append((top_spce[i][0], top_spce[i][1]/len(words_spce_flat)))

#amzn
tf_amzn=[]
for i in range(len(top_amzn)):
    tf_amzn.append((top_amzn[i][0], top_amzn[i][1]/len(words_amzn_flat)))
```

In [253…

```
print(tf_gme[0:5])
print(tf_spy[0:5])
print(tf_tsla[0:5])
print(tf_spce[0:5])
print(tf_amzn[0:5])
```

```
[('buy', 0.005571144468357534), ('short', 0.0053532023864657965), ('going', 0.
005285095485874629), ('like', 0.005271474105756395), ('get', 0.005148881684692
293)]
[('spy', 0.018888635418204264), ('market', 0.006567276338213389), ('like', 0.0
```

```
06017095852175897), ('p', 0.005713825633043035), ('going', 0.00520390225573999
2)]
[('like', 0.005998031070665354), ('buy', 0.005253679736835787), ('get', 0.0049
55939203303959), ('market', 0.004898312003265542), ('stock', 0.004883905203255
936)]
[('like', 0.0067212156633547635), ('go', 0.00625709079657579), ('stock', 0.005
758586310035411), ('buy', 0.005724206690274005), ('get', 0.0056898270705126)]
[('like', 0.004922573515693394), ('market', 0.0042911063591612965), ('going',
0.0038031544654774036), ('time', 0.0037600998866229424), ('one', 0.00373139683
40533014)]
```

IDF calculation

Calculation by using the formula: log(Number of documents/number of documents the word appear in). where the number of documents will be the length of the list_tokens.

Need to find number of documents the words appear in. (Since IDF for one word is the same, only needs to calculate one time each word)

In [193…

```
#buy
docs_with_buy=0
for i in range(len(list_tokens)):
    if 'buy' in list_tokens[i]:
        docs_with_buy+=1
#short
docs_with_short=0
for i in range(len(list_tokens)):
    if 'short' in list_tokens[i]:
        docs_with_short+=1
#going
docs_with_going=0
for i in range(len(list_tokens)):
    if 'going' in list_tokens[i]:
        docs_with_going+=1
#like
docs_with_like=0
for i in range(len(list_tokens)):
    if 'like' in list_tokens[i]:
        docs_with_like+=1
#get
docs_with_get=0
for i in range(len(list_tokens)):
    if 'get' in list_tokens[i]:
        docs_with_get+=1
#spy
docs_with_spy=0
for i in range(len(list_tokens)):
    if 'spy' in list_tokens[i]:
        docs_with_spy+=1
#market
docs_with_market=0
for i in range(len(list_tokens)):
    if 'market' in list_tokens[i]:
        docs_with_market+=1
#p
docs_with_p=0
for i in range(len(list_tokens)):
    if 'p' in list_tokens[i]:
        docs_with_p+=1
#stock
docs_with_stock=0
for i in range(len(list_tokens)):
    if 'stock' in list_tokens[i]:
```

```
        docs_with_stock+=1
#go
docs_with_go=0
for i in range(len(list_tokens)):
    if 'go' in list_tokens[i]:
        docs_with_go+=1
#time
docs_with_time=0
for i in range(len(list_tokens)):
    if 'time' in list_tokens[i]:
        docs_with_time+=1
#one
docs_with_one=0
for i in range(len(list_tokens)):
    if 'one' in list_tokens[i]:
        docs_with_one+=1

#know
docs_with_know=0
for i in range(len(list_tokens)):
    if 'know' in list_tokens[i]:
        docs_with_know+=1
#money
docs_with_money=0
for i in range(len(list_tokens)):
    if 'money' in list_tokens[i]:
        docs_with_money+=1
#would
docs_with_would=0
for i in range(len(list_tokens)):
    if 'would' in list_tokens[i]:
        docs_with_would+=1
#also
docs_with_also=0
for i in range(len(list_tokens)):
    if 'also' in list_tokens[i]:
        docs_with_also+=1
```

In [194…

```
#Making a list of all the numbers from above to have easier code to iterate t
list_with_numbers=[('buy', docs_with_buy), ('short', docs_with_short), ('goin
```

In [195…

```
IDF_values=[]
for i in range(len(list_with_numbers)):
    IDF_values.append((list_with_numbers[i][0],math.log(len(list_tokens)/list_

IDF_values
```

Out[195…

```
[('buy', 1.7092449465395803),
 ('short', 2.516102451794195),
 ('going', 1.6616262711589307),
 ('like', 1.8765250164535898),
 ('get', 1.6292332442144688),
 ('spy', 2.458342081577906),
 ('market', 1.632774170417683),
 ('p', 2.5026453109141555),
 ('stock', 1.7644847241540806),
 ('go', 1.738141600724539),
 ('time', 1.7769143434253172),
 ('one', 1.8765250164535898),
 ('know', 1.7780650915384952),
 ('money', 1.7356554115986131),
```

```
 ('would', 1.9368320519235682),
 ('also', 2.129326636430573)]
```

Exercise 5: TF-IDF

Top 10 tf for the 5 stocks

In [254…

```
print(tf_gme[0:10])
print(tf_spy[0:10])
print(tf_tsla[0:10])
print(tf_spce[0:10])
print(tf_amzn[0:10])
```

```
[('buy', 0.005571144468357534), ('short', 0.0053532023864657965), ('going', 0.
005285095485874629), ('like', 0.005271474105756395), ('get', 0.005148881684692
293), ('know', 0.00479472580161822), ('stock', 0.00464489062031765), ('time',
0.004440569918544147), ('go', 0.004345220257716512), ('one', 0.004249870596888
877)]
[('spy', 0.018888635418204264), ('market', 0.006567276338213389), ('like', 0.0
06017095852175897), ('p', 0.005713825633043035), ('going', 0.00520390225573999
2), ('get', 0.005123388038271092), ('go', 0.005104601387528347), ('money', 0.0
04887213000362314), ('time', 0.00481206639739134), ('buy', 0.00465640557695146
3)]
[('like', 0.005998031070665354), ('buy', 0.005253679736835787), ('get', 0.0049
55939203303959), ('market', 0.004898312003265542), ('stock', 0.004883905203255
936), ('going', 0.004715825869810551), ('go', 0.0045621533363747685), ('mone
y', 0.00453333973635556), ('one', 0.004514130669676087), ('time', 0.0044853170
69656878)]
[('like', 0.0067212156633547635), ('go', 0.00625709079657579), ('stock', 0.005
758586310035411), ('buy', 0.005724206690274005), ('get', 0.0056898270705126),
('going', 0.0056210678309897896), ('money', 0.005242892013614329), ('market',
0.004761577336954653), ('time', 0.0047443875270739505), ('one', 0.004349021899
817788)]
[('like', 0.004922573515693394), ('market', 0.0042911063591612965), ('going',
0.0038031544654774036), ('time', 0.0037600998866229424), ('one', 0.00373139683
40533014), ('stock', 0.0036165846237747384), ('would', 0.003530475466065816),
('get', 0.0035017724134961755), ('also', 0.003487420887211355), ('buy', 0.0034
300147820720732)]
```

In [271…

```
#The function makes it able to sort by the 2nd element, the tf and not
#alphabetically by words
def takeSecond(elem):
    return elem[1]
```

Top 10 TF-IDF for $GME. Calculated by TF*IDF for the given word.

In [292…

```
TF_IDF_gme=[]
for i in range(len(tf_gme[0:10])):
    for j in range(len(IDF_values)):
        if tf_gme[i][0]==IDF_values[j][0]:
            TF_IDF_gme.append((tf_gme[i][0], tf_gme[i][1]*IDF_values[j][1]))

TF_IDF_gme.sort(key=takeSecond, reverse=True)
TF_IDF_gme
```

Out[292…

```
[('short', 0.013469205649537127),
 ('like', 0.009892053033039193),
 ('buy', 0.009522450528982053),
 ('going', 0.008781853504912757),
 ('know', 0.008525334571356286),
 ('get', 0.008388729211227686),
```

```
 ('stock', 0.008195838544917065),
 ('one', 0.007974988491752528),
 ('time', 0.007890512381244087),
 ('go', 0.007552608094248071)]
```

Top 10 TF-IDF for $SPY

In [293…

```
TF_IDF_spy=[]
for i in range(len(tf_spy[0:10])):
    for j in range(len(IDF_values)):
        if tf_spy[i][0]==IDF_values[j][0]:
            TF_IDF_spy.append((tf_spy[i][0], tf_spy[i][1]*IDF_values[j][1]))

TF_IDF_spy.sort(key=takeSecond, reverse=True)
TF_IDF_spy
```

Out[293…

```
[('spy', 0.04643472731215443),
 ('p', 0.01429967892791626),
 ('like', 0.011291230893007202),
 ('market', 0.010722879175030045),
 ('go', 0.008872520026779225),
 ('going', 0.008646940700680792),
 ('time', 0.008550629803039663),
 ('money', 0.008482517691713945),
 ('get', 0.008347194114962014),
 ('buy', 0.007958937701443007)]
```

Top 10 TF-IDF for $TSLA

In [294…

```
TF_IDF_tsla=[]
for i in range(len(tf_tsla[0:10])):
    for j in range(len(IDF_values)):
        if tf_tsla[i][0]==IDF_values[j][0]:
            TF_IDF_tsla.append((tf_tsla[i][0], tf_tsla[i][1]*IDF_values[j][1]

TF_IDF_tsla.sort(key=takeSecond, reverse=True)
TF_IDF_tsla
```

Out[294…

```
[('like', 0.011255455353569446),
 ('buy', 0.00897982554092396),
 ('stock', 0.00861757612536173),
 ('one', 0.008470879129187574),
 ('get', 0.008074380906328579),
 ('market', 0.007997837317578874),
 ('time', 0.00797002423588372),
 ('go', 0.007929668502837236),
 ('money', 0.007868315646020557),
 ('going', 0.007835940155488127)]
```

Top 10 TF-IDF for $SPCE

In [302…

```
TF_IDF_spce=[]
for i in range(len(tf_spce[0:10])):
    for j in range(len(IDF_values)):
        if tf_spce[i][0]==IDF_values[j][0]:
            TF_IDF_spce.append((tf_spce[i][0], tf_spce[i][1]*IDF_values[j][1]

TF_IDF_spce.sort(key=takeSecond, reverse=True)
TF_IDF_spce
```

Out[302…

```
[('like', 0.012612529333264923),
 ('go', 0.010875709813039024),
 ('stock', 0.010160937576780297),
```

```
 ('buy', 0.0097840713582989),
 ('going', 0.009340113979938983),
 ('get', 0.00927005541711055),
 ('money', 0.00909985389585686),
 ('time', 0.008430370247625872),
 ('one', 0.008161048392112596),
 ('market', 0.007774580486225774)]
```

Top 10 TF-IDF for $AMZN

In [295…

```
TF_IDF_amzn=[]
for i in range(len(tf_amzn[0:10])):
    for j in range(len(IDF_values)):
        if tf_amzn[i][0]==IDF_values[j][0]:
            TF_IDF_amzn.append((tf_amzn[i][0], tf_amzn[i][1]*IDF_values[j][1]

TF_IDF_amzn.sort(key=takeSecond, reverse=True)
TF_IDF_amzn
```

Out[295…

```
[('like', 0.009237332347530552),
 ('also', 0.007425858187583479),
 ('market', 0.0070064076257536296),
 ('one', 0.007002059505416745),
 ('would', 0.006837938041206071),
 ('time', 0.006681375421252215),
 ('stock', 0.006381408322261059),
 ('going', 0.006319421373112654),
 ('buy', 0.005862735432812751),
 ('get', 0.005705204029741104)]
```

Since I only have the IDF for the topwords the words doesnt change, but the order of them change. This makes sense as now they are dependant on other documents aswell.

(was told to only find the IDF for the top words of each stock)

Exercise 6: Word cloud

Creation of the wordcloud. The wordclouds can be found in a seperate folder in the assignment. I did one wordcloud based on TF-IDF and one based of TF.

In [281…

```
wc= WordCloud(
    background_color='white',
    height=600,
    width=400
)
```

GME:

In [298…

```
wc.generate(str(TF_IDF_gme))
wc.to_file('GME.png')
```

Out[298…

```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

In [313…

```
wc.generate(str(top_gme))
wc.to_file('GME_tf.png')
```

Out[313…

```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

SPY:

In [299…
```
wc.generate(str(TF_IDF_spy))
wc.to_file('SPY.png')
```

Out[299…
```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

In [314…
```
wc.generate(str(top_spy))
wc.to_file('SPY_tf.png')
```

Out[314…
```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

TSLA:

In [300…
```
wc.generate(str(TF_IDF_tsla))
wc.to_file('TSLA.png')
```

Out[300…
```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

In [315…
```
wc.generate(str(top_tsla))
wc.to_file('TSLA_tf.png')
```

Out[315…
```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

SPCE:

In [303…
```
wc.generate(str(TF_IDF_spce))
wc.to_file('spce.png')
```

Out[303…
```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

In [316…
```
wc.generate(str(top_spce))
wc.to_file('SPCE_tf.png')
```

Out[316…
```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

AMZN:

In [304…
```
wc.generate(str(TF_IDF_amzn))
wc.to_file('AMZN.png')
```

Out[304…
```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

In [317…
```
wc.generate(str(top_amzn))
wc.to_file('AMZN_tf.png')
```

Out[317…
```
<wordcloud.wordcloud.WordCloud at 0x7f90b72b8340>
```

Lastly I've just added the dataframe to show that the textsubmission, tokens and stocks columns has been added

In [319…
```
df
```

Out[319…

	created_utc	title	selftext	score	textsubmission	tokens
0	1586173811	What is the Fed actually buying?	Okay, I may actually just be retarded. On my d...	1	What is the Fed actually buying? Okay, I may a...	[okay, may, actually, retarded, defense, every...
1	1586173320	I didn't learn about puts because I was lazy	Beginning of the this virus shit, everyone was...	1	I didn't learn about puts because I was lazy B...	[beginning, virus, shit, everyone, talking, pu...
2	1586173268	HOT TAKE	Literally everyone has free time on their hand...	1	HOT TAKE Literally everyone has free time on t...	[literally, everyone, free, time, hands, mean,...
3	1586172639	Fuck you Gordon	Gordon I believed in you, I can't even begin t...	1	Fuck you Gordon Gordon I believed in you, I ca...	[gordon, believed, even, begin, describe, disa...
4	1586171822	Can't find a picture	Someone uploaded a ohoto of the stock market h...	1	Can't find a picture Someone uploaded a ohoto ...	[someone, uploaded, ohoto, stock, market, hist...
...	...	...	...	...	...	...
82237	1602007302	Hurricane Delta (BECN) 🔥	\nHurricane Delta is looking like it is going ...	1	Hurricane Delta (BECN) 🔥 \nHurricane Delta is ...	[hurricane, delta, looking, like, going, hit, ...
82238	1602006818	Made 40k on Nike. Next play? CROCS motherfucker	# 1. Introduction\n\n[Proof that I'm lucky](h...	1	Made 40k on Nike. Next play? CROCS motherfucke...	[introduction, proof, lucky, way, going, struc...
82239	1602006029	Please screenshot the whole timeline, not just...	I could nut over your retarded failures just f...	1	Please screenshot the whole timeline, not just...	[could, nut, retarded, failures, fine, really,...
82240	1602005968	What is your price target for Tesla in 40 years?	I am 26 and currently max out my roth each yea...	1	What is your price target for Tesla in 40 year...	[currently, max, roth, year, robinhood, specul...
82241	1601822856	White House infected folks will infect Senators	They all gonna get the rona, \n\nThis means no...	1	White House infected folks will infect Senator...	[gonna, get, rona, means, checkies, stimulus, ...

82242 rows × 7 columns