NLP and Sentiment Analysis

Using data from CrowdFlower, I'm going to perform different NLP models and sentiment analysis on tweets talking about Apple and Google.

Should I make this two datasets? Begin by separating tweets into pandas dataframes, one for google, one of apple. The business understanding could be a 3rd party service acting to relay the sentiment of each companies products based on tweets. I will give advertising and marketing recommendations to each of the companies.

Reminder - This is a classification task

Preliminary thoughts on process -

1. pre-preperation - vectorize all tweets at start
2. Classify all tweets as positive, negative, neutral
3. exploration - Begin exploring products that are discussed positive or negatively and check word associativity to determine the reasons for those sentiments.
4. Give advertising recommendations - Advertising recommendations: Marketing research shows we should advertise to people who like our products. If twitter is generally positive, suggest more money spent on advertising, if twitter is generally negative, perhaps less money should be spent. Or considering advertising that highlights the best aspects of the discussed products, and counters/discusses the improvements of negatively talked about products.
5. Give product development recommendations - Overwhelmingly negatively talked about products should be improved based on user feedback, determine most discussed feedback

Blog Post recapping this project and my thoughts:
https://exumexaminesdata.blogspot.com/2023/03/nlp-and-sentiment-analysis-of-tech.html

Summary

For this analysis I used F1 score as my evaluation metric, this score takes both false positives and false negatives into account. It is suitable for uneven class distribution problems.

I began by investigating the data. From the link provided for downloading the data we know it comes from around 2011 and 2013. Unfortunately the data only has 9000 tweets and the class imbalance is very prevalent with 60% of the tweets being "No emotion" which is a hinderance for training. A large amount of tweets were also considered to not be aimed at any particular brand or company, which I found to be untrue. I didn't want to mess up anything in the data and decided to leave it as is instead of imputing what tweets were aimed at different brands based on words used in the tweet.

For feature generation I tried a bunch of different things, including number of sentences, if a tweet contains an emoji, and vectorizers. The first batch of models I tried included multinomialNB and an SVC. The SVC had better accuracy and F1, but had a generally poor F1 score of only 58%. I then tried using logistic regression, random forest, and XGBoost. These models showed increased performance. I then went on to create a perceptron neural net using Keras. This model performed the best, with a 74% accuracy on the training set and 66% on the validation set. I created a new random validation set of 1800 rows and the neural net obtained an F1 score of 75%. Before declaring this the best model I decided to try a Word2Vec approach using the Random Forest and XGB because they performed well previously. The random forest and XGB had accuracy's of 67% and 68% respectively, but had F1 scores of 95% and 93%. The validation set was created again using 1800 randomly sampled rows from the training set.

The accuracy and F1 of the Neural Net is incredibly impressive and makes me think there was no overfitting. The F1 score is almost too good to be true when talking about the random forest and XGB models.

Resources

These notebooks and sites were a useful tools and references in completing this analysis

1. https://www.kaggle.com/code/tanulsingh077/twitter-sentiment-extaction-analysis-eda-and-model
2. https://www.kaggle.com/code/nitin194/twitter-sentiment-analysis-word2vec-doc2vec
3. https://www.kaggle.com/code/prakharrathi25/sentiment-analysis-using-bert
4. https://github.com/learn-co-curriculum/dsc-classification-with-word-embeddings-codealong

In [1]:

```
# imports

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from matplotlib.ticker import MaxNLocator
import seaborn as sns

# NLTK
from nltk.tokenize import RegexpTokenizer
from nltk import FreqDist
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
from nltk.corpus import stopwords

from nltk.stem.snowball import SnowballStemmer
from nltk.tokenize import sent_tokenize

#Sklearn
from sklearn.metrics import plot_confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
```

```python
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix

# warnings
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

In [2]:
```python
stopwords_list = stopwords.words('english')
```

In [3]:
```python
# create df with data and inspect
df = pd.read_csv('Data/data.csv', engine='python')
df.head()
```

Out[3]:

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_pro
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative em
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive em
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive em
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative em
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive em

In [4]:
```python
# Change column names for ease of use
df.columns = ['tweet', 'subject', 'emotion']
```

In [5]:
```python
# Check emotion distribution
df['emotion'].value_counts()
```

Out[5]:
```
No emotion toward brand or product    5389
Positive emotion                      2978
Negative emotion                       570
I can't tell                           156
Name: emotion, dtype: int64
```

In [6]:
```
# Store any unknown emotion values elsewhere and then drop them from the main DF
# We can potentially use this later as a production version of the learner to gi

unknowns_df = df.loc[df['emotion'] == "I can't tell"]

df = df.drop(unknowns_df.index.tolist())
```

In [7]:
```
# Check NaNs
df.isna().sum()
```

Out[7]:
```
tweet          1
subject     5655
emotion        0
dtype: int64
```

The number NaN's in the subject column is a bit alarming. I can't really do any simple Imputing to reduce that number, and I can drop over half my dataset. Either have to live with it and move on, or iterate through the df and see if the creators of the data missed things. Could perhaps just use an If/Then to check is if a group of apple or apple related words is in the tweet and assign that tweet to apple, same with google.

In [8]:
```
#Drop NaN values only if tweet is NaN
df = df.dropna(subset=['tweet'])
df.isna().sum()
```

Out[8]:
```
tweet          0
subject     5654
emotion        0
dtype: int64
```

In [9]:
```
df.head()
```

Out[9]:

	tweet	subject	emotion
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion

In [10]:
```
# Lower case all tweets
df['tweet'] = df['tweet'].str.lower()
```

In [11]:
```
df.subject.value_counts()
```

Out[11]:
```
iPad                               942
Apple                              659
iPad or iPhone App                 470
Google                             429
iPhone                             296
Other Google product or service    292
Android App                         81
Android                             78
Other Apple product or service      35
Name: subject, dtype: int64
```

In [12]:

```
#investigate other google product tweets
google_rows = df[df['subject'] == 'Other Google product or service']
google_rows
```

Out[12]:

	tweet	subject	emotion
13	gotta love this #sxsw google calendar featurin...	Other Google product or service	Positive emotion
27	someone started an #austin @partnerhub group i...	Other Google product or service	Positive emotion
163	just left #sxsw tradeshow demo of @mention at ...	Other Google product or service	Positive emotion
198	sweet new 3-d google maps demo going on in bal...	Other Google product or service	Positive emotion
199	more than 35 million miles per day are driving...	Other Google product or service	Positive emotion
...	...	...	...
8989	it's crazy how much culture is documented in g...	Other Google product or service	Positive emotion
8992	looks very interesting rt@mention google to la...	Other Google product or service	Positive emotion
9006	creativity prompt: use google maps to virtuall...	Other Google product or service	Positive emotion
9025	absolutely! rt @mention timely good schtuff f...	Other Google product or service	Positive emotion
9080	diller says google tv "might be run over ...	Other Google product or service	Negative emotion

292 rows × 3 columns

I think for now, its ok to just impute the missing values based on Apple, iPhone, iPad, Google, or Andriod, and not worry too much about the related products until later. Later I can use word association to find tweets with 'app' in them as well to further expand subject

In [13]:

```
# # Impute NaN values in subject because things were missed

# apple words
df.loc[df['tweet'].str.contains('apple', case=False) & df['subject'].isna(), 'su
df.loc[df['tweet'].str.contains('iphone', case=False) & df['subject'].isna(), 's
df.loc[df['tweet'].str.contains('ipad', case=False) & df['subject'].isna(), 'sub
df.loc[df['tweet'].str.contains('itunes', case=False) & df['subject'].isna(), 's

# google
df.loc[df['tweet'].str.contains('google', case=False) & df['subject'].isna(), 's

# andriod
df.loc[df['tweet'].str.contains('andriod', case=False) & df['subject'].isna(), '
```

In [14]:

```
#check NaN again
df.isna().sum()
```

```
Out[14]: tweet         0
         subject     907
         emotion       0
         dtype: int64
```

That was a massive massive success. Lets move on to tokenizing and removing stopwords

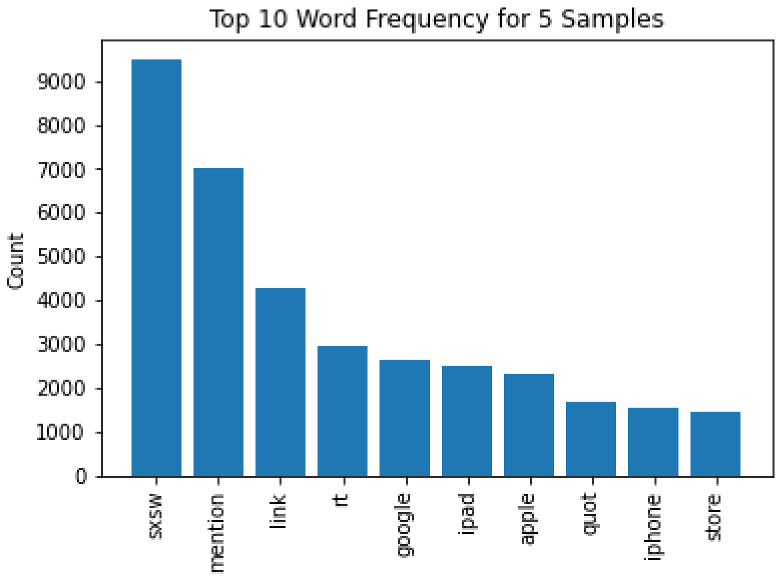
```
In [15]: # Tokenize Tweets

         basic_token_pattern = r"(?u)\b\w\w+\b"

         tokenizer = RegexpTokenizer(basic_token_pattern)

         df["tweet_tokenized"] = df["tweet"].apply(tokenizer.tokenize)

         # Display full text
         #df.style.set_properties(**{'text-align': 'left'})
```

```
In [16]: #remove stopwords

         # this helper funtion returns a list with any stopwords in the original list rem
         def remove_stopwords(token_list):
             """
             Given a list of tokens, return a list where the tokens
             that are also present in stopwords_list have been
             removed
             """
             return [w for w in token_list if w not in stopwords_list]

         df["tweet_without_stopwords"] = df["tweet_tokenized"].apply(remove_stopwords)
```

I think now would be a good time to create frequency distributions for any Apple related tweets, google related tweets, and then again with positive, negative, and no emotion tweets

```
In [17]: def visualize_top_10(freq_dist, title):

             # Extract data for plotting
             top_10 = list(zip(*freq_dist.most_common(10)))
             tokens = top_10[0]
             counts = top_10[1]

             # Set up plot and plot data
             fig, ax = plt.subplots()
             ax.bar(tokens, counts)

             # Customize plot appearance
             ax.set_title(title)
             ax.set_ylabel("Count")
             ax.yaxis.set_major_locator(MaxNLocator(integer=True))
             ax.tick_params(axis="x", rotation=90)
```

```
In [18]: entire_df__freq_dist = FreqDist(df["tweet_without_stopwords"].explode())
         visualize_top_10(entire_df__freq_dist, "Top 10 Word Frequency")
```



It might be worth removing sxsw, mention, link, and rt as well. They don't give me any information.

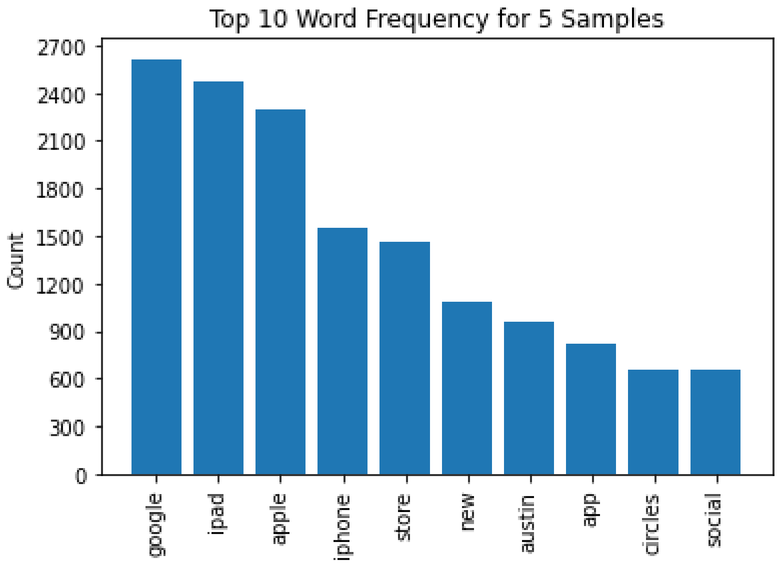
1. #sxsw is the tech event most of these tweets are talking about
2. mention refers quoting someone
3. link refers to links in tweets
4. rt refers to a retweet
5. quot refers to a quote retweet
6. amp refers to an ampersand

In [19]:

```
stopwords_list.append('sxsw')
stopwords_list.append('mention')
stopwords_list.append('link')
stopwords_list.append('rt')
stopwords_list.append('quot')
stopwords_list.append('amp')
df["tweet_without_stopwords"] = df["tweet_tokenized"].apply(remove_stopwords)
```

In [20]:

```
entire_df__freq_dist = FreqDist(df["tweet_without_stopwords"].explode())
visualize_top_10(entire_df__freq_dist, "Top 10 Word Frequency")
```



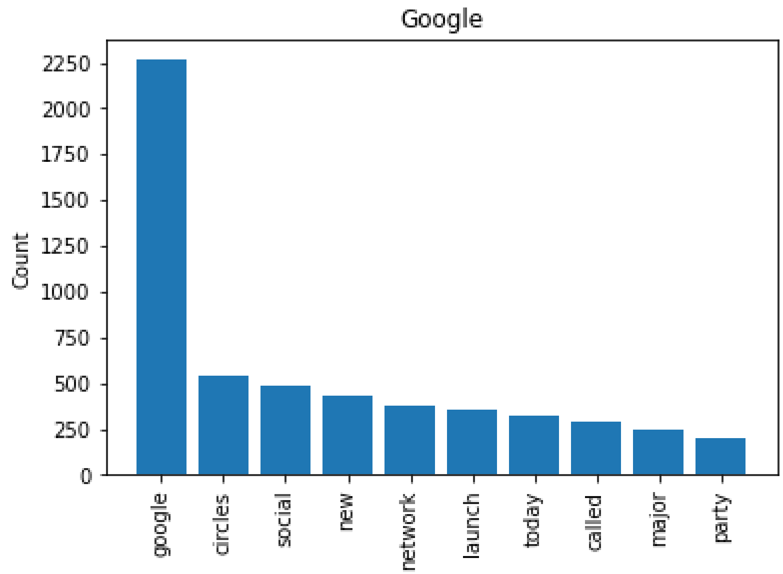
In [21]:
```
df.subject.value_counts()
```

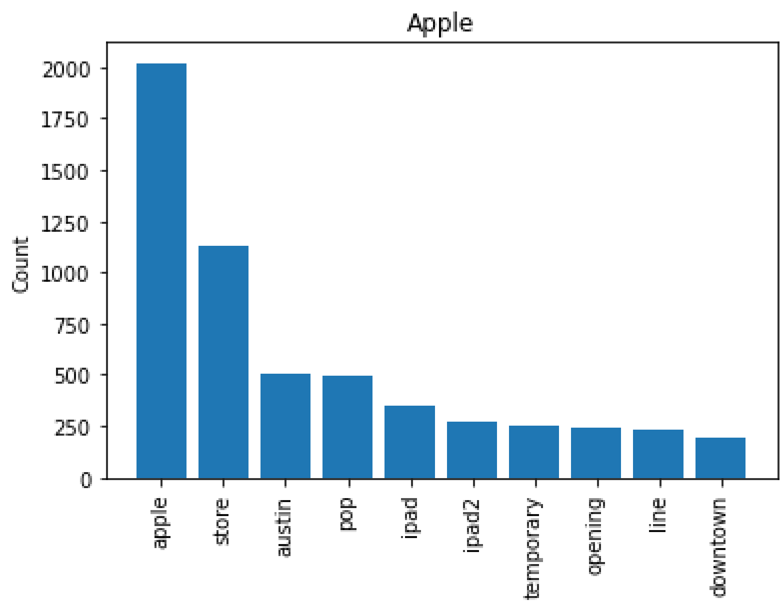
Out[21]:
```
Google                             2093
iPad                               1916
Apple                              1841
iPhone                             1166
iPad or iPhone App                  527
Other Google product or service     292
Android App                          81
Android                              78
Other Apple product or service       35
Name: subject, dtype: int64
```

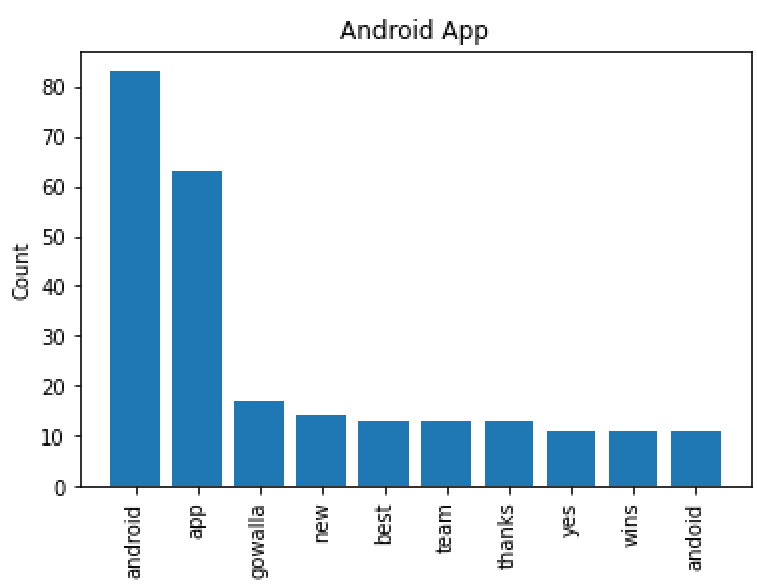
In [22]:
```
# Function that iterates through the different Subjects and plots their distribu

def freq_distribution_by_subject(df):
    #subjects = ['Google', 'iPad', 'Apple', 'iPhone', 'iPad or iPhone App']
    subjects = set(df['subject'].values.tolist())
    subjects.pop()

    #in the loop, create a freq dist for only a subject, then call the visualize

    for sub in subjects:
        print(sub)
        subject_df = df[df['subject'] == sub]
        subject_df_freq_dist = FreqDist(subject_df["tweet_without_stopwords"].ex
        visualize_top_10(subject_df_freq_dist, sub)
```

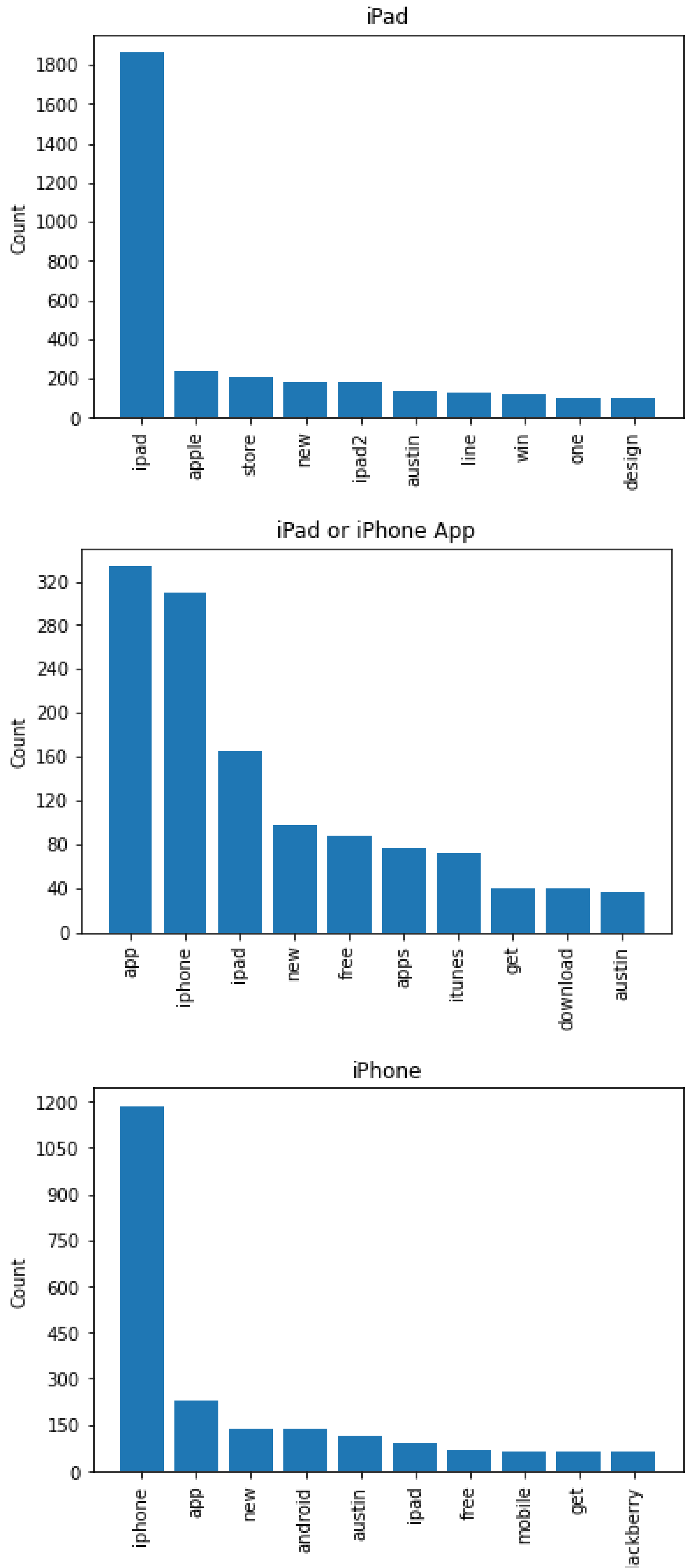
In [23]:
```
freq_distribution_by_subject(df)
```

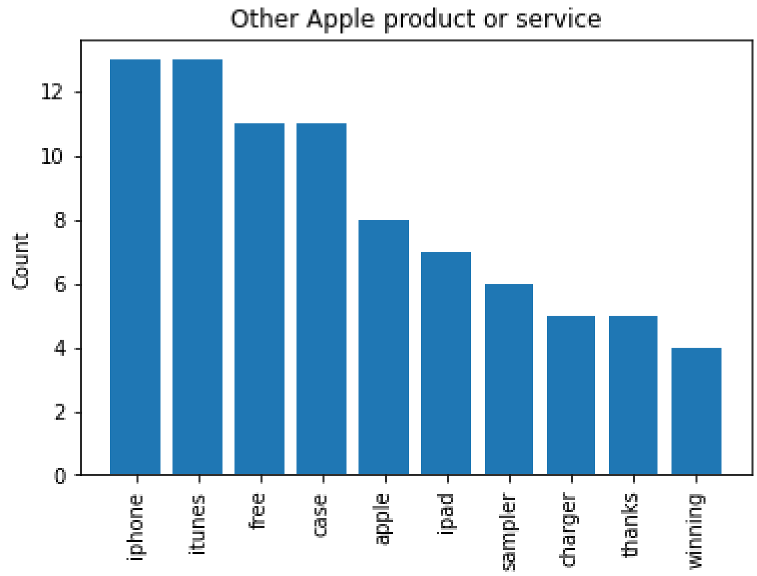
```
Google
Apple
Android App
iPad
iPad or iPhone App
iPhone
Other Apple product or service
Android
Other Google product or service
```

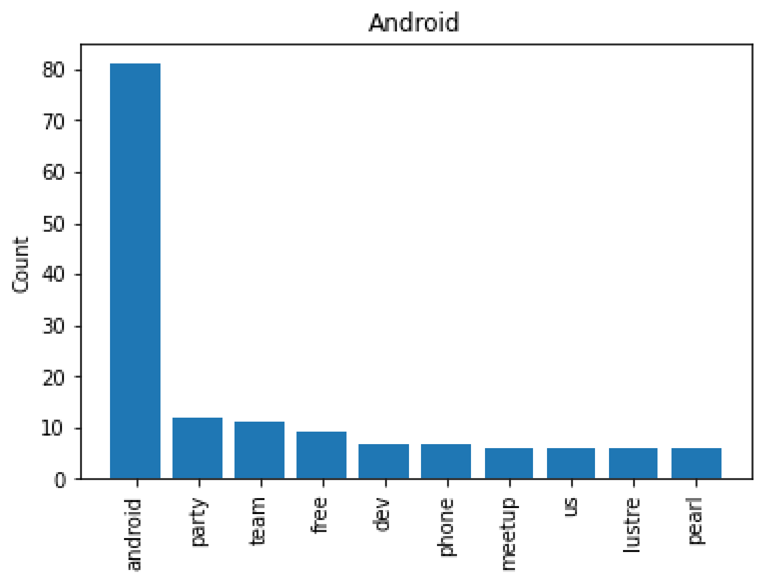

Google
Count
0
250
500
750
1000
1250
1500
1750
2000
2250
google
circles
social
new
network
launch
today
called
major
party

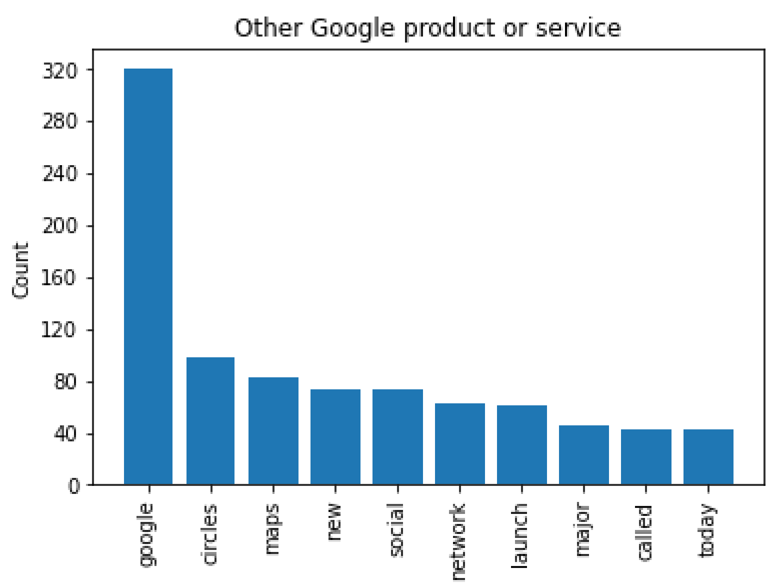

Apple
Count
0
250
500
750
1000
1250
1500
1750
2000
apple
store
austin
pop
ipad
ipad2
temporary
opening
line
downtown


Android App
Count
0
10
20
30
40
50
60
70
80
android
app
gowalla
new
best
team
thanks
yes
wins
andoid


iPad
Count
0
200
400
600
800
1000
1200
1400
1600
1800
ipad
apple
store
new
ipad2
austin
line
win
one
design
iPad or iPhone App
Count
0
40
80
120
160
200
240
280
320
app
iphone
ipad
new
free
apps
itunes
get
download
austin
iPhone
Count
0
150
300
450
600
750
900
1050
1200
iphone
app
new
android
austin
ipad
free
mobile
get
blackberry







Most of these are probably not needed as the max word count is quite low, we will keep them for now.

Modeling

I'm going to start by using MultinomialNB and potentially iterating to a deep learning neural net to determine sentimet

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

Out[24]:

	tweet	subject	emotion	tweet_tokenized	tweet_without_stopwords
0	.@wesley83 i have a 3g iphone. after 3 hrs twe...	iPhone	Negative emotion	[wesley83, have, 3g, iphone, after, hrs, tweet...	[wesley83, 3g, iphone, hrs, tweeting, rise_aus...
1	@jessedee know about @fludapp ? awesome ipad/i...	iPad or iPhone App	Positive emotion	[jessedee, know, about, fludapp, awesome, ipad...	[jessedee, know, fludapp, awesome, ipad, iphon...
2	@swonderlin can not wait for #ipad 2 also. the...	iPad	Positive emotion	[swonderlin, can, not, wait, for, ipad, also, ...	[swonderlin, wait, ipad, also, sale]
3	@sxsw i hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion	[sxsw, hope, this, year, festival, isn, as, cr...	[hope, year, festival, crashy, year, iphone, app]
4	@sxtxstate great stuff on fri #sxsw: marissa m...	Google	Positive emotion	[sxtxstate, great, stuff, on, fri, sxsw, maris...	[sxtxstate, great, stuff, fri, marissa, mayer,...

In [25]:
```
# Start by setting up our training and test sets

y = df['emotion']
X = df['tweet']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
```

In [26]:
```
X_train.head()
```

Out[26]:

	tweet
3363	having a great time at the google party #sxswi...
3204	@mention from:ubersocial for iphone now in the...
4460	are you in town for #sxsw? be sure to check in...
2311	the ironic tee has been usurped by the ipad 2 ...
6298	rt @mention marissa mayer: google will connect...

In [27]:
```
# Check distibution of training set
y_train.value_counts()
```

Out[27]:
```
emotion
No emotion toward brand or product     4032
```

```
Positive emotion                      2259
Negative emotion                       411
dtype: int64
```

In [28]:
```
# Check distibution of test set
y_test.value_counts()
```

Out[28]:
```
emotion
No emotion toward brand or product    1356
Positive emotion                       719
Negative emotion                       159
dtype: int64
```

In [29]:
```
#Create Baseline Model, create a TFidVectorizer

tfidf = TfidfVectorizer(max_features=10)

# Fit the vectorizer on X_train["text"] and transform it
X_train_vectorized = tfidf.fit_transform(X_train['tweet'])

# Visually inspect the 10 most common words
pd.DataFrame.sparse.from_spmatrix(X_train_vectorized, columns=tfidf.get_feature_
```

Out[29]:

	at	for	google	ipad	link	mention	rt	sxsw	tl
0	0.561447	0.000000	0.598490	0.000000	0.000000	0.000000	0.000000	0.260088	0.5088(
1	0.000000	0.523624	0.000000	0.000000	0.388421	0.355803	0.000000	0.219420	0.4292(
2	0.000000	0.823886	0.397220	0.000000	0.000000	0.000000	0.000000	0.172621	0.0000(
3	0.320355	0.000000	0.000000	0.341491	0.000000	0.000000	0.000000	0.148403	0.8710!
4	0.000000	0.000000	0.503358	0.000000	0.387227	0.354709	0.489748	0.218746	0.4279
...	...	...	...	...	...	...	...	...	...
6697	0.434081	0.000000	0.462720	0.000000	0.355965	0.326072	0.450209	0.402171	0.0000(
6698	0.468943	0.000000	0.000000	0.000000	0.000000	0.704520	0.486367	0.217235	0.0000(
6699	0.383120	0.000000	0.000000	0.000000	0.314175	0.287792	0.397356	0.177479	0.6944
6700	0.329455	0.000000	0.000000	0.351191	0.000000	0.742439	0.000000	0.152618	0.2985(
6701	0.000000	0.761199	0.000000	0.000000	0.564652	0.000000	0.000000	0.318974	0.0000(

6702 rows × 10 columns

In [30]:
```
baseline_model = MultinomialNB()

# Evaluate the classifier on X_train_vectorized and y_train
baseline_cv = cross_val_score(baseline_model, X_train_vectorized, y_train['emoti
baseline_cv
```

Out[30]:
```
array([0.60178971, 0.601044  , 0.60149254, 0.60149254, 0.60223881])
```

In [31]:
```
y_train.value_counts(normalize=True)
```

Out[31]:
```
emotion
No emotion toward brand or product    0.601611
Positive emotion                      0.337064
```

```
Negative emotion                       0.061325
dtype: float64
```

Ok, so the baseline model is 60% accurate, but based on the distibution, if it only guess no emotion, thats what it would score, lets see if we can raise that value.

Start by Removing Stopwords

In [32]:

```
#Removing Stopwords

tfidf = TfidfVectorizer(
    max_features=10,
    stop_words=stopwords_list
)

# Fit the vectorizer on X_train["text"] and transform it
X_train_vectorized = tfidf.fit_transform(X_train['tweet'])

# Visually inspect the vectorized data
pd.DataFrame.sparse.from_spmatrix(X_train_vectorized, columns=tfidf.get_feature_
```

Out[32]:

	app	apple	austin	circles	google	ipad	iphone	new	social	store
0	0.000000	0.00000	0.000000	0.0	1.0	0.0	0.00000	0.0	0.0	0.000000
1	0.656654	0.00000	0.000000	0.0	0.0	0.0	0.52949	0.0	0.0	0.537072
2	0.000000	0.00000	0.000000	0.0	1.0	0.0	0.00000	0.0	0.0	0.000000
3	0.000000	0.00000	0.000000	0.0	0.0	1.0	0.00000	0.0	0.0	0.000000
4	0.000000	0.00000	0.000000	0.0	1.0	0.0	0.00000	0.0	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...
6697	0.000000	0.00000	0.000000	0.0	1.0	0.0	0.00000	0.0	0.0	0.000000
6698	0.000000	0.00000	0.000000	0.0	0.0	0.0	1.00000	0.0	0.0	0.000000
6699	0.000000	0.00000	0.000000	0.0	0.0	0.0	0.00000	0.0	0.0	0.000000
6700	0.000000	0.00000	0.000000	0.0	0.0	1.0	0.00000	0.0	0.0	0.000000
6701	0.000000	0.75127	0.500424	0.0	0.0	0.0	0.00000	0.0	0.0	0.430313

6702 rows × 10 columns

In [33]:

```
stopwords_removed_cv = cross_val_score(baseline_model, X_train_vectorized, y_tra
stopwords_removed_cv
```

Out[33]:

```
array([0.60253542, 0.61073826, 0.60895522, 0.60671642, 0.59925373])
```

In [34]:

```
print(baseline_cv.mean())
print(stopwords_removed_cv.mean())
```

```
0.6016115173572851
0.6056398098990506
```

Removing stopwords is very slightly better so we will keep them removed going forward. Lets now begin stemming words as well.

In [35]:
```python
stemmer = SnowballStemmer(language="english")

def stem_and_tokenize(document):
    tokens = tokenizer.tokenize(document)
    return [stemmer.stem(token) for token in tokens]

stemmed_stopwords = [stemmer.stem(word) for word in stopwords_list]
```

In [36]:
```python
tfidf = TfidfVectorizer(max_features=10,
                        stop_words=stemmed_stopwords,
                        tokenizer=stem_and_tokenize)

# Fit the vectorizer on X_train["text"] and transform it
X_train_vectorized = tfidf.fit_transform(X_train['tweet'])

# Visually inspect the vectorized data
pd.DataFrame.sparse.from_spmatrix(X_train_vectorized, columns=tfidf.get_feature_
```

Out[36]:

	app	appl	austin	circl	googl	ipad	iphon	launch	new	store
0	0.00000	0.000000	0.000000	0.0	1.0	0.0	0.000000	0.0	0.0	0.000000
1	0.63101	0.000000	0.000000	0.0	0.0	0.0	0.545373	0.0	0.0	0.551720
2	0.00000	0.000000	0.000000	0.0	1.0	0.0	0.000000	0.0	0.0	0.000000
3	0.00000	0.000000	0.000000	0.0	0.0	1.0	0.000000	0.0	0.0	0.000000
4	0.00000	0.000000	0.000000	0.0	1.0	0.0	0.000000	0.0	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...
6697	0.00000	0.000000	0.000000	0.0	1.0	0.0	0.000000	0.0	0.0	0.000000
6698	0.00000	0.000000	0.000000	0.0	0.0	0.0	1.000000	0.0	0.0	0.000000
6699	0.00000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.000000
6700	0.00000	0.000000	0.000000	0.0	0.0	1.0	0.000000	0.0	0.0	0.000000
6701	0.00000	0.751687	0.500963	0.0	0.0	0.0	0.000000	0.0	0.0	0.428955

6702 rows × 10 columns

In [37]:
```python
stemmed_cv = cross_val_score(baseline_model, X_train_vectorized, y_train['emotio
stemmed_cv
```

Out[37]:
```
array([0.59731544, 0.61222968, 0.60746269, 0.60223881, 0.59552239])
```

In [38]:
```python
print(baseline_cv.mean())
print(stopwords_removed_cv.mean())
print(stemmed_cv.mean())
```

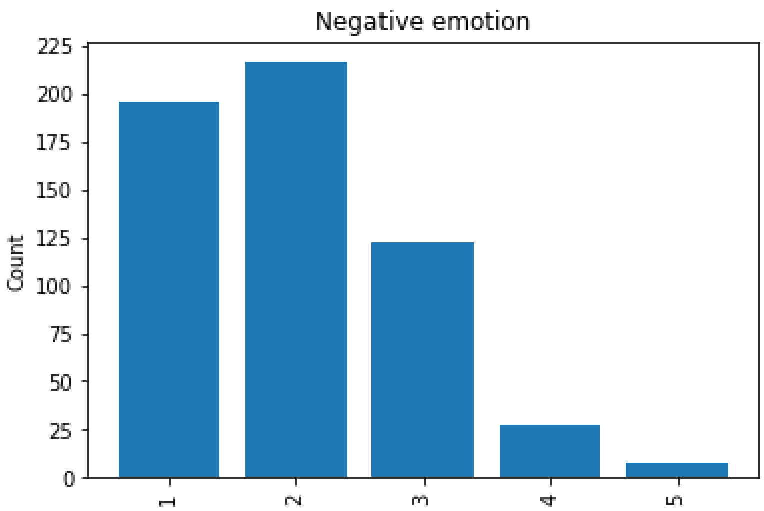
```
0.6016115173572851
0.6056398098990506
0.6029537992364797
```

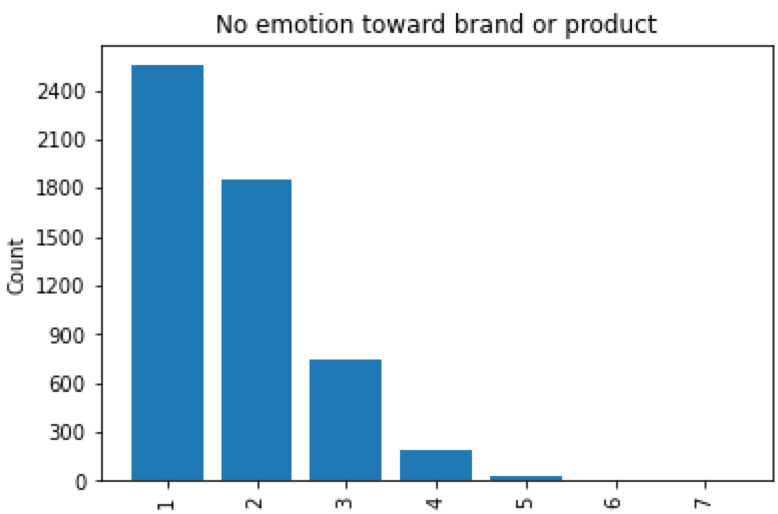
Well I didn't really expect that to do better after seeing it removed the letter e from many important words. Is sentance tokenizing valuable? Most tweets are single sentances regardless when capped at 140 or 280 characters...

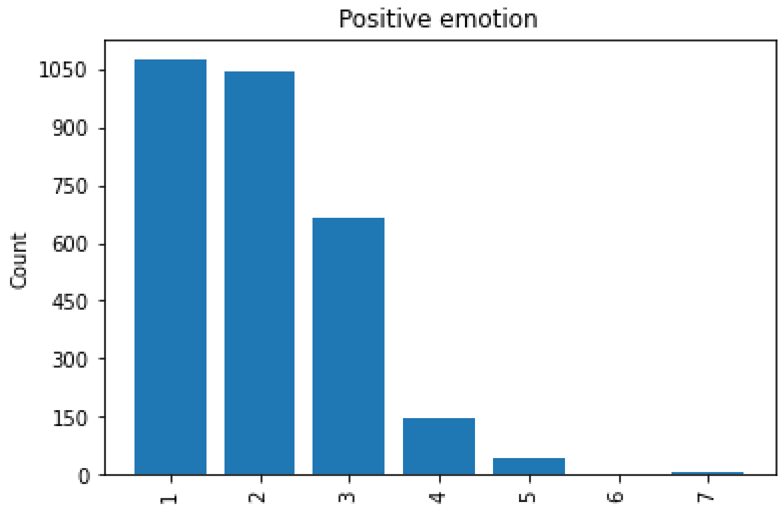
In [39]:
```python
df["num_sentences"] = df["tweet"].apply(lambda x: len(sent_tokenize(x)))
```

In [40]:
```python
def freq_sent_distribution_by_emotion(df):
    emotions = set(df['emotion'].values.tolist())

    #in the loop, create a freq dist for only a subject, then call the visualize

    for em in emotions:
        emotion_df = df[df['emotion'] == em]
        emotion_df_freq_dist = FreqDist(emotion_df["num_sentences"].explode())
        visualize_top_10(emotion_df_freq_dist, em)
```

In [41]:
```python
freq_sent_distribution_by_emotion(df)
```







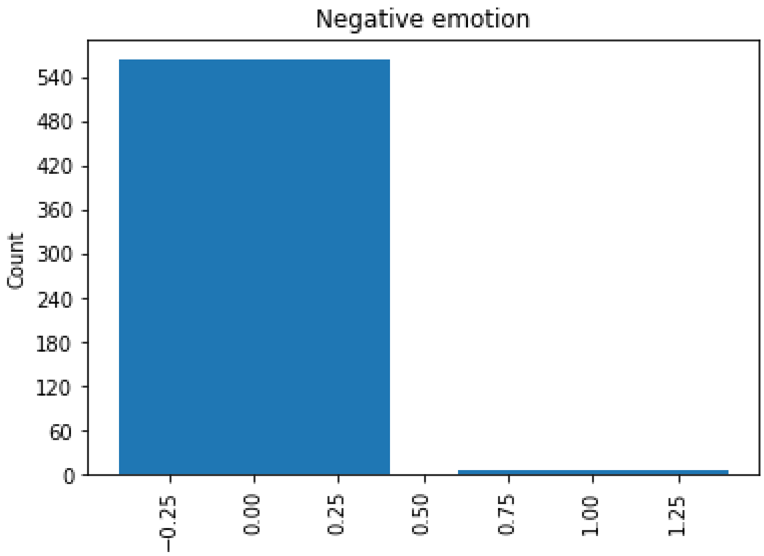
Ok maybe I'm wrong about what I expected, lets try creating another model and seeing if this can help.

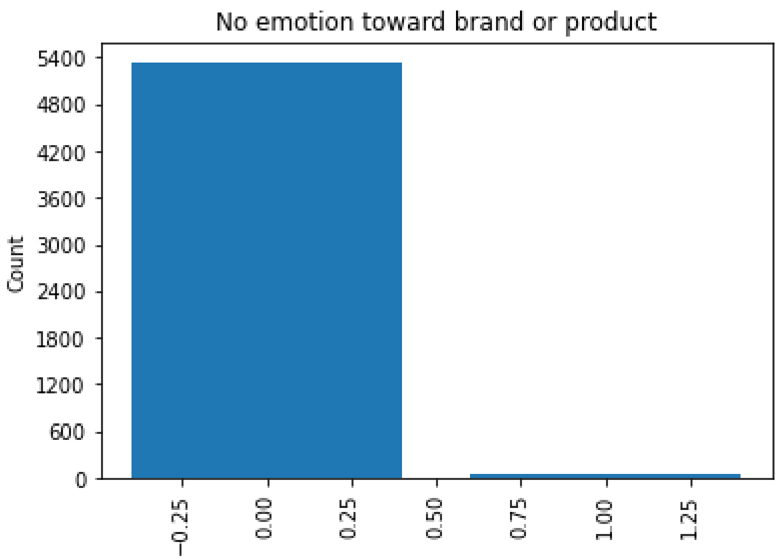
In [42]:
```
X_train["num_sentences"] = X_train['tweet'].apply(lambda x: len(sent_tokenize(x)
X_train.num_sentences.value_counts()
```

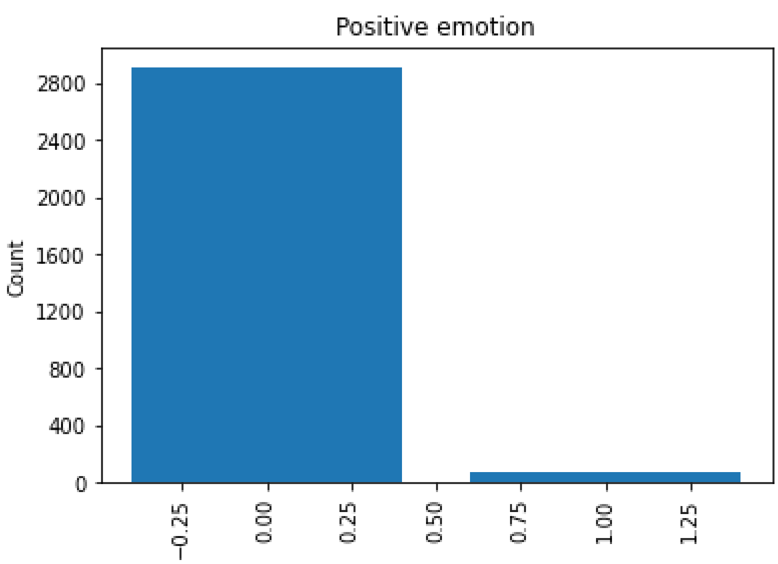
Out[42]:
```
1    2911
2    2312
3    1133
4     277
5      62
6       4
7       3
Name: num_sentences, dtype: int64
```

Lets try adding an emoticons deature as well

In [43]:
```
# Emoticons

emoticon_query = r'(?:[\:;X=B][-^]?[)\]3D([OP/\\|])(?:(?=\s))'

df["contains_emoticon"] = df["tweet"].str.contains(emoticon_query)

def freq_distribution_by_emotion(df):
    emotions = set(df['emotion'].values.tolist())

    #in the loop, create a freq dist for only a subject, then call the visualize

    for em in emotions:
        emotion_df = df[df['emotion'] == em]
        emotion_df_freq_dist = FreqDist(emotion_df["contains_emoticon"].explode(
        visualize_top_10(emotion_df_freq_dist, em)

freq_distribution_by_emotion(df)
```







In [44]:

```
X_train["contains_emoticon"] = X_train["tweet"].str.contains(emoticon_query)
X_train.head()
```

Out[44]:

	tweet	num_sentences	contains_emoticon
3363	having a great time at the google party #sxswi...	1	False
3204	@mention from:ubersocial for iphone now in the...	1	False
4460	are you in town for #sxsw? be sure to check in...	3	False
2311	the ironic tee has been usurped by the ipad 2 ...	1	False
6298	rt @mention marissa mayer: google will connect...	1	False

In [45]:
```
X_train.contains_emoticon.value_counts()
```

Out[45]:
```
False    6600
True      102
Name: contains_emoticon, dtype: int64
```

In [46]:
```
X_train.shape
```

Out[46]:
```
(6702, 3)
```

In [47]:
```
tfidf = TfidfVectorizer(
    max_features=10,
    stop_words=stopwords_list,
)

# Fit the vectorizer on X_train["text"] and transform it
X_train_vectorized = tfidf.fit_transform(X_train['tweet'])

# Create a full df of vectorized + engineered features
X_train_vectorized_df = pd.DataFrame(X_train_vectorized.toarray(), columns=tfidf

preprocessed_X_train = pd.concat([X_train_vectorized_df, X_train.reset_index()[[
                                 axis=1)

preprocessed_X_train
```

Out[47]:

	app	apple	austin	circles	google	ipad	iphone	new	social	store	num_
0	0.000000	0.00000	0.000000	0.0	1.0	0.0	0.00000	0.0	0.0	0.000000	
1	0.656654	0.00000	0.000000	0.0	0.0	0.0	0.52949	0.0	0.0	0.537072	
2	0.000000	0.00000	0.000000	0.0	1.0	0.0	0.00000	0.0	0.0	0.000000	
3	0.000000	0.00000	0.000000	0.0	0.0	1.0	0.00000	0.0	0.0	0.000000	
4	0.000000	0.00000	0.000000	0.0	1.0	0.0	0.00000	0.0	0.0	0.000000	
...	...	...	...	...	...	...	...	...	...	...	
6697	0.000000	0.00000	0.000000	0.0	1.0	0.0	0.00000	0.0	0.0	0.000000	
6698	0.000000	0.00000	0.000000	0.0	0.0	0.0	1.00000	0.0	0.0	0.000000	
6699	0.000000	0.00000	0.000000	0.0	0.0	0.0	0.00000	0.0	0.0	0.000000	
6700	0.000000	0.00000	0.000000	0.0	0.0	1.0	0.00000	0.0	0.0	0.000000	
6701	0.000000	0.75127	0.500424	0.0	0.0	0.0	0.00000	0.0	0.0	0.430313	

6702 rows × 12 columns

In [48]:
```
#Score the new model
preprocessed_cv = cross_val_score(baseline_model, preprocessed_X_train, y_train[
preprocessed_cv
```

Out[48]:
```
array([0.60850112, 0.61148397, 0.60895522, 0.60447761, 0.59776119])
```

In [49]:
```
print(baseline_cv.mean())
print(stopwords_removed_cv.mean())
print(stemmed_cv.mean())
print(preprocessed_cv.mean())
```

```
0.6016115173572851
0.6056398098990506
0.6029537992364797
0.6062358231215288
```

The Preprocessed is the best version but it's still incredibly marginal. How can I increase it?

In [50]:
```
#Trying more max features

tfidf = TfidfVectorizer(
    max_features=200,
    stop_words=stopwords_list,
)

# Fit the vectorizer on X_train["text"] and transform it
X_train_vectorized = tfidf.fit_transform(X_train['tweet'])

# Create a full df of vectorized + engineered features
X_train_vectorized_df = pd.DataFrame(X_train_vectorized.toarray(), columns=tfidf

final_X_train = pd.concat([X_train_vectorized_df, X_train.reset_index()[["num_se
                                 axis=1)

final_X_train
```

Out[50]:

	10	11	2011	30	6th	already	also	android	anyone	app	...	win	winning
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.000000	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.300485	...	0.000000	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.000000	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.000000	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.000000	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6697	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.000000	0.0
6698	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.000000	0.0
6699	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.000000	0.0
6700	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.461711	0.0
6701	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.000000	0.0

6702 rows × 202 columns

In [51]:
```
final_cv = cross_val_score(baseline_model, final_X_train, y_train['emotion'])
final_cv
```

Out[51]:
```
array([0.6383296 , 0.65175242, 0.64328358, 0.64477612, 0.63955224])
```

In [52]:
```
print('baseline: ', baseline_cv.mean())
print('stopwords removed: ', stopwords_removed_cv.mean())
print('stemmed: ', stemmed_cv.mean())
print('preprocessed (-stemmed, + Number of sentances and contains emoticons): ',
print('final (preprocessed + Max Features increased): ', final_cv.mean())
```

```
baseline:  0.6016115173572851
stopwords removed:  0.6056398098990506
stemmed:  0.6029537992364797
preprocessed (-stemmed, + Number of sentances and contains emoticons):  0.606235
8231215288
final (preprocessed + Max Features increased):  0.643538793727114
```

Testing using an SVC instead of MultinomialNB

In [53]:
```
from sklearn.svm import SVC

svc_clf = SVC(kernel='linear')
```

In [54]:
```
svc_cv = cross_val_score(svc_clf, final_X_train, y_train['emotion'])
svc_cv.mean()
```

Out[54]:
```
0.6480150700635525
```

Validation on the test set

Lets use the svc

In [55]:
```
final_model = SVC(kernel='linear')

final_model.fit(final_X_train, y_train)
final_model.score(final_X_train, y_train)
```

Out[55]:
```
0.6605490898239331
```

In [56]:
```
#Create X test and engineered features

X_test_vectorized = tfidf.transform(X_test["tweet"])

X_test["num_sentences"] = X_test["tweet"].apply(lambda x: len(sent_tokenize(x)))
X_test["contains_emoticon"] = X_test["tweet"].str.contains(emoticon_query)
```

In [57]:
```
#
X_test_vectorized_df = pd.DataFrame(X_test_vectorized.toarray(), columns=tfidf.g

final_X_test = pd.concat([X_test_vectorized_df, X_test.reset_index()[["num_sente
                                  axis=1)
final_X_test
```

Out[57]:

	10	11	2011	30	6th	already	also	android	anyone	app	...	win	winnin
0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	...	0.0	0.(
1	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	...	0.0	0.(
2	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	...	0.0	0.(
3	0.0	0.0	0.589882	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	...	0.0	0.(
4	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.306804	0.000000	0.000000	...	0.0	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	..
2229	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	...	0.0	0.(
2230	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	...	0.0	0.(
2231	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.778426	0.000000	...	0.0	0.(
2232	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.831881	...	0.0	0.(
2233	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	...	0.0	0.(

2234 rows × 202 columns

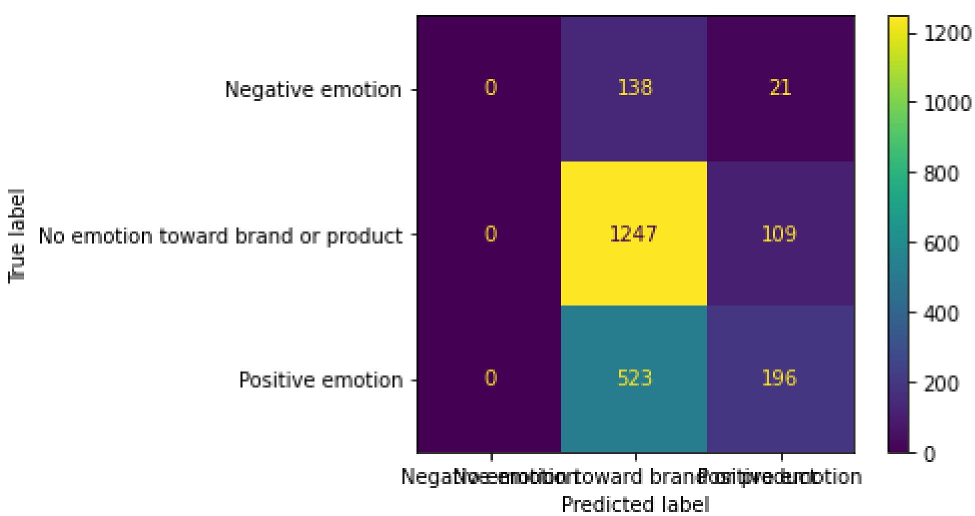
In [58]:

```
#final model test score
final_model.score(final_X_test, y_test)
```

Out[58]: 0.6459265890778872

In [59]:

```
#Plot confusion matrix
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(final_model, final_X_test, y_test);
```



In [60]:

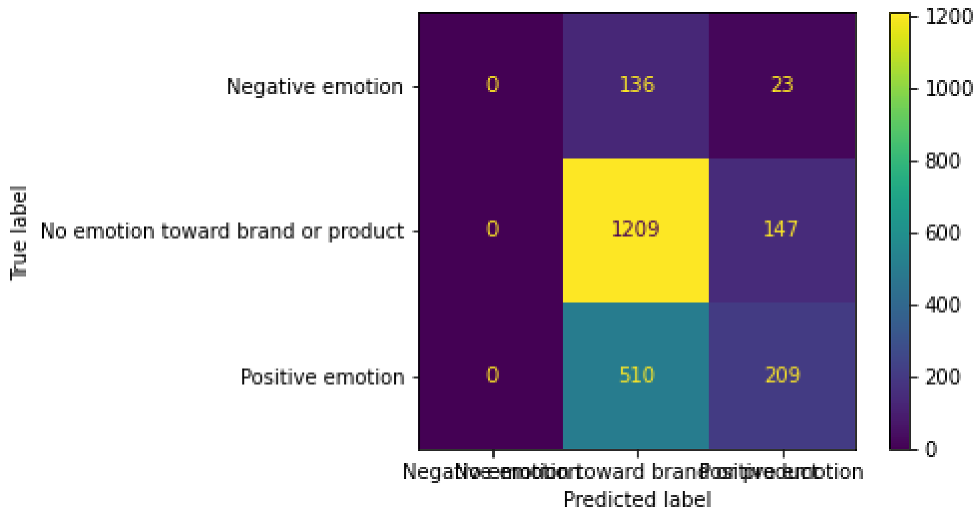
```
# F1 Score of svc model
svc_preds = final_model.predict(final_X_test)
scv_F1 = f1_score(y_test, svc_preds, average='weighted')
scv_F1
```

Out[60]: 0.5845216199204728

This model can't predict negative labels for some reason, or it just learned not to? But its still doing better than random guessing of 'No Emotion' which is kind of impressive.

MultinomialNB Confusion Matrix

In [61]:
```
baseline_model.fit(final_X_train, y_train)
plot_confusion_matrix(baseline_model, final_X_test, y_test);
```



In [62]:
```
#MultinomialNB F1 score
multiNB_preds = baseline_model.predict(final_X_test)
multiBN_f1 = f1_score(y_test, multiNB_preds, average='weighted')
multiBN_f1
```

Out[62]:
```
0.5796038842412904
```

Logistic Regression and Random Forests

In [63]:
```
# Use pipelines to create the models
rf =  Pipeline([('Random Forest', RandomForestClassifier(n_estimators=100, verbo
lr = Pipeline([('Logistic Regression', LogisticRegression())])
xgb =  Pipeline([('XGB Clf', XGBClassifier())])
```

In [64]:
```
# Create list of models
models = [('Random Forest', rf),
          ('Logistic Regression', lr),
          ('XGBoost', xgb)]
```

In [65]:
```
# Train the models and obtain the scores
scores = [(name, cross_val_score(model, final_X_train, y_train, cv=5).mean()) fo
scores
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

Out[65]:

```
[('Random Forest', 0.6590586218794172),
 ('Logistic Regression', 0.650404465368905),
 ('XGBoost', 0.6602528743308068)]
```

Testing sentiment analysis with VADER

In [66]:

```python
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()

#Load the data and preprocess similarly to before

vader_df = pd.read_csv('Data/data.csv', engine='python')
vader_df.columns = ['tweet', 'subject', 'emotion']

vader_df['emotion'] = vader_df['emotion'].replace("I can't tell", "No emotion to
vader_df['emotion'] = vader_df['emotion'].replace("No emotion toward brand or pr


vader_df = vader_df.dropna(subset=['tweet'])

vader_df.head()
```

Out[66]:

	tweet	subject	emotion
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion

In [67]:

```python
vader_df.emotion.value_counts()
```

Out[67]:

```
No emotion          5544
Positive emotion    2978
Negative emotion     570
Name: emotion, dtype: int64
```

In [68]:

```python
vader_df['scores'] = vader_df['tweet'].apply(lambda tweet: sid.polarity_scores(t

vader_df.head()
```

Out[68]:

	tweet	subject	emotion	scores
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion	{'neg': 0.203, 'neu': 0.797, 'pos': 0.0, 'comp...
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion	{'neg': 0.0, 'neu': 0.576, 'pos': 0.424, 'comp...
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion	{'neg': 0.0, 'neu': 0.663, 'pos': 0.337, 'comp...
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion	{'neg': 0.0, 'neu': 0.796, 'pos': 0.204, 'comp...

In [69]:

```
vader_df['compound'] = vader_df['scores'].apply(lambda score_dict: score_dict['c

vader_df.head()
```

Out[69]:

	tweet	subject	emotion	scores	compound
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion	{'neg': 0.203, 'neu': 0.797, 'pos': 0.0, 'comp...	-0.6800
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion	{'neg': 0.0, 'neu': 0.576, 'pos': 0.424, 'comp...	0.9100
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	0.0000
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion	{'neg': 0.0, 'neu': 0.663, 'pos': 0.337, 'comp...	0.7269
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion	{'neg': 0.0, 'neu': 0.796, 'pos': 0.204, 'comp...	0.6249

In [70]:

```
def emotion_helper(score):
    if score >= .333:
        return "Positive emotion"
    elif score <= -.333:
        return "Negative emotion"
    else:
        return "No emotion"

vader_df['comp_score'] = vader_df['compound'].apply(emotion_helper)

vader_df.head()
```

Out[70]:

	tweet	subject	emotion	scores	compound	comp_score
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion	{'neg': 0.203, 'neu': 0.797, 'pos': 0.0, 'comp...	-0.6800	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion	{'neg': 0.0, 'neu': 0.576, 'pos': 0.424, 'comp...	0.9100	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also.	iPad	Positive emotion	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0,	0.0000	No emotion

	tweet	subject	emotion	scores	compound	comp_score
	The...			'compound...		
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion	{'neg': 0.0, 'neu': 0.663, 'pos': 0.337, 'comp...	0.7269	Positive emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion	{'neg': 0.0, 'neu': 0.796, 'pos': 0.204, 'comp...	0.6249	Positive emotion

In [71]:
```
vader_df['Correct'] = vader_df.apply(lambda x: True if x['emotion'] == x['comp_s
```

In [72]:
```
vader_df.head()
```

Out[72]:

	tweet	subject	emotion	scores	compound	comp_score	Correct
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion	{'neg': 0.203, 'neu': 0.797, 'pos': 0.0, 'comp...	-0.6800	Negative emotion	True
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion	{'neg': 0.0, 'neu': 0.576, 'pos': 0.424, 'comp...	0.9100	Positive emotion	True
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	0.0000	No emotion	False
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion	{'neg': 0.0, 'neu': 0.663, 'pos': 0.337, 'comp...	0.7269	Positive emotion	False
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion	{'neg': 0.0, 'neu': 0.796, 'pos': 0.204, 'comp...	0.6249	Positive emotion	True

In [73]:
```
vader_df.Correct.value_counts()
```

Out[73]:
```
True      5192
False     3900
Name: Correct, dtype: int64
```

In [74]:
```
5192/(3900+5192)
```

Out[74]:
```
0.5710514738231413
```

This is worse than my initial attempts going to scrap VADER and move onto something else

Trying to use BERT

In [75]:
```
#Unused

import torch
from transformers import AutoTokenizer, DistilBertForSequenceClassification
```

```python
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-unc

inputs = tokenizer("I Love my girlfriend!", return_tensors="pt")

with torch.no_grad():
    logits = model(**inputs).logits

predicted_class_ids = torch.arange(0, logits.shape[-1])[torch.sigmoid(logits).sq

# To train a model on `num_labels` classes, you can pass `num_labels=num_labels`
num_labels = 3
model = DistilBertForSequenceClassification.from_pretrained(
    "distilbert-base-uncased", num_labels=num_labels, problem_type="multi_label_
)

labels = torch.sum(
    torch.nn.functional.one_hot(predicted_class_ids[None, :].clone(), num_classe
).to(torch.float)
loss = model(**inputs, labels=labels).loss
```

```
Some weights of the model checkpoint at distilbert-base-uncased were not used wh
en initializing DistilBertForSequenceClassification: ['vocab_layer_norm.weight',
'vocab_transform.bias', 'vocab_projector.weight', 'vocab_transform.weight', 'voc
ab_projector.bias', 'vocab_layer_norm.bias']
- This IS expected if you are initializing DistilBertForSequenceClassification f
rom the checkpoint of a model trained on another task or with another architectu
re (e.g. initializing a BertForSequenceClassification model from a BertForPreTra
ining model).
- This IS NOT expected if you are initializing DistilBertForSequenceClassificati
on from the checkpoint of a model that you expect to be exactly identical (initi
alizing a BertForSequenceClassification model from a BertForSequenceClassificati
on model).
Some weights of DistilBertForSequenceClassification were not initialized from th
e model checkpoint at distilbert-base-uncased and are newly initialized: ['class
ifier.bias', 'classifier.weight', 'pre_classifier.weight', 'pre_classifier.bia
s']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
Some weights of the model checkpoint at distilbert-base-uncased were not used wh
en initializing DistilBertForSequenceClassification: ['vocab_layer_norm.weight',
'vocab_transform.bias', 'vocab_projector.weight', 'vocab_transform.weight', 'voc
ab_projector.bias', 'vocab_layer_norm.bias']
- This IS expected if you are initializing DistilBertForSequenceClassification f
rom the checkpoint of a model trained on another task or with another architectu
re (e.g. initializing a BertForSequenceClassification model from a BertForPreTra
ining model).
- This IS NOT expected if you are initializing DistilBertForSequenceClassificati
on from the checkpoint of a model that you expect to be exactly identical (initi
alizing a BertForSequenceClassification model from a BertForSequenceClassificati
on model).
Some weights of DistilBertForSequenceClassification were not initialized from th
e model checkpoint at distilbert-base-uncased and are newly initialized: ['class
ifier.bias', 'classifier.weight', 'pre_classifier.weight', 'pre_classifier.bia
s']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

Neural Net Attempt

In [76]:
```
# Neural Net imports

from keras.preprocessing.sequence import pad_sequences
from keras.layers import Input, Dense, LSTM, Embedding
from keras.layers import Dropout, Activation, Bidirectional, GlobalMaxPool1D
from keras.models import Sequential
from keras import initializers, regularizers, constraints, optimizers, layers
from keras.preprocessing import text, sequence
```

In [77]:
```
#Data cleaning from the start, but shorter

nn_df = pd.read_csv('Data/data.csv', engine='python')
nn_df.columns = ['tweet', 'subject', 'emotion']
nn_df = nn_df.drop(unknowns_df.index.tolist())
nn_df = nn_df.dropna(subset=['tweet'])
nn_df['tweet'] = nn_df['tweet'].str.lower()
```

In [78]:
```
# Create a tokenizer using the 20,000 most used words
# Fit on the tweets
# Create a list of the tweets
# Set the list of tweets as your training data and pad them for even input into

tokenizer = text.Tokenizer(num_words=20000)
tokenizer.fit_on_texts(list(nn_df['tweet']))
list_tokenized_tweets = tokenizer.texts_to_sequences(df['tweet'])
X_t = sequence.pad_sequences(list_tokenized_tweets, maxlen=140)
```

In [79]:
```
#Set target (y) value as emotion
target = nn_df['emotion']
y = pd.get_dummies(target).values
```

In [80]:
```
# Create the model and layers
# Text models should use an embedding layer as the start
# Final layer is 3 for the output size (3 classes)

model = Sequential()

embedding_size = 128
model.add(Embedding(20000, embedding_size))
model.add(LSTM(25, return_sequences=True))
model.add(GlobalMaxPool1D())
model.add(Dropout(0.5))
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
```

In [81]:
```
# Compile Model
# Use 'categorical_crossentropy' for multi category classification
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

In [82]:
```python
# Summary print out
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, None, 128)         2560000   
_________________________________________________________________
lstm (LSTM)                  (None, None, 25)          15400     
_________________________________________________________________
global_max_pooling1d (Global (None, 25)                0         
_________________________________________________________________
dropout (Dropout)            (None, 25)                0         
_________________________________________________________________
dense (Dense)                (None, 50)                1300      
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 153       
=================================================================
Total params: 2,576,853
Trainable params: 2,576,853
Non-trainable params: 0
_________________________________________________________________
```

In [83]:
```python
#Train the model
model.fit(X_t, y, epochs=3, batch_size=32, validation_split=0.1)
```

```
Epoch 1/3
252/252 [==============================] - 19s 74ms/step - loss: 0.8926 - accura
cy: 0.5856 - val_loss: 0.7829 - val_accuracy: 0.6107
Epoch 2/3
252/252 [==============================] - 16s 64ms/step - loss: 0.7592 - accura
cy: 0.6677 - val_loss: 0.7007 - val_accuracy: 0.6577
Epoch 3/3
252/252 [==============================] - 16s 65ms/step - loss: 0.6083 - accura
cy: 0.7538 - val_loss: 0.7023 - val_accuracy: 0.6711
```

Out[83]:
```
<tensorflow.python.keras.callbacks.History at 0x7f915dcae040>
```

It looks like 3 epochs is actually the sweet spot and anything past that gives diminishing results. At 3 epochs we see .85 accuracy on the training data and .66 accuracy on the validation set. The training accuracy is much higher than our simpler models and the validation accuracy is right around where the training accuracy was so I would say this is a successful test.

In [84]:
```python
keras_preds = model.predict_classes(X_t)
```

```
WARNING:tensorflow:From <ipython-input-84-e192ed4d5f15>:1: Sequential.predict_cl
asses (from tensorflow.python.keras.engine.sequential) is deprecated and will be
removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model doe
s multi-class classification   (e.g. if it uses a `softmax` last-layer activatio
n).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary cla
ssification   (e.g. if it uses a `sigmoid` last-layer activation).
```

In [85]:
```python
keras_preds
```

Out[85]:
```
array([0, 2, 2, ..., 1, 1, 1])
```

In [86]:
```
target = pd.DataFrame(y)
target.columns = [0, 1, 2]
target = target.stack()
target = pd.Categorical(target[target!=0].index.get_level_values(1))
target = pd.DataFrame(target)
target.columns = ['Emotion']
```

In [87]:
```
print(confusion_matrix(target, keras_preds))
```

```
[[ 207  234  129]
 [  30 5094  264]
 [  14  996 1968]]
```

In [88]:
```
# Create Validation set
# Make X_t a dataframe
X_full = pd.DataFrame(X_t)

# Create a random sample of 700 rows from X
X_val = X_full.sample(n=1800, random_state=42)

# Create a subset of Y with the same indices as X_subset
Y_val = target.loc[X_val.index]
```

In [89]:
```
# Predict on validation set
val_preds = model.predict_classes(X_val)
```

In [90]:
```
# Keras F1 Score
keras_f1 = f1_score(Y_val, val_preds, average='weighted')
keras_f1
```

Out[90]:
```
0.8023441831895548
```

Not too much of a surprise that the neural net has the best F1 score so far. This is our best model at the moment in both accuracy and F1

Some Additional EDA

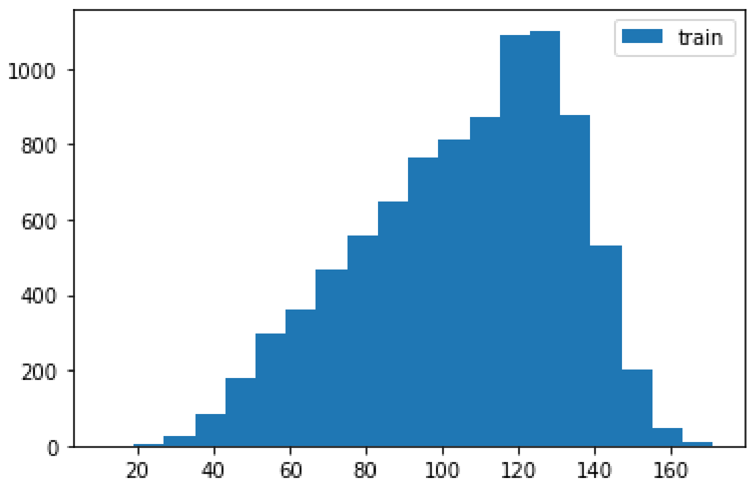
In [91]:
```
df.head()
```

Out[91]:

	tweet	subject	emotion	tweet_tokenized	tweet_without_stopwords	num_sentences	co
0	.@wesley83 i have a 3g iphone. after 3 hrs twe...	iPhone	Negative emotion	[wesley83, have, 3g, iphone, after, hrs, tweet...	[wesley83, 3g, iphone, hrs, tweeting, rise_aus...	5	
1	@jessedee know about @fludapp ? awesome ipad/i...	iPad or iPhone App	Positive emotion	[jessedee, know, about, fludapp, awesome, ipad...	[jessedee, know, fludapp, awesome, ipad, iphon...	3	
2	@swonderlin can not wait for #ipad 2 also. the...	iPad	Positive emotion	[swonderlin, can, not, wait, for, ipad, also, ...	[swonderlin, wait, ipad, also, sale]	2	

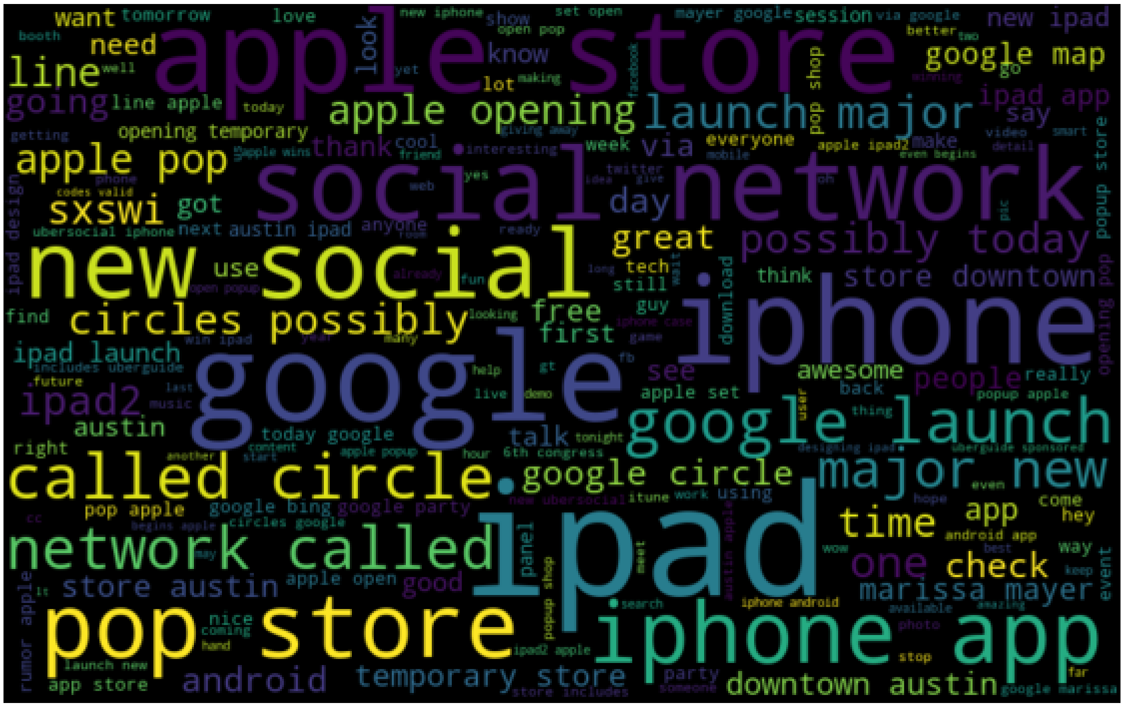
	tweet	subject	emotion	tweet_tokenized	tweet_without_stopwords	num_sentences	co
3	@sxsw i hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion	[sxsw, hope, this, year, festival, isn, as, cr...	[hope, year, festival, crashy, year, iphone, app]	2	
4	@sxtxstate great stuff on fri #sxsw: marissa m...	Google	Positive emotion	[sxtxstate, great, stuff, on, fri, sxsw, maris...	[sxtxstate, great, stuff, fri, marissa, mayer,...	1	

In [92]:
```
plt.hist(nn_df['tweet'].str.len(), bins=20, label='train')
plt.legend()
plt.show()
```



In [93]:
```
df['tidy_tweets'] = df.tweet_without_stopwords.apply(lambda x: ' '.join(x))
```

In [94]:
```
all_words = ' '.join([text for text in df['tidy_tweets']])

from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```



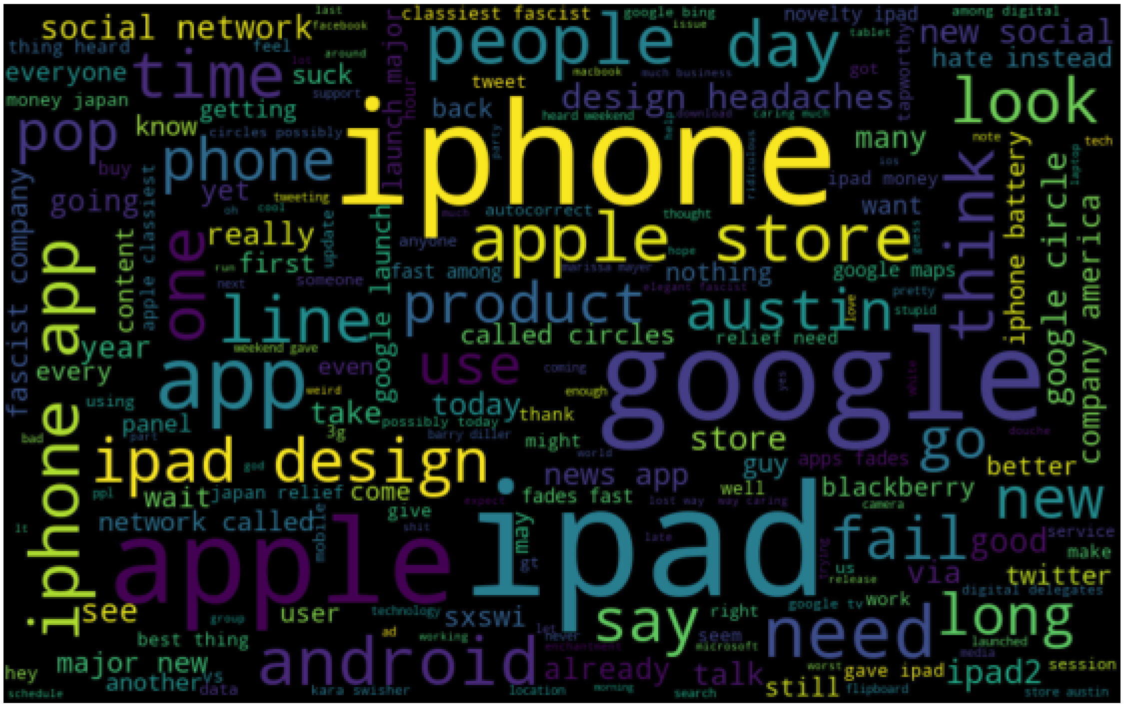
In [95]:

```
#positive words

pos_words =' '.join([text for text in df['tidy_tweets'][df['emotion'] == 'Positi

wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

In [96]:

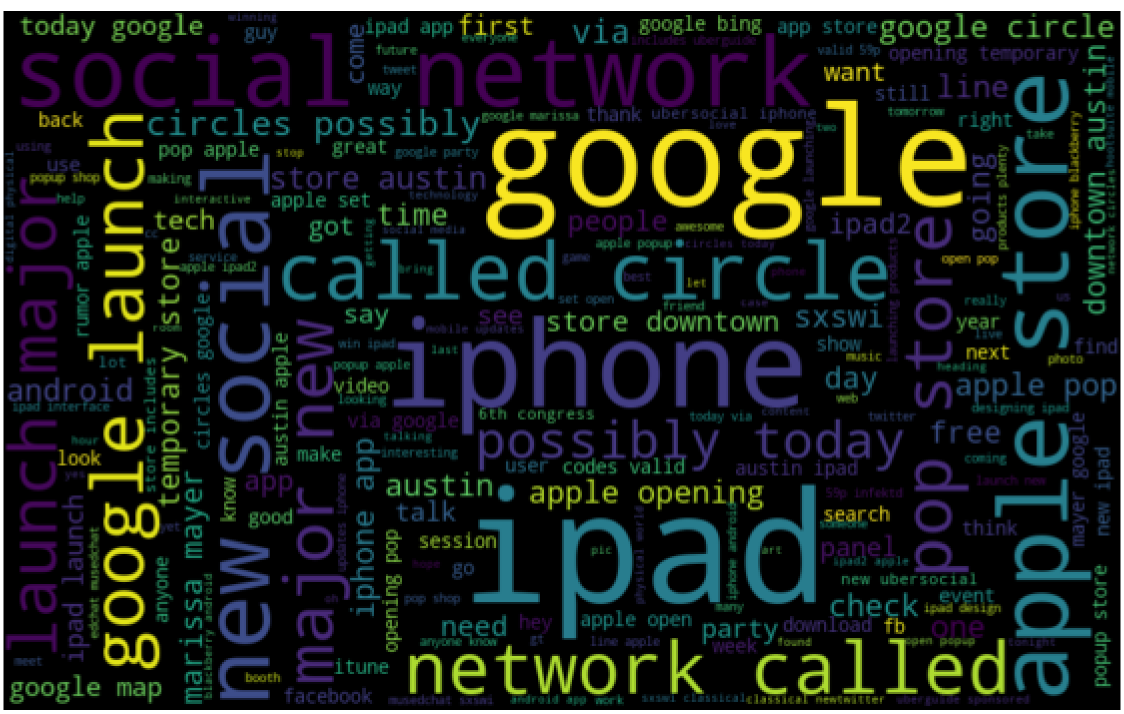
```
negative_words = ' '.join([text for text in df['tidy_tweets'][df['emotion'] == '

wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110)
```

```
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```



In [97]:

```
neutral_words = ' '.join([text for text in df['tidy_tweets'][df['emotion'] == 'N

wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```



In [98]:
```python
tokenized_tweet = df['tidy_tweets'].apply(lambda x: x.split()) # tokenizing

import gensim

model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            size=200, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2, # Ignores all words with total frequency lower than 2.
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 32, # no.of cores
            seed = 34
)

model_w2v.train(tokenized_tweet, total_examples= len(df['tidy_tweets']), epochs=
```

Out[98]:
```
(1320662, 1671440)
```

In [99]:
```python
df['tidy_tweets']
```

Out[99]:
```
0       wesley83 3g iphone hrs tweeting rise_austin de...
1       jessedee know fludapp awesome ipad iphone app ...
2                            swonderlin wait ipad also sale
3                  hope year festival crashy year iphone app
4       sxtxstate great stuff fri marissa mayer google...
                               ...
9088                                        ipad everywhere
9089    wave buzz interrupt regularly scheduled geek p...
9090    google zeiger physician never reported potenti...
9091    verizon iphone customers complained time fell ...
9092                              google tests check offers
Name: tidy_tweets, Length: 8936, dtype: object
```

In [100…
```python
total_vocabulary = []

for tweet in df['tweet_without_stopwords']:
    for word in tweet:
        total_vocabulary.append(word)

total_vocabulary = set(total_vocabulary)
```

In [101…
```python
len(total_vocabulary)
print('There are {} unique tokens in the dataset.'.format(len(total_vocabulary))
```

```
There are 9453 unique tokens in the dataset.
```

In [102…
```python
glove = {}
with open('glove.twitter.27B.50d.txt', 'rb') as f:
    for line in f:
        parts = line.split()
        word = parts[0].decode('utf-8')
        if word in total_vocabulary:
            vector = np.array(parts[1:], dtype=np.float32)
            glove[word] = vector
```

In [103…
```python
class W2vVectorizer(object):
```

```
    def __init__(self, w2v):
        # Takes in a dictionary of words and vectors as input
        self.w2v = w2v
        if len(w2v) == 0:
            self.dimensions = 0
        else:
            self.dimensions = len(w2v[next(iter(glove))])

    # Note: Even though it doesn't do anything, it's required that this object i
    # it can't be used in a scikit-learn pipeline
    def fit(self, X, y):
        return self

    def transform(self, X):
        return np.array([
            np.mean([self.w2v[w] for w in words if w in self.w2v]
```

Word2Vec of the previous models

In [104…

```
xgb = Pipeline([('Word2Vec Vectorizer', W2vVectorizer(glove)),
              ('XGB Clf', XGBClassifier())])
rf =  Pipeline([('Word2Vec Vectorizer', W2vVectorizer(glove)),
              ('Random Forest', RandomForestClassifier(n_estimators=100, verbose
svc = Pipeline([('Word2Vec Vectorizer', W2vVectorizer(glove)),
                ('Support Vector Machine', SVC())])
lr = Pipeline([('Word2Vec Vectorizer', W2vVectorizer(glove)),
              ('Logistic Regression', LogisticRegression())])
```

In [105…

```
models = [('Random Forest', rf),
          ('Support Vector Machine', svc),
          ('Logistic Regression', lr),
          ('XGBoost', xgb)]
```

In [106…

```
scores = [(name, cross_val_score(model, df['tweet_without_stopwords'], df['emoti
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    3.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    3.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    3.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    3.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    3.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

In [107…

```
scores
```

Out[107… [('Random Forest', 0.6820725498222935),
 ('Support Vector Machine', 0.6497306547786713),
 ('Logistic Regression', 0.6337273047075009),
 ('XGBoost', 0.6827426266510932)]

In [108…
```
#Fit, Predict, F1

rf.fit(df['tweet_without_stopwords'], df['emotion'])

rf_preds = rf.predict(df['tweet_without_stopwords'])

rf_f1 = f1_score(df['emotion'], rf_preds, average="weighted")

print("F1 Score: ", rf_f1)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    4.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
F1 Score:  0.9483871540487269
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.1s finished
```

In [109…
```
#Fit, Predict, F1

xgb.fit(df['tweet_without_stopwords'], df['emotion'])

xgb_preds = xgb.predict(df['tweet_without_stopwords'])

xgb_f1 = f1_score(df['emotion'], xgb_preds, average="weighted")

print("F1 Score: ", xgb_f1)
```

```
F1 Score:  0.9473074686501557
```

In [111…
```
# Create Validation set

# Create a random sample of 1800 rows from X
X_val = df.sample(n=1800, random_state=42)

# Create a subset of Y with the same indices as X_subset
Y_val = X_val['emotion']
```

In [112…
```
rf_preds = rf.predict(X_val['tweet_without_stopwords'])

rf_f1 = f1_score(Y_val, rf_preds, average="weighted")

print("Random Forest F1 Score: ", rf_f1)
```

```
Random Forest F1 Score:  0.9464691347279439
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

In [113…
```
xgb_preds = xgb.predict(X_val['tweet_without_stopwords'])

xgb_f1 = f1_score(Y_val, xgb_preds, average="weighted")

print("XGB F1 Score: ", xgb_f1)
```

```
XGB F1 Score:  0.9398811741374118
```

The F1 score of our random forest is incredible. This is the best model and by far better than the neural net

Lets explore similar words and offer google and apple reccomendations

In [114…

```
model_w2v.wv.most_similar(positive="google")
```

Out[114…

```
[('brother', 0.5848442316055298),
 ('incorrect', 0.5755859613418579),
 ('22sxsw', 0.5551255941390991),
 ('hee', 0.5488891005516052),
 ('pointing', 0.5466156005859375),
 ('nptech', 0.545316219329834),
 ('ne', 0.5447436571121216),
 ('nadja', 0.5431619882583618),
 ('spokewoman', 0.5382939577102661),
 ('marisa', 0.5367705821990967)]
```

In [115…

```
model_w2v.wv.most_similar(negative=["google"])
```

Out[115…

```
[('co', 0.0047996994107961655),
 ('kindle', 0.0021751169115304947),
 ('winning', 0.0006275922060012817),
 ('16gb', -0.0008450224995613098),
 ('mophie', -0.0037463074550032616),
 ('bar', -0.00474444217979908),
 ('pure', -0.004777221009135246),
 ('screen', -0.004891552031040192),
 ('black', -0.008278627879917622),
 ('pack', -0.01591699942946434)]
```

In [116…

```
model_w2v.wv.most_similar(positive="apple")
```

Out[116…

```
[('sixth', 0.6366628408432007),
 ('5000', 0.6305968165397644),
 ('tcrn', 0.6255877017974854),
 ('shut', 0.6254369616508484),
 ('impulse', 0.624621570110321),
 ('applestore', 0.6233795285224915),
 ('rage', 0.622353434562683 1),
 ('toptweets', 0.6208226680755615),
 ('geekfest', 0.6204372048377991),
 ('brian_lam', 0.6194248795509338)]
```

In [117…

```
model_w2v.wv.most_similar(negative=["apple"])
```

Out[117…

```
[('catch', -0.007958658039569855),
 ('twitter', -0.034292034804821014),
 ('rules', -0.03449346497654915),
 ('panel', -0.04481664299964905),
 ('browser', -0.0509837344288826),
 ('nyt', -0.05322345346212387),
 ('start', -0.05585930123925209),
 ('excel', -0.05867619812488556),
 ('need', -0.06025959178805351),
 ('scrape', -0.06044682115316391)]
```

In [118…

```
model_w2v.wv.most_similar(positive="iphone")
```

Out[118…

```
[('desperate', 0.5116559267044067),
 ('sync', 0.49874168634414673),
 ('handy', 0.4877375066280365),
 ('wew', 0.4863353669643402),
```

```
 ('fully', 0.4829857647418976),
 ('schedules', 0.4818391799926758),
 ('charts', 0.4782484769821167),
 ('communications', 0.4768584072589874),
 ('merchant', 0.47590404748916626),
 ('hobo', 0.4734790027141571)]
```

In [119…
```
model_w2v.wv.most_similar(negative=["iphone"])
```

Out[119…
```
[('jobs', 0.029164083302021027),
 ('guy', 0.01111283153295517),
 ('street', 0.007235661149024963),
 ('part', -0.0006067678332328796),
 ('owners', -0.008246984332799911),
 ('front', -0.010976498946547508),
 ('head', -0.016630683094263077),
 ('steve', -0.018890613690018654),
 ('miss', -0.019209370017051697),
 ('sales', -0.019321508705615997)]
```

In [120…
```
model_w2v.wv.most_similar(positive="ipad")
```

Out[120…
```
[('adam', 0.49525368213653564),
 ('attn', 0.4878769516944885),
 ('relic', 0.4720078408718109),
 ('resist', 0.4668174982070923),
 ('channels', 0.4613021910190582),
 ('kenny', 0.46007850766181946),
 ('smarty', 0.45260828733444214),
 ('commercial', 0.45218873023986816),
 ('baby', 0.4521023631095886),
 ('smileyparty', 0.45200440287590027)]
```

In [121…
```
model_w2v.wv.most_similar(negative=["ipad"])
```

Out[121…
```
[('system', 0.02862054854631424),
 ('engine', -0.006649543531239033),
 ('town', -0.009210258722305298),
 ('hotpot', -0.009825445711612701),
 ('info', -0.018528716638684273),
 ('recommendation', -0.02047230675816536),
 ('become', -0.027191689237952232),
 ('directions', -0.027908606454730034),
 ('nerds', -0.03062290884554386),
 ('behind', -0.0346079058945179)]
```

In [122…
```
model_w2v.wv.most_similar(positive="app")
```

Out[122…
```
[('wew', 0.5776689052581787),
 ('yayrt', 0.5335705280303955),
 ('forbes', 0.5316839814186096),
 ('sync', 0.5285147428512573),
 ('wedig', 0.5284035205841064),
 ('workspace', 0.5283859968185425),
 ('lightbox', 0.5249691009521484),
 ('concertgoers', 0.5216178894042969),
 ('nicely', 0.5132554173469543),
 ('casa', 0.5123087763786316)]
```

In [123…
```
model_w2v.wv.most_similar(negative=["app"])
```

Out[123…
```
[('30', 0.008424434810876846),
 ('front', 0.0036735422909259796),
 ('headline', -0.005456060171127319),
```

```
('room', -0.006305336952209473),
('guy', -0.007032092660665512),
('recipe', -0.015117660164833069),
('matt', -0.016400093212723732),
('instead', -0.017644822597503662),
('existence', -0.02015220746397972),
('pay', -0.020419329404830933)]
```

Apple

1. One of the most positive words associated with Apple is Store / applestore. People love the Apple Store, it was revolutionary when first introduced. How can you market the store and highlight how great it is, and get people to come in?
2. Some common Positive iphone words are handy and schedules. People love how great the iphone is as a personal device for day to day tasks. Create marketing to highlight businessmen using the iphone for scheduling meetings, calender apointments, etc. Or families scheduling playdates for children and soccer games etc.
3. A recurring negative word is sales. Prices are high and there aren't enough sales for iphones, ipads, etc.