E. Berke Tezcan

TABLE OF CONTENTS

Click to jump to matching Markdown Header.

INTRODUCTION

> Explain the point of your project and what question you are trying to answer with your modeling.

Business Problem

Summary of the business problem you are trying to solve, and the data questions that you plan to answer to solve them.

Questions to consider:

- What are the business's pain points related to this project?
- How did you pick the data analysis question(s) that you did?
- Why are these questions important from a business perspective?

OBTAIN

In [1]:
```
#Option to run gridsearches within the notebook
run_gridsearches = False
```

In [2]:
```
import pandas as pd
import nltk
```

We will be using a dataset from data.world provided by CrowdFlower which has tweets about Apple and Google from the South by Southwest (SXSW) conference. The tweet labels were crowdsourced and reflect which emotion they convey and what product/service/company this emotion is directed at based on the content.

In [3]:
```
df = pd.read_csv('data/tweet_data_apple_google.csv', encoding='latin-1')
df.head()
```

Out[3]:

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_product
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

The column names are unnecessarily long and difficult to read. We can rename the columns to make it easier for readability and interpretability.

In [4]:
```
df.columns = ['Tweet', 'Product/Brand', 'Emotion']
df.head()
```

Out[4]:

	Tweet	Product/Brand	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 [5]:
```
df['Product/Brand'].unique()
```

Out[5]:
```
array(['iPhone', 'iPad or iPhone App', 'iPad', 'Google', nan, 'Android',
       'Apple', 'Android App', 'Other Google product or service',
       'Other Apple product or service'], dtype=object)
```

SCRUB

Addressing Missing/Placeholder Values

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Tweet          9092 non-null   object
 1   Product/Brand  3291 non-null   object
 2   Emotion        9093 non-null   object
dtypes: object(3)
memory usage: 213.2+ KB
```

It seems like we are missing the body of text for 1 tweet and a total of 5,802 tags for which product/company the corresponding tweet was about. Let's start with looking at the missing tweet.

In [7]:
```
df[df['Tweet'].isna()]
```

Out[7]:

	Tweet	Product/Brand	Emotion
6	NaN	NaN	No emotion toward brand or product

It seems like we are missing all the information from this row of data and without text data this row is of no use to us so we can go ahead and drop it.

In [8]:
```
df = df[df['Tweet'].isna() == False]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9092 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Tweet          9092 non-null   object
 1   Product/Brand  3291 non-null   object
 2   Emotion        9092 non-null   object
dtypes: object(3)
memory usage: 284.1+ KB
```

Now, let's take a closer look at the product/company column's missing values.

In [9]:
```
df[df['Product/Brand'].isna()].head()
```

Out[9]:

	Tweet	Product/Brand	Emotion
5	@teachntech00 New iPad Apps For #SpeechTherapy...	NaN	No emotion toward brand or product
16	Holler Gram for iPad on the iTunes App Store -...	NaN	No emotion toward brand or product
32	Attn: All #SXSW frineds, @mention Register fo...	NaN	No emotion toward brand or product
33	Anyone at #sxsw want to sell their old iPad?	NaN	No emotion toward brand or product
34	Anyone at #SXSW who bought the new iPad want ...	NaN	No emotion toward brand or product

It looks like these tweets are not really directed towards a specific product or brand so we can go

ahead and fill the null values with "Unknown" as a placeholder value for the time being.

In [10]:
```
df['Product/Brand'].fillna('Unknown', inplace=True)
```

In [11]:
```
df.isna().sum()
```

Out[11]:
```
Tweet           0
Product/Brand   0
Emotion         0
dtype: int64
```

Scrubbing Emotion Column

In [12]:
```
df['Emotion'].value_counts()
```

Out[12]:
```
No emotion toward brand or product    5388
Positive emotion                      2978
Negative emotion                       570
I can't tell                           156
Name: Emotion, dtype: int64
```

Similar to the initial column names, the values in the emotion column can be cleaned up a bit for easy interpretability and to shorten the amount of typing for the code.

In [13]:
```
emotion_dict = {'Positive emotion': 'Positive', 'Negative emotion': 'Negative',
                'No emotion toward brand or product': 'Neutral', "I can't tell": 'Unkno
df['Emotion'] = df['Emotion'].map(emotion_dict)
df.head()
```

Out[13]:

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

In [14]:
```
df['Emotion'].value_counts()
```

Out[14]:
```
Neutral     5388
Positive    2978
Negative     570
Unknown      156
Name: Emotion, dtype: int64
```

Now we can take a look at the tweets with "Unknown" emotion values to see if we can notice any patterns or tell easily whether the tweet has a negative, neutral or positive emotion.

In [15]:
```
pd.set_option("display.max_colwidth", 300)
df[df['Emotion']=='Unknown']
```

Out[15]:

	Tweet	Product/Brand	Emotion
90	Thanks to @mention for publishing the news of @mention new medical Apps at the #sxswi conf. blog {link} #sxsw #sxswh	Unknown	Unknown

	Tweet	Product/Brand	Emotion
102	□ÛÏ@mention "Apple has opened a pop-up store in Austin so the nerds in town for #SXSW can get their new iPads. {link} #wow	Unknown	Unknown
237	Just what America needs. RT @mention Google to Launch Major New Social Network Called Circles, Possibly Today {link} #sxsw	Unknown	Unknown
341	The queue at the Apple Store in Austin is FOUR blocks long. Crazy stuff! #sxsw	Unknown	Unknown
368	Hope it's better than wave RT @mention Buzz is: Google's previewing a social networking platform at #SXSW: {link}	Unknown	Unknown
...	...	...	...
9020	It's funny watching a room full of people hold their iPad in the air to take a photo. Like a room full of tablets staring you down. #SXSW	Unknown	Unknown
9032	@mention yeah, we have @mention , Google has nothing on us :) #SXSW	Unknown	Unknown
9037	@mention Yes, the Google presentation was not exactly what I was expecting. #sxsw	Unknown	Unknown
9058	"Do you know what Apple is really good at? Making you feel bad about your Xmas present!" - Seth Meyers on iPad2 #sxsw #doyoureallyneedthat?	Unknown	Unknown
9066	How much you want to bet Apple is disproportionately stocking the #SXSW pop-up store with iPad 2? The influencer/hipsters thank you	Apple	Unknown

156 rows × 3 columns

Arguably, these tweets are difficult to classify without more context. Some of the tweets could be taken as sarcastic or genuine depending on the context.

Since we need to have labels for our models, these tweets will not be useful in developing our model. Luckily they only make up approximately 1.7% of our data so we can go ahead and drop these rows.

In [16]:
```
df = df[df['Emotion']!='Unknown']
df['Emotion'].value_counts()
```

Out[16]:
```
Neutral     5388
Positive    2978
Negative     570
Name: Emotion, dtype: int64
```

Checking for and Addressing Duplicates

Now that we've addressed null values, and cleaned up our dataset a little bit more, next step is to check for duplicates to see if we have the same tweets represented multiple times.

In [17]:
```
len(df[df.duplicated()])
```

Out[17]:
```
22
```

It looks like we have 22 tweets that are duplicated in the dataset. Let's take a look at the duplicates for reference.

In [18]:
```
df[df.duplicated()]
```

Out[18]:

	Tweet	Product/Brand	Emotion
468	Before It Even Begins, Apple Wins #SXSW {link}	Apple	Positive
776	Google to Launch Major New Social Network Called Circles, Possibly Today {link} #sxsw	Unknown	Neutral
2232	Marissa Mayer: Google Will Connect the Digital & Physical Worlds Through Mobile - {link} #sxsw	Unknown	Neutral
2559	Counting down the days to #sxsw plus strong Canadian dollar means stock up on Apple gear	Apple	Positive
3950	Really enjoying the changes in Gowalla 3.0 for Android! Looking forward to seeing what else they & Foursquare have up their sleeves at #SXSW	Android App	Positive
3962	#SXSW is just starting, #CTIA is around the corner and #googleio is only a hop skip and a jump from there, good time to be an #android fan	Android	Positive
4897	Oh. My. God. The #SXSW app for iPad is pure, unadulterated awesome. It's easier to browse events on iPad than on the website!!!	iPad or iPhone App	Positive
5338	RT @mention □÷¼ GO BEYOND BORDERS! □÷_ {link} □ã_ #edchat #musedchat #sxsw #sxswi #classical #newTwitter	Unknown	Neutral
5341	RT @mention □÷¼ Happy Woman's Day! Make love, not fuss! □÷_ {link} □ã_ #edchat #musedchat #sxsw #sxswi #classical #newTwitter	Unknown	Neutral
5881	RT @mention Google to Launch Major New Social Network Called Circles, Possibly Today {link} #sxsw	Unknown	Neutral
5882	RT @mention Google to Launch Major New Social Network Called Circles, Possibly Today {link} #SXSW	Unknown	Neutral
5883	RT @mention Google to Launch Major New Social Network Called Circles, Possibly Today {link} #sxsw	Unknown	Neutral
5884	RT @mention Google to Launch Major New Social Network Called Circles, Possibly Today {link} #SXSW	Unknown	Neutral
5885	RT @mention Google to Launch Major New Social Network Called Circles, Possibly Today {link} #sxsw	Unknown	Neutral
6296	RT @mention Marissa Mayer: Google Will Connect the Digital & Physical Worlds Through Mobile - {link} #sxsw	Google	Positive
6297	RT @mention Marissa Mayer: Google Will Connect the Digital & Physical Worlds Through Mobile - {link} #SXSW	Unknown	Neutral
6298	RT @mention Marissa Mayer: Google Will Connect the Digital & Physical Worlds Through Mobile - {link} #sxsw	Google	Positive
6299	RT @mention Marissa Mayer: Google Will Connect the Digital & Physical Worlds Through Mobile - {link} #SXSW	Unknown	Neutral
6300	RT @mention Marissa Mayer: Google Will Connect the Digital & Physical Worlds Through Mobile - {link} #sxsw	Unknown	Neutral
6546	RT @mention RT @mention Google to Launch Major New Social Network Called Circles, Possibly Today {link} #sxsw	Unknown	Neutral

	Tweet	Product/Brand	Emotion
8483	I just noticed DST is coming this weekend. How many iPhone users will be an hour late at SXSW come Sunday morning? #SXSW #iPhone	iPhone	Negative
8747	Need to buy an iPad2 while I'm in Austin at #sxsw. Not sure if I'll need to Q up at an Austin Apple store?	iPad	Positive

In [19]:
```
df.drop_duplicates(keep='first', inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8914 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Tweet          8914 non-null   object
 1   Product/Brand  8914 non-null   object
 2   Emotion        8914 non-null   object
dtypes: object(3)
memory usage: 278.6+ KB
```

Now that the duplicates have been addressed too we can perform an EDA on the data.

EXPLORE

With the business problem in mind, it is important to isolate and analyze positive and negative tweets as a whole as well as on a company and product basis. We will start our exploration with positive tweets.

Tweets with Positive Sentiment

In [20]:
```
#parsing positive tweets into new df
df_positive = df[df['Emotion']=='Positive']
#verifying that neutral and negative tweets have been removed
df_positive['Emotion'].value_counts()
```

Out[20]:
```
Positive    2970
Name: Emotion, dtype: int64
```

In [21]:
```
#parsing tweets into a list
corpus_pos = df_positive['Tweet'].to_list()
corpus_pos[:5]
```

Out[21]:
```
["@jessedee Know about @fludapp ? Awesome iPad/iPhone app that you'll likely appreciate
for its design. Also, they're giving free Ts at #SXSW",
 '@swonderlin Can not wait for #iPad 2 also. They should sale them down at #SXSW.',
 "@sxtxstate great stuff on Fri #SXSW: Marissa Mayer (Google), Tim O'Reilly (tech books/
conferences) &amp; Matt Mullenweg (Wordpress)",
 '#SXSW is just starting, #CTIA is around the corner and #googleio is only a hop skip an
d a jump from there, good time to be an #android fan',
 'Beautifully smart and simple idea RT @madebymany @thenextweb wrote about our #hollergr
am iPad app for #sxsw! http://bit.ly/ieaVOB']
```

Tokenization

For tokenizing the tweets we will be using the TweetTokenizer throughout this project since it has built-in functionality such as processing handles and hashtags correctly unlike other tokenizers. We

will drop any handles from the tweets since we are mainly focused on the content of the tweet.

In [22]:
```
from nltk import TweetTokenizer
import string

#functionize tokenization of tweets
def tokenize_tweets(corpus, preserve_case=False, strip_handles=True):
    tokenizer = TweetTokenizer(preserve_case=preserve_case, strip_handles=strip_handles
    tokens = tokenizer.tokenize(','.join(corpus))
    return tokens
```

In [23]:
```
#tokenize positive tweets
tokens_pos = tokenize_tweets(corpus_pos)
```

In [24]:
```
#displaying 10 most common tokens
from nltk import FreqDist
freq = FreqDist(tokens_pos)
freq.most_common(10)
```

Out[24]:
```
[(',', 4050),
 ('#sxsw', 2983),
 ('.', 2230),
 ('the', 1590),
 ('!', 1241),
 ('link', 1214),
 ('{', 1210),
 ('}', 1210),
 ('to', 1154),
 ('at', 1019)]
```

We can see that our tokens contain a lot of punctuation and stop words like "to" and "at" since we haven't removed these yet. Prior to removing these though, we will be lemmatizing the tokens to make sure we're capturing any stop words that may be generated with this process.

Lemmatization

In [26]:
```
from nltk.stem.wordnet import WordNetLemmatizer
# nltk.download('wordnet')

#functionize lemmatization of tokens
def lemmatize_tokens(tokens_list):
    lemmatizer = WordNetLemmatizer()
    tokens_lemm = [lemmatizer.lemmatize(word) for word in tokens_list]
    return tokens_lemm
```

In [27]:
```
#lemmatizing positive tweet tokens
tokens_pos_lemm = lemmatize_tokens(tokens_pos)
```

In [28]:
```
#displaying most common 10 tokens
from nltk import FreqDist
freq = FreqDist(tokens_pos_lemm)
freq.most_common(10)
```

Out[28]:
```
[(',', 4050),
 ('#sxsw', 2983),
 ('.', 2230),
 ('the', 1590),
 ('!', 1241),
 ('link', 1218),
```

```
('{', 1210),
('}', 1210),
('to', 1154),
('at', 1019)]
```

We can see that we still have punctuation and stop words in our list since we did not remove these yet. Now that we lemmatized our tokens, since the stop words and punctuation arguably does not reveal anything about the sentiment of the tweets, we can go ahead and remove them. This will allow us to see more relevant information and will allow for models to perform better.

Stop Word/Punctuation Removal

In [29]:
```
#getting stopwords from nltk
from nltk.corpus import stopwords
stop_lst = stopwords.words('english')
stop_lst += list(string.punctuation)
#adding additional characters and empty string to stop words
additional_punc = ['“','”','...',"''",'’','``','']
stop_lst += additional_punc
```

In [30]:
```
#functionizing removal of stop words
def remove_stop(tokens, stop_list=stop_lst):

    """This function takes removes stop words from a given tokens list. """

    #encoding/decoding tokens to eliminate unrecognized symbols and eliminating externa
    tokens_stopped = [word.encode('ascii','ignore').decode() for word in tokens
                      if (word not in stop_list) & (word.startswith('http') == False)]

    return tokens_stopped
```

In [31]:
```
#removing stop words from lemmatized tokens
tokens_pos_ls = remove_stop(tokens_pos_lemm)
```

In [32]:
```
#displaying most common 50 tokens
from nltk import FreqDist
freq = FreqDist(tokens_pos_ls)
freq.most_common(50)
```

Out[32]:
```
[('#sxsw', 2983),
 ('link', 1218),
 ('ipad', 1010),
 ('rt', 931),
 ('apple', 711),
 ('google', 602),
 ('2', 595),
 ('store', 554),
 ('iphone', 466),
 ('', 443),
 ('app', 387),
 ('new', 358),
 ('austin', 250),
 ('get', 181),
 ('#apple', 174),
 ('launch', 173),
 ('android', 161),
 ('party', 151),
 ('pop-up', 151),
 ('sxsw', 144),
 ('line', 143),
```

```
('time', 136),
('great', 135),
('via', 132),
('#ipad2', 129),
('day', 124),
('social', 122),
('free', 120),
('cool', 119),
("i'm", 115),
('like', 115),
('map', 115),
('one', 114),
('win', 112),
('today', 111),
('ha', 108),
('circle', 107),
('w', 104),
('go', 104),
('come', 103),
('wa', 100),
('#sxswi', 96),
('awesome', 93),
('#ipad', 93),
('love', 93),
('good', 92),
('network', 91),
('mobile', 90),
('temporary', 89),
('downtown', 88)]
```

When we look at our updated most common 50 words list above, we can see that we still have some words that do not provide us with useful information about the sentiment of the tweets. Since we already know that the tweets are from the SXSW Conference, we can remove hashtags like '#sxsw' by adding them to the stop words list. Additionally, words like 'link' and 'rt' seem to be referring to external links and retweets respectively. However, 'link' could also have been used with its dictionary meaning. In order to understand how a word appears in our corpus we can build a function that can take a random sample of tweets out of the corpus, check whether the word we are looking for appears in any of the tweets and display the tweets that the word appears in.

In [33]:
```
#adding sxsw to stop list
stop_lst += ['#sxsw', '#sxswi', 'sxsw']
```

In [34]:
```
#defining a function that will provide context for a given word

import numpy as np
def context_finder(word, corpus, n_samples=5, n_count=5):
    """This function takes n_samples with each sample having n_count tweets from the gi
    and displays tweets that have the specified word in them. The goal of the function
    some context about a word."""

    i=0

    for _ in list(range(0,n_samples)):
        sample = np.random.choice(corpus, n_count)
        for tweet in sample:
            if word in tweet:
                print(tweet)
                i += 1
    print('-----------------------------------')
    print(f'Out of {n_count*n_samples} tweets analyzed, {i} tweets had the word "{word}
```

In [35]:
```
#verifying that 'link' is used in reference to external web links
context_finder('link', corpus_pos)
```

```
Google knows everything!! Map of people traffic around the world #sxtxstate #sxsw {link}
Genius idea RT @mention Apple Popup Store at #SXSW {link}
Monday's Adfonic Daily is Out! Read about #iphone white release #Google social #sxsw and
all things #mobile here ÐÜ {link}
ÛÏ@mention Google to Launch Major New Social Network Called Circles, Possibly Today {li
nk} #sxswÛ @mention  bigger than gaga
@mention Hey Taariq, howdy from Texas, fav #sXsw app so far is the official one {link} &
lt;--android-or get Iphone #UsGuys
Grouped{in} nice app for groups, aber leider nur fÌ_r iPhone.  {link} #sxsw
Smart move by Apple opening popup store for #iPad2 launch in downtown Austin during #SXS
W. {link}
great talk about the Google Doodles and in the end they hand out printed signed doodles
#SXSW  {link}
Best photo booth so far at Google's League of Extraordinary H4ckers! #sxsw  {link}
@mention at #cnngrill #sxsw rockin the #iphone couldn't be cooler {link}
I threw this together to keep on my iPad to attempt to keep up with all the after hours
stuff this weekend. #sxsw  {link}
RT @mention Here's Google Realtime for #SXSW {link} via @mention
-----------------------------------
Out of 25 tweets analyzed, 12 tweets had the word "link" in them.
```

We see that the word 'link' is mainly used to refer to web links that were removed when the data was being parsed into the dataset. This doesn't provide us with any valuable insights into what the tweet may be about so we can go ahead and add the word to our stop words list.

In [36]:
```
stop_lst += ['link']
```

Now, we can take a look at 'RT' to confirm that it was used to refer to retweets.

In [37]:
```
context_finder('RT', corpus_pos)
```

```
RT @mention Great session by @mention on iPad design. Good UI insights. #tapworthy #HISX
SW #SXSW
RT @mention #SXSW sponsored by Apple, plaid, and boys I'm attracted to.
yup. RT @mention @mention at #sxsw: &amp;quot;apple comes up with cool technology no one's e
ver heard of because they don't go to conferences&amp;quot;
RT @mention Hey #SXSW- ers RT @mention RT @mention Last day to get the #LP Austin iPhone
app for free {link} #travel
RT @mention #SXSW hire qualified and talented women consciously early on in #startups to
achieve success - Google did!
RT @mention Preparing for #sxsw talk about #google doodles with @mention It will be fun!
Friday, 5pm: {link}
-----------------------------------
Out of 25 tweets analyzed, 6 tweets had the word "RT" in them.
```

We can see that in all cases 'RT', or 'rt' since our tokenizer changed it to lower case, was used to denote a retweet. Therefore, we can add it to our stop words list as well.

In [38]:
```
stop_lst += ['rt']
```

In [39]:
```
#updating the tokens list with updated stop words list
tokens_pos_ls = remove_stop(tokens_pos_ls, stop_list=stop_lst)
```

50 Most Frequent Words in Positive Tweets

One way to get some valuable insights about the tweets is to take a look at the most frequently occuring words in the tweets.

In [40]:
```
#defining a function to find the n most common tokens

def find_frequent(tokens, n=50):
    freq = FreqDist(tokens)
    display(freq.most_common(n))
```

In [41]:
```
#displaying 50 most frequent words
find_frequent(tokens_pos_ls)
```

```
[('ipad', 1010),
 ('apple', 711),
 ('google', 602),
 ('2', 595),
 ('store', 554),
 ('iphone', 466),
 ('app', 387),
 ('new', 358),
 ('austin', 250),
 ('get', 181),
 ('#apple', 174),
 ('launch', 173),
 ('android', 161),
 ('party', 151),
 ('pop-up', 151),
 ('line', 143),
 ('time', 136),
 ('great', 135),
 ('via', 132),
 ('#ipad2', 129),
 ('day', 124),
 ('social', 122),
 ('free', 120),
 ('cool', 119),
 ("i'm", 115),
 ('like', 115),
 ('map', 115),
 ('one', 114),
 ('win', 112),
 ('today', 111),
 ('ha', 108),
 ('circle', 107),
 ('w', 104),
 ('go', 104),
 ('come', 103),
 ('wa', 100),
 ('awesome', 93),
 ('#ipad', 93),
 ('love', 93),
 ('good', 92),
 ('network', 91),
 ('mobile', 90),
 ('temporary', 89),
 ('downtown', 88),
 ('opening', 88),
 ('people', 82),
 ('open', 82),
 ('#iphone', 82),
 ('got', 81),
 ('apps', 78)]
```

Perhaps, not surprisingly, we see that the most common words include 'apple' and 'google', but there are additional words such as 'circle', 'launch' and 'store' in here that show us more about what

the people were excited about. However, we are only looking at 50 words and there may be additional information here that we are missing. To address this, we can visualize this information in word clouds.

WordCloud with Product/Company Information for Positive Tweets

In [42]:
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

#defining a function for wordcloud generation
def generate_wordcloud(tokens, collocations=False, background_color='black', colormap='
    ## Initalize a WordCloud
    wordcloud = WordCloud(collocations=collocations, background_color=background_color,
                          width=500, height=300)

    ## Generate wordcloud from tokens
    wordcloud.generate(','.join(tokens))

    ## Plot with matplotlib
    if display:
        plt.figure(figsize = (12, 15), facecolor = None)
        plt.imshow(wordcloud)
        plt.axis('off');
    return wordcloud
```

We can start with generating a word cloud for the tokens we were looking at above.

In [43]:
```
#generating word cloud
cloud_pos_w_company = generate_wordcloud(tokens_pos_ls, collocations=True)
```

This is a great visualization to get a better sense of what the public was tweeting about in a positive way. We can see that a lot of people seem to be excited about iPads and the iPad 2 launch and

unsurprisingly, are tweeting about Apple and Google. We can dive even deeper and remove product/company names from our word cloud to see if we can find additional details.

WordCloud without Product/Company Information for Positive Tweets

In [44]:
```
#removing product/company names
stop_lst_no_comp = stop_lst + ['ipad', 'apple', 'google', 'iphone', '#apple',
                               '#google', '#ipad', '#iphone', 'android']
tokens_pos_ls = remove_stop(tokens_pos_ls, stop_list=stop_lst_no_comp)
```

In [45]:
```
cloud_pos_wo_company = generate_wordcloud(tokens_pos_ls, collections=True)
```

We can now see additional details about the tweets' contents. A lot of people were excited about a new apple store, perhaps some new apps and being in Austin.

Bigrams for Positive Tweets

Similar to the most common 50 words, looking at the most common bigrams can also be helpful in providing context to these tweets.

In [46]:
```
from nltk.collocations import *
bigram_measures = nltk.collocations.BigramAssocMeasures()
```

In [47]:
```
#initializing finder
finder_pos = BigramCollocationFinder.from_words(tokens_pos_ls)
```

In [48]:
```
#getting frequency information from finder
bigrams = finder_pos.score_ngrams(bigram_measures.raw_freq)
```

In [49]:
```
bigrams[:50]
```

Out[49]:
```
[(('pop-up', 'store'), 0.004340211680499505),
 (('social', 'network'), 0.0032741947765171707),
 (('temporary', 'store'), 0.00304576258280667),
 (('new', 'social'), 0.00293154648595142),
 (('store', 'downtown'), 0.0026650422599558366),
 (('downtown', 'austin'), 0.002626970227670753),
 (('2', 'launch'), 0.002208177872534836),
 (('called', 'circle'), 0.0021701058402497525),
 (('network', 'called'), 0.0021701058402497525),
 (('marissa', 'mayer'), 0.0021320338079646693),
 (('launch', 'major'), 0.002055889743394502),
 (('major', 'new'), 0.002055889743394502),
 (('popup', 'store'), 0.002055889743394502),
 (('store', 'austin'), 0.0019036016142541688),
 (('pop', 'store'), 0.0017513134851138354),
 (('austin', '2'), 0.0016370973882585853),
 (('opening', 'temporary'), 0.0016370973882585853),
 (('possibly', 'today'), 0.0015609533236884184),
 (('circle', 'possibly'), 0.001522881291403335),
 (('even', 'begin'), 0.001522881291403335),
 (('cool', 'technology'), 0.0012944490976928348),
 (('ever', 'heard'), 0.0012944490976928348),
 (("one's", 'ever'), 0.0012944490976928348),
 (('app', 'store'), 0.0012563770654077514),
 (('begin', 'win'), 0.001218305033122668),
 (('go', 'conference'), 0.001218305033122668),
 (('technology', "one's"), 0.001218305033122668),
 (('come', 'cool'), 0.0011802330008375847),
 (('heard', 'go'), 0.0011802330008375847),
 (('temp', 'store'), 0.0011421609685525014),
 (('pop-up', 'shop'), 0.0009518008071270844),
 (('set', 'open'), 0.000913728774842001),
 (('new', 'app'), 0.0008756567425569177),
 (('store', '2'), 0.0008756567425569177),
 (('marketing', 'expert'), 0.0008375847102718343),
 (('open', 'pop-up'), 0.0008375847102718343),
 (('2', 'take'), 0.0007995126779867509),
 (('giving', 'away'), 0.0007995126779867509),
 (('includes', 'uberguide'), 0.0007995126779867509),
 (('new', '#ubersocial'), 0.0007995126779867509),
 (('rumor', 'opening'), 0.0007995126779867509),
 (('school', 'marketing'), 0.0007995126779867509),
 (('win', '2'), 0.0007995126779867509),
 (('2', 'year'), 0.0007614406457016675),
 (('love', 'come'), 0.0007614406457016675),
 (('store', 'includes'), 0.0007614406457016675),
 (('#ubersocial', 'app'), 0.0007233686134165842),
 (('get', '2'), 0.0007233686134165842),
 (('opening', 'pop-up'), 0.0007233686134165842),
 (('popup', 'shop'), 0.0007233686134165842)]
```

Once again, we can see that a lot of people are excited about a new and temporary pop-up store downtown, ipad 2 launch, Google's Marissa Mayer and a new social network called circle.

Now that we have a preliminary understanding of the positive tweets, we can take a look at the negative tweets.

Tweets with Negative Sentiment

In [50]:
```
#parsing positive tweets into new df
df_negative = df[df['Emotion']=='Negative']
```

```
#verifying that neutral and negative tweets have been removed
df_negative['Emotion'].value_counts()
```

Out[50]:
```
Negative    569
Name: Emotion, dtype: int64
```

In [51]:
```
#parsing tweets into a list
corpus_neg = df_negative['Tweet'].to_list()
corpus_neg[:5]
```

Out[51]:
```
['.@wesley83 I have a 3G iPhone. After 3 hrs tweeting at #RISE_Austin, it was dead!  I n
eed to upgrade. Plugin stations at #SXSW.',
 "@sxsw I hope this year's festival isn't as crashy as this year's iPhone app. #sxsw",
 'I just noticed DST is coming this weekend. How many iPhone users will be an hour late
at SXSW come Sunday morning? #SXSW #iPhone',
 '@mention  - False Alarm: Google Circles Not Coming Now\x89ÛÒand Probably Not Ever? -
{link} #Google #Circles #Social #SXSW',
 'Again? RT @mention Line at the Apple store is insane.. #sxsw']
```

Tokenization/Lemmatization/Stop Word Removal

In [52]:
```
#tokenize tweets
tokens_neg = tokenize_tweets(corpus_neg)
#lemmatize tweets
tokens_neg_lemm = lemmatize_tokens(tokens_neg)
#remove stop words & punctuation using the same stop words list
tokens_neg_ls = remove_stop(tokens_neg_lemm, stop_list = stop_lst)
```

50 Most Frequent Words in Negative Tweets

In [53]:
```
find_frequent(tokens_neg_ls)
```

```
[('ipad', 179),
 ('iphone', 145),
 ('google', 136),
 ('apple', 100),
 ('2', 81),
 ('', 69),
 ('app', 60),
 ('store', 47),
 ('new', 43),
 ('like', 43),
 ('need', 35),
 ('ha', 31),
 ('circle', 29),
 ('design', 29),
 ('people', 29),
 ('social', 28),
 ('apps', 26),
 ('get', 25),
 ('wa', 24),
 ('austin', 23),
 ('think', 23),
 ('time', 23),
 ('launch', 22),
 ('one', 22),
 ('day', 21),
 ('today', 21),
 ('look', 21),
 ('line', 20),
 ('say', 20),
 ('android', 19),
```

```
('#ipad', 19),
('would', 19),
('network', 18),
('phone', 18),
('headache', 17),
('news', 17),
('go', 17),
('long', 17),
('product', 17),
("i've", 16),
("i'm", 16),
('battery', 16),
('user', 15),
('thing', 15),
('#apple', 15),
('good', 15),
('see', 15),
('much', 15),
('company', 15),
('america', 15)]
```

Once again, we see that Apple and Google are mentioned quite a few times in the negative tweets along with words such as 'design' and 'launch'. We can visualize and try to dig deeper.

WordCloud with Product/Company Information for Negative Tweets

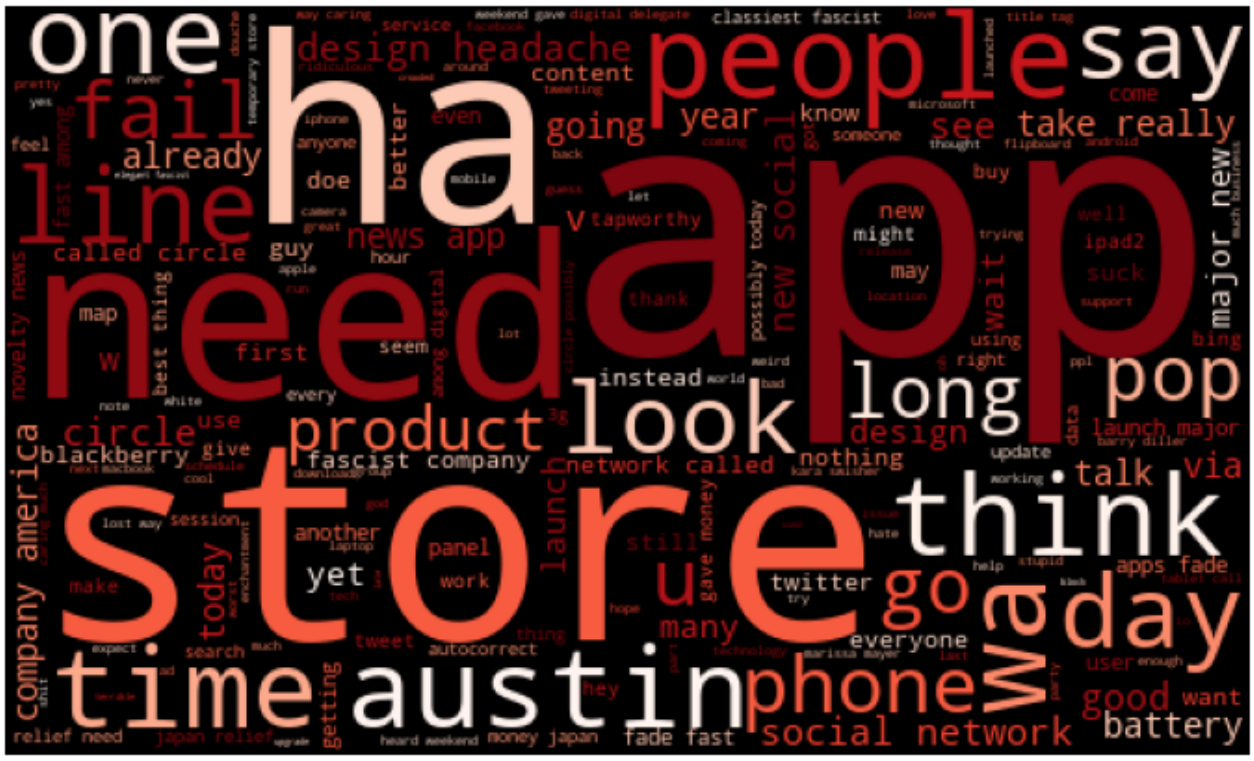
In [54]:
```
#generating word cloud for negative tweets
cloud_neg_w_company = generate_wordcloud(tokens_neg_ls, colormap='Reds', collocations=T
```

It looks like the iPad and the iPhone are mentioned often in the negative tweets along with their designs. We can remove the company names and product lines to see if we can see any additional information.

WordCloud without Product/Company Information for Negative

Tweets

In [55]:
```
#removing company/product information from the tokens
tokens_neg_ls = remove_stop(tokens_neg_ls, stop_list=stop_lst_no_comp)
#generating word cloud
cloud_neg_wo_company = generate_wordcloud(tokens_neg_ls, colormap='Reds', collocations=
```



We can see that most negative tweets were talking about stores and apps. Additionally, there are a lot of opinions being shared based on words like 'think' or 'fail'.

Bigrams for Negative Tweets

In [56]:
```
#initializing finder
finder_neg = BigramCollocationFinder.from_words(tokens_neg_ls)
#getting frequency information from finder
bigrams_neg = finder_neg.score_ngrams(bigram_measures.raw_freq)
#displaying top 50 bigrams
bigrams_neg[:50]
```

Out[56]:
```
[(('design', 'headache'), 0.003249235474006116),
 (('new', 'social'), 0.0030581039755351682),
 (('social', 'network'), 0.0028669724770642203),
 (('company', 'america'), 0.002484709480122324),
 (('fascist', 'company'), 0.0022935779816513763),
 (('major', 'new'), 0.0022935779816513763),
 (('network', 'called'), 0.002102446483180428),
 (('called', 'circle'), 0.00191131498470948),
 (('launch', 'major'), 0.00191131498470948),
 (('fade', 'fast'), 0.0017201834862385322),
 (('fast', 'among'), 0.0017201834862385322),
 (('news', 'apps'), 0.0017201834862385322),
 (('novelty', 'news'), 0.0017201834862385322),
 (('#japan', 'relief'), 0.0015290519877675841),
 (('2', 'money'), 0.0015290519877675841),
 (('best', 'thing'), 0.0015290519877675841),
 (('classiest', 'fascist'), 0.0015290519877675841),
```

```
(('gave', '2'), 0.0015290519877675841),
(("i've", 'heard'), 0.0015290519877675841),
(('need', '2'), 0.0015290519877675841),
(('relief', 'need'), 0.0015290519877675841),
(('thing', "i've"), 0.0015290519877675841),
(('among', 'digital'), 0.001337920489296636),
(('digital', 'delegate'), 0.001337920489296636),
(('kara', 'swisher'), 0.001337920489296636),
(('money', '#japan'), 0.001337920489296636),
(('pop-up', 'store'), 0.001337920489296636),
(('possibly', 'today'), 0.001337920489296636),
(('app', 'store'), 0.0011467889908256881),
(('apps', 'fade'), 0.0011467889908256881),
(('circle', 'possibly'), 0.0011467889908256881),
(('heard', 'weekend'), 0.0011467889908256881),
(('marissa', 'mayer'), 0.0011467889908256881),
(('store', 'austin'), 0.0011467889908256881),
(('weekend', 'gave'), 0.0011467889908256881),
(('barry', 'diller'), 0.00095565749235474),
(('caring', 'much'), 0.00095565749235474),
(('elegant', 'fascist'), 0.00095565749235474),
(('line', '2'), 0.00095565749235474),
(('look', 'like'), 0.00095565749235474),
(('lost', 'way'), 0.00095565749235474),
(('much', 'business'), 0.00095565749235474),
(('tablet', 'call'), 0.00095565749235474),
(('temporary', 'store'), 0.00095565749235474),
(('way', 'caring'), 0.00095565749235474),
(('2', 'take'), 0.0007645259938837921),
(('2011', 'novelty'), 0.0007645259938837921),
(('alarm', 'circle'), 0.0007645259938837921),
(('back', 'button'), 0.0007645259938837921),
(('battery', 'life'), 0.0007645259938837921)]
```

Looking at the bigrams once again, we can see that the new social network, Circle, is being mentioned negatively (while it was also mentioned positively before), and there seems to be some comments about the companies themselves.

Comparison of Positive & Negative Tweets

It is easier to compare the wordclouds we generated above when they are side-by-side.

In [57]:
```
def compare_wordclouds(wc1, wc2):
    fig, ax = plt.subplots(figsize=(30,20), ncols=2)
    ax[0].imshow(wc1)
    ax[0].set_xticks([])
    ax[0].set_yticks([])
    ax[1].imshow(wc2)
    ax[1].set_xticks([])
    ax[1].set_yticks([])
    plt.tight_layout();
```

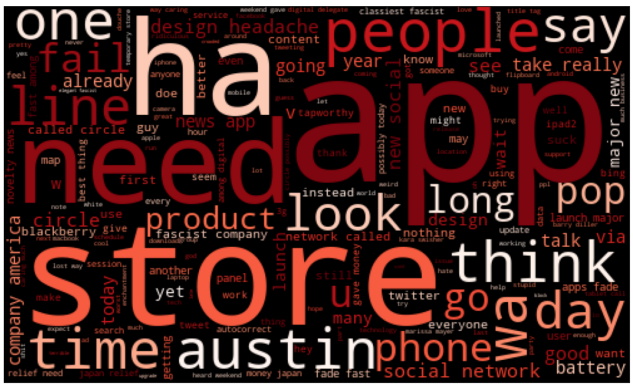
With Company/Product Information

In [58]:
```
compare_wordclouds(cloud_pos_w_company, cloud_neg_w_company)
```


As discussed above, we can see that both companies, Apple and Google as well as their products are mentioned extensively in both negative and positive tweets. Even though these wordclouds are a good starting point, it is difficult to extract any specific insights about the companies or products based on these.

Without Company/Product Information

In [59]:
```
compare_wordclouds(cloud_pos_wo_company, cloud_neg_wo_company)
```

Removing the company and product names help in seeing additional information, but since we have both companies represented in each group of tweets, it is not possible to tell which company's store is mentioned in a negative way for example. In order to address this we can make our analysis more granular and look at tweets not only based on their sentiment but also based on specific companies and specific products.

Tweets Related to Apple

Feature Engineering Company Column

In [60]:
```
df['Product/Brand'].unique()
```

Out[60]:
```
array(['iPhone', 'iPad or iPhone App', 'iPad', 'Google', 'Unknown',
       'Android', 'Apple', 'Android App',
       'Other Google product or service',
       'Other Apple product or service'], dtype=object)
```

In [61]:
```
brand_dict={'iPhone': 'Apple', 'iPad or iPhone App': 'Apple', 'iPad': 'Apple',
            'Google': 'Google', 'Unknown': 'Unknown',
            'Android': 'Google', 'Apple': 'Apple', 'Android App': 'Google',
            'Other Google product or service': 'Google',
```

```
            'Other Apple product or service': 'Apple'}
df['Brand'] = df['Product/Brand'].map(brand_dict)
df['Brand'].unique()
```

Out[61]:
```
array(['Apple', 'Google', 'Unknown'], dtype=object)
```

In [62]:
```
df_apple = df[df['Brand']=='Apple']
corpus_apple = df_apple['Tweet'].to_list()
```

In [63]:
```
#tokenize tweets
tokens_apple = tokenize_tweets(corpus_apple)
#lemmatize tweets
tokens_apple_lemm = lemmatize_tokens(tokens_apple)
#remove stop words & punctuation using the same stop words list
tokens_apple_ls = remove_stop(tokens_apple_lemm, stop_list = stop_lst)
```

Word Cloud

In [64]:
```
#generating word cloud for negative tweets
generate_wordcloud(tokens_apple_ls, colormap='Blues', collocations=True)
```

Out[64]:
```
<wordcloud.wordcloud.WordCloud at 0x2ad84de4d90>
```

In [65]:
```
tokens_apple_ls = remove_stop(tokens_apple_ls, stop_list = stop_lst_no_comp)
generate_wordcloud(tokens_apple_ls, colormap='Blues', collocations=True)
```

Out[65]:
```
<wordcloud.wordcloud.WordCloud at 0x2ad89139b50>
```

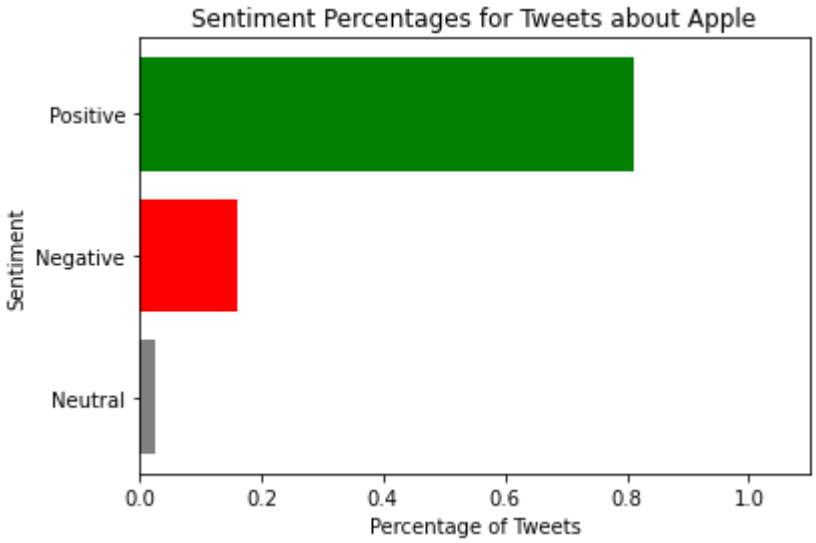
Percentage of Positive, Neutral and Negative Sentiment

In [66]:
```
df_apple_sentiment = pd.DataFrame(df_apple['Emotion'].value_counts(normalize=True)).res
df_apple_sentiment.columns=['Emotion', 'Percentage']
display(df_apple_sentiment)
df_apple_sentiment.sort_values('Percentage', ascending=True, inplace=True)
```

	Emotion	Percentage
0	Positive	0.811431
1	Negative	0.161452
2	Neutral	0.027117

In [67]:
```
fig, ax = plt.subplots()
ax.barh(df_apple_sentiment['Emotion'], df_apple_sentiment['Percentage'], color=['Grey',
ax.set_xlim(0, 1.1)
ax.set_xlabel('Percentage of Tweets')
ax.set_ylabel('Sentiment')
ax.set_title('Sentiment Percentages for Tweets about Apple')
```

Out[67]:
```
Text(0.5, 1.0, 'Sentiment Percentages for Tweets about Apple')
```

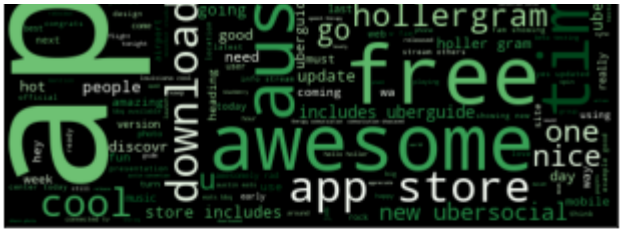


Sentiment Analysis on Apple Related Products/Services

In [68]:
```
def generate_product_wordcloud(company, emotion, stop_lst, colormap='Greens'):
    company_df = df[df['Brand']==company]
    final_df = company_df[company_df['Emotion']==emotion]
    product_lst = list(final_df['Product/Brand'].unique())

    wordclouds={}

    for product in product_lst:
        token_df = final_df[final_df['Product/Brand']==product]
        corpus = token_df['Tweet'].to_list()
        #tokenize tweets
        tokens_company = tokenize_tweets(corpus)
        #Lemmatize tweets
        tokens_company_lemm = lemmatize_tokens(tokens_company)
        #remove stop words & punctuation using the same stop words list
        tokens_company_ls = remove_stop(tokens_company_lemm, stop_list = stop_lst)
        wordclouds[product] = generate_wordcloud(tokens_company_ls, collocations=True,
                                                 colormap=colormap, display=False)

    fig, ax = plt.subplots(figsize=(12,15),
                           nrows=len(wordclouds))
    for i, product in enumerate(product_lst):
        ax[i].set_title(product)
        ax[i].set_xticks([])
        ax[i].set_yticks([])
        ax[i].imshow(wordclouds[product])
        plt.tight_layout();

    return wordclouds, fig
```

Positive Sentiment by Product/Service Type

In [69]:
```
wordcloud_apple_pos, fig_apple_pos = generate_product_wordcloud('Apple', 'Positive', st
```

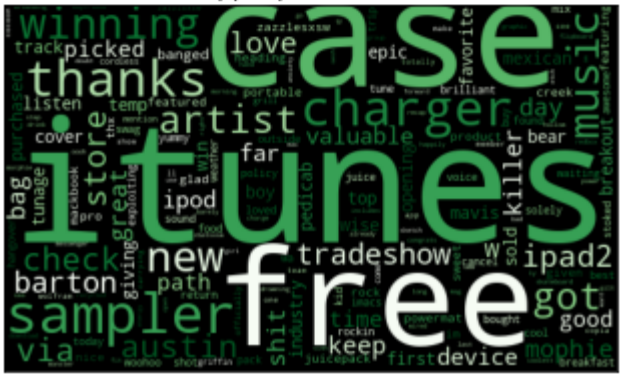


iPad

Apple

iPhone

Other Apple product or service



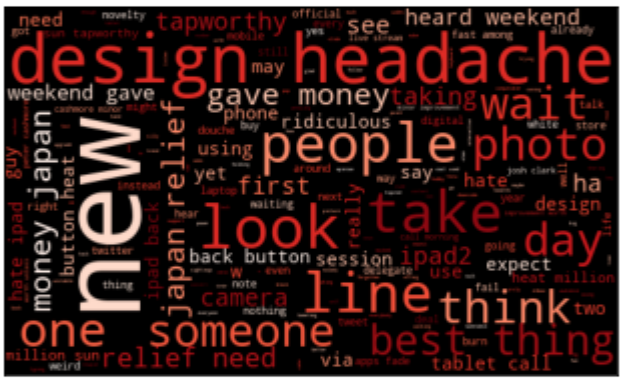
Negative Sentiment by Product/Service Type

In [70]:
```
wordcloud_apple_neg, fig_apple_neg = generate_product_wordcloud('Apple', 'Negative', st
```

iPhone

iPad or iPhone App

iPad



Apple

Other Apple product or service

Tweets Related to Google

In [71]:
```
df_google = df[df['Brand']=='Google']
corpus_google = df_google['Tweet'].to_list()
```

In [72]:
```
#tokenize tweets
tokens_google = tokenize_tweets(corpus_google)
#lemmatize tweets
tokens_google_lemm = lemmatize_tokens(tokens_google)
#remove stop words & punctuation using the same stop words list
tokens_google_ls = remove_stop(tokens_google_lemm, stop_list = stop_lst)
```

Word Cloud

In [73]:
```
#generating word cloud for negative tweets
generate_wordcloud(tokens_google_ls, colormap='Blues', collocations=True)
```

Out[73]: <wordcloud.wordcloud.WordCloud at 0x2ad84c90ee0>

In [74]:
```
tokens_google_ls = remove_stop(tokens_google_ls, stop_list = stop_lst_no_comp)
generate_wordcloud(tokens_google_ls, colormap='Blues', collocations=True)
```

Out[74]: <wordcloud.wordcloud.WordCloud at 0x2ad84f50730>

Percentage of Positive, Neutral and Negative Sentiment

In [75]:

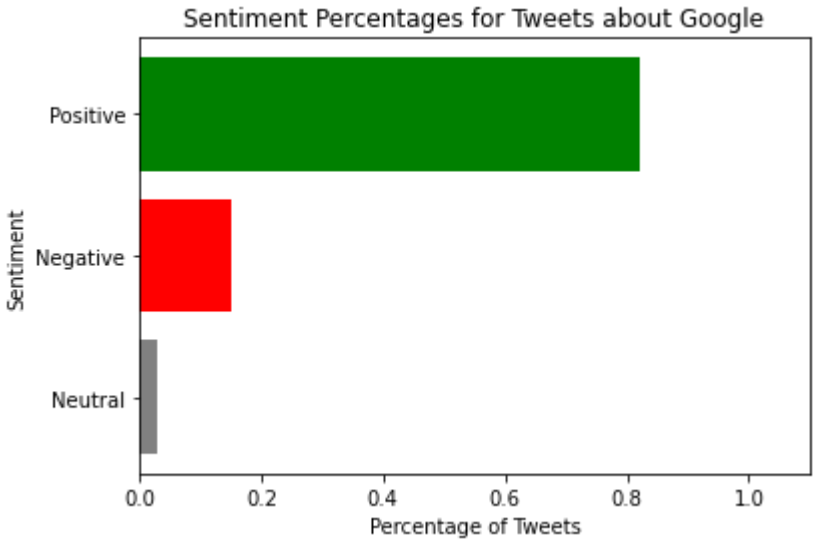
```
df_google_sentiment = pd.DataFrame(df_google['Emotion'].value_counts(normalize=True)).r
df_google_sentiment.columns=['Emotion', 'Percentage']
display(df_google_sentiment)
df_google_sentiment.sort_values('Percentage', ascending=True, inplace=True)
```

	Emotion	Percentage
0	Positive	0.820776
1	Negative	0.149543
2	Neutral	0.029680

In [76]:

```
fig, ax = plt.subplots()
ax.barh(df_google_sentiment['Emotion'], df_google_sentiment['Percentage'], color=['Grey
ax.set_xlim(0, 1.1)
ax.set_xlabel('Percentage of Tweets')
ax.set_ylabel('Sentiment')
ax.set_title('Sentiment Percentages for Tweets about Google')
```

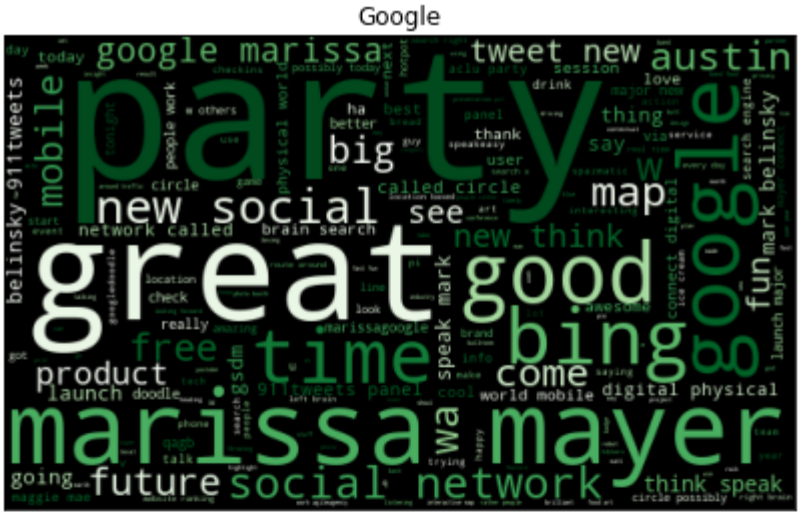
Out[76]: Text(0.5, 1.0, 'Sentiment Percentages for Tweets about Google')



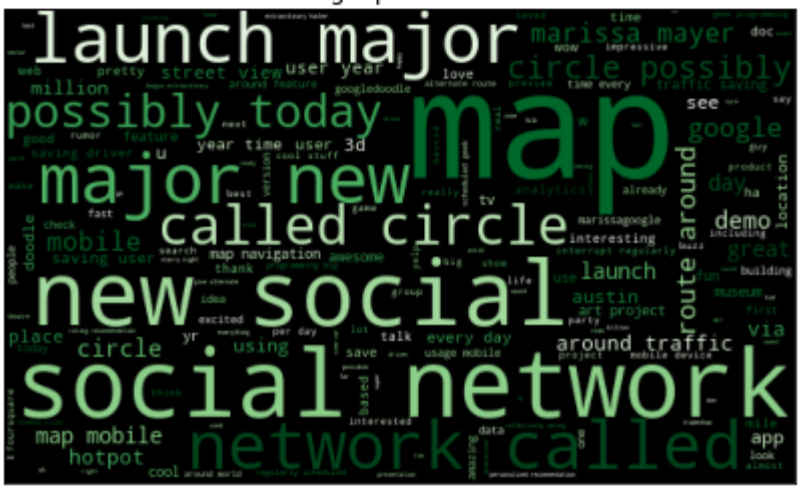
Sentiment Analysis on Google Related Products/Services

Positive Sentiment by Product/Service Type

In [77]:
```
wordcloud_google_pos, fig_google_pos = generate_product_wordcloud('Google', 'Positive',
```



Other Google product or service



Negative Sentiment by Product/Service Type

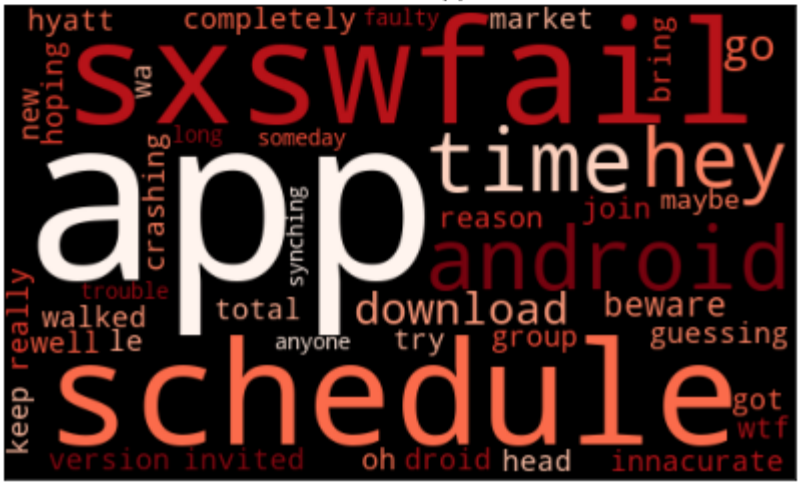
In [78]:
```
wordcloud_google_neg, fig_google_neg = generate_product_wordcloud('Google', 'Negative',
```

Google

Android

Other Google product or service

Android App



MODEL

Binary Classification

Data Preparation

In [79]:
```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVec
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegressionCV
```

Functionizing Preprocessing of Tweets

In [80]:
```
def preprocess_tweets(X, lemmatization=True, stop_list=stop_lst):
#     corpus = X.to_list()
    tokens = tokenize_tweets(X)#corpus)

    if lemmatization:
        tokens = lemmatize_tokens(tokens)

    tokens = remove_stop(tokens, stop_list=stop_list)
    return tokens
```

In [81]:
```
tokens = preprocess_tweets(df['Tweet'])
tokens[:5]
```

Out[81]:
```
['3g', 'iphone', '3', 'hr', 'tweeting']
```

Functionizing Evaluation of Classification Models

In [82]:
```
from sklearn.metrics import classification_report, plot_roc_curve, plot_confusion_matri

def classification(y_true, y_pred, X_test, X_train, clf):
    """This function shows the classification report,
    the confusion matrix as well as the ROC curve for evaluation of model quality.

    y_true: Correct y values, typically y_test that comes from the train_test_split per
    y_pred: Predicted y values by the model.
    clf: classifier model that was fit to training data.
    X_test: X_test values from train_test_split
    X_train: X_train values from train_test_split"""

    #training and test scores
    print(f"Training Score: {round(clf.score(X_train, y_train),2)} \
            Test Score:{round(clf.score(X_test, y_true),2)}")

    #Classification report
    print("\n")
    print("CLASSIFICATION REPORT")
    print("-----------------------------------------")
    print(classification_report(y_true=y_true, y_pred=y_pred))

    #Creating a figure/axes for confusion matrix and ROC curve
    fig, ax = plt.subplots(ncols=2, figsize=(12, 5))

    #Plotting the normalized confusion matrix
    plot_confusion_matrix(estimator=clf, X=X_test, y_true=y_true, cmap='Blues',
                          normalize='true', ax=ax[0], display_labels=['Negative', 'Posi

    #Plotting the ROC curve
    plot_roc_curve(estimator=clf, X=X_test, y=y_true, ax=ax[1])

    #Plotting the 50-50 guessing plot for reference
    ax[1].plot([0,1], [0,1], ls='--', color='orange')
```

Parsing Data to New Dataframe

Because we are going to be developing a model for binary classification, we need to remove neutral emotions from our tweets and binarize our target column which is the 'Emotion' column.

In [83]:
```
#removing neutral tweets for binary classification
df_binary = df[df['Emotion']!='Neutral']
```

```
#verifying that neutral tweets have been removed
df_binary['Emotion'].unique()
```

Out[83]:
```
array(['Negative', 'Positive'], dtype=object)
```

In [84]:
```
#binarizing emotion column
emotion_dict = {'Negative': 0, 'Positive': 1}
df_binary['Emotion'] = df_binary['Emotion'].map(emotion_dict)
```

```
<ipython-input-84-c21124fb8c6a>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_binary['Emotion'] = df_binary['Emotion'].map(emotion_dict)
```

train_test_split of Dataset

In [85]:
```
#train_test_split the dataset
y = df_binary['Emotion']
X = df_binary['Tweet']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

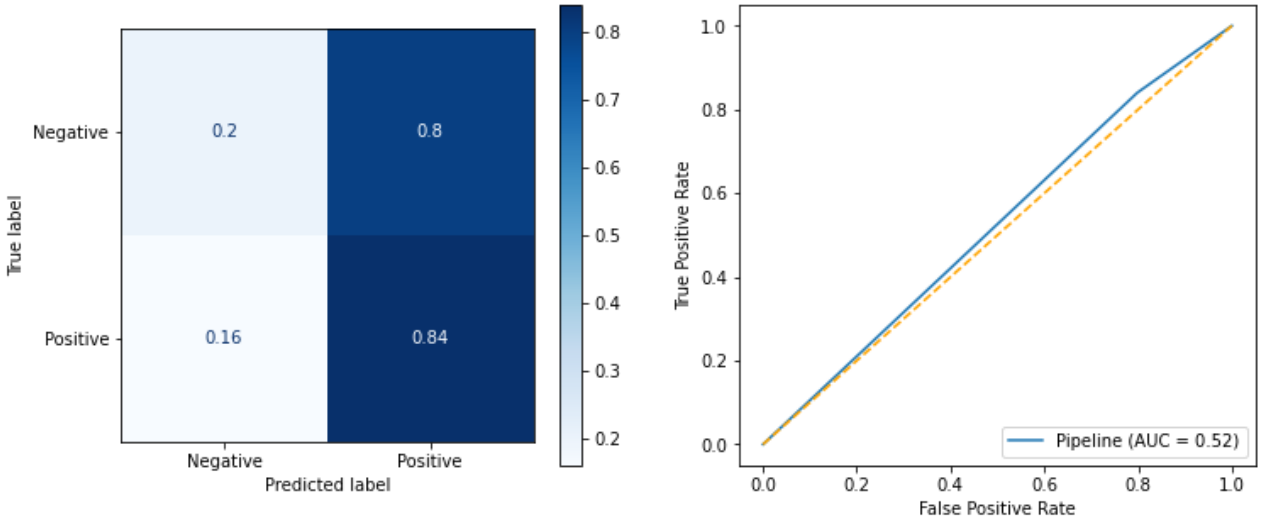
Modelling

Dummy Classifier - Baseline

In [86]:
```
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True)
```

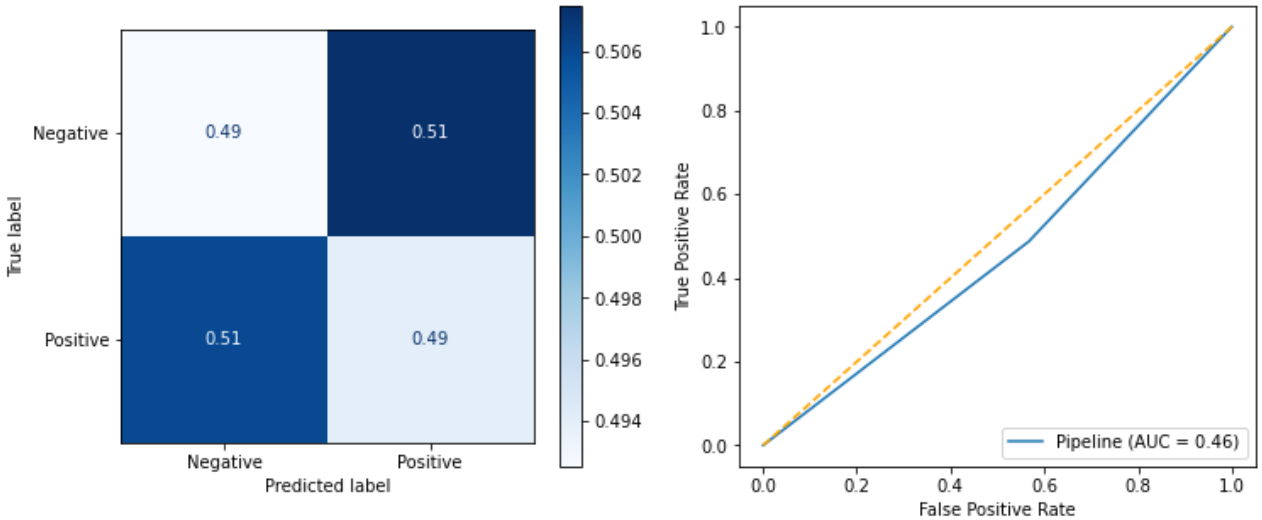
In [87]:
```
from sklearn.dummy import DummyClassifier

clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', DummyClassifier(random_state=42))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\dummy.py:131: FutureWa
rning: The default value of strategy will change from stratified to prior in 0.24.
  warnings.warn("The default value of strategy will change from "
Training Score: 0.73             Test Score:0.74


CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.18      0.20      0.19       134
           1       0.86      0.84      0.85       751

    accuracy                           0.74       885
   macro avg       0.52      0.52      0.52       885
weighted avg       0.75      0.74      0.75       885
```



In [88]:
```
from imblearn.over_sampling import RandomOverSampler
import imblearn.pipeline

clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                    ('os', RandomOverSampler(random_state=42)),
                    ('clf', DummyClassifier())])
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\dummy.py:131: FutureWa
rning: The default value of strategy will change from stratified to prior in 0.24.
  warnings.warn("The default value of strategy will change from "
Training Score: 0.49             Test Score:0.51


CLASSIFICATION REPORT
--------------------------------------------
              precision    recall  f1-score   support

           0       0.14      0.43      0.21       134
           1       0.83      0.52      0.64       751

    accuracy                           0.50       885
   macro avg       0.49      0.47      0.42       885
weighted avg       0.73      0.50      0.57       885
```



Multinomial Bayes

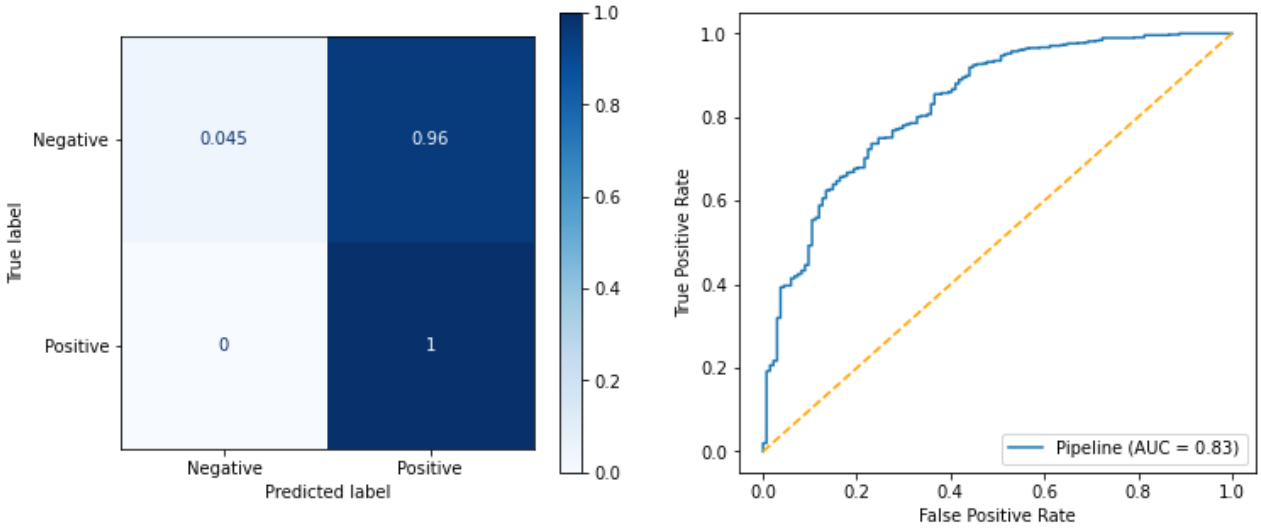
Pipelines for Preprocessing and Modelling

In [89]:
```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize, stop_
                     ('clf', MultinomialNB())])
```

MNB Model #1

In [90]:
```
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.86            Test Score:0.86


CLASSIFICATION REPORT
---------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.04      0.09       134
           1       0.85      1.00      0.92       751

    accuracy                           0.86       885
   macro avg       0.93      0.52      0.50       885
weighted avg       0.88      0.86      0.79       885
```



It looks like our model is predicting almost all predictions as positive which may be caused by a class imbalance issue. We can check the value_counts of our target column to see if there is an imbalance.

In [91]:
```
y.value_counts(normalize=True)
```

Out[91]:
```
1    0.83922
0    0.16078
Name: Emotion, dtype: float64
```

Not surprisingly, there is a class imbalance where 84% of the tweets are considered positive, and 16% are considered negative. Before we employ any oversampling or undersampling to address this class imbalance we can try to tune the model a bit and see if it improves the performance at all. Since Multinomial Naive Bayes is not a very complex model, the parameters that we can tune are

fairly limited. We will be trying out different values for the alpha values (learning rate) in this case. Additionally, we are optimizing the model for f1 macro average since we would like the overall quality of our model to be the best rather than focusing on correctly identifying a negative or positive value like in the case of optimizing for recall.

Hyperparameter Tuning with GridSearchCV

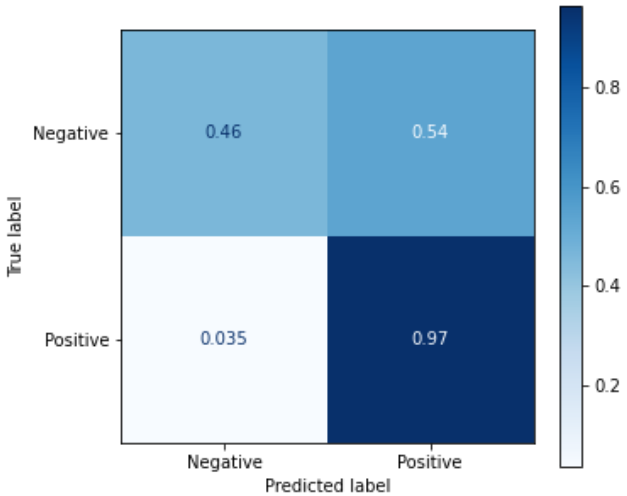
In [92]:
```
if run_gridsearches == True:
    param_grid = {'clf__alpha': [0.001, 0.01, 0.1, 1],
                  'vectorizer__stop_words': [stop_lst, 'english']}
    gridsearch = GridSearchCV(clf_pipe, param_grid=param_grid, scoring='recall_macro')

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

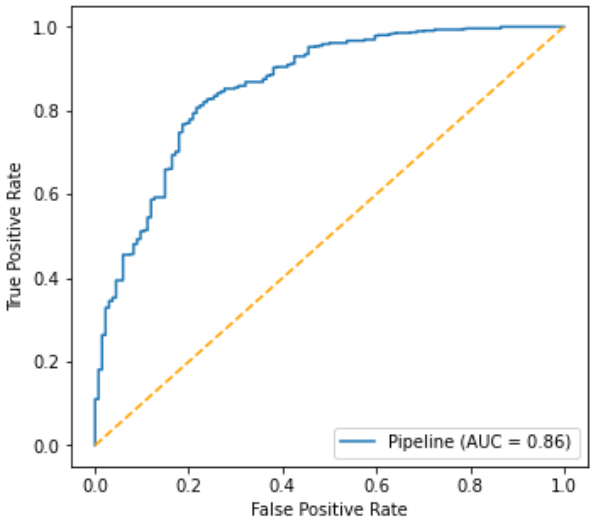
It looks like the stop_lst that we established performed better than the default 'english' stop words list and the learning rate of 0.1 was the best value amongst all the values we tried. Let's try these parameters in a new model.

Tuned MNB Model #1

In [93]:
```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize, stop_
                    ('clf', MultinomialNB(alpha=0.01))])
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.98            Test Score:0.89


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.70      0.46      0.56       134
           1       0.91      0.97      0.94       751

    accuracy                           0.89       885
   macro avg       0.81      0.71      0.75       885
weighted avg       0.88      0.89      0.88       885
```





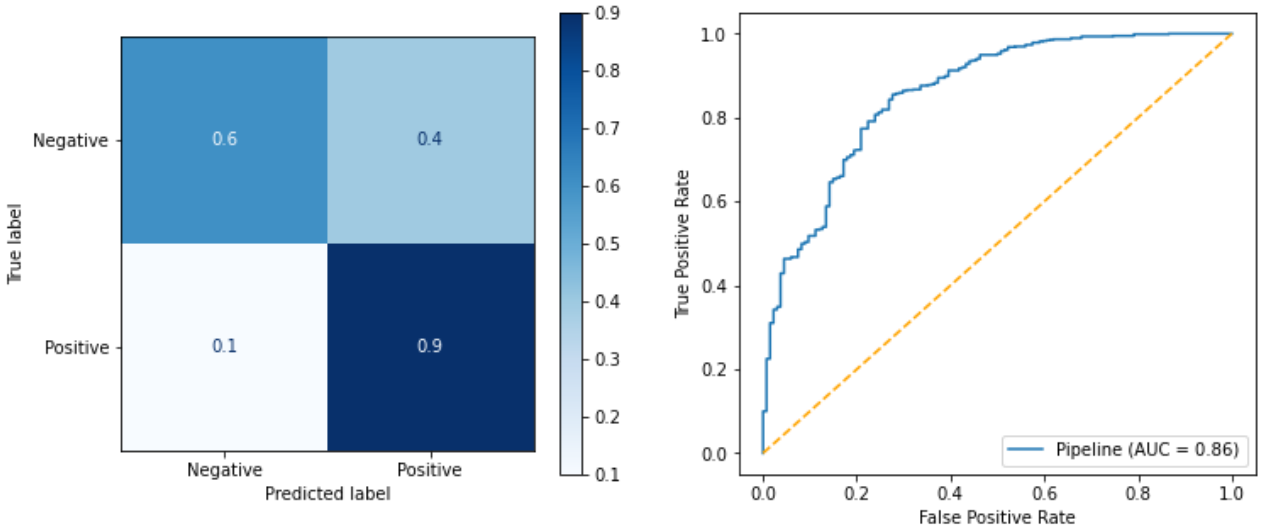
Our model's predictions for negative tweets improved from 0% to 46% with this gridsearch. Our

AUC score also increased to 0.86 from 0.83. The results are still not ideal since the performance of the model for True Negative values is still fairly low. We can try Random Over Sampling to address the imbalance between positive and negative tweets and see if it helps.

MNB Model #2 - Random Over Sampling

In [94]:
```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                    ('os', RandomOverSampler(random_state=42)),
                    ('clf', MultinomialNB(alpha=0.1))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.98            Test Score:0.86


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.52      0.60      0.56       134
           1       0.93      0.90      0.91       751

    accuracy                           0.86       885
   macro avg       0.72      0.75      0.74       885
weighted avg       0.87      0.86      0.86       885
```



Even though the model's performance on correctly identifying positive tweets has gone down from 98% to 90%, the model is performing much better (60% compared to 38%) in identifying negative tweets now that the data has been oversampled. We can also try to tune this model which uses the oversampled data.

Tuning Random Oversampled MNB Model #2

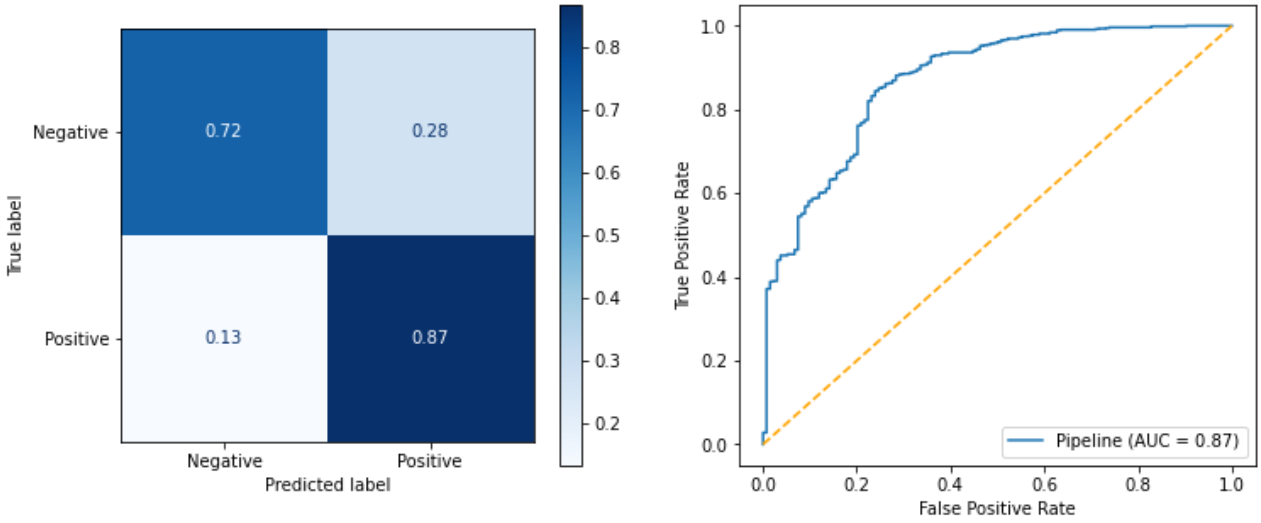
In [95]:
```
if run_gridsearches == True:
    param_grid = {'clf__alpha': [0.001, 0.01, 0.1, 1],
                  'vectorizer__stop_words': [stop_lst, 'english']}
    gridsearch = GridSearchCV(clf_pipe, param_grid=param_grid, scoring='recall_macro')

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

We can see that the alpha value of 1 and our custom stop_lst were the best parameters for optimizing the recall macro average.

Tuned MNB Model #2

In [96]:
```
from imblearn.over_sampling import RandomOverSampler
import imblearn.pipeline

clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                                                                      stop_words=stop_ls
                                       ('os', RandomOverSampler()),
                                       ('clf', MultinomialNB(alpha=1))])
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.95            Test Score:0.85


CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.49      0.72      0.59       134
           1       0.95      0.87      0.90       751

    accuracy                           0.85       885
   macro avg       0.72      0.80      0.75       885
weighted avg       0.88      0.85      0.86       885
```



With the tuned parameters, the model's performance in identifying negative tweets has increased to 72% from the prior 60%. With this adjustment however, we are compromising on correctly identifying the positive tweets at 86% compared to the 90% prior. Since we ran our gridsearch to optimize for the recall macro score this behavior is expected.

Now that we have tried several options for the Multinomial Naive Bayes model, we can try and evaluate a more complex model such as the Random Forest Classifier.

Random Forest

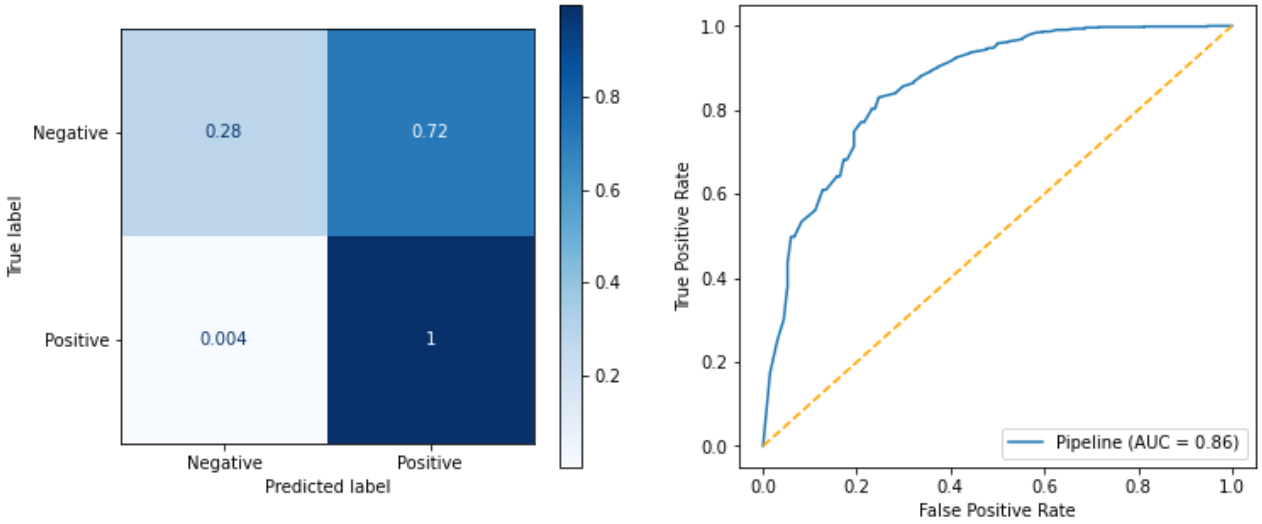
Since Random Forest has a class_weight parameter that we can specify to try to address the class

imbalance issue without having to oversample, we will try this option in our base model.

RF Model #1

In [97]:
```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', RandomForestClassifier(class_weight='balanced',
                                                    random_state=42))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 1.0             Test Score:0.89


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.93      0.28      0.43       134
           1       0.89      1.00      0.94       751

    accuracy                           0.89       885
   macro avg       0.91      0.64      0.69       885
weighted avg       0.89      0.89      0.86       885
```



We can see that the Random Forest model is performing similarly to the initial Multinomial Naive Bayes model: the performance of identifying negative tweets is fairly low at 28% while the positive tweet identification is 100%. Additionally, based on the training and test scores, it seems like the model is overfit. Next, we will try to tune the model.

Hyperparameter Tuning with GridSearchCV

In [98]:
```
if run_gridsearches == True:
    param_grid = {'clf__criterion': ['gini', 'entropy'],
            'clf__max_depth': [10, 20, None],
            'clf__min_samples_leaf': [1, 2, 3]
           }

    gridsearch = GridSearchCV(estimator=clf_pipe, param_grid = param_grid, scoring='rec
```

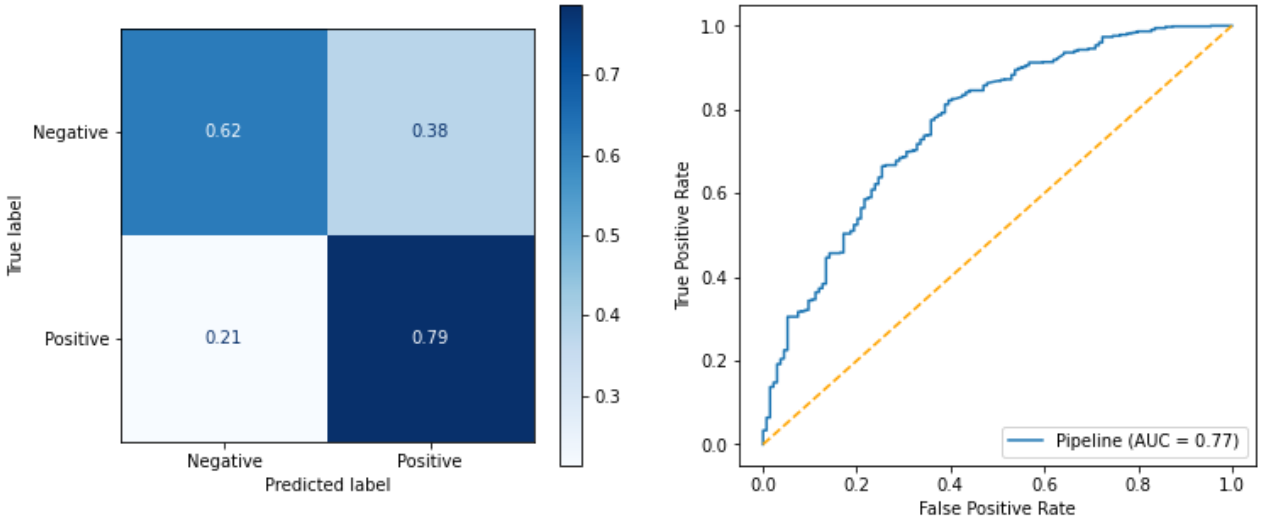
```
    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

Now that we have the best parameters for optimizing the recall macro score, we can go ahead and see how the model performs with these.

Tuned RF Model #1

In [99]:
```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', RandomForestClassifier(class_weight='balanced',
                                                    random_state=42,
                                                    criterion='gini', max_depth=10,
                                                    min_samples_leaf=3))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.78            Test Score:0.76


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.34      0.62      0.44       134
           1       0.92      0.79      0.85       751

    accuracy                           0.76       885
   macro avg       0.63      0.70      0.64       885
weighted avg       0.83      0.76      0.79       885
```



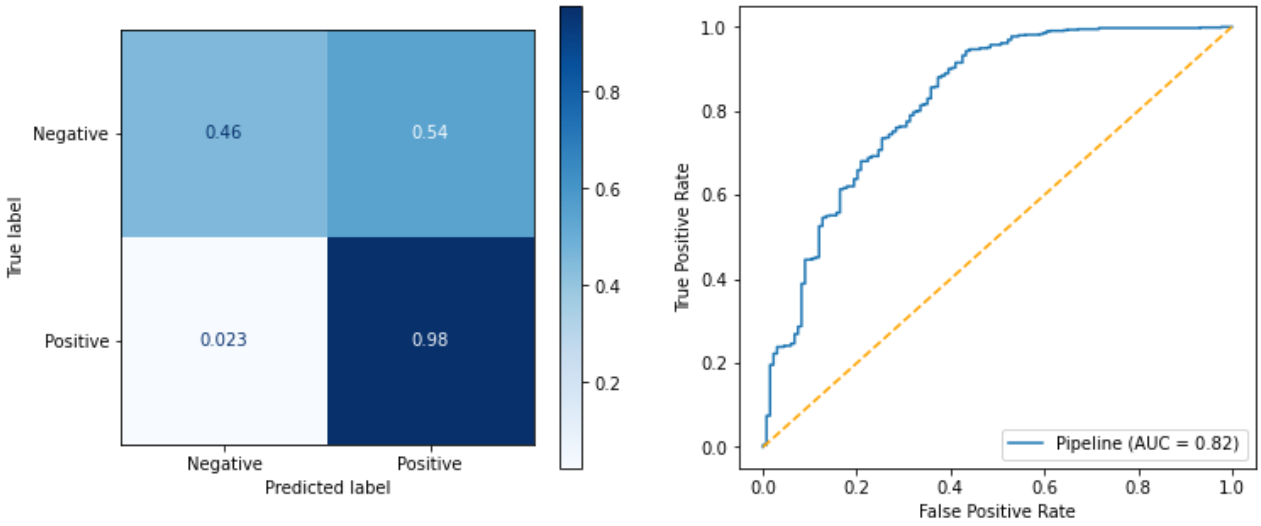
After tuning the model, we can see that the model correctly identified 62% of the negative tweets which is a 34% improvement on the prior model while positive tweet identification went down to 79% compared to 100%. We can once again try to oversample the data and see if the performance of our model improves.

RF Model #2 - Random Over Sampling

In [100…
```
from imblearn.over_sampling import RandomOverSampler
import imblearn.pipeline
```

```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                    ('os', RandomOverSampler(random_state=42)),
                    ('clf', RandomForestClassifier(class_weight='balanced',
                                                   random_state=42,
                                                   criterion='entropy', max_depth=None
                                                   min_samples_leaf=2))])
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.99            Test Score:0.9


CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.78      0.46      0.58       134
           1       0.91      0.98      0.94       751

    accuracy                           0.90       885
   macro avg       0.85      0.72      0.76       885
weighted avg       0.89      0.90      0.89       885
```



The random oversampled model seems to perform worse in identifying negative tweet at 46% while it is better at identifying positive tweets at 98%. We can once again try to tune the random oversampled model to see if we can improve on these values.

Tuning Random Oversampled RF Model #2

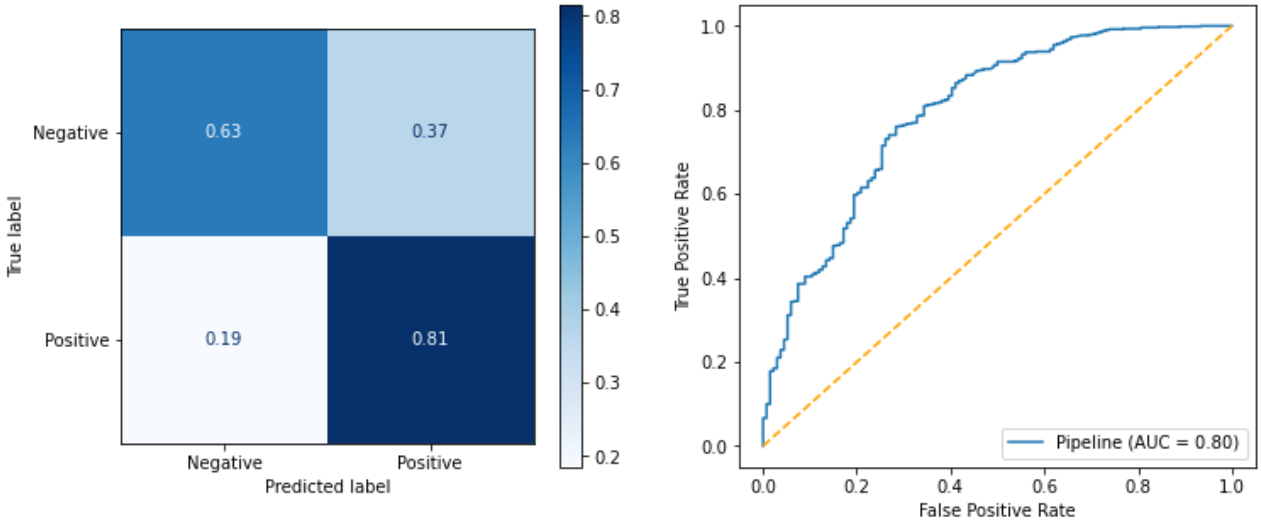
In [101…

```
if run_gridsearches == True:
    param_grid = {'clf__criterion': ['gini', 'entropy'],
            'clf__max_depth': [10, 20, None],
            'clf__min_samples_leaf': [1, 2, 3],
            'vectorizer__norm': ['l1', 'l2']}

    gridsearch = GridSearchCV(estimator=clf_pipe, param_grid = param_grid,
                              scoring='recall_macro', n_jobs=2)

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

Tuned RF Model #2

In [102…
```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                                                                      stop_words=stop_l
                                                                      norm='l2')),
                                       ('os', RandomOverSampler(random_state=42)),
                                       ('clf', RandomForestClassifier(class_weight='bal
                                                   random_state=42,
                                                   criterion='gini', max_depth=20,
                                                   min_samples_leaf=1))])
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.86            Test Score:0.79


CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.38      0.63      0.47       134
           1       0.93      0.81      0.87       751

    accuracy                           0.79       885
   macro avg       0.65      0.72      0.67       885
weighted avg       0.84      0.79      0.81       885
```



This is definitely the best model out of all the Random Forest models with 63% correct identification of negative tweets, and 81% correct identification of positive tweets. Lastly, we can try out logistic regression models for the task at hand to see if we can improve on the results we got from the Multinomial Naive Bayes and Random Forest models.

Logistic Regression

Similar to the Random Forest models, Logistic Regression takes in an optional class_weight argument. Since we know that we have a class imbalance, we will once again try the model with class_weight='balanced' to begin with.

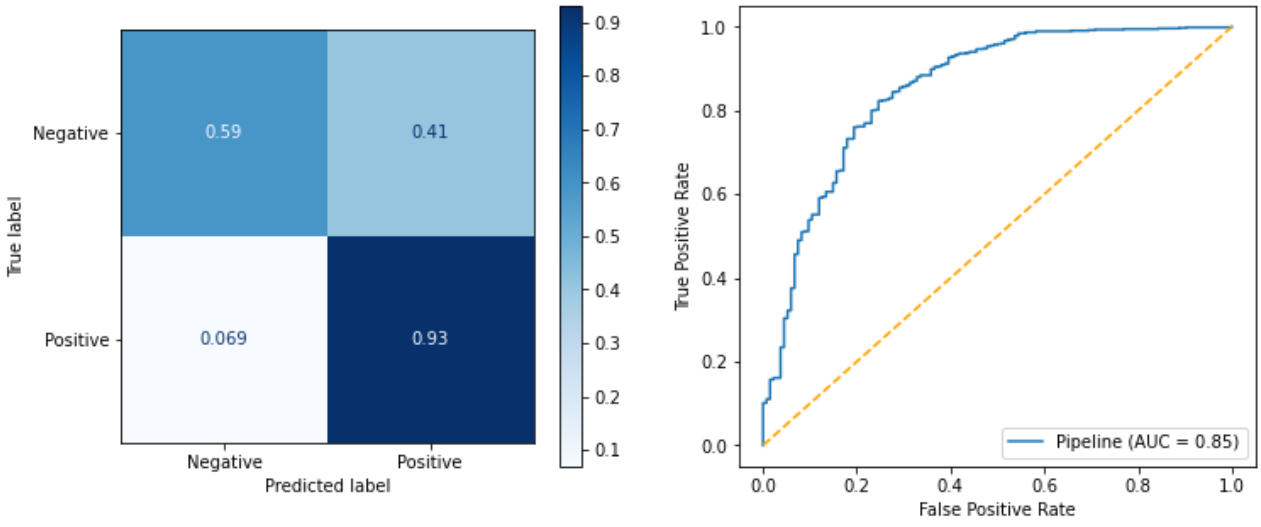
LR Model #1

In [103…
```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', LogisticRegressionCV(class_weight='balanced',
                                                  random_state=42, cv=3))])
```

```
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Training Score: 1.0             Test Score:0.88


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.60      0.59      0.60       134
           1       0.93      0.93      0.93       751

    accuracy                           0.88       885
   macro avg       0.77      0.76      0.76       885
weighted avg       0.88      0.88      0.88       885
```



The model correctly identified 59% of the negative and 93% of the positive tweets. However, we are getting some warnings stating that our model's maximum iteration value was too low and that the model was unable to converge. We can add this parameter as well as other parameters to tune our model and try to get better performance.

Hyperparameter Tuning with GridSearchCV

In [104…

```
if run_gridsearches == True:
```

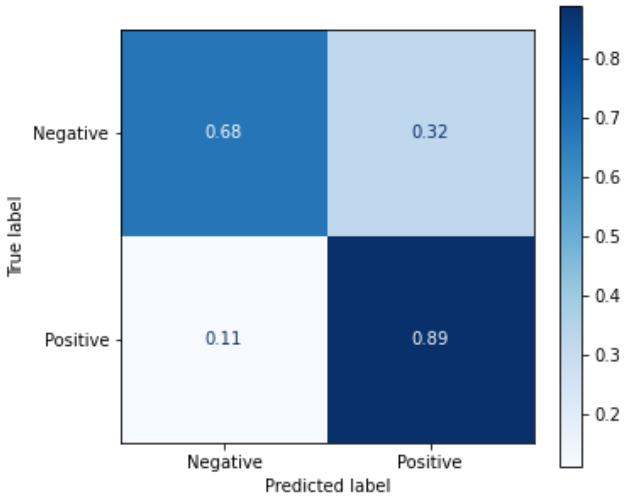
```
    param_grid = {'clf__class_weight': ['balanced'],
                  'clf__max_iter': [100, 500, 1000],
                  'clf__Cs': [[0.01], [0.1], [1]],
                  'clf__solver': ['liblinear', 'lbfgs', 'sag']}

    gridsearch = GridSearchCV(estimator=clf_pipe, param_grid = param_grid, scoring='rec

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

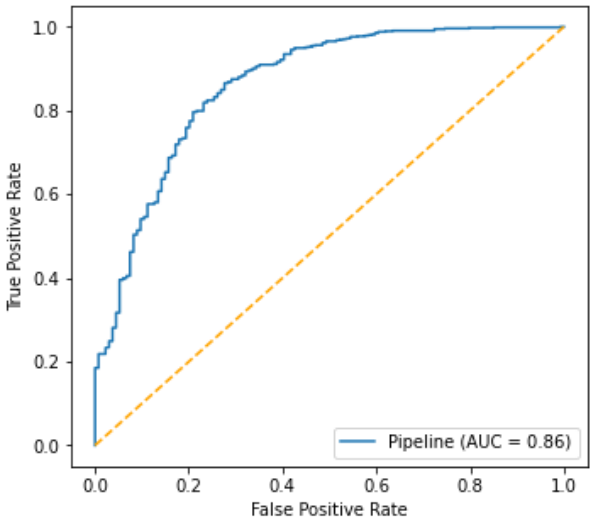
Tuned LR Model #1

In [105…

```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', LogisticRegressionCV(class_weight='balanced',
                                                  random_state=42, cv=3,
                                                  Cs=[1], max_iter=100, solver='libline

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.95            Test Score:0.86


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.52      0.68      0.59       134
           1       0.94      0.89      0.91       751

    accuracy                           0.86       885
   macro avg       0.73      0.78      0.75       885
weighted avg       0.88      0.86      0.86       885
```





LR Model #2 - Random Oversampling

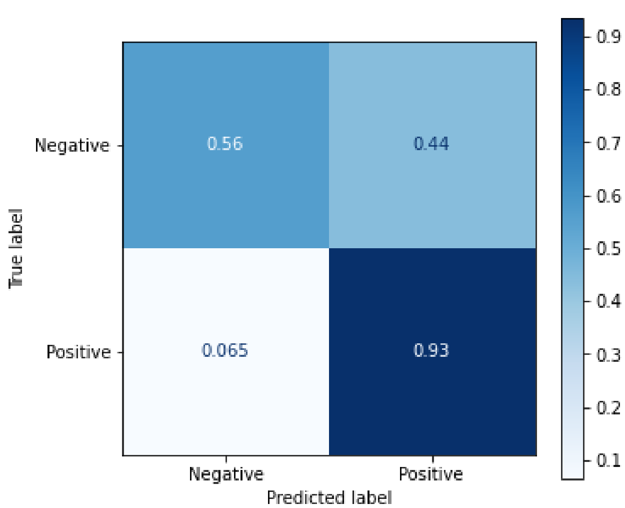
In [106…

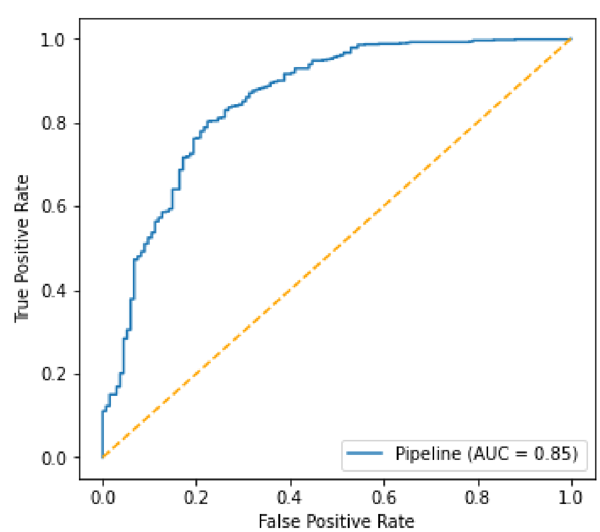
```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                                                                      stop_words=stop_l
                                                                      norm='l2')),
                                       ('os', RandomOverSampler(random_state=42)),
                                       ('clf', LogisticRegressionCV(class_weight='balan
                                                                    random_state=42, cv
```

```
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Training Score: 1.0             Test Score:0.88


CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.60      0.56      0.58       134
           1       0.92      0.93      0.93       751

    accuracy                           0.88       885
   macro avg       0.76      0.75      0.75       885
weighted avg       0.87      0.88      0.88       885
```





Tuning Random Oversampled LR Model #2

In [107…
```
if run_gridsearches == True:
```

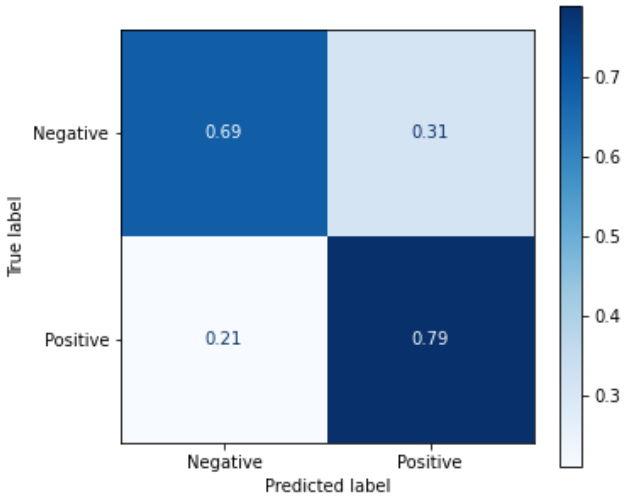
```
    param_grid = {'clf__max_iter': [100, 500, 1000],
                  'clf__Cs': [[0.01], [0.1], [1]],
                  'clf__solver': ['liblinear', 'lbfgs', 'sag']}

    gridsearch = GridSearchCV(estimator=clf_pipe, param_grid = param_grid, scoring='rec

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

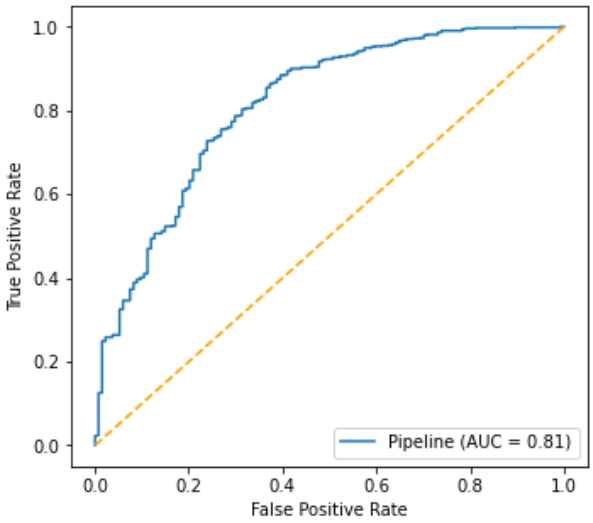
Tuned LR Model #2

In [108…

```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                                                                       stop_words=stop_l
                                                                       norm='l2')),
                                       ('os', RandomOverSampler(random_state=42)),
                                       ('clf', LogisticRegressionCV(class_weight='balan
                                                                    random_state=42, cv
                                                                    Cs=1, solver='libli

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
classification(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.83            Test Score:0.77


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.37      0.69      0.48       134
           1       0.93      0.79      0.85       751

    accuracy                           0.77       885
   macro avg       0.65      0.74      0.67       885
weighted avg       0.85      0.77      0.80       885
```





Multiclass Classification

Data Preparation

In [109…

```
df['Emotion'].unique()
```

Out[109… array(['Negative', 'Positive', 'Neutral'], dtype=object)

In [110…
```
#binarizing emotion column
emotion_dict = {'Negative': 0, 'Neutral':1, 'Positive': 2}
df['Emotion'] = df['Emotion'].map(emotion_dict)
df['Emotion'].value_counts()
```

Out[110…
```
1    5375
2    2970
0     569
Name: Emotion, dtype: int64
```

In [111…
```
y= df['Emotion']
X= df['Tweet']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [112…
```
y_train.value_counts()
```

Out[112…
```
1    4040
2    2231
0     414
Name: Emotion, dtype: int64
```

In [113…
```
def multiclass_eval(y_true, y_pred, X_test, X_train, clf, n_class=3):
    """This function shows the classification report,
    the confusion matrix as well as the ROC curve for evaluation of model quality.

    y_true: Correct y values, typically y_test that comes from the train_test_split per
    y_pred: Predicted y values by the model.
    clf: classifier model that was fit to training data.
    X_test: X_test values from train_test_split
    X_train: X_train values from train_test_split"""

    #training and test scores
    print(f"Training Score: {round(clf.score(X_train, y_train),2)} \
            Test Score:{round(clf.score(X_test, y_true),2)}")

    #Classification report
    print("\n")
    print("CLASSIFICATION REPORT")
    print("-----------------------------------------")
    print(classification_report(y_true=y_true, y_pred=y_pred))

    #Creating a figure/axes for confusion matrix and ROC curve
    fig, ax = plt.subplots(ncols=2, figsize=(12, 5))

    #Plotting the normalized confusion matrix
    plot_confusion_matrix(estimator=clf, X=X_test, y_true=y_true, cmap='Greens',
                          normalize='true', ax=ax[0], display_labels=['Negative', 'Neut

    pred_prob = clf.predict_proba(X_test)

    #Plotting the ROC curve
    fpr={}
    tpr={}
    thresh={}

    for i in range(n_class):
        fpr[i], tpr[i], thresh[i] = roc_curve(y_true, pred_prob[:,i], pos_label=i)
```

```
    ax[1].plot(fpr[0], tpr[0], linestyle='--',color='red', label='Negative vs Rest')
    ax[1].plot(fpr[1], tpr[1], linestyle='--',color='blue', label='Neutral vs Rest')
    ax[1].plot(fpr[2], tpr[2], linestyle='--',color='green', label='Positive vs Rest')
    ax[1].set_title('Multiclass ROC curve')
    ax[1].set_xlabel('False Positive Rate')
    ax[1].set_ylabel('True Positive rate')
    ax[1].legend(loc='best')

    #Plotting the 50-50 guessing plot for reference
    ax[1].plot([0,1], [0,1], ls='--', color='orange')
```

Modelling

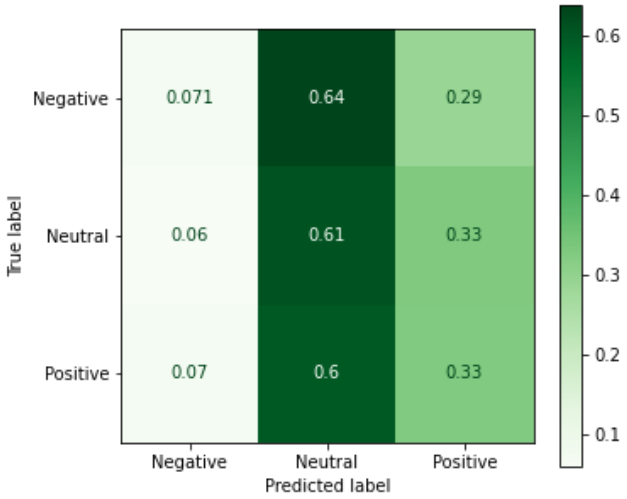
Dummy Classifier - Baseline

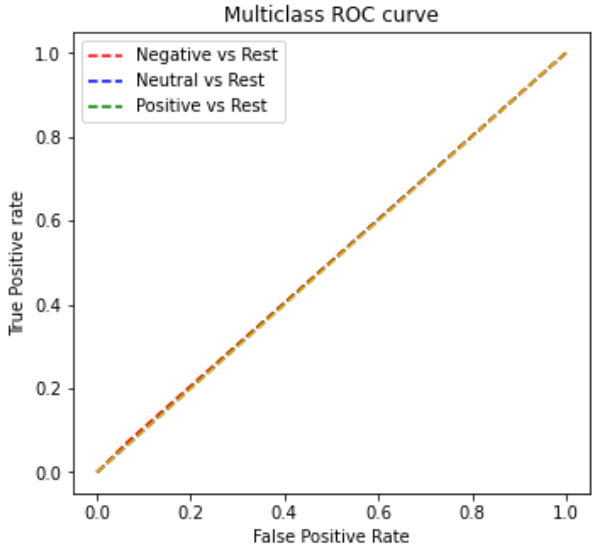
Vanilla Dummy Model

In [114…

```
from sklearn.dummy import DummyClassifier

clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', DummyClassifier(random_state=42))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\dummy.py:131: FutureWa
rning: The default value of strategy will change from stratified to prior in 0.24.
  warnings.warn("The default value of strategy will change from "
Training Score: 0.48             Test Score:0.48


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.08      0.07      0.07       155
           1       0.60      0.61      0.60      1335
           2       0.33      0.33      0.33       739

    accuracy                           0.48      2229
   macro avg       0.34      0.34      0.34      2229
weighted avg       0.48      0.48      0.48      2229
```

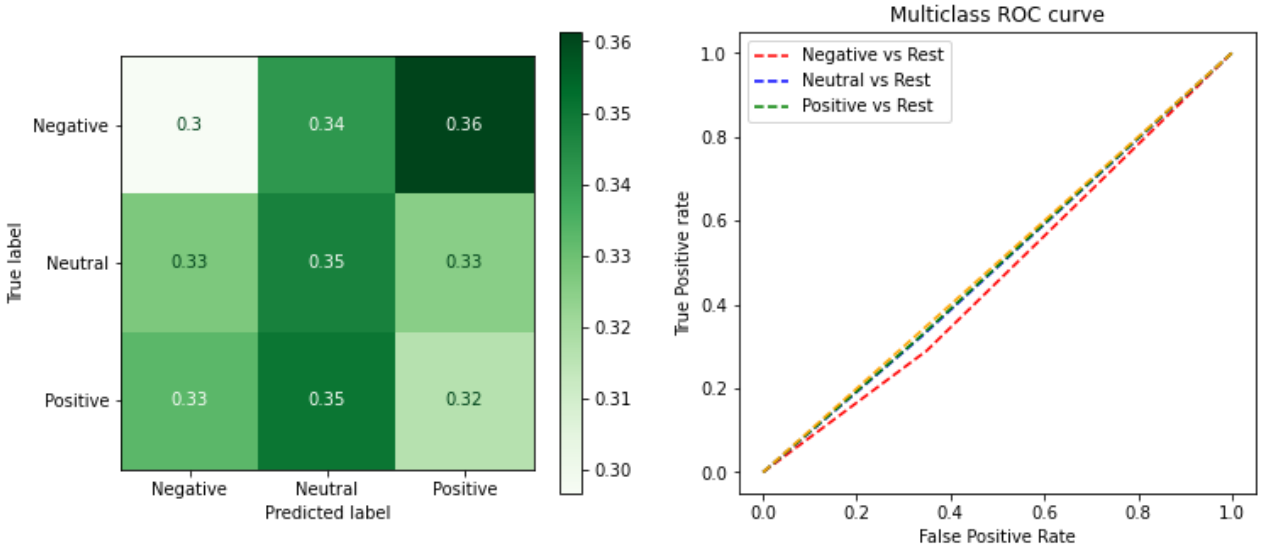




Random Oversampled Dummy Model

In [115…
```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                    ('os', RandomOverSampler(random_state=42)),
                    ('clf', DummyClassifier())])
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\dummy.py:131: FutureWa
rning: The default value of strategy will change from stratified to prior in 0.24.
  warnings.warn("The default value of strategy will change from "
Training Score: 0.33             Test Score:0.34


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.06      0.27      0.09       155
           1       0.59      0.32      0.42      1335
           2       0.33      0.34      0.34       739

    accuracy                           0.33      2229
   macro avg       0.33      0.31      0.28      2229
weighted avg       0.47      0.33      0.37      2229
```



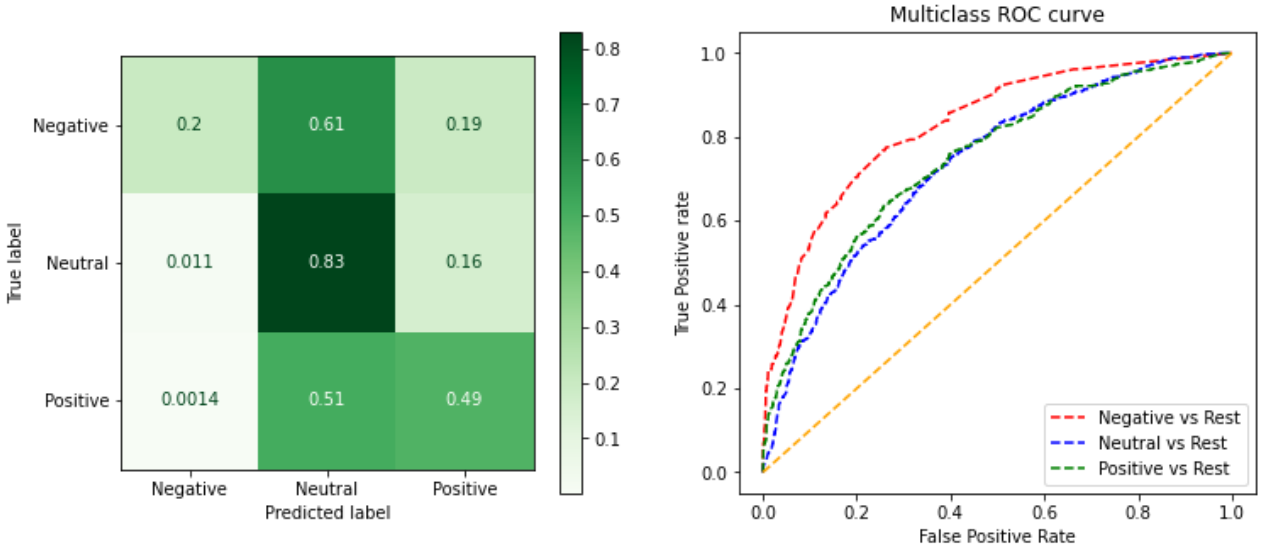
Random Forest

RF Model #1

In [116…

```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', RandomForestClassifier(class_weight='balanced',
                                                    random_state=42))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.96            Test Score:0.67


CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.66      0.20      0.31       155
           1       0.70      0.83      0.76      1335
           2       0.60      0.49      0.54       739

    accuracy                           0.67      2229
   macro avg       0.65      0.51      0.53      2229
weighted avg       0.66      0.67      0.65      2229
```



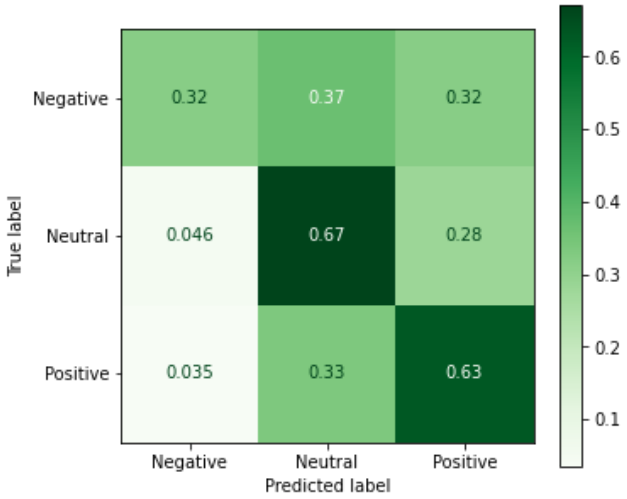
Hyperparameter Tuning with GridSearchCV

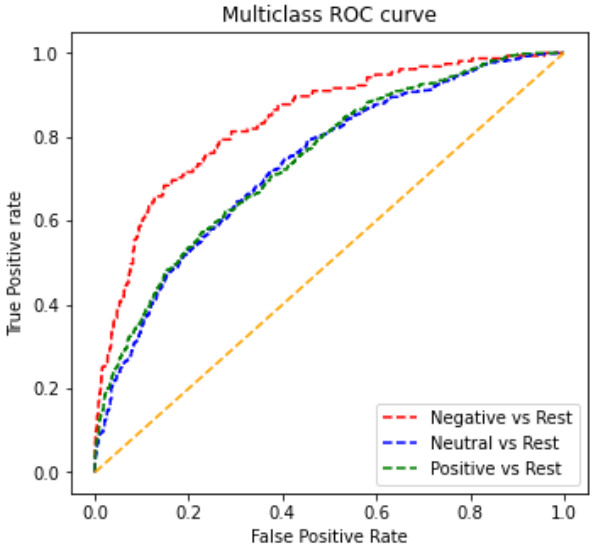
In [117…
```
if run_gridsearches == True:

    param_grid = {'clf__criterion': ['gini', 'entropy'],
            'clf__max_depth': [10, 20, None],
            'clf__min_samples_leaf': [1, 2, 3]}

    gridsearch = GridSearchCV(estimator=clf_pipe, param_grid = param_grid, scoring='rec

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

Tuned RF Model #1

In [118…
```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', RandomForestClassifier(class_weight='balanced',
                                                    criterion='entropy',
                                                    min_samples_leaf=2,
                                                    random_state=42))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.83            Test Score:0.63


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.36      0.32      0.34       155
           1       0.75      0.67      0.71      1335
           2       0.52      0.63      0.57       739

    accuracy                           0.63      2229
   macro avg       0.54      0.54      0.54      2229
weighted avg       0.65      0.63      0.64      2229
```



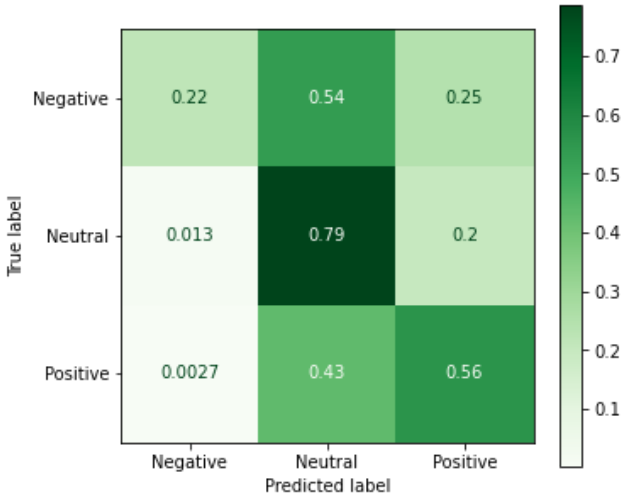


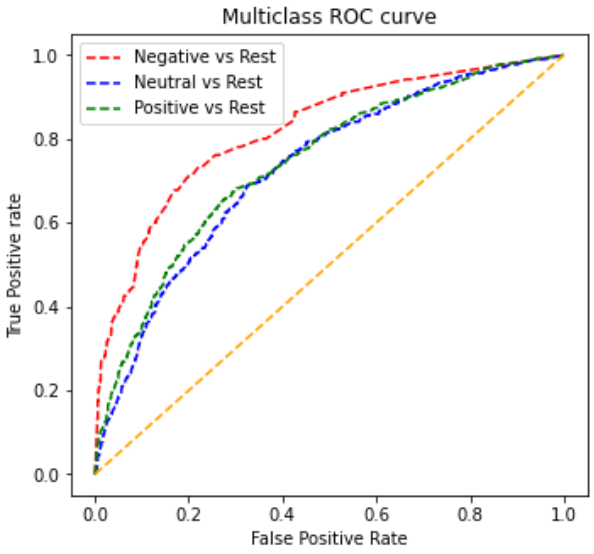
RF Model #2 - Random Oversampling

In [130…

```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                    ('os', RandomOverSampler(random_state=42)),
                    ('clf', RandomForestClassifier(class_weight='balanced'))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.96            Test Score:0.67


CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.63      0.22      0.33       155
           1       0.72      0.79      0.75      1335
           2       0.58      0.56      0.57       739

    accuracy                           0.67      2229
   macro avg       0.64      0.52      0.55      2229
weighted avg       0.67      0.67      0.66      2229
```





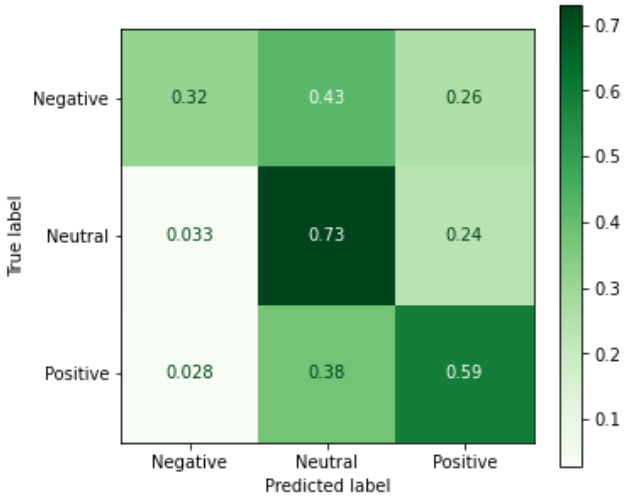
Tuning Random Oversampled RF Model #2

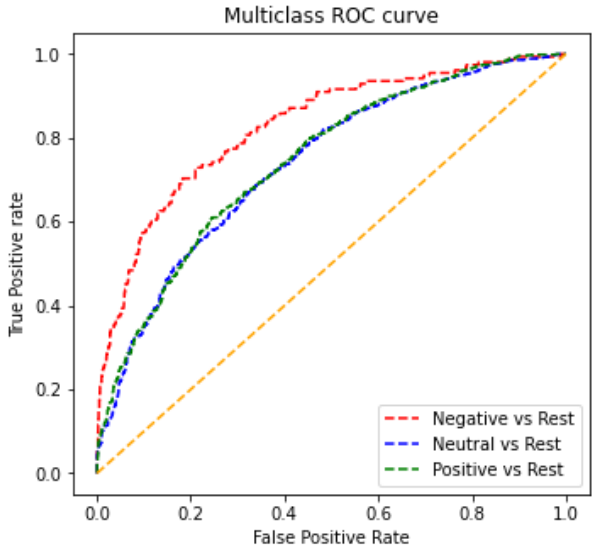
In [131…
```
if run_gridsearches == True:
    param_grid = {'clf__criterion': ['gini', 'entropy'],
            'clf__max_depth': [10, 20, None],
            'clf__min_samples_leaf': [1, 2, 3],
            'vectorizer__norm': ['l1', 'l2']}

    gridsearch = GridSearchCV(estimator=clf_pipe, param_grid = param_grid,
                              scoring='recall_macro', n_jobs=2)

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

Tuned RF Model #2

In [132…
```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                    ('os', RandomOverSampler(random_state=42)),
                    ('clf', RandomForestClassifier(class_weight='balanced',
                                                   criterion='entropy',
                                                   min_samples_leaf=2,
                                                   random_state=42))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.88            Test Score:0.66


CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.43      0.32      0.36       155
           1       0.74      0.73      0.73      1335
           2       0.55      0.59      0.57       739

    accuracy                           0.66      2229
   macro avg       0.57      0.55      0.56      2229
weighted avg       0.65      0.66      0.65      2229
```





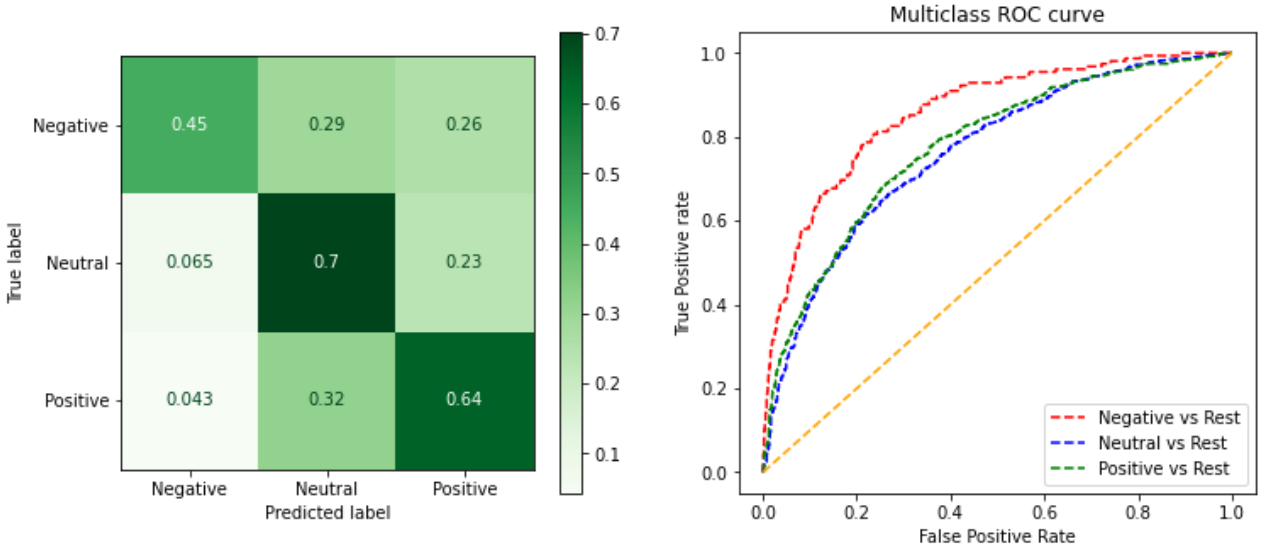
Logistic Regression

LR Model #1

In [120…

```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                     stop_words=stop_lst)),
                     ('clf', LogisticRegressionCV(class_weight='balanced',
                                                     random_state=42, cv=3))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Training Score: 0.89             Test Score:0.66


CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.37      0.45      0.41       155
           1       0.77      0.70      0.73      1335
           2       0.57      0.64      0.60       739

    accuracy                           0.66      2229
   macro avg       0.57      0.60      0.58      2229
weighted avg       0.68      0.66      0.67      2229
```



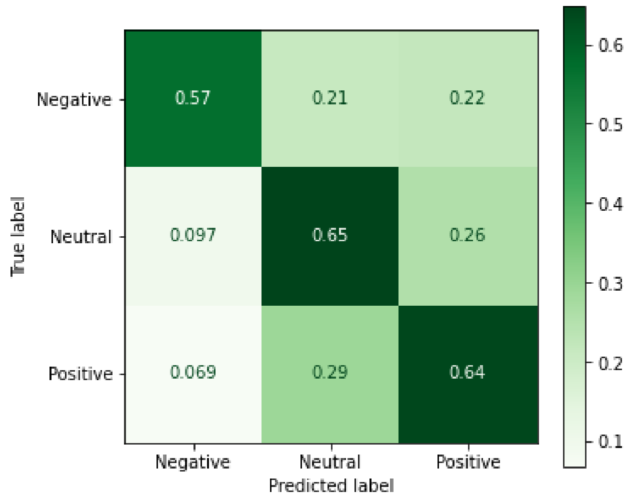
Hyperparameter Tuning with GridSearchCV

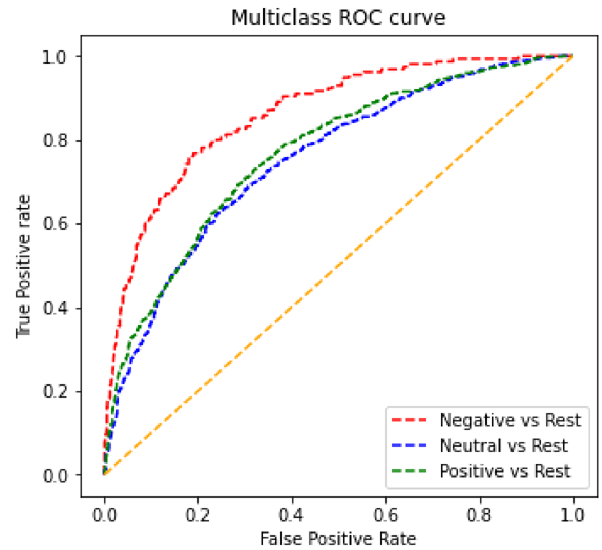
In [121…
```
if run_gridsearches == True:
    param_grid = {'clf__class_weight': ['balanced'],
                  'clf__max_iter': [100, 500, 1000],
                  'clf__Cs': [[0.01], [0.1], [1]],
                  'clf__solver': ['liblinear', 'lbfgs', 'sag']}

    gridsearch = GridSearchCV(estimator=clf_pipe, param_grid = param_grid, scoring='rec

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

Tuned LR Model #1

In [122…
```
clf_pipe = Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                                    stop_words=stop_lst)),
                     ('clf', LogisticRegressionCV(class_weight='balanced',
                                                  random_state=42, cv=3, solver='liblin
                                                  max_iter=100, Cs=[1]))])

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.81            Test Score:0.64


CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.33      0.57      0.42       155
           1       0.78      0.65      0.71      1335
           2       0.56      0.64      0.60       739

    accuracy                           0.64      2229
   macro avg       0.55      0.62      0.57      2229
weighted avg       0.67      0.64      0.65      2229
```





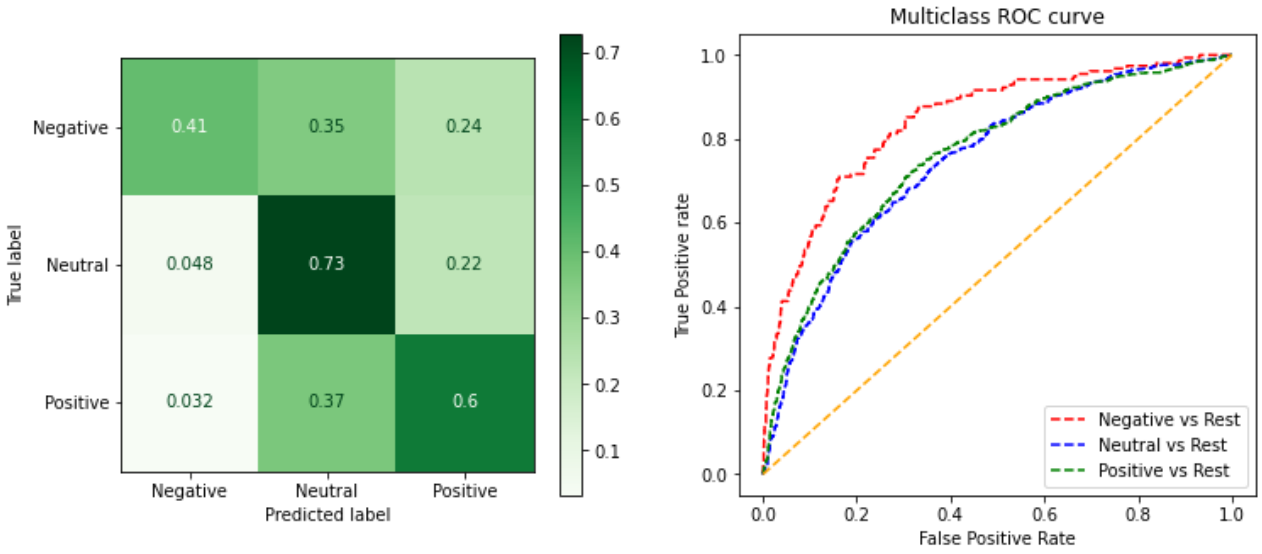
LR Model #2 - Random Oversampling

In [123…

```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                                                                       stop_words=stop_l
                                                                       norm='l2')),
                                       ('os', RandomOverSampler(random_state=42)),
                                       ('clf', LogisticRegressionCV(class_weight='balan
                                                                     random_state=42, cv
clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\berke\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Training Score: 0.95             Test Score:0.66


CLASSIFICATION REPORT
----------------------------------------
              precision    recall  f1-score   support

           0       0.42      0.41      0.41       155
           1       0.75      0.73      0.74      1335
           2       0.57      0.60      0.59       739

    accuracy                           0.66      2229
   macro avg       0.58      0.58      0.58      2229
weighted avg       0.67      0.66      0.67      2229
```



Tuning Random Oversampled LR Model #2

In [124…

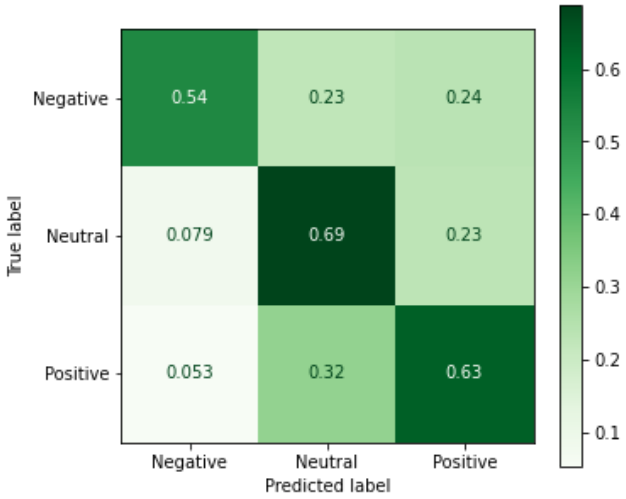
```
if run_gridsearches == True:
    param_grid = {'clf__max_iter': [100, 500, 1000],
                  'clf__Cs': [[0.01], [0.1], [1]],
                  'clf__solver': ['liblinear', 'lbfgs', 'sag']}

    gridsearch = GridSearchCV(estimator=clf_pipe, param_grid = param_grid, scoring='rec

    gridsearch.fit(X_train,  y_train)
    gridsearch.best_params_
```

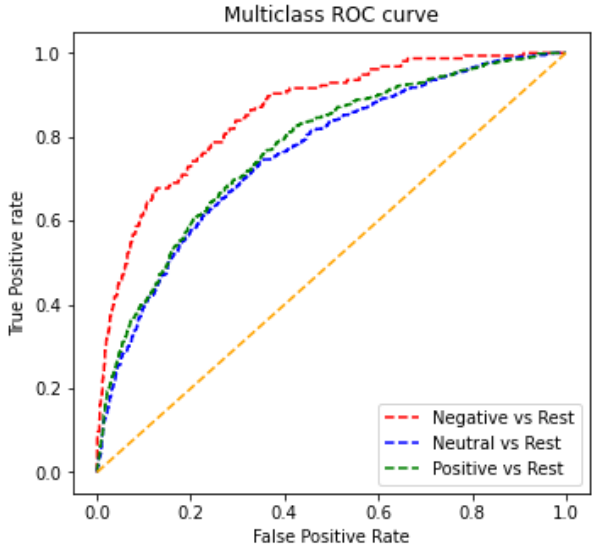
Tuned LR Model #2

In [125…

```
clf_pipe = imblearn.pipeline.Pipeline([('vectorizer', TfidfVectorizer(tokenizer=tokeniz
                                                                       stop_words=stop_l
                                                                       norm='l2')),
                                       ('os', RandomOverSampler(random_state=42)),
                                       ('clf', LogisticRegressionCV(class_weight='balan
                                                                    random_state=42, cv
                                                                    max_iter=100, Cs=[1

clf_pipe.fit(X_train, y_train)
y_pred = clf_pipe.predict(X_test)
multiclass_eval(y_test, y_pred, X_test, X_train, clf_pipe)
```

```
Training Score: 0.84            Test Score:0.66

CLASSIFICATION REPORT
-----------------------------------------
              precision    recall  f1-score   support

           0       0.36      0.54      0.43       155
           1       0.77      0.69      0.73      1335
           2       0.57      0.63      0.60       739

    accuracy                           0.66      2229
   macro avg       0.57      0.62      0.59      2229
weighted avg       0.68      0.66      0.67      2229
```





iNTERPRET

Evaluate how well your work solves the stated business problem.

Questions to consider:

- How do you interpret the results?
- How well does your model fit your data? How much better is this than your baseline model?
- How confident are you that your results would generalize beyond the data you have?
- How confident are you that this model would benefit the business if put into use?

In [ ]:

CONCLUSIONS & RECOMMENDATIONS

Provide your conclusions about the work you've done, including any limitations or next steps.

Questions to consider:

- What would you recommend the business do as a result of this work?

- What are some reasons why your analysis might not fully solve the business problem?
- What else could you do in the future to improve this project?

In [126…
```
# from PIL import Image
```

In [127…
```
# neg_mask = np.array(Image.open("images/download (1).png"))
# print(neg_mask)

# def transform_format(val):
#     if val == 0:
#         return 255
#     else:
#         return val

# transformed_wine_mask = np.ndarray((neg_mask.shape[0],neg_mask.shape[1]), np.int32)

# for i in range(len(neg_mask)):
#     transformed_wine_mask[i] = list(map(transform_format, neg_mask[i]))

# wordcloud = WordCloud(collocations=True, background_color='white', colormap='Greens',
#                       width=800, height=600, mask=transformed_wine_mask)
# wordcloud.generate(','.join(tokens_neg_ls))
# plt.figure(figsize = (10, 20), facecolor = None)
# plt.imshow(wordcloud)
# plt.axis('off');
```

In [ ]:

In [ ]: