Classifying Bitcoin Tweet Sentiment

Overview

This project seeks to build a model that accurately classifies tweets about Bitcoin as having either positive or negative sentiment. Unlabeled tweets classified by this model could ultimately could be used to analyze time trends on Bitcoin sentiment and assess the predictive power of tweets on future price movements of the cryptocurrency.

Methods

First, I inspect the data and prepare the text for modeling. Next, I run several machine learning classification models (Multinomial Naive Bayes, Logistic Regression, Linear Support Vector Classifier, Random Forest) with a number of vectorizers (TF-IDF, Count Vectorizer) and hyper-tune to select a best-performing model. Finally, I visually represent model performance and feature importance.

Data Sources

The Twitter data is sourced from Kaggle and contains approximately one million tweets about Bitcoin during the period of February to August 2021. The tweets are pre-labeled with positive or negative sentiment, with approximately 53% tagged as negative and 47% positive.

Import Statements, Data Preparation

In [2]:
```
import pandas as pd
import numpy as np

import nltk
from nltk.probability import FreqDist
from nltk.corpus import stopwords, gutenberg
from nltk.tokenize import regexp_tokenize, word_tokenize, RegexpTokenizer
from nltk.tokenize import TweetTokenizer
from nltk.collocations import *
from wordcloud import WordCloud
from gensim.models import Word2Vec
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import LinearSVC

import string
import re
import datetime

import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import seaborn as sns

from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, f1_score, recall_score, \
precision_score, confusion_matrix, classification_report, roc_curve, auc, \
average_precision_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_recall_curve

np.random.seed(42) # setting the seed

# only need to run these once
#nltk.download("gutenberg")
#nltk.download("stopwords")
#pip install tweepy
```

Loading and Inspecting Data

Upon initial inspection, it looks like there are some columns that won't be relevant to my analysis and some null values I will need to handle. A few of the columns will also need to be converted to an appropriate data type.

In [2]:
```
data = pd.read_csv('data/bitcoin_tweets1000000.csv', encoding='ISO-8859-1', inc
data.head()
```

```
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/I
Python/core/interactiveshell.py:3145: DtypeWarning: Columns (0,5,6,7,8,13) hav
e mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[2]:

	user_name	user_location	user_description	user_created	user_followers	user_friends
0	DeSota Wilson	Atlanta, GA	Biz Consultant, real estate, fintech, startups...	2009-04-26 20:05:09	8534	7605
1	CryptoND	NaN	ðŸ˜ BITCOINLIVE is a Dutch platform aimed at ...	2019-10-17 20:12:10	6769	1532
2	Tdlmatias	London, England	IM Academy : The best #forex, #SelfEducation, ...	2014-11-10 10:50:37	128	332
3	Crypto is the future	NaN	I will post a lot of buying signals for BTC tr...	2019-09-28 16:48:12	625	129
4	Alex Kirchmaier ðŸ‡¦ðŸ‡¹ðŸ‡¸ðŸ‡ª #FactsSupersp...	Europa	Co-founder @RENJERJerky | Forbes 30Under30 | I...	2016-02-03 13:15:55	1249	1472

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000025 entries, 0 to 999999
Data columns (total 16 columns):
 #   Column            Non-Null Count    Dtype  
---  ------            --------------    -----  
 0   user_name         999981 non-null   object 
 1   user_location     536681 non-null   object 
 2   user_description  889249 non-null   object 
 3   user_created      1000000 non-null  object 
 4   user_followers    1000000 non-null  object 
 5   user_friends      1000000 non-null  object 
 6   user_favourites   999997 non-null   object 
 7   user_verified     999996 non-null   object 
 8   date              999995 non-null   object 
 9   text              999993 non-null   object 
 10  hashtags          983199 non-null   object 
 11  source            996633 non-null   object 
 12  is_retweet        999946 non-null   object 
 13  cleanText         999987 non-null   object 
 14  Polarity Score    999988 non-null   float64
 15  sentiment         999986 non-null   float64
dtypes: float64(2), object(14)
memory usage: 129.7+ MB
```

In [4]:
```
data.shape
```
Out[4]:
```
(1000025, 16)
```

In [5]:
```
# 53% negative, 47% positive - fairly balanced
print(data[['sentiment']].value_counts(normalize=True))
```
```
sentiment
0.0          0.527057
1.0          0.472943
dtype: float64
```

In [6]:
```
data.describe()
```
Out[6]:

	Polarity Score	sentiment
count	999988.000000	999986.000000
mean	0.144692	0.472943
std	0.271044	0.499268
min	-1.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.286508	1.000000
max	1.000000	1.000000

In [7]:
```
data.isna().sum()
```
Out[7]:
```
user_name               44
user_location      463344
user_description   110776
user_created            25
user_followers          25
user_friends            25
user_favourites         28
user_verified           29
date                    30
text                    32
hashtags             16826
source                3392
is_retweet              79
cleanText               38
Polarity Score          37
sentiment               39
dtype: int64
```

In [45]:
```
# loading in bitcoin price data from feb 2021 to aug 2021
btc_price = pd.read_csv('data/BTC-USD.csv')
btc_price.head()
```

Out[45]:

	Date	Open	High	Low	Close	Adj Close	Volu
0	2021-02-01	33114.578125	34638.214844	32384.228516	33537.175781	33537.175781	614004000
1	2021-02-02	33533.199219	35896.882813	33489.218750	35510.289063	35510.289063	63088585
2	2021-02-03	35510.820313	37480.187500	35443.984375	37472.089844	37472.089844	61166818
3	2021-02-04	37475.105469	38592.175781	36317.500000	36926.066406	36926.066406	68838074
4	2021-02-05	36931.546875	38225.906250	36658.761719	38144.308594	38144.308594	58598066

In [9]:
```
btc_price.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 212 entries, 0 to 211
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       212 non-null    object 
 1   Open       212 non-null    float64
 2   High       212 non-null    float64
 3   Low        212 non-null    float64
 4   Close      212 non-null    float64
 5   Adj Close  212 non-null    float64
 6   Volume     212 non-null    int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 11.7+ KB
```

Initial Data Cleaning

In this section, I will drop irrelevant columns, deal with NaNs, and convert certain columns into the appropriate data type.

In [10]:
```
# remove columns we don't need
data = data.drop(columns=['user_created', 'cleanText', 'user_location',
                          'user_description'])
```

In [11]:
```
# drop rows where date, text or sentiment is NaN
data = data.dropna(subset=['text', 'date', 'sentiment'])
print(data.shape)
data.isna().sum()
```

```
(999986, 12)
```

Out[11]:
```
user_name              19
user_followers          0
user_friends            0
user_favourites         0
user_verified           0
date                    0
text                    0
hashtags            16792
source               3357
is_retweet             43
Polarity Score          0
sentiment               0
dtype: int64
```

The date column needs to be transformed into a datetime format. The below commented out cell results in an error given there is text in some of the dates. The remaining cells handle the transformation.

In [12]:
```
#data['date'] = [datetime.datetime.strptime(date, '%Y-%m-%d %H:%M:%S') for date
#data.head()
```

In [13]:
```
# visually inspecting rows with improperly formatted date data
display(data[data['date'].str.contains('ETH|BTC|btc')])
# these columns are shifted two to the right

# what about all columns with missing source and retweet?
data[data['source'].isna() & data['is_retweet'].isna()] # the rest of them look
```

	user_name	user_followers	user_friends	user_favourites	user_verified
64943	Can roam the worldï¼lt's nine to five againáµ...	36	False	2021-04-07 16:23:03	@krakenfx #ETH #BTC If you want to become po...
137068	*Muhammad Yasir* hello stalker nice to tweet ...	499	False	2021-06-23 14:51:12	Official ESHOP Airdrop. If You Missed Meme Tok...
180575	â€¢ Learn n To Do d'Best...!!!!!!!!	127	False	2021-06-22 13:31:16	@pufferswap Nice project\n\n@karnoto_hendrik \...
693194	FB- Xiomara CastaÃ±eda	531	False	2021-07-26 11:54:15	#btc to the moon ðŸš ðŸš 77774
697397	Pin bb : 26ea62f8 . Line : baliratih_bali	21	False	2021-07-26 11:48:46	#btc to the moon ðŸš ðŸš 46059

Out[13]:

	user_name	user_followers	user_friends	user_favourites	user_verifiec
64943	Can roam the worldï¼Œlt's nine to five againáµ...	36	False	2021-04-07 16:23:03	@krakenfx #ETH #BTC If you want to become po...
137068	*Muhammad Yasir* hello stalker nice to tweet ...	499	False	2021-06-23 14:51:12	Official ESHOP Airdrop. If You Missed Meme Tok...
180575	â€¢ Learn n To Do d'Best...!!!!!!!!	127	False	2021-06-22 13:31:16	@pufferswap Nice project\n\n@karnoto_hendrik \...
228487	DayTradeIdeas	9190	287	2229	False
240490	DayTradeIdeas	9191	287	2229	False
316615	Millionaire Box	225	158	229	False
378317	Aldrich Baron	5	12	12	False
398481	Evan	57	331	727	False
398693	Evan	57	331	727	False
398716	Evan	57	331	727	False
398738	Evan	57	331	727	False
398749	Evan	57	331	727	False
398760	Evan	57	331	727	False
398771	Evan	57	331	727	False

	user_name	user_followers	user_friends	user_favourites	user_verified
398784	Evan	57	331	727	False
419332	Adi Gualterio	8	10	12	False
580682	topstonks	2094	1194	150	False
583965	topstonks	2094	1194	150	False
588624	topstonks	2094	1194	150	False
594259	topstonks	2094	1194	150	False
622716	topstonks	2132	1193	150	False
630083	topstonks	2131	1193	150	False
633045	topstonks	2131	1193	150	False
635741	topstonks	2131	1193	150	False
636724	topstonks	2131	1193	150	False
643994	topstonks	2131	1193	150	False
651480	topstonks	2131	1193	150	False
651777	Teddy wood	20	12	5	False
677577	topstonks	2132	1193	150	False
678707	topstonks	2132	1193	150	False

	user_name	user_followers	user_friends	user_favourites	user_verifiec
679901	topstonks	2132	1193	150	False
683604	topstonks	2132	1193	150	False
693194	FB- Xiomara CastaÃ±eda	531	False	2021-07-26 11:54:15	#btc to the moon ðŸš ðŸš 77774
697397	Pin bb : 26ea62f8 . Line : baliratih_bali	21	False	2021-07-26 11:48:46	#btc to the moon ðŸš ðŸš 46059
703630	SynchroBitâ„¢ Hybrid Exchange	11107	19	226	False
718123	Aldrich Baron	6	19	11	False
737926	Crypto Joe	845	219	239	False
738150	Crypto Joe	845	219	239	False
808576	topstonks	2177	1195	149	False
813984	topstonks	2177	1195	149	False
864616	Millionaire Box	233	158	229	False
870434	Our Tokenomics	313	75	128	False
875897	Our Tokenomics	313	75	128	False

In [14]:
```
# dropping rows where date contains text
data = data[~data['date'].str.contains('ETH|BTC|btc')].reset_index(drop=True)
```

In [15]:
```
# transforming text data into date format
data['date'] = [datetime.datetime.strptime(date, '%Y-%m-%d %H:%M:%S') for date
data['date'] = pd.to_datetime(data['date']).dt.date
```

In [16]:
```
# checking that the dates look right
display(data['date'].max())
display(data['date'].min())
```

```
datetime.date(2021, 8, 21)
datetime.date(2021, 2, 5)
```

In [17]:
```
# replacing empty retweets with string
data['is_retweet'][data['is_retweet'].isna()] = 'n/a'
```

```
<ipython-input-17-806753b2bf5c>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['is_retweet'][data['is_retweet'].isna()] = 'n/a'
```

In [46]:
```
# converting date to datetime format
btc_price['Date'] = [datetime.datetime.strptime(date, '%Y-%m-%d') \
                     for date in btc_price['Date']]
#btc_price['Date'] = pd.to_datetime(btc_price['Date']).dt.date

# dataframe with just date and closing price
price_df = btc_price[['Date', 'Close']]
price_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 212 entries, 0 to 211
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    212 non-null    datetime64[ns]
 1   Close   212 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 3.4 KB
```

Text Preprocessing & Feature Engineering

In this section I will take a deeper look at the actual Tweet content and prepare it for modeling.

In [19]:
```
# only keeping the relevant columns
text_data = data[['text', 'date', 'sentiment', 'is_retweet']]
text_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999981 entries, 0 to 999980
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   text        999981 non-null  object
 1   date        999981 non-null  object
 2   sentiment   999981 non-null  float64
 3   is_retweet  999981 non-null  object
dtypes: float64(1), object(3)
memory usage: 30.5+ MB
```

In [20]:
```
text_data['text'][1]
```

Out[20]:
```
'ð\x9f\x98\x8e Today, that\'s this #Thursday, we will do a "ð\x9f\x8e¬ Take 2" with our friend @LeoWandersleb, #Btc #wallet #security expeâ\x80¦ https://t.co/go6aDgRml5'
```

I will use regex to handle some of the unwanted text, such as urls and Twitter handles.

In [21]:
```
#url pattern - want to remove
pattern = "(http|ftp|https):\/\/([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@?^=%&:\/~+#-]
print(re.findall(pattern, text_data['text'][0]))

#pattern = "(http|ftp|https):\/\/t.co\/[a-zA-Z0-9\-\.]{8}"
pattern = "(http|ftp|https):\/\/(t.co\/[a-zA-Z0-9\-\.]{8})"
print(re.findall(pattern, text_data['text'][0]))

pattern = "https:\/\/t.co\/[a-zA-Z0-9\-\.]{8}"
print(re.findall(pattern, text_data['text'][0]))

# apostrophe pattern
pattern = "([a-zA-Z]+(?:'[a-z]+)?)"
print(re.findall(pattern, text_data['text'][0]))
```

```
[('https', 't.co', '/xaaZmaJKiV'), ('https', 't.co', '/sgBxMkP1SI')]
[('https', 't.co/xaaZmaJK'), ('https', 't.co/sgBxMkP1')]
['https://t.co/xaaZmaJK', 'https://t.co/sgBxMkP1']
['Blue', 'Ridge', 'Bank', 'shares', 'halted', 'by', 'NYSE', 'after', 'bitcoin', 'ATM', 'announcement', 'https', 't', 'co', 'xaaZmaJKiV', 'MyBlueRidgeBank', 'https', 't', 'co', 'sgBxMkP', 'SI']
```

In [22]:
```
# removing all urls

pattern_https = "https:\/\/t.co\/[a-zA-Z0-9\-\.]{8}"
pattern_www = "www\.[a-z]?\.?(com)+|[a-z]+\.(com)"
#string = text_data['text'][1]
repl = ''

#text_ex = re.sub(pattern_http, repl, string, count=0, flags=0)

text_data['text'] = [re.sub(pattern_https, repl, string, count=0, flags=0)\
                     for string in text_data['text']]
text_data['text'] = [re.sub(pattern_www, repl, string, count=0, flags=0)\
                     for string in text_data['text']]

text_data['text'][1]

#pattern = "([a-zA-Z]+(?:'[a-z]+)?)"
#print(re.findall(pattern, text_ex))
```

```
<ipython-input-22-f020dda00306>:10: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['text'] = [re.sub(pattern_https, repl, string, count=0, flags=0)\
<ipython-input-22-f020dda00306>:12: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['text'] = [re.sub(pattern_www, repl, string, count=0, flags=0)\
```

Out[22]:

```
'ð\x9f\x98\x8e Today, that\'s this #Thursday, we will do a "ð\x9f\x8e¬ Take 2"
with our friend @LeoWandersleb, #Btc #wallet #security expeâ\x80¦ 15'
```

In [23]:

```python
# remove twitter handles

pattern_handle = "@[A-Za-z0-9]+"
repl = ''

text_data['text'] = [re.sub(pattern_handle, repl, string, count=0, flags=0)\
                     for string in text_data['text']]
```

```
<ipython-input-23-265c887e34dc>:6: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['text'] = [re.sub(pattern_handle, repl, string, count=0, flags=0)\
```

In [24]:

```python
# remove numbers

pattern_num = r"\b\d+\b" #"\b(\d+)\b"
repl = ''

#print(re.findall(pattern_num, text_data['text'][3]))

text_data['text'] = [re.sub(pattern_num, repl, string, count=0, flags=0)\
                     for string in text_data['text']]

text_data['text'].head()
```

```
<ipython-input-24-e2631955cc8c>:8: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['text'] = [re.sub(pattern_num, repl, string, count=0, flags=0)\
```

Out[24]:

```
0    Blue Ridge Bank shares halted by NYSE after #b...
1    ðŸ˜  Today, that's this #Thursday, we will do ...
2    Guys evening, I have read this article about B...
3    $BTC A big chance in a billion! Price: \. (// ...
4    This network is secured by   nodes as of today...
Name: text, dtype: object
```

Converting everything into lower case so that we don't treat the same word with different casing as different words.

In [25]:
```
text_data['text'] = text_data["text"].str.lower()
```

```
<ipython-input-25-4b865df3999c>:1: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['text'] = text_data["text"].str.lower()
```

Looking at how different tokenizers treat the text data:

In [26]:
```
tweet_tokenizer = TweetTokenizer()
print(tweet_tokenizer.tokenize(text_data['text'][3]))
print(text_data['text'][3])
#text_data['tweet_tokenized'] = text_data["text"].apply(tweet_tokenizer.tokeniz
```

```
['$', 'btc', 'a', 'big', 'chance', 'in', 'a', 'billion', '!', 'price', ':',
'\\', '.', '(', '/', '/', ':)', '#bitcoin', '#fx', '#btc', '#crypto']
$btc a big chance in a billion! price: \. (// :) #bitcoin #fx #btc #crypto
```

In [27]:
```
# tokenizing with regex tokenizer
basic_token_pattern = r"(?u)\b\w\w+\b"
tokenizer = RegexpTokenizer(basic_token_pattern)
tokenizer.tokenize(text_data['text'][3])
```

Out[27]:
```
['btc',
 'big',
 'chance',
 'in',
 'billion',
 'price',
 'bitcoin',
 'fx',
 'btc',
 'crypto']
```

The regex tokenizer seems to isolate just the key words in the text. We will go with this one.

In [28]:
```
# tokenizing with regex tokenizer
basic_token_pattern = r"(?u)\b\w\w+\b"

tokenizer = RegexpTokenizer(basic_token_pattern)

# Create new column with tokenized data
text_data["text_tokenized"] = text_data["text"].apply(tokenizer.tokenize)
```

```
<ipython-input-28-f44a4f87e3e5>:7: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data["text_tokenized"] = text_data["text"].apply(tokenizer.tokenize)
```

The below code replaces domain-specific abbreviations with the full word. This will be helpful for visualizations. Some groupings include: btc/bitcoin, eth/ethereum, crypto/cryptocurrency, doge/dogecoin.

In [29]:
```
# abbreviation dictionary
abbv_dict = {'btc': 'bitcoin',
             'ethereum': 'eth',
             'cryptocurrency': 'crypto',
             'dogecoin': 'doge'}

# tokenized text back to single string for functionality
text_data['text2'] = [" ".join(lst) for lst in text_data['text_tokenized']]

# replacing key with value
for key in abbv_dict.keys():
    text_data['text2'] = [text_str.replace(key, abbv_dict[key]) for text_str in

# before and after
print(text_data['text'][1])
print(text_data['text2'][1])
```

```
<ipython-input-29-b4dab91e48a2>:8: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['text2'] = [" ".join(lst) for lst in text_data['text_tokenized']]
<ipython-input-29-b4dab91e48a2>:12: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['text2'] = [text_str.replace(key, abbv_dict[key]) for text_str in
text_data['text2']]
ðŸ˜ today, that's this #thursday, we will do a "ðŸ ¬ take " with our friend ,
#btc #wallet #security expeâ ¦ 15
today that this thursday we will do take with our friend bitcoin wallet securi
ty expeâ 15
```

Confirming that the transformations worked:

In [30]:
```
print(text_data['text'][20000])
print(text_data['text2'][20000])
```

```
this is a nice setup for #btc to make a run back to ath. #bitcoin x5
this is nice setup for bitcoin to make run back to ath bitcoin x5
```

Tokenizing the new text data:

In [31]:
```
# tokenizing with regex tokenizer
basic_token_pattern = r"(?u)\b\w\w+\b"

tokenizer = RegexpTokenizer(basic_token_pattern)

# Create new column with tokenized data
```

```
text_data["text_tokenized"] = text_data["text2"].apply(tokenizer.tokenize)
text_data['text_tokenized'][1]
```

```
<ipython-input-31-1a2d027bd0bf>:7: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data["text_tokenized"] = text_data["text2"].apply(tokenizer.tokenize)
```

Out[31]:
```
['today',
 'that',
 'this',
 'thursday',
 'we',
 'will',
 'do',
 'take',
 'with',
 'our',
 'friend',
 'bitcoin',
 'wallet',
 'security',
 'expeâ',
 '15']
```

Creating a column that contains a list of the hashtags in each tweet:

In [32]:
```
text_data['hashtag'] = text_data['text'].\
apply(lambda x: re.findall(r'\B#\w*[a-zA-Z]+\w*', x))
```

```
<ipython-input-32-271edba60584>:1: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['hashtag'] = text_data['text'].\
```

In [33]:
```
# visually inspecting
display(text_data.head())
```

	text	date	sentiment	is_retweet	text_tokenized	text2	hashtag
0	blue ridge bank shares halted by nyse after #b...	2021-02-10	0.0	False	[blue, ridge, bank, shares, halted, by, nyse, ...	blue ridge bank shares halted by nyse after bi...	[#bitcoin]
1	ðŸ˜ today, that's this #thursday, we will do ...	2021-02-10	0.0	False	[today, that, this, thursday, we, will, do, ta...	today that this thursday we will do take with ...	[#thursday, #btc, #wallet, #security]
2	guys evening, i have read this article about b...	2021-02-10	0.0	False	[guys, evening, have, read, this, article, abo...	guys evening have read this article about bitc...	[]
3	$btc a big chance in a billion! price: \. (// ...	2021-02-10	0.0	False	[bitcoin, big, chance, in, billion, price, bit...	bitcoin big chance in billion price bitcoin fx...	[#bitcoin, #fx, #btc, #crypto]
4	this network is secured by nodes as of today...	2021-02-10	0.0	False	[this, network, is, secured, by, nodes, as, of...	this network is secured by nodes as of today s...	[#btc]

Now, we will remove stopwords and punctuation.

In [34]:
```
# storing stopwords list and adding punctuation
stopwords_list = stopwords.words('english')
stopwords_list += list(string.punctuation)

# function to remove stopwords
def remove_stopwords(token_list):
    """
    Given a list of tokens, return a list where the tokens
    that are also present in stopwords_list have been
    removed
    """
    return [word for word in token_list if word not in stopwords_list]
```

In [35]:
```
# applying remove stopwords function
text_data["text_without_stopwords"] = text_data["text_tokenized"].apply(remove_
```

```
<ipython-input-35-0e64d31a4f6b>:2: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data["text_without_stopwords"] = text_data["text_tokenized"].apply(remo
ve_stopwords)
```

Adding in some columns that represent the number of hashtags per tweet, as well as the length of each tweet, and whether the tweet contains a price.

In [36]:
```
# count of hashtags

text_data['hashtag_count'] = [len(x) for x in text_data['hashtag']]
```

```
<ipython-input-36-d813cf7205a0>: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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['hashtag_count'] = [len(x) for x in text_data['hashtag']]
```

In [37]:
```
# tweet length

text_data['tweet_length'] = [len(x) for x in text_data['text_tokenized']]
```

```
<ipython-input-37-a746883ab5df>: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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data['tweet_length'] = [len(x) for x in text_data['text_tokenized']]
```

In [38]:
```
# adding column showing whether the tweet contains price

price_query = r'\$(?:\d{1,3}[,.]?)+(?:\\d{1,2})?'
text_data["contains_price"] = text_data["text"].str.contains(price_query)
```

```
<ipython-input-38-66081249b850>:4: 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/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  text_data["contains_price"] = text_data["text"].str.contains(price_query)
```

Visually inspecting the data one last time. We can see that final tokenized text is lowercased, without urls, stopwords & numbers, and that our feature engineered columns have been added.

In [39]:
```
text_data.info() # reinspecting datatypes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999981 entries, 0 to 999980
Data columns (total 11 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   text                     999981 non-null  object 
 1   date                     999981 non-null  object 
 2   sentiment                999981 non-null  float64
 3   is_retweet               999981 non-null  object 
 4   text_tokenized           999981 non-null  object 
 5   text2                    999981 non-null  object 
 6   hashtag                  999981 non-null  object 
 7   text_without_stopwords   999981 non-null  object 
 8   hashtag_count            999981 non-null  int64  
 9   tweet_length             999981 non-null  int64  
 10  contains_price           999981 non-null  bool   
dtypes: bool(1), float64(1), int64(2), object(7)
memory usage: 77.2+ MB
```

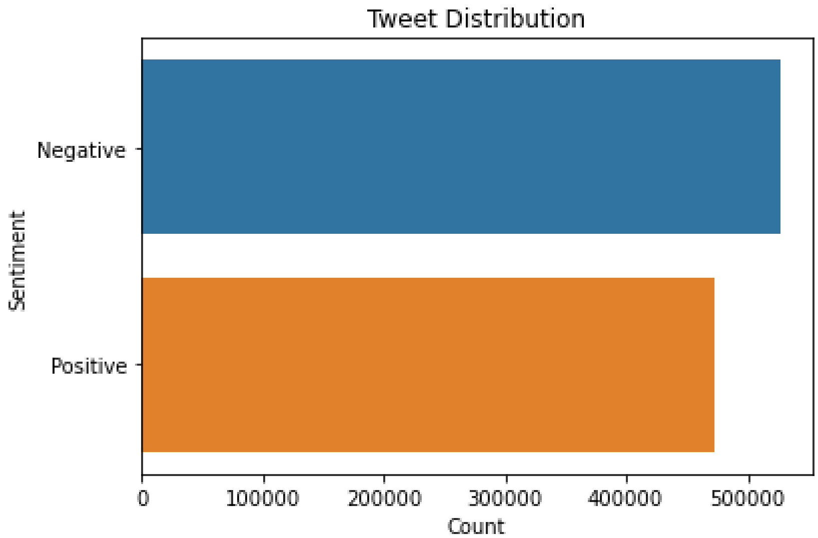
In [40]:
```
display(text_data.head())
```

	text	date	sentiment	is_retweet	text_tokenized	text2	hashtag	text_without_s
0	blue ridge bank shares halted by nyse after #b...	2021-02-10	0.0	False	[blue, ridge, bank, shares, halted, by, nyse, ...	blue ridge bank shares halted by nyse after bi...	[#bitcoin]	[blue, ridge, bai halted, n
1	ðŸ˜ today, that's this #thursday, we will do ...	2021-02-10	0.0	False	[today, that, this, thursday, we, will, do, ta...	today that this thursday we will do take with ...	[#thursday, #btc, #wallet, #security]	[today, thur friend, bitcc
2	guys evening, i have read this article about b...	2021-02-10	0.0	False	[guys, evening, have, read, this, article, abo...	guys evening have read this article about bitc...	[]	[guys, eve article, bitcoi
3	$btc a big chance in a billion! price: \. (// ...	2021-02-10	0.0	False	[bitcoin, big, chance, in, billion, price, bit...	bitcoin big chance in billion price bitcoin fx...	[#bitcoin, #fx, #btc, #crypto]	[bitcoin, bi billion, price
4	this network is secured by nodes as of today...	2021-02-10	0.0	False	[this, network, is, secured, by, nodes, as, of...	this network is secured by nodes as of today s...	[#btc]	[networl nodes, tc

Data Exploration

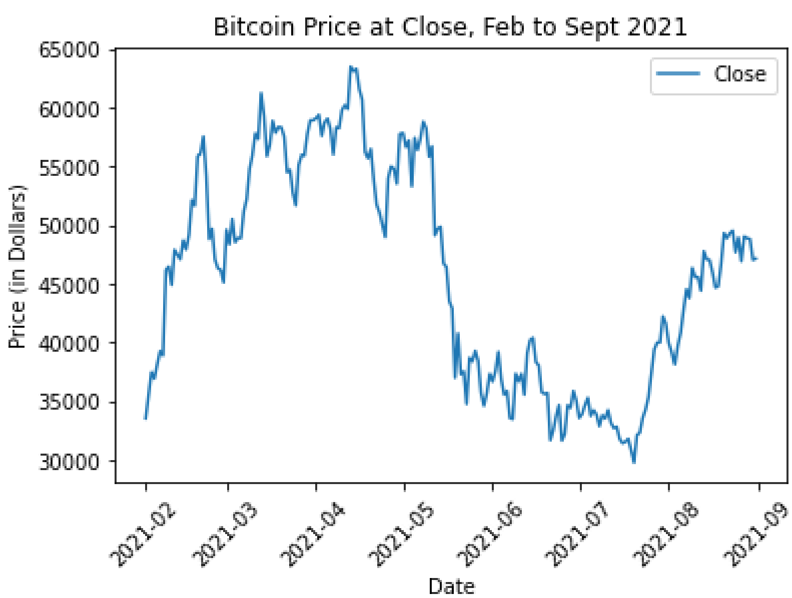
First, let's look at how balanced (or unbalanced) our dataset is. This looks fairly balanced to me, with slightly more tweets having negative sentiment.

In [8]:
```
# dataset looks fairly balanced
ax = sns.countplot(data = text_data, y = 'sentiment')
ax.set(xlabel='Count', ylabel='Sentiment', title='Tweet Distribution')
ax.set_yticklabels(['Negative', 'Positive'])
plt.savefig('images/tweet_distribution.png')
```



Looking at the price of Bitcoin during the same period as our Twitter data, as well as the average daily sentiment from the Tweets & daily volume. I want to see whether days of high Bitcoin Twitter volume correspond with days of overwhelmingly positive or negative sentiment. I also want to see how the price of Bitcoin moved during this time period.

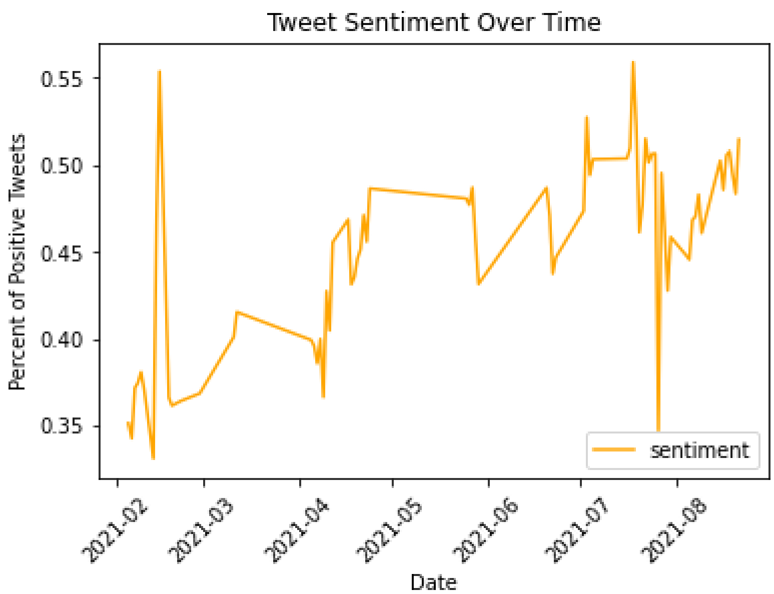
In [33]:
```
import matplotlib.dates as mdates
myFmt = mdates.DateFormatter('%m')

# price of bitcoin from feb 2021 to sept 2021
price_df.plot(x='Date')
plt.xticks(rotation=45)
plt.ylabel('Price (in Dollars)')
plt.title('Bitcoin Price at Close, Feb to Sept 2021')
ax.xaxis.set_major_formatter(myFmt)
```



In [55]:
```
# percent of positive tweets per day
grouped = text_data[['date', 'sentiment']].groupby('date').mean().reset_index()
grouped['date'] = pd.to_datetime(grouped['date']).dt.date
grouped.plot(x='date', c='orange')
plt.xticks(rotation=45)
```

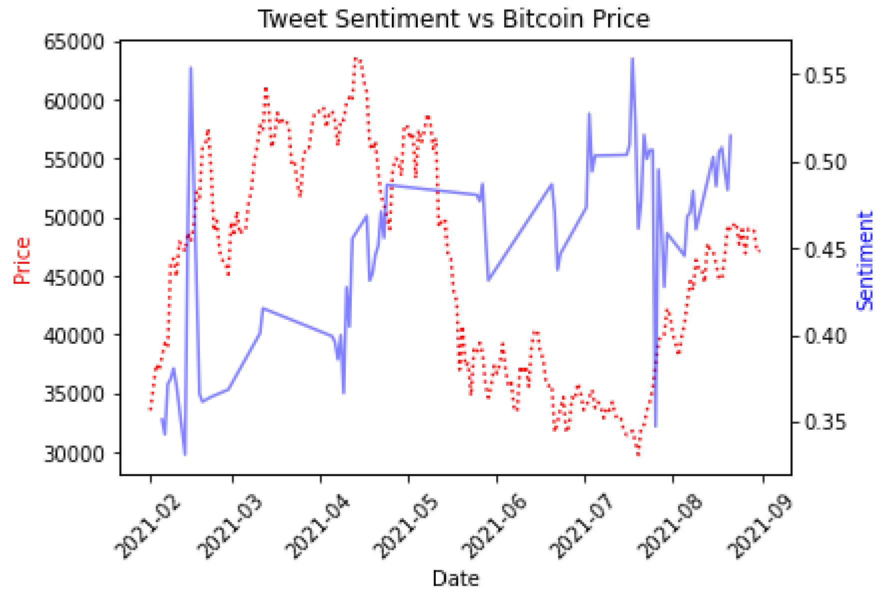
```
plt.xlabel('Date')
plt.ylabel('Percent of Positive Tweets')
plt.title('Tweet Sentiment Over Time')
len(grouped) # only 72 observations (unique days with tweets)
```

Out[55]: 72



In [75]:

```
# create figure and axis objects with subplots()
fig,ax = plt.subplots()
# make a plot
ax.plot(price_df.Date, price_df.Close, color="red", linestyle='dotted')
# set x-axis label
ax.set_xlabel("Date", fontsize = 10)
# set y-axis label
ax.set_ylabel("Price",color="red", fontsize=10)
plt.xticks(rotation=45)

# twin object for two different y-axis on the sample plot
ax2=ax.twinx()
# make a plot with different y-axis using second axis object
ax2.plot(grouped.date, grouped.sentiment, color="blue", alpha=0.5)
ax2.set_ylabel("Sentiment",color="blue",fontsize=10)

# title, show and save figure
plt.title('Tweet Sentiment vs Bitcoin Price')
plt.show()
plt.savefig('tweet_sentiment_vs_bitcoin_px.png')
```

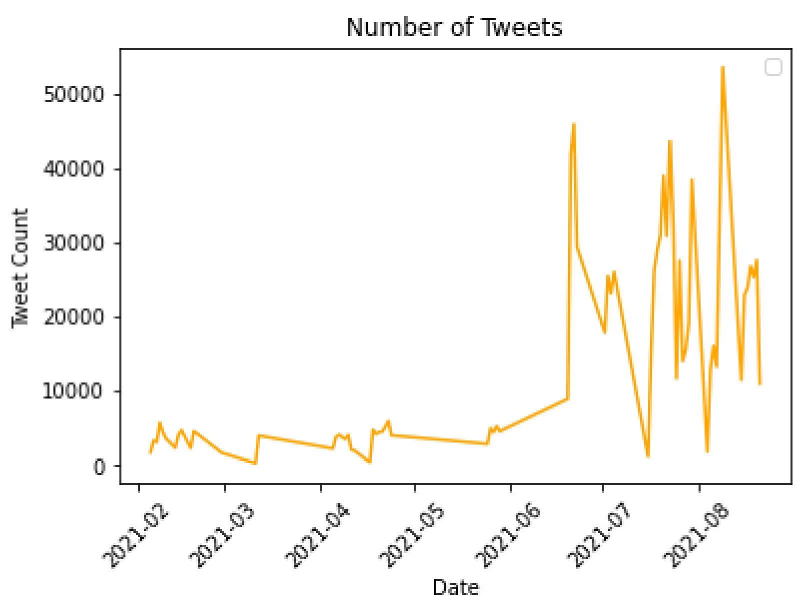



```
<Figure size 432x288 with 0 Axes>
```

Daily volume of Tweets on Bitcoin:

In [63]:
```
# count of tweets per day

count = text_data[['date', 'sentiment']].groupby('date').count().reset_index()
count['date'] = pd.to_datetime(count['date']).dt.date
count.plot(x='date', c='orange') # seems like activity spikes in the summer
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Tweet Count')
plt.legend('')
plt.title('Number of Tweets')
```

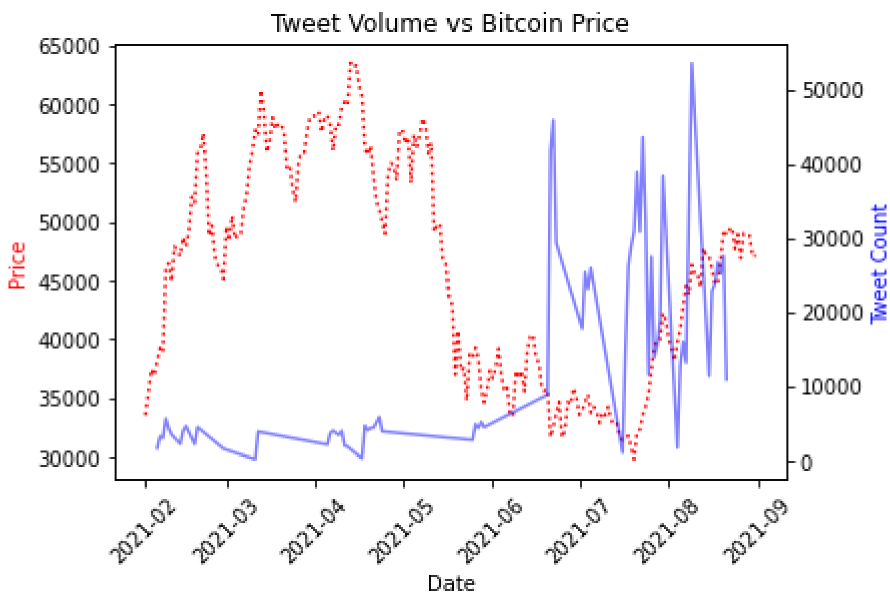
Out[63]:
```
Text(0.5, 1.0, 'Number of Tweets')
```



Bitcoin price & volume of tweets:

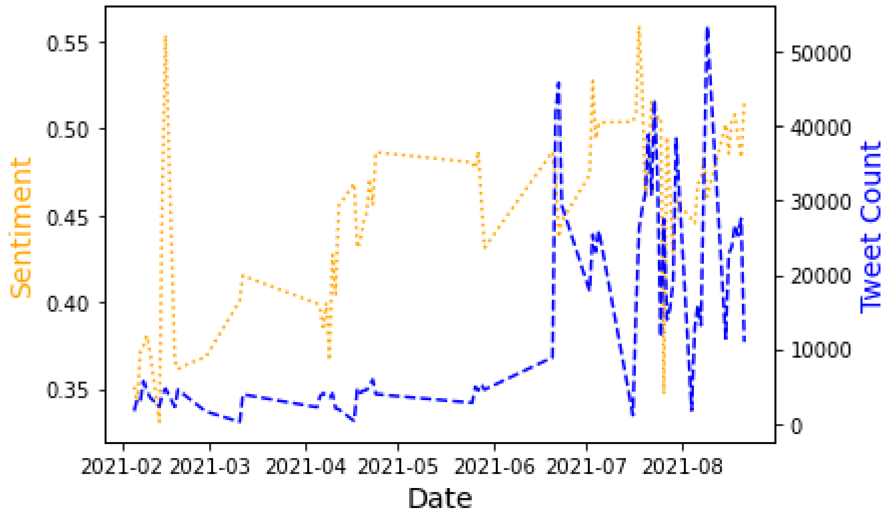
In [73]:
```
# create figure and axis objects with subplots()
fig,ax = plt.subplots()
```

```
# make a plot
ax.plot(price_df.Date, price_df.Close, color="red", linestyle='dotted')
# set x-axis label
ax.set_xlabel("Date", fontsize = 10)
# set y-axis label
ax.set_ylabel("Price", color="red", fontsize=10)
plt.xticks(rotation=45)

# twin object for two different y-axis on the sample plot
ax2=ax.twinx()
# make a plot with different y-axis using second axis object
ax2.plot(count.date, count.sentiment, color="blue", alpha=0.5)
ax2.set_ylabel("Tweet Count",color="blue",fontsize=10)

# title, show and save figure
plt.title('Tweet Volume vs Bitcoin Price')
plt.show()
plt.savefig('tweet_volume_vs_bitcoin_px.png')
```



Sentiment overlayed with Tweet Volume:

In [79]:
```
# create figure and axis objects with subplots()
fig,ax = plt.subplots()
# make a plot
ax.plot(grouped.date, grouped.sentiment,color="orange", linestyle='dotted')
# set x-axis label
ax.set_xlabel("Date", fontsize = 14)
# set y-axis label
ax.set_ylabel("Sentiment", color="orange",fontsize=14)

# twin object for two different y-axis on the sample plot
ax2=ax.twinx()
# make a plot with different y-axis using second axis object
ax2.plot(count.date, count.sentiment, color="blue", linestyle='dashed')
ax2.set_ylabel("Tweet Count",color="blue",fontsize=14)
plt.show()
```



Now, we look at the most common words, across all tweets, then positive & negative separately. There doesn't seem to be much of a distinction between positive and negative tweets in terms of the most frequent words. Perhaps bigrams would be more helpful to look at.

In [48]:
```
# distribution of most frequent words
freq_dist = FreqDist(text_data["text_without_stopwords"].explode())
```

In [49]:
```
# converting this into a dataframe
freq_dist_df = pd.DataFrame(pd.Series(freq_dist), columns=['count'])
freq_dist_df.head()
```

Out[49]:

	count
blue	2643
ridge	67
bank	6923
shares	3063
halted	39

In [50]:
```
# how does subsetting our text data by frequency reduce the set of words?

display(freq_dist_df.sort_values(by='count', ascending=False))

print("Total number of unique words: ", len(freq_dist_df))
print("Words appearing more than once: ", len(freq_dist_df[freq_dist_df['count'
print("Words appearing five or more times: ", len(freq_dist_df[freq_dist_df['cc
print("Words appearing 10 or more times: ", len(freq_dist_df[freq_dist_df['cour
print("Words appearing 50 or more times: ", len(freq_dist_df[freq_dist_df['cour
print("Words appearing 5000 or more times: ", len(freq_dist_df[freq_dist_df['cc
```

	count
bitcoin	1421651
crypto	449316
eth	249670
project	145770
doge	122002
...	...
leprechauns	1
banshees	1
uspresidentjoebiden	1
section230	1
crao	1

215464 rows × 1 columns

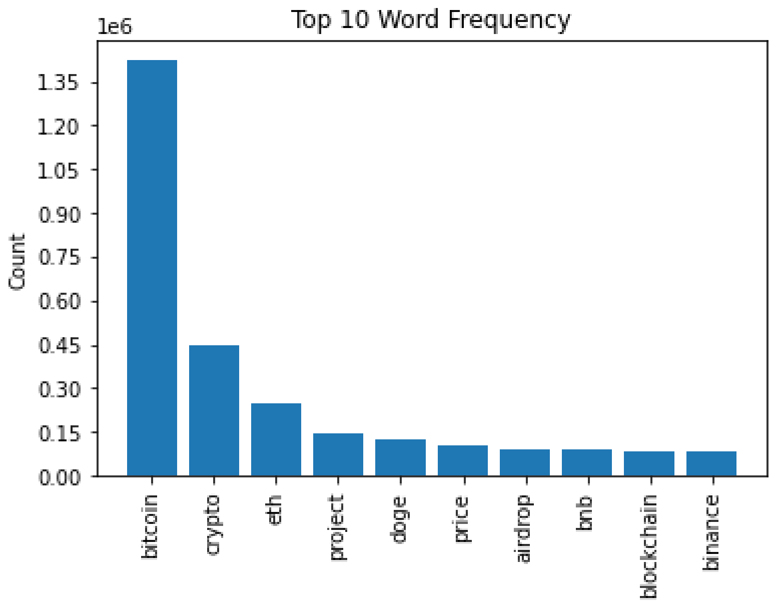
```
Total number of unique words:  215464
Words appearing more than once:  101721
Words appearing five or more times:  53081
Words appearing 10 or more times:  35318
Words appearing 50 or more times:  14612
Words appearing 5000 or more times:  450
```

In [51]:
```
def visualize_top_10(freq_dist, title):

    '''
    Function to visualize the top 10 most common words
    '''
    # data gathering
    top_10 = list(zip(*freq_dist.most_common(10)))
    tokens = top_10[0]
    counts = top_10[1]

    # plotting
    fig, ax = plt.subplots()
    ax.bar(tokens, counts)
    ax.set_title(title)
    ax.set_ylabel("Count")
    ax.yaxis.set_major_locator(MaxNLocator(integer=True))
    ax.tick_params(axis="x", rotation=90)
```

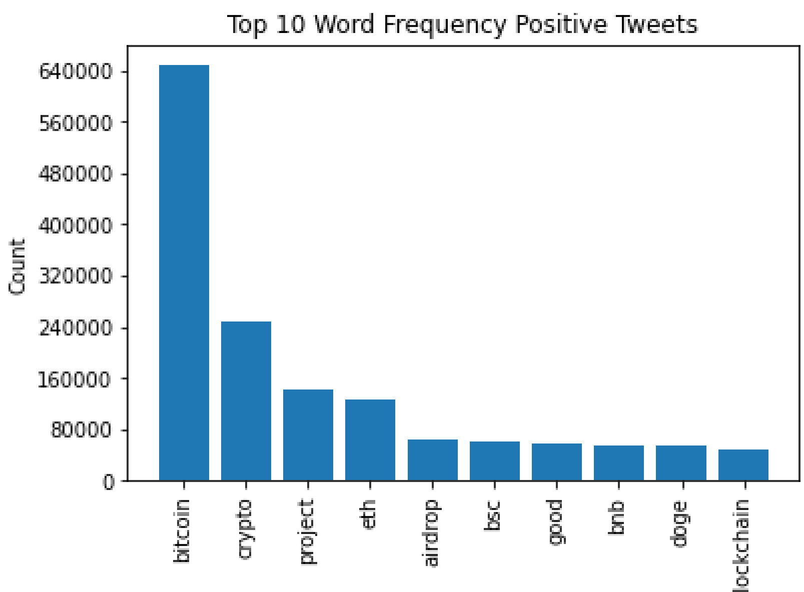
In [52]:
```
def visualize_top_20(freq_dist, title):

    '''
    Function to visualize the top 20 most common words
    '''

    # data gathering
    top_20 = list(zip(*freq_dist.most_common(20)))
    tokens = top_20[0]
    counts = top_20[1]

    # plotting
```

```
    fig, ax = plt.subplots()
    ax.bar(tokens, counts)
    ax.set_title(title)
    ax.set_ylabel("Count")
    ax.yaxis.set_major_locator(MaxNLocator(integer=True))
    ax.tick_params(axis="x", rotation=90)
```

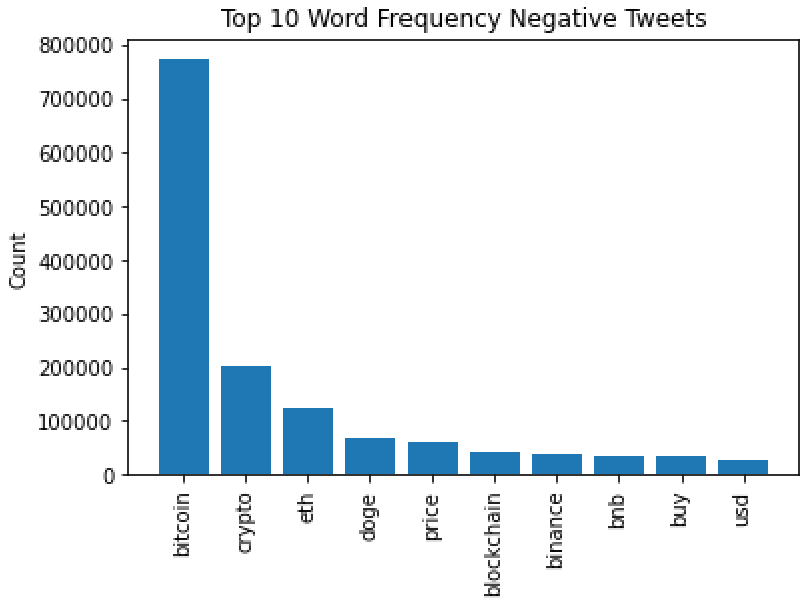
In [53]:
```
# top 10 word freq for all tweets
sample_freq_dist = FreqDist(text_data["text_without_stopwords"].explode())
visualize_top_10(sample_freq_dist, "Top 10 Word Frequency")
```



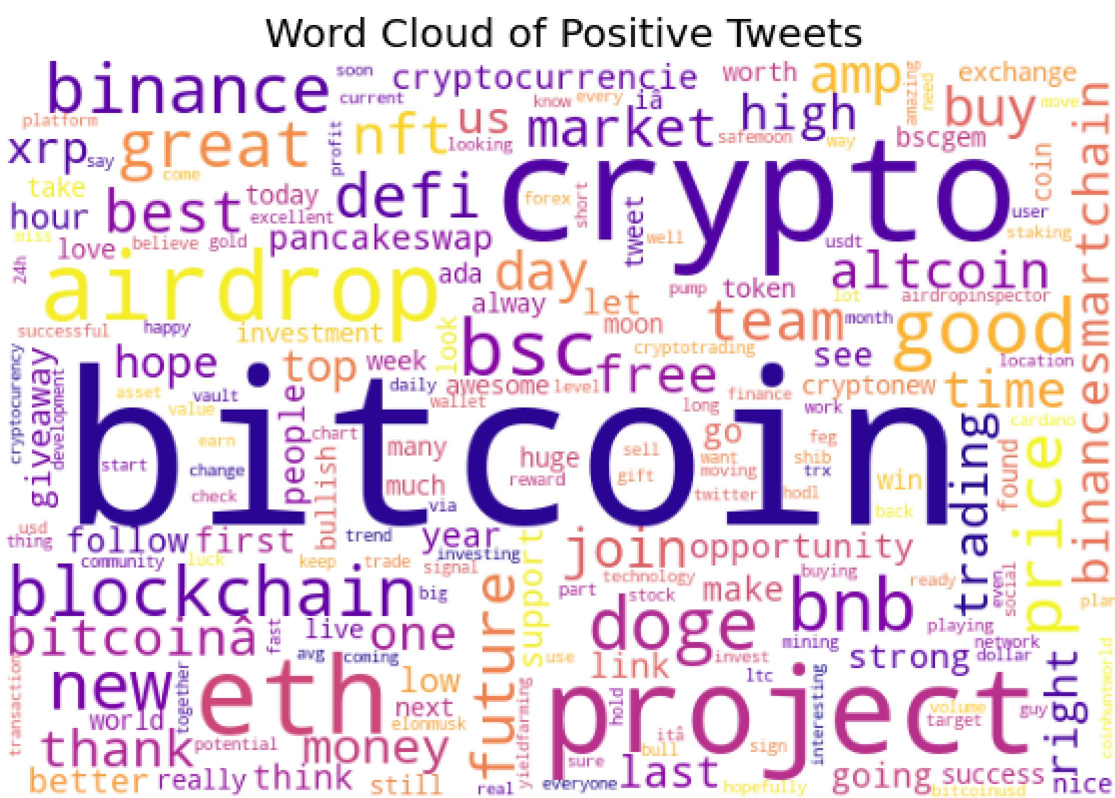
In [54]:
```
# top 10 word freq for positive tweets
sample_freq_dist = FreqDist(text_data["text_without_stopwords"][text_data['sent
visualize_top_10(sample_freq_dist, "Top 10 Word Frequency Positive Tweets")
```



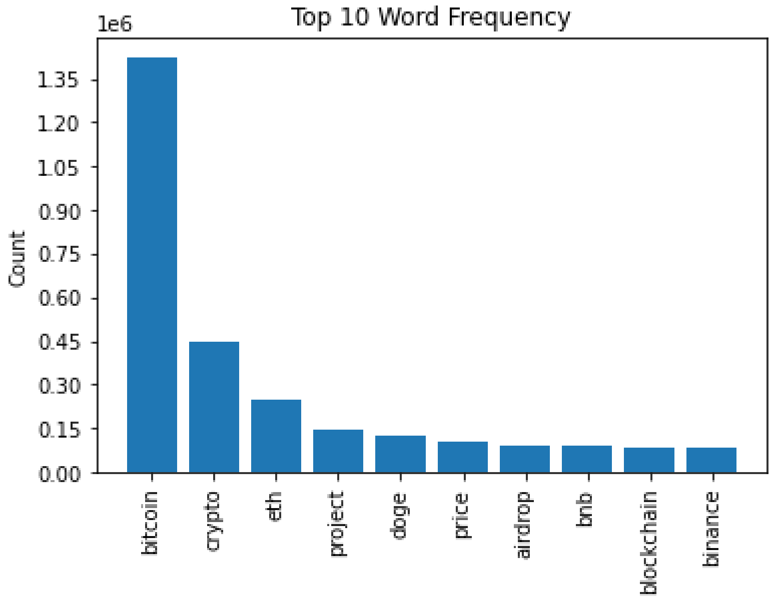
In [55]:
```
# top 10 word freq for negative tweets
sample_freq_dist = FreqDist(text_data["text_without_stopwords"][text_data['sent
visualize_top_10(sample_freq_dist, "Top 10 Word Frequency Negative Tweets")
```



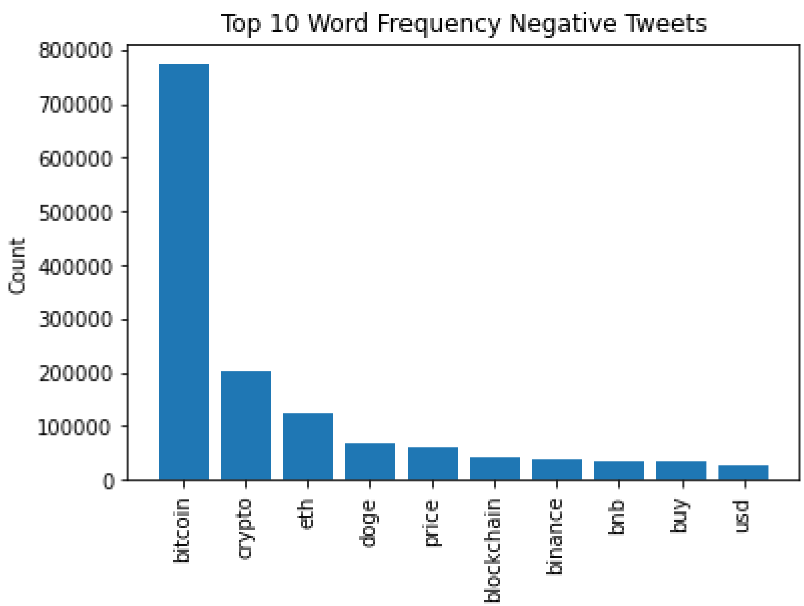
In [56]:
```python
# word cloud for positive sentiment tweets
wordcloud = WordCloud(max_words = 200, stopwords = None, collocations = False,
                      width = 600, height = 400, background_color = 'white',
                      colormap = 'plasma')
wordcloud.generate(",".join(text_data["text_without_stopwords"]\
                            [text_data['sentiment']==1].explode()))

plt.figure(figsize = (10, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis('off')
plt.title('Word Cloud of Positive Tweets', fontsize=20)
plt.savefig('images/positive_wordcloud.png')
```



In [57]:
```
# top 10 word freq for all tweets
sample_freq_dist = FreqDist(text_data["text_without_stopwords"].explode())
visualize_top_10(sample_freq_dist, "Top 10 Word Frequency")
```

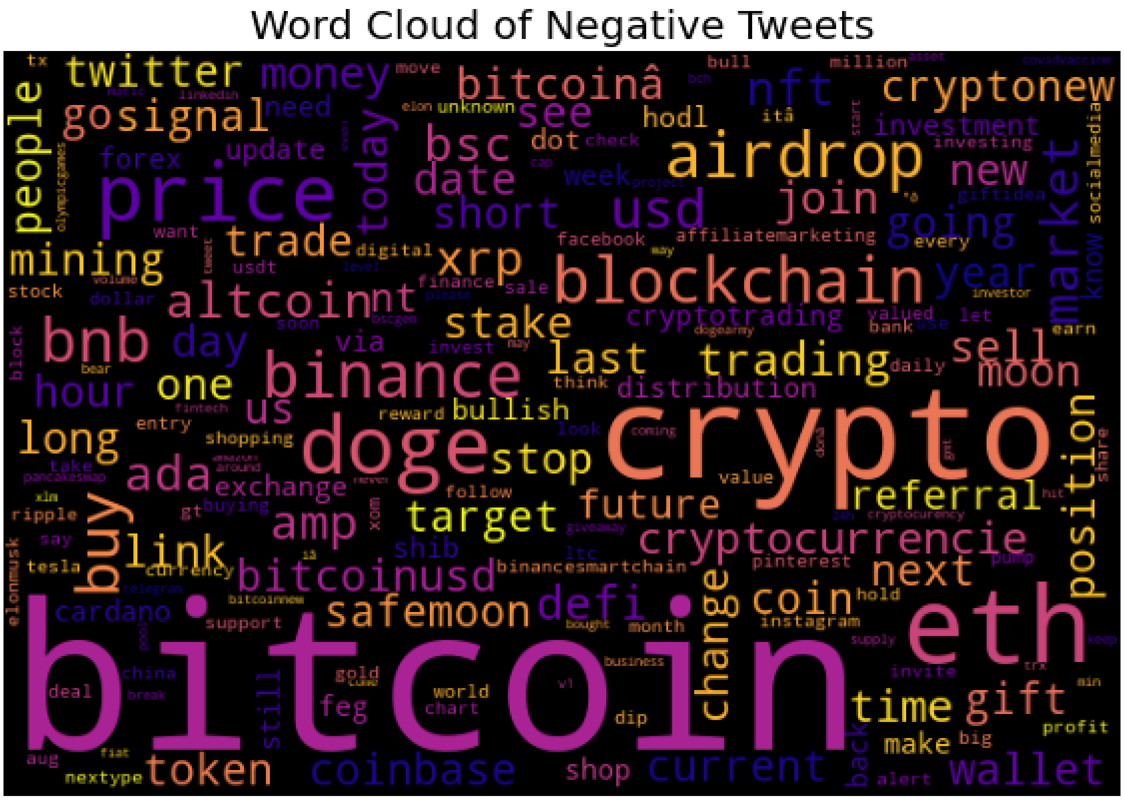


In [58]:
```
# top 10 word freq for positive tweets
sample_freq_dist = FreqDist(text_data["text_without_stopwords"][text_data['sent
visualize_top_10(sample_freq_dist, "Top 10 Word Frequency Negative Tweets")
```



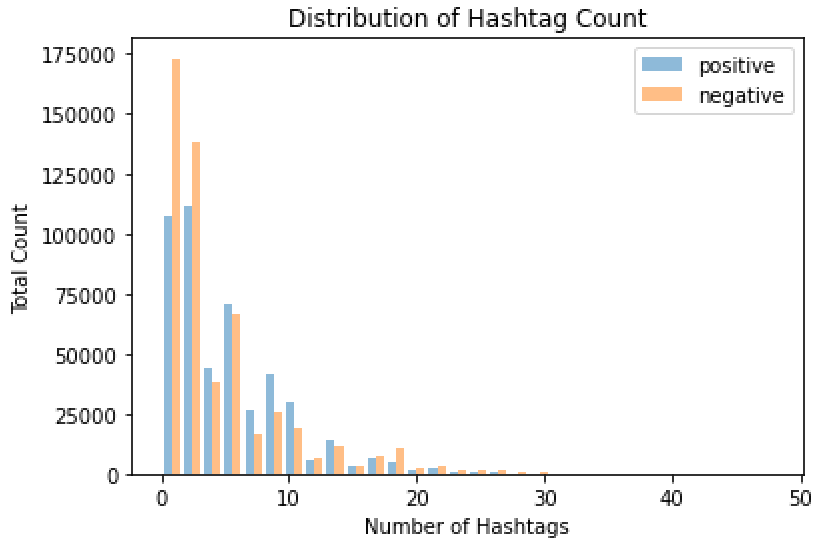
In [59]:
```
# word cloud for negative sentiment tweets
wordcloud = WordCloud(max_words = 200, stopwords = None, collocations = False,
                      width = 600, height = 400, background_color = 'black',
                      colormap = 'plasma')

wordcloud.generate(",".join(map(str,text_data["text_without_stopwords"]\
                                [text_data['sentiment']==0].explode())))

plt.figure(figsize = (10, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis('off')
```

```
plt.title('Word Cloud of Negative Tweets', fontsize=20)
plt.savefig('images/negative_wordcloud.png')
```



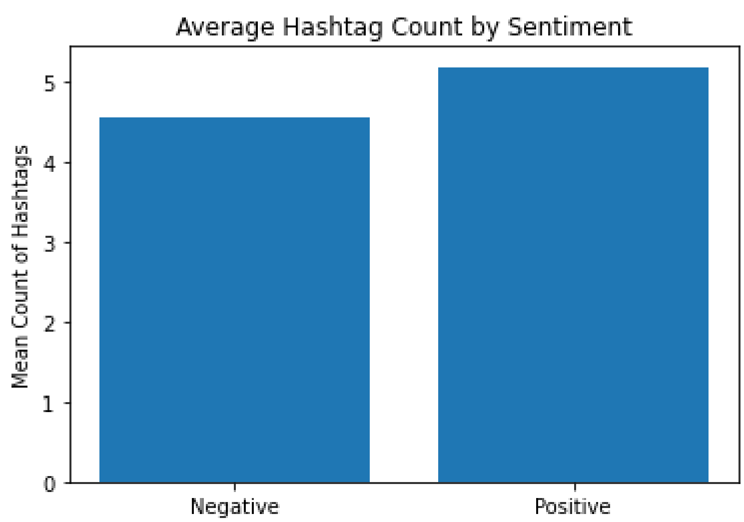
Based on the distribution of hashtag count across positive and negative tweets, it seems as though negative tweets tend to have more hashtags. Again, there doesn't seem to be much of a distinction between positive and negative tweets when it comes to the most frequent words used in hashtags.

In [60]:
```
# hashtag distribution
plt.hist([text_data['hashtag_count'][text_data['sentiment']==1],
          text_data['hashtag_count'][text_data['sentiment']==0]],
         bins=30, alpha=0.5, label=['positive', 'negative'])
plt.legend(loc='upper right')
plt.xlabel('Number of Hashtags')
plt.ylabel('Total Count')
plt.title('Distribution of Hashtag Count')
plt.show()
```

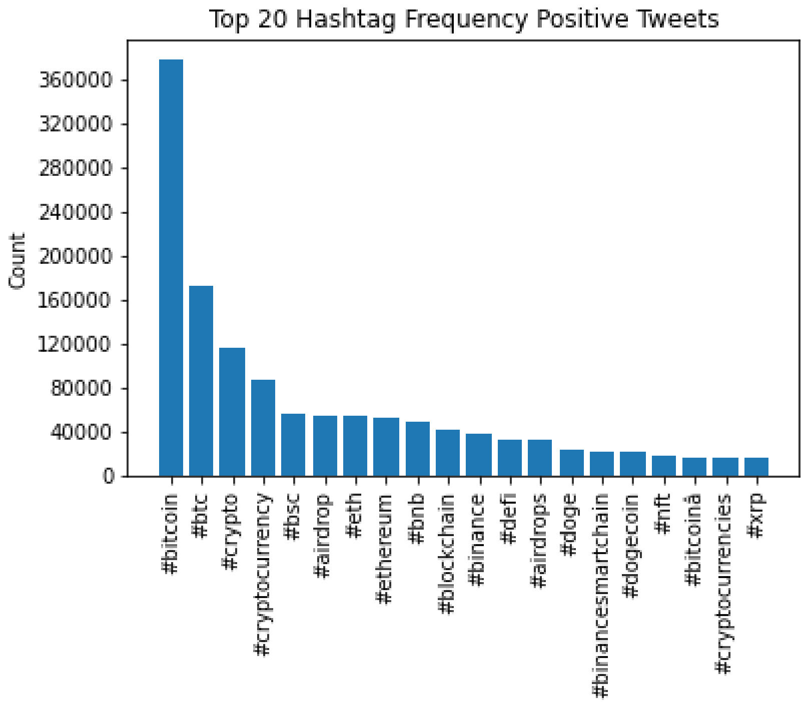


In [61]:
```
# grouping hashtag count by sentiment

h_count = text_data[['sentiment', 'hashtag_count']].groupby('sentiment').mean()
plt.bar(x='sentiment', height='hashtag_count', data=h_count, label=['Positive',
plt.xticks(h_count['sentiment'], ['Negative', 'Positive'])
plt.ylabel('Mean Count of Hashtags')
plt.title('Average Hashtag Count by Sentiment')
```

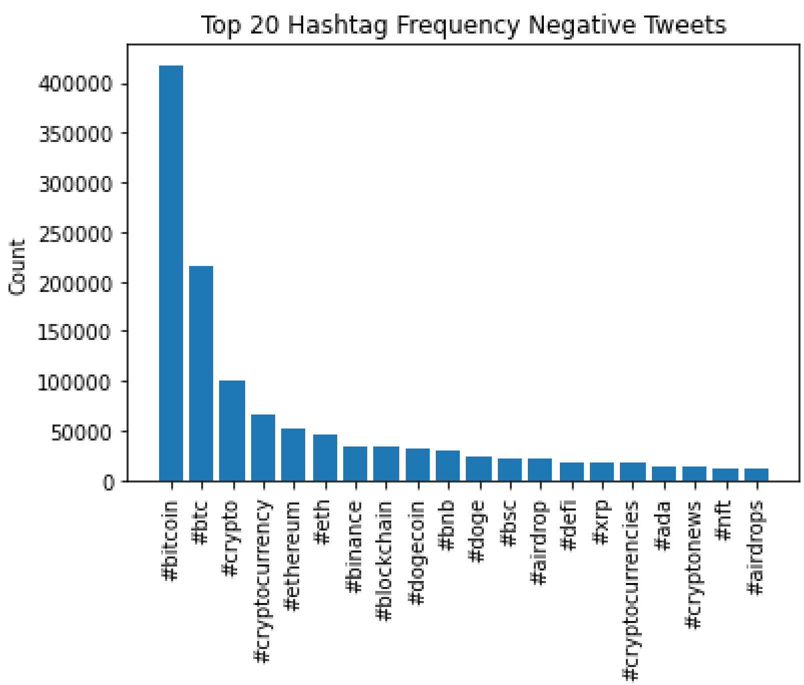
Out[61]:
```
Text(0.5, 1.0, 'Average Hashtag Count by Sentiment')
```



In [62]:
```
# top 20 hashtag freq for positive tweets
sample_freq_dist = FreqDist(text_data["hashtag"][text_data['sentiment']==1].exp
visualize_top_20(sample_freq_dist, "Top 20 Hashtag Frequency Positive Tweets")
```



In [63]:
```
# top hashtags for negative tweets
sample_freq_dist = FreqDist(text_data["hashtag"][text_data['sentiment']==0].exp
visualize_top_20(sample_freq_dist, "Top 20 Hashtag Frequency Negative Tweets")
```



None of the tweets are retweets, it seems. We can get rid of this column.

In [64]:
```
# is retweet grouped by sentiment

r_count = text_data[['sentiment', 'is_retweet']].groupby('is_retweet').count().
r_count # no retweets
```

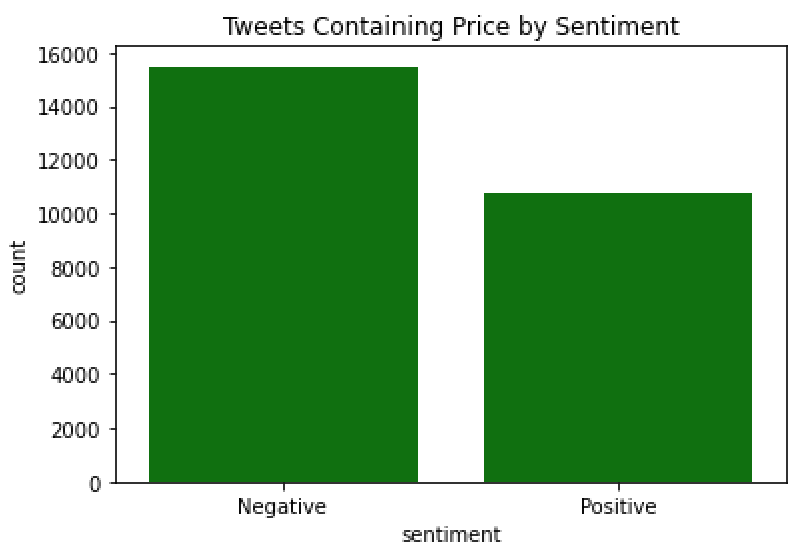
Out[64]:

	is_retweet	sentiment
0	False	967198
1	False	32745
2	n/a	38

In [65]:
```
# dropping is_retweet column since the only values are False and N/A
text_data = text_data.drop('is_retweet', axis=1)
```

Finally, a plot looking at the tweets that contain a price value. Negative tweets tend to have a price more frequently than positive tweets.

In [26]:
```
g = sns.countplot(x=text_data['sentiment'][text_data['contains_price']==1],
                  label=['Positive', 'Negative'], color='g')
g.set_title("Tweets Containing Price by Sentiment")
g.set_xticklabels(['Negative', 'Positive'])
plt.savefig('images/contains_price.png')
```



Preparing to Model

Here, I created a number of functions to help streamline the modeling process.

In [67]:
```
'''
Function that stores relevant scoring metrics
'''

def return_scores(y_train, y_hat_train, y_test, y_hat_test):
    #r_train = recall_score(y_train, y_hat_train)
    #r_test = recall_score(y_test, y_hat_test)

    f1_train = f1_score(y_train, y_hat_train)
    f1_test = f1_score(y_test, y_hat_test)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_h
    roc_auc = auc(false_positive_rate, true_positive_rate)

    pr_auc = average_precision_score(y_test, y_hat_test)
```

```
    ac_train = accuracy_score(y_train, y_hat_train)
    ac_test = accuracy_score(y_test, y_hat_test)

    return f1_train, f1_test, ac_train, ac_test, roc_auc, pr_auc
```

In [68]:
```
## ADD RUNTIME COLUMN? ##

'''
Creates a data frame with various scores for each model
'''
# column names
scores = pd.DataFrame(columns = ['f1_train', 'f1_test', 'accuracy_train',
                                 'accuracy_test', 'roc_auc', 'pr_auc',
                                 'model', 'vectorizer'])


def scoreTable(model, model_name, vectorizer, y_train, y_hat_train, y_test, y_h

    # storing scores
    f1_train, f1_test, ac_train, ac_test, roc_auc, pr_auc = \
    return_scores(y_train, y_hat_train, y_test, y_hat_test)

    # list of scores
    score_list = []
    score_list.extend((f1_train, f1_test, ac_train, ac_test, roc_auc, pr_auc,
                       str(model), str(vectorizer)))

    # adding scores to score table
    scores.loc[model_name] = score_list
    return scores
```

In [69]:
```
'''
Putting it all together, function that runs and evaluates model
'''

def model_eval(model, model_name, vectorizer):

    X_train_vectorized = vectorizer.fit_transform(X_train['text'])
    model.fit(X_train_vectorized, y_train)

    X_test_vectorized = vectorizer.transform(X_test['text'])

    y_hat_train = model.predict(X_train_vectorized)
    y_hat_test = model.predict(X_test_vectorized)

    col_names = vectorizer.get_feature_names()

    return scoreTable(model, model_name, vectorizer, y_train, y_hat_train,
                      y_test, y_hat_test)

# could try using pipeline here next
```

In [70]:
```
'''
Function that runs, evaluates model and stores variables
'''

def model_eval_store(model, model_name, vectorizer):
```

```
    X_train_vectorized = vectorizer.fit_transform(X_train['text'])
    model.fit(X_train_vectorized, y_train)

    X_test_vectorized = vectorizer.transform(X_test['text'])

    y_hat_train = model.predict(X_train_vectorized)
    y_hat_test = model.predict(X_test_vectorized)

    col_names = vectorizer.get_feature_names()

    return X_train_vectorized, X_test_vectorized, y_hat_train, y_hat_test, col_
    scoreTable(model, model_name, vectorizer, y_train, y_hat_train, y_test, y_h
```

Train Test Split

Splitting the data into train / test sets for model evaluation.

In [71]:
```
text_data.head()
```

Out[71]:

	text	date	sentiment	text_tokenized	text2	hashtag	text_without_stopwords	h
0	blue ridge bank shares halted by nyse after #b...	2021-02-10	0.0	[blue, ridge, bank, shares, halted, by, nyse, ...	blue ridge bank shares halted by nyse after bi...	[#bitcoin]	[blue, ridge, bank, shares, halted, nyse, bitc...	
1	ðŸ˜ today, that's this #thursday, we will do ...	2021-02-10	0.0	[today, that, this, thursday, we, will, do, ta...	today that this thursday we will do take with ...	[#thursday, #btc, #wallet, #security]	[today, thursday, take, friend, bitcoin, walle...	
2	guys evening, i have read this article about b...	2021-02-10	0.0	[guys, evening, have, read, this, article, abo...	guys evening have read this article about bitc...	[]	[guys, evening, read, article, bitcoin, would,...	
3	$btc a big chance in a billion! price: \. (// ...	2021-02-10	0.0	[bitcoin, big, chance, in, billion, price, bit...	bitcoin big chance in billion price bitcoin fx...	[#bitcoin, #fx, #btc, #crypto]	[bitcoin, big, chance, billion, price, bitcoin...	
4	this network is secured by nodes as of today...	2021-02-10	0.0	[this, network, is, secured, by, nodes, as, of...	this network is secured by nodes as of today s...	[#btc]	[network, secured, nodes, today, soon, biggest...	

In [3]:
```
# saving to csv file so that we do not have to rerun all of the above code
# if we restart the kernel. uncomment if any changes to the above made
#text_data.to_csv('data/text_data_preprocessed.csv')

# uncomment the following if kernel is restarted
text_data = pd.read_csv('data/text_data_preprocessed.csv')
```

In [73]:
```
X = text_data.drop('sentiment', axis=1)
y = text_data['sentiment']

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

In [74]:
```
print(y_train.value_counts(normalize=True)) #fairly balanced
print(y_test.value_counts(normalize=True))
```

```
0.0    0.527191
1.0    0.472809
Name: sentiment, dtype: float64
0.0    0.526648
1.0    0.473352
Name: sentiment, dtype: float64
```

Models

Baseline Model

Our baseline model is a Naive Bayers classifier with a TF-IDF Vectorizer. Resulting in 81% accuracy, I hope to make some improvements with different vectorizers and classifiers.

In [75]:
```
# Instantiate a MultinomialNB classifier as baseline model
baseline_model = MultinomialNB()
```

In [76]:
```
tfidf = TfidfVectorizer(max_features=450) # includes words that occur 5000 or m

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

# Visually inspecting the matrix
pd.DataFrame.sparse.from_spmatrix(X_train_vectorized, columns=tfidf.get_feature
```

Out[76]:

	24h	_binance	_saylor	about	above	ada	address	affiliatemarketing	after	again
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
749980	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
749981	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
749982	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
749983	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
749984	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

749985 rows × 450 columns

In [77]:
```
# We should still have the same number of rows
assert X_train_vectorized.shape[0] == X_train.shape[0]

# The vectorized version should have 450 columns, since max_features=450
assert X_train_vectorized.shape[1] == 450
```

In [78]:
```
# Evaluate the classifier on X_train_vectorized and y_train
baseline_cv = cross_val_score(baseline_model, X_train_vectorized, y_train)
```

```
baseline_cv
```

Out[78]:
```
array([0.80949619, 0.81026954, 0.8099162 , 0.80902285, 0.80762282])
```

In [79]:
```
# add baseline to score table

# fit on train, transform test
baseline_model.fit(X_train_vectorized, y_train)
X_test_vectorized = tfidf.transform(X_test['text'])

# store y_hats
y_hat_train_baseline = baseline_model.predict(X_train_vectorized)
y_hat_test_baseline = baseline_model.predict(X_test_vectorized)

# add to score table
display(scoreTable(baseline_model, 'baseline', tfidf, y_train,
                   y_hat_train_baseline, y_test, y_hat_test_baseline))
```

	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc	pr_auc	model
baseline	0.786558	0.786891	0.809624	0.809601	0.806211	0.74324	MultinomialNB()

Vectorizer Tuning

First, I will look at how changing the vectorizer affects the performance of the baseline model.

In [80]:
```
# running different vectorizers on same model, hypertuning vectorizers
runs = [(MultinomialNB(), 'baseline_tfidf2', TfidfVectorizer(max_features=450,
        (MultinomialNB(), 'baseline_tfidf3', TfidfVectorizer(stop_words=stopwor
        (MultinomialNB(), 'baseline_tfidf4', TfidfVectorizer(ngram_range=(2,2))
        (MultinomialNB(), 'baseline_tfidf5', TfidfVectorizer(ngram_range=(2,2),
        (MultinomialNB(), 'baseline_tfidf6', TfidfVectorizer(max_features=5000,
        (MultinomialNB(), 'baseline_cv', CountVectorizer(max_features=5000, sto
        (MultinomialNB(), 'baseline_cv2', CountVectorizer(stop_words=stopwords_
        (MultinomialNB(), 'baseline_cv3', CountVectorizer(ngram_range=(2,2))),
        (MultinomialNB(), 'baseline_cv4', CountVectorizer(stop_words=stopwords_
       ]
```

Count Vectorizer is clearly the outperforming vectorizer. We will use that for our models and tune the hyperparameters for the vectorizer a bit.

In [81]:
```
for run in runs:
    model_eval(*run)

display(scores)
```

	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc	pr_auc	
baseline	0.786558	0.786891	0.809624	0.809601	0.806211	0.743240	Multino
baseline_tfidf2	0.807107	0.807663	0.828027	0.828113	0.824789	0.767082	Multino
baseline_tfidf3	0.878890	0.857826	0.889078	0.869726	0.867730	0.817014	Multino
baseline_tfidf4	0.943631	0.865710	0.947507	0.875794	0.874275	0.822871	Multino
baseline_tfidf5	0.952206	0.850712	0.955678	0.866042	0.863020	0.817584	Multino
baseline_tfidf6	0.853068	0.851155	0.866793	0.864946	0.862458	0.813036	Multino
baseline_cv	0.860531	0.858882	0.870629	0.868974	0.867627	0.812584	Multino
baseline_cv2	0.886531	0.868851	0.893919	0.877126	0.876252	0.821322	Multino
baseline_cv3	0.958314	0.867798	0.960734	0.875402	0.874822	0.817498	Multino
baseline_cv4	0.967555	0.858882	0.969429	0.869114	0.867714	0.813043	Multino

In [82]:
```
# Top 3 are TF 1000, CV 1000 and CV 5000 with stopwords
```

In [83]:
```
# running a few more, focusing on reducing features

runs = [(MultinomialNB(), 'tf_1000', TfidfVectorizer(max_features=1000, stop_wo
        (MultinomialNB(), 'tf_5000', TfidfVectorizer(max_features=5000, stop_wo
        #(MultinomialNB(), 'tf_all', TfidfVectorizer(max_features=450, stop_wor
        (MultinomialNB(), 'cv_1000', CountVectorizer(max_features=1000, stop_wo
        (MultinomialNB(), 'cv_5000', CountVectorizer(max_features=5000, stop_wo
        #(MultinomialNB(), 'cv_all', CountVectorizer(max_features=450, stop_wor
       ]

for run in runs:
    model_eval(*run)

display(scores)
```

	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc	pr_auc	
baseline	0.786558	0.786891	0.809624	0.809601	0.806211	0.743240	Multino
baseline_tfidf2	0.807107	0.807663	0.828027	0.828113	0.824789	0.767082	Multino
baseline_tfidf3	0.878890	0.857826	0.889078	0.869726	0.867730	0.817014	Multino
baseline_tfidf4	0.943631	0.865710	0.947507	0.875794	0.874275	0.822871	Multino
baseline_tfidf5	0.952206	0.850712	0.955678	0.866042	0.863020	0.817584	Multino
baseline_tfidf6	0.853068	0.851155	0.866793	0.864946	0.862458	0.813036	Multino
baseline_cv	0.860531	0.858882	0.870629	0.868974	0.867627	0.812584	Multino
baseline_cv2	0.886531	0.868851	0.893919	0.877126	0.876252	0.821322	Multino
baseline_cv3	0.958314	0.867798	0.960734	0.875402	0.874822	0.817498	Multino
baseline_cv4	0.967555	0.858882	0.969429	0.869114	0.867714	0.813043	Multino
tf_1000	0.829989	0.830757	0.846330	0.846690	0.844070	0.788640	Multino
tf_5000	0.853068	0.851155	0.866793	0.864946	0.862458	0.813036	Multino
cv_1000	0.832415	0.832493	0.845976	0.845762	0.843937	0.783671	Multino
cv_5000	0.860531	0.858882	0.870629	0.868974	0.867627	0.812584	Multino

In [84]:
```
# storing these results in a new variable so as not to lose them
scores_vec = scores
```

Model Tuning

Now that we have selected a best-performing vectorizer, we will run a number of different classifiers. The Random Forest Classifier had an hour+ runtime so I am commenting this out for now. It was not our top model, with only about 90% accuracy and F1 scores.

In [85]:
```
# testing different models using model_eval function, with best vectorizer

# resetting score table, only keeping certain rows
scores = scores.loc[['baseline', 'cv_5000', 'tf_1000', 'tf_5000']]

# testing best vectorizer on different classifiers
runs = [(LogisticRegression(), 'logreg', CountVectorizer(stop_words = stopwords
        (LinearSVC(), 'svc', CountVectorizer(stop_words = stopwords_list, max_f
        (LinearSVC(), 'svc_tf', TfidfVectorizer(stop_words = stopwords_list, ma
        #(RandomForestClassifier(), 'rf', CountVectorizer(stop_words = stopword

for run in runs:
    model_eval(*run)
```

```
display(scores)
```

```
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/s
klearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to conv
erge (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-regress
ion
  n_iter_i = _check_optimize_result(
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/s
klearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to converge, inc
rease the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc	pr_auc	
baseline	0.786558	0.786891	0.809624	0.809601	0.806211	0.743240	Multinomi
cv_5000	0.860531	0.858882	0.870629	0.868974	0.867627	0.812584	Multinomi
tf_1000	0.829989	0.830757	0.846330	0.846690	0.844070	0.788640	Multinomi
tf_5000	0.853068	0.851155	0.866793	0.864946	0.862458	0.813036	Multinomi
logreg	0.975459	0.975219	0.976928	0.976660	0.976336	0.965158	LogisticRegres
svc	0.975020	0.975202	0.976505	0.976636	0.976327	0.964988	Linear
svc_tf	0.919939	0.921505	0.927123	0.928391	0.926347	0.903400	Linear

Uncomment the following if you want to run the (lengthy) random forest classifiers:

In [86]:
```
#model_eval(RandomForestClassifier(), 'rf', CountVectorizer(stop_words = stopwo
#model_eval(RandomForestClassifier(), 'rf_5000', CountVectorizer(stop_words = s
```

Now, we will hypertune the top 2 models:

In [87]:
```
runs = [(LogisticRegression(solver='saga', max_iter=200), 'logreg2',
         CountVectorizer(stop_words = stopwords_list, max_features=5000)),
        (LinearSVC(class_weight='balanced'), 'svc2',
         CountVectorizer(stop_words = stopwords_list, max_features=5000))]

for run in runs:
    model_eval(*run)

display(scores)
```

```
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/s
klearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/s
klearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to converge, inc
rease the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc	pr_auc	
baseline	0.786558	0.786891	0.809624	0.809601	0.806211	0.743240	
cv_5000	0.860531	0.858882	0.870629	0.868974	0.867627	0.812584	
tf_1000	0.829989	0.830757	0.846330	0.846690	0.844070	0.788640	
tf_5000	0.853068	0.851155	0.866793	0.864946	0.862458	0.813036	
logreg	0.975459	0.975219	0.976928	0.976660	0.976336	0.965158	
svc	0.975020	0.975202	0.976505	0.976636	0.976327	0.964988	
svc_tf	0.919939	0.921505	0.927123	0.928391	0.926347	0.903400	
logreg2	0.973500	0.973493	0.975072	0.975028	0.974712	0.962488	LogisticReg
svc2	0.975031	0.975286	0.976494	0.976696	0.976430	0.964715	LinearSVC(cla

Now we will add in the engineered features to see if that improves model performance. I will need to limit max features to 500 for the concatination process as it is very RAM-intensive.

In [88]:
```python
# baseline for comparison purposes & storing initial X_train/X_test
X_train_vec, X_test_vec, y_hat_train, y_hat_test, col_names, score_table  = \
model_eval_store(LogisticRegression(), 'logreg_500',
                 CountVectorizer(stop_words = stopwords_list, max_features=500)
```

```
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/s
klearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to conv
erge (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-regress
ion
  n_iter_i = _check_optimize_result(
```

Adding in engineered features to X_train for final training dataset:

In [89]:
```python
# converting to dense matrix
X_train_vec_df = pd.DataFrame(X_train_vec.todense(), columns=col_names)
```

```
# label encoding the contains_price column, which was formerly boolean
encoder = LabelEncoder()
contains_price_train = pd.Series(encoder.fit_transform(X_train['contains_price'
                                 name='contains_price')

# combining all features into one final X_train dataset
X_train_feats = pd.concat([X_train_vec_df,
                           contains_price_train,
                           X_train[['hashtag_count']].reset_index(drop=True),
                           X_train[['tweet_length']].reset_index(drop=True)
                          ], axis=1)

# visually inspecting final_X_train
X_train_feats.head()
```

Out[89]:

	20th	24h	30k	40k	50k	_binance	_saylor	account	ada	address	...	year	years	yet
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0

5 rows × 503 columns

Doing the same for X_test:

In [90]:
```
X_test_vec_df = pd.DataFrame(X_test_vec.todense(), columns=col_names)

contains_price_test = pd.Series(encoder.transform(X_test['contains_price']),
                                name='contains_price')

X_test_feats = pd.concat([X_test_vec_df,
                          contains_price_test,
                          X_test[['hashtag_count']].reset_index(drop=True),
                          X_test[['tweet_length']].reset_index(drop=True)
                         ], axis=1)
```

Fitting and scoring the model with the final training and test data. Adding these features actually reduces accuracy and F1. Additionally, none of the added features show up in the top 20 or bottom 20 in terms of regression coefficients, meaning they are not that impactful to the model.

In [91]:
```
vec = CountVectorizer(max_features=500, stop_words = stopwords_list)
final_model = LogisticRegression()

# fitting on final X train
final_model.fit(X_train_feats, y_train)

# printing accuracy for both train and test data
print(final_model.score(X_train_feats, y_train))
print(final_model.score(X_test_feats, y_test))

# final_model preds
```

```
y_hat_train_final = final_model.predict(X_train_feats)
y_hat_test_final = final_model.predict(X_test_feats)

# printing score table
scoreTable(final_model, 'final_feat', vec, y_train, y_hat_train_final, y_test,
display(scores)
```

```
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/s
klearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to conv
erge (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-regress
ion
  n_iter_i = _check_optimize_result(
0.8853363733941345
0.8856701707227316
```

	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc	pr_auc	
baseline	0.786558	0.786891	0.809624	0.809601	0.806211	0.743240	
cv_5000	0.860531	0.858882	0.870629	0.868974	0.867627	0.812584	
tf_1000	0.829989	0.830757	0.846330	0.846690	0.844070	0.788640	
tf_5000	0.853068	0.851155	0.866793	0.864946	0.862458	0.813036	
logreg	0.975459	0.975219	0.976928	0.976660	0.976336	0.965158	
svc	0.975020	0.975202	0.976505	0.976636	0.976327	0.964988	
svc_tf	0.919939	0.921505	0.927123	0.928391	0.926347	0.903400	
logreg2	0.973500	0.973493	0.975072	0.975028	0.974712	0.962488	LogisticF
svc2	0.975031	0.975286	0.976494	0.976696	0.976430	0.964715	LinearSVC(
logreg_500	0.876133	0.877229	0.891312	0.892026	0.888125	0.861656	
final_feat	0.870518	0.871220	0.885336	0.885670	0.882196	0.849002	

The words in the following lists make a lot of sense to me. 'good', 'great', 'best' are among the words with the most positive coefficient in the regression, while 'sale', 'hard' and 'min' are among the words with the most negative coefficients. Remember that positive sentiment is represented as 1.0 and negative is 0, so positive coefficients contribute to the positive classification.

In [92]:
```
## Top coefficients
coef_df = pd.DataFrame(final_model.coef_, columns=X_train_feats.columns).transp
#coef_df.to_csv('coef_logreg.csv')

coef_df.columns=['coef']
coef_df['coef_abs'] = abs(coef_df['coef'])

coef_df.sort_values(by='coef_abs', ascending=False)[:20]
```

Out[92]:

	coef	coef_abs
good	6.385818	6.385818
free	5.330733	5.330733
great	5.225242	5.225242
best	5.162797	5.162797
top	4.833108	4.833108
worth	4.613380	4.613380
live	4.278694	4.278694
first	4.178541	4.178541
nice	4.025447	4.025447
new	3.973376	3.973376
love	3.916086	3.916086
right	3.884669	3.884669
win	3.773904	3.773904
high	3.528038	3.528038
better	3.503939	3.503939
many	3.465246	3.465246
thanks	3.455079	3.455079
ready	3.344462	3.344462
strong	3.311165	3.311165
real	3.203788	3.203788

In [93]:
```
coef_df.sort_values(by='coef', ascending=True)[:20]
```

Out[93]:

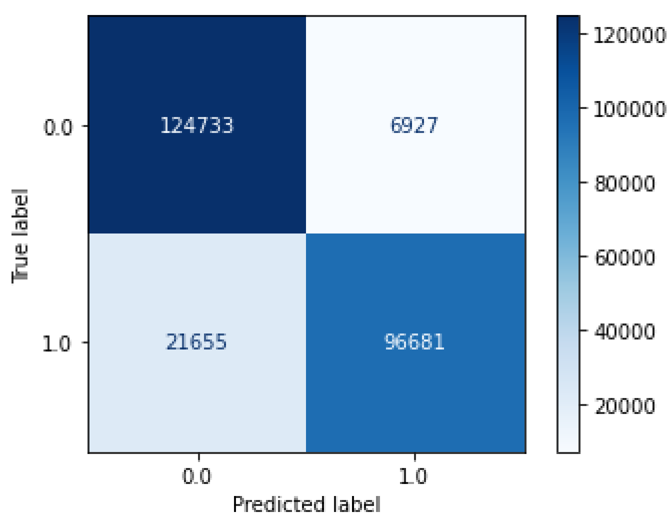
	coef	coef_abs
eshop	-1.705893	1.705893
stakes	-1.621810	1.621810
game	-1.531563	1.531563
tx	-1.286905	1.286905
hard	-1.210386	1.210386
min	-1.145056	1.145056
date	-1.110689	1.110689
play	-1.076156	1.076156
little	-0.962048	0.962048
invite	-0.939984	0.939984
unknown	-0.931374	0.931374
xom	-0.904324	0.904324
alert	-0.867881	0.867881
distribution	-0.867169	0.867169
valued	-0.800727	0.800727
sale	-0.792811	0.792811
pinterest	-0.791570	0.791570
moon	-0.760816	0.760816
nt	-0.753639	0.753639
shop	-0.736833	0.736833

In [94]:

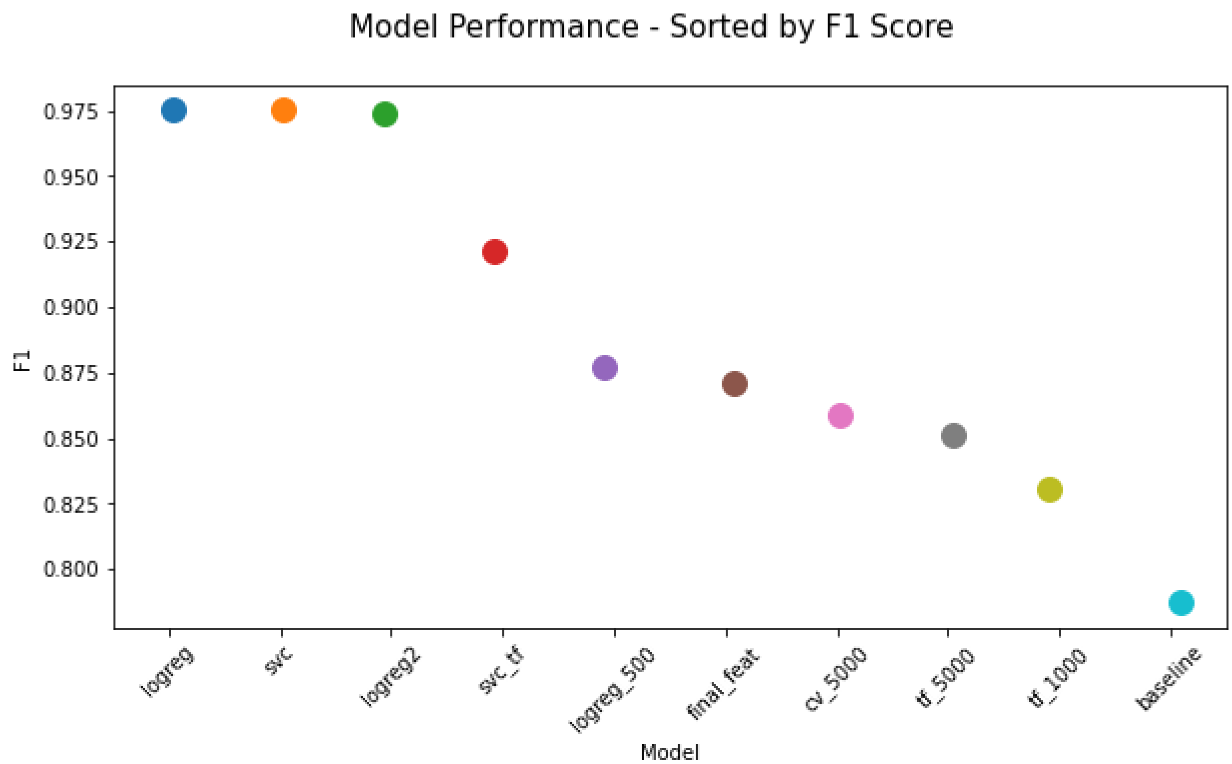
```
# plotting confusion matrix
plot_confusion_matrix(final_model, X_test_feats, y_test, cmap=plt.cm.Blues)
```

Out[94]:

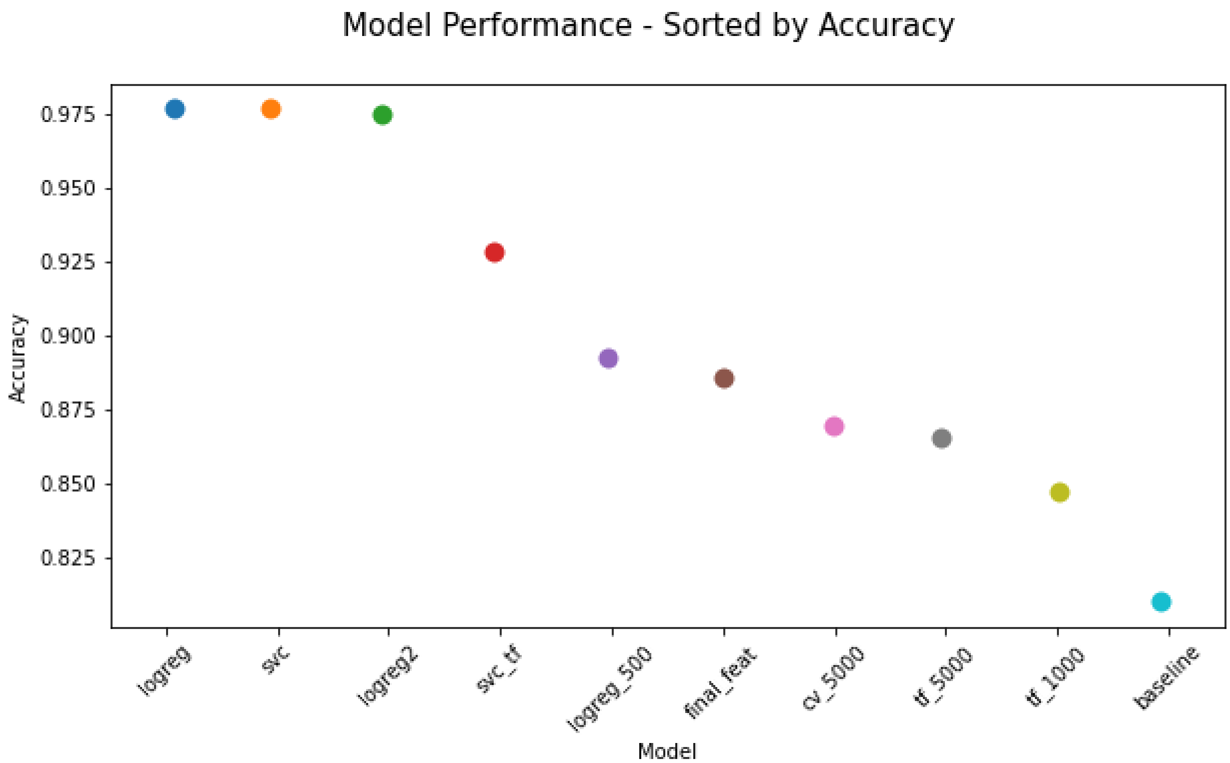
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f999db11d
30>
```



In [95]:
```
## Model Comparison - Sorted by F1 score
scores_viz_f1 = scores.sort_values(by=['f1_test'], ascending=False)[1:].reset_i

fig, ax = plt.subplots(figsize=(10, 5))
sns.stripplot(x="index", y="f1_test", data=scores_viz_f1, size=13)
plt.xticks(rotation = 45)
ax.set_xlabel("Model", fontsize=10)
ax.set_ylabel("F1", fontsize=10)
fig.suptitle("Model Performance - Sorted by F1 Score", fontsize=15)
plt.savefig('images/model_f1_comp.png')
```



In [96]:
```
## Model Comparison - Sorted by Accuracy
scores_viz_ac = scores.sort_values(by=['accuracy_test'], ascending=False)[1:].r

fig, ax = plt.subplots(figsize=(10, 5))
sns.stripplot(x="index", y="accuracy_test", data=scores_viz_ac, size=10)
plt.xticks(rotation = 45)
ax.set_xlabel("Model", fontsize=10)
ax.set_ylabel("Accuracy", fontsize=10)
fig.suptitle("Model Performance - Sorted by Accuracy", fontsize=15)
plt.savefig('images/model_ac_comp.png')
```



Final Model

The best-peforming model was a logistic regression classifier, with stopwords removed and max features capped at 5000. None of the engineered parameters improved performance significantly enough to include them in the classifier.

In [98]:
```
# storing the final X_train and X_test
final_X_train, final_X_test, y_hat_train, y_hat_test, col_names, score_table  =
model_eval_store(LogisticRegression(), 'final',
                 CountVectorizer(stop_words = stopwords_list, max_features=500(

display(score_table)
```

```
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/s
klearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to conv
erge (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-regress
ion
  n_iter_i = _check_optimize_result(
```

	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc	pr_auc	
baseline	0.786558	0.786891	0.809624	0.809601	0.806211	0.743240	
cv_5000	0.860531	0.858882	0.870629	0.868974	0.867627	0.812584	
tf_1000	0.829989	0.830757	0.846330	0.846690	0.844070	0.788640	
tf_5000	0.853068	0.851155	0.866793	0.864946	0.862458	0.813036	
logreg	0.975459	0.975219	0.976928	0.976660	0.976336	0.965158	
svc	0.975020	0.975202	0.976505	0.976636	0.976327	0.964988	
svc_tf	0.919939	0.921505	0.927123	0.928391	0.926347	0.903400	
logreg2	0.973500	0.973493	0.975072	0.975028	0.974712	0.962488	LogisticF
svc2	0.975031	0.975286	0.976494	0.976696	0.976430	0.964715	LinearSVC(
logreg_500	0.876133	0.877229	0.891312	0.892026	0.888125	0.861656	
final_feat	0.870518	0.871220	0.885336	0.885670	0.882196	0.849002	
final	0.975459	0.975219	0.976928	0.976660	0.976336	0.965158	

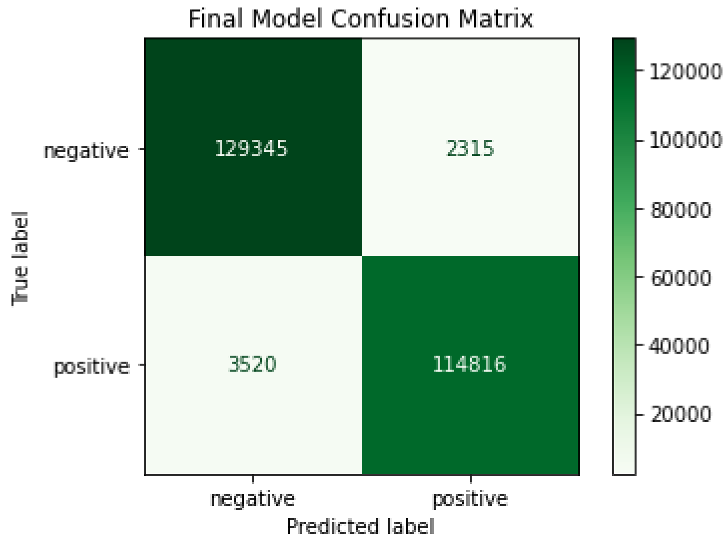
Let's look at some visualizations of the final model performance.

In [99]:
```
final_model = LogisticRegression()
final_model.fit(final_X_train, y_train)
```

```
/Users/natalyadoris/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/s
klearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to conv
erge (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-regress
ion
  n_iter_i = _check_optimize_result(
```

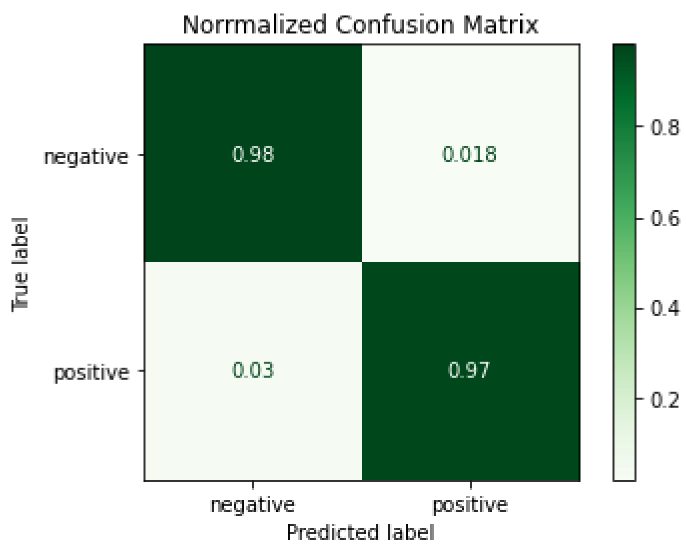
Out[99]:
```
LogisticRegression()
```

In [100…
```
# Confusion matrix
g = plot_confusion_matrix(final_model, final_X_test, y_test, cmap=plt.cm.Greens
g.ax_.set_title('Final Model Confusion Matrix')
g.ax_.xaxis.set_ticklabels(['negative', 'positive'])
g.ax_.yaxis.set_ticklabels(['negative', 'positive'])

plt.savefig('images/final_confusion_matrix.png')
```



In [101…

```
# Normalized Confusion Matrix
g = plot_confusion_matrix(final_model, final_X_test, y_test, cmap=plt.cm.Greens
                          normalize='true')
g.ax_.set_title('Norrmalized Confusion Matrix')
g.ax_.xaxis.set_ticklabels(['negative', 'positive'])
g.ax_.yaxis.set_ticklabels(['negative', 'positive'])

plt.savefig('images/final_confusion_matrix_normalized.png')
```



Plotting feature importance for the final model.

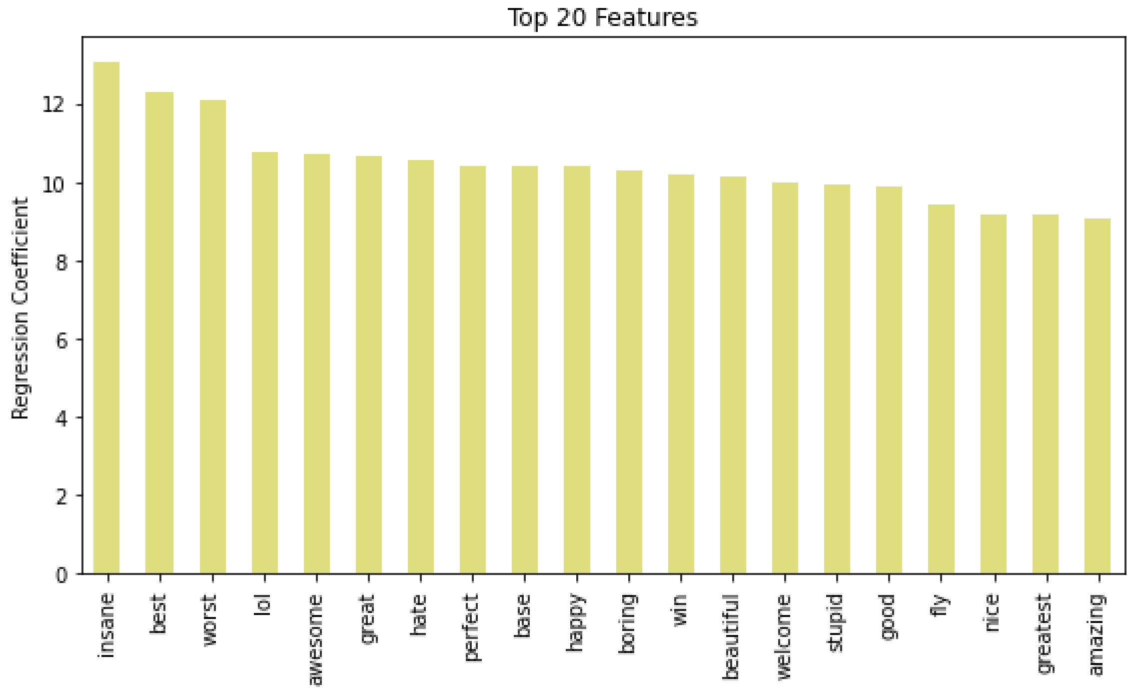
In [102…

```
# Dataframe of regression coefficients for each feature (word)
coef_df = pd.DataFrame(final_model.coef_, columns=col_names).transpose()
coef_df.columns=['coef']
coef_df['coef_abs'] = abs(coef_df['coef'])
```

In [103…

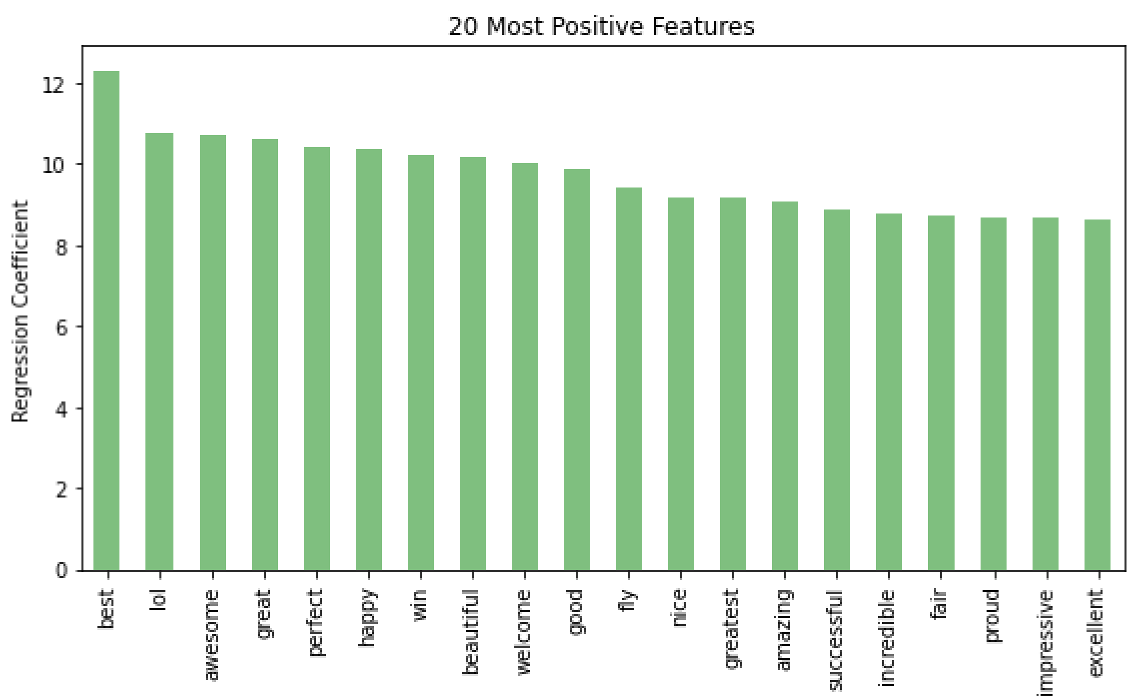
```
# Plotting the top 20 features (pos or neg)
top_20 = coef_df.sort_values(by='coef_abs', ascending=False)[:20]
fig, ax = plt.subplots(figsize = (8,5))
top_20['coef_abs'].plot.bar(ax=ax, color='y', alpha=0.5)
ax.set_title("Top 20 Features")
ax.set_ylabel("Regression Coefficient")
```

```
fig.tight_layout()
plt.savefig('images/feat_importance_top20.png')
```



In [104…

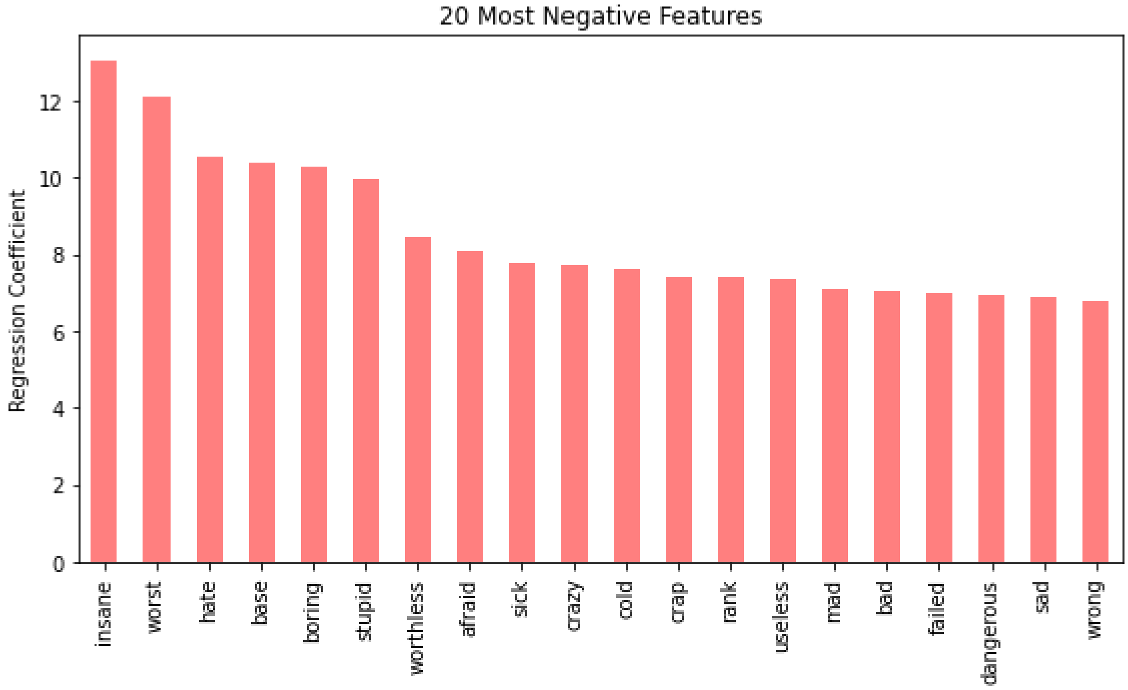
```
# Plotting the 20 most positive features
bottom_20 = coef_df.sort_values(by='coef', ascending=False)[:20]
fig, ax = plt.subplots(figsize = (8,5))
bottom_20['coef'].plot.bar(ax=ax, color='g', alpha=0.5)
ax.set_title("20 Most Positive Features")
ax.set_ylabel("Regression Coefficient")
fig.tight_layout()
plt.savefig('images/feat_importance_pos20.png')
```



In [105…

```
# Plotting the 20 most negative features
bottom_20 = coef_df.sort_values(by='coef', ascending=True)[:20]
fig, ax = plt.subplots(figsize = (8,5))
```

```
bottom_20['coef_abs'].plot.bar(ax=ax, color='r', alpha=0.5)
ax.set_title("20 Most Negative Features")
ax.set_ylabel("Regression Coefficient")
fig.tight_layout()
plt.savefig('images/feat_importance_neg20.png')
```



Conclusion, Recommendations and Next Steps

- A Logistic Regression model was the best-performing classifier, with Count Vectorization used to process the annotated tweets
- 97% accuracy, 97% F1 score indicates model captures positive cases (recall) without casting too wide a net, i.e. little misclassification in either direction (precision)
- Words important to the model included 'best', 'awesome', 'successful', 'insane', 'worst', 'worthless'
- Positive tweets had more hashtags on average, negative tweets more frequently contained a price
- Next steps include running the model on real-time Tweets about Bitcoin, pulled via Twitter API, and conducting Time Series Analysis to undestand the predictive power of Tweet sentiment on the price of BTC

Appendix

In [ ]:
```
#X_train_vec_df = pd.DataFrame(X_train_vec.toarray())
#X_train_vec_df.columns = col_names

encoder = LabelEncoder()
contains_price_train = pd.Series(encoder.fit_transform(X_train['contains_price'

final_X_train = pd.concat([X_train_vec_df,
                           contains_price_train,
```

```
                             X_train[['hashtag_count']],
                             X_train[['tweet_length']]
                            ], axis=1)

#final_X_train.head()

# I've had performance issues concatenating a large number of DataFrames to a
```

In [ ]:
```
final_model = LinearSVC()

cv = CountVectorizer(stop_words = stopwords_list)

#final_X_train = cv.fit_transform(X_train['text'])

#####
X_train_vec = cv.fit_transform(X_train['text'])
X_train_vec_df = pd.DataFrame(X_train_vec.toarray(),
                              columns=cv.get_feature_names())

# label encoding boolean column
encoder = LabelEncoder()
contains_price_train = pd.Series(encoder.fit_transform(X_train['contains_price'

final_X_train = pd.concat([X_train_vec_df,
                           contains_price_train,
                           X_train[['hashtag_count']],
                           X_train[['tweet_length']]
                          ], axis=1)

####

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

# X test
#final_X_test = cv.transform(X_test["text"])

####
X_test_vec = cv.transform(X_test["text"])
X_test_vec_df = pd.DataFrame(X_test_vec.toarray(),
                              columns=cv.get_feature_names())

contains_price_test = pd.Series(encoder.transform(X_test['contains_price']))

final_X_test = pd.concat([X_test_vec,
                          contains_price_test,
                          X_test[['hashtag_count']],
                          X_test[['tweet_length']]
                         ], axis=1)


print(final_model.score(final_X_test, y_test))

plot_confusion_matrix(final_model, final_X_test, y_test)
```

In [ ]:
```
display(scores)

# plotting confusion matrix
plot_confusion_matrix(final_model, final_X_test, y_test, cmap=plt.cm.Blues)
```

```
#ax.title('Final Model Confusion Matrix')
#plt.show()
plt.savefig('images/final_confusion_matrix.png')
```

In [ ]:
```
# Precision Recall Curve

#fit logistic regression model to dataset
classifier = LogisticRegression()
classifier.fit(final_X_train, y_train)

#use logistic regression model to make predictions
y_score = classifier.predict_proba(final_X_test)[:, 1]

#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_test, y_score)

#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()
plt.savefig('images/precision_recall_curve.png')
```

In [ ]:
```
y_hat_train_final = final_model.predict(final_X_train)
y_hat_test_final = final_model.predict(final_X_test)
display(scoreTable(final_model, 'final_model', y_train, y_hat_train_final, y_te
```

In [ ]:
```
#pd.DataFrame.sparse.from_spmatrix(final_X_train, columns=col_names)
```

In [ ]:
```
# Confusion Matrix - with seaborn
fig, ax = plt.subplots(figsize = (5,5))
cm = confusion_matrix(y_test, y_hat_test)
sns.heatmap(cm, annot=True, fmt='g', ax=ax)
ax.set_title('Final Model Confusion Matrix')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Let's see if we get better insights from looking at pairs of words. We will only focus on the top 450 most frequent words due to long runtimes.

In [ ]:
```
# top 450 words

# explode
#text_data["hashtag"][text_data['sentiment']==1].explode()

# subset for only top 450 words
# value count bigrams, trigrams, 4-grams

#nltk.bigrams()
```

```
#n_gram_pos = (pd.Series(nltk.ngrams(text_data['text_without_stopwords'][text_c
#n_gram_neg = (pd.Series(nltk.ngrams(tokens_neg, 2)).value_counts())[:10] #Sett
#n_gram_pos
```

In [ ]:
```
# long runtime
model_eval(LogisticRegression(solver='saga', penalty='l1', max_iter=5000), 'log
           CountVectorizer(stop_words = stopwords_list, max_features=5000))
```