SENTIMENT ANALYSIS OF MARKETING

Data analysis for sentiment analysis of marketing typically involves exploring and understanding the characteristics of your dataset to inform subsequent steps in your analysis. Below are steps for performing data analysis on a marketing dataset for sentiment analysis:

1. Load the Dataset:

- Import your marketing dataset into your chosen data analysis environment (e.g., Python with pandas).

```python
import pandas as pd

# Load your dataset
df = pd.read_csv("marketing_data.csv")
```

2. Data Overview:

- Start by getting a high-level understanding of your data by checking the first few rows and the basic statistics.

```python
# Display the first few rows
print(df.head())

# Get summary statistics for numerical columns
print(df.describe())
```

3. Data Cleaning:

- Check for missing values in the dataset and decide how to handle them (e.g., imputation or removal).

```python
# Check for missing values
print(df.isnull().sum())

# Handle missing values (if needed)
# df.dropna(subset=["text"], inplace=True)
```

4. **Data Distribution:**

- Analyze the distribution of sentiment labels in your dataset.

```python
sentiment_counts = df["Sentiment"].value_counts()
print(sentiment_counts)
```

5. **Text Preprocessing:**

- Preprocess the text data by techniques like lowercasing, tokenization, stop word removal, and lemmatization/stemming.

```python
# Example preprocessing (you can use NLTK, spaCy, or other libraries)
df["text"] = df["text"].str.lower()
```

```
# Tokenization, stop word removal, and lemmatization/stemming can also be applied here.
```

6. **Text Length Analysis**:

- Explore the distribution of text lengths, as it can affect sentiment analysis.

```python
df["Text Length"] = df["text"].apply(len)
print(df["Text Length"].describe())
```

7. **Word Frequency Analysis**:

- Analyze the most frequent words in both positive and negative marketing content to identify important keywords.

```python
# Example: Top words in positive and negative text
positive_texts = df[df["Sentiment"] == "Positive"]["text"]
negative_texts = df[df["Sentiment"] == "Negative"]["text"]

def get_top_words(texts, top_n=10):
    words = " ".join(texts).split()
    word_freq = pd.Series(words).value_counts()
    return word_freq.head(top_n)

print("Top words in positive content:")
print(get_top_words(positive_texts))
```

```
print("\nTop words in negative content:")
print(get_top_words(negative_texts))
```

8. **Visualization:**

- Create data visualizations to better understand the data, such as histograms, word clouds, or time series plots if your dataset includes timestamps.

```python
import matplotlib.pyplot as plt
import wordcloud

# Example: Word cloud for positive and negative texts
positive_wordcloud = wordcloud.WordCloud().generate(" ".join(positive_texts))
negative_wordcloud = wordcloud.WordCloud().generate(" ".join(negative_texts))

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(positive_wordcloud, interpolation="bilinear")
plt.title("Word Cloud for Positive Sentiments")

plt.subplot(1, 2, 2)
plt.imshow(negative_wordcloud, interpolation="bilinear")
plt.title("Word Cloud for Negative Sentiments")

plt.show()
```

9. **Correlation Analysis:**

- Examine potential correlations between numerical features and sentiment labels, if relevant.

```python
correlation_matrix = df.corr()
print(correlation_matrix)
```

10. **Additional Analyses**:

- Depending on your specific dataset, you may perform more advanced analyses, such as sentiment trends over time, topic modeling, or sentiment correlation with other marketing metrics.

Data analysis is a crucial step in the sentiment analysis process, helping you understand your data's characteristics, uncover potential biases or patterns, and guide your preprocessing and modeling decisions. The specific analyses you perform can vary depending on your dataset and research goals.

Sentiment Analysis on Twitter tweets

In [1]:
```
import re
import sys
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import string
import nltk

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

from gensim.models import KeyedVectors
from sklearn.manifold import TSNE

%load_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:
```
train = pd.read_csv('../input/twitter-tweets-data/train_tweet.csv')
test = pd.read_csv('../input/twitter-tweets-data/test_tweets.csv')

print(train.shape)
print(test.shape)
```

```
(31962, 3)
(17197, 2)
```

In [3]:
```
train.head()
```

Out[3]:

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation

In [4]:
```
test.head()
```

Out[4]:

	id	tweet
0	31963	#studiolife #aislife #requires #passion #dedic...
1	31964	@user #white #supremacists want everyone to s...
2	31965	safe ways to heal your #acne!! #altwaystohe...
3	31966	is the hp and the cursed child book up for res...
4	31967	3rd #bihday to my amazing, hilarious #nephew...

In [5]:
```
train.isnull().any()
test.isnull().any()
```

Out[5]:
```
id       False
tweet    False
dtype: bool
```

In [6]:
```
# checking out the negative comments from the train set

train[train['label'] == 0].head(10)
```

Out[6]:

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation
5	6	0	[2/2] huge fan fare and big talking before the...
6	7	0	@user camping tomorrow @user @user @user @use...
7	8	0	the next school year is the year for exams.ð ...
8	9	0	we won!!! love the land!!! #allin #cavs #champ...
9	10	0	@user @user welcome here ! i'm it's so #gr...

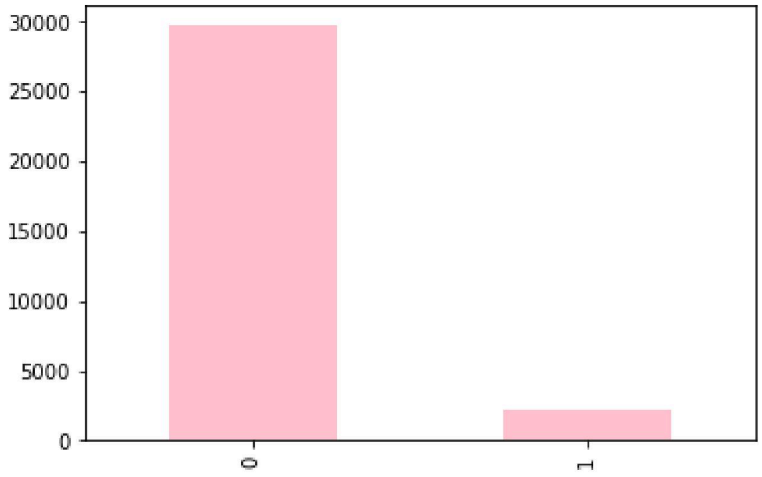
In [7]:
```
# checking out the postive comments from the train set

train[train['label'] == 1].head(10)
```

Out[7]:

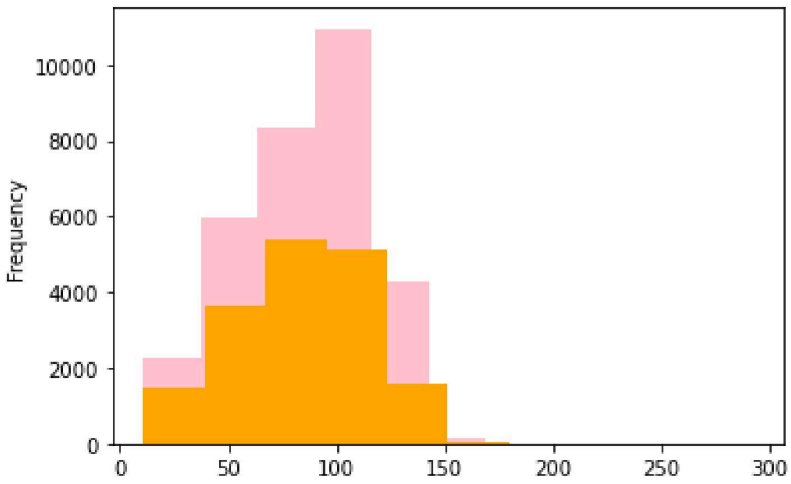
	id	label	tweet
13	14	1	@user #cnn calls #michigan middle school 'buil...
14	15	1	no comment! in #australia #opkillingbay #se...
17	18	1	retweet if you agree!
23	24	1	@user @user lumpy says i am a . prove it lumpy.
34	35	1	it's unbelievable that in the 21st century we'...
56	57	1	@user lets fight against #love #peace
68	69	1	ð ©the white establishment can't have blk fol...
77	78	1	@user hey, white people: you can call people '...
82	83	1	how the #altright uses & insecurity to lu...
111	112	1	@user i'm not interested in a #linguistics tha...

In [8]:
```
train['label'].value_counts().plot.bar(color = 'pink', figsize = (6, 4))
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3233fd5190>
```



In [9]:
```
# checking the distribution of tweets in the data

length_train = train['tweet'].str.len().plot.hist(color = 'pink', figsize =
length_test = test['tweet'].str.len().plot.hist(color = 'orange', figsize =
```



In [10]:
```
# adding a column to represent the length of the tweet

train['len'] = train['tweet'].str.len()
```

```
test['len'] = test['tweet'].str.len()

train.head(10)
```

Out[10]:

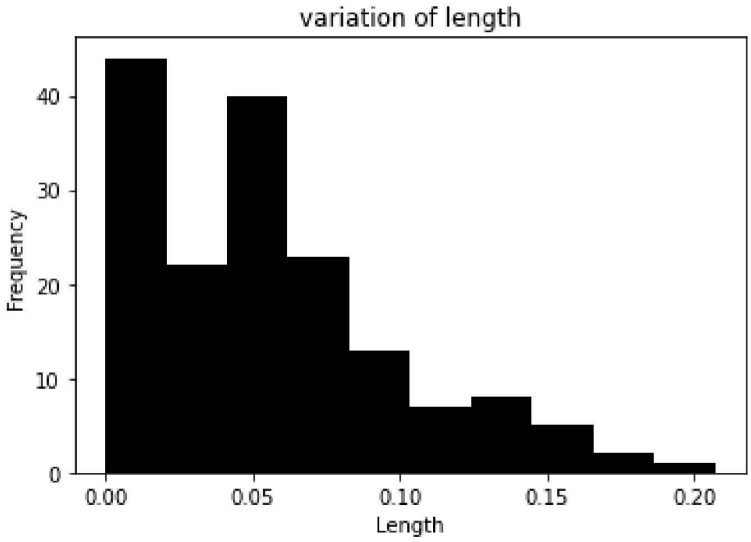
	id	label	tweet	len
0	1	0	@user when a father is dysfunctional and is s...	102
1	2	0	@user @user thanks for #lyft credit i can't us...	122
2	3	0	bihday your majesty	21
3	4	0	#model i love u take with u all the time in ...	86
4	5	0	factsguide: society now #motivation	39
5	6	0	[2/2] huge fan fare and big talking before the...	116
6	7	0	@user camping tomorrow @user @user @user @use...	74
7	8	0	the next school year is the year for exams.ð ...	143
8	9	0	we won!!! love the land!!! #allin #cavs #champ...	87
9	10	0	@user @user welcome here ! i'm it's so #gr...	50

In [11]:
```
train.groupby('label').describe()
```

Out[11]:

	id								
	count	mean	std	min	25%	50%	75%	max	cou
label									
0	29720.0	15974.454441	9223.783469	1.0	7981.75	15971.5	23965.25	31962.0	2972(
1	2242.0	16074.896075	9267.955758	14.0	8075.25	16095.0	24022.00	31961.0	224:

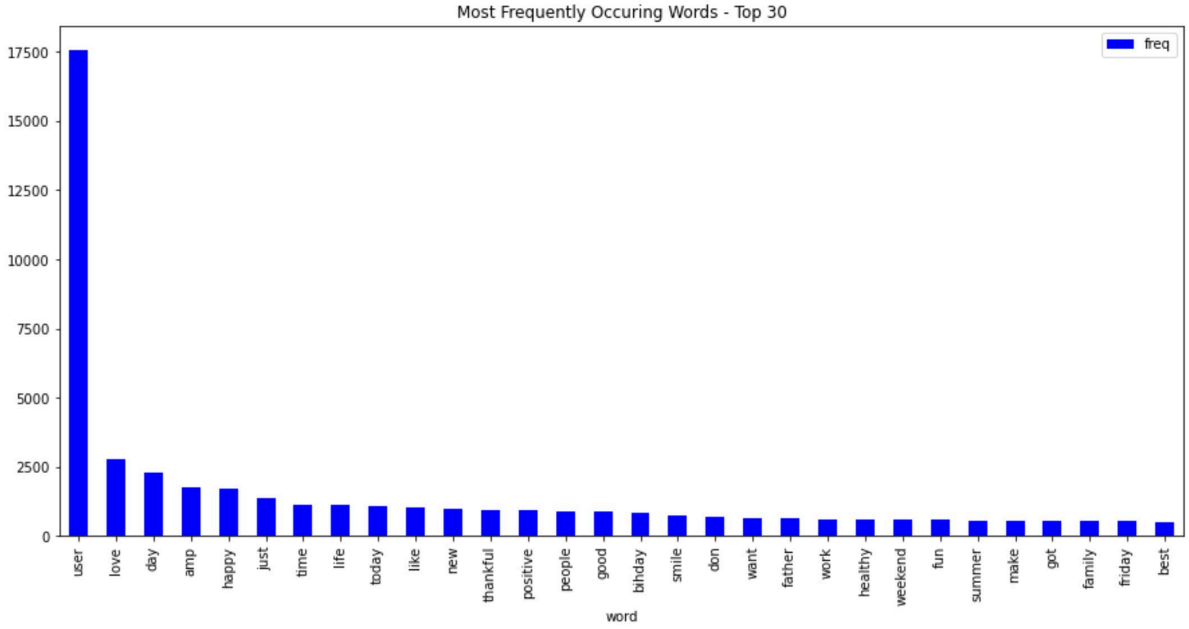
In [12]:
```
train.groupby('len').mean()['label'].plot.hist(color = 'black', figsize = (6
plt.title('variation of length')
plt.xlabel('Length')
plt.show()
```



In [13]:
```
from sklearn.feature_extraction.text import CountVectorizer


cv = CountVectorizer(stop_words = 'english')
words = cv.fit_transform(train.tweet)
```

```
sum_words = words.sum(axis=0)

words_freq = [(word, sum_words[0, i]) for word, i in cv.vocabulary_.items()]
words_freq = sorted(words_freq, key = lambda x: x[1], reverse = True)

frequency = pd.DataFrame(words_freq, columns=['word', 'freq'])

frequency.head(30).plot(x='word', y='freq', kind='bar', figsize=(15, 7), co
plt.title("Most Frequently Occuring Words - Top 30")
```

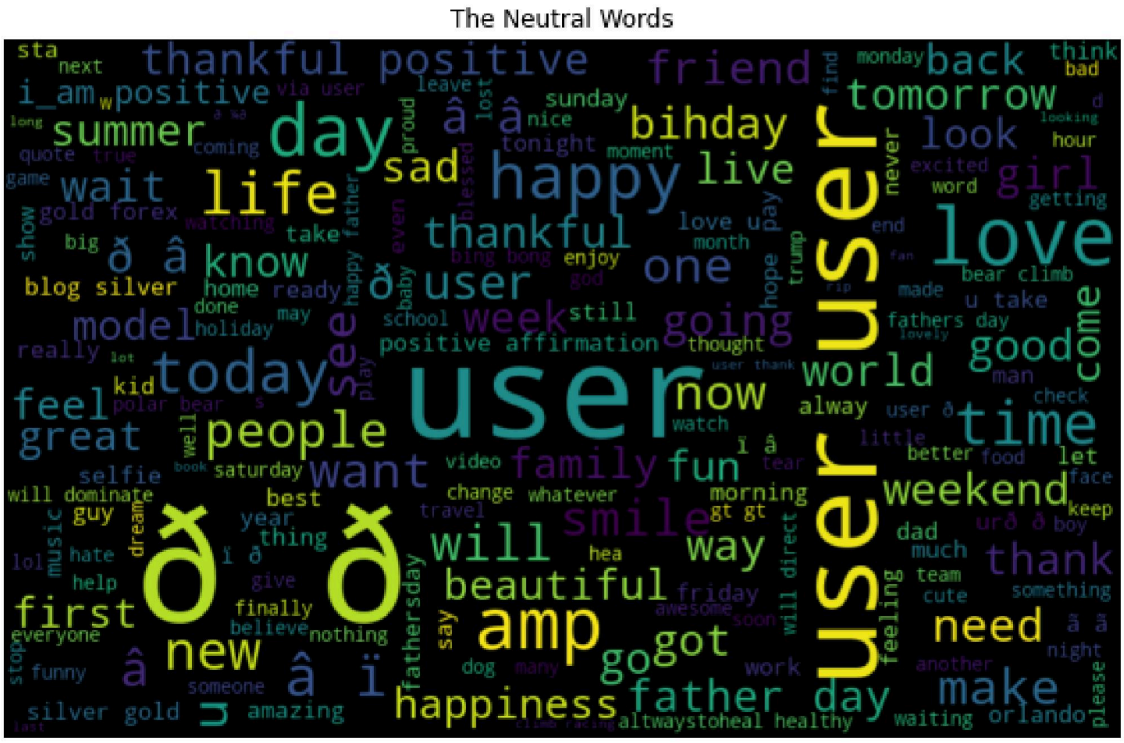
Out[13]: Text(0.5, 1.0, 'Most Frequently Occuring Words - Top 30')



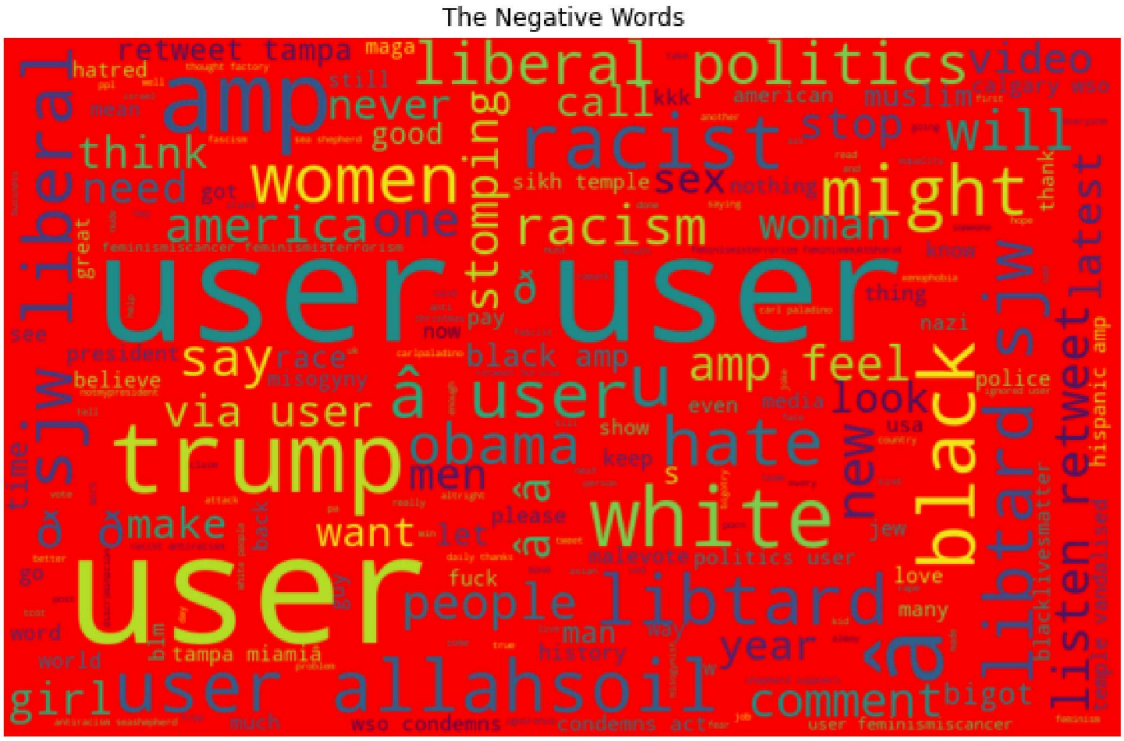
In [14]:
```
from wordcloud import WordCloud

wordcloud = WordCloud(background_color = 'white', width = 1000, height = 100

plt.figure(figsize=(10,8))
plt.imshow(wordcloud)
plt.title("WordCloud - Vocabulary from Reviews", fontsize = 22)
```

Out[14]: Text(0.5, 1.0, 'WordCloud - Vocabulary from Reviews')

In [15]:
```
normal_words =' '.join([text for text in train['tweet'][train['label'] == 0]

wordcloud = WordCloud(width=800, height=500, random_state = 0, max_font_size
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('The Neutral Words')
plt.show()
```



In [16]:
```
negative_words =' '.join([text for text in train['tweet'][train['label'] ==

wordcloud = WordCloud(background_color = 'red', width=800, height=500, rando
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('The Negative Words')
plt.show()
```

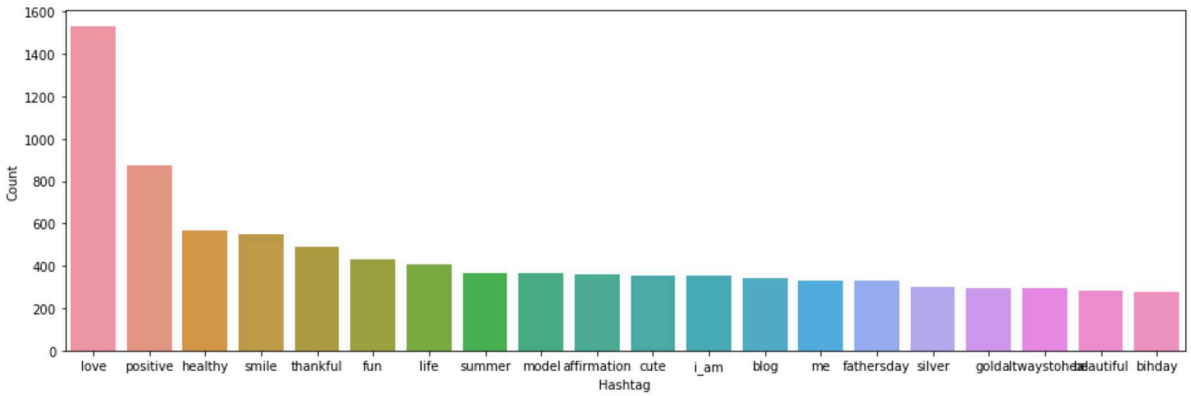


In [17]:
```
# collecting the hashtags

def hashtag_extract(x):
    hashtags = []

    for i in x:
        ht = re.findall(r"#(\w+)", i)
        hashtags.append(ht)

    return hashtags
```

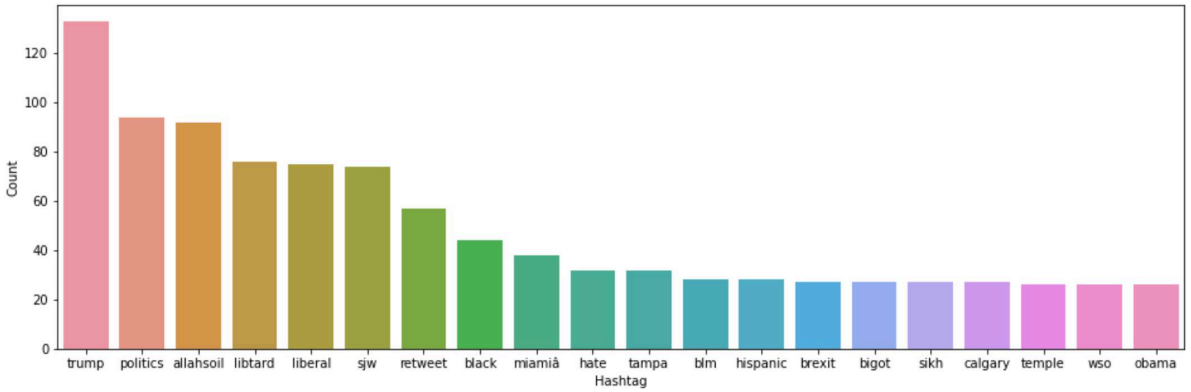
In [18]:
```
# extracting hashtags from non racist/sexist tweets
HT_regular = hashtag_extract(train['tweet'][train['label'] == 0])

# extracting hashtags from racist/sexist tweets
HT_negative = hashtag_extract(train['tweet'][train['label'] == 1])

# unnesting list
HT_regular = sum(HT_regular,[])
HT_negative = sum(HT_negative,[])
```

In [19]:
```
a = nltk.FreqDist(HT_regular)
d = pd.DataFrame({'Hashtag': list(a.keys()),
                  'Count': list(a.values())})

# selecting top 20 most frequent hashtags
d = d.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
```

```
ax.set(ylabel = 'Count')
plt.show()
```



In [20]:
```
a = nltk.FreqDist(HT_negative)
d = pd.DataFrame({'Hashtag': list(a.keys()),
                  'Count': list(a.values())})

# selecting top 20 most frequent hashtags
d = d.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.show()
```



In [21]:
```
# tokenizing the words present in the training set
tokenized_tweet = train['tweet'].apply(lambda x: x.split())

# importing gensim
import gensim

# creating a word to vector model
model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            size=200, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2,
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 2, # no.of cores
            seed = 34)

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

Out[21]:
```
(6110210, 8411580)
```

In [22]:
```
model_w2v.wv.most_similar(positive = "dinner")
```

Out[22]:
```
[('spaghetti', 0.6641602516174316),
 ('#prosecco', 0.600884199142456),
 ('dining', 0.5966363549232483),
 ('toes', 0.5956881046295166),
 ('#teamlh', 0.5933343172073364),
 ('willow', 0.5903596878051758),
 ('kayak,', 0.5897651314735413),
 ('enroute', 0.5896981954574585),
 ('ð\x9f\x91\x8dð\x9f\x8f»ð\x9f\x91\x8dð\x9f\x8f»ð\x9f\x91\x8dð\x9f\x8f»â\x
9d¤ï¸\x8fâ\x9d¤ï¸\x8f',
  0.5896886587142944),
 ('7!', 0.5892139673233032)]
```

In [23]:
```
model_w2v.wv.most_similar(positive = "cancer")
```

Out[23]:
```
[('law.', 0.7288531064987183),
 ('aol', 0.7151804566383362),
 ('champion,', 0.7151604890823364),
 ('ownership', 0.7127461433410645),
 ('roots', 0.7095333933830261),
 ('absurd.', 0.7082613110542297),
 ('ways.', 0.7044087648391724),
 ('#merica', 0.7002105712890625),
 ("society's", 0.699997067451477),
 ('level.', 0.699558436870575)]
```

In [24]:
```
model_w2v.wv.most_similar(positive = "apple")
```

Out[24]:
```
[('"mytraining"', 0.7106212377548218),
 ('mytraining', 0.7001934051513672),
 ('training"', 0.698296070098877),
 ('app,', 0.6650559902191162),
 ('app', 0.6147618293762207),
 ('"my', 0.5902618169784546),
 ('heroku', 0.5865270495414734),
 ('ta', 0.5804429054260254),
 ('bees', 0.578994870185852),
 ('humans.', 0.5708395838737488)]
```

In [25]:
```
model_w2v.wv.most_similar(negative = "hate")
```

Out[25]:
```
[('#apple', -0.019129138439893723),
 ('#games', -0.0472930222749710 1),
 ('us.', -0.04782523587346077),
 ('#hype', -0.051519300788640976),
 ('#fundraising', -0.06459417194128036),
 ('members', -0.06525585055351257),
 ('#grateful', -0.07078631222248077),
 ('now.', -0.07505059242248535),
 ('you?', -0.07548440247774124),
 ('#yay', -0.0784306451678276)]
```

In [26]:
```
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models.doc2vec import LabeledSentence
```

```
/opt/conda/lib/python3.7/site-packages/tqdm/std.py:666: FutureWarning: The
Panel class is removed from pandas. Accessing it from the top-level namespa
ce will also be removed in the next version
  from pandas import Panel
```

In [27]:
```
def add_label(twt):
    output = []
    for i, s in zip(twt.index, twt):
        output.append(LabeledSentence(s, ["tweet_" + str(i)]))
```

```
    return output

# label all the tweets
labeled_tweets = add_label(tokenized_tweet)

labeled_tweets[:6]
```

Out[27]:
```
[LabeledSentence(words=['@user', 'when', 'a', 'father', 'is', 'dysfunctiona
l', 'and', 'is', 'so', 'selfish', 'he', 'drags', 'his', 'kids', 'into', 'hi
s', 'dysfunction.', '#run'], tags=['tweet_0']),
 LabeledSentence(words=['@user', '@user', 'thanks', 'for', '#lyft', 'credi
t', 'i', "can't", 'use', 'cause', 'they', "don't", 'offer', 'wheelchair',
'vans', 'in', 'pdx.', '#disapointed', '#getthanked'], tags=['tweet_1']),
 LabeledSentence(words=['bihday', 'your', 'majesty'], tags=['tweet_2']),
 LabeledSentence(words=['#model', 'i', 'love', 'u', 'take', 'with', 'u', 'a
ll', 'the', 'time', 'in', 'urð\x9f\x93±!!!', 'ð\x9f\x98\x99ð\x9f\x98\x8eð\x
9f\x91\x84ð\x9f\x91', 'ð\x9f\x92¦ð\x9f\x92¦ð\x9f\x92¦'], tags=['tweet_3']),
 LabeledSentence(words=['factsguide:', 'society', 'now', '#motivation'], ta
gs=['tweet_4']),
 LabeledSentence(words=['[2/2]', 'huge', 'fan', 'fare', 'and', 'big', 'talk
ing', 'before', 'they', 'leave.', 'chaos', 'and', 'pay', 'disputes', 'whe
n', 'they', 'get', 'there.', '#allshowandnogo'], tags=['tweet_5'])]
```

In [28]:
```
# removing unwanted patterns from the data

import re
import nltk

nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [29]:
```
train_corpus = []

for i in range(0, 31962):
  review = re.sub('[^a-zA-Z]', ' ', train['tweet'][i])
  review = review.lower()
  review = review.split()

  ps = PorterStemmer()

  # stemming
  review = [ps.stem(word) for word in review if not word in set(stopwords.w

  # joining them back with space
  review = ' '.join(review)
  train_corpus.append(review)
```

In [30]:
```
test_corpus = []

for i in range(0, 17197):
  review = re.sub('[^a-zA-Z]', ' ', test['tweet'][i])
  review = review.lower()
  review = review.split()

  ps = PorterStemmer()

  # stemming
  review = [ps.stem(word) for word in review if not word in set(stopwords.w

  # joining them back with space
```

```
  review = ' '.join(review)
  test_corpus.append(review)
```

In [31]:
```
# creating bag of words

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features = 2500)
x = cv.fit_transform(train_corpus).toarray()
y = train.iloc[:, 1]

print(x.shape)
print(y.shape)
```

```
(31962, 2500)
(31962,)
```

In [32]:
```
# creating bag of words

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features = 2500)
x_test = cv.fit_transform(test_corpus).toarray()

print(x_test.shape)
```

```
(17197, 2500)
```

In [33]:
```
# splitting the training data into train and valid sets

from sklearn.model_selection import train_test_split

x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size = 0.25

print(x_train.shape)
print(x_valid.shape)
print(y_train.shape)
print(y_valid.shape)
```

```
(23971, 2500)
(7991, 2500)
(23971,)
(7991,)
```

In [34]:
```
# standardization

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)
x_valid = sc.transform(x_valid)
x_test = sc.transform(x_test)
```

In [35]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

model = RandomForestClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
```

```
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 0.999123941429227
Validation Accuracy : 0.9524465023151045
F1 score : 0.6146044624746451
[[7308  124]
 [ 256  303]]
```

In [36]:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 0.9851487213716574
Validation Accuracy : 0.9416843949443123
f1 score : 0.5933682373472949
[[7185  247]
 [ 219  340]]
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:764: 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [37]:

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 0.9991656585040257
Validation Accuracy : 0.9310474283569015
f1 score : 0.5350210970464135
[[7123  309]
 [ 242  317]]
```

In [38]:
```
from sklearn.svm import SVC

model = SVC()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 0.978181969880272
Validation Accuracy : 0.9521962207483419
f1 score : 0.4986876640419947
[[7419   13]
 [ 369  190]]
```

In [39]:
```
from xgboost import XGBClassifier

model = XGBClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

```
Training Accuracy : 0.9603687789412206
Validation Accuracy : 0.9555750218996371
f1 score : 0.5748502994011976
[[7396   36]
 [ 319  240]]
```

Thank you