Importing Packages

In [1]:
```
pip install cufflinks
```

```
Requirement already satisfied: prometheus-client in /Users/Raj/opt/anaconda3/lib/python
3.9/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->
cufflinks) (0.14.1)
Requirement already satisfied: jinja2 in /Users/Raj/opt/anaconda3/lib/python3.9/site-pa
ckages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(2.11.3)
Requirement already satisfied: terminado>=0.8.3 in /Users/Raj/opt/anaconda3/lib/python
3.9/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->
cufflinks) (0.13.1)
Requirement already satisfied: Send2Trash>=1.8.0 in /Users/Raj/opt/anaconda3/lib/python
3.9/site-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->
cufflinks) (1.8.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /Users/Raj/opt/anaconda3/li
b/python3.9/site-packages (from packaging->ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflin
ks) (3.0.9)
Requirement already satisfied: bleach in /Users/Raj/opt/anaconda3/lib/python3.9/site-pa
ckages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.
0->cufflinks) (1.5.0)
Requirement already satisfied: pandocfilters>=1.4.1 in /Users/Raj/opt/anaconda3/lib/pyt
hon3.9/site-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.5.0->ip
```

In [2]:
```
# importing packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import warnings
warnings.filterwarnings('ignore')
import pickle
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")

# importing packages for Plotly visualizations
import plotly
from plotly import graph_objs
import cufflinks as cf
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)
plotly.offline.init_notebook_mode()

# import NLP packages
import multiprocessing
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from yellowbrick.text import FreqDistVisualizer, TSNEVisualizer
from wordcloud import WordCloud
from gensim.models.doc2vec import TaggedDocument
from gensim.models import Doc2Vec

# import modeling packages
from sklearn import utils, svm
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoosting
from sklearn.model_selection import train_test_split

%reload_ext autoreload
%autoreload 2
import sys
sys.path.append("py/")
from utils import *
# from config import Keys
from preprocess import *
```

In [3]:
```
pip install preprocess
```

```
Requirement already satisfied: preprocess in /Users/Raj/opt/anaconda3/lib/python3.9/sit
e-packages (2.0.0)
Requirement already satisfied: future in /Users/Raj/opt/anaconda3/lib/python3.9/site-pa
ckages (from preprocess) (0.18.2)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
pip install yellowbrick
```

```
Requirement already satisfied: yellowbrick in /Users/Raj/opt/anaconda3/lib/python3.9/si
te-packages (1.5)
Requirement already satisfied: scikit-learn>=1.0.0 in /Users/Raj/opt/anaconda3/lib/pyth
on3.9/site-packages (from yellowbrick) (1.0.2)
Requirement already satisfied: cycler>=0.10.0 in /Users/Raj/opt/anaconda3/lib/python3.
9/site-packages (from yellowbrick) (0.11.0)
Requirement already satisfied: scipy>=1.0.0 in /Users/Raj/opt/anaconda3/lib/python3.9/s
ite-packages (from yellowbrick) (1.9.1)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in /Users/Raj/opt/anaconda3/li
b/python3.9/site-packages (from yellowbrick) (3.5.2)
Requirement already satisfied: numpy>=1.16.0 in /Users/Raj/opt/anaconda3/lib/python3.9/
site-packages (from yellowbrick) (1.20.0)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/Raj/opt/anaconda3/lib/python
3.9/site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.0.9)
Requirement already satisfied: packaging>=20.0 in /Users/Raj/opt/anaconda3/lib/python3.
9/site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (21.3)
Requirement already satisfied: pillow>=6.2.0 in /Users/Raj/opt/anaconda3/lib/python3.9/
site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.4.0)
Requirement already satisfied: python-dateutil>=2.7 in /Users/Raj/opt/anaconda3/lib/pyt
hon3.9/site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/Raj/opt/anaconda3/lib/python
3.9/site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.4.2)
Requirement already satisfied: fonttools>=4.22.0 in /Users/Raj/opt/anaconda3/lib/python
3.9/site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/Raj/opt/anaconda3/lib/pyt
hon3.9/site-packages (from scikit-learn>=1.0.0->yellowbrick) (2.2.0)
Requirement already satisfied: joblib>=0.11 in /Users/Raj/opt/anaconda3/lib/python3.9/s
ite-packages (from scikit-learn>=1.0.0->yellowbrick) (1.1.0)
Requirement already satisfied: six>=1.5 in /Users/Raj/opt/anaconda3/lib/python3.9/site-
packages (from python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Importing Data

In [5]:
```
# import dataframe into notebook
df = pd.read_csv("/Users/Raj/NLP-Project/labeled_data.csv", index_col=0)
df.head()
```

Out[5]:

id	count	hate_speech	offensive_language	neither	class	tweet
1	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
2	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
3	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
4	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
5	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

In [6]:
```
# get dimensions of dataframe
df.shape
```

Out[6]:
```
(24783, 6)
```

In [7]:
```
# rename columns for easier reference
df = df.rename(columns={"hate_speech": 'hate', "offensive_language": 'offensive', "neithe
df.head()
```

Out[7]:

id	count	hate	offensive	neutral	target	tweet
1	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
2	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
3	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
4	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
5	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

Target Variable

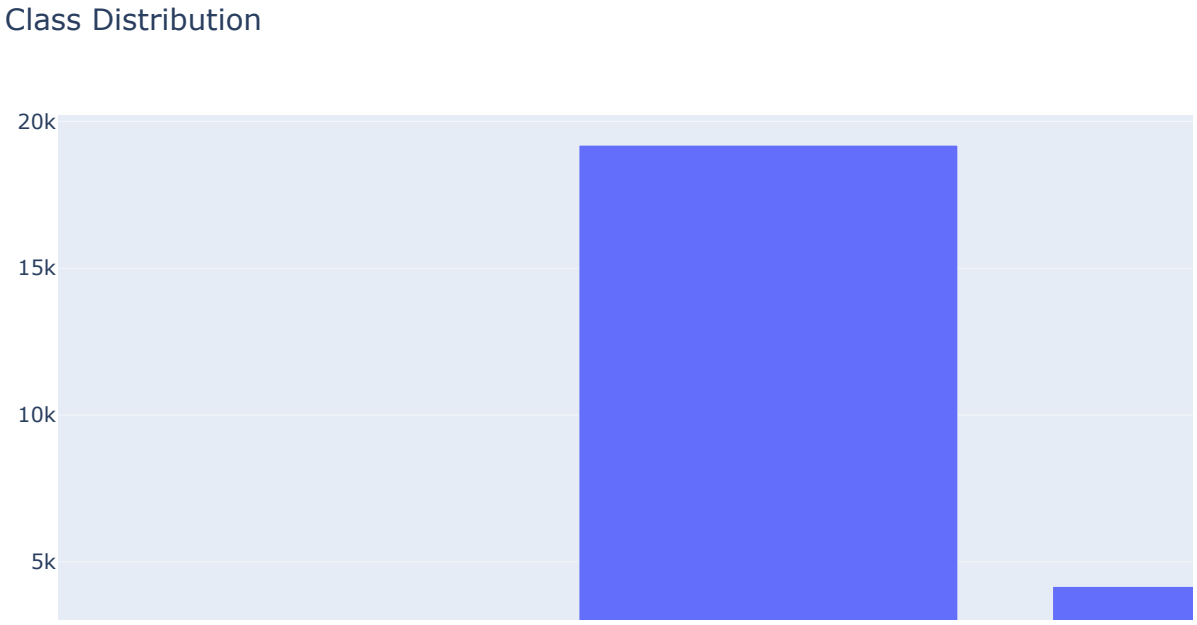
Once we have identified our target variable, we want to visualize the distribution. The figure below indicates that overwhelmingly tweets categorized as offensive totaling over 19,000, while hate tweets comprise a mere 1430.

The major challenge of automated hate speech detection is the separation of hate speech from offensive language. The methodology behind this study was to collect tweets that contained terms from the Hatebase.org lexicon.

Hate speech, as defined by ALA, is any form of expression intending to vilify, humiliate, or incite hatred against a group or an individual on the basis of race, religion, skin color, sexual or gender identity, ethnicity, disability, or national origin.

While it is protected by the First Amendment, if it incites criminal activity or threats of violence against a person or group, then it can be criminalized.

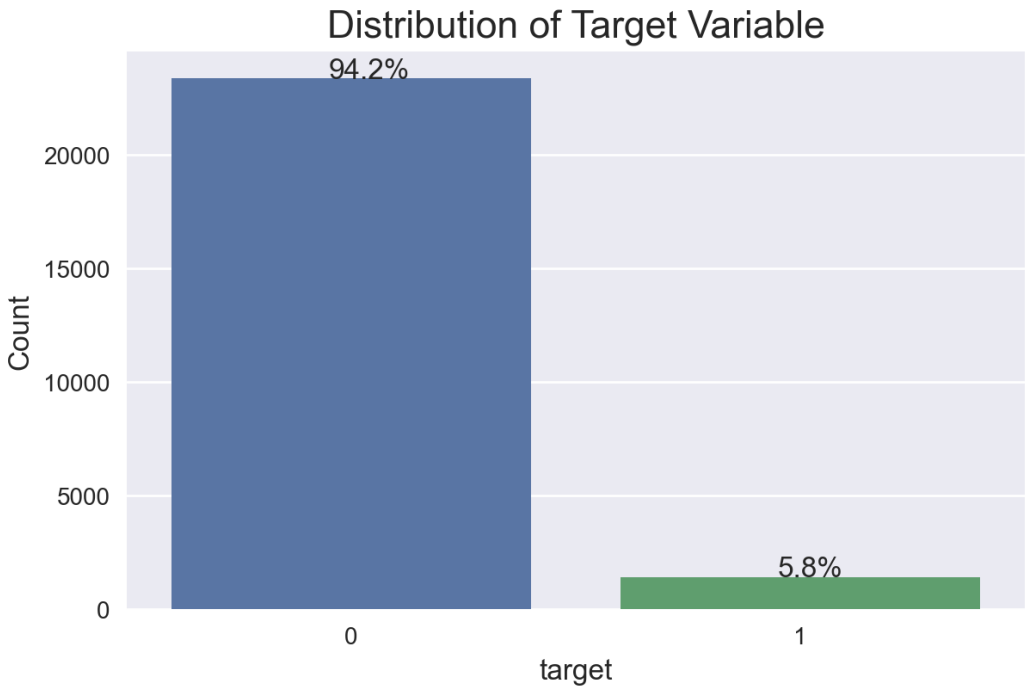
In [8]:
```
import seaborn as sns
# display class distribution
hate = len(df[df['target'] == 0])
off = len(df[df['target'] == 1])
neu = len(df[df['target'] == 2])
dist = [
    graph_objs.Bar(
        x=["hate", "offensive", "neutral"],
        y=[hate, off, neu],
)]
plotly.offline.iplot({"data":dist, "layout":graph_objs.Layout(title="Class Distribution")}
```



In [9]:
```
# create hate and non-hate categories by combining offensive and neutral categories
df.target = df.target.replace([2], 1)
df.target = df.target.replace([0, 1], [1, 0])
df.target.value_counts()
```

Out[9]:
```
0    23353
1     1430
Name: target, dtype: int64
```

In [10]:
```
# create visualization for new target variable distribution
def barplot(df, feature, title):
    fig, ax = plt.subplots(figsize=(6, 4))
    sns.countplot(x=feature, data=df, ax=ax)
    plt.title(title, fontsize=16)
    plt.xlabel("target", fontsize=12)
    plt.ylabel("Count", fontsize=12)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    total = len(df.target)
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total)
        x = p.get_x() + p.get_width() / 2 - 0.05
        y = p.get_y() + p.get_height()
        ax.annotate(percentage, (x, y), size=12)
    fig.show()
    fig.savefig("/Users/Raj/NLP-Project/Project_milestone-5/images/target_distribution.pn

plt.style.use('seaborn')
barplot(df, 'target', 'Distribution of Target Variable')
```



In [11]:
```
# display first few lines of tweet texts
df.tweet.head(20)
```

Out[11]:
```
id
1     !!! RT @mayasolovely: As a woman you shouldn't...
2     !!!!! RT @mleew17: boy dats cold...tyga dwn ba...
3     !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
4     !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
5     !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
6     !!!!!!!!!!!!!!!!!!"@T_Madison_x: The shit just...
7     !!!!!!"@__BrighterDays: I can not just sit up ...
8     !!!!&#8220;@selfiequeenbri: cause I'm tired of...
9     " &amp; you might not get ya bitch back &amp; ...
10    " @rhythmixx_ :hobbies include: fighting Maria...
11    " Keeks is a bitch she curves everyone " lol I...
12                    " Murda Gang bitch its Gang Land "
13    " So hoes that smoke are losers ? " yea ... go...
14       " bad bitches is the only thing that i like "
15                               " bitch get up off me "
16                      " bitch nigga miss me with it "
17                               " bitch plz whatever "
18                             " bitch who do you love "
19                    " bitches get cut off everyday B "
20                    " black bottle &amp; a bad bitch "
Name: tweet, dtype: object
```

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

Out[12]:
```
0    23353
1     1430
Name: target, dtype: int64
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24783 entries, 1 to 24783
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   count      24783 non-null  int64 
 1   hate       24783 non-null  int64 
 2   offensive  24783 non-null  int64 
 3   neutral    24783 non-null  int64 
 4   target     24783 non-null  int64 
 5   tweet      24783 non-null  object
dtypes: int64(5), object(1)
memory usage: 1.3+ MB
```

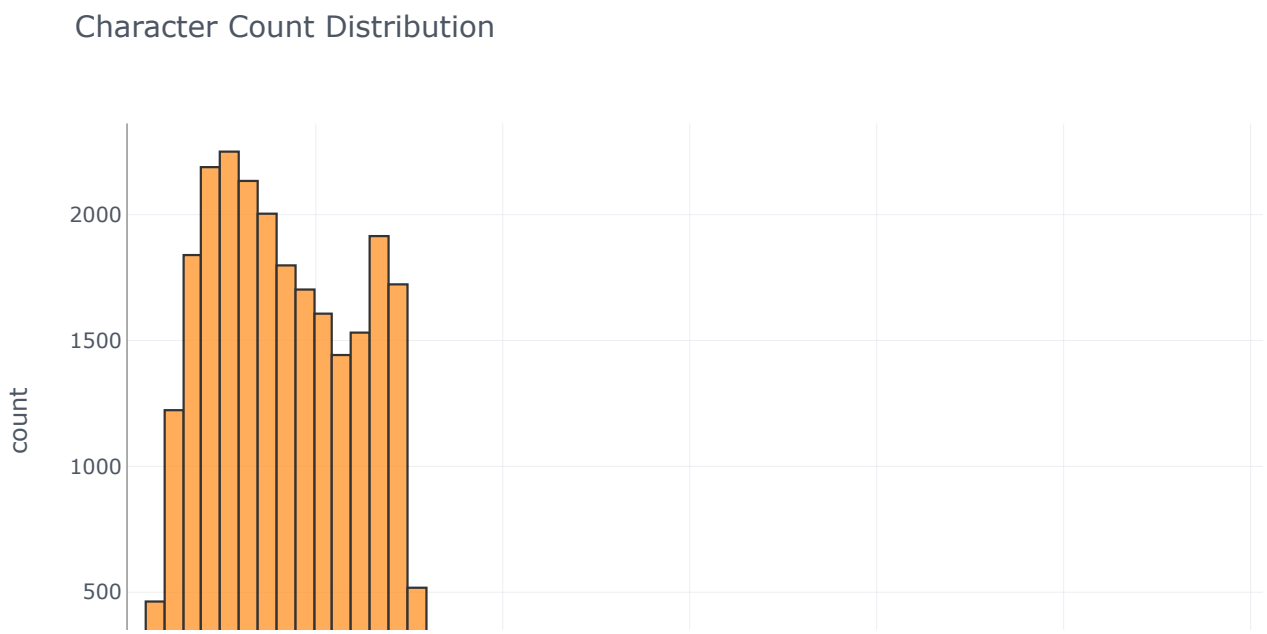
Initial EDA

Number of Characters Per Tweet

In [22]:
```
# create function to ccount number of characters in a tweet
def num_of_chars(df, col):
    df['char_ct'] = df[col].str.len()
    print(df[[col, 'char_ct']].head())

num_of_chars(df, 'tweet')
```

```
                                               tweet  char_ct
id
1   !!! RT @mayasolovely: As a woman you shouldn't...      140
2   !!!!! RT @mleew17: boy dats cold...tyga dwn ba...       85
3   !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...      120
4   !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...       62
5   !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...      137
```

In [23]:
```
# create visualization to display character count distribution
df['char_ct'].iplot(
    kind='hist',
    bins=100,
    xTitle='character count',
    linecolor='black',
    yTitle='count',
    title='Character Count Distribution')
```

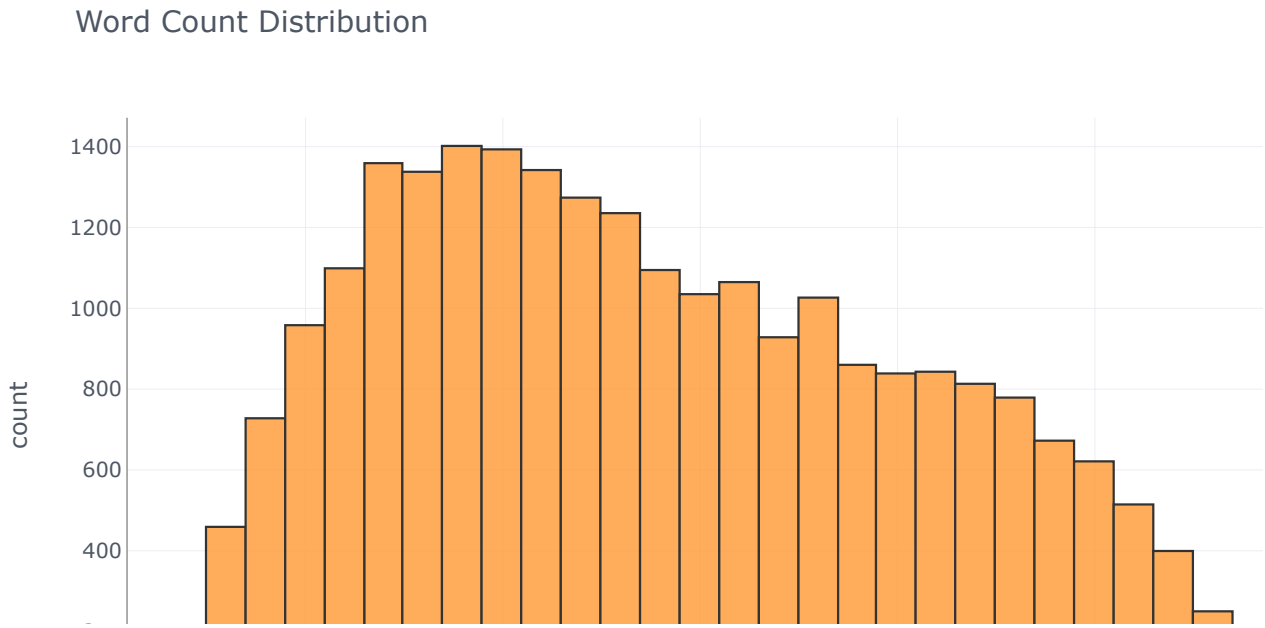


Word Count Per Tweet

In [17]:
```
# create functions to count number of words in tweet
def num_of_words(df, col):
    df['word_ct'] = df[col].apply(lambda x: len(str(x).split(" ")))
    print(df[[col, 'word_ct']].head())

num_of_words(df, 'tweet')
```

```
                                                tweet  word_ct
id
1   !!! RT @mayasolovely: As a woman you shouldn't...       25
2   !!!!! RT @mleew17: boy dats cold...tyga dwn ba...       16
3   !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...       21
4   !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...        9
5   !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...       26
```

In [24]:
```
# create visualization for word count distribution
df['word_ct'].iplot(
    kind='hist',
    bins=40,
    xTitle='word count',
    linecolor='black',
    yTitle='count',
    title='Word Count Distribution')
```



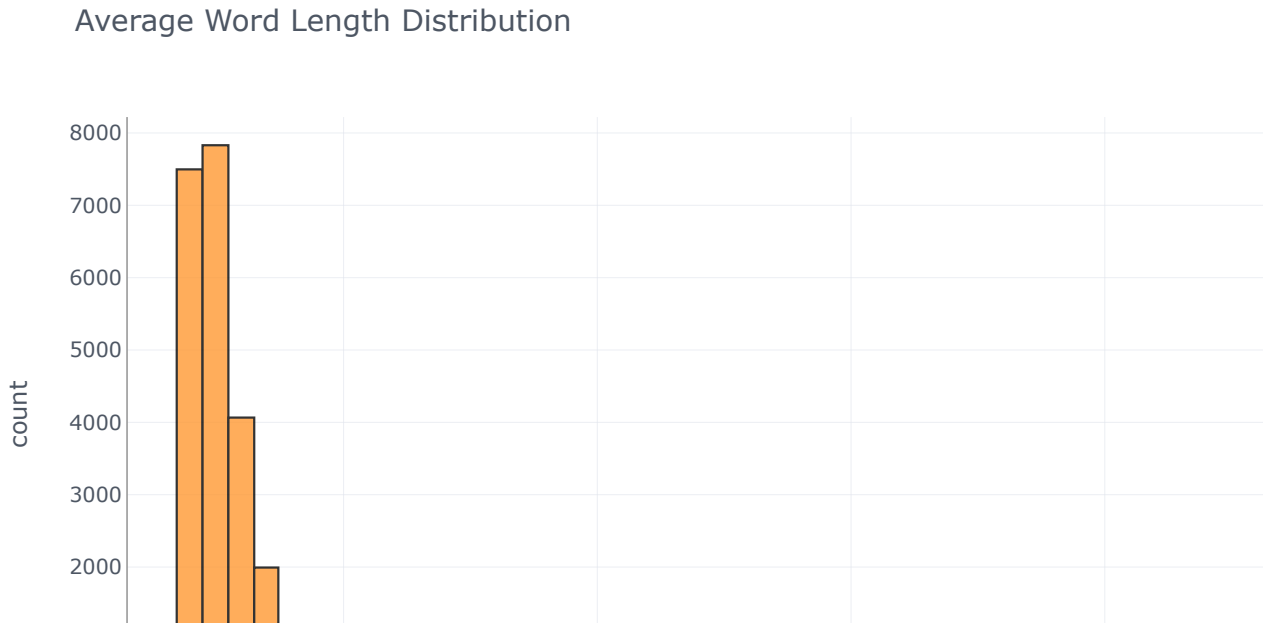
Average Word Length Per Tweet

In [14]:
```
from nltk.probability import FreqDist
```

In [28]:
```
# create function to calculate average word length and then average word length per tweet
def avg_word(sentence):
    words = sentence.split()
    return (sum(len(word) for word in words)/len(words))

def avg_word_length(df, col):
    df['avg_wrd'] = df[col].apply(lambda x: avg_word(x))
    print(df[[col, 'avg_wrd']].head())

avg_word_length(df, 'tweet')
```

```
                                                tweet   avg_wrd
id
1   !!! RT @mayasolovely: As a woman you shouldn't...  4.640000
2   !!!!! RT @mleew17: boy dats cold...tyga dwn ba...  4.375000
3   !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...  4.761905
4   !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...  6.000000
5   !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...  4.307692
```

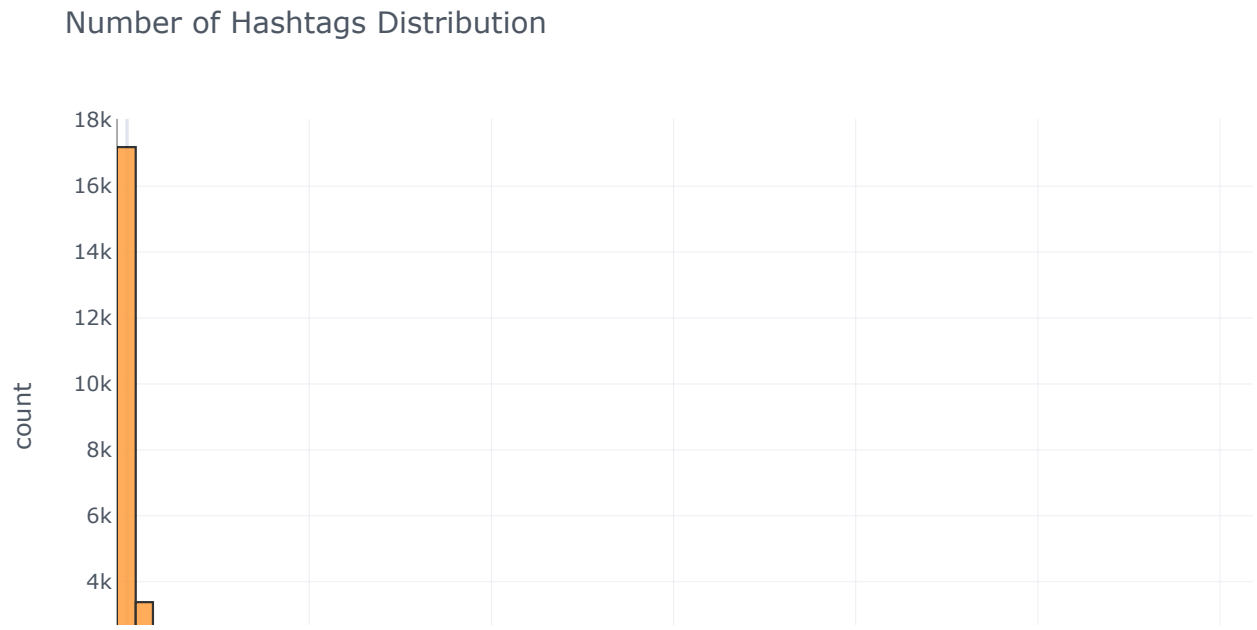
In [29]:
```
# create visualization for average word length distribution
df['avg_wrd'].iplot(
    kind='hist',
    bins=60,
    xTitle='average word length',
    linecolor='black',
    yTitle='count',
    title='Average Word Length Distribution')
```



In [30]:
```
# create function to count number of hashtags per tweet
def hash_ct(df, col):
    df['hash_ct'] = df[col].apply(lambda x: len(re.split(r'#', str(x)))-1)
    print(df[[col, 'hash_ct']].head())

hash_ct(df, 'tweet')
```

```
                                               tweet  hash_ct
id
1   !!! RT @mayasolovely: As a woman you shouldn't...        0
2   !!!!! RT @mleew17: boy dats cold...tyga dwn ba...        0
3   !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...        0
4   !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...        0
5   !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...        1
```

In [31]:
```
# create visualization for displaying hashtag distribution
df['hash_ct'].iplot(
    kind='hist',
    bins=100,
    xTitle='hashtags count',
    linecolor='black',
    yTitle='count',
    title='Number of Hashtags Distribution')
```



In [40]:
```
def plot_frequency(data):
    """
    Ploting words frequency distribution
    from corpus. data should be list of lists with strings
    """
    words_lst = []
    for tweet in data:
        for word in tweet:
            words_lst.append(word)

    fdist = FreqDist(words_lst)
    plt.figure(figsize=(10,6))
    fdist.plot(30)
    plt.show()
```

In [41]:
```
def tokenize_text(text):

    """
    Tocanize document and create visualization of most recent words
    Wiil filter data with stopwords
    """
    tokens = nltk.word_tokenize(text)

    stopwords_removed = [token for token in tokens if token not in stop_words]

    return stopwords_removed
```

In [42]:
```
processed_data = df['tweet'].tolist()
```

In [88]:
```
tokens = []
for sublist in processed_data:
    for item in sublist:
        tokens.append(item)
```

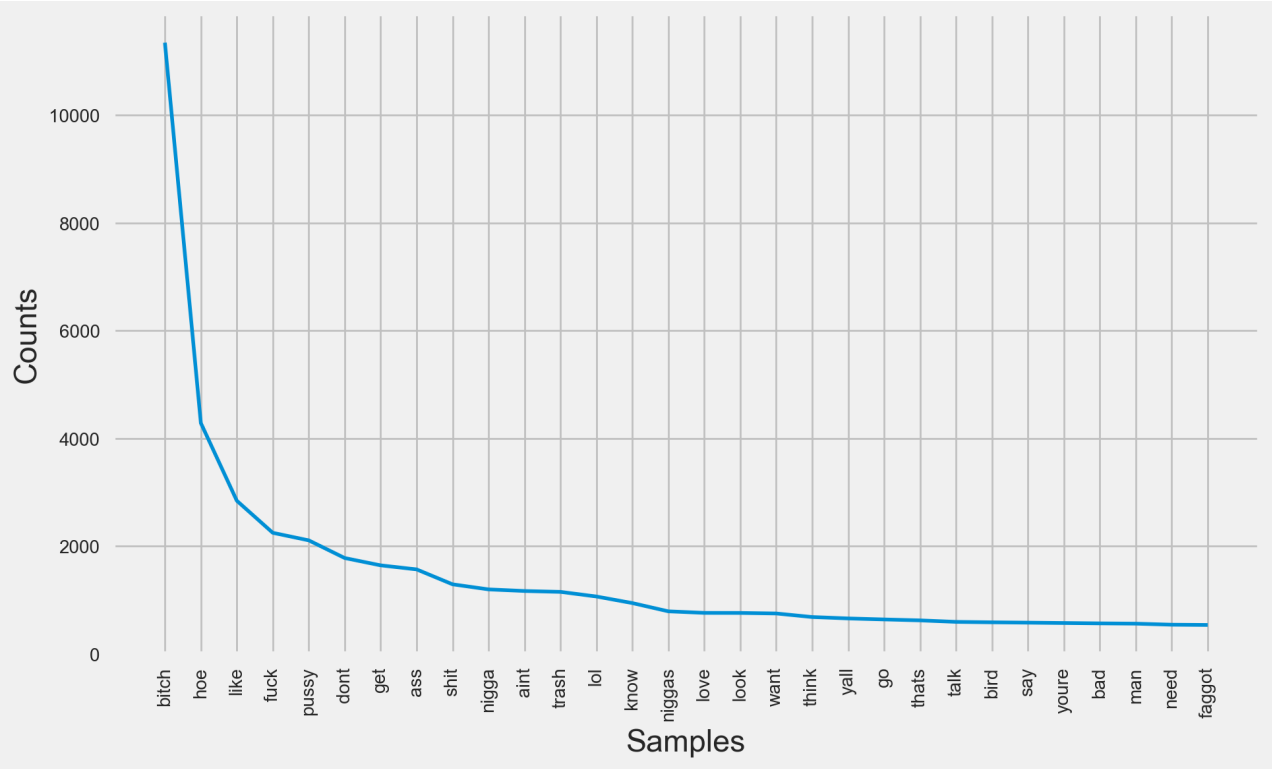
In [ ]:
```
tokens
```

In [46]:
```
processed_data
```

Out[46]:
```
[['woman', 'shouldnt', 'complain', 'clean', 'house', 'man', 'trash'],
 ['boy', 'dats', 'coldtyga', 'dwn', 'bad', 'cuffin', 'dat', 'hoe', 'place'],
 ['dawg', 'fuck', 'bitch', 'start', 'confuse', 'shit'],
 ['look', 'like', 'tranny'],
 ['shit', 'hear', 'true', 'faker', 'bitch', 'tell'],
 ['shit', 'blow', 'meclaim', 'faithful', 'somebody', 'fuck', 'hoe'],
 ['sit', 'hate', 'bitch', 'get', 'shit', 'go'],
 ['cause', 'tire', 'big', 'bitch', 'come', 'skinny', 'girls'],
 ['bitch', 'thats'],
 ['hobbies', 'include', 'fight', 'mariam', 'bitch'],
 ['keeks', 'bitch', 'curve', 'lol', 'walk', 'conversation', 'like', 'smh'],
 ['murda', 'gang', 'bitch', 'gang', 'land'],
 ['hoe', 'smoke', 'losers', 'yea'],
 ['bad', 'bitch', 'thing', 'like'],
 ['bitch'],
 ['bitch', 'nigga', 'miss'],
 ['bitch', 'plz'],
 ['bitch', 'love'],
 ['bitch', 'cut', 'everyday'],
```

```
#frequency distribution of tokens in corpus
```

In [47]:
```
plot_frequency(processed_data)
```



In [49]:
```
def freq_wrds_class(data, n = 20, show= True):

    """
    Returns list of 2 tuples that represents frequency
    of words in document

    data - Series with string data
    n - number of most common words to show
    """

    protc_data = list(map(tokenize_text, data))

    total_vocab = set()
    for comment in protc_data:
        total_vocab.update(comment)

    if show:
        print('Total words in vocab : {}'.format(len(total_vocab)))
        print (30*'-')
        print('Top {} most frequent words:'.format(n))
        flat_data = [item for sublist in protc_data for item in sublist]
        freq = FreqDist(flat_data)
        return freq.most_common(n)
    flat_data = [item for sublist in protc_data for item in sublist]
    freq = FreqDist(flat_data)

    return freq
```

```
#Most Frequent Words for Each Class
```

In [50]:
```
df_freq_hate = df[df['target']==1]
df_freq_not_hate = df[df['target']==0]
```

In [51]:
```
data_hate = df_freq_hate['tweet']
data_not_hate = df_freq_not_hate['tweet']
```

In [52]:
```
data_hate
```

Out[52]:
```
id
86                                          [queer, gaywad]
90       [alsarabsss, hes, beaner, smh, tell, hes, mexi...
111      [youre, fuck, gay, blacklist, hoe, hold, tehgo...
185      [lmfaoooo, hate, black, people, theres, black,...
203                                         [nigger, lmfao]
                               ...
24577                         [guy, biggest, faggot, omfg]
24686    [name, offensive, kike, wop, kraut, wetback, j...
24752                    [pussy, ass, nigga, know, nigga]
24777                                       [youre, niggers]
24778    [youre, retard, hope, type, diabetes, die, sug...
Name: tweet, Length: 1430, dtype: object
```

In [129]:
```
def normalized_word_fqncy(data, n=25):

    frqncy = freq_wrds_class(data, n, show = False)
    total_w_count = sum(frqncy.values())
    top = hate_freq.most_common(25)
    print("Word \t\t Normalized Frequency")
    print()
    for word in top:
        normalized_frequency = word[1]/hate_total_word_count
        print("{} \t\t {:.4}".format(word[0], normalized_frequency))
```

Hashtags Count

In [ ]:
```
```

Data Preprocessing:

In [32]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
plt.style.use('fivethirtyeight')
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import warnings
warnings.filterwarnings('ignore')

from collections import Counter
import re as regex
import pickle

import plotly
import cufflinks as cf
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)
import plotly.offline
plotly.offline.init_notebook_mode()
from plotly.offline import iplot
from plotly import graph_objs

from sklearn.model_selection import train_test_split

import nltk
from nltk.corpus import stopwords
import string
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from yellowbrick.text import FreqDistVisualizer
from yellowbrick.text.tsne import tsne
from wordcloud import WordCloud
from nltk.stem.porter import PorterStemmer
from textblob import TextBlob, Word


%reload_ext autoreload
%autoreload 2

import sys
sys.path.append("../py")
from utils import *
from preprocess import *
```

In [34]:
```
import numpy as np
import seaborn as sns
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score, auc,
from tqdm import tqdm
import re
import nltk
from nltk.stem.porter import PorterStemmer
from textblob import Word
import datetime
import pandas as pd
import requests
import sys
sys.path.append("../py")
# from typedconfig import Config, key, section
import gensim
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import matplotlib.pyplot as plt

def accuracy(y, y_hat):
    y_y_hat = list(zip(y, y_hat))
    tp = sum([1 for i in y_y_hat if i[0] == 1 and i[1] == 1])
    tn = sum([1 for i in y_y_hat if i[0] == 0 and i[1] == 0])
    return (tp + tn) / float(len(y_y_hat))

def f1(y, y_hat):
    precision_score = precision(y, y_hat)
    recall_score = recall(y, y_hat)
    numerator = precision_score * recall_score
    denominator = precision_score + recall_score
    return 2 * (numerator / denominator)

def precision(y, y_hat):
    y_y_hat = list(zip(y, y_hat))
    tp = sum([1 for i in y_y_hat if i[0] == 1 and i[1] == 1])
    fp = sum([1 for i in y_y_hat if i[0] == 0 and i[1] == 1])
    return tp / float(tp + fp)

def recall(y, y_hat):
    # Your code here
    y_y_hat = list(zip(y, y_hat))
    tp = sum([1 for i in y_y_hat if i[0] == 1 and i[1] == 1])
    fn = sum([1 for i in y_y_hat if i[0] == 1 and i[1] == 0])
    return tp / float(tp + fn)

def group_list(lst, size=100):
    """
    Generate batches of 100 ids in each
    Returns list of strings with , seperated ids
    """
    new_list =[]
    idx = 0
    while idx < len(lst):
        new_list.append(
            ','.join([str(item) for item in lst[idx:idx+size]])
        )
        idx += size
    return new_list


def tweets_request(tweets_ids):
    """
    Make a request to Tweeter API
    """
    df_lst = []
```

```
    for batch in tqdm(tweets_ids):
        url = "https://api.twitter.com/2/tweets?ids={}&&tweet.fields=created_at,entities,
        payload={}
        headers = {'Authorization': 'Bearer ' + keys['bearer_token'],
        'Cookie': 'personalization_id="v1_hzpv7qXpjB6CteyAHDWYQQ==; guest_id=v1%3A161498
        r = requests.request("GET", url, headers=headers, data=payload)
        data = r.json()
        if 'data' in data.keys():
            df_lst.append(pd.DataFrame(data['data']))

    return pd.concat(df_lst)

def aucy(X, y, model):
    """
        Function to calculate ROC-AUC Score based on predict_proba(X)
        where X is feature values, y is target values, and model is instantiated model va
    """
    probs = model.predict_proba(X)[:,1]
    return roc_auc_score(y, probs)

def auc2(X, y, model):
    """
        Function to calculate ROC-AUC Score based on decision_function(X)
        where X is feature values, y is target values, and model is instantiated model va
    """
    probs = model.decision_function(X)
    return roc_auc_score(y, probs)

def aps(X, y, model):
    """
        Function to calculate PR-AUC Score based on predict_proba(X)
        where X is feature values, y is target values, and model is instantiated model va
    """
    probs = model.predict_proba(X)[:,1]
    return average_precision_score(y, probs)

def aps2(X, y, model):
    """
        Function to calculate PR-AUC Score based on decision_function(X)
        where X is feature values, y is target values, and model is instantiated model va
    """
    probs = model.decision_function(X)
    return average_precision_score(y, probs)

def get_metrics_confusion(X, y, y_pred, model):
    """
        Function to get accuracy, F1, ROC-AUC, recall, precision, PR-AUC scores followed
        where X is feature dataset, y is target dataset, and model is instantiated model
    """
    acc = accuracy_score(y, y_pred)
    f1 = f1_score(y, y_pred)
    roc_auc = aucy(X, y, model)
    rec = recall_score(y, y_pred)
    prec = precision_score(y, y_pred)
    logloss = log_loss(y, y_pred)

    pr_auc = aps(X, y, model)

    print('Accuracy: ', acc)
    print('F1 Score: ', f1)
    print('ROC-AUC: ', roc_auc)
    print('Recall: ', rec)
    print('Precision: ', prec)
    print('PR-AUC: ', pr_auc)
    print('Log loss: ', logloss )

    cnf = confusion_matrix(y, y_pred)
```

```
    group_names = ['TN','FP','FN','TP']
    group_counts = ['{0:0.0f}'.format(value) for value in cnf.flatten()]
    group_percentages = ['{0:.2%}'.format(value) for value in cnf.flatten()/np.sum(cnf)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(group_names, group_counts, group_
    labels = np.asarray(labels).reshape(2,2)
    fig, ax = plt.subplots(figsize=(4,4))
    sns.heatmap(cnf, annot=labels, fmt='', cmap='Blues', annot_kws={'size':14}, cbar=Fals
```

```
def get_confusion(y, y_pred):
    cnf = confusion_matrix(y, y_pred)
    group_names = ['TN','FP','FN','TP']
    group_counts = ['{0:0.0f}'.format(value) for value in cnf.flatten()]
    group_percentages = ['{0:.2%}'.format(value) for value in cnf.flatten()/np.sum(cnf)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(group_names, group_counts, group_
    labels = np.asarray(labels).reshape(2,2)
    fig, ax = plt.subplots(figsize=(4,4))
    sns.heatmap(cnf, annot=labels, fmt='', cmap='Blues', annot_kws={'size':14}, cbar=Fals
```

```
def get_metrics(X, y, y_pred, model):
    """
        Function to get training and validation F1, recall, precision, PR AUC scores
    """
    acc = accuracy_score(y, y_pred)
    f1 = f1_score(y, y_pred)
    rec = recall_score(y, y_pred)
    prec = precision_score(y, y_pred)
    roc_auc = auc(X, y, model)
    pr_auc = aps(X, y, model)

    print('Accuracy: ', acc)
    print('F1: ', f1)
    print('Recall: ', rec)
    print('Precision: ', prec)
    print('ROC-AUC: ', roc_auc)
    print('PR-AUC: ', pr_auc)


def get_metrics_2(X, y, y_pred, model):
    """
        Function to get training and validation F1, recall, precision, PR AUC scores
        Instantiate model and pass the model into function
        Pass X_train, y_train, X_val, Y_val datasets
        Pass in calculated model.predict(X) for y_pred
    """
    ac = accuracy_score(y, y_pred)
    f1 = f1_score(y, y_pred)
    rc = recall_score(y, y_pred)
    pr = precision_score(y, y_pred)
    rocauc = auc2(X, y, model)
    prauc = aps2(X, y, model)

    print('Accuracy: ', ac)
    print('F1: ', f1)
    print('Recall: ', rc)
    print('Precision: ', pr)
    print('ROC-AUC: ', rocauc)
    print('PR-AUC: ', prauc)

def get_metrics_3(X, y, y_pred, model):
    acc = accuracy_score(y, y_pred)
    print('Accuracy: ', acc)
    f1 = f1_score(y, y_pred, average="binary", pos_label="1")
    print('F1: ', f1)
    rec = recall_score(y, y_pred, average="binary", pos_label="1")
    print('Recall: ', rec)
    prec = precision_score(y, y_pred, average="binary", pos_label="1")
    print('Precision: ', prec)
```

```
    y_probs = model.predict_proba(X)[:,1]
    roc_auc = roc_auc_score(y, y_probs)
    print('ROC-AUC: ', roc_auc)
    pr_auc = average_precision_score(y, y_probs, pos_label="1")
    print('PR-AUC: ', pr_auc)

def num_of_words(df, col):
    df['word_ct'] = df[col].apply(lambda x: len(str(x).split(" ")))
    print(df[[col, 'word_ct']])

def num_of_chars(df, col):
    df['char_ct'] = df[col].str.len()
    print(df[[col, 'char_ct']])

def avg_word(sentence):
    words = sentence.split()
    return (sum(len(word) for word in words)/len(words))

def avg_word_length(df, col):
    df['avg_wrd'] = df[col].apply(lambda x: avg_word(x))
    print(df[[col, 'avg_wrd']].head())


stop_words = set(stopwords.words('english'))
def no_stopwords(text):
    lst = [word for word in text if word not in stop_words]
    return lst

def term_frequency(df):
    tf1 = (df['tweet'].apply(lambda x: pd.value_counts(x.split(" "))).sum(axis=0).reset_i
    tf1.columns = ['words', 'tf']
    tf1 = tf1.sort_values(by='tf', ascending=False).reset_index()
    return tf1

def stemming(token_list):
    """
        Function for stemming via PorterStemmer()
        Pass in list of tokens and returns a list of stemmed tokens
    """
    ss = PorterStemmer()
    lst = [ss.stem(w) for w in token_list]
    return lst

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(my_tags))
    target_names = my_tags
    plt.xticks(tick_marks, target_names, rotation=45)
    plt.yticks(tick_marks, target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

def evaluate_prediction(predictions, target, title="Confusion matrix"):
    print('accuracy %s' % accuracy_score(target, predictions))
    cm = confusion_matrix(target, predictions, labels=my_tags)
    print('confusion matrix\n %s' % cm)
    print('(row=expected, col=predicted)')

    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plot_confusion_matrix(cm_normalized, title + ' Normalized')

def predict(vectorizer, classifier, data):
    data_features = vectorizer.transform(data['plot'])
    predictions = classifier.predict(data_features)
```

```
target = data['tag']
evaluate_prediction(predictions, target)
```

In [35]:
```
import re
import sys
import nltk
sys.path.append("../py")
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer


def remove_users(df, col):
    df[col] = df[col].apply(lambda x: re.sub(r'(RT\s@[A-Za-z]+[A-Za-z0-9-_]+)', '', str(x
    df[col] = df[col].apply(lambda x: re.sub(r'(@[A-Za-z0-9-_]+)', '', str(x))) # remove 

def remove_special_char(df, col):
    df[col] = df[col].apply(lambda x: re.sub(r'&[\S]+?;', '', str(x)))
    df[col] = df[col].apply(lambda x: re.sub(r'[^\w\s]', r'', str(x)))
    df[col] = df[col].apply(lambda x: re.sub(r'#', ' ', str(x)))

def remove_links(df, col):
    df[col] = df[col].apply(lambda x: re.sub(r'http\S+', '', str(x)))  # remove http link
    df[col] = df[col].apply(lambda x: re.sub(r'bit.ly/\S+', '', str(x)))  # remove bit.ly

def remove_numerics(df, col):
    """function to remove numbers or words with digits"""
    df[col] = df[col].apply(lambda x: re.sub(r'\w*\d\w*', r'', str(x)))

def remove_whitespaces(df, col):
    """function to remove any double or more whitespaces to single and any leading and tra
    df[col] = df[col].apply(lambda x: re.sub(r'\s\s+', ' ', str(x)))
    df[col] = df[col].apply(lambda x: re.sub(r'(\A\s+|\s+\Z)', '', str(x)))

def lemmatize(token):
    """Returns lemmatization of a token"""
    return WordNetLemmatizer().lemmatize(token, pos='v')

def tokenize(tweet):
    """Returns tokenized representation of words in lemma form excluding stopwords"""
    result = []
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(tweet)
    for token in word_tokens:
        if token.lower not in stop_words and len(token) > 2:  # drops words with less tha
            result.append(lemmatize(token))
    return result


def preprocess_tweets(df, col):
    """master function to preprocess tweets"""
    remove_users(df, col)
    remove_links(df, col)
    remove_special_char(df, col)
    remove_whitespaces(df, col)
    remove_numerics(df, col)
    tokenize_and_lemmatize(df, col)
    return df


def preprocess(tweet):
    result = re.sub(r'(RT\s@[A-Za-z]+[A-Za-z0-9-_]+)', '', tweet)
    result = re.sub(r'(@[A-Za-z0-9-_]+)', '', result)
    result = re.sub(r'http\S+', '', result)
    result = re.sub(r'bit.ly/\S+', '', result)
    # result = re.sub(r'(.)\1+', r'\1\1', result)
    result = " ".join(re.findall('[A-Z][^A-Z]*', result))
    result = re.sub(r'&[\S]+?;', '', result)
    result = re.sub(r'#', ' ', result)
```

```
    result = re.sub(r'[^\w\s]', r'', result)
    result = re.sub(r'\w*\d\w*', r'', result)
    result = re.sub(r'\s\s+', ' ', result)
    result = re.sub(r'(\A\s+|\s+\Z)', '', result)
    result = tokenize(result)
    return list(result)
```

In [36]:

```
 nltk
ltk.corpus import stopwords
ltk.tokenize import word_tokenize, sent_tokenize
ltk.stem import PorterStemmer, WordNetLemmatizer
ltk import tokenize
ltk.corpus import subjectivity
ltk.sentiment import SentimentAnalyzer, SentimentIntensityAnalyzer
ltk.sentiment.util import *

it learn
klearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
 numpy as np
klearn.metrics import (accuracy_score, roc_auc_score, confusion_matrix, roc_curve, auc,
                       mean_squared_error, log_loss, precision_recall_curve, classificatio
                       precision_recall_fscore_support,ConfusionMatrixDisplay)
klearn import preprocessing
klearn.model_selection import train_test_split, GridSearchCV, validation_curve
klearn.ensemble import RandomForestClassifier
klearn.linear_model import LogisticRegression, SGDClassifier, RidgeClassifier, Perceptron,
klearn.naive_bayes import BernoulliNB, ComplementNB, MultinomialNB
klearn.pipeline import make_pipeline
 matplotlib.pyplot as plt

im
 gensim
 gensim.corpora as corpora
ensim.utils import simple_preprocess
ensim.models import CoherenceModel
ensim.models import word2vec

ollections import Counter
otlib inline
 matplotlib.pyplot as plt
 seaborn as sns
print import pprint

ity
ime import time
 pandas as pd
 sqlite3
 regex as re
 matplotlib.pyplot as plt
ordcloud import WordCloud
```

In [37]:
```
import re
import sys
import nltk
sys.path.append("../py")
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer


def remove_users(df, col):
    df[col] = df[col].apply(lambda x: re.sub(r'(RT\s@[A-Za-z]+[A-Za-z0-9-_]+)', '', str(x
    df[col] = df[col].apply(lambda x: re.sub(r'(@[A-Za-z0-9-_]+)', '', str(x))) # remove 

def remove_special_char(df, col):
    df[col] = df[col].apply(lambda x: re.sub(r'&[\S]+?;', '', str(x)))
    df[col] = df[col].apply(lambda x: re.sub(r'[^\w\s]', r'', str(x)))
    df[col] = df[col].apply(lambda x: re.sub(r'#', ' ', str(x)))

def remove_links(df, col):
    df[col] = df[col].apply(lambda x: re.sub(r'http\S+', '', str(x)))  # remove http link
    df[col] = df[col].apply(lambda x: re.sub(r'bit.ly/\S+', '', str(x)))  # remove bit.ly

def remove_numerics(df, col):
    """function to remove numbers or words with digits"""
    df[col] = df[col].apply(lambda x: re.sub(r'\w*\d\w*', r'', str(x)))

def remove_whitespaces(df, col):
    """function to remove any double or more whitespaces to single and any leading and tra
    df[col] = df[col].apply(lambda x: re.sub(r'\s\s+', ' ', str(x)))
    df[col] = df[col].apply(lambda x: re.sub(r'(\A\s+|\s+\Z)', '', str(x)))

def lemmatize(token):
    """Returns lemmatization of a token"""
    return WordNetLemmatizer().lemmatize(token, pos='v')

def tokenize(tweet):
    """Returns tokenized representation of words in lemma form excluding stopwords"""
    result = []
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(tweet)
    for token in word_tokens:
        if token.lower not in stop_words and len(token) > 2:  # drops words with less tha
            result.append(lemmatize(token))
    return result


def preprocess_tweets(df, col):
    """master function to preprocess tweets"""
    remove_users(df, col)
    remove_links(df, col)
    remove_special_char(df, col)
    remove_whitespaces(df, col)
    remove_numerics(df, col)
    tokenize_and_lemmatize(df, col)
    return df


def preprocess(tweet):
    result = re.sub(r'(RT\s@[A-Za-z]+[A-Za-z0-9-_]+)', '', tweet)
    result = re.sub(r'(@[A-Za-z0-9-_]+)', '', result)
    result = re.sub(r'http\S+', '', result)
    result = re.sub(r'bit.ly/\S+', '', result)
    # result = re.sub(r'(.)\1+', r'\1\1', result)
    result = " ".join(re.findall('[A-Z][^A-Z]*', result))
    result = re.sub(r'&[\S]+?;', '', result)
    result = re.sub(r'#', ' ', result)
```

```
    result = re.sub(r'[^\w\s]', r'', result)
    result = re.sub(r'\w*\d\w*', r'', result)
    result = re.sub(r'\s\s+', ' ', result)
    result = re.sub(r'(\A\s+|\s+\Z)', '', result)
    result = tokenize(result)
    return list(result)
```

In [38]:
```
def lemmatize(token):
    """Returns lemmatization of a token"""
    return WordNetLemmatizer().lemmatize(token, pos='v')

def tokenize(tweet):
    """Returns tokenized representation of words in lemma form excluding stopwords"""
    result = []
    for token in gensim.utils.simple_preprocess(tweet):
        if token not in gensim.parsing.preprocessing.STOPWORDS \
                and len(token) > 2:  # drops words with less than 3 characters
            result.append(lemmatize(token))
    return result

def tokenize_and_lemmatize(df, col):
    df[col] = df[col].apply(lambda x: tokenize(x))
#     df.tweet = df.tweet.apply(lambda x: str(x)[1:-1])
```

In [39]:
```
preprocess_tweets(df, 'tweet')
df.head()
```

Out[39]:

id	count	hate	offensive	neutral	target	tweet	word_ct	char_ct	avg_wrd	hash_ct
1	3	0	0	3	0	[woman, shouldnt, complain, clean, house, man,...	25	140	4.640000	0
2	3	0	3	0	0	[boy, dats, coldtyga, dwn, bad, cuffin, dat, h...	16	85	4.375000	0
3	3	0	3	0	0	[dawg, fuck, bitch, start, confuse, shit]	21	120	4.761905	0
4	3	0	2	1	0	[look, like, tranny]	9	62	6.000000	0
5	6	0	6	0	0	[shit, hear, true, faker, bitch, tell]	26	137	4.307692	1

Here is the thought process involved with each of the specific steps we identified working with the dataset to prepare the data for the modeling process:

- We removed callouts or usernames, which is preceded by @. They contain no useful information.
- We removed character references, which includes HTML character references, but also emojis, unicode characters. We decided not to convert any emojis into sentiment words.
- We removed the hash from the hashtags and decided to keep the hashtag text because they are often words or word-like and are used to connect similar ideas across the platform. We could analyze the hashtags in a future project.
- We removed the Twitter codes RT and QT for retweet and quotetweet. We decided to keep the retweeted words, as it conveys important information while others have removed all the text after RT.
- We removed the HTML links since a lot of users link a website reference as part of the tweet.
- We then removed any punctuation. We did not convert contractions into the uncontracted words.

- We then lowercased all the tweets for tokenizing.
- We removed any numbers and number containing words for tokenization and vectorizing.
- We removed any extra whitespace(s) between words and any leading and trailing whitespaces.

Additional steps before modeling includes stopword removal, tokenization, lemmatizing, stemming, and/or vectorizing.

In [54]:
```
# display first five rows of dataframe
df.head()
```

Out[54]:

id	count	hate	offensive	neutral	target	tweet	word_ct	char_ct	avg_wrd	hash_ct
1	3	0	0	3	0	[woman, shouldnt, complain, clean, house, man,...	25	140	4.640000	0
2	3	0	3	0	0	[boy, dats, coldtyga, dwn, bad, cuffin, dat, h...	16	85	4.375000	0
3	3	0	3	0	0	[dawg, fuck, bitch, start, confuse, shit]	21	120	4.761905	0
4	3	0	2	1	0	[look, like, tranny]	9	62	6.000000	0
5	6	0	6	0	0	[shit, hear, true, faker, bitch, tell]	26	137	4.307692	1

Train-Validation-Test Split

In [55]:
```
# separate dataframe into respective classes
hate = df[df.target == 1]
non_hate = df[df.target == 0]
```

In [56]:
```
# separate features from target variable for train_test_splt
X_h = hate.tweet
y_h = hate.target
X_nh = non_hate.tweet
y_nh = non_hate.target

# perform 75-25 training-validation split and 15-10 validation-testing split on dataset
X_h_tr, X_h_val, y_h_tr, y_h_val = train_test_split(X_h, y_h, test_size=0.25, random_state
X_h_val, X_h_tt, y_h_val, y_h_tt = train_test_split(X_h_val, y_h_val, test_size=0.4, rando
X_nh_tr, X_nh_val, y_nh_tr, y_nh_val = train_test_split(X_nh, y_nh, test_size=0.25, randor
X_nh_val, X_nh_tt, y_nh_val, y_nh_tt = train_test_split(X_nh_val, y_nh_val, test_size=0.4
```

In [57]:
```
# concatenate hate and non-hate dataframe to reform entire training dataset
X_tr = pd.concat((X_h_tr, X_nh_tr), ignore_index=True)
y_tr = pd.concat((y_h_tr, y_nh_tr), ignore_index=True)
train = pd.concat([X_tr, y_tr], axis=1)

# # remove brackets around the list to create a list of strings
train['tweet2'] = train.tweet.apply(lambda x: str(x)[1:-1])
train.head()
```

Out[57]:

	tweet	target	tweet2
0	[reject, constantly, house, threaten, rape, mo...	1	'reject', 'constantly', 'house', 'threaten', '...
1	[convince, lame, nigger, liver, believe, cuz, ...	1	'convince', 'lame', 'nigger', 'liver', 'believ...
2	[peace, fag, remember, best, lux, support, dro...	1	'peace', 'fag', 'remember', 'best', 'lux', 'su...
3	[haha, ight, nig, calm, yoself]	1	'haha', 'ight', 'nig', 'calm', 'yoself'
4	[tits, better, look, face, make, like, asian, ...	1	'tits', 'better', 'look', 'face', 'make', 'lik...

In [58]:
```
# concatenate hate and non-hate dataframes to reform entire validation dataset
X_val = pd.concat((X_h_val, X_nh_val), ignore_index=True)
y_val = pd.concat((y_h_val, y_nh_val), ignore_index=True)
val = pd.concat([X_val, y_val], axis=1)

# remove brackets around the list to create a list of string
val['tweet2'] = val.tweet.apply(lambda x: str(x)[1:-1])
val.head()
```

Out[58]:

	tweet	target	tweet2
0	[lbum, fotos, gaywrites, make, project, queer,...	1	'lbum', 'fotos', 'gaywrites', 'make', 'project...
1	[yay, america, israel, jew, hat, muslim, trash...	1	'yay', 'america', 'israel', 'jew', 'hat', 'mus...
2	[miss, ofay, friends, day, scar, recent, happe...	1	'miss', 'ofay', 'friends', 'day', 'scar', 'rec...
3	[trash, darkskin, nigga, steal, damn, garbage]	1	'trash', 'darkskin', 'nigga', 'steal', 'damn',...
4	[cody, call, people, nigger, hes, fuck, spaz]	1	'cody', 'call', 'people', 'nigger', 'hes', 'fu...

In [59]:
```
X_tt = pd.concat((X_h_tt, X_nh_tt), ignore_index=True)
y_tt = pd.concat((y_h_tt, y_nh_tt), ignore_index=True)
test = pd.concat([X_tt, y_tt], axis=1)

# remove brackets around the list to create a list of string
test['tweet2'] = test.tweet.apply(lambda x: str(x)[1:-1])
test.head()
```

Out[59]:

	tweet	target	tweet2
0	[johnny, rebel, nigger, day]	1	'johnny', 'rebel', 'nigger', 'day'
1	[favorite, nigger, work, plantation, remember,...	1	'favorite', 'nigger', 'work', 'plantation', 'r...
2	[go, prestigious, establishments, clearly, sup...	1	'go', 'prestigious', 'establishments', 'clearl...
3	[westvirginia, white, trash]	1	'westvirginia', 'white', 'trash'
4	[fuck, brett, farve, redneck, ass, stuckup, do...	1	'fuck', 'brett', 'farve', 'redneck', 'ass', 's...

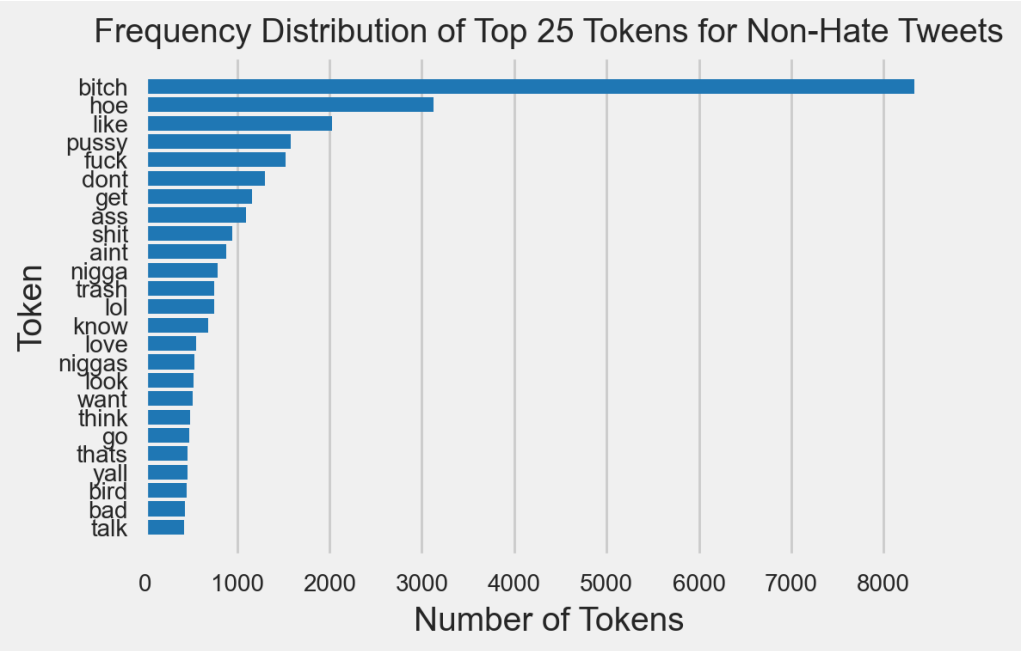
More EDA

Frequency Distributions

In [60]:
```
# split back into minority and majority classes for visualizations
zero = train[train.target == 0]
one = train[train.target == 1]
```

In [61]:
```
# create list of tokens for
zero_tokens = []
for index, row in zero.iterrows():
    zero_tokens.extend(row['tweet'])

one_tokens = []
for index, row in one.iterrows():
    one_tokens.extend(row['tweet'])
```

In [62]:
```
# convert collection of text documents to matrix of token counts
vec = CountVectorizer()

# learn vocabulary dictionary to return document-term matrix
docs = vec.fit_transform(zero_tokens)

# array mapping from feature integer indices to feature name
features = vec.get_feature_names()

# use Yellowbrick implementation of visualizing token frequency distribution
visualizer = FreqDistVisualizer(features=features, orient='h', n=25, size=(540, 360), colo
visualizer.fit(docs)
custom_viz = visualizer.ax
custom_viz.set_xlabel('Number of Tokens', fontsize=14)
custom_viz.set_ylabel('Token', fontsize=14)
custom_viz.set_title("Frequency Distribution of Top 25 Tokens for Non-Hate Tweets", fonts
custom_viz.figure.show()
```

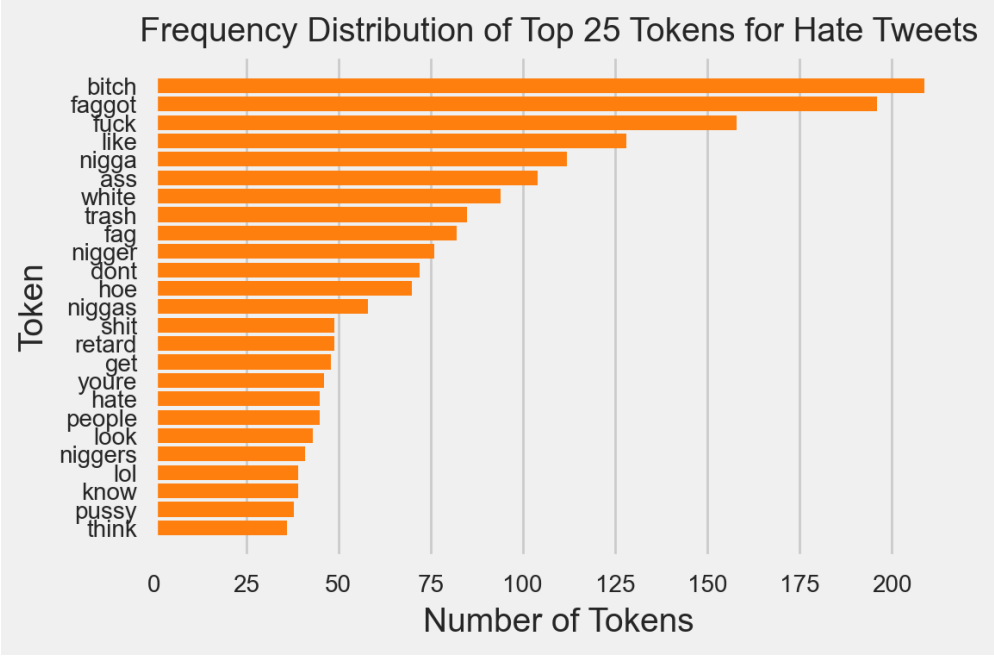


In [63]:
```
custom_viz.figure.savefig("images/freq_dist_zero.png")
```

Some of the top phrases in the negative class are: 'bitch', 'hoe', 'pussy', 'fuck', 'nigga'. 'Bitch' , 'fuck', and 'nigga' are 'Hoe' and 'pussy' appears in less frequency in the positive class.

Notably, 'nigger', 'white', 'trash', 'retard', 'queer', 'gay', 'fag' and 'faggot' are almost exclusively in the positive class.

In [64]:
```
# create visualization for positive class
vec_one = CountVectorizer()
docs_one = vec_one.fit_transform(one_tokens)
features_one = vec_one.get_feature_names()

visualizer_one = FreqDistVisualizer(features=features_one, orient='h', n=25, size=(540, 3
visualizer_one.fit(docs_one)
custom_viz_one = visualizer_one.ax
custom_viz_one.set_xlabel('Number of Tokens', fontsize=14)
custom_viz_one.set_ylabel('Token', fontsize=14)
custom_viz_one.set_title("Frequency Distribution of Top 25 Tokens for Hate Tweets", fonts
custom_viz_one.figure.show()
```



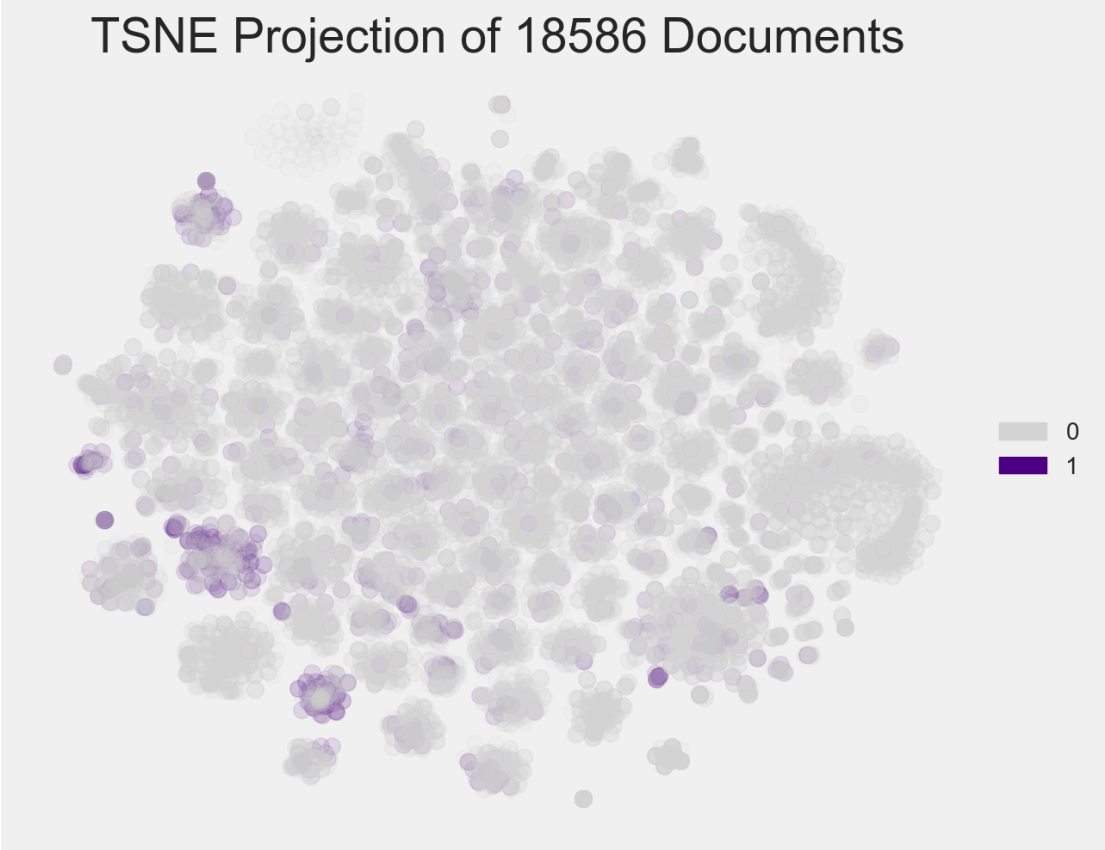
In [65]:
```
custom_viz_one.figure.savefig("images/freq_dist_one.png")
```

t-SNE Corpus Visualization

In [66]:
```
# create TSNE visualization for negative class
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(train.tweet2)
y = train.target
```

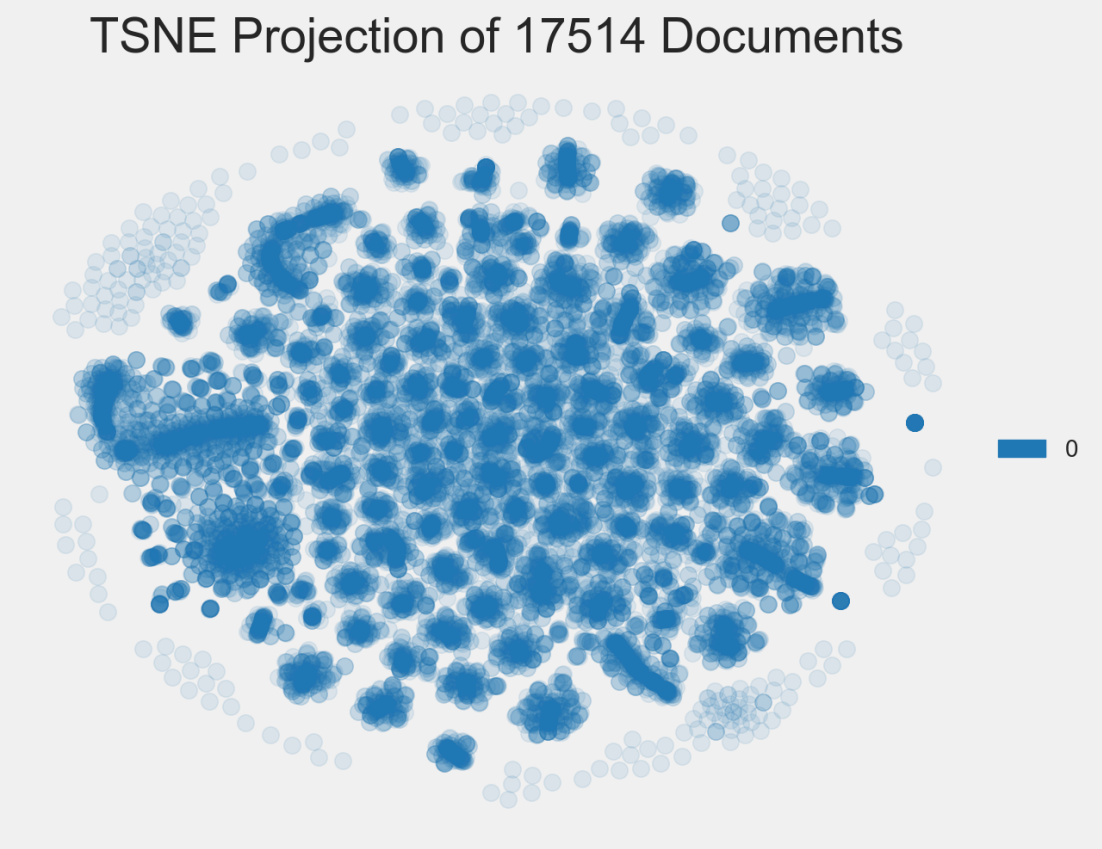
In [67]:
```
ualizer = TSNEVisualizer(alpha=0.1, colors=['lightgray', 'indigo'], decompose='svd', decom
ualizer.fit(X, y)
ualizer.show(outpath="/Users/Raj/NLP-Project/Automated-Hate-Tweet-Detection/images/tsne.pn
```

Out[67]:
```
<AxesSubplot:title={'center':'TSNE Projection of 18586 Documents'}>
```



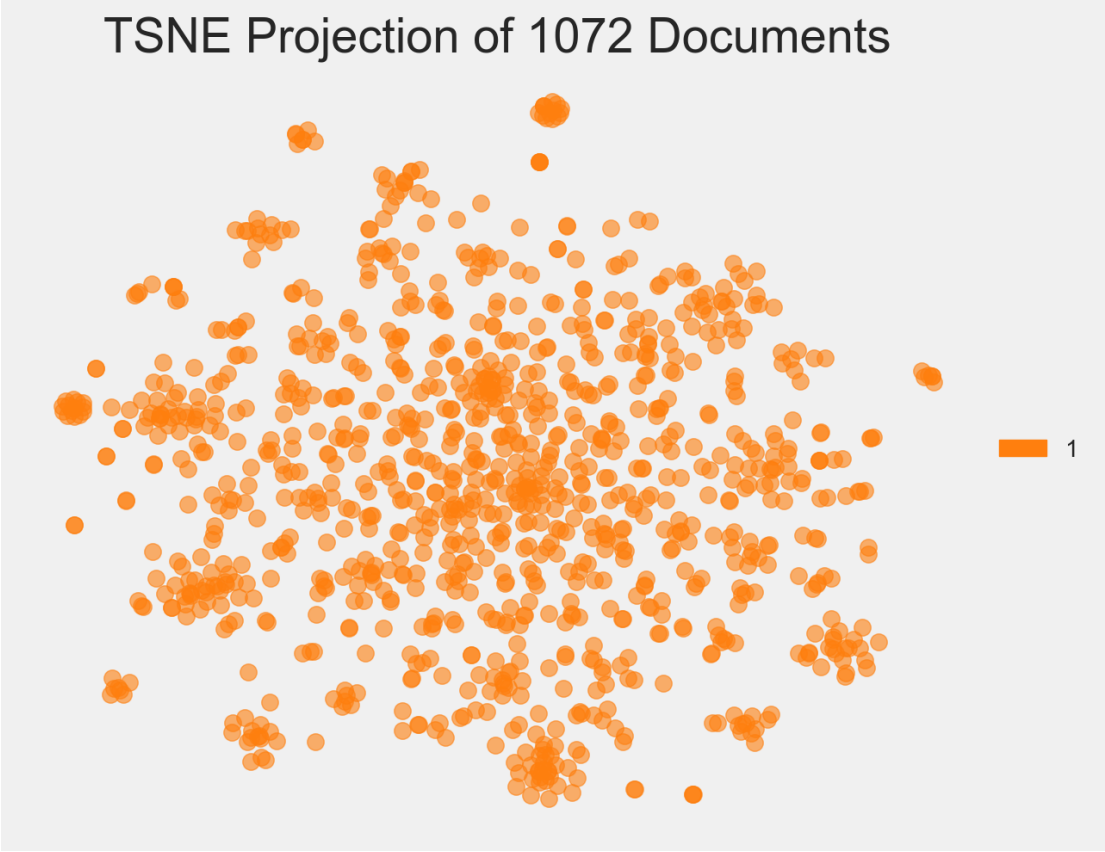
In [68]:
```
# create TSNE visualization for negative class
tfidf = TfidfVectorizer()
X_zero = tfidf.fit_transform(zero.tweet2)
y_zero = zero.target

visualizer = TSNEVisualizer(alpha=0.1, colors=['tab:blue'], decompose='svd', decompose_by=
visualizer.fit(X_zero, y_zero)
visualizer.show(outpath="images/tsne_zero.png")
```

Out[68]:
```
<AxesSubplot:title={'center':'TSNE Projection of 17514 Documents'}>
```



In [69]:
```
# create TSNE visualization for negative class
tfidf = TfidfVectorizer()
X_one = tfidf.fit_transform(one.tweet2)
y_one = one.target

visualizer = TSNEVisualizer(alpha=0.6, decompose='svd', colors=['tab:orange'], decompose_l
visualizer.fit(X_one, y_one)
visualizer.show(outpath="images/tsne_one.png")
```

Out[69]:
```
<AxesSubplot:title={'center':'TSNE Projection of 1072 Documents'}>
```

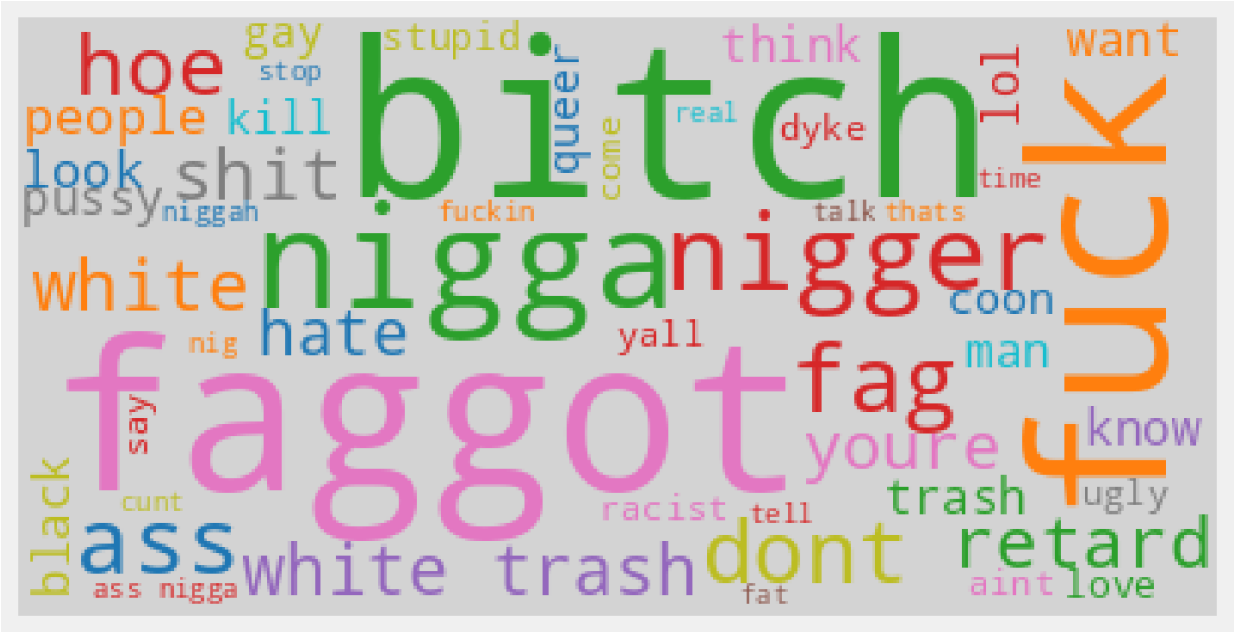


There is still strong overlap between the two classes as evinced by the diagrams. But there are clusters at the extremes (topmost, bottommost, far left, far right) that appear in the minority class TSNE that does not seem to appear in the majoity class.

Wordcloud

In [70]:
```
text = ' '.join(zero_tokens)

# Initialize wordcloud object
wc = WordCloud(background_color='lightgray', colormap='tab10', max_words=50)

# Generate and plot wordcloud
plt.imshow(wc.generate(text))
plt.axis('off')
plt.show()
```

In [71]:
```
text = ' '.join(one_tokens)

# Initialize wordcloud object
wc = WordCloud( background_color='lightgray', colormap='tab10', max_words=50)

# Generate and plot wordcloud
plt.imshow(wc.generate(text))
plt.axis('off')
plt.show()
```



In [89]:
```
tagged = nltk.pos_tag(tokens)
```

In [90]:
```
word_counter = {}
for word in tokens:
    token = str(word)
    if token in word_counter:
        word_counter[token] += 1
    else:
        word_counter[token] = 1
```

In [91]:
```
word_counter
```

Out[91]:
```
{'woman': 101,
 'shouldnt': 30,
 'complain': 46,
 'clean': 52,
 'house': 133,
 'man': 563,
 'trash': 1153,
 'boy': 189,
 'dats': 24,
 'coldtyga': 1,
 'dwn': 2,
 'bad': 568,
 'cuffin': 10,
 'dat': 290,
 'hoe': 4285,
 'place': 77,
 'dawg': 42,
 'fuck': 2249,
 'bitch': 11344,
```

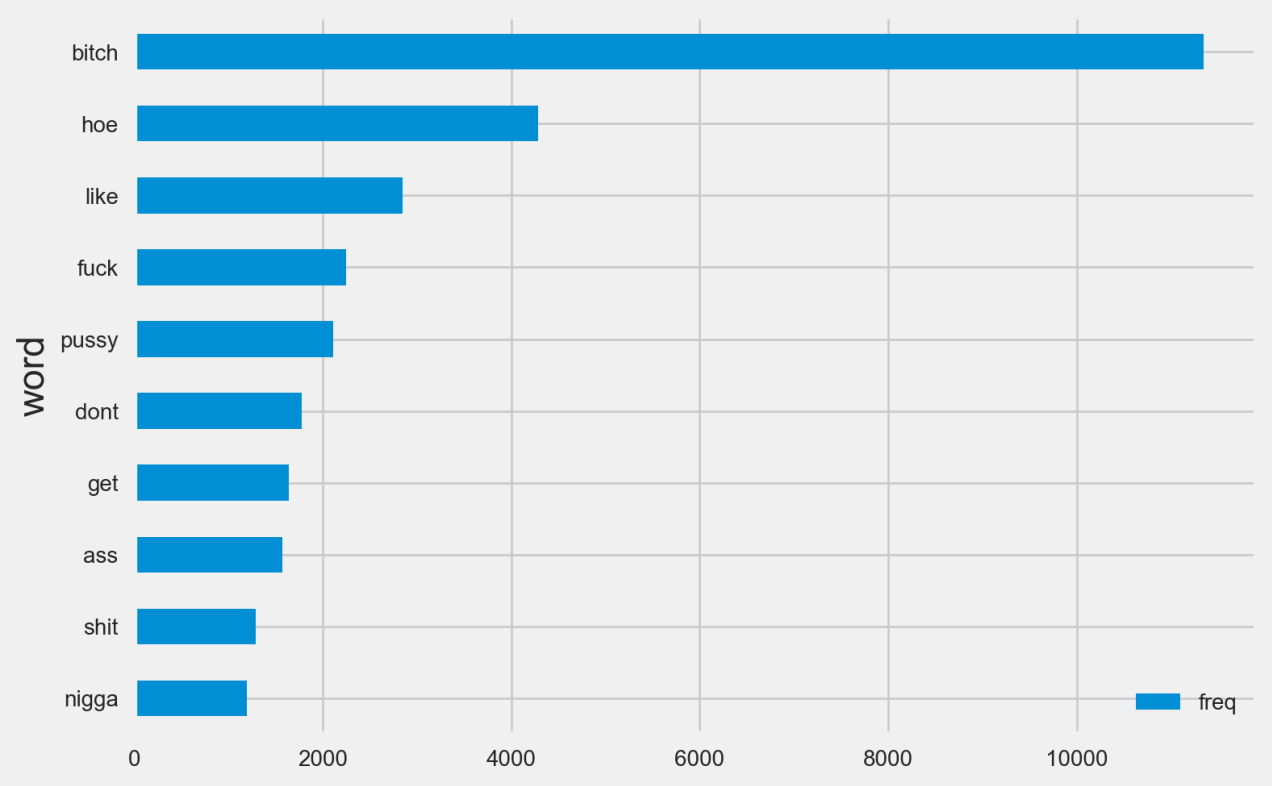
In [92]:
```
sorted_word_counts = sorted(word_counter.items(), key=lambda x:x[1], reverse=True)
sorted_word_counts
```

Out[92]:
```
[('bitch', 11344),
 ('hoe', 4285),
 ('like', 2844),
 ('fuck', 2249),
 ('pussy', 2109),
 ('dont', 1781),
 ('get', 1645),
 ('ass', 1570),
 ('shit', 1293),
 ('nigga', 1199),
 ('aint', 1169),
 ('trash', 1153),
 ('lol', 1067),
 ('know', 945),
 ('niggas', 792),
 ('love', 763),
 ('look', 762),
 ('want', 752),
 ('think', 686),
```

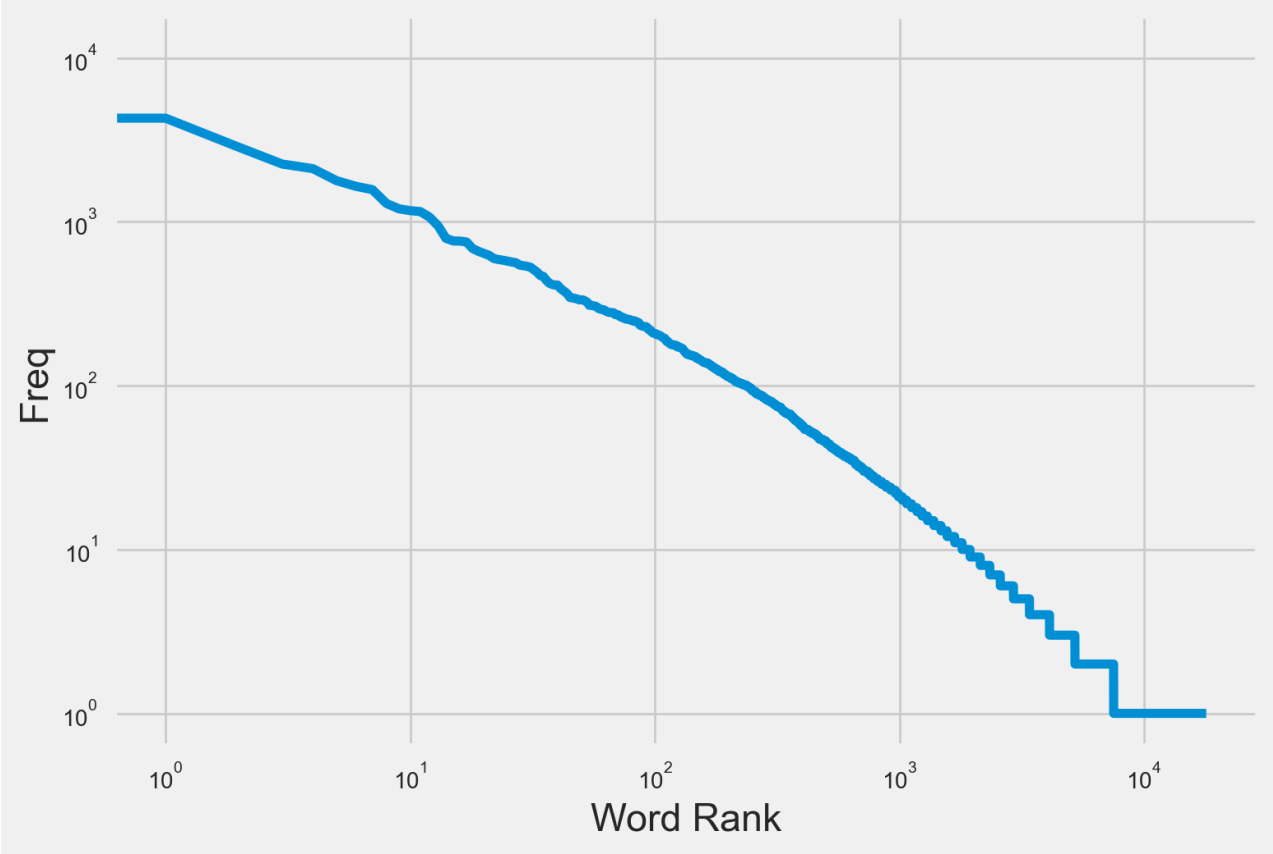
In [94]:
```
most_common_words = sorted_word_counts[0:10]
most_common_words = pd.DataFrame(most_common_words)
most_common_words.columns = ['word', 'freq']
```

In [95]:
```
most_common_words.sort_values(by='freq',ascending=True).plot(x='word', kind='barh')
```

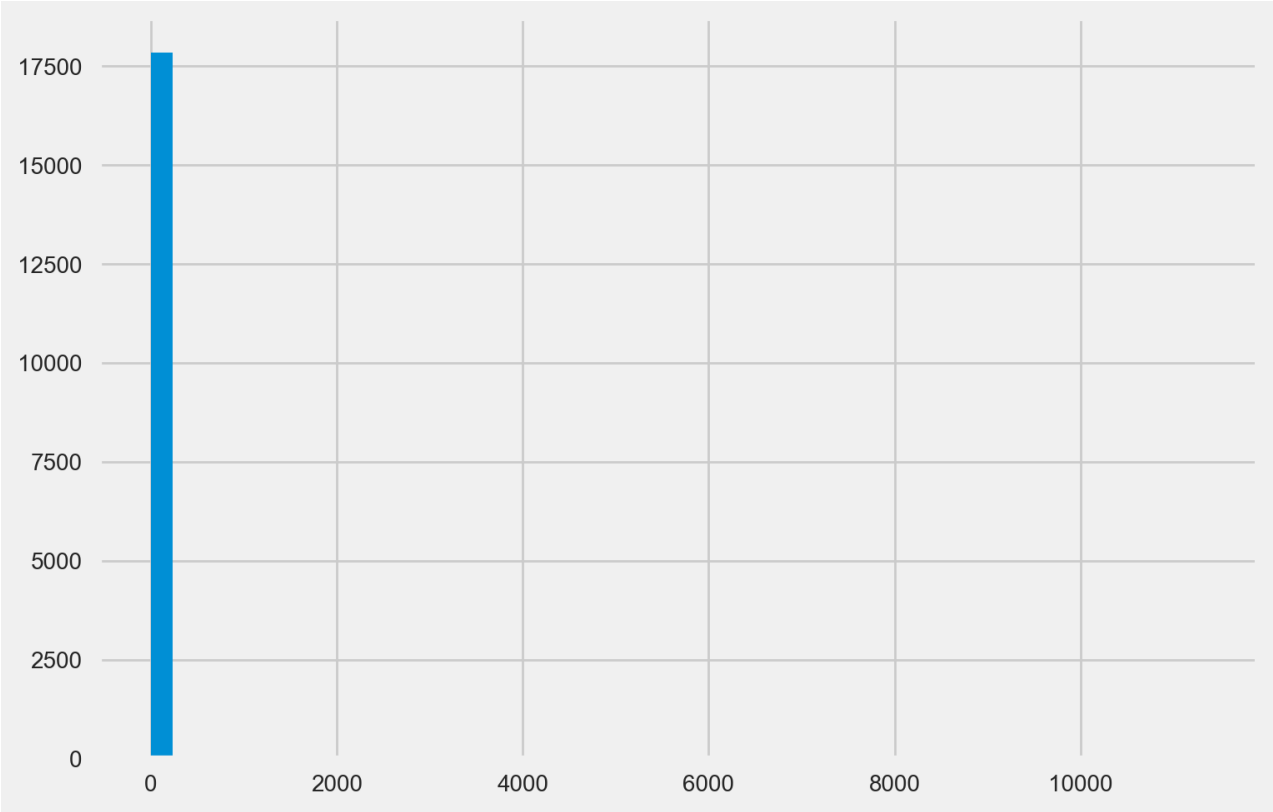
Out[95]:
```
<AxesSubplot:ylabel='word'>
```



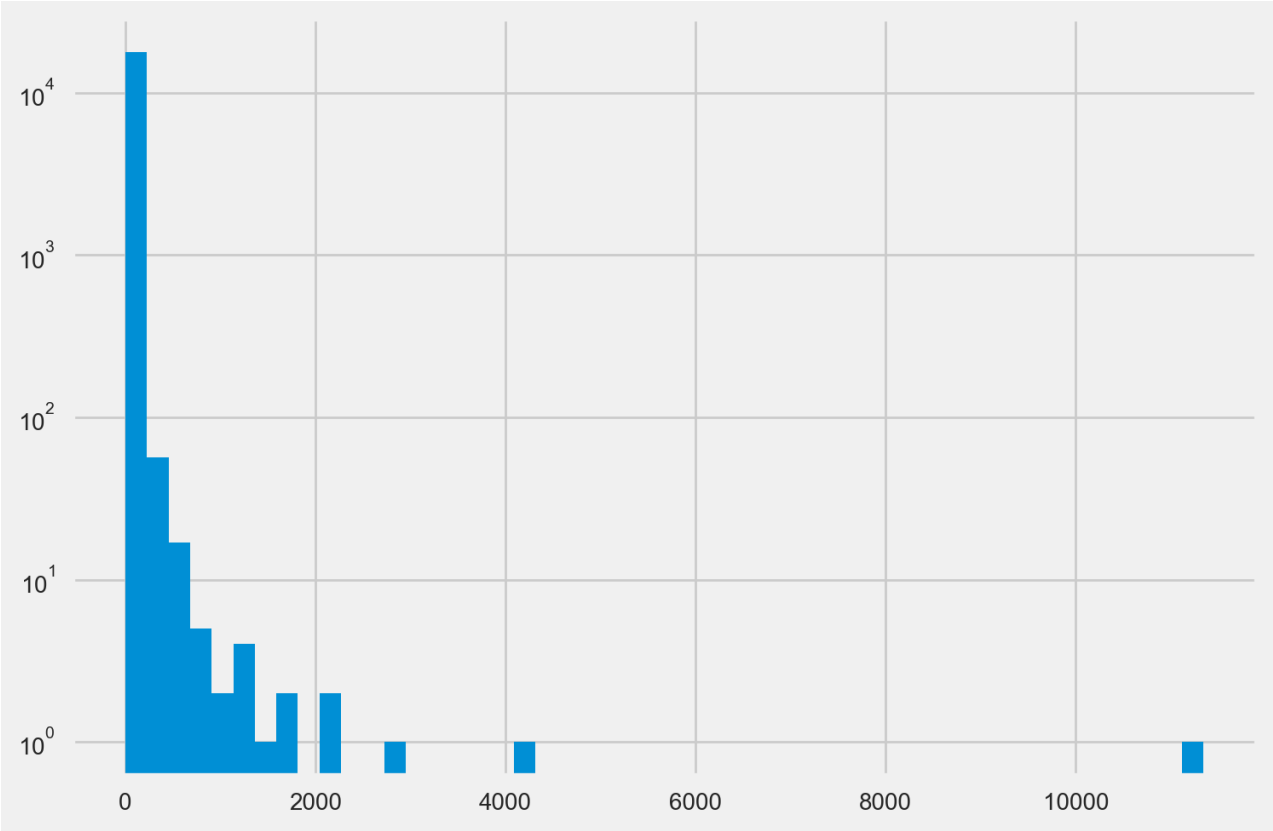
In [96]:
```
import matplotlib.pyplot as plt
sorted_word_counts = sorted(list(word_counter.values()), reverse=True)

plt.loglog(sorted_word_counts)
plt.ylabel("Freq")
plt.xlabel("Word Rank");
```



In [97]:
```
plt.hist(sorted_word_counts, bins=50);
```



In [98]:
```
plt.hist(sorted_word_counts, bins=50, log=True);
```



In [99]:
```
#sci-kit learn
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import numpy as np
from sklearn.metrics import (accuracy_score, roc_auc_score, confusion_matrix, roc_curve, a
                             mean_squared_error, log_loss, precision_recall_curve, classi
                             precision_recall_fscore_support,ConfusionMatrixDisplay)
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV, validation_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier, RidgeClassifier, Perc
from sklearn.naive_bayes import BernoulliNB, ComplementNB, MultinomialNB
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt
```

In [100]:
```
import nltk

SOS = "<s> "
EOS = "</s>"
UNK = "<UNK>"

def add_sentence_tokens(sentences, n):

    sos = SOS * (n-1) if n > 1 else SOS
    return ['{}{} {}'.format(sos, s, EOS) for s in sentences]

def replace_singletons(tokens):

    vocab = nltk.FreqDist(tokens)
    return [token if vocab[token] > 1 else UNK for token in tokens]

def preprocess(sentences, n):
    """Add SOS/EOS/UNK tokens to given sentences and tokenize.

    Args:
        sentences (list of str): the sentences to preprocess.
        n (int): order of the n-gram model which will use these sentences.
    Returns:
        The preprocessed sentences, tokenized by words.

    """
    sentences = add_sentence_tokens(sentences, n)
    tokens = ' '.join(sentences).split(' ')
    tokens = replace_singletons(tokens)
    return tokens
```

In [101]:
```
class LanguageModel(object):
    """An n-gram language model trained on a given corpus.

    For a given n and given training corpus, constructs an n-gram language
    model for the corpus by:
    1. preprocessing the corpus (adding SOS/EOS/UNK tokens)
    2. calculating (smoothed) probabilities for each n-gram

    Also contains methods for calculating the perplexity of the model
    against another corpus, and for generating sentences.

    Args:
        train_data (list of str): list of sentences comprising the training corpus.
        n (int): the order of language model to build (i.e. 1 for unigram, 2 for bigram, e
        laplace (int): lambda multiplier to use for laplace smoothing (default 1 for add-1

    """

    def __init__(self, text_list, n, laplace=1):
        self.n = n
        self.laplace = laplace
        self.tokens = tokens
        self.vocab  = nltk.FreqDist(self.tokens)
        self.model  = self._create_model()

    def _smooth(self):
        """Apply Laplace smoothing to n-gram frequency distribution.

        Here, n_grams refers to the n-grams of the tokens in the training corpus,
        while m_grams refers to the first (n-1) tokens of each n-gram.

        Returns:
            dict: Mapping of each n-gram (tuple of str) to its Laplace-smoothed
            probability (float).

        """
        vocab_size = len(self.vocab)
        self.tokens = preprocess(text_list, ngram_size)
        n_grams = nltk.ngrams(self.tokens, self.n)
        n_vocab = nltk.FreqDist(n_grams)

        m_grams = nltk.ngrams(self.tokens, self.n-1)
        m_vocab = nltk.FreqDist(m_grams)

        def smoothed_count(n_gram, n_count):
            m_gram = n_gram[:-1]
            m_count = m_vocab[m_gram]
            return (n_count + self.laplace) / (m_count + self.laplace * vocab_size)

        return { n_gram: smoothed_count(n_gram, count) for n_gram, count in n_vocab.items

    def _create_model(self):
        """Create a probability distribution for the vocabulary of the training corpus.

        If building a unigram model, the probabilities are simple relative frequencies
        of each token with the entire corpus.

        Otherwise, the probabilities are Laplace-smoothed relative frequencies.

        Returns:
            A dict mapping each n-gram (tuple of str) to its probability (float).

        """
        if self.n == 1:
            num_tokens = len(self.tokens)
            return { (unigram,): count / num_tokens for unigram, count in self.vocab.items
```

```
        else:
            return self._smooth()
```

The traditional epithets are not found in exclusively in the hate category, only the less traditional words often in the form of hashtags can be found exclusively as hate speech. That would make sense. in terms pf

- sexual orientation: teabagged, girlboy, azflooding, azmonsoon, molester, cousintoucher, theyfaggots, dicklicker
- sex: wenchs
- race/ethnicity/religion: osamas, spicskkk, niggerous, nigglets. nigress, ovenjew, westvirginia, texarkana, ching, chong, maoists, mexicannigger

One clear distinction is the difference in use of nigga versus the n word. When people say the f word against homosexuals, it is more often in the derogatory sense. The p word can be just offensive or sexist, i.e. males use the p word to denigrate guys, which can be offensive but not considered hate speech.

Modeling Process:

Back to top

In [76]:
```
train.tweet2
```

Out[76]:
```
0        'reject', 'constantly', 'house', 'threaten', '...
1        'convince', 'lame', 'nigger', 'liver', 'believ...
2        'peace', 'fag', 'remember', 'best', 'lux', 'su...
3                  'haha', 'ight', 'nig', 'calm', 'yoself'
4        'tits', 'better', 'look', 'face', 'make', 'lik...
                              ...
18581                              'miss', 'lil', 'bitch'
18582    'gotta', 'hoe', 'smh', 'aint', 'captain', 'sav...
18583        'lmao', 'yeah', 'bitch', 'lil', 'shit', 'rip'
18584                               'tbt', 'bad', 'bitch'
18585                  'hoe', 'act', 'know', 'imma', 'let'
Name: tweet2, Length: 18586, dtype: object
```

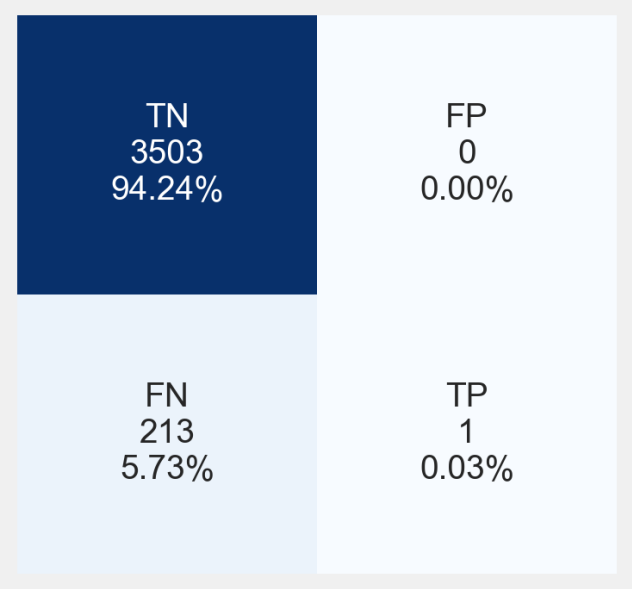
In [77]:
```
# assign feature and target variables
X_tr = train.tweet2
X_val = val.tweet2
y_tr = train.target
y_val = val.target

# vectorize tweets for modeling
vec = TfidfVectorizer()
tfidf_tr = vec.fit_transform(X_tr)
tfidf_val = vec.transform(X_val)
```

In [78]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss
```

Multinomial Naive Bayes

In [103]:
```
nb = MultinomialNB().fit(tfidf_tr, y_tr)
y_pred_nb = nb.predict(tfidf_val)
get_metrics_confusion(tfidf_val, y_val, y_pred_nb, nb)
```

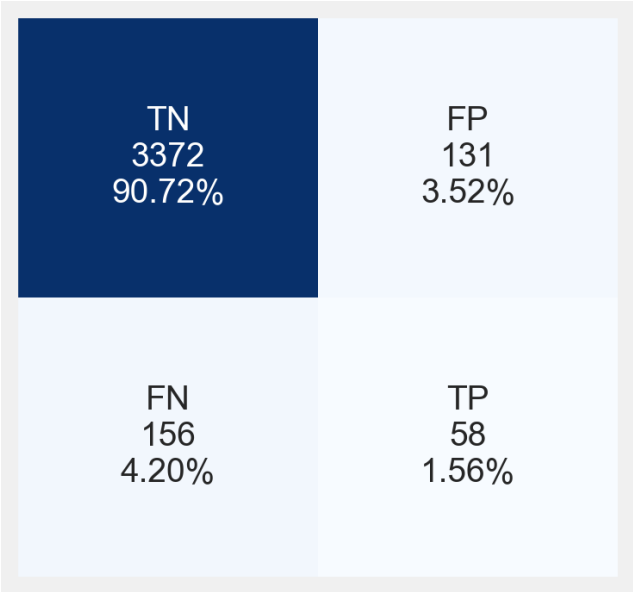
```
Accuracy:  0.9426957223567393
F1 Score:  0.009302325581395347
ROC-AUC:  0.7158910519954859
Recall:  0.004672897196261682
Precision:  1.0
PR-AUC:  0.17537404097663506
Log loss:  1.9792196319924609
```



Decision Tree Classifier

In [104]:
```
dt = DecisionTreeClassifier().fit(tfidf_tr, y_tr)
y_pr_dt_tr = dt.predict(tfidf_tr)
y_pr_dt_val = dt.predict(tfidf_val)
get_metrics_confusion(tfidf_val, y_val, y_pr_dt_val, nb)
# get_metrics(tfidf_tr, y_tr, tfidf_val, y_val, y_pr_dt_tr, y_pr_dt_val, dt)
```

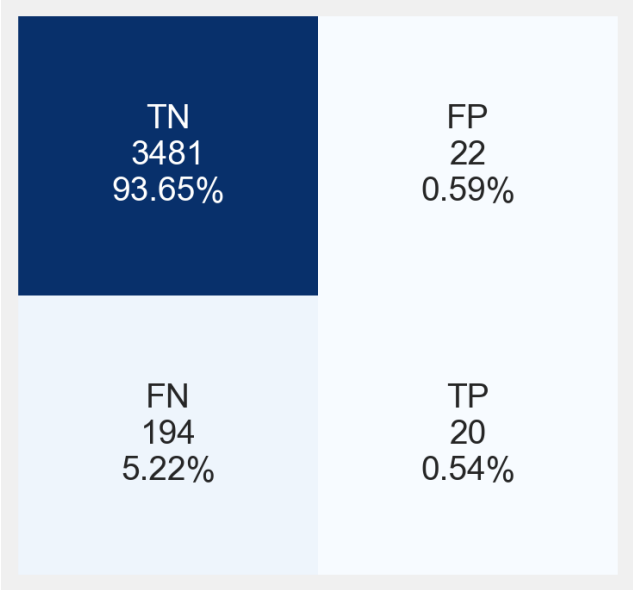
```
Accuracy:  0.9227871939736346
F1 Score:  0.2878411910669975
ROC-AUC:  0.7158910519954859
Recall:  0.27102803738317754
Precision:  0.30687830687830686
PR-AUC:  0.17537404097663506
Log loss:  2.666864022761023
```



Random Forest

In [105]:
```
rf = RandomForestClassifier(n_estimators=100).fit(tfidf_tr, y_tr)
y_pred_rf = rf.predict(tfidf_val)
get_metrics_confusion(tfidf_val, y_val, y_pred_rf, rf)
```

```
Accuracy:  0.9418886198547215
F1 Score:  0.15625
ROC-AUC:  0.8270914649926232
Recall:  0.09345794392523364
Precision:  0.47619047619047616
PR-AUC:  0.3209943416834018
Log loss:  2.00710069745606
```



In [106]:
```
mse = (y_val-y_pred_rf)**2
print(f"MSE: {mse.mean():0.2f} (+/- {mse.std():0.2f})")
```

```
MSE: 0.06 (+/- 0.23)
```

In [107]:
```
mae = np.abs(y_val-y_pred_rf)

print(f"MAE: {mae.mean():0.2f} (+/- {mae.std():0.2f})")
```

```
MAE: 0.06 (+/- 0.23)
```

In [108]:
```
mse = (y_val-y_pred_rf)**2

rmse = np.sqrt(mse.mean())

print(f"RMSE: {rmse:0.2f}")
```

```
RMSE: 0.24
```

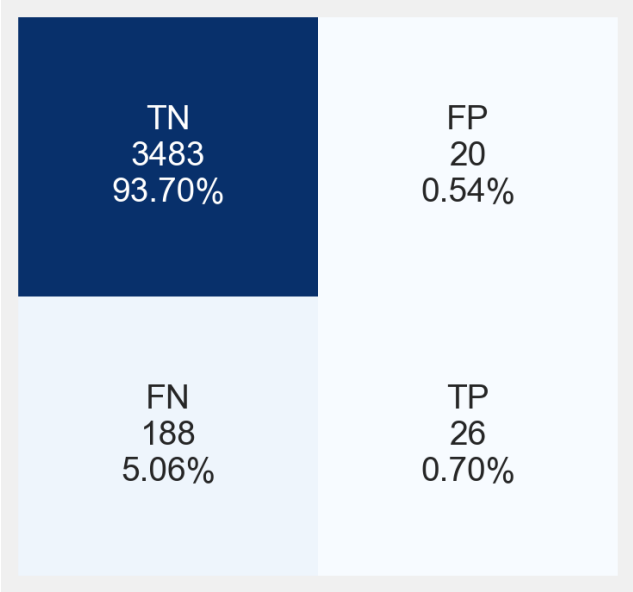
In [109]:
```
# R^2 coefficient of determination
SE_line = sum((y_val-y_pred_rf)**2)
SE_mean = sum((y_val-y_val.mean())**2)

r2 = 1-(SE_line/SE_mean)

print(f"R^2 coefficient of determination: {r2*100:0.2f}%")
```

```
R^2 coefficient of determination: -7.10%
```

Logistic Regression

In [110]:
```
log = LogisticRegression().fit(tfidf_tr, y_tr)
y_pred_log = log.predict(tfidf_val)
get_metrics_confusion(tfidf_val, y_val, y_pred_log, log)
```

```
Accuracy:  0.9440408931934355
F1 Score:  0.2
ROC-AUC:  0.8740065257816397
Recall:  0.12149532710280374
Precision:  0.5652173913043478
PR-AUC:  0.35577806219015606
Log loss:  1.9327633796314303
```



In [111]:
```
mse = (y_val-y_pred_log)**2
print(f"MSE: {mse.mean():0.2f} (+/- {mse.std():0.2f})")
```

```
MSE: 0.06 (+/- 0.23)
```

In [112]:
```
mae = np.abs(y_val-y_pred_log)

print(f"MAE: {mae.mean():0.2f} (+/- {mae.std():0.2f})")
```

```
MAE: 0.06 (+/- 0.23)
```

In [113]:
```
mse = (y_val-y_pred_log)**2

rmse = np.sqrt(mse.mean())

print(f"RMSE: {rmse:0.2f}")
```

```
RMSE: 0.24
```

In [114]:
```
# R^2 coefficient of determination
SE_line = sum((y_val-y_pred_log)**2)
SE_mean = sum((y_val-y_val.mean())**2)

r2 = 1-(SE_line/SE_mean)

print(f"R^2 coefficient of determination: {r2*100:0.2f}%")
```

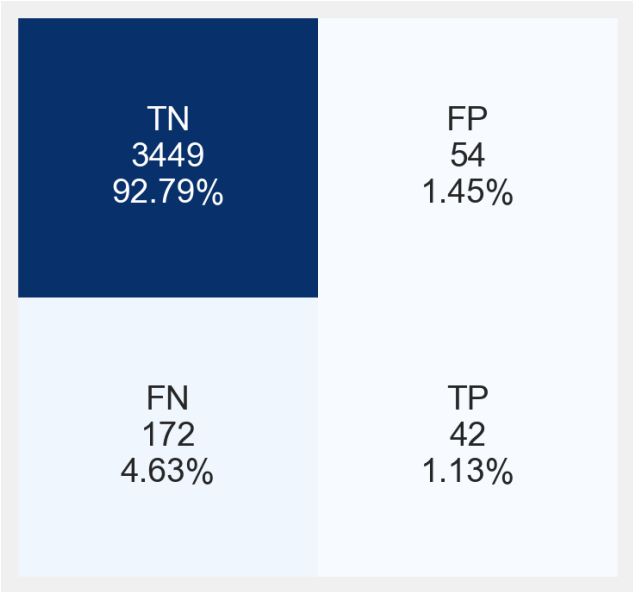
```
R^2 coefficient of determination: -3.13%
```

Support Vector Machine

In [115]:
```
svc = svm.LinearSVC(random_state=42).fit(tfidf_tr, y_tr)
y_pred_svc = svc.predict(tfidf_val)
get_metrics_2(tfidf_val, y_val, y_pred_svc, svc)
```

```
Accuracy:  0.9391982781813291
F1:  0.2709677419354839
Recall:  0.19626168224299065
Precision:  0.4375
ROC-AUC:  0.8362738480501359
PR-AUC:  0.32572730182740484
```

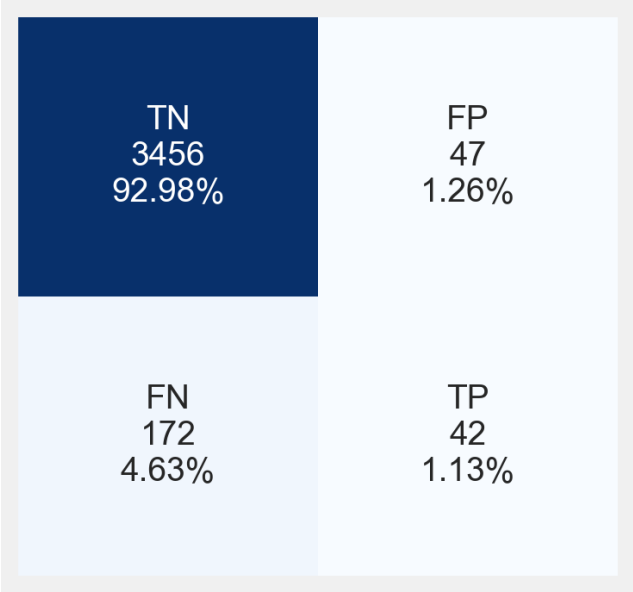
In [116]:
```
get_confusion(y_val, y_pred_svc)
```



Adaboost

In [117]:
```
abc = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),
    n_estimators=200
    ).fit(tfidf_tr, y_tr)
y_pred_abc = abc.predict(tfidf_val)
get_metrics_confusion(tfidf_val, y_val, y_pred_abc, abc)
```

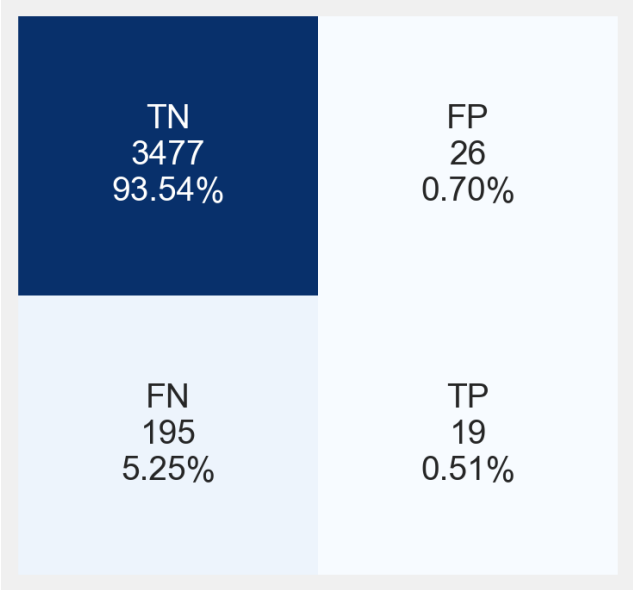
```
Accuracy:  0.9410815173527038
F1 Score:  0.2772277227722772
ROC-AUC:  0.8154605798501151
Recall:  0.19626168224299065
Precision:  0.47191011235955055
PR-AUC:  0.31202635048997956
Log loss:  2.0349824082767447
```



Gradient Boosting

In [118]:
```
gbc = GradientBoostingClassifier().fit(tfidf_tr, y_tr)
y_pred_gbc = gbc.predict(tfidf_val)
get_metrics_confusion(tfidf_val, y_val, y_pred_gbc, gbc)
```

```
Accuracy:  0.9405434490180253
F1 Score:  0.1467181467181467
ROC-AUC:  0.8394506977997499
Recall:  0.08878504672897196
Precision:  0.4222222222222222
PR-AUC:  0.3344834312611418
Log loss:  2.0535621126737817
```

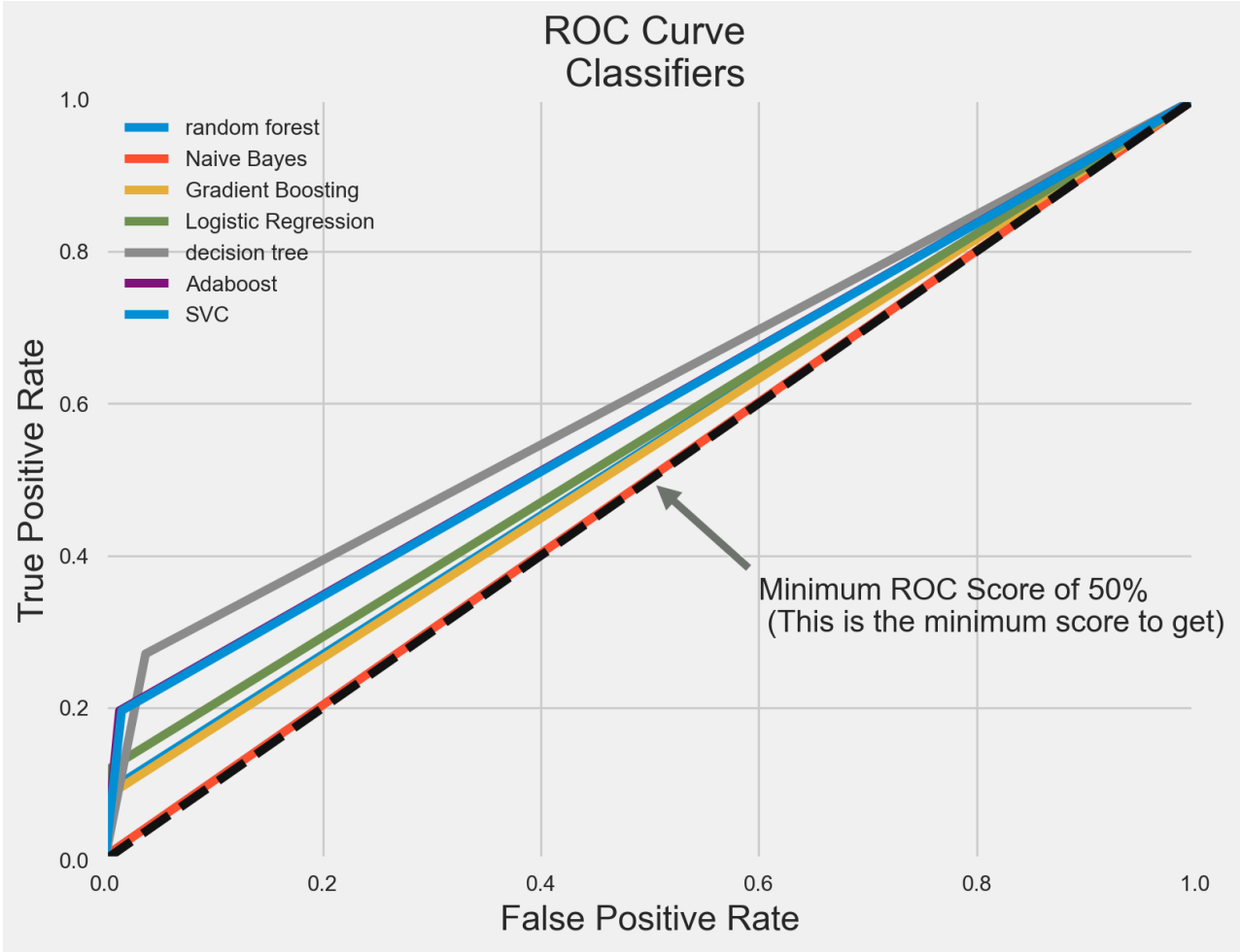


In [119]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

print('Logistic Regression Score: ', roc_auc_score(y_val, y_pred_log))
print('Naive Bayes Score: ', roc_auc_score(y_val, y_pred_nb))
print('Random Forest Score: ', roc_auc_score(y_val, y_pred_rf))
print('Decision Tree Score: ', roc_auc_score(y_val, y_pr_dt_val))
print('Grad Boosting Score: ', roc_auc_score(y_val, y_pred_gbc))
print('Adaboost Score: ', roc_auc_score(y_val, y_pred_abc))
print('SVC Score: ', roc_auc_score(y_val, y_pred_svc))
```

```
Logistic Regression Score:  0.5578929675765232
Naive Bayes Score:  0.5023364485981309
Random Forest Score:  0.5435888063902503
Decision Tree Score:  0.6168157600561335
Grad Boosting Score:  0.5406814185971437
Adaboost Score:  0.5914223055805303
SVC Score:  0.5904231619893229
```

In [120]:
```
from sklearn.metrics import roc_curve

log_fpr, log_tpr, threshold = roc_curve(y_val, y_pred_log)
nav_fpr, nav_tpr, threshold = roc_curve(y_val, y_pred_nb)
rand_fpr, rand_tpr, thresold = roc_curve(y_val, y_pred_rf)
dt_fpr, dt_tpr, threshold = roc_curve(y_val, y_pr_dt_val)
grad_fpr, grad_tpr, threshold = roc_curve(y_val, y_pred_gbc)
ada_fpr, ada_tpr, thresold = roc_curve(y_val, y_pred_abc)
svc_fpr, svc_tpr, thresodl = roc_curve(y_val, y_pred_svc)
```

In [121]:
```
oh_roc_curve_multiple(rand_fpr, rand_tpr, nav_fpr, nav_tpr,grad_fpr, grad_tpr, log_fpr, lo
.figure(figsize=(8,6))
.title('ROC Curve \n Classifiers', fontsize=18)
.plot(rand_fpr, rand_tpr, label='random forest')
.plot(nav_fpr, nav_tpr, label='Naive Bayes')
.plot(grad_fpr, grad_tpr, label='Gradient Boosting')
.plot(log_fpr, log_tpr, label='Logistic Regression')
.plot(dt_fpr, dt_tpr, label='decision tree')
.plot(ada_fpr, ada_tpr, label='Adaboost')
.plot(svc_fpr, svc_tpr, label='SVC')

.plot([0, 1], [0, 1], 'k--')
.axis([0, 1, 0, 1])
.xlabel('False Positive Rate', fontsize=16)
.ylabel('True Positive Rate', fontsize=16)
.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.5, 0.5),
        arrowprops=dict(facecolor='#6E726D', shrink=0.05),
        )
.legend()

oc_curve_multiple(rand_fpr, rand_tpr, nav_fpr, nav_tpr,grad_fpr, grad_tpr, log_fpr, log_tp
v()
```



In [127]:
```
# Import required libraries for performance metrics
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_validate

# Define dictionary with performance metrics
scoring = {'accuracy':make_scorer(accuracy_score),
           'precision':make_scorer(precision_score),
           'recall':make_scorer(recall_score),
           'f1_score':make_scorer(f1_score)}

# Import required libraries for machine learning classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

# Instantiate the machine learning classifiers
log_model = LogisticRegression(max_iter=10000)
svc_model = LinearSVC(dual=False)
dtr_model = DecisionTreeClassifier()
rfc_model = RandomForestClassifier()
gnb_model = MultinomialNB()
gbc_model = GradientBoostingClassifier()



# Define the models evaluation function
def models_evaluation(X, y, folds):

    '''
    X : data set features
    y : data set target
    folds : number of cross-validation folds

    '''

    # Perform cross-validation to each machine learning classifier
    log = cross_validate(log_model, X, y, cv=folds, scoring=scoring)
    svc = cross_validate(svc_model, X, y, cv=folds, scoring=scoring)
    dtr = cross_validate(dtr_model, X, y, cv=folds, scoring=scoring)
    rfc = cross_validate(rfc_model, X, y, cv=folds, scoring=scoring)
    gnb = cross_validate(gnb_model, X, y, cv=folds, scoring=scoring)
    gbc = cross_validate(gbc_model, X, y, cv=folds, scoring=scoring)

    # Create a data frame with the models perfoamnce metrics scores
    models_scores_table = pd.DataFrame({'Logistic Regression':[log['test_accuracy'].mean(
                                                               log['test_precision'].mean
                                                               log['test_recall'].mean(),
                                                               log['test_f1_score'].mean(

                                        'Support Vector Classifier':[svc['test_accuracy'].me
                                                                     svc['test_precision'].m
                                                                     svc['test_recall'].mea
                                                                     svc['test_f1_score'].me

                                        'Decision Tree':[dtr['test_accuracy'].mean(),
                                                         dtr['test_precision'].mean(),
                                                         dtr['test_recall'].mean(),
                                                         dtr['test_f1_score'].mean()],

                                        'Random Forest':[rfc['test_accuracy'].mean(),
```

```
                                        rfc['test_precision'].mean(),
                                        rfc['test_recall'].mean(),
                                        rfc['test_f1_score'].mean()],

                                 'Multinomial Naive Bayes':[gnb['test_accuracy'].mea
                                                            gnb['test_precision'].mean(
                                                            gnb['test_recall'].mean(),
                                                            gnb['test_f1_score'].mean()

                                  'Gradient Boosting classifier':[gbc['test_accuracy
                                                            gbc['test_precision'].mean(
                                                            gbc['test_recall'].mean(),
                                                            gbc['test_f1_score'].mean()

                                 index=['Accuracy', 'Precision', 'Recall', 'F1 Score

    # Add 'Best Score' column
    models_scores_table['Best Score'] = models_scores_table.idxmax(axis=1)

    # Return models performance metrics scores data frame
    return(models_scores_table)

# Run models_evaluation function
models_evaluation(tfidf_tr, y_tr, 5)
```

Out[127]:

	Logistic Regression	Support Vector Classifier	Decision Tree	Random Forest	Multinomial Naive Bayes	Gradient Boosting classifier	Best Score
Accuracy	0.944313	0.943075	0.926934	0.943452	0.941999	0.940600	Logistic Regression
Precision	0.620524	0.526611	0.324547	0.563765	0.000000	0.430069	Logistic Regression
Recall	0.089524	0.158557	0.247198	0.084882	0.000000	0.087711	Decision Tree
F1 Score	0.155845	0.242996	0.279618	0.147261	0.000000	0.145273	Decision Tree

In [ ]: