Importing Packages

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

```
Requirement already satisfied: cufflinks in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (0.17.3)
Requirement already satisfied: numpy>=1.9.2 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from cufflinks)
(1.20.0)
Requirement already satisfied: setuptools>=34.4.1 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from cuffl
inks) (67.3.2)
Requirement already satisfied: plotly>=4.1.1 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from cufflinks)
(5.9.0)
Requirement already satisfied: pandas>=0.19.2 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from cufflink
s) (1.4.4)
Requirement already satisfied: ipywidgets>=7.0.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from cuffli
nks) (7.6.5)
Requirement already satisfied: ipython>=5.3.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from cufflink
s) (7.31.1)
Requirement already satisfied: colorlover>=0.2.1 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from cuffli
nks) (0.3.0)
Requirement already satisfied: six>=1.9.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from cufflinks)
(1.16.0)
Requirement already satisfied: traitlets>=4.2 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from ipython>=
5.3.0->cufflinks) (5.1.1)
```

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, GradientBoostingClassifier
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/site-packages (2.0.0)
Requirement already satisfied: future in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from preprocess) (0.1
8.2)
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/site-packages (1.5)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (fro
m yellowbrick) (3.5.2)
Requirement already satisfied: scipy>=1.0.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from yellowbric
k) (1.9.1)
Requirement already satisfied: cycler>=0.10.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from yellowbri
ck) (0.11.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from yell
owbrick) (1.0.2)
Requirement already satisfied: numpy>=1.16.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from yellowbric
k) (1.20.0)
Requirement already satisfied: packaging>=20.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from matplotl
ib!=3.0.0,>=2.0.2->yellowbrick) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from matplot
lib!=3.0.0,>=2.0.2->yellowbrick) (3.0.9)
Requirement already satisfied: fonttools>=4.22.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from matplo
tlib!=3.0.0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from matplo
tlib!=3.0.0,>=2.0.2->yellowbrick) (1.4.2)
Requirement already satisfied: python-dateutil>=2.7 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from mat
plotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from matplotli
b!=3.0.0,>=2.0.2->yellowbrick) (9.4.0)
Requirement already satisfied: joblib>=0.11 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from scikit-lear
n>=1.0.0->yellowbrick) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from sci
kit-learn>=1.0.0->yellowbrick) (2.2.0)
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.
```

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', "neither": 'neutral', "class": 'targe
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

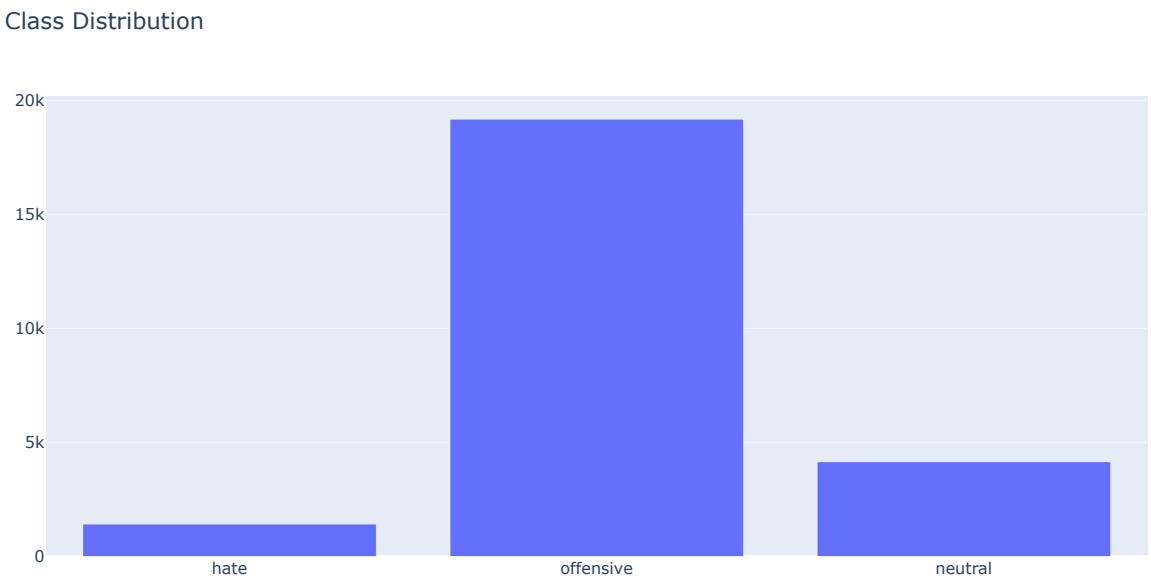
Once we have identified our target variable, we want to visualize the distribution. The figure below indicates that overwhelmingly tweets categorized as offensive totaling over 19,000, while hate tweets comprise a mere 1430.

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.

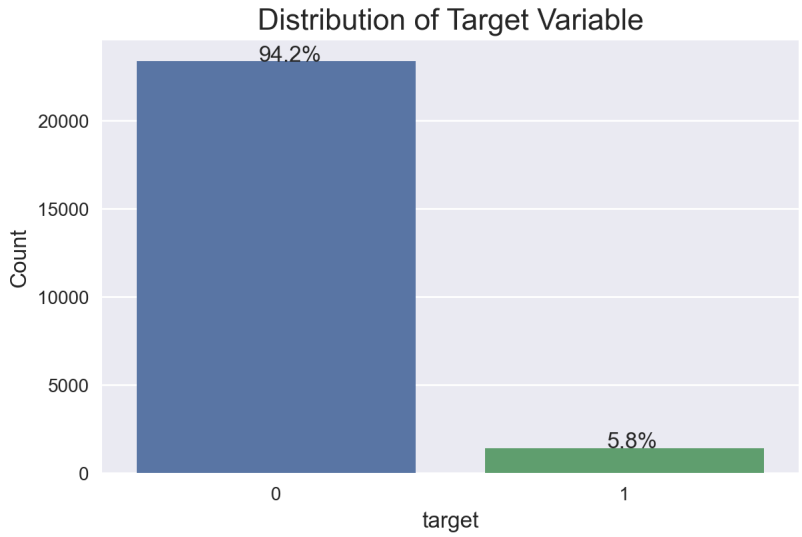
In [8]:
```
import seaborn as sns
# display class distribution
hate = len(df[df['target'] == 0])
off = len(df[df['target'] == 1])
neu = len(df[df['target'] == 2])
dist = [
    graph_objs.Bar(
        x=["hate", "offensive", "neutral"],
        y=[hate, off, neu],
)]
plotly.offline.iplot({"data":dist,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

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;amp; you might not get ya bitch back &amp;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;amp; a bad bitch "
Name: tweet, dtype: object
```

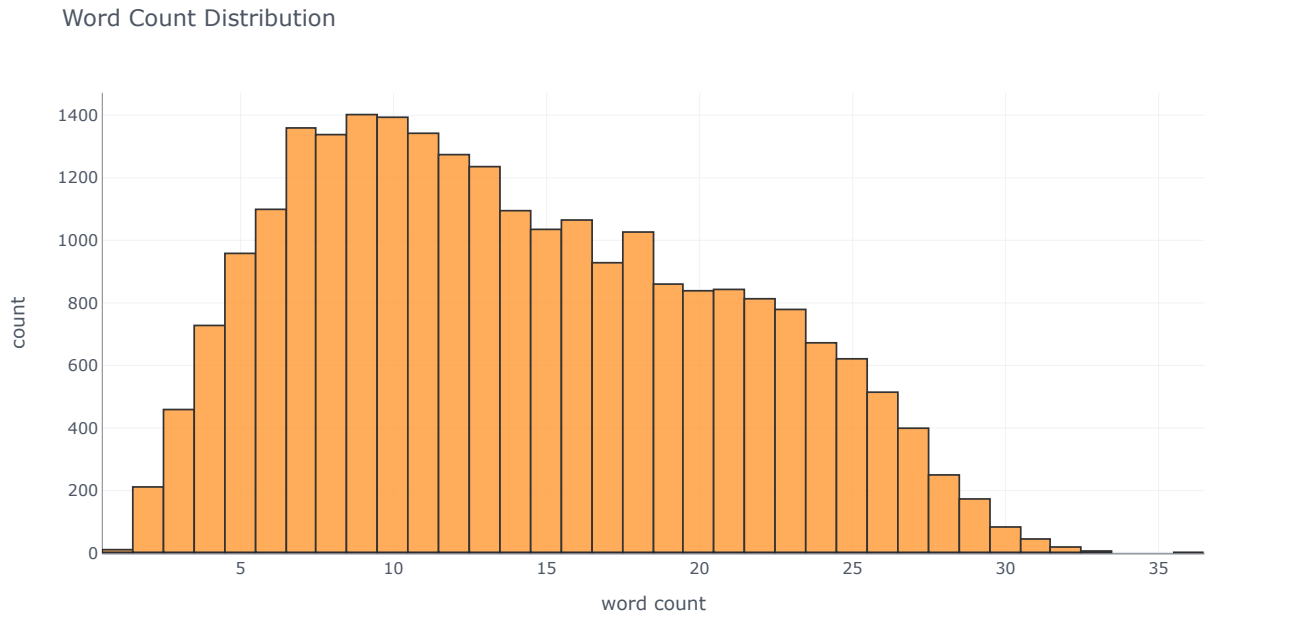
Initial EDA

Word Count Per Tweet

In [12]:
```
# 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 [13]:
```
# 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')
```

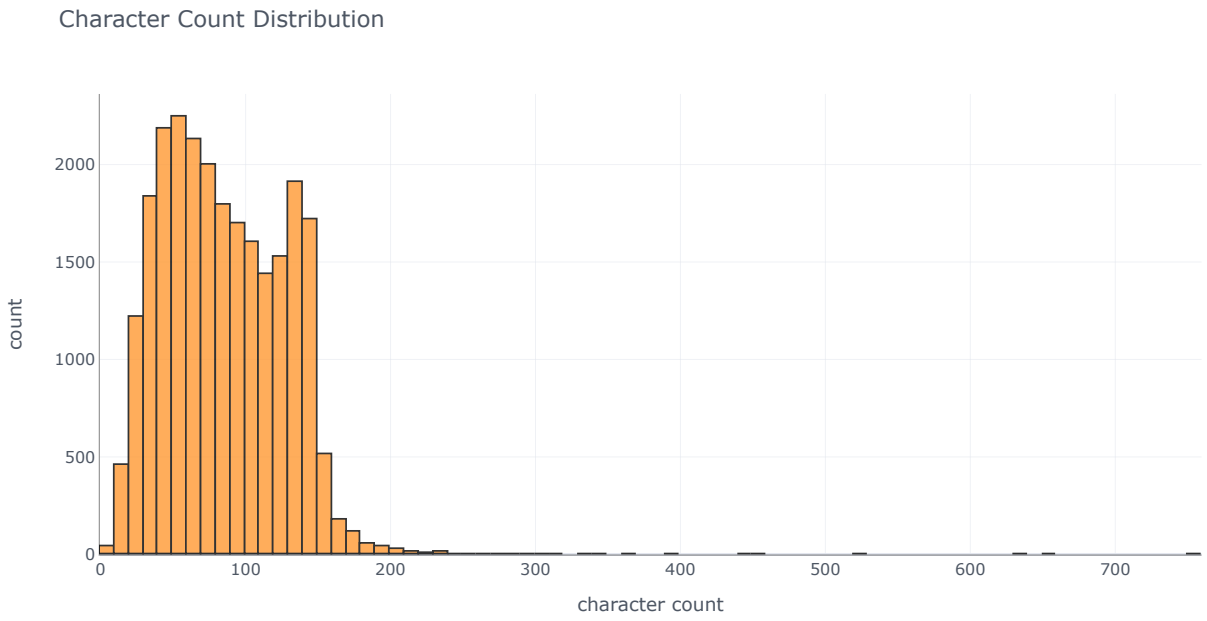


Number of Characters Per Tweet

In [14]:
```
# 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 [15]:
```
# 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')
```

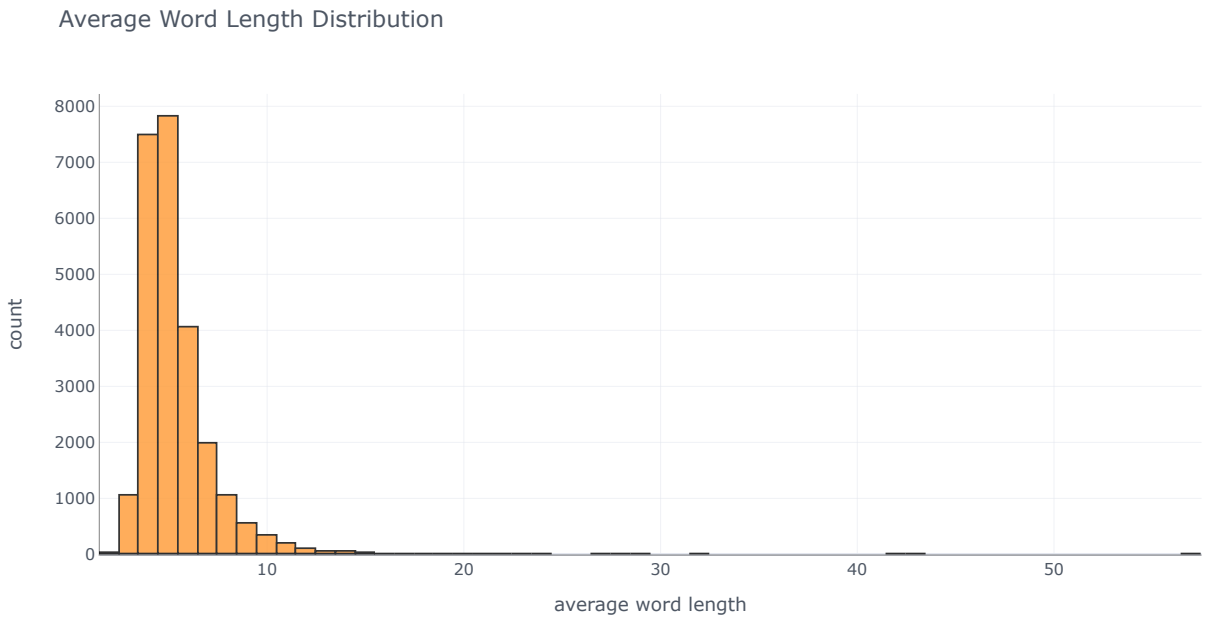


Average Word Length Per Tweet

In [16]:
```
# 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 [17]:
```
# 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')
```

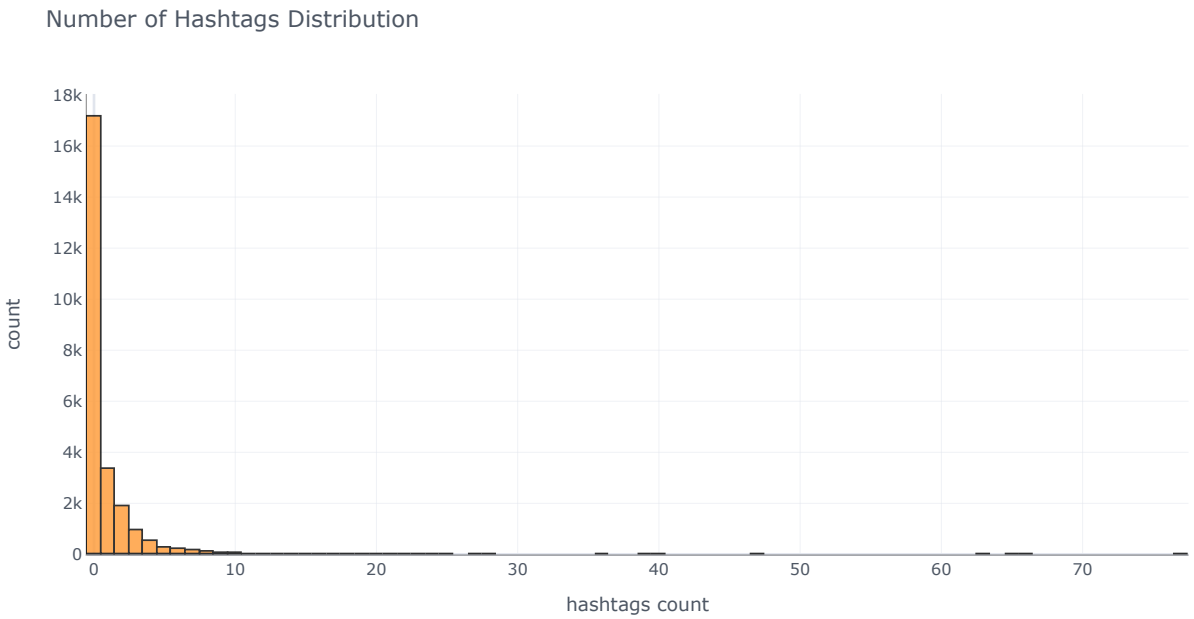


Hashtags Count

In [18]:
```
# 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 [19]:
```
# 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')
```



Export to plot.ly »

Data Preprocessing:

In [20]:
```
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 [78]:
```
pip install typedconfig
```

```
ERROR: Could not find a version that satisfies the requirement typedconfig (from versions: none)
ERROR: No matching distribution found for typedconfig
Note: you may need to restart the kernel to use updated packages.
```

In [79]:

```
auc, average_precision_score, confusion_matrix, roc_auc_score
```

```
ies,geo,id,public_metrics,text&user.fields=description,entities,id,location,name,public_metrics,username".format(batch)
```

```
1498381400435837'}
```

```
l variable
```

```
l variable
```

```
l variable
```

```
l variable
```

```
wed by confusion matrix
del variable
```

```
f)]
oup_percentages)]

False, xticklabels=False, yticklabels=False)
```

```
f)]
oup_percentages)]

False, xticklabels=False, yticklabels=False)
```

```
et_index())
```

In [72]:
```
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))) # remove re-tweet
    df[col] = df[col].apply(lambda x: re.sub(r'(@[A-Za-z0-9-_]+)', '', str(x))) # remove tweeted at

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

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 trailing whitespaces"""
    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 than 3 characters
            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 [73]:
```
#nltk
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import tokenize
from nltk.corpus import subjectivity
from nltk.sentiment import SentimentAnalyzer, SentimentIntensityAnalyzer
from nltk.sentiment.util import *

#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, auc,
                             mean_squared_error, log_loss, precision_recall_curve, classification_report,
                             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, Perceptron, PassiveAggressiveClass
from sklearn.naive_bayes import BernoulliNB, ComplementNB, MultinomialNB
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
from gensim.models import word2vec

from collections import Counter
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pprint import pprint

# Utility
from time import time
import pandas as pd
import sqlite3
import regex as re
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

In [74]:
```
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))) # remove re-tweet
    df[col] = df[col].apply(lambda x: re.sub(r'(@[A-Za-z0-9-_]+)', '', str(x))) # remove tweeted at

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

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 trailing whitespaces"""
    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 than 3 characters
            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 [75]:
```
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 [52]:
```
preprocess_tweets(df, 'tweet')
df.head()
```

Out[52]:

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 [53]:
```
# display first five rows of dataframe
df.head()
```

Out[53]:

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 [54]:
```
# separate dataframe into respective classes
hate = df[df.target == 1]
non_hate = df[df.target == 0]
```

In [30]:
```
# 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=42)
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, random_state=42)
X_nh_tr, X_nh_val, y_nh_tr, y_nh_val = train_test_split(X_nh, y_nh, test_size=0.25, random_state=42)
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, random_state=42)
```

In [31]:
```
# 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[31]:

	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 [32]:
```
# 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[32]:

	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 [33]:
```
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[33]:

	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...

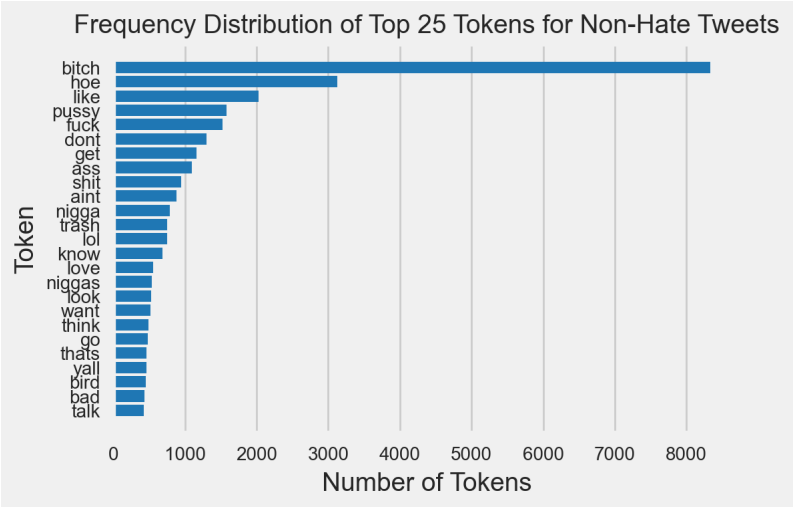
More EDA

Frequency Distributions

In [34]:
```
# split back into minority and majority classes for visualizations
zero = train[train.target == 0]
one = train[train.target == 1]
```

In [35]:
```
# 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 [37]:
```
# 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), color='tab:blue')
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", fontsize=14)
custom_viz.figure.show()
```

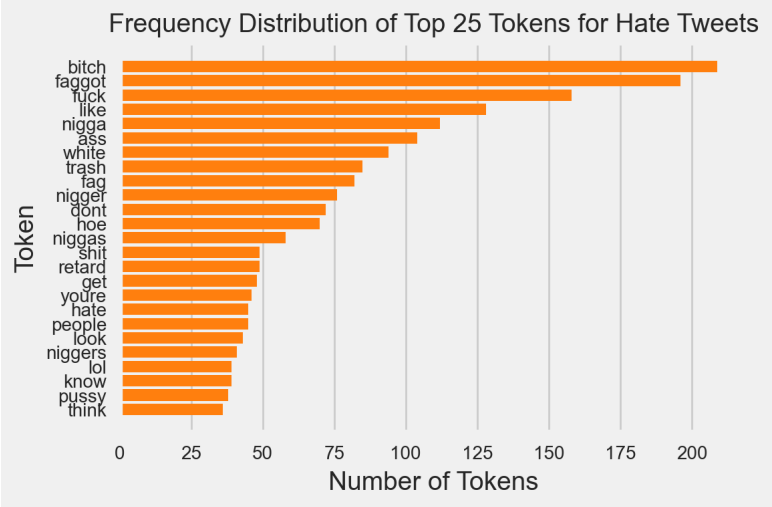


In [38]:
```
custom_viz.figure.savefig("images/freq_dist_zero.png")
```

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 [39]:
```
# 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, 360), color='tab:orange')
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", fontsize=14)
custom_viz_one.figure.show()
```



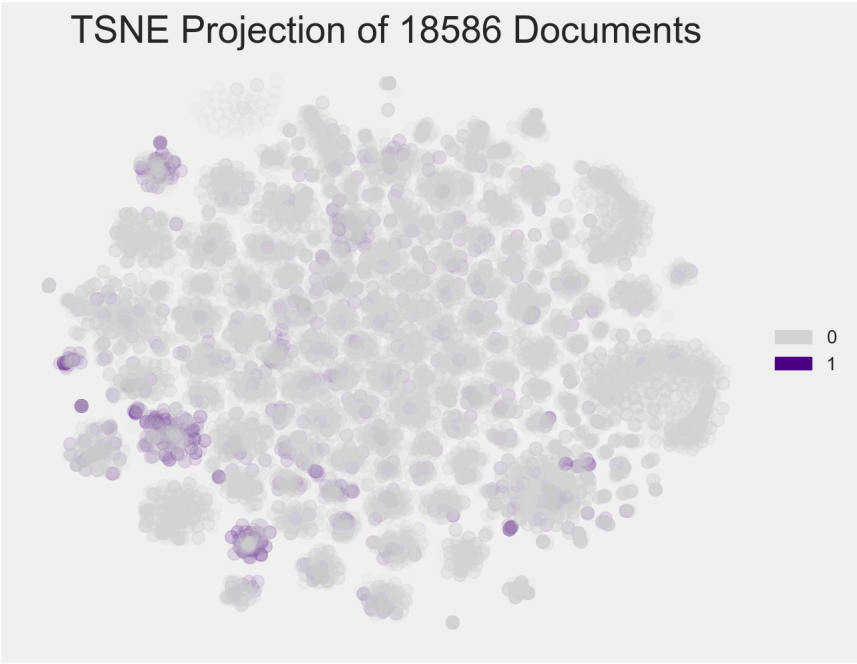
In [40]:
```
custom_viz_one.figure.savefig("images/freq_dist_one.png")
```

t-SNE Corpus Visualization

In [43]:
```
# create TSNE visualization for negative class
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(train.tweet2)
y = train.target
```

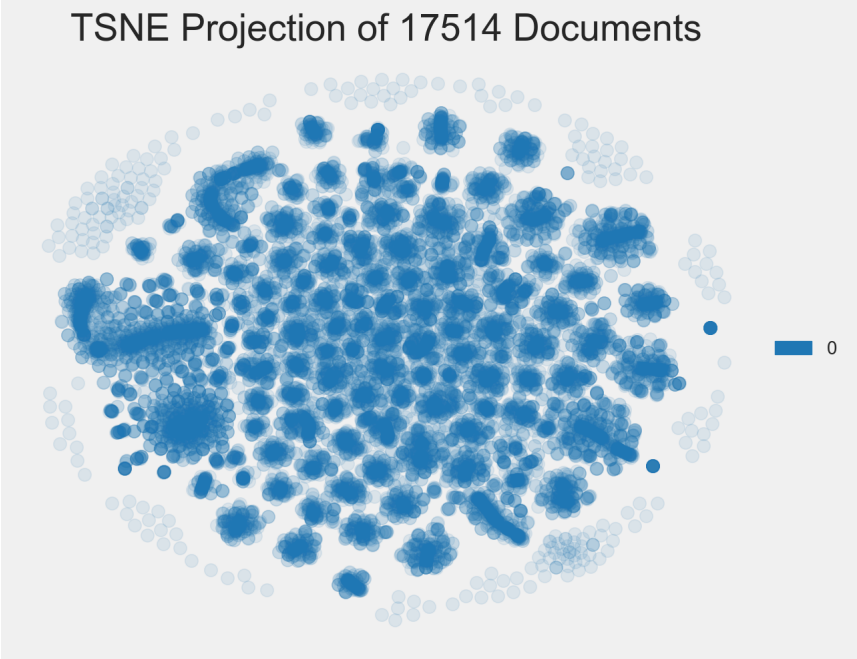
In [57]:
```
visualizer = TSNEVisualizer(alpha=0.1, colors=['lightgray', 'indigo'], decompose='svd', decompose_by=100, random_state=
visualizer.fit(X, y)
visualizer.show(outpath="/Users/Raj/NLP-Project/Automated-Hate-Tweet-Detection/images/tsne.png")
```

Out[57]:
```
<AxesSubplot:title={'center':'TSNE Projection of 18586 Documents'}>
```



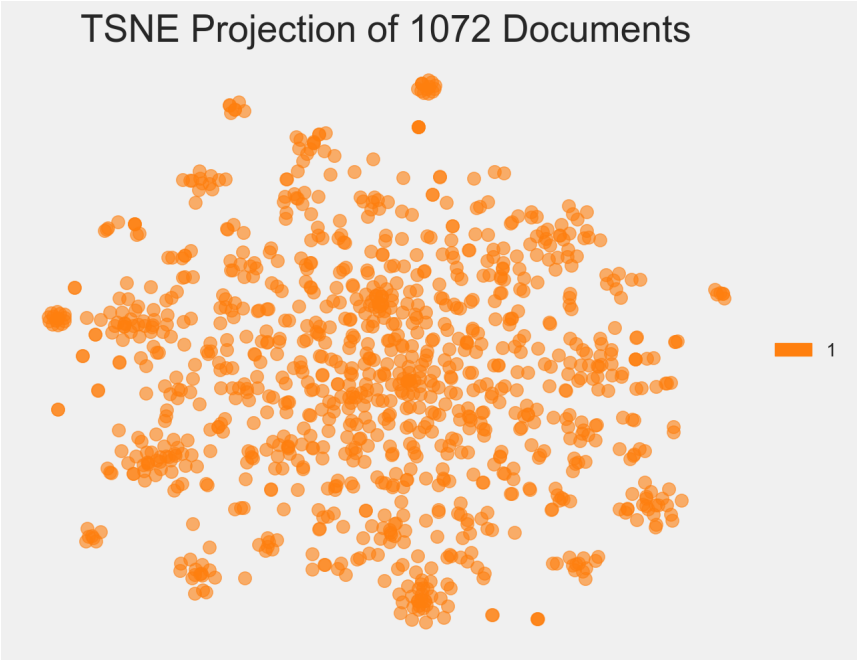
In [58]:
```
# create TSNE visualization for negative class
tfidf = TfidfVectorizer()
X_zero = tfidf.fit_transform(zero.tweet2)
y_zero = zero.target

visualizer = TSNEVisualizer(alpha=0.1, colors=['tab:blue'], decompose='svd', decompose_by=100, random_state=42)
visualizer.fit(X_zero, y_zero)
visualizer.show(outpath="images/tsne_zero.png")
```

Out[58]:
```
<AxesSubplot:title={'center':'TSNE Projection of 17514 Documents'}>
```



In [61]:
```
# 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_by=100, random_state=42)
visualizer.fit(X_one, y_one)
visualizer.show(outpath="images/tsne_one.png")
```

Out[61]:
```
<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 [63]:
```
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 [64]:
```
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()
```

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 [65]:
```
train.tweet2
```

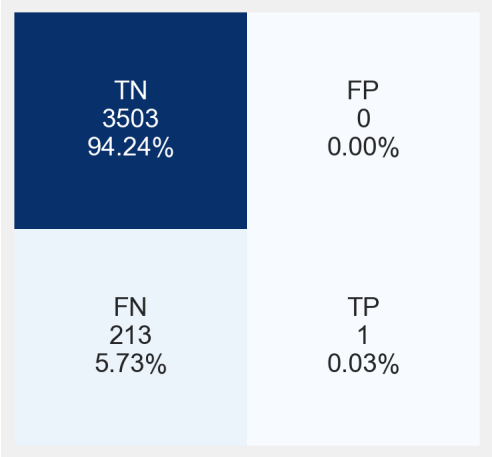
Out[65]:
```
0        'reject', 'constantly', 'house', 'threaten', '...
1        'convince', 'lame', 'nigger', 'liver', 'believ...
2        'peace', 'fag', 'remember', 'best', 'lux', 'su...
3                 'haha', 'ight', 'nig', 'calm', 'yoself'
4        'tits', 'better', 'look', 'face', 'make', 'lik...
                              ...
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 [66]:
```
# 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)
```

Multinomial Naive Bayes

In [80]:
```
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)
```

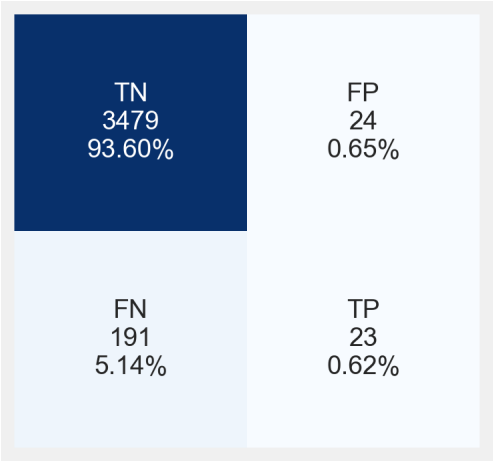
```
Accuracy:  0.9426957223567393
F1 Score:  0.009302325581395347
ROC-AUC:  0.7158910519954859
Recall:  0.004672897196261682
Precision:  1.0
PR-AUC:  0.17537404097663506
```



Random Forest

In [81]:
```
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)
```

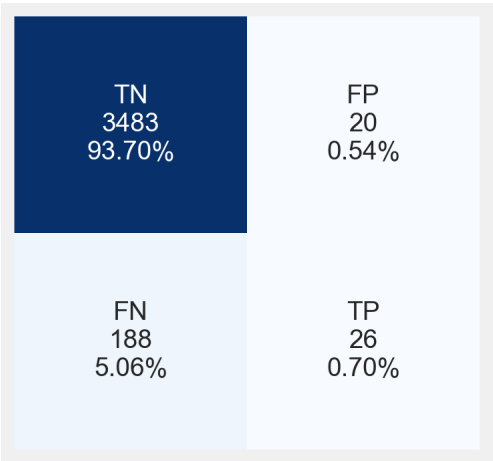
```
Accuracy:  0.9421576540220608
F1 Score:  0.17624521072796934
ROC-AUC:  0.8377812342424785
Recall:  0.10747663551401869
Precision:  0.48936170212765956
PR-AUC:  0.3318635391357664
```



Logistic Regression

In [82]:
```
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
```

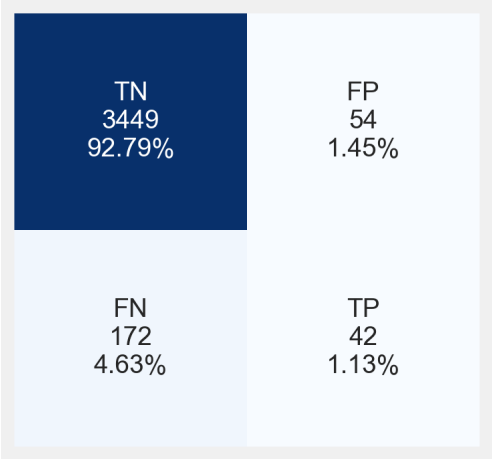


Support Vector Machine

In [83]:
```
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
```

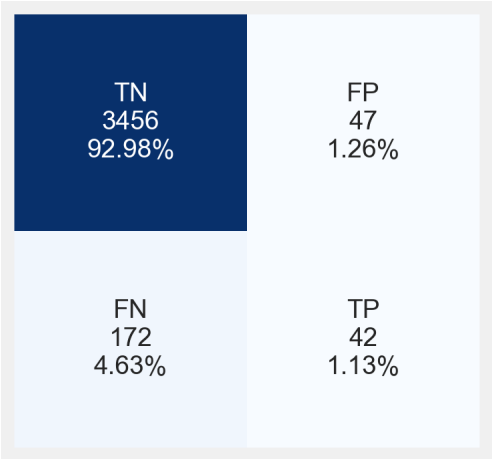
In [84]:
```
get_confusion(y_val, y_pred_svc)
```



Adaboost

In [85]:
```
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:  0.8154632477902786
Recall:  0.19626168224299065
Precision:  0.47191011235955055
PR-AUC:  0.3120265175640685
```



Gradient Boosting

In [86]:
```
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:  0.9418886198547215
F1 Score:  0.16923076923076924
ROC-AUC:  0.8390598445658061
Recall:  0.102803738317757
Precision:  0.4782608695652174
PR-AUC:  0.35071283104264606
```