CODING:

pip install pandas-profiling

#Here I have removed all greek characters, @,numbers.

#Also I have removed 'hmm' word and it's variants

#Creating a dictionary to map words such as luv to love, wud to would.Need more suggestions on this.nyc:nice

#Removed stop words

#Performed lemmatization

import pandas\_profiling

import nltk

import numpy as np

import pandas as pd

import re

import matplotlib.pyplot as plt

import seaborn as sb

from nltk.corpus import stopwords

import warnings

warnings.filterwarnings("ignore")

import unidecode

from wordcloud import WordCloud

from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

from nltk.stem import PorterStemmer

nltk.download('punkt')

from nltk.tokenize import word\_tokenize

import matplotlib.animation as animation

import operator

import plotly.express as px

from collections import Counter

%matplotlib inline

df = pd.read\_csv('train\_E6oV3lV.csv')

df.head()

pandas\_profiling.ProfileReport(df)

df.head()

df.shape

df.drop\_duplicates(inplace = True)

df.shape

df['tweet'].isna().sum()

df['label'].isna().sum()

#Data doesn't contain duplicate values neither does it contain missing values

df.shape

#Code to remove @

df['clean\_tweet'] = df['tweet'].apply(lambda x : ' '.join([tweet for tweet in x.split()if not tweet.startswith("@")]))

df.head()

#Removing numbers

df['clean\_tweet'] = df['clean\_tweet'].apply(lambda x : ' '.join([tweet for tweet in x.split() if not tweet == '\d\*']))

df.head()

pip install unidecode

#Removing all the greek characters using unidecode library

df['clean\_tweet'] = df['clean\_tweet'].apply(lambda x : ' '.join([unidecode.unidecode(word) for word in x.split()]))

df.head(10)

#To check the disappearance of greek symbols

df['clean\_tweet'][7]

#Removing the word 'hmm' and it's variants

df['clean\_tweet'] = df['clean\_tweet'].apply(lambda x : ' '.join([word for word in x.split() if not word == 'h(m)+' ]))

df.head()

#Code for removing slang words

d = {'luv':'love','wud':'would','lyk':'like','wateva':'whatever','ttyl':'talk to you later',

'kul':'cool','fyn':'fine','omg':'oh my god!','fam':'family','bruh':'brother',

'cud':'could','fud':'food'} ## Need a huge dictionary

words = "I luv myself"

words = words.split()

reformed = [d[word] if word in d else word for word in words]

reformed = " ".join(reformed)

reformed

df['clean\_tweet'] = df['clean\_tweet'].apply(lambda x : ' '.join(d[word] if word in d else word for word in x.split()))

df.head(30)

#Finding words with # attached to it

df['#'] = df['clean\_tweet'].apply(lambda x : ' '.join([word for word in x.split() if word.startswith('#')]))

df.head()

frame = df['#']

frame.head()

type(frame)

frame = pd.DataFrame(frame)

frame = frame.rename({'#':'Count(#)'},axis = 'columns')

frame.head()

frame[frame['Count(#)'] == ''] = 'No hashtags'

frame.head()

df.head()

data\_frame = pd.concat([df,frame],axis = 1)

data\_frame.head(10)

data\_frame.drop('#',axis = 1,inplace = True)

data\_frame.head(10)

#Column showing whether the corresponding tweet has a hash tagged word or not

data\_frame = data\_frame.rename({'Count(#)':'Hash words'},axis = 'columns')

data\_frame.head()

nltk.download('stopwords')

#Removing stopwords

data\_frame['clean\_tweet'] = data\_frame['clean\_tweet'].apply(lambda x : ' '.join([word for word in x.split() if not word in set(stopwords.words('english'))]))

data\_frame.head()

df['clean\_tweet\_final'] = df['clean\_tweet'].apply(lambda x : ' '.join([tweet for tweet in x.split()if not tweet.startswith("#")]))

df.head(30)

df.to\_csv('clean\_tweets\_1.csv',index=False,columns=['label','clean\_tweet\_final'])

#Lemmitization

lemmatizer = WordNetLemmatizer()

data\_frame['clean\_tweet'] = data\_frame['clean\_tweet'].apply(lambda x : ' '.join([lemmatizer.lemmatize(word) for word in x.split()]))

#Stemming

ps = PorterStemmer()

adwait = data\_frame

#adwait.head()

data\_frame['clean\_tweet'] = data\_frame['clean\_tweet'].apply(lambda x : ' '.join([ps.stem(word) for word in x.split()]))

data\_frame.head()

#Tokenization

corpus = []

for i in range(0,31962):

tweet = data\_frame['clean\_tweet'][i]

tweet = tweet.lower()

tweet = tweet.split()

tweet = [ps.stem(word) for word in tweet if not word in set(stopwords.words('english'))]

tweet = ' '.join(tweet)

corpus.append(tweet)

#Ensuring all the tweets are tokenized into individual words

len(corpus)

corpus

pip install wordcloud

normal\_words = ' '.join([word for word in data\_frame['clean\_tweet'][data\_frame['label'] == 0]])

wordcloud = WordCloud(width = 800, height = 500, max\_font\_size = 110,max\_words = 100).generate(normal\_words)

print('Normal words')

plt.figure(figsize= (12,8))

plt.imshow(wordcloud, interpolation = 'bilinear',cmap='viridis')

plt.axis('off')

normal\_words = ' '.join([word for word in data\_frame['clean\_tweet'][data\_frame['label'] == 1]])

wordcloud = WordCloud(width = 800, height = 500, max\_font\_size = 110,max\_words = 100).generate(normal\_words)

print('Normal words')

plt.figure(figsize= (12,8))

plt.imshow(wordcloud, interpolation = 'bilinear')

plt.axis('off')

#Collecting positive hashtags

hash\_positive = []

hash\_negative = []

def hashtag\_extract(x):

hashtags = []

# Loop over the words in the tweet

for i in x:

ht = re.findall(r"#(\w+)", i)

hashtags.append(ht)

return hashtags

hash\_positive = hashtag\_extract(data\_frame['clean\_tweet'][data\_frame['label'] == 0])

# extracting hashtags from racist/sexist tweets

hash\_negative = hashtag\_extract(data\_frame['clean\_tweet'][data\_frame['label'] == 1])

# Converting a multidimensional list to a 1-D list

hash\_positive = sum(hash\_positive,[])

hash\_negative = sum(hash\_negative,[])

q = Counter(hash\_positive)

q = dict(q.most\_common())

l\_positive\_count = list(q.values())

l\_positive\_count[0:4]

r = Counter(hash\_negative)

r = dict(r.most\_common())

l\_negative\_count = list(r.values())

l\_negative\_count[0:4]

l\_positive\_values = list(q.keys())

l\_positive\_values[0:4]

l\_negative\_values = list(r.keys())

l\_negative\_values[0:4]

#Creating a dataframe to represent top 20 positive and negative hash words

l1 = pd.DataFrame(l\_positive\_values[0:20],columns = ['Positive\_Words'])

l2 = pd.DataFrame(l\_positive\_count[0:20],columns = ['Positive\_Count'])

l3 = pd.DataFrame(l\_negative\_values[0:20],columns = ['Negative\_Words'])

l4 = pd.DataFrame(l\_negative\_count[0:20],columns = ['Negative\_Count'])

z = pd.concat([l1,l2,l3,l4],axis = 1)

z

#Animated plot for positive words with their frequency

fig = px.bar(z, x="Positive\_Words", y="Positive\_Count",animation\_frame="Positive\_Count",

hover\_name="Positive\_Words")

fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 1200

fig.show()

# Animated plot for negative words with their frequency

fig = px.bar(z, x="Negative\_Words", y="Negative\_Count",animation\_frame="Negative\_Count",

hover\_name="Negative\_Words")

fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 1200

fig.show()

#Normal histogram of positive words

fig = px.bar(z, x="Positive\_Words", y="Positive\_Count",

hover\_name="Positive\_Words",color = 'Positive\_Count')

fig.show()

#Normal histogram of negative words

fig = px.bar(z, x="Negative\_Words", y="Negative\_Count",

hover\_name="Negative\_Words",color= 'Negative\_Count')

fig.show()

data\_frame.head()

#Techniques to convert the tweets into Bag-of-Words, TF-IDF, and Word Embeddings

#Building various classifiers: -

#TF-IDF approach

#pip install sklearn

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer(max\_df=0.90, min\_df=2,stop\_words='english')

# TF-IDF feature matrix

X1 = tfidf\_vectorizer.fit\_transform(corpus).toarray()

Y1 = df.loc[:,'label'].values

from sklearn.model\_selection import train\_test\_split

X1\_train, X1\_test, Y1\_train, Y1\_test = train\_test\_split(data\_frame['clean\_tweet'], data\_frame['label'], test\_size = 0.3, random\_state=0, shuffle = True, stratify=data\_frame['label'])

vectorizer = TfidfVectorizer()

X1\_train\_vect = vectorizer.fit\_transform(X1\_train)

Y1 = df.loc[:,'label'].values

#Random Forest using pipelines

from sklearn.pipeline import Pipeline

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

rf = Pipeline([('tfidf', TfidfVectorizer()), ('rf', RandomForestClassifier())])

rf.fit(X1\_train, Y1\_train)

y\_pred = rf.predict(X1\_test)

print(pd.crosstab(Y1\_test,y\_pred,rownames=['Actual'],colnames=['Predicted']))

print(classification\_report(Y1\_test, y\_pred))

print(rf.predict([""]))

jupyter nbconvert --to html data-cleaning.ipynb

pip install nbconvert