```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string
import nltk
import warnings
%matplotlib inline

warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('Twitter Sentiments.csv')
df.head()
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

```
# datatype info
df.info()
```

```
# removes pattern in the input text
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for word in r:
        input_txt = re.sub(word, "", input_txt)
    return input_txt
```

```
df.head()
```

```
# remove twitter handles (@user)
df['clean_tweet'] = np.vectorize(remove_pattern)(df['tweet'], "@[\w]*")
```

```
df.head()
```

```
# remove special characters, numbers and punctuations
df['clean_tweet'] = df['clean_tweet'].str.replace("[^a-zA-Z#]", " ")
df.head()
```

```
# remove short words
df['clean_tweet'] = df['clean_tweet'].apply(lambda x: " ".join([w for w in x.split() if le
df.head()
```

```
# individual words considered as tokens
tokenized_tweet = df['clean_tweet'].apply(lambda x: x.split())
tokenized_tweet = df['clean_tweet'].apply(lambda x: x.split())
tokenized_tweet.head()


# stem the words
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()

tokenized_tweet = tokenized_tweet.apply(lambda sentence: [stemmer.stem(word) for word in s
tokenized_tweet.head()


# combine words into single sentence
for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = " ".join(tokenized_tweet[i])

df['clean_tweet'] = tokenized_tweet
df.head()


# !pip install wordcloud


from wordcloud import WordCloud,ImageColorGenerator
from PIL import Image
import urllib
import requests


all_words = " ".join([sentence for sentence in df['clean_tweet']])


# combining the image with the dataset
Mask = np.array(Image.open(requests.get('http://clipart-library.com/image_gallery2/Twitter

# We use the ImageColorGenerator library from Wordcloud
# Here we take the color of the image and impose it over our wordcloud
image_colors = ImageColorGenerator(Mask)

# Now we use the WordCloud function from the wordcloud library
wc = WordCloud(background_color='black', height=1500, width=4000,mask=Mask).generate(all_w


# visualize the frequent words



wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(

# plot the graph
plt.figure(figsize=(15,8))
plt.imshow(wc.recolor(color_func=image_colors), interpolation='hamming')
plt.axis('off')
plt.show()


all_words = " ".join([sentence for sentence in df['clean_tweet'][df['label']==0]])
```

```
# combining the image with the dataset
Mask = np.array(Image.open(requests.get('http://clipart-library.com/image_gallery2/Twitter

# We use the ImageColorGenerator library from Wordcloud
# Here we take the color of the image and impose it over our wordcloud
image_colors = ImageColorGenerator(Mask)

# Now we use the WordCloud function from the wordcloud library
wc = WordCloud(background_color='black', height=1500, width=4000,mask=Mask).generate(all_w


# frequent words visualization for +ve


wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(

# plot the graph
plt.figure(figsize=(15,8))
plt.imshow(wc.recolor(color_func=image_colors), interpolation='hamming')
plt.axis('off')
plt.show()


all_words = " ".join([sentence for sentence in df['clean_tweet'][df['label']==1]])



# combining the image with the dataset
Mask = np.array(Image.open(requests.get('http://clipart-library.com/image_gallery2/Twitter

# We use the ImageColorGenerator library from Wordcloud
# Here we take the color of the image and impose it over our wordcloud
image_colors = ImageColorGenerator(Mask)

# Now we use the WordCloud function from the wordcloud library
wc = WordCloud(background_color='black', height=1500, width=4000,mask=Mask).generate(all_w


# frequent words visualization for -ve

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(

# plot the graph
plt.figure(figsize=(15,8))
plt.imshow(wc.recolor(color_func=image_colors), interpolation='hamming')
plt.axis('off')
plt.show()


# extract the hashtag
def hashtag_extract(tweets):
    hashtags = []
    # loop words in the tweet
    for tweet in tweets:
        ht = re.findall(r"#(\w+)", tweet)
        hashtags.append(ht)
    return hashtags


# extract hashtags from non-racist/sexist tweets
ht_positive = hashtag_extract(df['clean_tweet'][df['label']==0])

# extract hashtags from racist/sexist tweets
ht_negative = hashtag_extract(df['clean_tweet'][df['label']==1])


ht_positive[:5]


# unnest list
ht_positive = sum(ht_positive, [])
ht_negative = sum(ht_negative, [])


ht_positive[:5]


freq = nltk.FreqDist(ht_positive)
d = pd.DataFrame({'Hashtag': list(freq.keys()),
                 'Count': list(freq.values())})
d.head()


# select top 10 hashtags
d = d.nlargest(columns='Count', n=10)
plt.figure(figsize=(15,9))
sns.barplot(data=d, x='Hashtag', y='Count')
plt.show()


freq = nltk.FreqDist(ht_negative)
d = pd.DataFrame({'Hashtag': list(freq.keys()),
                 'Count': list(freq.values())})
d.head()


# select top 10 hashtags
d = d.nlargest(columns='Count', n=10)
plt.figure(figsize=(15,9))
sns.barplot(data=d, x='Hashtag', y='Count')
plt.show()



# feature extraction
from sklearn.feature_extraction.text import CountVectorizer
bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='eng
bow = bow_vectorizer.fit_transform(df['clean_tweet'])


# bow[0].toarray()


from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(bow, df['label'], random_state=42, tes


from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score



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


# testing
pred = model.predict(x_test)
f1_score(y_test, pred)


accuracy_score(y_test,pred)


# use probability to get output
pred_prob = model.predict_proba(x_test)
pred = pred_prob[:, 1] >= 0.3
pred = pred.astype(np.int)

f1_score(y_test, pred)


accuracy_score(y_test,pred)


pred_prob[0][1] >= 0.3
```