

```
# installing kaggle library
```

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in  
/usr/local/lib/python3.11/dist-packages (1.7.4.5)  
Requirement already satisfied: bleach in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)  
Requirement already satisfied: certifi>=14.05.14 in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (2025.4.26)  
Requirement already satisfied: charset-normalizer in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (3.4.2)  
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-  
packages (from kaggle) (3.10)  
Requirement already satisfied: protobuf in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (5.29.5)  
Requirement already satisfied: python-dateutil>=2.5.3 in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (2.9.0.post0)  
Requirement already satisfied: python-slugify in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)  
Requirement already satisfied: requests in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)  
Requirement already satisfied: setuptools>=21.0.0 in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (75.2.0)  
Requirement already satisfied: six>=1.10 in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)  
Requirement already satisfied: text-unidecode in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (1.3)  
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-  
packages (from kaggle) (4.67.1)  
Requirement already satisfied: urllib3>=1.15.1 in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (2.4.0)  
Requirement already satisfied: webencodings in  
/usr/local/lib/python3.11/dist-packages (from kaggle) (0.5.1)
```

UPLOAD Kaggle.json file

```
# configuring the path of Kaggle.json file
```

```
!mkdir -p ~/.kaggle  
!cp kaggle.json ~/.kaggle/  
!chmod 600 ~/.kaggle/kaggle.json
```

Importing Twitter Sentiment Dataset

```
# API to fetch the dataset from kaggle
```

```
!kaggle datasets download -d kazanova/sentiment140
```

Dataset URL: <https://www.kaggle.com/datasets/kazanova/sentiment140>

License(s): other

Downloading sentiment140.zip to /content

```
0% 0.00/80.9M [00:00<?, ?B/s]
100% 80.9M/80.9M [00:00<00:00, 1.11GB/s]
```

```
# Extracting the compressed dataset
from zipfile import ZipFile
dataset = '/content/sentiment140.zip'

with ZipFile(dataset, 'r') as zip:
    zip.extractall()
    print('The dataset is extracted')
```

The dataset is extracted

Importing the Dependencies

```
import numpy as np
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

import nltk
nltk.download('stopwords')
```

[nltk\_data] Downloading package stopwords to /root/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

True

```
# Printing the stopwords in English
print(stopwords.words('english'))
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all',
'am', 'an', 'and', 'any', 'are', 'aren', "aren't", 'as', 'at', 'be',
'because', 'been', 'before', 'being', 'below', 'between', 'both',
'but', 'by', 'can', 'couldn', "couldn't", 'd', 'did', 'didn',
"didn't", 'do', 'does', 'doesn', "doesn't", 'doing', 'don', "don't",
'down', 'during', 'each', 'few', 'for', 'from', 'further', 'had',
'hadn', "hadn't", 'has', 'hasn', "hasn't", 'have', 'haven', "haven't",
'having', 'he', "he'd", "he'll", 'her', 'here', 'hers', 'herself',
'he's', 'him', 'himself', 'his', 'how', 'i', "i'd", 'if', "i'll",
'i'm', 'in', 'into', 'is', 'isn', "isn't", 'it', "it'd", "it'll",
"it's", 'its', 'itself', "i've", 'just', 'll', 'm', 'ma', 'me',
'mightn', "mightn't", 'more', 'most', 'mustn', "mustn't", 'my',
'myself', 'needn', "needn't", 'no', 'nor', 'not', 'now', 'o', 'of',
'off', 'on', 'once', 'only', 'or', 'other', 'our', 'ours',
'ourselves', 'out', 'over', 'own', 're', 's', 'same', 'shan',
```

```
"shan't", 'she', "she'd", "she'll", "she's", 'should', 'shouldn',  
"shouldn't", "should've", 'so', 'some', 'such', 't', 'than', 'that',  
"that'll", 'the', 'their', 'theirs', 'them', 'themselves', 'then',  
'there', 'these', 'they', "they'd", "they'll", "they're", "they've",  
'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 've',  
'very', 'was', 'wasn', "wasn't", 'we', "we'd", "we'll", "we're",  
'were', 'weren', "weren't", "we've", 'what', 'when', 'where', 'which',  
'while', 'who', 'whom', 'why', 'will', 'with', 'won', "won't",  
'wouldn', "wouldn't", 'y', 'you', "you'd", "you'll", 'your', "you're",  
'yours', 'yourself', 'yourselves', "you've"]
```

## Data Processing

```
# loading the data from csv file to pandas dataframe  
twitter_data =  
pd.read_csv('/content/training.1600000.processed.noemoticon.csv',  
encoding='ISO-8859-1')  
  
# Checking the number of rows and columns  
twitter_data.shape  
  
(1599999, 6)  
  
# printing first 5 rows of the dataframe  
twitter_data.head()  
  
{"type": "dataframe", "variable_name": "twitter_data"}  
  
# Naming the columns and reading the dataset again  
column_names = ['target', 'id', 'date', 'flag', 'user', 'text']  
twitter_data =  
pd.read_csv('/content/training.1600000.processed.noemoticon.csv',  
names=column_names, encoding='ISO-8859-1')  
  
# Checking the number of rows and columns  
twitter_data.shape  
  
(1600000, 6)  
  
# printing first 5 rows of the dataframe  
twitter_data.head()  
  
{"type": "dataframe", "variable_name": "twitter_data"}  
  
# counting the number of missing values in the dataset  
twitter_data.isnull().sum()  
  
target    0  
id        0  
date      0  
flag      0
```

```

user      0
text      0
dtype: int64

# checking the distribution of target column
twitter_data['target'].value_counts()

target
0      800000
4      800000
Name: count, dtype: int64

```

Convert the target "4" to "1"

```

twitter_data.replace({'target':{4:1}}, inplace=True)

# checking the distribution of target column
twitter_data['target'].value_counts()

target
0      800000
1      800000
Name: count, dtype: int64

```

"0" - Negative Tweet "1" - Positive Tweet

## Stemming

Stemming is the process of reducing a word to its Root word

example: actor, actress, acting = act

```

port_stem = PorterStemmer()

def stemming(content):
    stemmed_content = re.sub('[^a-zA-Z]', ' ', content)
    stemmed_content = stemmed_content.lower()
    stemmed_content = stemmed_content.split()
    stemmed_content = [port_stem.stem(word) for word in stemmed_content
    if not word in stopwords.words('english')]
    stemmed_content = ' '.join(stemmed_content)
    return stemmed_content

twitter_data['stemmed_content'] = twitter_data['text'].apply(stemming)
# takes 50 minutes to complete execution

twitter_data.head()

{"type": "dataframe", "variable_name": "twitter_data"}
print(twitter_data['stemmed_content'])

```

```

0      switchfoot http twitpic com zl awww bummer sho...
1      upset updat facebook text might cri result sch...
2      kenichan dive mani time ball manag save rest g...
3              whole bodi feel itchi like fire
4              nationwideclass behav mad see
...
1599995              woke school best feel ever
1599996      thewdb com cool hear old walt interview http b...
1599997              readi mojo makeov ask detail
1599998      happi th birthday boo alll time tupac amaru sh...
1599999      happi charitytuesday thenspcc sparkschar speak...
Name: stemmed_content, Length: 1600000, dtype: object

```

```
print/twitter_data['target'])
```

```

0      0
1      0
2      0
3      0
4      0
...
1599995      1
1599996      1
1599997      1
1599998      1
1599999      1
Name: target, Length: 1600000, dtype: int64

```

```
# separating the data and label
```

```
X = twitter_data['stemmed_content'].values
```

```
Y = twitter_data['target'].values
```

```
print(X)
```

```

['switchfoot http twitpic com zl awww bummer shoulda got david carr
third day'
'upset updat facebook text might cri result school today also blah'
'kenichan dive mani time ball manag save rest go bound' ...
'readi mojo makeov ask detail'
'happi th birthday boo alll time tupac amaru shakur'
'happi charitytuesday thenspcc sparkschar speakinguph h']

```

```
print(Y)
```

```
[0 0 0 ... 1 1 1]
```

Splitting the data to training data and test data

```

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, stratify=Y, random_state=2)

```

```

print(X.shape, X_train.shape, X_test.shape)

(1600000,) (1280000,) (320000,)

print(X_train)

['watch saw iv drink lil wine' 'hatermagazin'
 'even though favourit drink think vodka coke wipe mind time think im
 gonna find new drink'
 ... 'eager monday afternoon'
 'hope everyon mother great day wait hear guy store tomorrow'
 'love wake folger bad voic deeper']

print(X_test)

['mmangen fine much time chat twitter hubbi back summer amp tend domin
 free time'
 'ah may show w ruth kim amp geoffrey sanhueza'
 'ishatara mayb bay area thang dammit' ...
 'destini nevertheless hooray member wonder safe trip' 'feel well'
 'supersandro thank']

# Converting the textual data to numerical data
vectorizer = TfidfVectorizer()

X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

print(X_train)

<Compressed Sparse Row sparse matrix of dtype 'float64'
 with 9453092 stored elements and shape (1280000, 461488)>
  Coords  Values
(0, 436713) 0.27259876264838384
(0, 354543) 0.3588091611460021
(0, 185193) 0.5277679060576009
(0, 109306) 0.3753708587402299
(0, 235045) 0.41996827700291095
(0, 443066) 0.4484755317023172
(1, 160636) 1.0
(2, 109306) 0.4591176413728317
(2, 124484) 0.1892155960801415
(2, 407301) 0.18709338684973031
(2, 129411) 0.29074192727957143
(2, 406399) 0.32105459490875526
(2, 433560) 0.3296595898028565
(2, 77929) 0.31284080750346344
(2, 443430) 0.3348599670252845
(2, 266729) 0.24123230668976975
(2, 409143) 0.15169282335109835
(2, 178061) 0.1619010109445149

```

```

(2, 150715)    0.18803850583207948
(2, 132311)    0.2028971570399794
(2, 288470)    0.16786949597862733
(3, 406399)    0.29029991238662284
(3, 158711)    0.4456939372299574
(3, 151770)    0.278559647704793
(3, 56476)     0.5200465453608686
:              :
(1279996, 318303) 0.21254698865277744
(1279996, 434014) 0.27189450523324465
(1279996, 390130) 0.2206474219107611
(1279996, 373144) 0.35212500999832036
(1279996, 238077) 0.5249170684084672
(1279996, 238078) 0.5606696159563151
(1279997, 5685)   0.48650358607431304
(1279997, 273084) 0.4353549002982409
(1279997, 112591) 0.7574829183045267
(1279998, 412553) 0.2816582375021589
(1279998, 93795)  0.21717768937055476
(1279998, 169461) 0.2659980990397061
(1279998, 124765) 0.32241752985927996
(1279998, 435463) 0.2851807874350361
(1279998, 153281) 0.28378968751027456
(1279998, 156297) 0.3137096161546449
(1279998, 162047) 0.34691726958159064
(1279998, 275288) 0.38703346602729577
(1279998, 385313) 0.4103285865588191
(1279999, 242268) 0.19572649660865402
(1279999, 31410)  0.248792678366695
(1279999, 435572) 0.31691096877786484
(1279999, 433612) 0.3607341026233411
(1279999, 135384) 0.6130934129868719
(1279999, 96224)  0.5416162421321443

```

```
print(X_test)
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
  with 2289192 stored elements and shape (320000, 461488)>
```

```

Coords  Values
(0, 15110)    0.1719352837797837
(0, 31168)    0.1624772418052177
(0, 67828)    0.26800375270827315
(0, 106069)   0.36555450010904555
(0, 132364)   0.255254889555786
(0, 138164)   0.23688292264071406
(0, 171378)   0.2805816206356074
(0, 271016)   0.45356623916588285
(0, 279082)   0.17825180109103442
(0, 388348)   0.2198507607206174
(0, 398906)   0.34910438732642673

```

(0, 409143)	0.3143047059807971
(0, 420984)	0.17915624523539805
(1, 6463)	0.30733520460524466
(1, 15110)	0.211037449588008
(1, 145393)	0.575262969264869
(1, 217562)	0.40288153995289894
(1, 256777)	0.28751585696559306
(1, 348135)	0.4739279595416274
(1, 366203)	0.24595562404108307
(2, 22532)	0.3532582957477176
(2, 34401)	0.37916255084357414
(2, 89448)	0.36340369428387626
(2, 183312)	0.5892069252021465
(2, 256834)	0.2564939661498776
:	:
(319994, 443794)	0.2782185641032538
(319995, 107868)	0.33399349737546963
(319995, 109379)	0.3020896484890833
(319995, 155493)	0.2770682832971669
(319995, 213324)	0.2683969144317079
(319995, 232891)	0.2574127854589077
(319995, 296662)	0.3992485679384015
(319995, 315813)	0.2848229914563413
(319995, 324496)	0.36131679336475747
(319995, 416257)	0.23816465111736282
(319995, 420984)	0.22631428606830148
(319995, 444934)	0.32110928175992615
(319996, 397506)	0.9101400928717545
(319996, 438709)	0.4143006291901984
(319997, 98792)	0.4463892055808332
(319997, 169411)	0.403381646999604
(319997, 261286)	0.37323893626855326
(319997, 288421)	0.48498483387153407
(319997, 349904)	0.32484594100566083
(319997, 416695)	0.29458327588067873
(319997, 444770)	0.2668297951055569
(319998, 130192)	0.6941927210956169
(319998, 438748)	0.719789181620468
(319999, 389755)	0.9577980203954275
(319999, 400636)	0.2874420848216212

## TRAINING THE MACHINE LEARNING MODEL

### LOGISTIC REGRESSION

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, Y_train)
LogisticRegression(max_iter=1000)
```



## MODEL EVALUATION

### ACCURACY SCORE

```
# accuracy score on the training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)

print('Accuracy score on the training data:', training_data_accuracy)

Accuracy score on the training data: 0.79871953125

# accuracy score on the test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)

print('Accuracy score on the test data:', test_data_accuracy)

Accuracy score on the test data: 0.77668125
```

Model accuracy = 77.8%

Saving the trained model

```
import pickle

filename = 'trained_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

Using the saved model for future prediction

```
# loading the saved model
loaded_model = pickle.load(open('/content/trained_model.sav', 'rb'))

X_new = X_test[200]
print(Y_test[200])

prediction = loaded_model.predict(X_new)
print(prediction)

if (prediction[0]==0):
    print('Negative Tweet')
else:
    print('Positive Tweet')

1
[1]
Positive Tweet

X_new = X_test[3]
print(Y_test[3])
```

```
prediction = loaded_model.predict(X_new)
print(prediction)

if (prediction[0]==0):
    print('Negative Tweet')
else:
    print('Positive Tweet')

0
[0]
Negative Tweet
```

**Sentiment Distribution (Before & After Cleaning)** Shows how many tweets are positive and negative.

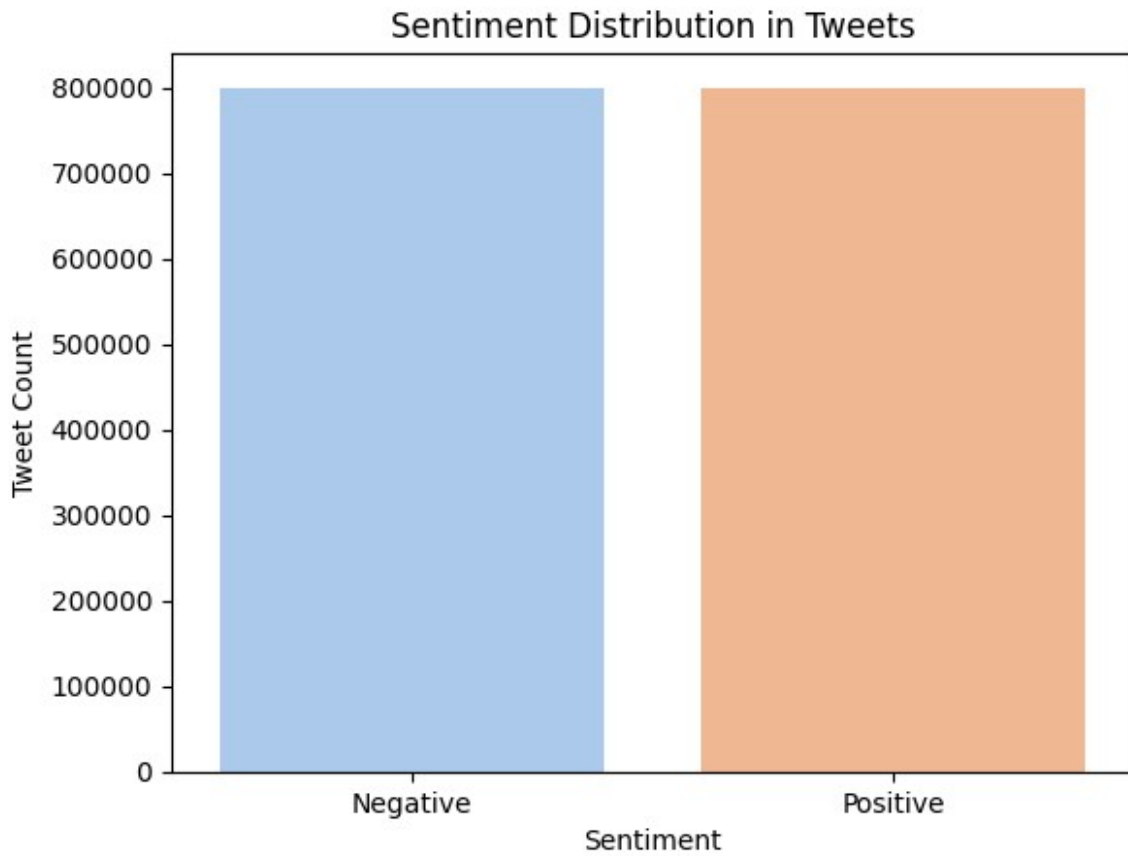
---

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot sentiment distribution
sns.countplot(x='target', data=twitter_data, palette='pastel')
plt.title('Sentiment Distribution in Tweets')
plt.xticks([0, 1], ['Negative', 'Positive'])
plt.xlabel('Sentiment')
plt.ylabel('Tweet Count')
plt.show()

<ipython-input-53-3585304609>:5: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

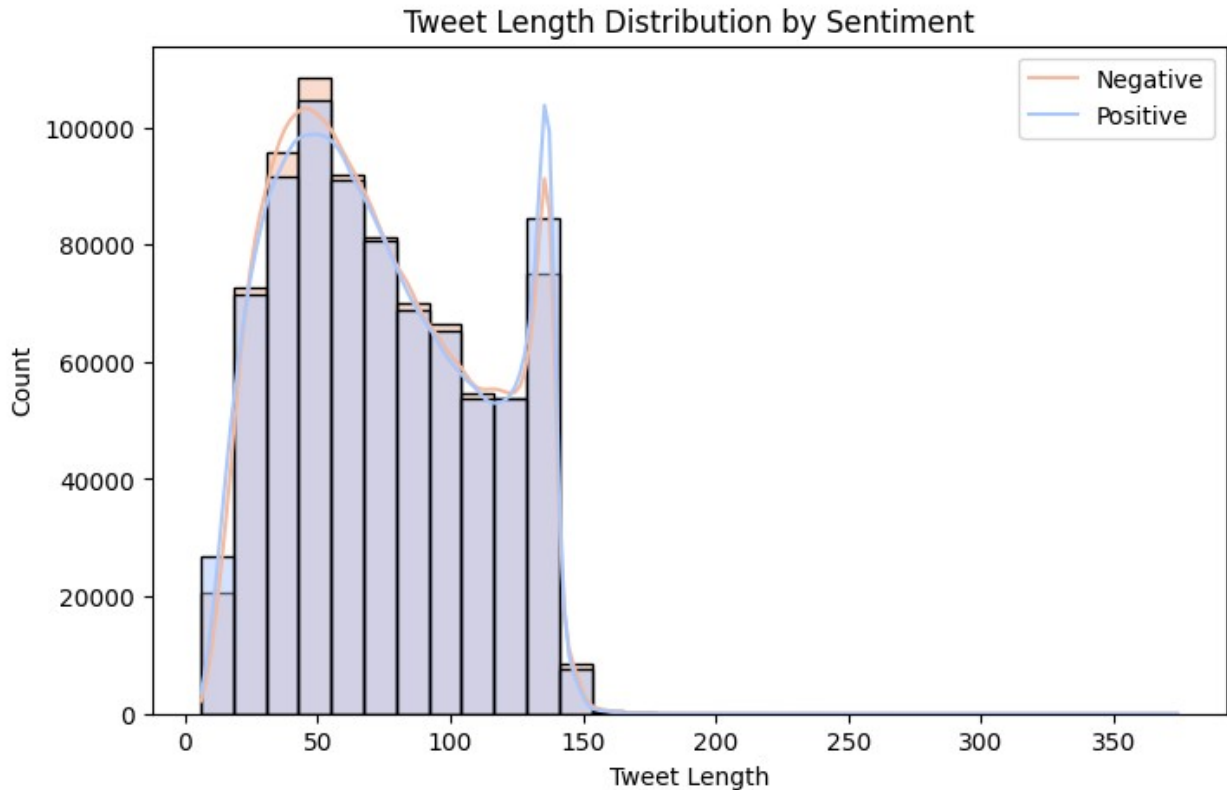
sns.countplot(x='target', data=twitter_data, palette='pastel')
```



**Tweet Length Distribution** Compare tweet lengths to check if longer tweets lean toward a sentiment.

```
twitter_data['text_length'] = twitter_data['text'].apply(len)

plt.figure(figsize=(8, 5))
sns.histplot(data=twitter_data, x='text_length', hue='target',
             bins=30, palette='coolwarm', kde=True)
plt.title('Tweet Length Distribution by Sentiment')
plt.xlabel('Tweet Length')
plt.ylabel('Count')
plt.legend(['Negative', 'Positive'])
plt.show()
```



WordCloud for Positive and Negative Tweets Shows frequent words in each sentiment.

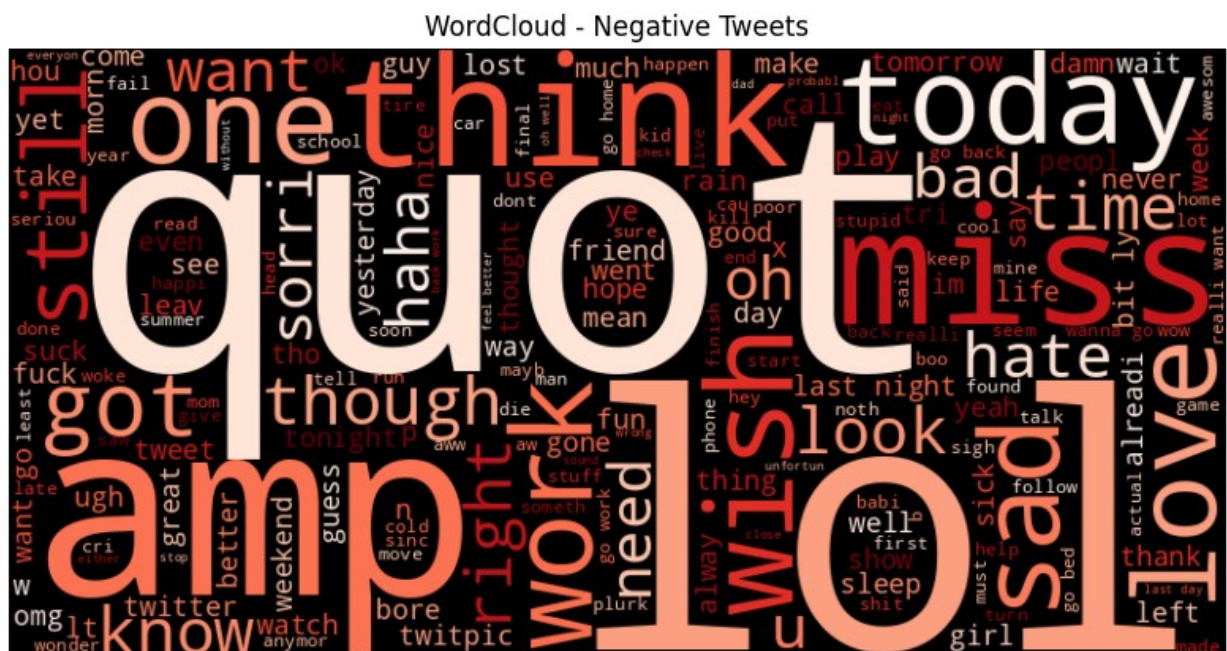
```
from wordcloud import WordCloud

# Separate texts by sentiment
positive_text = " ".join([twitter_data['stemmed_content']
                           for twitter_data in twitter_data_list
                           if twitter_data['target'] == 1])
negative_text = " ".join([twitter_data['stemmed_content']
                           for twitter_data in twitter_data_list
                           if twitter_data['target'] == 0])

# WordCloud for positive tweets
plt.figure(figsize=(10,5))
wordcloud_pos = WordCloud(width=800, height=400,
                           background_color='white').generate(positive_text)
plt.imshow(wordcloud_pos, interpolation='bilinear')
plt.axis('off')
plt.title("WordCloud - Positive Tweets")
plt.show()

# WordCloud for negative tweets
plt.figure(figsize=(10,5))
wordcloud_neg = WordCloud(width=800, height=400,
                           background_color='black', colormap='Reds').generate(negative_text)
plt.imshow(wordcloud_neg, interpolation='bilinear')
plt.axis('off')
```

```
plt.title("WordCloud - Negative Tweets")
plt.show()
```



**TF-IDF Feature Importance (Top Words)** See top words contributing to classification.

```
# Get feature names and weights
feature_names = vectorizer.get_feature_names_out()
coefs = model.coef [0]
```

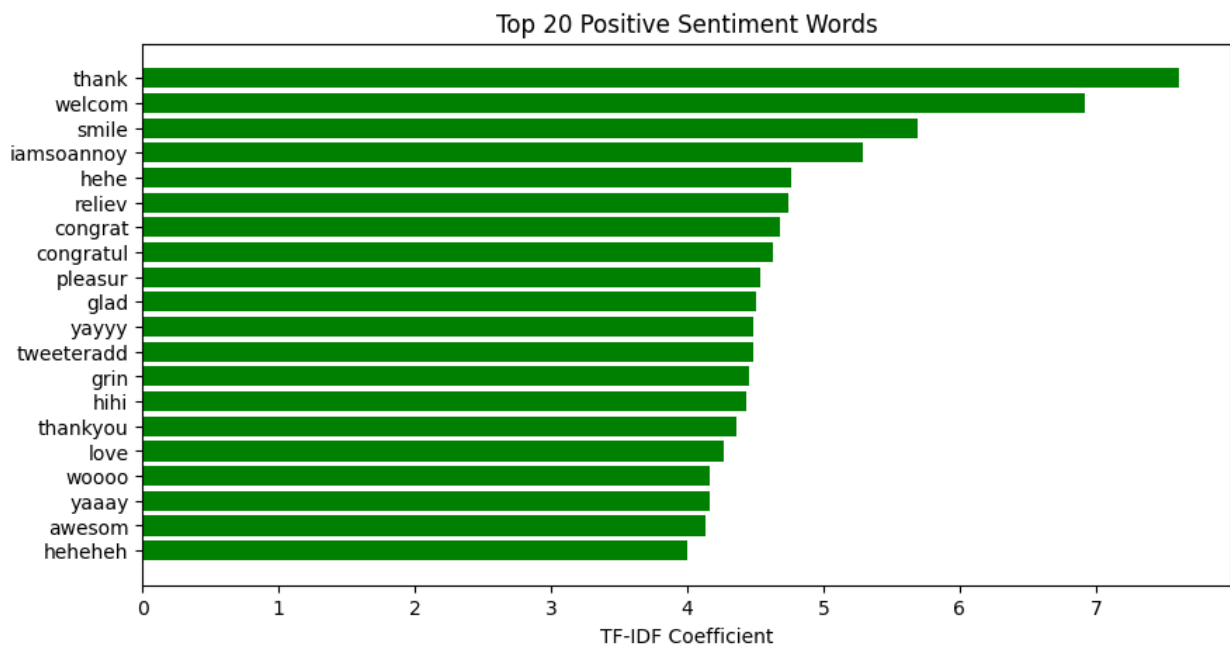
```

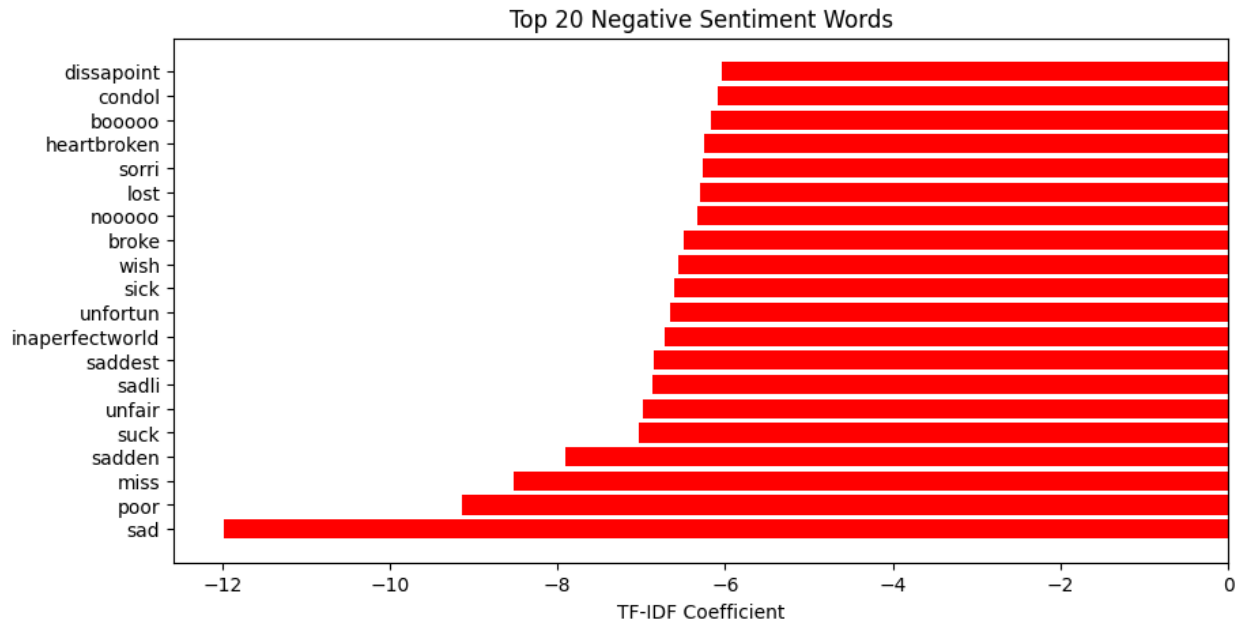
# Top positive and negative coefficients
top_positive_indices = np.argsort(coefs)[-20:]
top_negative_indices = np.argsort(coefs)[:20]

plt.figure(figsize=(10, 5))
plt.barh([feature_names[i] for i in top_positive_indices],
         coefs[top_positive_indices], color='green')
plt.title('Top 20 Positive Sentiment Words')
plt.xlabel('TF-IDF Coefficient')
plt.show()

plt.figure(figsize=(10, 5))
plt.barh([feature_names[i] for i in top_negative_indices],
         coefs[top_negative_indices], color='red')
plt.title('Top 20 Negative Sentiment Words')
plt.xlabel('TF-IDF Coefficient')
plt.show()

```



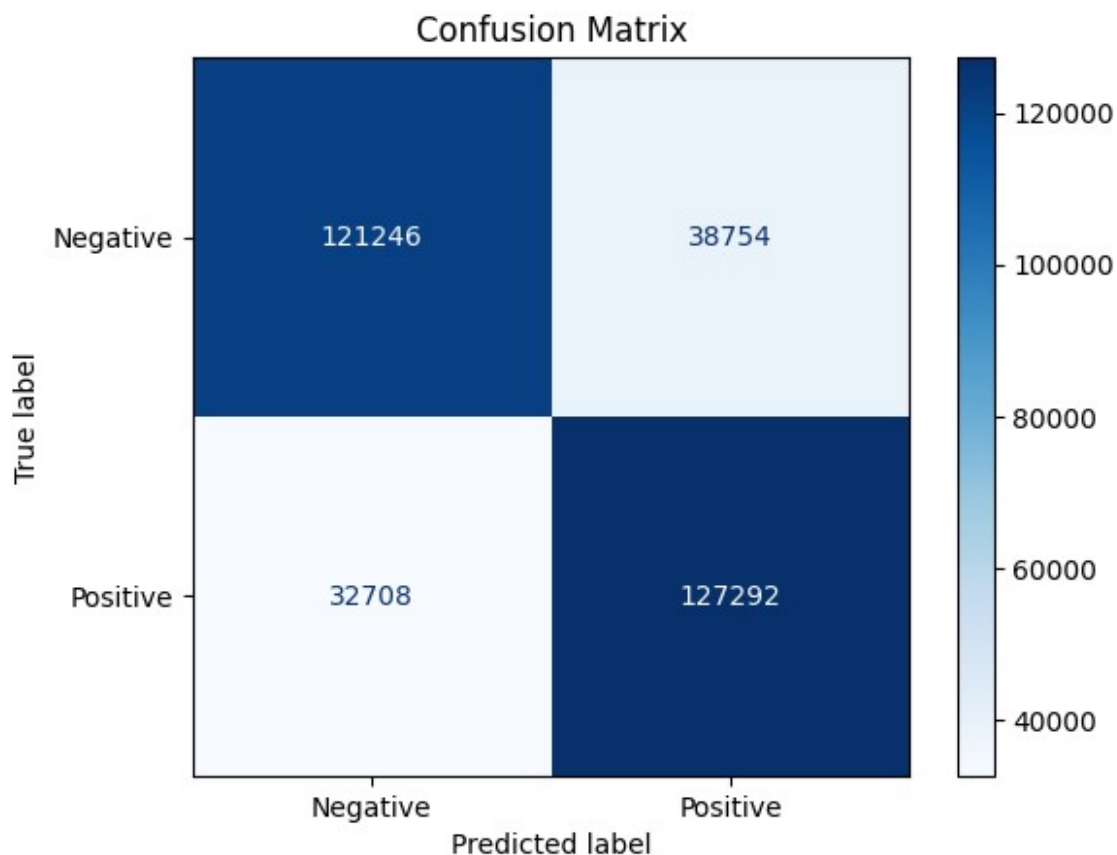


**Model Performance - Confusion Matrix** Visualize how many tweets were correctly/incorrectly classified.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate confusion matrix
cm = confusion_matrix(Y_test, X_test_prediction)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=['Negative', 'Positive'])

disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report

# Predict labels for the test data
y_pred = model.predict(X_test)

# Generate classification report
report = classification_report(Y_test, y_pred,
                               target_names=['Negative', 'Positive'])

# Print the report
print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

   Negative       0.79       0.76       0.77     160000
   Positive       0.77       0.80       0.78     160000

 accuracy                   0.78     320000
  macro avg       0.78       0.78       0.78     320000
 weighted avg     0.78       0.78       0.78     320000
```