1. Overview

This notebook focuses on preprocessing tweet data for sentiment analysis and emotion classification. The primary tasks include text cleaning, tokenization, and encoding categorical features. The dataset undergoes transformations using TF-IDF vectorization and OneHotEncoding to prepare it for machine learning models like Logistic Regression, Random Forest, and XGBoost.

The notebook also handles missing values, standardizes text by removing stopwords, and lemmatizes the tokens. Finally, the processed datasets are saved for future modeling and analysis.

2. Data Preparation

2.1 Importing Necessary Libraries

In [1]:
```
from collections import Counter
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import LabelEncoder
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, FunctionTra
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional, Dense,
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import OneHotEncoder
from nltk.corpus import wordnet
import category_encoders as ce
import pickle
import pandas as pd
import numpy as np
import re
import seaborn as sns
import nltk
import matplotlib.pyplot as plt
import warnings

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Suppress all warnings
warnings.simplefilter('ignore')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Usuario/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to C:\Users\Usuario/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Usuario/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [2]:
```
pd.set_option('display.max_colwidth', 1000)
```

2.2 Functions

Loading [MathJax]/extensions/Safe.js

In [3]:
```
def generalize_tweets(tweet_text):
    """
    Identify if the tweet is about a Google or Apple product, and replace
    with 'tecproduct'.

    Parameters:
    tweet_text (str): The text of the tweet.

    Returns:
    str: 'Google' if the tweet mentions a Google product, 'Apple' if the t
         'Both' if the tweet mentions both, 'Unknown' if it mentions neith
    """
    google_keywords = ['google', 'pixel', 'pixels', 'nexus', 'nexuses', 'a
                       'chromebook', 'chromebooks', 'nest', 'nests', 'stac
    apple_keywords = ['apple', 'apples', 'iphone', 'iphones', 'ipad', 'ipa
                      'macbooks', 'imac', 'imacs', 'watch', 'watches', 'ai
                      'appstore', 'ios', 'itunes']

    # Ensure tweet_text is a string
    if not isinstance(tweet_text, str):
        return 'Unknown'

    # Replace "app store" with "appstore" before tokenization
    tweet_text = tweet_text.replace("app store", "appstore")

    # Replace any occurrences of google_keywords and apple_keywords with '
    for keyword in google_keywords + apple_keywords:
        tweet_text = re.sub(rf'\b{keyword}\b', 'tecproduct', tweet_text, f

    # Replace @ followed by any text or numbers with 'user'
    tweet_text = re.sub(r'@\w+', 'user', tweet_text)

    # Remove # in front of tecproduct if there is
    tweet_text = re.sub(r'#tecproduct', 'tecproduct', tweet_text)

    # Replace # followed by any text or numbers with 'trend'
    tweet_text = re.sub(r'#\w+', 'trend', tweet_text)

    # Remove URLs
    tweet_text = re.sub(r'http\S+|www\S+|https\S+', 'urls', tweet_text, fl

    # Rename 1g, 2g, 3g, 4g, 5g, 6g, to 'monetwork'
    tweet_text = re.sub(r'\dg', 'monetwork', tweet_text)

    return tweet_text

```

Loading [MathJax]/extensions/Safe.js

In [4]:
```
def preprocess_product_mention(df):
    """
    Transforming product_mention to string
    """
    df['product_mention'] = df['product_mention'].astype(str)
    return df
```

3. Code

3.1 Import the database

In [5]:
```
# Load and prepare data
df = pd.read_csv('..\data\df_tweets_clean.csv')
```

3.2 Highlighting tech products

Let's make the tweets lowercase

In [6]:
```
df['tweet_text'] = df['tweet_text'].str.lower()

df.head()
```

Out[6]:

	tweet_text	emotion_type	product_mention
0	.@wesley83 i have a 3g iphone. after 3 hrs tweeting at #rise_austin, it was dead! i need to upgrade. plugin stations at #sxsw.	Not Positive emotion	Apple
1	@jessedee know about @fludapp ? awesome ipad/iphone app that you'll likely appreciate for its design. also, they're giving free ts at #sxsw	Positive emotion	Apple
2	@swonderlin can not wait for #ipad 2 also. they should sale them down at #sxsw.	Positive emotion	Apple
3	@sxsw i hope this year's festival isn't as crashy as this year's iphone app. #sxsw	Not Positive emotion	Apple
4	@sxtxstate great stuff on fri #sxsw: marissa mayer (google), tim o'reilly (tech books/conferences) & matt mullenweg (wordpress)	Positive emotion	Google

We have decided to rename all the technical products in the tweet_text column to a general name called tecproduct. As well as substituting all tags in a tweet with a generic name called User, and the # for another generic name called trend. Lastly, we have also replace all urls with a generic name called url. In this way, we will be able to have a more generalized tweet.

Loading [MathJax]/extensions/Safe.js

In [7]:
```
df['tweet_text'] = df['tweet_text'].map(generalize_tweets)
```

In [8]:
```
df.head()
```

Out[8]:

	tweet_text	emotion_type	product_mention
0	.user i have a monetwork tecproduct. after 3 hrs tweeting at trend, it was dead! i need to upgrade. plugin stations at trend.	Not Positive emotion	Apple
1	user know about user ? awesome tecproduct/tecproduct app that you'll likely appreciate for its design. also, they're giving free ts at trend	Positive emotion	Apple
2	user can not wait for tecproduct 2 also. they should sale them down at trend.	Positive emotion	Apple
3	user i hope this year's festival isn't as crashy as this year's tecproduct app. trend	Not Positive emotion	Apple
4	user great stuff on fri trend: marissa mayer (tecproduct), tim o'reilly (tech books/conferences) & matt mullenweg (wordpress)	Positive emotion	Google

3.3 Text cleaning

3.3.1 Stop Words

Let's now proceed to eliminate the stopwords

In [9]:
```
stopwords_to_remove = stopwords.words('english')

df['tweet_text'] = df['tweet_text'].map(lambda x: ' '.join([word for word

df.head()
```

Out[9]:

	tweet_text	emotion_type	product_mention
0	.user monetwork tecproduct. 3 hrs tweeting trend, dead! need upgrade. plugin stations trend.	Not Positive emotion	Apple
1	user know user ? awesome tecproduct/tecproduct app likely appreciate design. also, they're giving free ts trend	Positive emotion	Apple
2	user wait tecproduct 2 also. sale trend.	Positive emotion	Apple
3	user hope year's festival crashy year's tecproduct app. trend	Not Positive emotion	Apple
4	user great stuff fri trend: marissa mayer (tecproduct), tim o'reilly (tech books/conferences) & matt mullenweg (wordpress)	Positive emotion	Google

We will now proceed to remove strange characters and punctuation

In [10]:

```
# Remove strange characters and punctuation
strange_chars = '!"$%&\'()*+,-./:;<=>?[\\]^_`{|}~“!#•Ûªåçûïòóêâîô¼¾½±º¤¦¬á

df['tweet_text'] = df['tweet_text'].map(lambda x: x.translate(str.maketran

df.head()
```

Out[10]:

	tweet_text	emotion_type	product_mention
0	user monetwork tecproduct 3 hrs tweeting trend dead need upgrade plugin stations trend	Not Positive emotion	Apple
1	user know user awesome tecproduct tecproduct app likely appreciate design also they re giving free ts trend	Positive emotion	Apple
2	user wait tecproduct 2 also sale trend	Positive emotion	Apple
3	user hope year s festival crashy year s tecproduct app trend	Not Positive emotion	Apple
4	user great stuff fri trend marissa mayer tecproduct tim o reilly tech books conferences amp matt mullenweg wordpress	Positive emotion	Google

Now we shall proceed to eliminate numbers

In [11]:

```
# Remove numbers
df['tweet_text'] = df['tweet_text'].map(lambda x: re.sub(r'\d+', '', x))

df.head()
```

Out[11]:

	tweet_text	emotion_type	product_mention
0	user monetwork tecproduct hrs tweeting trend dead need upgrade plugin stations trend	Not Positive emotion	Apple
1	user know user awesome tecproduct tecproduct app likely appreciate design also they re giving free ts trend	Positive emotion	Apple
2	user wait tecproduct also sale trend	Positive emotion	Apple
3	user hope year s festival crashy year s tecproduct app trend	Not Positive emotion	Apple
4	user great stuff fri trend marissa mayer tecproduct tim o reilly tech books conferences amp matt mullenweg wordpress	Positive emotion	Google

We are going to eliminate letters that are on their own in each individual tweet

Loading [MathJax]/extensions/Safe.js

In [12]:
```
df['tweet_text'] = df['tweet_text'].map(lambda x: ' '.join([word for word

df.head()
```

Out[12]:

	tweet_text	emotion_type	product_mention
0	user monetwork tecproduct hrs tweeting trend dead need upgrade plugin stations trend	Not Positive emotion	Apple
1	user know user awesome tecproduct tecproduct app likely appreciate design also they re giving free ts trend	Positive emotion	Apple
2	user wait tecproduct also sale trend	Positive emotion	Apple
3	user hope year festival crashy year tecproduct app trend	Not Positive emotion	Apple
4	user great stuff fri trend marissa mayer tecproduct tim reilly tech books conferences amp matt mullenweg wordpress	Positive emotion	Google

3.4 Lematization

In [13]:
```
# Initialize the WordNet Lemmatizer
lemmatizer = WordNetLemmatizer()

# Lemmatize each word in tweet_text
df['tweet_text'] = df['tweet_text'].map(lambda x: ' '.join([lemmatizer.lem

df.head()
```

Out[13]:

	tweet_text	emotion_type	product_mention
0	user monetwork tecproduct hr tweeting trend dead need upgrade plugin station trend	Not Positive emotion	Apple
1	user know user awesome tecproduct tecproduct app likely appreciate design also they re giving free t trend	Positive emotion	Apple
2	user wait tecproduct also sale trend	Positive emotion	Apple
3	user hope year festival crashy year tecproduct app trend	Not Positive emotion	Apple
4	user great stuff fri trend marissa mayer tecproduct tim reilly tech book conference amp matt mullenweg wordpress	Positive emotion	Google

3.5 Tokenize

Loading [MathJax]/extensions/Safe.js

In [14]:
```
# Tokenize the tweet_text
df['tweet_text_tokenized'] = df['tweet_text'].map(lambda x: word_tokenize(

df.head()
```

Out[14]:

	tweet_text	emotion_type	product_mention	tweet_text_tokenized
0	user monetwork tecproduct hr tweeting trend dead need upgrade plugin station trend	Not Positive emotion	Apple	[user, monetwork, tecproduct, hr, tweeting, trend, dead, need, upgrade, plugin, station, trend]
1	user know user awesome tecproduct tecproduct app likely appreciate design also they re giving free t trend	Positive emotion	Apple	[user, know, user, awesome, tecproduct, tecproduct, app, likely, appreciate, design, also, they, re, giving, free, t, trend]
2	user wait tecproduct also sale trend	Positive emotion	Apple	[user, wait, tecproduct, also, sale, trend]
3	user hope year festival crashy year tecproduct app trend	Not Positive emotion	Apple	[user, hope, year, festival, crashy, year, tecproduct, app, trend]
4	user great stuff fri trend marissa mayer tecproduct tim reilly tech book conference amp matt mullenweg wordpress	Positive emotion	Google	[user, great, stuff, fri, trend, marissa, mayer, tecproduct, tim, reilly, tech, book, conference, amp, matt, mullenweg, wordpress]

3.6 Train test split

Let's first define our variables

In [15]:
```
y = df['emotion_type']
X = df.drop(['emotion_type'], axis=1)
```

We are going to transform the variable y to numeric. Because all models need their target to be numeric. We will use the LabelEncoder.

In [16]:
```
# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Fit the encoder and transform the 'emotion_type' column
df['emotion_type_encoded'] = label_encoder.fit_transform(df['emotion_type'

# The result is a new column 'emotion_type_encoded' with numeric values
y = df['emotion_type_encoded']
```

In [17]:
```
# X is the feature set and y is the target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35,
```

Loading [MathJax]/extensions/Safe.js

Let's compare the shapes of y_test and y_train to see if they're somewhat similar

In [18]:
```
y_test.value_counts(normalize=True)
```

Out[18]:
```
0    0.655877
1    0.344123
Name: emotion_type_encoded, dtype: float64
```

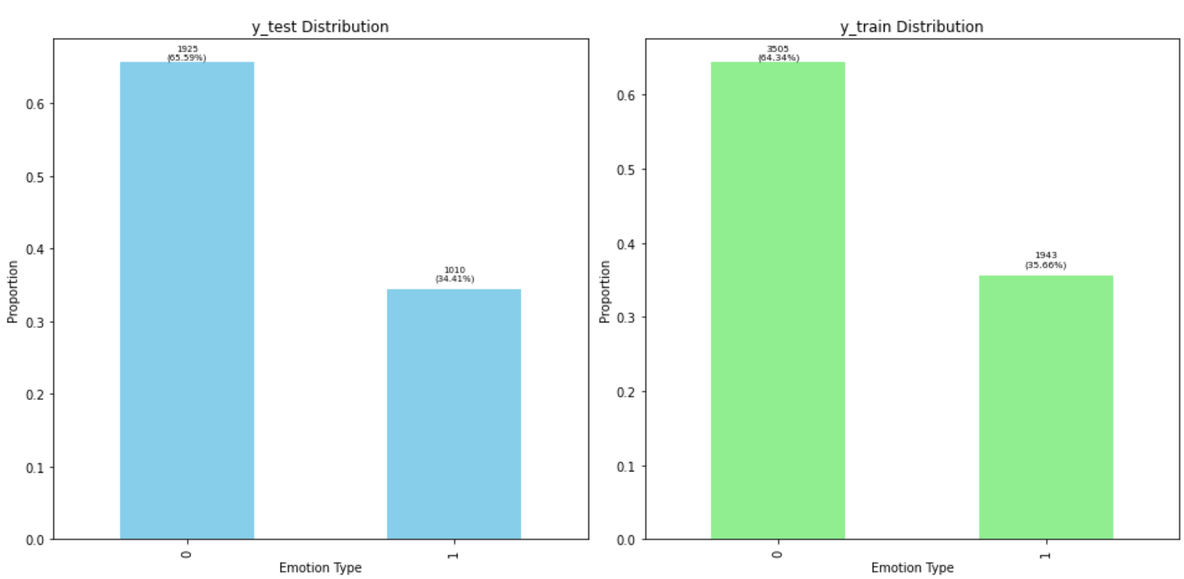
In [19]:
```
y_train.value_counts(normalize=True)
```

Out[19]:
```
0    0.643355
1    0.356645
Name: emotion_type_encoded, dtype: float64
```

Let's see a description of the distributions

Loading [MathJax]/extensions/Safe.js

In [20]:
```
# Create a larger figure
plt.figure(figsize=(14, 7))

# y_test Distribution
plt.subplot(1, 2, 1)
y_test_counts = y_test.value_counts(normalize=True)
y_test_abs_counts = y_test.value_counts()

y_test_counts.plot(kind='bar', color='skyblue')
plt.title('y_test Distribution')
plt.xlabel('Emotion Type')
plt.ylabel('Proportion')

for i, (count, pct) in enumerate(zip(y_test_abs_counts, y_test_counts)):
    vertical_position = pct + 0.002 if pct > 0.5 else pct + 0.01  # Small
    plt.text(i, vertical_position, f'{count}\n({pct:.2%})', ha='center', v

# y_train Distribution
plt.subplot(1, 2, 2)
y_train_counts = y_train.value_counts(normalize=True)
y_train_abs_counts = y_train.value_counts()

y_train_counts.plot(kind='bar', color='lightgreen')
plt.title('y_train Distribution')
plt.xlabel('Emotion Type')
plt.ylabel('Proportion')

for i, (count, pct) in enumerate(zip(y_train_abs_counts, y_train_counts)):
    vertical_position = pct + 0.002 if pct > 0.5 else pct + 0.01  # Small
    plt.text(i, vertical_position, f'{count}\n({pct:.2%})', ha='center', v

# Adjust layout to make more space around the plots
plt.tight_layout()
plt.subplots_adjust(left=0.05, right=0.95, top=0.9, bottom=0.1)
plt.show()

```



Loading [MathJax]/extensions/Safe.js

3.7 TFIDF

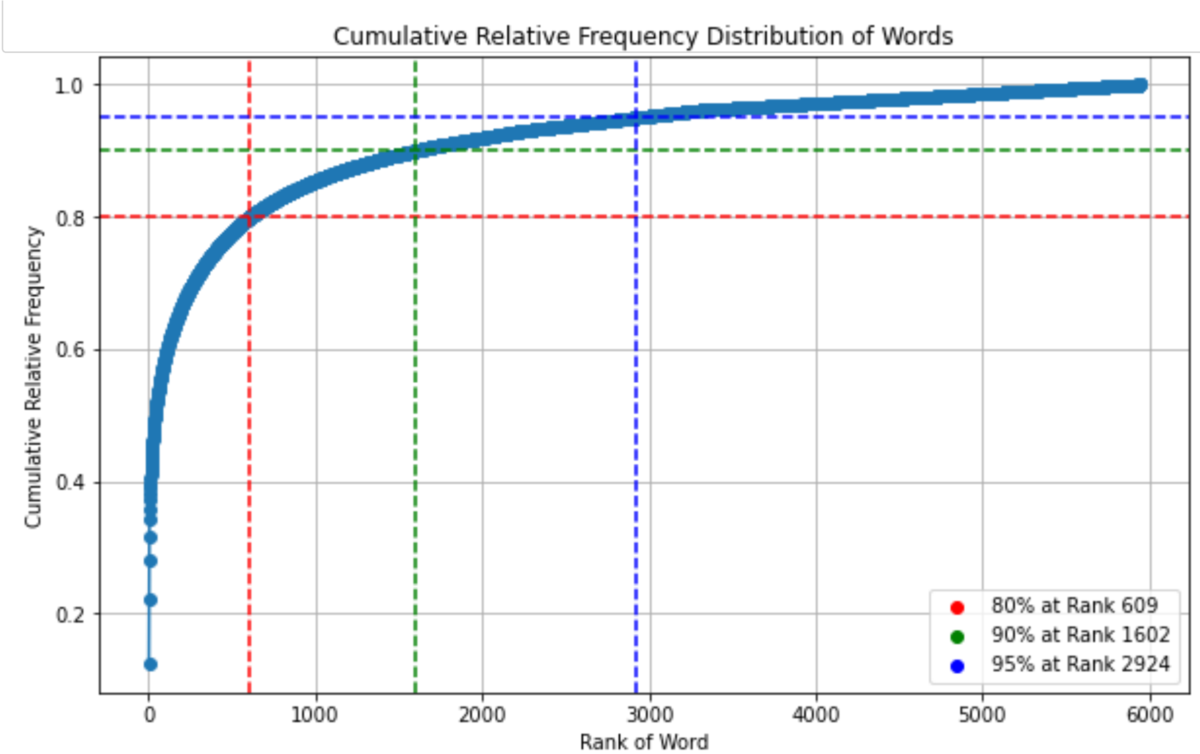
Before starting the procedure of vectorizing our corpus with tfidf, we want to use the a descriptive chart with the cumulative relative frequency of the words in the corpus so we can know the best threshold to set in the vectorizing process.

Loading [MathJax]/extensions/Safe.js

In [21]:

```python
# Flatten the list of lists into a single list
corpus = [word for sublist in X_train['tweet_text_tokenized'] for word in

# Count the frequency of each word
word_freq = Counter(corpus)

# Convert the word frequency to a pandas DataFrame
word_freq_df = pd.DataFrame(word_freq.items(), columns=['Word', 'Frequency

# Sort the DataFrame by frequency in descending order
word_freq_df = word_freq_df.sort_values(by='Frequency', ascending=False).r

# Calculate the cumulative frequency and cumulative relative frequency
word_freq_df['Cumulative Frequency'] = word_freq_df['Frequency'].cumsum()
word_freq_df['Cumulative Relative Frequency'] = word_freq_df['Cumulative F

# Find the rank where the cumulative relative frequency reaches 80%
rank_80_percent = word_freq_df[word_freq_df['Cumulative Relative Frequency
cumulative_80_percent = word_freq_df.loc[rank_80_percent - 1, 'Cumulative

# Find the rank where the cumulative relative frequency reaches 90%
rank_90_percent = word_freq_df[word_freq_df['Cumulative Relative Frequency
cumulative_90_percent = word_freq_df.loc[rank_90_percent - 1, 'Cumulative

# Find the rank where the cumulative relative frequency reaches 95%
rank_95_percent = word_freq_df[word_freq_df['Cumulative Relative Frequency
cumulative_95_percent = word_freq_df.loc[rank_95_percent - 1, 'Cumulative

# Plot the cumulative relative frequency distribution
plt.figure(figsize=(10, 6))
plt.plot(np.arange(1, len(word_freq_df) + 1), word_freq_df['Cumulative Rel
plt.title('Cumulative Relative Frequency Distribution of Words')
plt.xlabel('Rank of Word')
plt.ylabel('Cumulative Relative Frequency')

# Mark the point where the cumulative relative frequency reaches 80%
plt.scatter(rank_80_percent, cumulative_80_percent, color='red', label=f'8
plt.axvline(x=rank_80_percent, color='red', linestyle='--')
plt.axhline(y=cumulative_80_percent, color='red', linestyle='--')

# Mark the point where the cumulative relative frequency reaches 90%
plt.scatter(rank_90_percent, cumulative_90_percent, color='green', label=f
plt.axvline(x=rank_90_percent, color='green', linestyle='--')
plt.axhline(y=cumulative_90_percent, color='green', linestyle='--')

# Mark the point where the cumulative relative frequency reaches 95%
plt.scatter(rank_95_percent, cumulative_95_percent, color='blue', label=f'
plt.axvline(x=rank_95_percent, color='blue', linestyle='--')
plt.axhline(y=cumulative_95_percent, color='blue', linestyle='--')

plt.legend()
plt.grid(True)
plt.show()

```

Loading [MathJax]/extensions/Safe.js



As we can see, by only using 615 words, we have 80% of our entire corpus. And using 1598 words, we have a 90% of our entire corpus. With that in mind, in the past, we thought it made sense to use max_features of 80% in the vectorization process given that we wouldn't win much if we considered 1598 words as it only a gain of 10% more.

However, the results were not convincing and so we decided to eliminate the threshold and to get all the words. In this way we mitigated overfitting and improved the confusion matrix that is shown in notebook 03_modelling.

Knowing our max_features number, let's proceed and do the calculations of tfidf

Loading [MathJax]/extensions/Safe.js

In [22]:

```
# Step 1: Initialize the TfidfVectorizer with the desired max_features par
tfidf_vectorizer = TfidfVectorizer()

# Step 2: Fit the vectorizer on the tweet texts and transform the data int
tfidf_matrix_train = tfidf_vectorizer.fit_transform(X_train['tweet_text'])

# Step 3: Convert the TF-IDF matrix to a DataFrame for easier interpretati
# The DataFrame will have words as columns and documents as rows, with eac
tfidf_df_train = pd.DataFrame(tfidf_matrix_train.toarray(),
                              columns=tfidf_vectorizer.get_feature_names_c
                              index=X_train.index) # Keep the original inc

tfidf_df_train
```

Out[22]:

	aapl	aaron	ab	abacus	abba	abc	aber	ability	able	abnormal	...	zimride	zing	zite
8236	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
6433	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
6987	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
952	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
5769	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5734	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
5191	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
5390	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
860	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
7270	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0

5448 rows × 5918 columns

Now, let's apply the tf-idf on the X_test

Loading [MathJax]/extensions/Safe.js

In [23]:

```
# Step 1: Transform the tweet texts on the X_test with the vectorizer and
tfidf_matrix_test = tfidf_vectorizer.transform(X_test['tweet_text'])

# Step 2: Convert the TF-IDF matrix to a DataFrame for easier interpretati
# The DataFrame will have words as columns and documents as rows, with eac
tfidf_df_test = pd.DataFrame(tfidf_matrix_test.toarray(),
                             columns=tfidf_vectorizer.get_feature_names_ou
                             index=X_test.index) # Keep the original index

tfidf_df_test
```

Out[23]:

	aapl	aaron	ab	abacus	abba	abc	aber	ability	able	abnormal	...	zimride	zing
2865	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.379029	0.0	...	0.0	0.0
8174	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0
5933	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0
4857	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0
6858	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2662	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0
5958	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0
5840	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0
2154	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0
1344	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0

2935 rows × 5918 columns

We are first going to merge the product_mention and the y_train, y_test columns to tfidf_df_train and tfidf_df_test separately.

Loading [MathJax]/extensions/Safe.js

In [24]:

```
# For tfidf_df_train. Let's join tfidf_df_train and X_train by index
df_train = X_train[['product_mention']].join(tfidf_df_train)

# Let's now join y_train with df_train
df_train = df_train.join(y_train)

df_train
```

Out[24]:

	product_mention	aapl	aaron	ab	abacus	abba	abc	aber	ability	able	...	zing	zite
8236	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
6433	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
6987	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
952	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
5769	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5734	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
5191	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
5390	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
860	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
7270	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0

5448 rows × 5920 columns

In [25]:

```python
# For tfidf_df_test. Let's join tfidf_df_test and X_test by index
df_test = X_test[['product_mention']].join(tfidf_df_test)

# Let's now join y_test with df_test
df_test = df_test.join(y_test)

df_test
```

Out[25]:

	product_mention	aapl	aaron	ab	abacus	abba	abc	aber	ability	able	...	zing	z
2865	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.379029	...	0.0	(
8174	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(
5933	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(
4857	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(
6858	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(
...	...	...	...	...	...	...	...	...	...	...	...	...	
2662	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(
5958	Google	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(
5840	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(
2154	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(
1344	Apple	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	(

2935 rows × 5920 columns

We shall now proceed to save the pickle of the tfidf_vectorizer

In [26]:

```python
# Save the tfidf_vectorizer in a pickle folder called pickle_objects
with open('..\pickle_objects/tfidf_vectorizer.pkl', 'wb') as file:
    pickle.dump(tfidf_vectorizer, file)
```

Let's do a one hot encoder on the product_mention column of df_train

In [27]:

```
# Initialize the OneHotEncoder
onehot_encoder = OneHotEncoder(sparse=False, drop='first')  # drop='first'

# Fit and transform the 'product_mention' column
encoded_features = onehot_encoder.fit_transform(df_train[['product_mentior

# Convert the array back to a DataFrame
encoded_df = pd.DataFrame(encoded_features, columns=onehot_encoder.get_fea

# Concatenate the encoded columns back to the original DataFrame
df_train_encoded = pd.concat([df_train.reset_index(drop=True), encoded_df]

# Drop the 'product_mention' column
df_train_encoded.drop('product_mention', axis=1, inplace=True)

# Display the DataFrame with the encoded columns
df_train_encoded.head()
```

Out[27]:

	aapl	aaron	ab	abacus	abba	abc	aber	ability	able	abnormal	...	zms	zombie	zomg
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0

5 rows × 5921 columns

Let's do the One Hot Encoder for X_test

Loading [MathJax]/extensions/Safe.js

In [28]:

```python
# Transform the 'product_mention' column
encoded_features = onehot_encoder.transform(df_test[['product_mention']])

# Convert the array back to a DataFrame
encoded_df = pd.DataFrame(encoded_features, columns=onehot_encoder.get_fea

# Concatenate the encoded columns back to the original DataFrame
df_test_encoded = pd.concat([df_test.reset_index(drop=True), encoded_df],

# Drop the 'product_mention' column
df_test_encoded.drop('product_mention', axis=1, inplace=True)

# Display the DataFrame with the encoded columns
df_test_encoded.head()
```

Out[28]:

	aapl	aaron	ab	abacus	abba	abc	aber	ability	able	abnormal	...	zms	zombie	zor
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.379029	0.0	...	0.0	0.0	(
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	(
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	(
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	(
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	(

5 rows × 5921 columns

Let's save the pickle for the One Hot Encoder

In [29]:

```python
# Save the ohe in a pickle folder called pickle_objects
with open('..\pickle_objects/ohe.pkl', 'wb') as file:
    pickle.dump(onehot_encoder, file)
```

4. Export to csv

In [30]:

```python
# Save df_train_encoded to a CSV file
df_train_encoded.to_csv('../data/train_processed.csv', index=False)
# The index=False argument ensures that the DataFrame index is not writter

# Save df_test_encoded to a CSV file
df_test_encoded.to_csv('../data/test_processed.csv', index=False)
```

Loading [MathJax]/extensions/Safe.js