Sentiment Analysis of Apple vs Google

Business Understanding

The dataset presents an analysis of different emotions drawn out of tweets about either an Apple or Google product.The Businesses can better understand client impressions in real time by classifying tweet sentiment (positive, negative, and neutral) using natural language processing (NLP).

Problem Statement

With sizable and devoted user bases, Apple and Google are significant actors in the tech sector. Maintaining competitiveness, controlling brand reputation, and enhancing services all depend on knowing how customers feel about their products.

It is not scalable to manually track public sentiment given the hundreds of tweets regarding their products that are posted every day. An automated method is required in order to categorize tweets about Apple and Google into favorable, negative, and neutral sentiments.

Objectives

1. Build a model that can rate the sentiment of a Tweet based on its content. Using tweet content to base predictions.
2. Building a binary classifier to distinguish between positive and negative tweets.
3. Building a multiclass classification.
4. Evaluating model performance using evaluation metrics like Accuracy and Precision for binary and Multiclass: Macro F1-score, Weighted Accuracy, and per-class performance

Data Understanding

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
from tensorflow import keras
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classif
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from imblearn.over_sampling import SMOTE
from wordcloud import WordCloud
from wordcloud import WordCloud
nltk.download('punkt', quiet=True)
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
#loading the dataset
df = pd.read_csv('tweet_product_company.csv', encoding='latin-1')
df.head()
```

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Neg
1	@jessedee Know about @fludapp ?	iPad or iPhone App	Po

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column                                              Non-Null Count  Dtype 
---  ------                                              --------------  ----- 
 0   tweet_text                                          9092 non-null   object
 1   emotion_in_tweet_is_directed_at                     3291 non-null   object
 2   is_there_an_emotion_directed_at_a_brand_or_product  9093 non-null   object
dtypes: object(3)
memory usage: 213.2+ KB
```

```
df.isna().sum()
```

	0
tweet_text	1
emotion_in_tweet_is_directed_at	5802
is_there_an_emotion_directed_at_a_brand_or_product	0

dtype: int64

```
df['emotion_in_tweet_is_directed_at'].unique()
```

```
array(['iPhone', 'iPad or iPhone App', 'iPad', 'Google', nan, 'Android',
       'Apple', 'Android App', 'Other Google product or service',
       'Other Apple product or service'], dtype=object)
```

```
df['is_there_an_emotion_directed_at_a_brand_or_product'].unique()
```

```
array(['Negative emotion', 'Positive emotion',
       'No emotion toward brand or product', "I can't tell"], dtype=object)
```

```
df.describe()
```

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_
count	9092	3291	
unique	9065	9	
	RT @mention Marissa		

Data Cleaning

```
# Copy of the original dataset
tweets_df = df.copy()


#fill null values with a placeholder
tweets_df['emotion_in_tweet_is_directed_at'] = tweets_df['emotion_in_tweet_is_directed_at'].

#drop the nan values
tweets_df.dropna(inplace = True)

#verify the result
tweets_df.isna().sum()
```

	0
tweet_text	0
emotion_in_tweet_is_directed_at	0
is_there_an_emotion_directed_at_a_brand_or_product	0

dtype: int64

```
# Renaming the columns
tweets_df = tweets_df.rename(columns = {'emotion_in_tweet_is_directed_at': 'brand',
                            'is_there_an_emotion_directed_at_a_brand_or_product': 'sentiment
                            'tweet_text': 'tweet'})


# verifying the results
tweets_df['brand'].unique()
```

```
array(['iPhone', 'iPad or iPhone App', 'iPad', 'Google', 'unknown',
       'Android', 'Apple', 'Android App',
       'Other Google product or service',
       'Other Apple product or service'], dtype=object)
```

```
sentiment_map = {
    'Negative emotion': 'negative',
    'Positive emotion': 'positive',
    'No emotion toward brand or product': 'neutral',
    "I can't tell": 'unclear'
}

tweets_df['sentiment'] = tweets_df['sentiment'].map(sentiment_map)



# verifying the results
tweets_df.head()
```

	tweet	brand	sentiment
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	negative
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	positive
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	positive
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	negative
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	positive

```
tweets_df.isna().sum()
```

	0
tweet	0
brand	0
sentiment	0

dtype: int64

```
tweets_df.duplicated().sum()
```

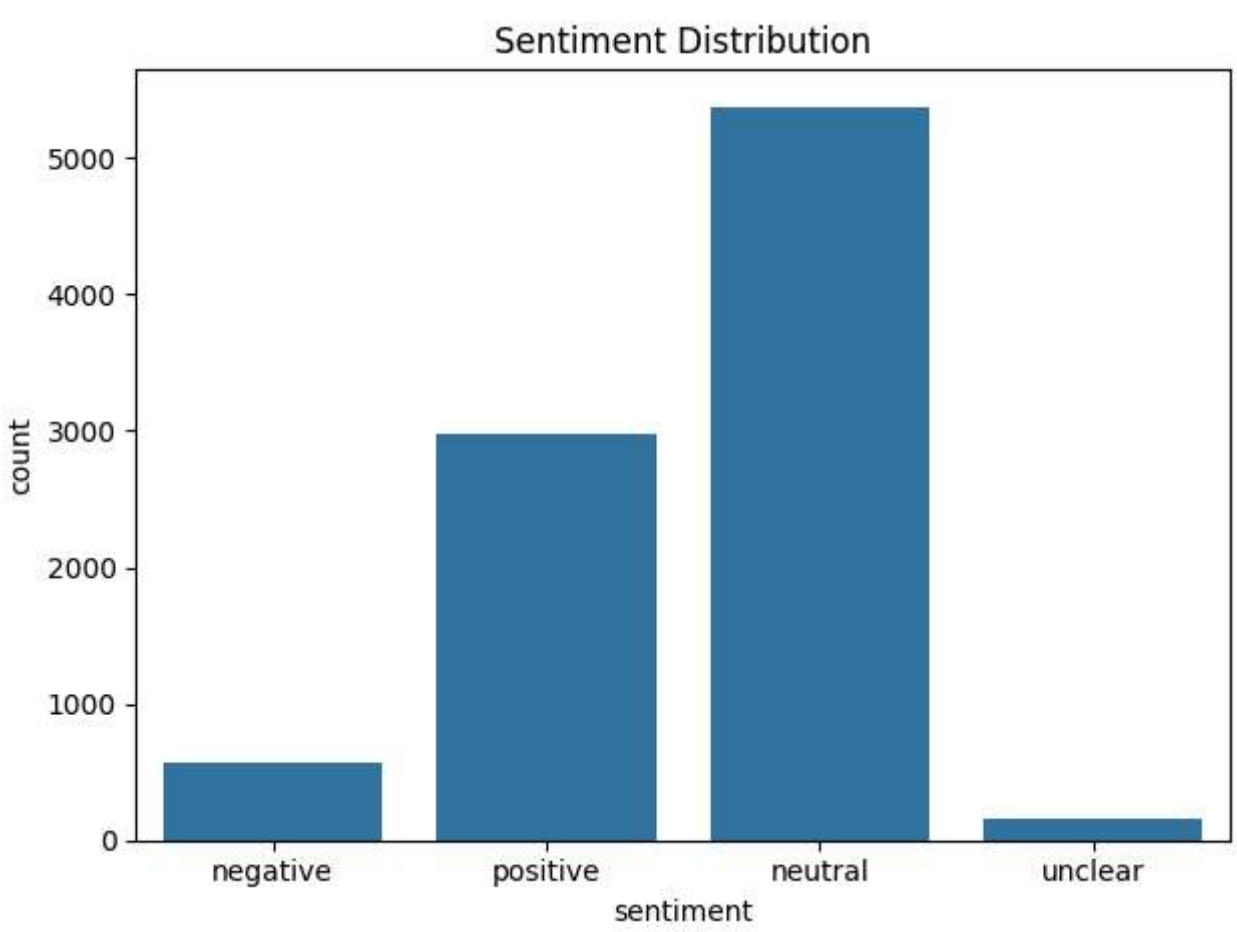
```
np.int64(22)
```

```
#droping the duplicates
tweets_df.drop_duplicates(inplace = True)


tweets_df.duplicated().sum()
```

```
np.int64(0)
```

EDA

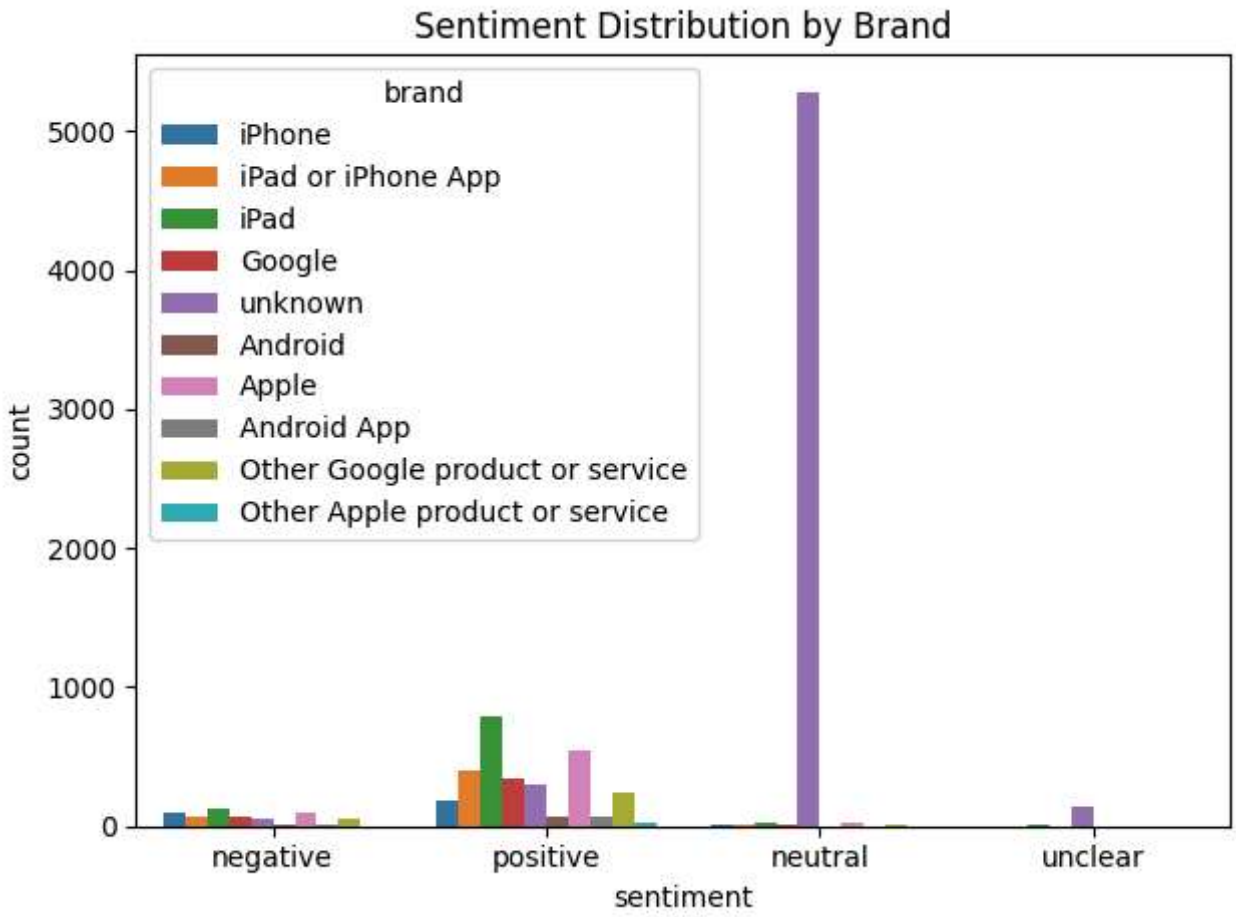
```
# sentiment distribution
sns.countplot(x = 'sentiment', data= tweets_df)
plt.title('Sentiment Distribution')
plt.tight_layout()
plt.show()
```



Overall,

Most tweets are neutral(no emotion) Far tweets express negative sentiment. This imbalance can make it harder for a model to learn how to correctly identify minority classes(especially negative).

```
# plot countplot showing the sentiment by their brand

sns.countplot(x='sentiment', hue='brand', data=tweets_df)
plt.title('Sentiment Distribution by Brand')
plt.tight_layout()
plt.show()
```

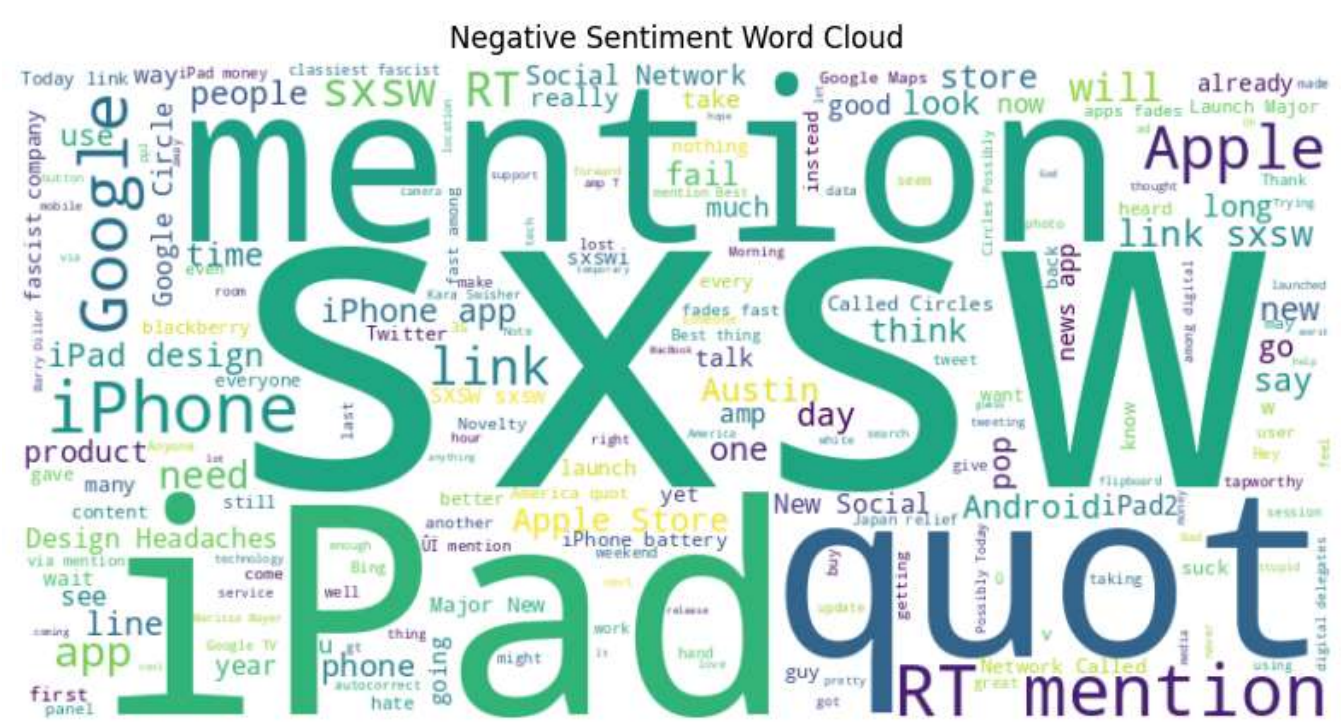


The data is largely dominated by neutral sentiments, especially from the unknown brand group.

Among known brands, iPad and Apple have the strongest positive sentiment.

There's no strong negative trend for any specific brand, which could be good news for all involved.

```
#word cloud for negative sentiments


text = ' '.join(tweets_df[tweets_df['sentiment'] == 'negative']['tweet'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Negative Sentiment Word Cloud")
plt.show()
```



This plot visualizes the frequency of words found in tweets with negative sentiments. Words such as SXSW, mention, ipad,quot and apple are larger which shows they are mentioned more frequently in negative contexts.

```
#word cloud for positive sentiments

text = ' '.join(tweets_df[tweets_df['sentiment'] == 'positive']['tweet'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Positive Sentiment Word Cloud")
plt.show()
```

This plot shows that words like iphone, ipad, apple, Google, SXSW, RT and link appear bolder and larger, which indicates they are mentioned more often in positive contexts

Text preprocessing

```
#cleaning the text
def clean_tweet(text):
    if not isinstance(text, str):
        return ""
    text = re.sub(r"@[A-Za-z0-9]+", "", text)
    text = re.sub(r"#", "", text)
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)
    return text

tweets_df['tweet'] = tweets_df['tweet'].apply(clean_tweet)



def preprocess_text(text):
    text = text.lower()


    text = ' '.join([word for word in text.split() if word not in stopwords.words('english')
```

```
    stemmer = PorterStemmer()

    text = ' '.join([stemmer.stem(word) for word in text.split()])

    return text


tweets_df['tweet'] = tweets_df['tweet'].apply(preprocess_text)


tweets_df['tweet']
```

	tweet
0	3g iphon 3 hr tweet riseaustin dead need upgra...
1	know awesom ipadiphon app youll like appreci d...
2	wait ipad 2 also sale sxsw
3	hope year festiv isnt crashi year iphon app sxsw
4	great stuff fri sxsw marissa mayer googl tim o...
...	...
9088	ipad everywher sxsw link
9089	wave buzz rt interrupt regularli schedul sxsw ...
9090	googl zeiger physician never report potenti ae...
9091	verizon iphon custom complain time fell back h...
9092	rt googl test checkin offer sxsw link

9070 rows × 1 columns

dtype: object

Preprocessing is executed by cleaning up tweet text, by removing noise like mentions, hashtags, and special characters, making the text more suitable for further analysis or modeling.

There is further preprocessing of the text data by:

Lowercasing: Converts all text to lowercase to ensure uniformity.

Stop Word Removal: Removes common words (like "the", "a", "is") that don't usually carry significant meaning.

Stemming: Reduces words to their root form (e.g., "running" becomes "run") to group similar words together.

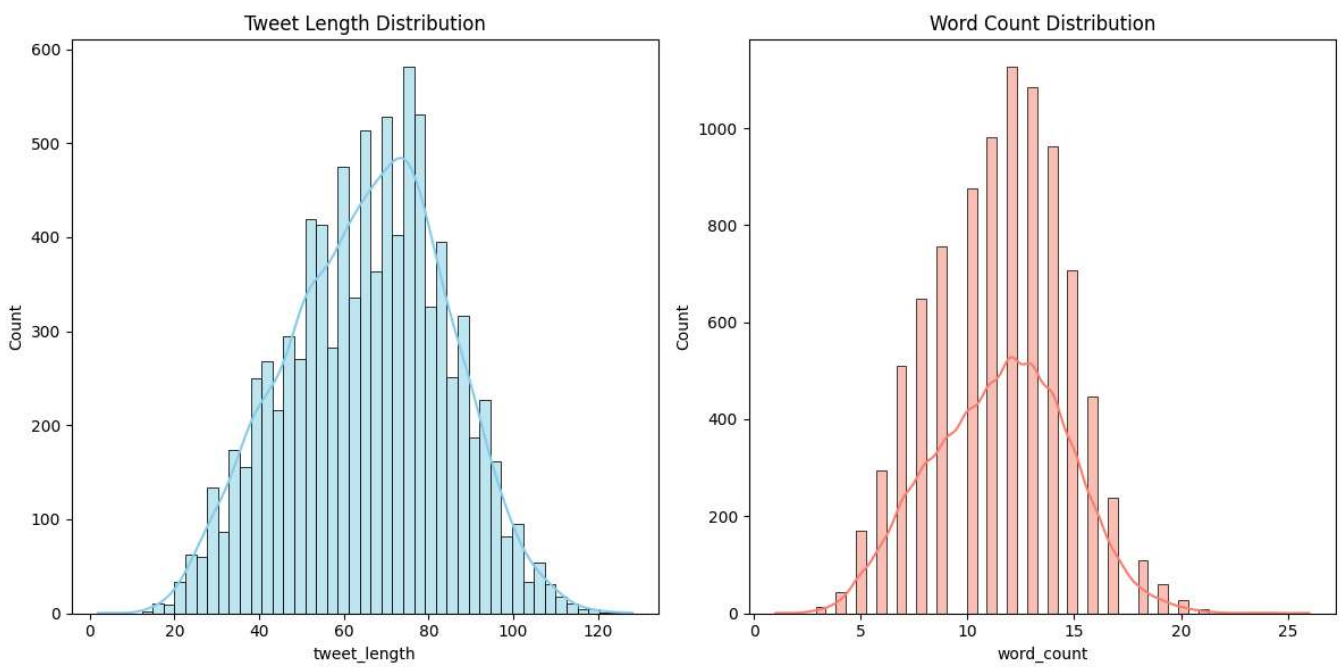
Feature Engineering

```
#calculate the length of each tweet in characters
tweets_df['tweet_length'] = tweets_df['tweet'].apply(len)
tweets_df['word_count'] = tweets_df['tweet'].apply(lambda x: len(x.split()))
#display the first 5 rows
tweets_df[['tweet', 'tweet_length', 'word_count']].head()
```

	tweet	tweet_length	word_count
0	3g iphon 3 hr tweet riseaustin dead need upgra...	67	12
1	know awesom ipadiphon app youll like appreci d...	80	14
2	wait ipad 2 also sale sxsw	26	6
3	hope year festiv isnt crashi year iphon app sxsw	48	9
4	great stuff fri sxsw marissa mayer googl tim o...	98	15

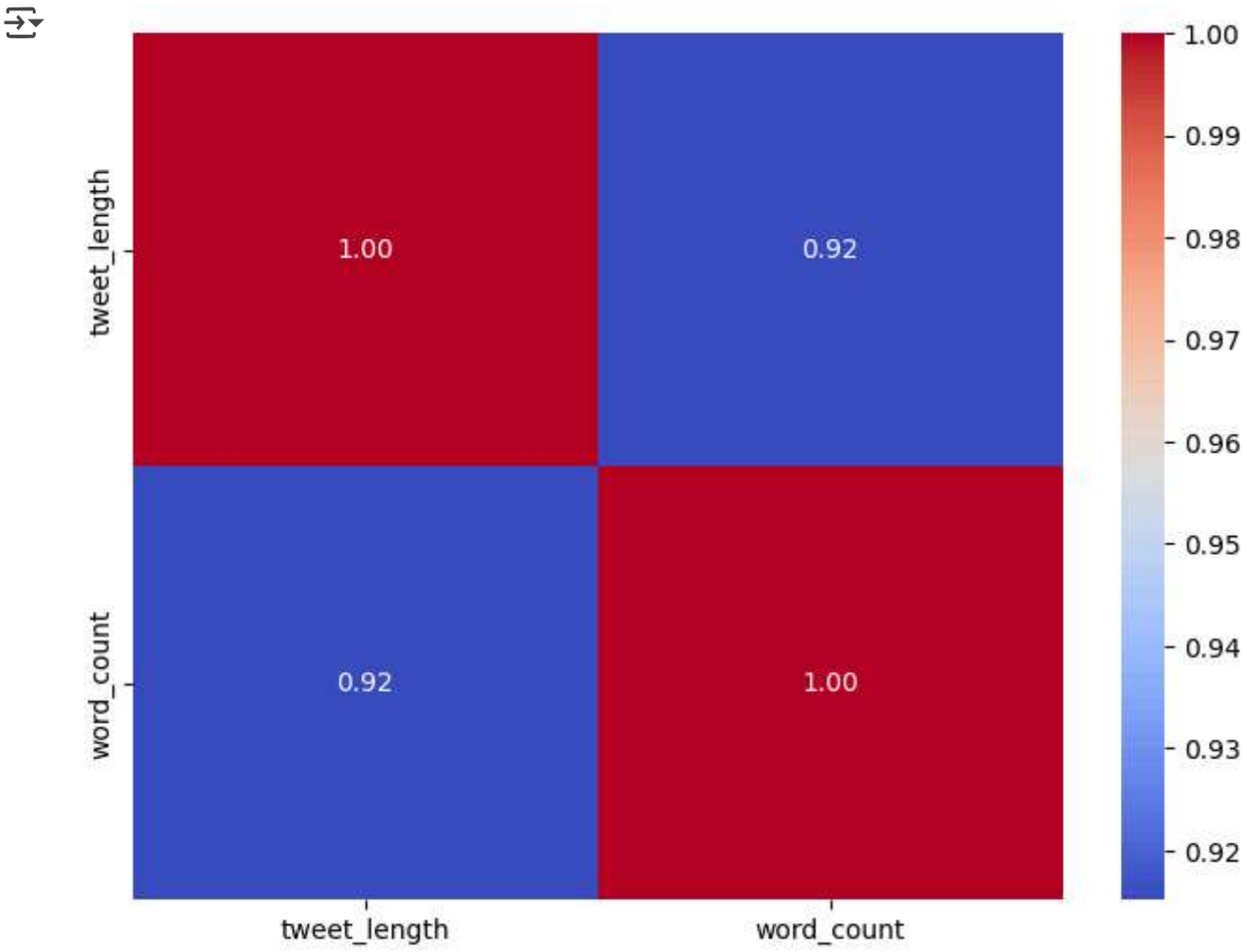
Feature engineering is executed by extracting basic information about the length and word count of tweets, which can be potentially useful for later analysis or modeling tasks.

```
#create a figure
plt.figure(figsize=(12, 6))

#create a subplot
plt.subplot(1, 2, 1)
#generate a histogram
sns.histplot(tweets_df['tweet_length'], kde=True, color='skyblue')
plt.title('Tweet Length Distribution')

# Plotting word count distribution
plt.subplot(1, 2, 2)
sns.histplot(tweets_df['word_count'], kde=True, color='salmon')
plt.title('Word Count Distribution')
#display the plot
plt.tight_layout()
plt.show()
```



Overall, there's two histograms that visualize the distribution of tweet lengths and word counts within the tweets_df DataFrame.

```
#Correlation
correlation_matrix = tweets_df[['tweet_length', 'word_count']].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f");
```



As the number of words in a tweet increases, the tweet length (likely measured in characters) also increases — and vice versa.

This is expected, but the correlation of 0.92 quantifies this relationship as very strong.

This suggests that either variable could be a good proxy for the other in modeling or exploratory data analysis.

Modeling

Label Encoding

```
# Create a LabelEncoder instance
label_encoder = LabelEncoder()

#Create a new dataframe containing positive and negative tweets
binary_df = tweets_df[tweets_df['sentiment'].isin(['positive', 'negative'])].copy()
```

```
#Encode the 'sentiment' column
binary_df['sentiment_encoded'] = label_encoder.fit_transform(binary_df['sentiment'])
#Display the count of each encoded sentiment
binary_df['sentiment_encoded'].value_counts()
```

sentiment_encoded	count
1	2970
0	569

dtype: int64

Binary Classification

```
#features and target
X = binary_df['tweet']
y = binary_df['sentiment_encoded']
#splitting the data into trainign and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Vectorizing the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

#print the shapes of the data
print("X_train vectorized shape:", X_train_vec.shape)
print("X_test vectorized shape:", X_test_vec.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train vectorized shape: (2831, 4611)
X_test vectorized shape: (708, 4611)
y_train shape: (2831,)
y_test shape: (708,)
```

```
#Bag of words
# Fiting the vectorizer to the 'tweet' column
X_counts = vectorizer.fit_transform(tweets_df['tweet'])

# Getting the vocabulary and suming up the counts for each word across all documents
vocab = vectorizer.get_feature_names_out()
word_counts = np.array(X_counts.sum(axis=0)).flatten()

# Creating a DataFrame to easily view
```

```
frequency_df = pd.DataFrame({'word': vocab, 'count': word_counts})
frequency_df = frequency_df.sort_values('count', ascending=False)

print(frequency_df.head(20))
```

```
          word       count
7340      sxsw  555.712868
4395      link  479.452663
6496        rt  378.504804
681       appl  355.128913
3215     googl  339.322860
3934      ipad  337.025969
7168     store  284.828963
3964     iphon  229.844676
5030       new  205.263842
827     austin  195.174591
4281    launch  190.483895
671        app  186.245439
1582     circl  160.653810
6954    social  152.992672
5675     popup  150.296550
5266      open  141.951639
7703     today  134.075607
5021   network  129.660129
595        amp  129.497295
3936     ipad2  118.484243
```

Logistic Regression Model

```
# Instatiate logistic regression
logreg_model = LogisticRegression(max_iter=1000)
# fit the model to the training data
logreg_model.fit(X_train_vec, y_train)

#make predictions on test data
y_pred_logreg = logreg_model.predict(X_test_vec)

accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
#print the accuracy and classification
print(f"Logistic Regression Accuracy: {accuracy_logreg}")
print(classification_report(y_test, y_pred_logreg))
```

```
Logistic Regression Accuracy: 0.8686440677966102
              precision    recall  f1-score   support

           0       0.93      0.13      0.23       106
           1       0.87      1.00      0.93       602

    accuracy                           0.87       708
   macro avg       0.90      0.57      0.58       708
```

```
weighted avg       0.88      0.87      0.82       708
```

Class Imbalance

```
# Applying SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train_vec, y_train)

# Splitting the resampled data into training and testing sets
X_train_resampled, X_test_resampled, y_train_resampled, y_test_resampled = train_test_split(
    X_resampled, y_resampled, test_size=0.2, random_state=42)

# Initialize and train the Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_logreg = logreg_model.predict(X_test_vec)

# Evaluate the model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print(f"Logistic Regression Accuracy: {accuracy_logreg}")
print(classification_report(y_test, y_pred_logreg))
```

```
Logistic Regression Accuracy: 0.8573446327683616
              precision    recall  f1-score   support

           0       0.52      0.66      0.58       106
           1       0.94      0.89      0.91       602

    accuracy                           0.86       708
   macro avg       0.73      0.78      0.75       708
weighted avg       0.87      0.86      0.86       708
```

Accuracy: 86% looks good at first glance. The model learns to be really good at classifying class 1

Multinomial NaiveBayes

```
# Initialize and train the Multinomial Naive Bayes model
mnb_model = MultinomialNB()
mnb_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_mnb = mnb_model.predict(X_test_resampled)
```

```
# Evaluate the model
accuracy_mnb = accuracy_score(y_test_resampled, y_pred_mnb)
print(f"Multinomial Naive Bayes Accuracy: {accuracy_mnb}")
print(classification_report(y_test_resampled, y_pred_mnb))
```

```
Multinomial Naive Bayes Accuracy: 0.9092827004219409
              precision    recall  f1-score   support

           0       0.87      0.96      0.91       474
           1       0.96      0.86      0.90       474

    accuracy                           0.91       948
   macro avg       0.91      0.91      0.91       948
weighted avg       0.91      0.91      0.91       948
```

Hyperparameter Tuning

```
# Define the parameter grid for MultinomialNB
param_grid = {
    'alpha': [0.1, 0.5, 1.0, 2.0],
    'fit_prior': [True, False]
}

# Create a MultinomialNB classifier
mnb_model = MultinomialNB()

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=mnb_model, param_grid=param_grid, cv=5, scoring='accura

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train_resampled, y_train_resampled)

# Print the best hyperparameters and the corresponding accuracy
print("Best hyperparameters:", grid_search.best_params_)
print("Best accuracy:", grid_search.best_score_)

# Evaluate the best model on the test set
best_mnb_model = grid_search.best_estimator_
y_pred_mnb = best_mnb_model.predict(X_test_resampled)
accuracy_mnb = accuracy_score(y_test_resampled, y_pred_mnb)

print(f"Multinomial Naive Bayes Accuracy (best model): {accuracy_mnb}")
print(classification_report(y_test_resampled, y_pred_mnb))
```

```
Best hyperparameters: {'alpha': 0.1, 'fit_prior': True}
Best accuracy: 0.9252883378702906
Multinomial Naive Bayes Accuracy (best model): 0.930379746835443
              precision    recall  f1-score   support
```

```
                0       0.89      0.98      0.93       474
                1       0.98      0.88      0.93       474

         accuracy                           0.93       948
        macro avg       0.93      0.93      0.93       948
     weighted avg       0.93      0.93      0.93       948
```

Confusion Matrix based on binary

```
cm = confusion_matrix(y_test, y_pred_logreg)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

The model performs quite well with high precision (few false positives) and high recall (few false negatives).

An accuracy of 85.7% is decent, but the F1 score is even more informative in this context at 91.4%, which shows a good balance between precision and recall.

In the confusion matrix there is four categories:

True Negatives (70) – Correctly predicted negative cases.

False Positives (36) – Incorrectly predicted positive cases when they were actually negative.

False Negatives (65) – Incorrectly predicted negative cases when they were actually positive.

True Positives (537) – Correctly predicted positive cases.

NN-Models

ANN Model (Sigmoid)

```
# Define the ANN model
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train_vec.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(X_train_resampled, y_train_resampled, epochs=10, batch_size=32, validation_split=0

# Make predictions on the test set
y_pred_prob = model.predict(X_test_resampled)
y_pred = (y_pred_prob > 0.5).astype(int)

# Evaluate the model
accuracy = accuracy_score(y_test_resampled, y_pred)
precision = precision_score(y_test_resampled, y_pred)
recall = recall_score(y_test_resampled, y_pred)
f1 = f1_score(y_test_resampled, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
```

```
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.6809 - loss: 0.6610 - val_accuracy
Epoch 2/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9621 - loss: 0.2163 - val_accuracy
Epoch 3/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9926 - loss: 0.0475 - val_accuracy
Epoch 4/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9990 - loss: 0.0146 - val_accuracy
Epoch 5/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9988 - loss: 0.0092 - val_accuracy:
Epoch 6/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9995 - loss: 0.0049 - val_accuracy:
Epoch 7/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9993 - loss: 0.0043 - val_accuracy:
Epoch 8/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9995 - loss: 0.0039 - val_accuracy:
Epoch 9/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9988 - loss: 0.0053 - val_accuracy:
Epoch 10/10
95/95 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9981 - loss: 0.0051 - val_accuracy
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Accuracy: 0.9588607594936709
Precision: 0.9865771812080537
Recall: 0.930379746835443
F1-score: 0.9576547231270358
```

Multiclass Classification

```
# Create a LabelEncoder instance
label_encoder = LabelEncoder()

# Filter for positive, negative, and neutral sentiments
multiclass_df = tweets_df[tweets_df['sentiment'].isin(['positive', 'negative', 'neutral'])].

# Encode the 'sentiment' column
multiclass_df['sentiment_encoded'] = label_encoder.fit_transform(multiclass_df['sentiment'])

# Display the value counts of encoded sentiments
print(multiclass_df['sentiment_encoded'].value_counts())
```

```
sentiment_encoded
1    5375
2    2970
0     569
Name: count, dtype: int64
```

```
# Split the data into training and testing sets
X = multiclass_df['tweet']
y = multiclass_df['sentiment_encoded']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

print("X_train vectorized shape:", X_train_vec.shape)
print("X_test vectorized shape:", X_test_vec.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train vectorized shape: (7131, 7727)
X_test vectorized shape: (1783, 7727)
y_train shape: (7131,)
y_test shape: (1783,)
```

```
multiclass_pipe = Pipeline([('tfidf', TfidfVectorizer(stop_words='english')),
    ('nb', MultinomialNB())])
multiclass_pipe.fit(X_train, y_train)
y_pred = multiclass_pipe.predict(X_test)


print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.01      0.02       117
           1       0.65      0.95      0.77      1077
           2       0.69      0.24      0.36       589

    accuracy                           0.66      1783
   macro avg       0.78      0.40      0.38      1783
weighted avg       0.69      0.66      0.59      1783
```

Class 0: Precision: 1.00 When it predicts class 0, it's always correct.

Recall: 0.01 But it almost never predicts class 0 correctly.

F1-score: 0.02 Very poor overall performance for class 0.

Class 1: Precision: 0.65

Recall: 0.95 Model does a great job detecting class 1.

F1-score: 0.77 Strong performance.

Class 2: Precision: 0.69

Recall: 0.24 Model misses most class 2 instances.

F1-score: 0.36 Needs improvement.

Class 1 dominates predictions.

The model performs very well on Class 1.

Class 0 is almost completely misclassified, with only 1 correct prediction.

Class 2 is mostly confused with Class 1, indicating a need to improve class separation, especially between Class 1 and Class 2.

Class Imbalance

```
# Apply SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train_vec, y_train)

# Split the resampled data into training and testing sets
X_train_resampled, X_test_resampled, y_train_resampled, y_test_resampled = train_test_split(
    X_resampled, y_resampled, test_size=0.2, random_state=42
)

# Initialize and train the Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_logreg = logreg_model.predict(X_test_resampled)

# Evaluate the model
accuracy_logreg = accuracy_score(y_test_resampled, y_pred_logreg)
print(f"Logistic Regression Accuracy: {accuracy_logreg}")
print(classification_report(y_test_resampled, y_pred_logreg))
```

```
Logistic Regression Accuracy: 0.7999224505622334
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       872
           1       0.77      0.68      0.72       904
           2       0.72      0.74      0.73       803

    accuracy                           0.80      2579
   macro avg       0.79      0.80      0.80      2579
weighted avg       0.80      0.80      0.80      2579
```

Class 0 is dominating in performance.

Class 1 has the lowest recall the model struggles more to identify actual class 1s.

Class 2 is performing moderately well.

```
from collections import Counter
print("Class distribution before SMOTE:", Counter(y_train))
print("Class distribution after SMOTE:", Counter(y_resampled))
```

```
Class distribution before SMOTE: Counter({1: 4298, 2: 2381, 0: 452})
Class distribution after SMOTE: Counter({1: 4298, 2: 4298, 0: 4298})
```

ANN Model (Softmax)

```
# Define the ANN model
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(X_train_vec.shape[1],)),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_resampled, y_train_resampled, epochs=20, batch_size=64, validation_split=0

# Make predictions on the test set
y_pred_prob = model.predict(X_test_resampled)
y_pred = np.argmax(y_pred_prob, axis=1)

# Evaluate the model
accuracy = accuracy_score(y_test_resampled, y_pred)
precision = precision_score(y_test_resampled, y_pred, average='macro')
recall = recall_score(y_test_resampled, y_pred, average='macro')
f1 = f1_score(y_test_resampled, y_pred, average='macro')


print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
print(classification_report(y_test_resampled, y_pred))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 5s 24ms/step - accuracy: 0.5107 - loss: 1.0021 - val_acc
Epoch 2/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 24ms/step - accuracy: 0.8255 - loss: 0.4240 - val_acc
Epoch 3/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9072 - loss: 0.2424 - val_acc
Epoch 4/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 32ms/step - accuracy: 0.9284 - loss: 0.1799 - val_acc
Epoch 5/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 24ms/step - accuracy: 0.9450 - loss: 0.1406 - val_acc
Epoch 6/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.9581 - loss: 0.1175 - val_acc
Epoch 7/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.9633 - loss: 0.0996 - val_acc
Epoch 8/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 30ms/step - accuracy: 0.9632 - loss: 0.0899 - val_acc
Epoch 9/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.9656 - loss: 0.0872 - val_acc
Epoch 10/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.9649 - loss: 0.0846 - val_acc
Epoch 11/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.9680 - loss: 0.0774 - val_acc
Epoch 12/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 30ms/step - accuracy: 0.9650 - loss: 0.0813 - val_acc
Epoch 13/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.9729 - loss: 0.0660 - val_acc
Epoch 14/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.9713 - loss: 0.0624 - val_acc
Epoch 15/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 6s 32ms/step - accuracy: 0.9736 - loss: 0.0598 - val_acc
Epoch 16/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.9730 - loss: 0.0619 - val_acc
Epoch 17/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 6s 26ms/step - accuracy: 0.9717 - loss: 0.0584 - val_acc
Epoch 18/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - accuracy: 0.9740 - loss: 0.0526 - val_acc
Epoch 19/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.9763 - loss: 0.0507 - val_acc
Epoch 20/20
129/129 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.9780 - loss: 0.0492 - val_acc
81/81 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
Accuracy: 0.8255137650252036
Precision: 0.8256971627105986
Recall: 0.8276879240600449
F1-score: 0.8221326044065637
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       872
           1       0.82      0.66      0.73       904
           2       0.72      0.83      0.77       803

    accuracy                           0.83      2579
   macro avg       0.83      0.83      0.82      2579
weighted avg       0.83      0.83      0.82      2579
```

LSTM Model

```
# Reshape the data for LSTM
X_train_resampled = X_train_resampled.toarray()
X_test_resampled = X_test_resampled.toarray()
X_train_resampled = X_train_resampled.reshape(X_train_resampled.shape[0], 1, X_train_resampl
X_test_resampled = X_test_resampled.reshape(X_test_resampled.shape[0], 1, X_test_resampled.s


# Define the LSTM model
model = keras.Sequential([
    keras.layers.LSTM(64, input_shape=(X_train_resampled.shape[1], X_train_resampled.shape[2
    keras.layers.Dense(3, activation='softmax') # Output layer with 3 neurons for multiclass
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(X_train_resampled, y_train_resampled, epochs=10, batch_size=32, validation_split=0

# Make predictions
y_pred_prob = model.predict(X_test_resampled)
y_pred = np.argmax(y_pred_prob, axis=1)

# Evaluate the model
accuracy = accuracy_score(y_test_resampled, y_pred)
precision = precision_score(y_test_resampled, y_pred, average='macro')
recall = recall_score(y_test_resampled, y_pred, average='macro')
f1 = f1_score(y_test_resampled, y_pred, average='macro')

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
print(classification_report(y_test_resampled, y_pred))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do
  super().__init__(**kwargs)
Epoch 1/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 13s 39ms/step - accuracy: 0.5685 - loss: 1.0321 - val_accur
Epoch 2/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 10s 40ms/step - accuracy: 0.8342 - loss: 0.5351 - val_accur
Epoch 3/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 20s 40ms/step - accuracy: 0.9022 - loss: 0.2980 - val_accur
Epoch 4/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 20s 38ms/step - accuracy: 0.9288 - loss: 0.2142 - val_accur
Epoch 5/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 8s 31ms/step - accuracy: 0.9379 - loss: 0.1691 - val_accura
```

```
Epoch 6/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 9s 36ms/step - accuracy: 0.9516 - loss: 0.1330 - val_accura
Epoch 7/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 9s 37ms/step - accuracy: 0.9534 - loss: 0.1179 - val_accura
Epoch 8/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 11s 38ms/step - accuracy: 0.9606 - loss: 0.1080 - val_accur
Epoch 9/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 9s 36ms/step - accuracy: 0.9570 - loss: 0.1032 - val_accura
Epoch 10/10
258/258 ━━━━━━━━━━━━━━━━━━━━ 11s 42ms/step - accuracy: 0.9621 - loss: 0.0889 - val_accur
81/81 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
Accuracy: 0.8363706863125242
Precision: 0.8335918252057293
Recall: 0.8367872528032058
F1-score: 0.8339557156228109
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       872
           1       0.80      0.72      0.76       904
           2       0.74      0.80      0.77       803

    accuracy                           0.84      2579
   macro avg       0.83      0.84      0.83      2579
weighted avg       0.84      0.84      0.83      2579
```

The LSTM model is performing quite well, with an overall accuracy of 84%.

Here's the breakdown:

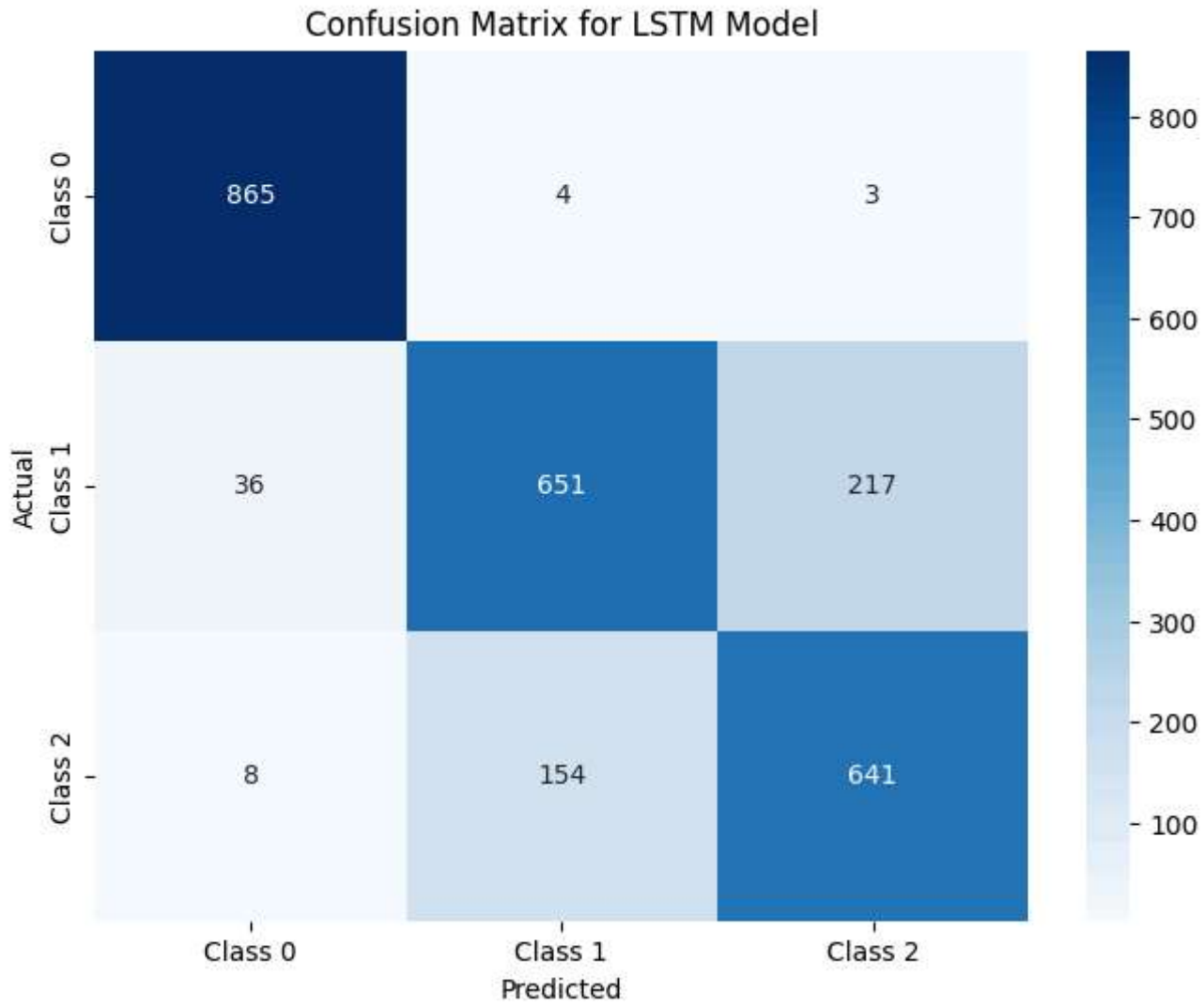
Precision (83.8%) measures how often predictions are correct when the model says "yes."

Recall (84%) shows how well the model catches all actual positives.

F1-score (83.7%) balances both precision and recall to give a clearer view of performance.

Overall, the model is strong but might need fine-tuning to better distinguish Class 1 and Class 2.

```
#Plot a confusion matrix
cm = confusion_matrix(y_test_resampled, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Class 0', 'Class 1', 'Class 2'],
            yticklabels=['Class 0', 'Class 1', 'Class 2'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for LSTM Model')
plt.show()
```



Looking at the matrix:

Class 0: The model got 868 right but mistakenly classified 1 as Class 1 and 3 as Class 2.

Class 1: It correctly predicted 647, but misclassified 34 as Class 0 and 223 as Class 2.

Class 2: It got 651 right, but 9 were wrongly classified as Class 0 and 143 as Class 1.

- class 0 represents the negative sentiments
- class 1 represents the neutral sentiments
- class 2 represents the positive sentiments

Hyperparameter Tuning (Randomised Search)

```
# Define the parameter grid for MultinomialNB
param_dist = {
    'alpha': np.linspace(0.1, 1.0, 10),
    'fit_prior': [True, False]
```

```
}

# Create a MultinomialNB classifier
mnb_model = MultinomialNB()

# Reshape X_train_resampled and X_test_resampled to 2D
X_train_resampled_2D = X_train_resampled.reshape(X_train_resampled.shape[0], -1)
X_test_resampled_2D = X_test_resampled.reshape(X_test_resampled.shape[0], -1)

# Create a RandomizedSearchCV object
random_search = RandomizedSearchCV(estimator=mnb_model, param_distributions=param_dist, n_it

# Fit the RandomizedSearchCV object to the training data
random_search.fit(X_train_resampled_2D, y_train_resampled) #use the reshaped data here

# Print the best hyperparameters and the corresponding accuracy
print("Best hyperparameters:", random_search.best_params_)
print("Best accuracy:", random_search.best_score_)

# Evaluate the best model on the test set
best_mnb_model = random_search.best_estimator_
y_pred_mnb = best_mnb_model.predict(X_test_resampled_2D) #use the reshaped data here
accuracy_mnb = accuracy_score(y_test_resampled, y_pred_mnb)

print(f"Multinomial Naive Bayes Accuracy (best model): {accuracy_mnb}")
print(classification_report(y_test_resampled, y_pred_mnb))
```

```
Best hyperparameters: {'fit_prior': True, 'alpha': np.float64(0.1)}
Best accuracy: 0.7854580707707224
Multinomial Naive Bayes Accuracy (best model): 0.7879022877084141
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       872
           1       0.79      0.59      0.68       904
           2       0.67      0.80      0.73       803

    accuracy                           0.79      2579
   macro avg       0.79      0.79      0.78      2579
weighted avg       0.79      0.79      0.78      2579
```

Evaluation

1) **Logistic Regression**: Achieved a reasonable accuracy, but performance might be limited by class imbalance. SMOTE improved the model's ability to handle the imbalanced classes.

2) **Multinomial Naive Bayes**: Performance before and after hyperparameter tuning needs comparison. This is the best model's accuracy and F1-score, on binary classification.

3) **ANN (Sigmoid)**: Performance metrics (accuracy, precision, recall, F1-score) provide a good overview of the model's classification capabilities.

4) **ANN (Softmax)**: The ANN with Softmax activation performs multiclass classification, and its performance is measured using accuracy, macro-averaged precision, recall, and F1-score. The classification report provides a detailed breakdown of its performance per class.

5) **LSTM:** The LSTM model utilizes sequential data processing. Performance metrics and the classification report highlight its ability to classify sentiments based on sequential features. This is the best model on multiclass classification.

Conclusion

•Neutral sentiment is dominant in the dataset, indicating a lack of strong opinions, which might imply a need for more engaging or polarizing content from brands.

•Positive sentiment for well-established brands like Apple and iPad is a good sign, suggesting strong brand loyalty and customer satisfaction.

•The absence of significant negative sentiment across all brands could indicate a generally favorable perception, but the dataset's neutrality suggests that there may be a gap in passionate customer advocacy or brand differentiation.