NLP Model For Twitter Sentiment Analysis on Google and Apple products

The goal of this project is to develop a Natural Language Processing (NLP) model that analyzes and rate sentiment in tweets depending on their content, related to Google and Apple products.The dataset sourced from Kaggle, contains tweets labeled as positive, negative and Neutral

Why Sentiment Analysis Matters ¶

In today's digital landscape,sentiment analysis plays a vital role in understanding public perception. Companies use this method to gain insightful information form massive volumes of textual data such as social media comments,news stories and customer evaluations. We can improve decision-making and business intelligence by implementing NLP algorithms like Support Vector Machines(SVM) and Naive bayes.

The main objective of this project is to :

1. Preprocess Twitter data(tokenization, cleaning and vectorization)
2. Apply machine learning models for sentiment classification
3. Asses model performance using important metrics

Our goal is to create an effective sentiment analysis model by the project's conclusion that offers insightful information about consumer perception of Apple and Google products

Summary

Dataset Overview

This dataset contains information about tweets, including their date, classification, and other relevant features. For this task, I extracted only two columns:

- **Feature Variable (Text/Tweets)**: The actual tweet content.
- **Sentiment**: The rating of each tweet, categorized as negative, positive, or neutral.

Data Preparation

The data preparation phase included **data cleaning** and **data preprocessing** using the pandas and NLTK libraries.

1. Data Cleaning (Using Pandas)

- The dataset had **no missing values**, so no imputation was required.
- **Removed duplicate tweets**—for instance, one tweet appeared 304 times.
- Dropped irrelevant sentiment labels**, such as 'not_relevant', to retain only the core sentiments **(negative, positive, and neutral)**.
- **Converted sentiments to integer values** to ensure compatibility with machine learning models.

2. Data Preprocessing (Using NLTK)

- **Removed hyperlinks, usernames, single-character words, and hashtags** (including their values) using **regular expressions**, as they do not meaningfully contribute to sentiment analysis.
- **Eliminated stopwords and punctuation ** using the **NLTK corpus library** for stopwords and the **string library** for punctuation, as these do not add significant meaning to sentences.
- **Applied lemmatization ** using the **WordNet Lemmatizer**, converting words to their root form (e.g., "running" → "run").

Data Visualization

To explore and understand the dataset, I used:

- **Seaborn's** countplot to visualize the distribution of target sentiment classes.
- **WordCloud** to generate a visual representation of the most common words in the dataset.

Modeling

- Used **Scikit-learn's model_selection** library to split the dataset into training and testing sets.
- Implemented **pipelines** to streamline **vectorization, SMOTE (Synthetic Minority Over-sampling Technique), and classification models**.
- Applied **TF-IDF Vectorizer** to convert text data into numerical representations.

- Used **SMOTE** to address **class imbalance**, as one sentiment category comprised **more than 50%** of the dataset.
- Evaluated various **machine learning algorithms** for classification to determine the best-performing model.
- Created **custom functions** to automate repetitive processes such as model fitting and prediction.

Evaluation Metrics

To assess model performance, I used the following evaluation metrics:

- **Accuracy Score**: The proportion of correctly classified instances out of the total instances.
- **Precision Score**: The ratio of correctly predicted positive instances to total predicted positive instances.
- **Recall Score**: The ratio of correctly predicted positive instances to actual positive instances in the dataset.
- **ROC Curve**: A graphical representation of the true positive rate versus the false positive rate.

Model Evaluation & Predictions

- The **test set** obtained from `train_test_split (X_test, y_test)` was used for model evaluation and making predictions.
- Additionally, I developed a **custom function** that allows classification of sentiment based on user input.

Step 1: Importing libraries

In [68]:

```
#Basic Python Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Natural language processing libraries
from nltk.corpus import stopwords
import string
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer,word_tokenize
import re
from nltk import FreqDist
from wordcloud import WordCloud
#scikit-learn
from sklearn.model_selection import train_test_split,RandomizedSearchCV,GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
from sklearn.preprocessing import label_binarize
#Evaluation metrics
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,roc_curve,auc,r2_score,f1_sc
#imbalanced-learn
from imblearn.pipeline import Pipeline  # Use imbalanced-learn's Pipeline
from imblearn.over_sampling import SMOTE
```

Step 2: Understanding the dataset

In [2]:
```
#Read the data from the csv file as a dataframe and dispaly the first five rows
data = pd.read_csv('data/Apple-Twitter-Sentiment-DFE.csv',encoding='latin-1')
data.head()
```

Out[2]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	sentiment	sentiment:confidence	date	id	query	s
0	623495513	True	golden	10	NaN	3	0.6264	Mon Dec 01 19:30:03 +0000 2014	5.400000e+17	#AAPL OR @Apple	
1	623495514	True	golden	12	NaN	3	0.8129	Mon Dec 01 19:43:51 +0000 2014	5.400000e+17	#AAPL OR @Apple	
2	623495515	True	golden	10	NaN	3	1.0000	Mon Dec 01 19:50:28 +0000 2014	5.400000e+17	#AAPL OR @Apple	
3	623495516	True	golden	17	NaN	3	0.5848	Mon Dec 01 20:26:34 +0000 2014	5.400000e+17	#AAPL OR @Apple	
4	623495517	False	finalized	3	12/12/14 12:14	3	0.6474	Mon Dec 01 20:29:33 +0000 2014	5.400000e+17	#AAPL OR @Apple	

For this project we will use the text column as the feature variable and sentiment column as the target variable

In [3]:
```
# extracting the text and sentiment column and previewing first five rows
data = data[['text','sentiment']]
data.head()
```

Out[3]:

	text	sentiment
0	#AAPL:The 10 best Steve Jobs emails ever...htt...	3
1	RT @JPDesloges: Why AAPL Stock Had a Mini-Flas...	3
2	My cat only chews @apple cords. Such an #Apple...	3
3	I agree with @jimcramer that the #IndividualIn...	3
4	Nobody expects the Spanish Inquisition #AAPL	3

In [4]:
```
#Check the number of records and features using the shape method
data.shape
print(f'This dataset contains {data.shape[0]} rows')
print(f'This dataset contains {data.shape[1]} columns')
```

```
This dataset contains 3886 rows
This dataset contains 2 columns
```

In [5]:
```
#Access information about our dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3886 entries, 0 to 3885
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   text       3886 non-null   object
 1   sentiment  3886 non-null   object
dtypes: object(2)
memory usage: 60.8+ KB
```

Both our columns have non-null values, there are no missing data points ensuring our dataset is ready for futher processing and model

In [6]:
```
# conforming that there are no null values
data.isna().sum()
```

Out[6]:
```
text         0
sentiment    0
dtype: int64
```

In [7]:
```
#get summary statistics of our data
data.describe()
```

Out[7]:

	text	sentiment
count	3886	3886
unique	3219	4
top	RT @OneRepublic: Studio at 45,000 ft. One out...	3
freq	304	2162

```
we see a tweet that appears 304 times we may need to drop duplicates
sentiment has 4 categories with 3 (Neutral) taking up 55.64% this created an imbalance in the dataset which we will need to handle.There is 4 sentiment values instead of the expected 3 we may need to check out the exra one
```

step 3: Data cleaning

In [8]:
```
# check for and remove duplicates
data[data.duplicated()]
```

Out[8]:

	text	sentiment
32	RT @thehill: Justice Department cites 18th cen...	3
34	RT @thehill: Justice Department cites 18th cen...	3
38	RT @thehill: Justice Department cites 18th cen...	3
42	RT @thehill: Justice Department cites 18th cen...	3
45	RT @thehill: Justice Department cites 18th cen...	3
...	...	...
3846	RT @TeamCavuto: Protesters stage #DieIn protes...	3
3852	RT @TeamCavuto: Protesters stage #DieIn protes...	3
3855	RT @Ecofantasy: Thinking of upgrading to #Yose...	1
3878	RT @shannonmmiller: Love the @Apple is support...	5
3885	RT @SwiftKey: We're so excited to be named to ...	5

643 rows × 2 columns

In [9]:
```
#Drop the above duplicates
data.drop_duplicates(inplace=True)
```

In [10]:
```
#check sentiment labels
data['sentiment'].value_counts()
```

Out[10]:
```
3               1681
1               1102
5                379
not_relevant      81
Name: sentiment, dtype: int64
```

In [11]:
```
#drop the not_relevant label
data = data.query('~(sentiment=="not_relevant")') #data = data.query('sentiment !="not_relevant"')
#confirm sentiment 'not_relevant' has been removed
data['sentiment'].value_counts()
```

Out[11]:
```
3    1681
1    1102
5     379
Name: sentiment, dtype: int64
```

In [12]:
```
#convert sentiment column to int
data['sentiment'] = data['sentiment'].astype(int)
#confirm removal using info method
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3162 entries, 0 to 3884
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   text       3162 non-null   object
 1   sentiment  3162 non-null   int32 
dtypes: int32(1), object(1)
memory usage: 61.8+ KB
```

Step 3:Preprocess text

In [13]:
```
def clean_text(text):
    stopword_list = stopwords.words('english')
    stopword_list += string.punctuation
    lemmatizer = WordNetLemmatizer()

    #remove hyperlinks,usermames,words with 1 character,hashtags and their values
    text =re.sub(r"https?:[^\s]+|@[\S]+|\b\w\b|\#\w+|\.\.+", "", text)

    # Tokenize the text
    text = word_tokenize(text)

    # Lowercase the text and remove stopwords
    text = [word.lower() for word in text if word.lower() not in stopword_list]

     #Lemmatize
    text = [lemmatizer.lemmatize(word)for word in text]

    # join processed text as a single string
    text = ' '.join(text)
    return text
```

Here we are performing some pre-processing on the data befoee converting it into vectors and passing it to the machine learning model.

we create a function that iterates through each record,and does the following

1. using regular expression
   - we get rid of hyperlinks in the data that was showing the link where the tweet came from,
     - we also remove '@username' since the username of the person who tweeted doesn't help in rating a sentiment
     - Remove twitter hashtags together with the value since they do not contribute to a sentiment meaningfully
     - Remove single character words
2. Remove stop words such as 'i','me' which do not add much value and convert them to lowercase so that eg word bad and BAd are treated as the same thing
3. Lemmatize words to reduce words to their base so that word like run and running are treated as the same thing

In [14]:
```
#Apply the function to the text column
data['cleaned_text'] = data['text'].apply(clean_text)
```

In [15]:
```
### confirm data procesed
data[['text','cleaned_text']].head()
```

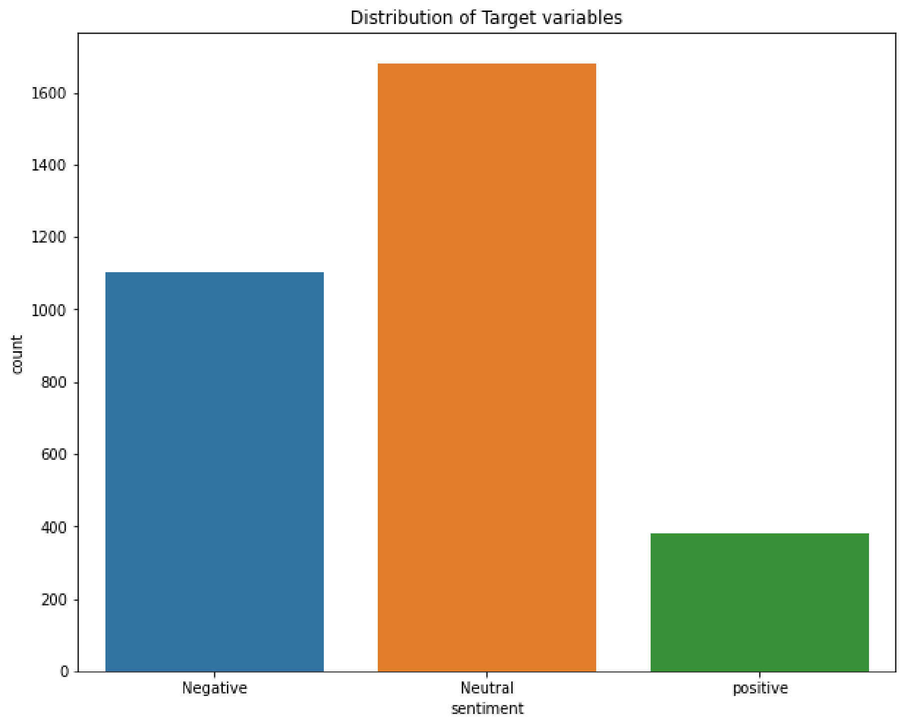
Out[15]:

	text	cleaned_text
0	#AAPL:The 10 best Steve Jobs emails ever...htt...	10 best steve job email ever
1	RT @JPDesloges: Why AAPL Stock Had a Mini-Flas...	rt aapl stock mini-flash crash today aapl
2	My cat only chews @apple cords. Such an #Apple...	cat chew cord
3	I agree with @jimcramer that the #IndividualIn...	agree trade extended today pullback good see
4	Nobody expects the Spanish Inquisition #AAPL	nobody expects spanish inquisition

Step5: EDA

Data Visualization of Target variables

In [16]:
```
fig, ax = plt.subplots(figsize=(10,8))
ax= sns.countplot(data = data, x='sentiment')
#using xtickslabels to label values 1 as negative,3 as neutral and 5 as positive
ax.set(title='Distribution of Target variables',xticklabels=['Negative','Neutral','positive']);
```




```
The label data looks imbalanced where neutral takes more than 50 % of the values and positive values very less(11%)
```

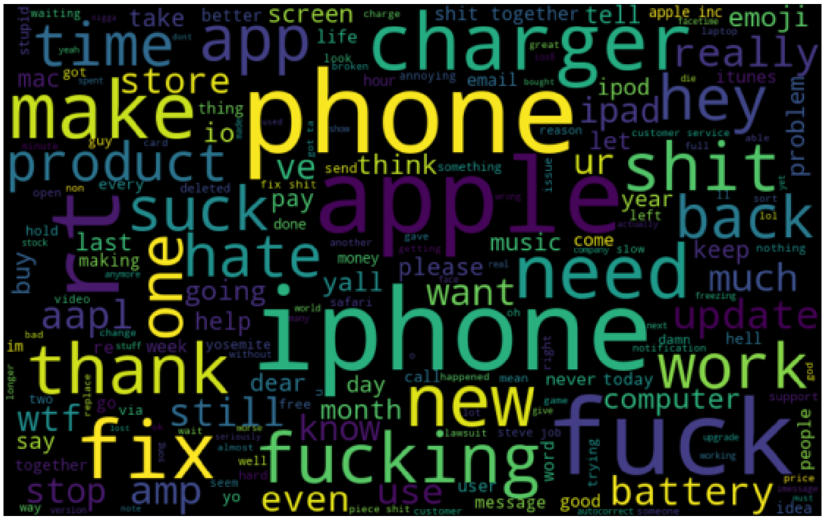
Check most common words in the dataset using word cloud

visualization where frequent words appear enlarged as compared to less frequent words

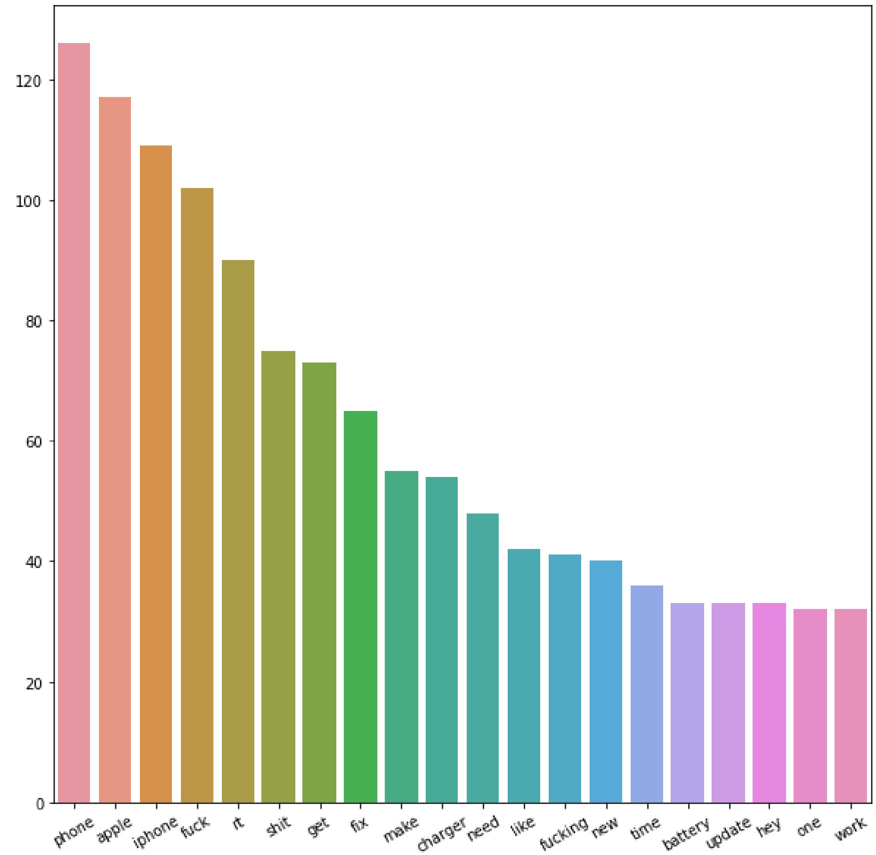
common words for the negative sentiments

In [17]:

```
#Negative words
negative_words = ' '.join(data.query("sentiment==1")['cleaned_text'])
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(negative_words)

plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off');
```



In [18]:
```python
## Creating FreqDist for whole BoW , keeping the 20 most common tokens
negative_tokens = word_tokenize(negative_words) #tokenize data
all_fdist = FreqDist(negative_tokens).most_common(20)

## Conversion to Pandas series via Python Dictionary for easier plotting
all_fdist = pd.Series(dict(all_fdist))

## Setting figure, ax into variables
fig, ax = plt.subplots(figsize=(10,10))

## Seaborn plotting using Pandas attributes + xtick rotation for ease of viewing
all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
plt.xticks(rotation=30);
```



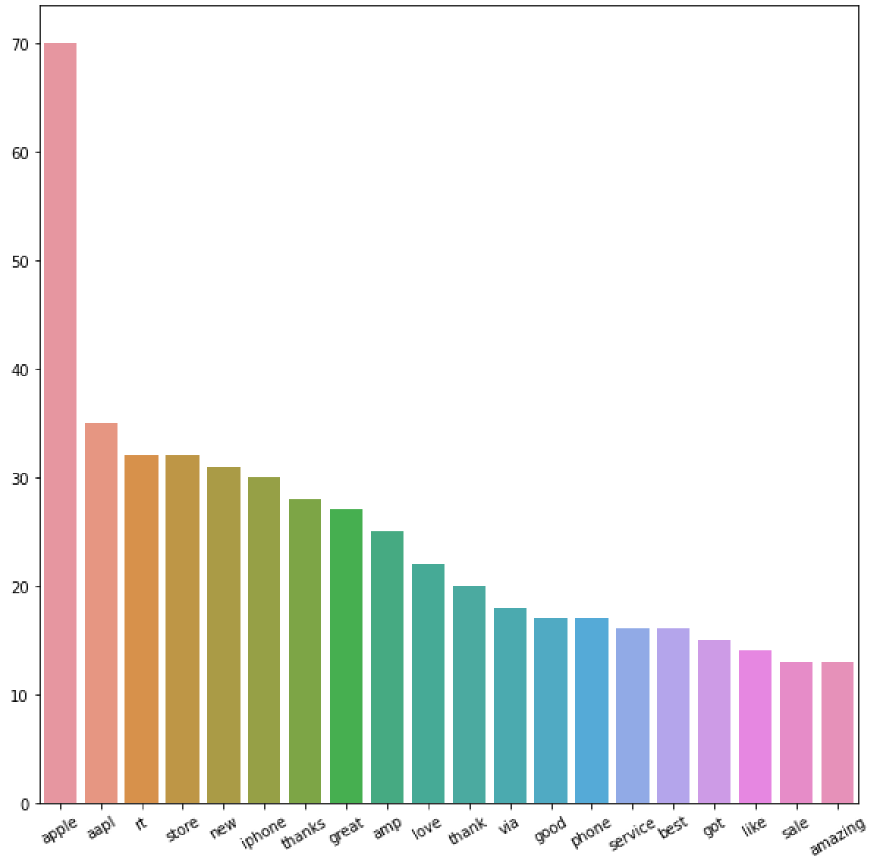
common words for the positive sentiments

In [19]:

```
#positive words
positive_words = ' '.join(data.query('sentiment==5')['cleaned_text'])
wordcloud = WordCloud(width=800, height=500, random_state=21 ,max_font_size=110).generate(positive_words)

plt.figure(figsize=(10,7))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off');
```

In [20]:

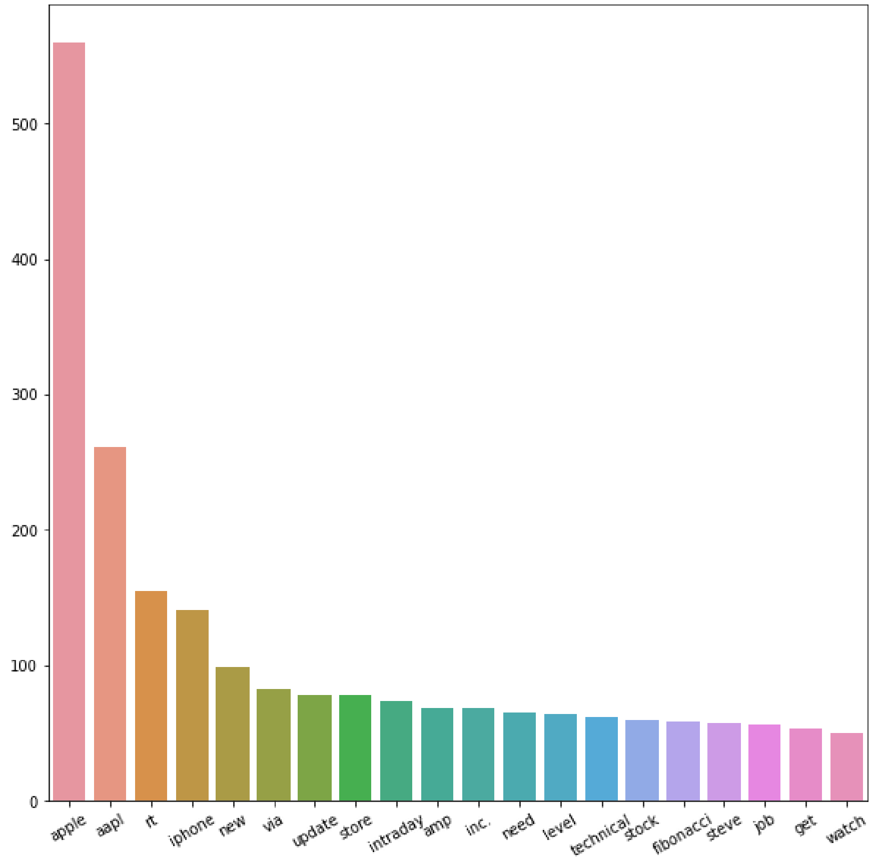
```python
## Creating FreqDist for whole BoW , keeping the 20 most common tokens
positive_tokens = word_tokenize(positive_words)
all_fdist = FreqDist(positive_tokens).most_common(20)

## Conversion to Pandas series via Python Dictionary for easier plotting
all_fdist = pd.Series(dict(all_fdist))

## Setting figure, ax into variables
fig, ax = plt.subplots(figsize=(10,10))

## Seaborn plotting using Pandas attributes + xtick rotation for ease of viewing
all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
plt.xticks(rotation=30);
```



In [21]:

```
#neutral
neutral_words = ' '.join(data.query("sentiment==3")['cleaned_text'])
from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(neutral_words)

plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

In [22]:
```
## Creating FreqDist for whole BoW , keeping the 20 most common tokens
neutral_tokens = word_tokenize(neutral_words)
all_fdist = FreqDist(neutral_tokens).most_common(20)

## Conversion to Pandas series via Python Dictionary for easier plotting
all_fdist = pd.Series(dict(all_fdist))

## Setting figure, ax into variables
fig, ax = plt.subplots(figsize=(10,10))

## Seaborn plotting using Pandas attributes + xtick rotation for ease of viewing
all_plot = sns.barplot(x=all_fdist.index, y=all_fdist.values, ax=ax)
plt.xticks(rotation=30);
```



step 6: Modelling

In this section i will conduct a train_test split for modelling then use pipelines to streamline vectorization, smote and modelling process. I will then evaluate the models using accuray_score,precision,recall,f1 score and roc_auc The idea behind choosing these models is that we want to run all the classifiers on the dataset ranging from simple ones to complex models, and then try to find the one which gives the best performance among them and tune it

In [23]:
```
#split data into x and y
X = data['cleaned_text']
y = data['sentiment']

#split data into a training and test set
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,stratify=y,shuffle=True,random_state=42)
```

We select our feature variable as X and target variable as y. Then split our data into train and test data for modelling training and evaluation purposes.Shuffle and stratify ensures a well-balanced dataset split,which improves model performance and evaluation reliability

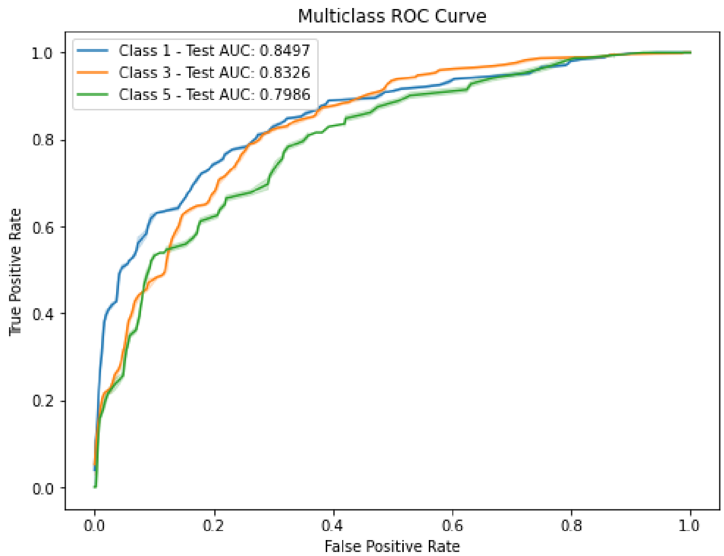
In [24]:
```
#create pipeline to streamline vectorization and for modelling, start with base model
#(Logistic Regression)
pipeline = Pipeline([
    ('tdif',TfidfVectorizer(ngram_range=(1,2))),
    ('smote',SMOTE()),
    ('model',LogisticRegression())
])
```

Pipeline that does text vectorization i.e converting raw text into numerical features using tfidfVectorizer,smote to balance the target class 'sentiment' and using Logistic regression to classify text

In [25]:
```python
#Function to train the model ,make predictions and calculate evaluation metrics
def modelling(pipe):
    pipe.fit(X_train, y_train)

    # Predict train and test data
    y_hat_train = pipe.predict(X_train)
    y_hat_test = pipe.predict(X_test)

    # Get accuracy, precision, recall, and F1-score
    base_train_accuracy = accuracy_score(y_train, y_hat_train)
    base_test_accuracy = accuracy_score(y_test, y_hat_test)
    base_train_precision = precision_score(y_train, y_hat_train, average='weighted')
    base_test_precision = precision_score(y_test, y_hat_test, average='weighted')
    base_train_recall = recall_score(y_train, y_hat_train, average='weighted')
    base_test_recall = recall_score(y_test, y_hat_test, average='weighted')
    base_train_f1 = f1_score(y_train, y_hat_train, average='weighted')
    base_test_f1 = f1_score(y_test, y_hat_test, average='weighted')

    # Binarize labels for multiclass ROC curve
    classes = sorted(set(y_train))  # Get unique classes
    y_train_bin = label_binarize(y_train, classes=classes)
    y_test_bin = label_binarize(y_test, classes=classes)

    # Get prediction scores
    if hasattr(pipe, "decision_function"):
        y_score_train = pipe.decision_function(X_train)
        y_score_test = pipe.decision_function(X_test)
    else:
        y_score_train = pipe.predict_proba(X_train)
        y_score_test = pipe.predict_proba(X_test)

    # Compute ROC curve and AUC for each class
    train_auc_list, test_auc_list = [], []
    plt.figure(figsize=(8, 6))

    for i in range(len(classes)):
        train_fpr, train_tpr, _ = roc_curve(y_train_bin[:, i], y_score_train[:, i])
        test_fpr, test_tpr, _ = roc_curve(y_test_bin[:, i], y_score_test[:, i])

        train_auc = auc(train_fpr, train_tpr)
        test_auc = auc(test_fpr, test_tpr)

        train_auc_list.append(train_auc)
        test_auc_list.append(test_auc)

        sns.lineplot(x=test_fpr, y=test_tpr, label=f'Class {classes[i]} - Test AUC: {test_auc:.4f}')

    # Average AUC
    avg_train_auc = sum(train_auc_list) / len(train_auc_list)
    avg_test_auc = sum(test_auc_list) / len(test_auc_list)

    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Multiclass ROC Curve")
    plt.legend()
    plt.show()

    return {
        'Training Accuracy': base_train_accuracy,
        'Test Accuracy': base_test_accuracy,
#         'Training precision': base_train_precision,
        'Test precision': base_test_precision,
#         'Training recall': base_train_recall,
        'Test recall': base_test_recall,
#         'Training f1_score': base_train_f1,
        'Test f1_score': base_test_f1,
        'Average Train AUC': avg_train_auc,
        'Average Test AUC': avg_test_auc
    }

```

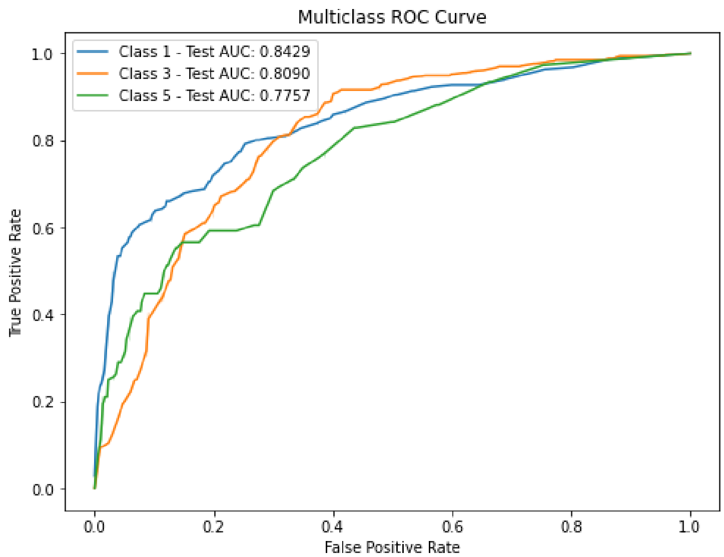
To avoid repetition for various models that we will be creating, we've defined a function that fits, predicts and calculates the accuracy_score

In [26]:
```
#Logistic Regression evaluation
logreg = modelling(pipeline)
logreg
```



Out[26]:
```
{'Training Accuracy': 0.9636219849742982,
 'Test Accuracy': 0.707740916271722,
 'Test precision': 0.7037161065532846,
 'Test recall': 0.707740916271722,
 'Test f1_score': 0.7054753826807029,
 'Average Train AUC': 0.9932681573749033,
 'Average Test AUC': 0.8269461973387789}
```

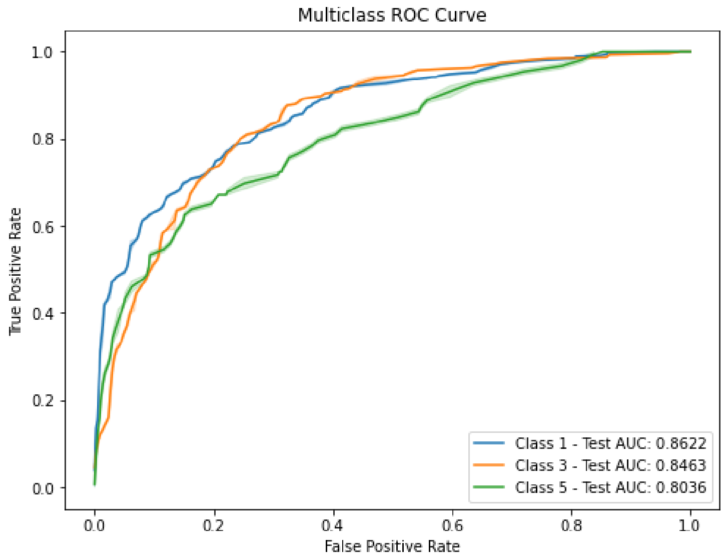
In [27]:
```
#randomForest
pipeline.set_params(model=RandomForestClassifier(random_state=42))
rdf = modelling(pipeline)
rdf
```



Out[27]:
```
{'Training Accuracy': 0.9905100830367735,
 'Test Accuracy': 0.7045813586097947,
 'Test precision': 0.7196297646534613,
 'Test recall': 0.7045813586097947,
 'Test f1_score': 0.681570616869526,
 'Average Train AUC': 0.9995458299107819,
 'Average Test AUC': 0.8091915324598483}
```

In [28]:

```
#multinomial naive bayes
pipeline.set_params(model = MultinomialNB())
nb =  modelling(pipeline)
nb
```

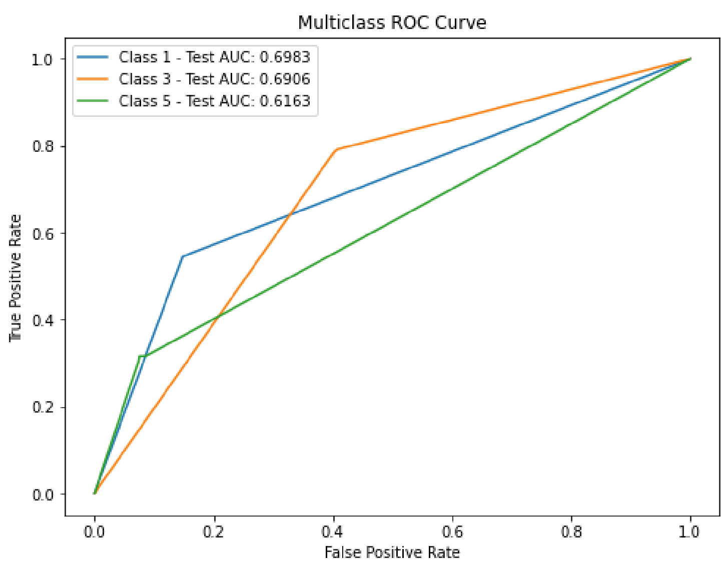


Out[28]:

```
{'Training Accuracy': 0.9418742586002372,
 'Test Accuracy': 0.6824644549763034,
 'Test precision': 0.7133060787791613,
 'Test recall': 0.6824644549763034,
 'Test f1_score': 0.6913711284026903,
 'Average Train AUC': 0.9889344354190913,
 'Average Test AUC': 0.8373466293718987}
```

In [29]:

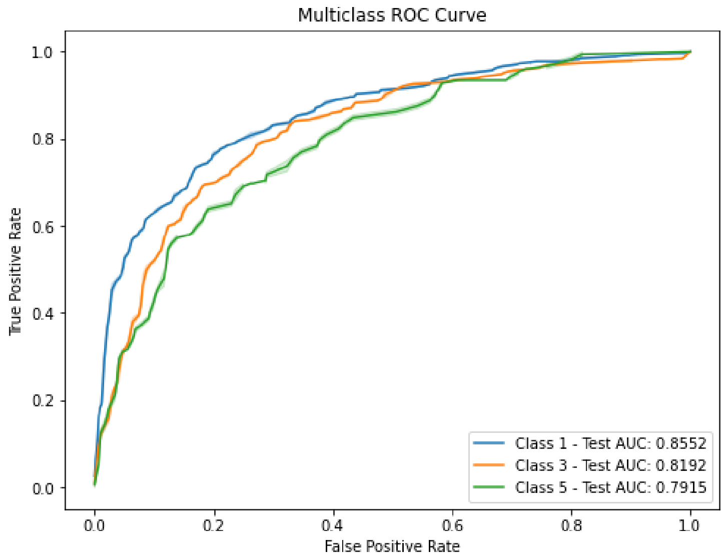
```
#Decision tree
pipeline.set_params(model = DecisionTreeClassifier(random_state=42))
dt = modelling(pipeline)
dt
```



Out[29]:

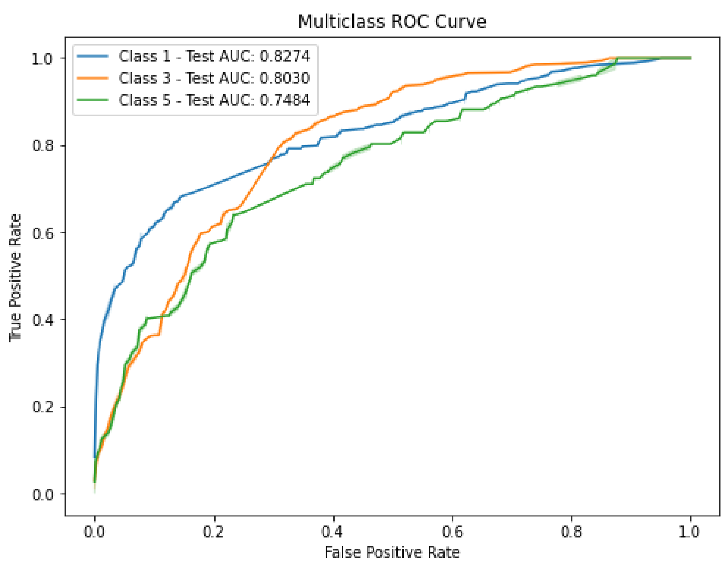
```
{'Training Accuracy': 0.9905100830367735,
 'Test Accuracy': 0.6477093206951027,
 'Test precision': 0.6397823451988728,
 'Test recall': 0.6477093206951027,
 'Test f1_score': 0.6397023128888808,
 'Average Train AUC': 0.999876025147195,
 'Average Test AUC': 0.6684032590271273}
```

In [30]:
```
#svc
pipeline.set_params(model = svm.SVC())
svm = modelling(pipeline)
svm
```



Out[30]:
```
{'Training Accuracy': 0.9806247528667458,
 'Test Accuracy': 0.7061611374407583,
 'Test precision': 0.6933178216904188,
 'Test recall': 0.7061611374407583,
 'Test f1_score': 0.6850618166156861,
 'Average Train AUC': 0.9953301054763962,
 'Average Test AUC': 0.8219852591406278}
```

In [31]:
```
pipeline.set_params(model = XGBClassifier())
xgb = modelling(pipeline)
xgb
```



Out[31]:
```
{'Training Accuracy': 0.8912613681296956,
 'Test Accuracy': 0.6919431279620853,
 'Test precision': 0.7020677275931184,
 'Test recall': 0.6919431279620853,
 'Test f1_score': 0.6728253975054899,
 'Average Train AUC': 0.9733294583555869,
 'Average Test AUC': 0.7929194002687371}
```

In [32]:
```
# Dictionary of model results
model_results = {
    "Logistic Regression": logreg,
    "Random Forest": rdf,
    "Naïve Bayes":nb,
    "Decision Tree": dt,
    "SVM": svm,
    "XGBoost": xgb,
}

# Convert dictionary to DataFrame
df_results = pd.DataFrame.from_dict(model_results, orient='index')

# Display the DataFrame
df_results

```

Out[32]:

	Training Accuracy	Test Accuracy	Test precision	Test recall	Test f1_score	Average Train AUC	Average Test AUC
Logistic Regression	0.963622	0.707741	0.703716	0.707741	0.705475	0.993268	0.826946
Random Forest	0.990510	0.704581	0.719630	0.704581	0.681571	0.999546	0.809192
Naïve Bayes	0.941874	0.682464	0.713306	0.682464	0.691371	0.988934	0.837347
Decision Tree	0.990510	0.647709	0.639782	0.647709	0.639702	0.999876	0.668403
SVM	0.980625	0.706161	0.693318	0.706161	0.685062	0.995330	0.821985
XGBoost	0.891261	0.691943	0.702068	0.691943	0.672825	0.973329	0.792919

In [33]:
```
#checking gaps between train and test accuracy
df_results['Training Accuracy'] - df_results['Test Accuracy']
```

Out[33]:
```
Logistic Regression    0.255881
Random Forest          0.285929
Naïve Bayes            0.259410
Decision Tree          0.342801
SVM                    0.274464
XGBoost                0.199318
dtype: float64
```

```
From the models tested we see varying degrees of overfitting and generalization ability across different models.Decision trees and random Forest performance might fail in real word due to overfitting

Higher AUC is seen in the models indicating models performs well in separating the classes
XGBoost has the best generalization as it has the smallest gap between the train and test accuracy

on simple models logistic regression performs well with a good balance between precision and recall

I will compare XGBoost and Logistic regression by tuning to determine the best model
```

Step:7 Hyperparameter tuning

We'll tune the following parameters: selecting optimal parameters for the model

- **n_estimators**: Number of boosting rounds
- **max_depth**: maximum tree depth (higher=more complex model)
- **learning_rate**: Step size to prevent overfitting
- **subsmaple**: Fraction of samples used per tree(lower=more regularizationO
- **colsample_bytree**: fraction of features used per tree
- **gamma**: minimum loss reduction reqquired for further splitting

In [34]:
```
#Define parameter grid
param_grid = {
    'model__n_estimators': [100, 300, 500],
    'model__max_depth': [3, 5, 7],
    'model__learning_rate': [ 0.1, 0.2,0.4],
    'model__subsample': [0.7, 0.8, 1.0],
    'model__colsample_bytree': [0.7, 0.8, 1.0],
    'model__gamma': [0, 0.1, 0.2]
}
```

In [35]:
```
#check pipeline
pipeline
```

Out[35]:
```
Pipeline
TfidfVectorizer
SMOTE
XGBClassifier
```

In [36]:
```
grid_search = GridSearchCV(
                            estimator = pipeline,
                            param_grid = param_grid,
                            scoring = 'accuracy',
                            verbose=2,
                            cv = 5,
                            n_jobs=-1
)
grid_search.fit(X_train, y_train)

#Get Best parameters
grid_search.best_params_

```

```
Fitting 5 folds for each of 729 candidates, totalling 3645 fits
```

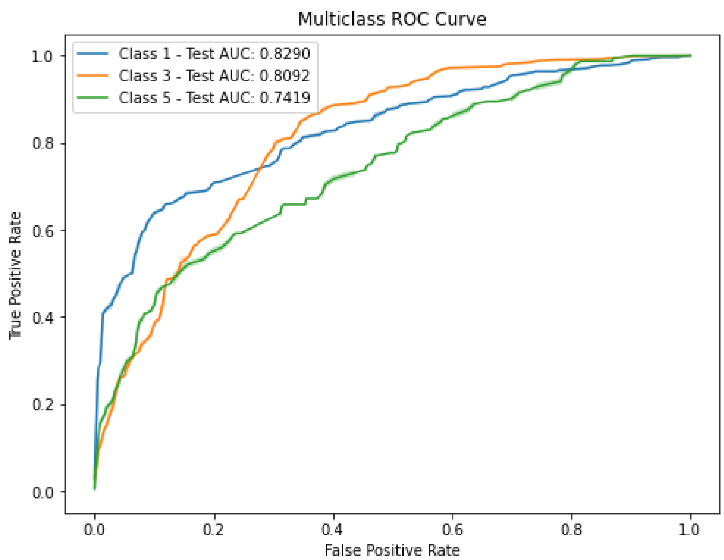
Out[36]:
```
{'model__colsample_bytree': 0.7,
 'model__gamma': 0,
 'model__learning_rate': 0.2,
 'model__max_depth': 3,
 'model__n_estimators': 500,
 'model__subsample': 1.0}
```

Retraining the model with the best parameters

In [51]:
```
pipeline.set_params(model=XGBClassifier(
                            learning_rate= 0.2,
                            max_depth= 3,
                            n_estimators= 500,
                            colsample_bytree =  0.7,
                            subsample = 1.0
))
```

Out[51]:
```
Pipeline
TfidfVectorizer
SMOTE
XGBClassifier
```

In [52]:
```python
#Evaluate the model with the best parameters
xgb_tuned = modelling(pipeline)
xgb_tuned
```



Out[52]:
```
{'Training Accuracy': 0.9161724001581653,
 'Test Accuracy': 0.7014218009478673,
 'Test precision': 0.7031791809042994,
 'Test recall': 0.7014218009478673,
 'Test f1_score': 0.6826564554491085,
 'Average Train AUC': 0.9858007103440465,
 'Average Test AUC': 0.7933539582783885}
```

In [53]:
```python
#comparing with the results before the tuning
df_results
```

Out[53]:

	Training Accuracy	Test Accuracy	Test precision	Test recall	Test f1_score	Average Train AUC	Average Test AUC
Logistic Regression	0.963622	0.707741	0.703716	0.707741	0.705475	0.993268	0.826946
Random Forest	0.990510	0.704581	0.719630	0.704581	0.681571	0.999546	0.809192
Naïve Bayes	0.941874	0.682464	0.713306	0.682464	0.691371	0.988934	0.837347
Decision Tree	0.990510	0.647709	0.639782	0.647709	0.639702	0.999876	0.668403
SVM	0.980625	0.706161	0.693318	0.706161	0.685062	0.995330	0.821985
XGBoost	0.891261	0.691943	0.702068	0.691943	0.672825	0.973329	0.792919

In [54]:
```python
xgb_tuned['Average Test AUC'] - df_results['Average Test AUC']
```

Out[54]:
```
Logistic Regression   -0.033592
Random Forest         -0.015838
Naïve Bayes           -0.043993
Decision Tree          0.124951
SVM                   -0.028631
XGBoost                0.000435
Name: Average Test AUC, dtype: float64
```

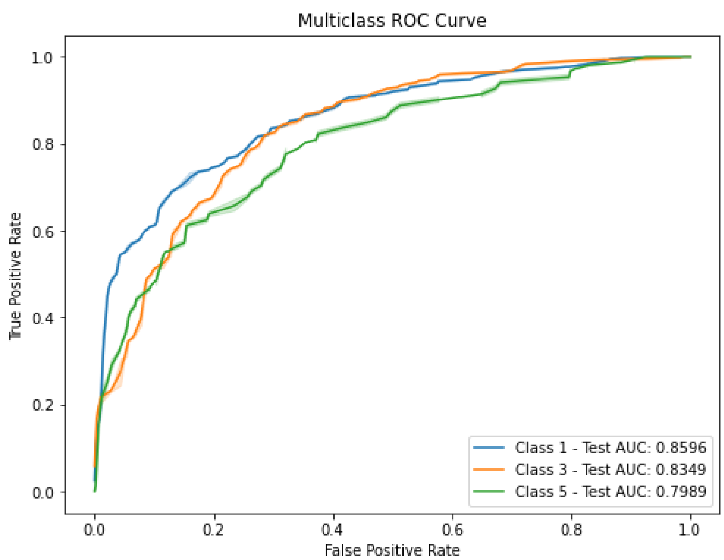
```
No major improvement in accuracy - The test accuracy and F1 score increased by 1% or less
AUC improved slightly - The Test AUC actually increased by just  0.0004
Tuning reduced slight overfitting -  meaning the model is generalizing slightly better.
```

Tune Logistic Regression

In [56]:
```
pipeline.set_params(model = LogisticRegression())
param_grid = {
    'model__C': [0.01, 0.1, 1, 10, 100],  # Regularization strength
    'model__penalty': ['l1', 'l2'],  # Type of regularization
    'model__solver': ['liblinear']  # Required for L1 penalty
}

grid = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy',n_jobs=-1)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best accuracy Score:", grid.best_score_)
```

```
Best Parameters: {'model__C': 10, 'model__penalty': 'l2', 'model__solver': 'liblinear'}
Best accuracy Score: 0.7018667084099715
```

In [57]:
```
pipeline.set_params(model = LogisticRegression(
                                              C= 10,
                                             penalty = 'l2',
                                             solver = 'liblinear'
))
logistic_tuned= modelling(pipeline)
logistic_tuned
```



Out[57]:
```
{'Training Accuracy': 0.9837880585211546,
 'Test Accuracy': 0.7251184834123223,
 'Test precision': 0.7219159277378586,
 'Test recall': 0.7251184834123223,
 'Test f1_score': 0.7233076101045702,
 'Average Train AUC': 0.9985778434373468,
 'Average Test AUC': 0.8311354483998646}
```

In [58]:
```
#compare with previous results
df_results
```

Out[58]:

	Training Accuracy	Test Accuracy	Test precision	Test recall	Test f1_score	Average Train AUC	Average Test AUC
Logistic Regression	0.963622	0.707741	0.703716	0.707741	0.705475	0.993268	0.826946
Random Forest	0.990510	0.704581	0.719630	0.704581	0.681571	0.999546	0.809192
Naïve Bayes	0.941874	0.682464	0.713306	0.682464	0.691371	0.988934	0.837347
Decision Tree	0.990510	0.647709	0.639782	0.647709	0.639702	0.999876	0.668403
SVM	0.980625	0.706161	0.693318	0.706161	0.685062	0.995330	0.821985
XGBoost	0.891261	0.691943	0.702068	0.691943	0.672825	0.973329	0.792919

There is an increase by 2% in accuracy, precision, recall and F1 score.

lets compare XG Boost and logistic regression

In [61]:
```
model_results = {
    "Logistic Regression_tuned": logistic_tuned,
    "XGBoost_tuned": xgb_tuned,
}

# Convert dictionary to DataFrame
df_results_tuned = pd.DataFrame.from_dict(model_results, orient='index')

# Display the DataFrame
df_results_tuned
```

Out[61]:

	Training Accuracy	Test Accuracy	Test precision	Test recall	Test f1_score	Average Train AUC	Average Test AUC
Logistic Regression_tuned	0.983788	0.725118	0.721916	0.725118	0.723308	0.998578	0.831135
XGBoost_tuned	0.916172	0.701422	0.703179	0.701422	0.682656	0.985801	0.793354

Logistic regression performs much better than XG Boost.We will logistic regression model to make our predictions

Step8: Make predictions

In [63]:
```
#create function to predict sentiment
pipeline.set_params(model =  LogisticRegression(
                                             C= 10,
                                            penalty = 'l2',
                                            solver = 'liblinear'))
pipeline.fit(X_train,y_train)
def sentiment_check(tweet):
    tweet_processed = clean_text(tweet)
    print(f'tweet:{tweet_processed}')
    prediction = pipeline.predict([tweet_processed])
    return "Negative Statement" if prediction[0] == 1 else "Neutral Statemnet" if prediction[0]==3 else "Posi

print(sentiment_check('this phones suck,its not what is marketed'))
```

```
tweet:phone suck marketed
Negative Statement
```

In [64]:
```
print(sentiment_check("I bought a new phone and it's so good"))
```

```
tweet:bought new phone good
Positive statement
```

In [65]:
```
print(sentiment_check("great item, good job"))
```

```
tweet:great item good job
Positive statement
```

In [66]:
```
print(sentiment_check("what a stupid cover"))
```

```
tweet:stupid cover
Negative Statement
```

Our model performs relatively well

we can look into using deep learning and other vectorization techniques to see if we can improve our models performance

Step 9: Conclusions and recommendations

Conclusion

- Logistic Regression outperformed XGBoost, achieving better accuracy and F1 score, making it the preferred model for sentiment classification.
- Hyperparameter tuning for XGBoost led to only marginal improvements (accuracy & F1 score increased by ≤1%, AUC by 0.0004).
- Tuning helped reduce overfitting slightly, meaning XGBoost generalized better than before, but still did not surpass Logistic Regression.

Recommendations

- Use Logistic Regression for final predictions since it performs better than XGBoost.
- Consider feature engineering (e.g., word embeddings like Word2Vec or BERT) to improve model performance further.
- Explore deep learning models (e.g., LSTMs or Transformers) if higher accuracy is required.
- Continue tuning XGBoost or test alternative ensemble methods if needed for comparison.

In [ ]:

```

```