Sentiment Analysis of Amazon Fine Food Reviews

Group 9 Members

Daniel Ogando

Joel Kioko

Humphrey Matagaro

Lydia Mangoa

Christine Ambasa

Bryan njogu

1 Business Understanding

The rapid growth of e-marketplaces like Amazon has led to an enormous amount of customer-generated data,such as product reviews, ratings, and feedback. This data contains important information regarding customer feedback, that can directly affect business decisions like product improvements, marketing campaigns, and customer reach programs.

For food business firms, understanding customer sentiment regarding their products is critical to product quality, improvement and customer satisfaction.

1.1 Business Impact

The models used here-in can help in;

1. Improved Product Quality: Finding negative comments informs restaurant owners of repeated problems in their products.
2. Improved Customer Experience: Through customer sentiment analysis, organizations are able to customize services as per customer requirements.
3. Data-Driven Decision Making: Sentiment analysis generates quantitative data for measuring customer satisfaction that enables more strategic decisions.

1.2 Objectives

1. Create a robust sentiment analysis model using advanced Natural Language Processing (NLP) techniques for classifying customer reviews as positive or negative.

2. Examine the performance of traditional ML (Logistic Regression, Random Forest, Naive Bayes and XGboost) and deep learning models (TextCNN, DistilBERT).
3. Provide actionable recommendations to businesses based on examining patterns in customer feedback.

1.3 Expected Outcomes

1. A sentiment analysis model with high predictive accuracy and generalizability.
2. Insights into customer sentiment trends that can inform business decisions.
3. Recommendations for deploying the models based on performance.

1.4 Limitations

1. Sentiment Classification: Automatically classifying reviews as "positive", "negative" or "neutral" from text content and matching ratings.
2. Class Imbalance: The dataset exhibits a skewed distribution of ratings, with a majority of positive reviews (70% of ratings are 4-5 stars), which may lead to better model performance on positive reviews.
3. High Dimensionality: Text data is inherently high-dimensional, and preprocessing and feature engineering must be effective processes transforming it into a state suitable for machine learning models.

1.5 Data source

https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews?resource=download

Given the large size of the Amazon Food Reviews dataset, which contains over 500,000 entries, the group performed random sampling using excel(we generated random numbers in a new column and sorted them which sorted the whole data and picked the first 7,658 cells) to select a subset of 7,658 reviews for analysis in order to make the data more manageable and efficient to work with.

1.6 Stakeholders

1. E-commerce Sites: Amazon and other online shopping websites can use this model to measure customer satisfaction and improve the quality of the products.
2. Product Manufacturers: Food businesses can employ sentiment analysis to identify areas of improvement in their products.
3. Marketing Teams: Direct marketing campaigns and customer interaction policies can be informed through sentiment insights.

4. Data Scientists: The project can be referenced for deploying sophisticated NLP techniques like BERT and text classification pipelines.

1.7 Loading the data and exploring.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('reviews3.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	21183	B002QWP89S	A8NX9GKEUAG8U	S. Kim	0	
1	24619	B0047E2I5U	A74ZHK7QFVQ29	Artìst	1	
2	18727	B000FFRU3U	A3K7XR1T8LBTB	K. Frappier	9	
3	36130	B000FBQ594	A2GJSO5UUMNLX0	Sunny	1	
4	3304	B005K4Q1VI	A1TOW634MZ0ELQ	kathyallen45	0	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7658 entries, 0 to 7657
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Id                      7658 non-null   int64
 1   ProductId               7658 non-null   object
 2   UserId                  7658 non-null   object
 3   ProfileName             7658 non-null   object
 4   HelpfulnessNumerator    7658 non-null   int64
 5   HelpfulnessDenominator  7658 non-null   int64
 6   Score                   7658 non-null   int64
 7   Time                    7658 non-null   int64
 8   Summary                 7657 non-null   object
 9   Text                    7658 non-null   object
dtypes: int64(5), object(5)
memory usage: 598.4+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	
count	7658.000000	7658.000000	7658.000000	7658.000000	7.65800(
mean	24951.374380	1.676547	2.252285	3.827239	1.29560(
std	14384.295842	5.863804	6.737611	1.435316	4.649418
min	6.000000	0.000000	0.000000	1.000000	1.06764!
25%	12670.500000	0.000000	0.000000	3.000000	1.27105(
50%	24694.000000	0.000000	1.000000	4.000000	1.30870
75%	37480.750000	2.000000	2.000000	5.000000	1.330387
max	49993.000000	187.000000	216.000000	5.000000	1.35121(

In [6]:
```
df.isna().sum()
```

Out[6]:
```
Id                      0
ProductId               0
UserId                  0
ProfileName             0
HelpfulnessNumerator    0
HelpfulnessDenominator  0
Score                   0
Time                    0
Summary                 1
Text                    0
dtype: int64
```

In [7]:
```
df.duplicated().sum()
```

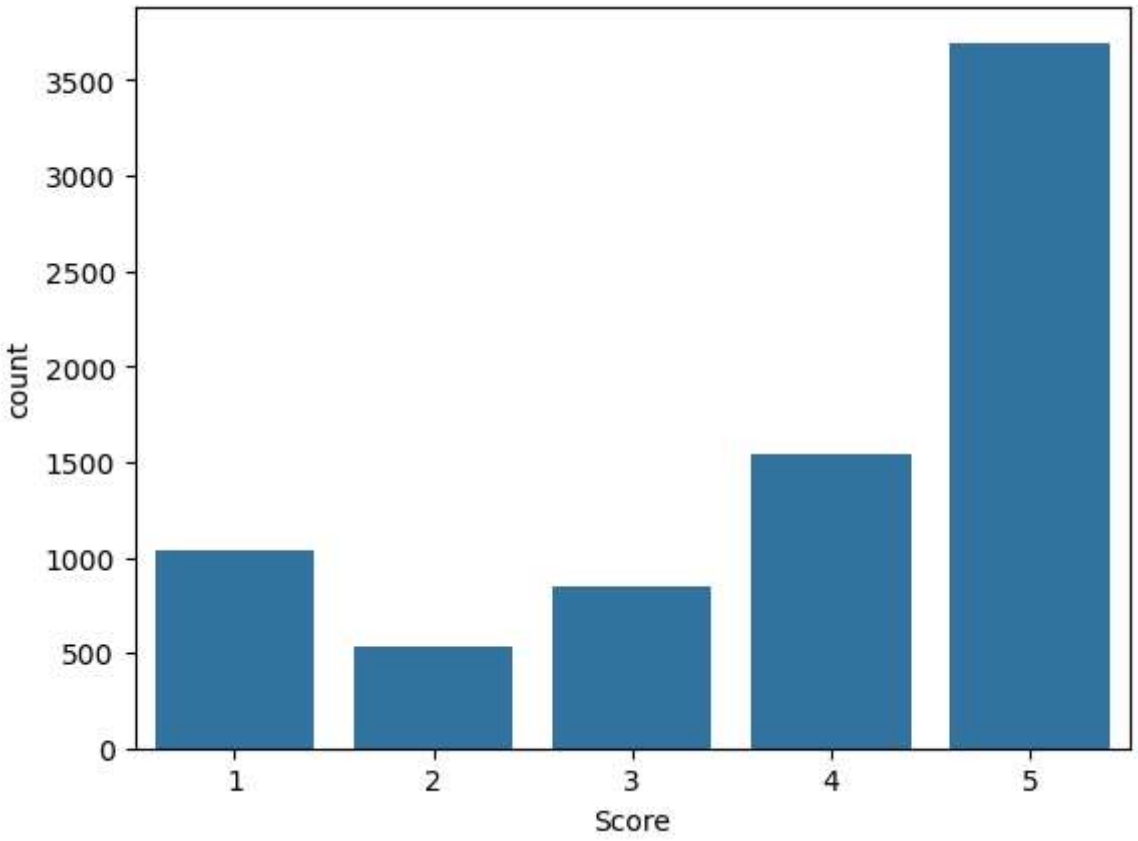
Out[7]:
```
0
```

1.8 Visualization of ratings distribution

Distribution of ratings(1,2,3,4,5)

In [8]:
```
sns.barplot(df['Score'].value_counts())
```

Out[8]:
```
<Axes: xlabel='Score', ylabel='count'>
```



From the graph above we can see that the data is skewed towards positive reviews. We can make the assumption that generally food that is being offered is made to a good standard that meets most customers expectations. However, due to varying tastes and preferences, and a few lapses in presentation of the food, some consumers will rate the food poorly as seen above.

The above scenario creates a limitation as mentioned earlier in the business understanding as a class imbalance

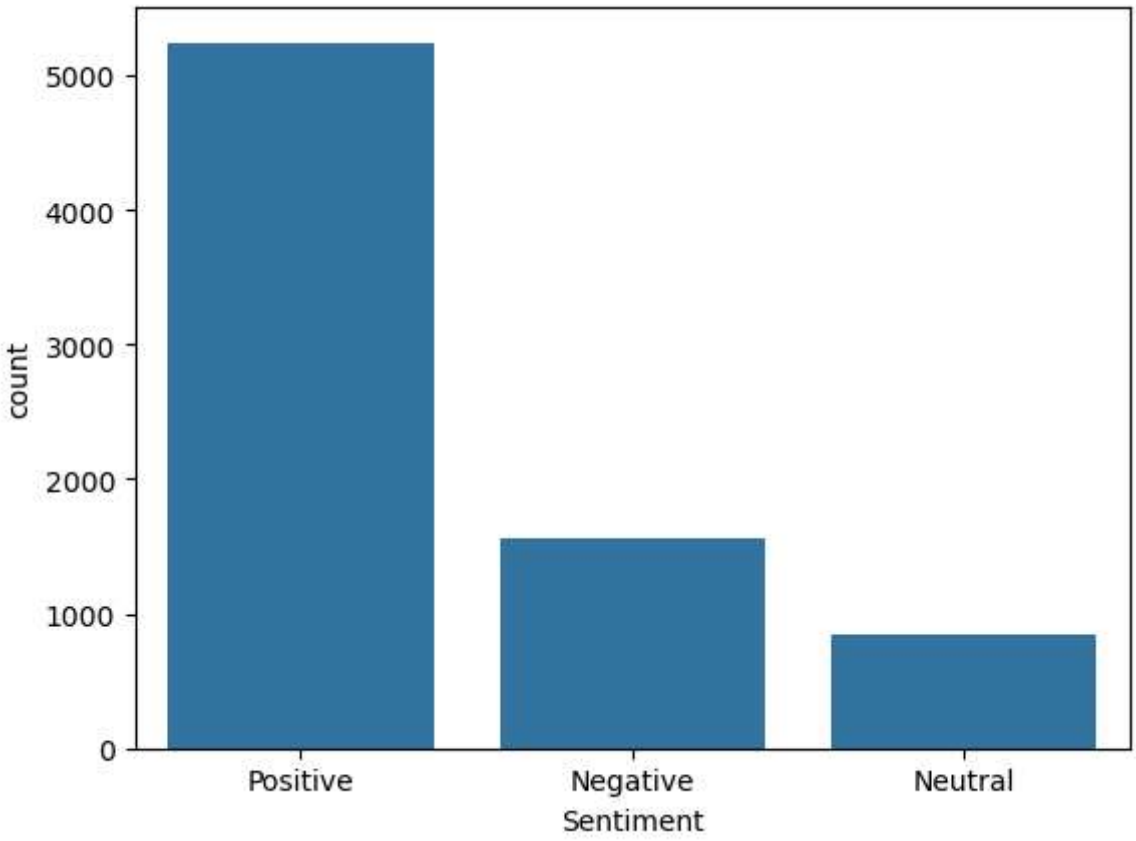
In [9]:
```
# Creating a new column named 'Sentiment' by mapping the ratings in the score colum
df['Sentiment'] = df['Score'].map({1: 'Negative', 2: 'Negative', 3: 'Neutral', 4: '
df.head()
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	21183	B002QWP89S	A8NX9GKEUAG8U	S. Kim	0	
1	24619	B0047E2I5U	A74ZHK7QFVQ29	Artìst	1	
2	18727	B000FFRU3U	A3K7XR1T8LBTB	K. Frappier	9	
3	36130	B000FBQ594	A2GJSO5UUMNLX0	Sunny	1	
4	3304	B005K4Q1VI	A1TOW634MZ0ELQ	kathyallen45	0	

In [10]:
```
sns.barplot(df['Sentiment'].value_counts())
```

Out[10]:
```
<Axes: xlabel='Sentiment', ylabel='count'>
```



The above plot still points to the skweness of the reviews. This further reinforces that the services offered have been well accepted by the customers.

2. Data Preparation & Pre-processing

Next we need to clean and prepare the data

In [11]:
```
# importing the NLP packages for cleaning
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('punkt', download_dir=nltk.data.find('corpora').path)
nltk.download('punkt', download_dir='~/nltk_data')
nltk.data.path.append('~/nltk_data')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')

#initializing the stopwords and Lemmantization
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/matagaro/nltk_data/corpora...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt to ~/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt to /Users/matagaro/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/matagaro/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/matagaro/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/matagaro/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Tokenizing the data in the text column to pull out the keywords for analysis.

In [12]:
```
# importinng and initializing the Tokenizer
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()

def clean_and_tokenize(text):
    text = str(text)
    text = re.sub(r"<.*?>", " ", text)
    text = re.sub(r"http\S+|www.\S+", " ", text)
    text = re.sub(r"[^a-zA-Z\s]", " ", text)
    text = re.sub(r"\s+", " ", text).strip().lower()

    tokens = tokenizer.tokenize(text)
    tokens = [t for t in tokens if t not in stop_words]
    tokens = [lemmatizer.lemmatize(t) for t in tokens]

    return tokens

df['Tokens'] = df['Text'].apply(clean_and_tokenize)

print(df[['Text', 'Tokens']].head())
```

```
                                                Text  \
0  My dog has been loving these chews for years! ...
1  I have a 6 year old westie that had the worst ...
2  I love Dr. McDougall's products, and this is m...
3  Wow, was I surprised when I first bit into one...
4  Wow what a deal. These taste just as good as t...

                                              Tokens
0  [dog, loving, chew, year, give, one, everyday,...
1  [year, old, westie, worst, food, allergy, coat...
2  [love, dr, mcdougall, product, favorite, insta...
3  [wow, surprised, first, bit, one, cooky, absol...
4  [wow, deal, taste, good, expensive, brand, def...
```

Generating TF-IDF scores

In [13]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')

tfidf_matrix = vectorizer.fit_transform(df['Text'].astype(str))

tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_name

print(f"\nTF-IDF matrix shape: {tfidf_df.shape}")
print("\nTF-IDF feature preview:")
print(tfidf_df.head())
```

```
TF-IDF matrix shape: (7658, 5000)

TF-IDF feature preview:
    00  000  0mg   10  100  10th   11  110        12  120  ...  zevia  zico  \
0  0.0  0.0  0.0  0.0  0.0   0.0  0.0  0.0  0.000000  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  0.131463  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

   zing  zip  zipfizz  ziploc  ziplock  zucchini  zuke  zukes
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
2   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
4   0.0  0.0      0.0     0.0      0.0       0.0   0.0    0.0

[5 rows x 5000 columns]
```

3. Modeling

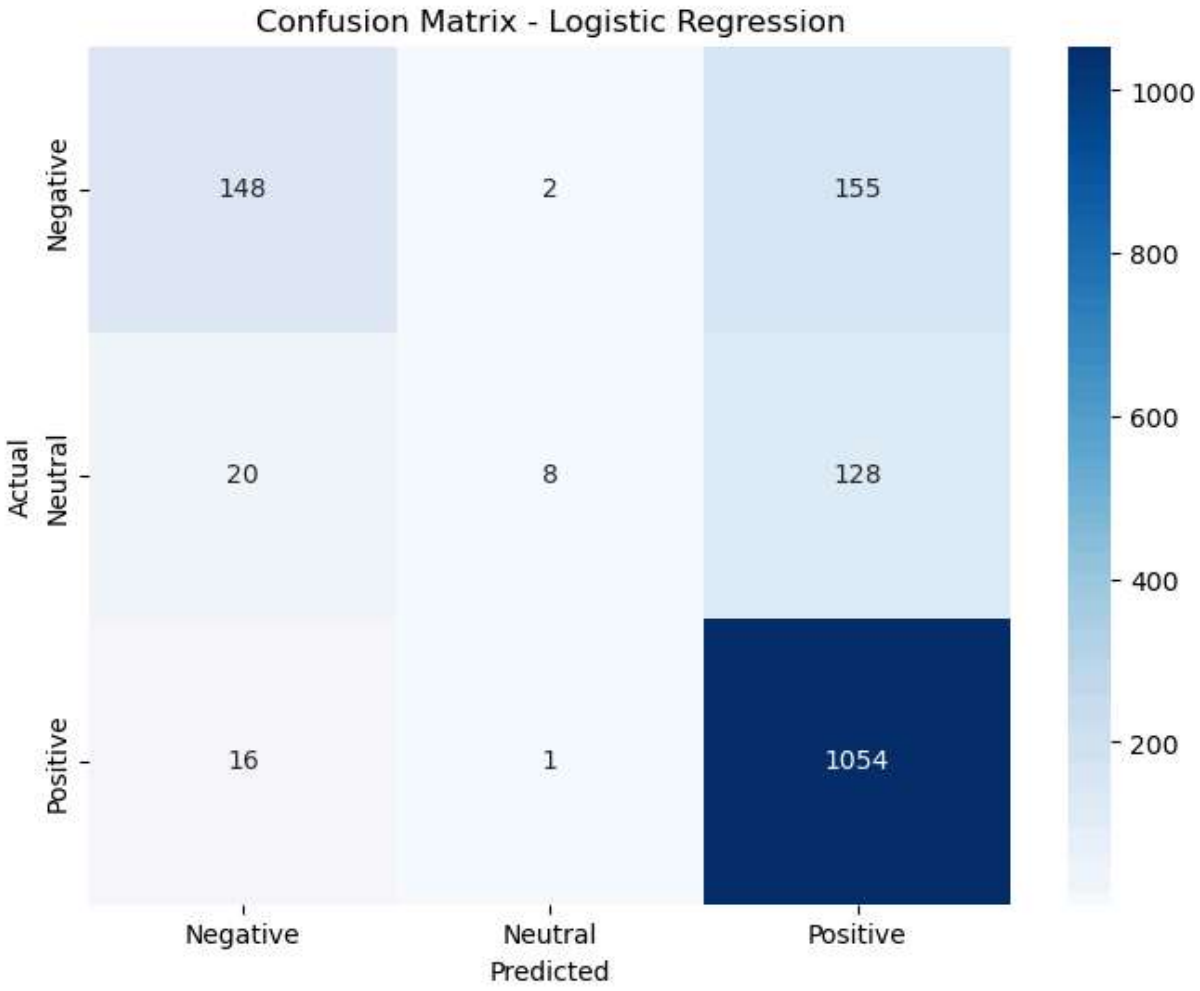
3.1 Logistic Regression

- We started off with a simple logistic regression model to gauge the performance

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df['Sentiment_Encoded'] = label_encoder.fit_transform(df['Sentiment'])

X_tfidf_train, X_tfidf_test, y_tfidf_train, y_tfidf_test = train_test_split(
    tfidf_df, df['Sentiment_Encoded'], test_size=0.2, random_state=42
)

logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_tfidf_train, y_tfidf_train)
```

```
y_pred_logreg = logreg.predict(X_tfidf_test)


print("Logistic Regression Classification Report:")
print(classification_report(y_tfidf_test, y_pred_logreg, target_names=label_encoder

# Confusion Matrix
cm = confusion_matrix(y_tfidf_test, y_pred_logreg)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```

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

    Negative       0.80      0.49      0.61       305
     Neutral       0.73      0.05      0.10       156
    Positive       0.79      0.98      0.88      1071

    accuracy                           0.79      1532
   macro avg       0.77      0.51      0.53      1532
weighted avg       0.79      0.79      0.74      1532
```



Summary analysis of the Logistic Regression model

The logistic regression model performs well for the majority class (Positive) with precision (0.79), recall (0.98), and F1-score (0.88) all being high. This indicates that the model is extremely good at identifying and correctly classifying positive instances, and this may be because it appears the most in the dataset.

The model performs poorly on the minority classes.

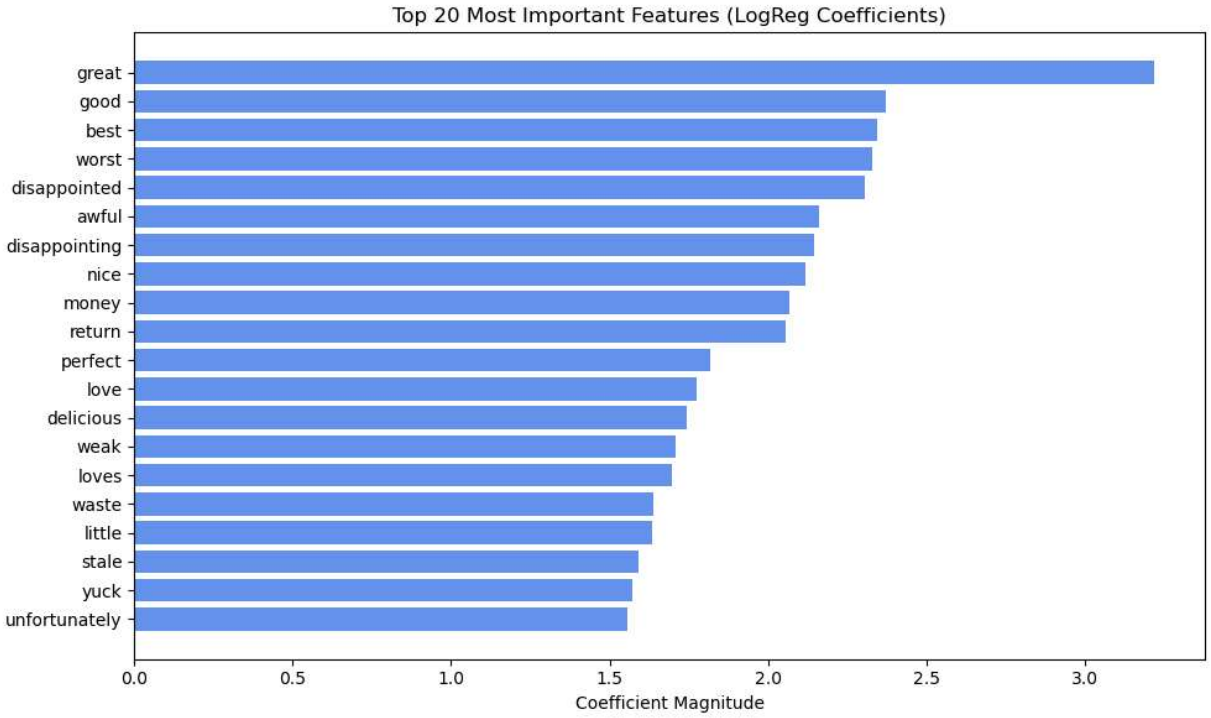
For the Negative class, although precision is very good at 0.80, the recall is much lower at 0.49, meaning many negative instances are being misclassified.

The Neutral class performs least but not too far behind Negative class, with a precision of (0.73), recall (0.05), and F1-score (0.10), meaning the model is failing to accurately identify or classify neutral sentiments. This is likely because of both the small size of the neutral class and the enormous imbalance of the positive class in the data.

The overall accuracy of the model is 0.79, but it is deceptive due to the severe class imbalance.

In [15]:
```
feat_importance = np.abs(logreg.coef_[0])
```

```
top_n = 20
top_indices = np.argsort(feat_importance)[-top_n:]

try:
    words = vectorizer.get_feature_names_out()
except:
    words = vectorizer.get_feature_names()

plt.figure(figsize=(10, 6))
plt.title('Top 20 Most Important Features (LogReg Coefficients)')
plt.barh(range(len(top_indices)), feat_importance[top_indices],
         color='cornflowerblue', align='center')
plt.yticks(range(len(top_indices)), [words[i] for i in top_indices])
plt.xlabel('Coefficient Magnitude')
plt.tight_layout()
plt.show()
```



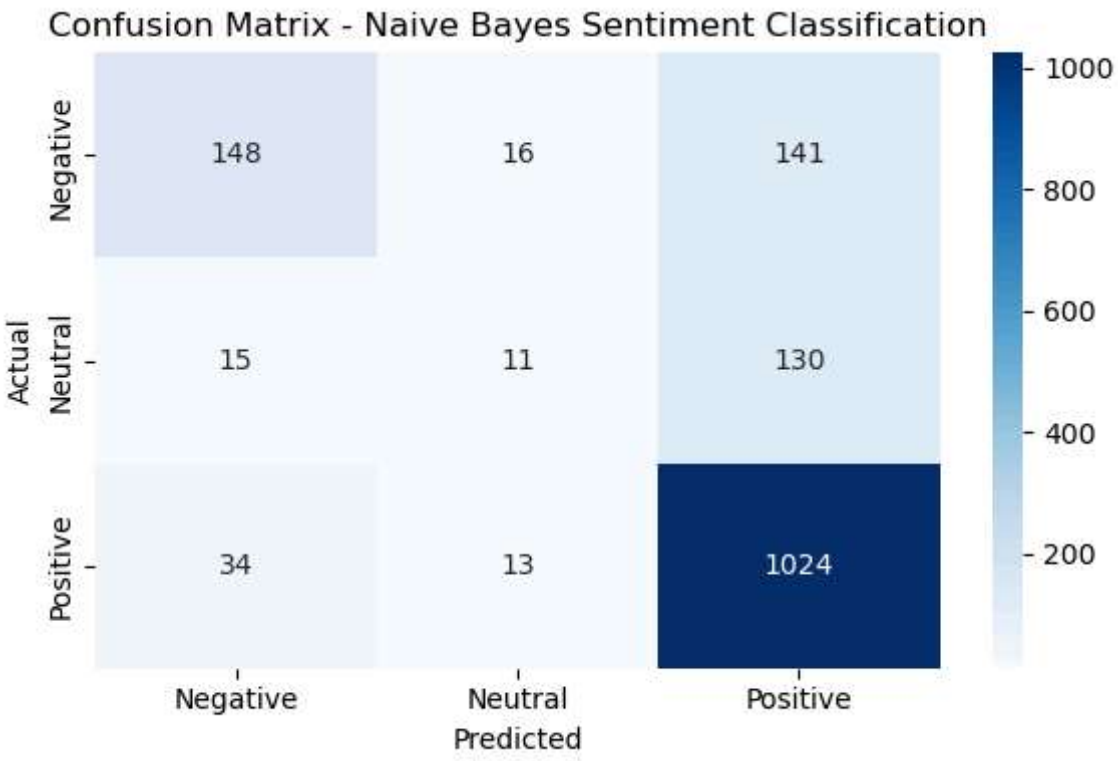
3.2. Naive Baiyes Model

In [16]:
```
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 classification_report, confusion_matrix, accuracy_score

X = df['Text']
y = df['Sentiment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

vectorizer = CountVectorizer(stop_words='english')
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

```
model = MultinomialNB()
model.fit(X_train_vec, y_train)
y_pred = model.predict(X_test_vec)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.7721932114882507

Classification Report:
              precision    recall  f1-score   support

    Negative       0.75      0.49      0.59       305
     Neutral       0.28      0.07      0.11       156
    Positive       0.79      0.96      0.87      1071

    accuracy                           0.77      1532
   macro avg       0.61      0.50      0.52      1532
weighted avg       0.73      0.77      0.73      1532


Confusion Matrix:
 [[ 148   16  141]
 [  15   11  130]
 [  34   13 1024]]
```

In [17]:
```
# plotting confusion matrix for Naive Bayes
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=model.classes_, yticklabels=model.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Naive Bayes Sentiment Classification')
plt.tight_layout()
plt.show()
```



Analysis of Naives Bayes Model

Negative Reviews

- Precision: 0.75 : When the model predicts a review as Negative, it's correct 75% of the time.
- Recall: 0.49 : Out of all actual Negative reviews, the model only identifies 49% correctly.
- F1-score: 0.59 : This is the balance between precision and recall, indicating moderate overall performance.

The model is somewhat responsive to Negative reviews; it makes fewer false positive errors (good precision) but misses many true Negative cases (low recall). This might be due to fewer Negative examples or overlapping vocabulary with other sentiment types.

Neutral Reviews

- Precision: 0.28 : Of those predicted as Neutral, only 28% are truly Neutral.
- Recall: 0.07 : The model correctly identifies only 7% of all Neutral reviews.
- F1-score: 0.11 : Extremely low, indicating poor performance.

The Neutral class is not well predicted, likely because it has less distinct features or a smaller sample size relative to Positive and Negative classes. The model almost always misclassifies Neutral reviews as either Positive or Negative.

Positive Reviews

- Precision: 0.79 : When the model predicts a review as Positive, 79% of those are indeed Positive.
- Recall: 0.96 : The model successfully captures 96% of the actual Positive reviews, which is excellent.
- F1-score: 0.87 : Indicates robust performance in identifying Positive reviews.

The model is highly effective at recognizing Positive reviews. This strong performance could be due to a larger number of Positive samples or more distinct word usage in Positive reviews compared to the other classes.

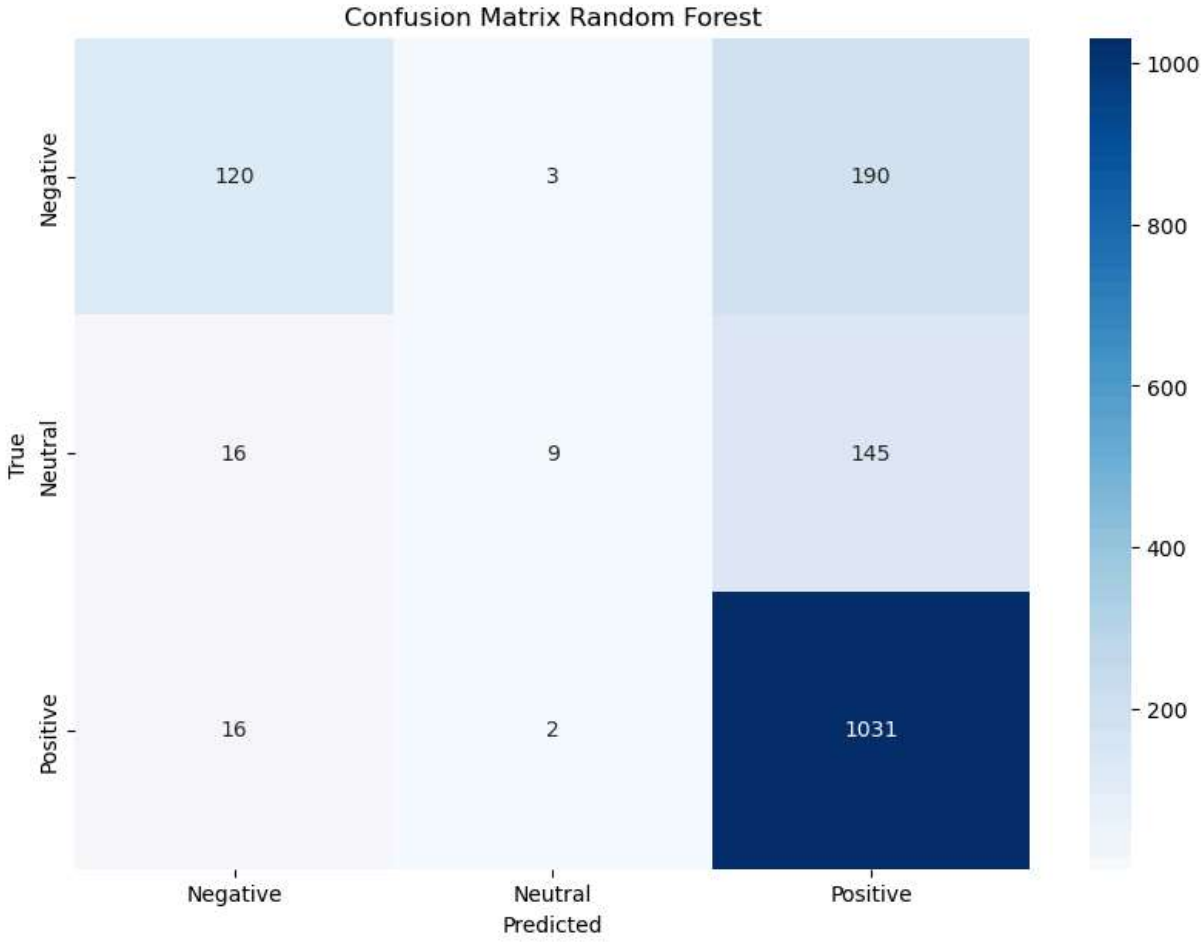
Overall accuracy of the model is 77%

3.3. Random forest Classifier

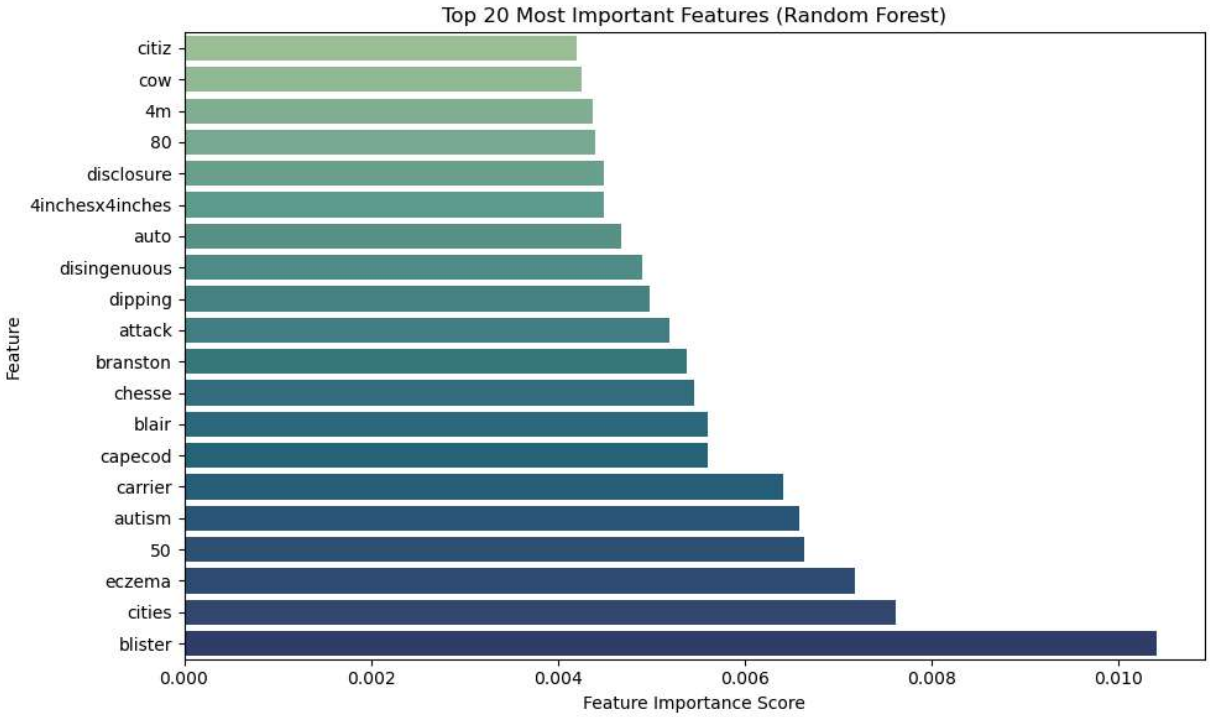
In [18]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

x = tfidf_df
y = df["Sentiment"]

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.2, random_sta
rf_model = RandomForestClassifier (n_estimators = 100, random_state = 42, n_jobs =
rf_model.fit(x_train, y_train)

y_pred = rf_model.predict(x_test)


print("Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Model Evaluation:
Accuracy: 0.7571801566579635

Classification Report:
              precision    recall  f1-score   support

    Negative       0.79      0.38      0.52       313
     Neutral       0.64      0.05      0.10       170
    Positive       0.75      0.98      0.85      1049

    accuracy                           0.76      1532
   macro avg       0.73      0.47      0.49      1532
weighted avg       0.75      0.76      0.70      1532


Confusion Matrix:
[[ 120    3  190]
 [  16    9  145]
 [  16    2 1031]]
```

In [19]:
```
cm = confusion_matrix ( y_test, y_pred, labels= ['Negative', 'Neutral', 'Positive']
plt.figure (figsize = (10, 7))
sns.heatmap (cm, annot = True, fmt='d', cmap = 'Blues', xticklabels=['Negative', 'N
             yticklabels=['Negative', 'Neutral', 'Positive'])
plt.title('Confusion Matrix Random Forest')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



In [20]:
```python
# Plotting feature importance for Random Classifier
feat_importance = rf_model.feature_importances_

try:
    feature_names = vectorizer.get_feature_names_out()
except:
    feature_names = vectorizer.get_feature_names()

top_n = 20
top_indices = np.argsort(feat_importance)[-top_n:]

top_features = [(feature_names[i], feat_importance[i]) for i in top_indices]
importance_df = pd.DataFrame(top_features, columns=['Feature', 'Importance'])

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette='crest')
plt.title('Top 20 Most Important Features (Random Forest)')
plt.xlabel('Feature Importance Score')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```



Summary of the Random forest model

Random Forest model performance was 76% on the test set with strong performance on positive sentiment prediction (98% recall) but weaker performance on negative (38% recall) and neutral sentiments (5% recall). From the confusion matrix, there were significant misclassifications, particularly on negative and neutral classes that had a tendency to be predicted as positive. Although the model works well in identifying positive sentiments, it is poor in distinguishing between negative and neutral case. This means that the model can be enhanced by addressing class imbalance or feature optimization for minority classes

3.4. XG BOOST MODEL

In [21]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /opt/anaconda3/lib/python3.12/site-package
s (3.0.0)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.12/site-packages
(from xgboost) (1.26.4)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.12/site-packages
(from xgboost) (1.13.1)
```

In [22]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from xgboost import XGBClassifier
df['label'] = df['Score'].apply(lambda x: 1 if x >= 4 else 0)
df_sampled = df.sample(n=5000, random_state=42)

vectorizer = TfidfVectorizer(max_features=1000, stop_words='english')
X = vectorizer.fit_transform(df_sampled['Text'])

y = df_sampled['label']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
model.fit(X_train, y_train)
from sklearn.metrics import accuracy_score, precision_score, classification_report

y_pred = model.predict(X_test)

# Metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print("\nClassification Report:\n", report)
```

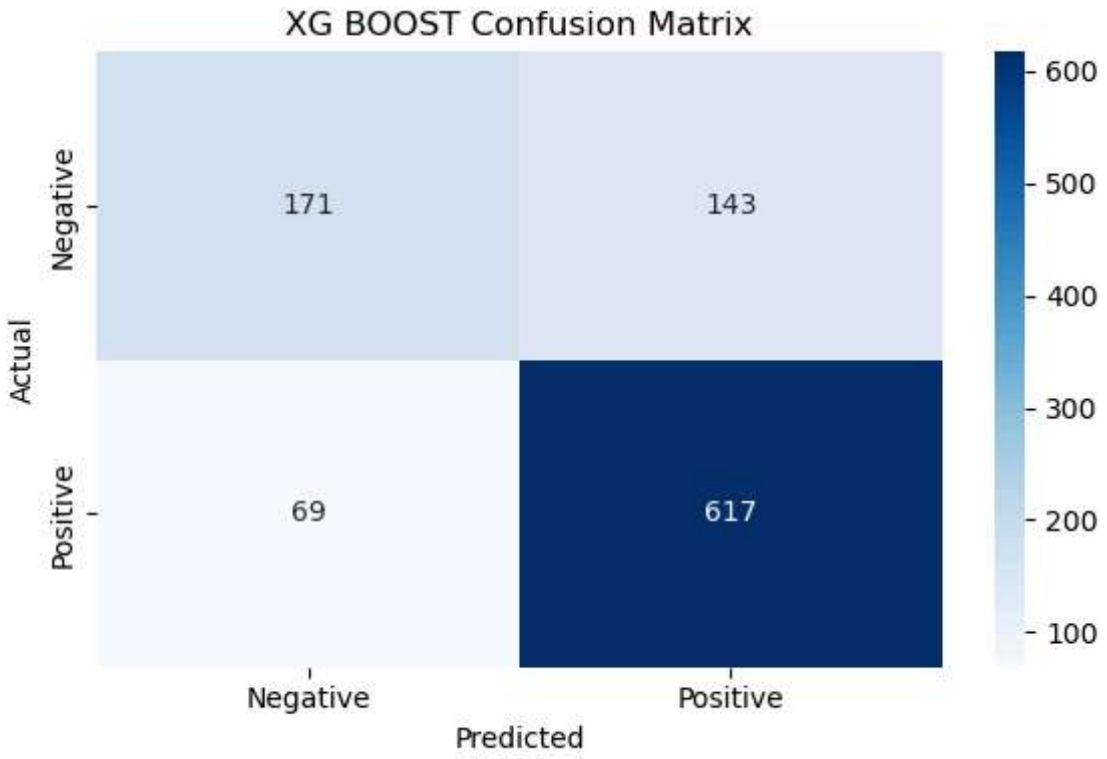
```
Accuracy: 0.7880
Precision: 0.8118

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.54      0.62       314
           1       0.81      0.90      0.85       686

    accuracy                           0.79      1000
   macro avg       0.76      0.72      0.74      1000
weighted avg       0.78      0.79      0.78      1000
```

Summary of the XG Boost Model

The XGBoost model achieves an accuracy of 79%, indicating reliable performance, particularly in predicting the positive(1) class. It performs strongly on positive(1) class , with excellent precision (0.81), recall (0.90), and F1-score (0.85), showing that it accurately and consistently identifies instances of this class. However, its performance on class negative & neutral(0) is notably weaker, with lower precision (0.71), recall (0.54), and F1-score (0.62), indicating a tendency to misclassify or overlook instances of this minority class.

In [23]:
```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=["Negative", "Positive"],
            yticklabels=["Negative", "Positive"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('XG BOOST Confusion Matrix')
```

```
plt.tight_layout()
plt.show()
```



3.5. TextCNN Model

In [24]:
```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /opt/anaconda3/lib/python3.12/site-pack
ages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /opt/anaconda3/lib/python3.12/site-
packages (from tensorflow) (2.2.2)
Requirement already satisfied: astunparse>=1.6.0 in /opt/anaconda3/lib/python3.12/si
te-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /opt/anaconda3/lib/python3.1
2/site-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /opt/anaconda
3/lib/python3.12/site-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /opt/anaconda3/lib/python3.12/sit
e-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/anaconda3/lib/python3.12/si
te-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /opt/anaconda3/lib/python3.12/site-packa
ges (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.
4,!=4.21.5,<6.0.0dev,>=3.20.3 in /opt/anaconda3/lib/python3.12/site-packages (from t
ensorflow) (4.25.3)
Requirement already satisfied: requests<3,>=2.21.0 in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.12/site-pack
ages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /opt/anaconda3/lib/python3.12/site-pac
kages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /opt/anaconda3/lib/python3.12/sit
e-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/anaconda3/lib/python
3.12/site-packages (from tensorflow) (4.11.0)
Requirement already satisfied: wrapt>=1.11.0 in /opt/anaconda3/lib/python3.12/site-p
ackages (from tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard~=2.19.0 in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /opt/anaconda3/lib/python3.12/site-pa
ckages (from tensorflow) (3.9.2)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /opt/anaconda3/lib/python3.1
2/site-packages (from tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in /opt/anaconda3/lib/python3.12/site-pa
ckages (from tensorflow) (3.11.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /opt/anaconda3/lib/python
3.12/site-packages (from tensorflow) (0.5.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /opt/anaconda3/lib/python3.12/s
ite-packages (from astunparse>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: rich in /opt/anaconda3/lib/python3.12/site-packages
(from keras>=3.5.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in /opt/anaconda3/lib/python3.12/site-packages
(from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /opt/anaconda3/lib/python3.12/site-packages
(from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/lib/python
3.12/site-packages (from requests<3,>=2.21.0->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.12/site-pa
```

```
ckages (from requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/python3.12/s
ite-packages (from requests<3,>=2.21.0->tensorflow) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.12/s
ite-packages (from requests<3,>=2.21.0->tensorflow) (2025.1.31)
Requirement already satisfied: markdown>=2.6.8 in /opt/anaconda3/lib/python3.12/site
-packages (from tensorboard~=2.19.0->tensorflow) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /opt/anacond
a3/lib/python3.12/site-packages (from tensorboard~=2.19.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /opt/anaconda3/lib/python3.12/site
-packages (from tensorboard~=2.19.0->tensorflow) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/anaconda3/lib/python3.12/si
te-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in /opt/anaconda3/lib/python3.1
2/site-packages (from rich->keras>=3.5.0->tensorflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /opt/anaconda3/lib/python
3.12/site-packages (from rich->keras>=3.5.0->tensorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in /opt/anaconda3/lib/python3.12/site-pack
ages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.0)
```

In [25]:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, D


MAX_SEQUENCE_LENGTH = 100
MAX_VOCAB_SIZE = 10000
embedding_dim = 100


tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
tokenizer.fit_on_texts(df['Text'])
sequences = tokenizer.texts_to_sequences(df['Text'])

from tensorflow.keras.preprocessing.sequence import pad_sequences

X = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')


label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['Sentiment'])
y = np.eye(len(label_encoder.classes_))[y]


X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand
```

In [26]:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, D
```

```
MAX_SEQUENCE_LENGTH = 100
MAX_VOCAB_SIZE = 10000
embedding_dim = 100

input_layer = Input(shape=(MAX_SEQUENCE_LENGTH,))


embedding_layer = Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=embedding_dim, inp

conv_3 = Conv1D(filters=128, kernel_size=3, activation='relu')(embedding_layer)
conv_4 = Conv1D(filters=128, kernel_size=4, activation='relu')(embedding_layer)
conv_5 = Conv1D(filters=128, kernel_size=5, activation='relu')(embedding_layer)

pool_3 = GlobalMaxPooling1D()(conv_3)
pool_4 = GlobalMaxPooling1D()(conv_4)
pool_5 = GlobalMaxPooling1D()(conv_5)

concat = Concatenate()([pool_3, pool_4, pool_5])

dropout = Dropout(0.5)(concat)


output = Dense(3, activation='softmax')(dropout)

model = Model(inputs=input_layer, outputs=output)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy

model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100)	0	-
embedding (Embedding)	(None, 100, 100)	1,000,000	input_layer[0][0]
conv1d (Conv1D)	(None, 98, 128)	38,528	embedding[0][0]
conv1d_1 (Conv1D)	(None, 97, 128)	51,328	embedding[0][0]
conv1d_2 (Conv1D)	(None, 96, 128)	64,128	embedding[0][0]
global_max_pooling… (GlobalMaxPooling1…	(None, 128)	0	conv1d[0][0]
global_max_pooling… (GlobalMaxPooling1…	(None, 128)	0	conv1d_1[0][0]
global_max_pooling… (GlobalMaxPooling1…	(None, 128)	0	conv1d_2[0][0]
concatenate (Concatenate)	(None, 384)	0	global_max_pooli… global_max_pooli… global_max_pooli…
dropout (Dropout)	(None, 384)	0	concatenate[0][0]
dense (Dense)	(None, 3)	1,155	dropout[0][0]

Total params: 1,155,139 (4.41 MB)

Trainable params: 1,155,139 (4.41 MB)

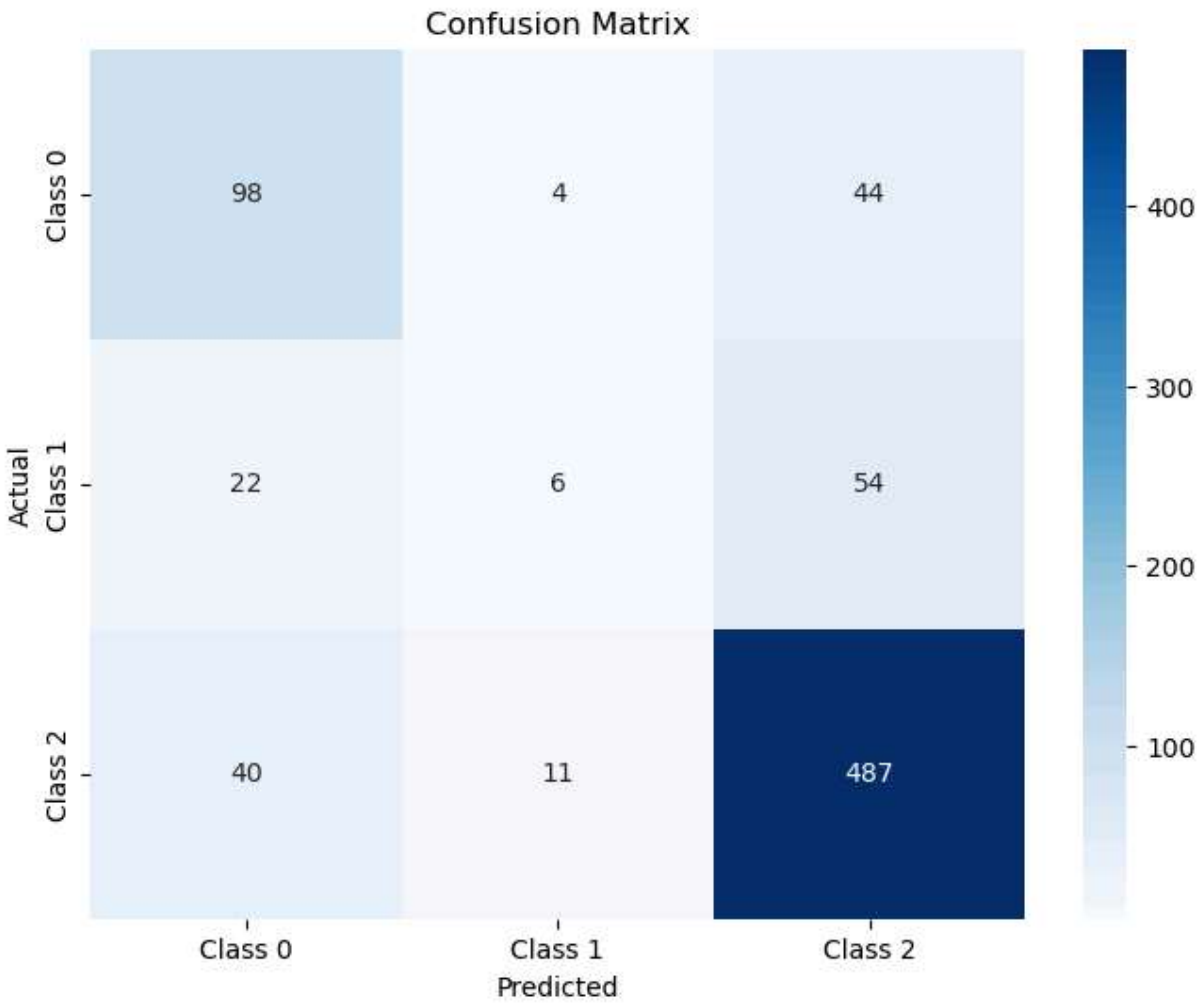
Non-trainable params: 0 (0.00 B)

In [27]:
```
#Train the model
history = model.fit(X_train, y_train, validation_split=0.1, epochs=5, batch_size=64
```

```
Epoch 1/5
87/87 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.6139 - loss: 0.8990 - val_accu
racy: 0.6688 - val_loss: 0.7853
Epoch 2/5
87/87 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7067 - loss: 0.7076 - val_accu
racy: 0.7390 - val_loss: 0.6491
Epoch 3/5
87/87 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.8102 - loss: 0.5108 - val_accu
racy: 0.7504 - val_loss: 0.6119
Epoch 4/5
87/87 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.8555 - loss: 0.3780 - val_accu
racy: 0.7569 - val_loss: 0.6081
Epoch 5/5
87/87 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9186 - loss: 0.2507 - val_accu
racy: 0.7553 - val_loss: 0.6416
```

In [28]:
```
from sklearn.metrics import confusion_matrix, classification_report, precision_scor

loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"TextCNN Accuracy: {acc:.4f}")
print(f"TextCNN Loss: {loss:.4f}")

y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)


y_true = np.argmax(y_test, axis=1)


precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')

print("\nAdditional Metrics:")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_true, y_pred))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)

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

```
TextCNN Accuracy: 0.7715
TextCNN Loss: 0.6150
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step

Additional Metrics:
Precision: 0.7320
Recall: 0.7715
F1-Score: 0.7437

Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.67      0.64       146
           1       0.29      0.07      0.12        82
           2       0.83      0.91      0.87       538

    accuracy                           0.77       766
   macro avg       0.58      0.55      0.54       766
weighted avg       0.73      0.77      0.74       766
```



Summary of TEXT CNN model

TextCNN is generally performing well with 0.77 accuracy, loss 0.6150, recall: 0.77, and F1-score: 0.74—indicating good overall balance between correctly identifying proper instances and minimizing false positives.

Performance, however, varies very differently between classes. The model does a good job for Class 2(positive) with very high precision (0.83), recall (0.91), and F1-score (0.87) indicating good capacity to identify and correctly classify this common class.

In contrast, Class 1(negative) performance is poor, with low precision (0.29), recall (0.07), and F1-score (0.87), indicating a consistent failure to recognize true instances of this class. Class 0(neutral) performance is moderate, with precision (0.61), recall (0.91), and F1-score (0.87).

3.6 DistilBERT Model Classifier

In [29]:
```
!pip install transformers
!pip install torch
!pip install PyTorch
```

```
!pip install "transformers[torch]"
!pip install tf-keras
```

```
Requirement already satisfied: transformers in /opt/anaconda3/lib/python3.12/site-pa
ckages (4.51.3)
Requirement already satisfied: filelock in /opt/anaconda3/lib/python3.12/site-packag
es (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /opt/anaconda3/lib/py
thon3.12/site-packages (from transformers) (0.30.2)
Requirement already satisfied: numpy>=1.17 in /opt/anaconda3/lib/python3.12/site-pac
kages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.12/site
-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /opt/anaconda3/lib/python3.12/site-pac
kages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /opt/anaconda3/lib/python3.12/si
te-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /opt/anaconda3/lib/python3.12/site-packag
es (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /opt/anaconda3/lib/python3.
12/site-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /opt/anaconda3/lib/python3.12/s
ite-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /opt/anaconda3/lib/python3.12/site-pack
ages (from transformers) (4.66.5)
Requirement already satisfied: fsspec>=2023.5.0 in /opt/anaconda3/lib/python3.12/sit
e-packages (from huggingface-hub<1.0,>=0.30.0->transformers) (2024.6.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/anaconda3/lib/pyth
on3.12/site-packages (from huggingface-hub<1.0,>=0.30.0->transformers) (4.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/lib/python
3.12/site-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.12/site-pa
ckages (from requests->transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/python3.12/s
ite-packages (from requests->transformers) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.12/s
ite-packages (from requests->transformers) (2025.1.31)
Requirement already satisfied: torch in /opt/anaconda3/lib/python3.12/site-packages
(2.6.0)
Requirement already satisfied: filelock in /opt/anaconda3/lib/python3.12/site-packag
es (from torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.10.0 in /opt/anaconda3/lib/pytho
n3.12/site-packages (from torch) (4.11.0)
Requirement already satisfied: networkx in /opt/anaconda3/lib/python3.12/site-packag
es (from torch) (3.3)
Requirement already satisfied: jinja2 in /opt/anaconda3/lib/python3.12/site-packages
(from torch) (3.1.4)
Requirement already satisfied: fsspec in /opt/anaconda3/lib/python3.12/site-packages
(from torch) (2024.6.1)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.12/site-pack
ages (from torch) (75.1.0)
Requirement already satisfied: sympy==1.13.1 in /opt/anaconda3/lib/python3.12/site-p
ackages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/anaconda3/lib/python3.12/s
ite-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/anaconda3/lib/python3.12/site
-packages (from jinja2->torch) (2.1.3)
Collecting PyTorch
  Using cached pytorch-1.0.2.tar.gz (689 bytes)
```

```
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: PyTorch
  Building wheel for PyTorch (setup.py) ... error
  error: subprocess-exited-with-error

  × python setup.py bdist_wheel did not run successfully.
  │ exit code: 1
  ╰─> [6 lines of output]
      Traceback (most recent call last):
        File "<string>", line 2, in <module>
        File "<pip-setuptools-caller>", line 34, in <module>
        File "/private/var/folders/08/bhwhfy712k376cznh50tnvsc0000gn/T/pip-install-7
8eqgmev/pytorch_db4f9bd8e2f5450bb50c51eefbba362e/setup.py", line 15, in <module>
          raise Exception(message)
      Exception: You tried to install "pytorch". The package named for PyTorch is "t
orch"
      [end of output]

  note: This error originates from a subprocess, and is likely not a problem with pi
p.
  ERROR: Failed building wheel for PyTorch
  Running setup.py clean for PyTorch
Failed to build PyTorch
ERROR: ERROR: Failed to build installable wheels for some pyproject.toml based proje
cts (PyTorch)
Requirement already satisfied: transformers[torch] in /opt/anaconda3/lib/python3.12/
site-packages (4.51.3)
Requirement already satisfied: filelock in /opt/anaconda3/lib/python3.12/site-packag
es (from transformers[torch]) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /opt/anaconda3/lib/py
thon3.12/site-packages (from transformers[torch]) (0.30.2)
Requirement already satisfied: numpy>=1.17 in /opt/anaconda3/lib/python3.12/site-pac
kages (from transformers[torch]) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.12/site
-packages (from transformers[torch]) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /opt/anaconda3/lib/python3.12/site-pac
kages (from transformers[torch]) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /opt/anaconda3/lib/python3.12/si
te-packages (from transformers[torch]) (2024.9.11)
Requirement already satisfied: requests in /opt/anaconda3/lib/python3.12/site-packag
es (from transformers[torch]) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /opt/anaconda3/lib/python3.
12/site-packages (from transformers[torch]) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /opt/anaconda3/lib/python3.12/s
ite-packages (from transformers[torch]) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /opt/anaconda3/lib/python3.12/site-pack
ages (from transformers[torch]) (4.66.5)
Requirement already satisfied: torch>=2.0 in /opt/anaconda3/lib/python3.12/site-pack
ages (from transformers[torch]) (2.6.0)
Requirement already satisfied: accelerate>=0.26.0 in /opt/anaconda3/lib/python3.12/s
ite-packages (from transformers[torch]) (1.6.0)
Requirement already satisfied: psutil in /opt/anaconda3/lib/python3.12/site-packages
(from accelerate>=0.26.0->transformers[torch]) (5.9.0)
Requirement already satisfied: fsspec>=2023.5.0 in /opt/anaconda3/lib/python3.12/sit
e-packages (from huggingface-hub<1.0,>=0.30.0->transformers[torch]) (2024.6.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/anaconda3/lib/pyth
```

```
on3.12/site-packages (from huggingface-hub<1.0,>=0.30.0->transformers[torch]) (4.11.
0)
Requirement already satisfied: networkx in /opt/anaconda3/lib/python3.12/site-packag
es (from torch>=2.0->transformers[torch]) (3.3)
Requirement already satisfied: jinja2 in /opt/anaconda3/lib/python3.12/site-packages
(from torch>=2.0->transformers[torch]) (3.1.4)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.12/site-pack
ages (from torch>=2.0->transformers[torch]) (75.1.0)
Requirement already satisfied: sympy==1.13.1 in /opt/anaconda3/lib/python3.12/site-p
ackages (from torch>=2.0->transformers[torch]) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/anaconda3/lib/python3.12/s
ite-packages (from sympy==1.13.1->torch>=2.0->transformers[torch]) (1.3.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/lib/python
3.12/site-packages (from requests->transformers[torch]) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.12/site-pa
ckages (from requests->transformers[torch]) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/python3.12/s
ite-packages (from requests->transformers[torch]) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.12/s
ite-packages (from requests->transformers[torch]) (2025.1.31)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/anaconda3/lib/python3.12/site
-packages (from jinja2->torch>=2.0->transformers[torch]) (2.1.3)
Requirement already satisfied: tf-keras in /opt/anaconda3/lib/python3.12/site-packag
es (2.19.0)
Requirement already satisfied: tensorflow<2.20,>=2.19 in /opt/anaconda3/lib/python3.
12/site-packages (from tf-keras) (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /opt/anaconda3/lib/python3.12/site-
packages (from tensorflow<2.20,>=2.19->tf-keras) (2.2.2)
Requirement already satisfied: astunparse>=1.6.0 in /opt/anaconda3/lib/python3.12/si
te-packages (from tensorflow<2.20,>=2.19->tf-keras) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /opt/anaconda3/lib/python3.1
2/site-packages (from tensorflow<2.20,>=2.19->tf-keras) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /opt/anaconda
3/lib/python3.12/site-packages (from tensorflow<2.20,>=2.19->tf-keras) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow<2.20,>=2.19->tf-keras) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /opt/anaconda3/lib/python3.12/sit
e-packages (from tensorflow<2.20,>=2.19->tf-keras) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/anaconda3/lib/python3.12/si
te-packages (from tensorflow<2.20,>=2.19->tf-keras) (3.4.0)
Requirement already satisfied: packaging in /opt/anaconda3/lib/python3.12/site-packa
ges (from tensorflow<2.20,>=2.19->tf-keras) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.
4,!=4.21.5,<6.0.0dev,>=3.20.3 in /opt/anaconda3/lib/python3.12/site-packages (from t
ensorflow<2.20,>=2.19->tf-keras) (4.25.3)
Requirement already satisfied: requests<3,>=2.21.0 in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow<2.20,>=2.19->tf-keras) (2.32.3)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.12/site-pack
ages (from tensorflow<2.20,>=2.19->tf-keras) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /opt/anaconda3/lib/python3.12/site-pac
kages (from tensorflow<2.20,>=2.19->tf-keras) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /opt/anaconda3/lib/python3.12/sit
e-packages (from tensorflow<2.20,>=2.19->tf-keras) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/anaconda3/lib/python
3.12/site-packages (from tensorflow<2.20,>=2.19->tf-keras) (4.11.0)
Requirement already satisfied: wrapt>=1.11.0 in /opt/anaconda3/lib/python3.12/site-p
```

```
ackages (from tensorflow<2.20,>=2.19->tf-keras) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow<2.20,>=2.19->tf-keras) (1.71.0)
Requirement already satisfied: tensorboard~=2.19.0 in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow<2.20,>=2.19->tf-keras) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /opt/anaconda3/lib/python3.12/site-pa
ckages (from tensorflow<2.20,>=2.19->tf-keras) (3.9.2)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /opt/anaconda3/lib/python3.1
2/site-packages (from tensorflow<2.20,>=2.19->tf-keras) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in /opt/anaconda3/lib/python3.12/site-pa
ckages (from tensorflow<2.20,>=2.19->tf-keras) (3.11.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /opt/anaconda3/lib/python
3.12/site-packages (from tensorflow<2.20,>=2.19->tf-keras) (0.5.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /opt/anaconda3/lib/python3.12/s
ite-packages (from astunparse>=1.6.0->tensorflow<2.20,>=2.19->tf-keras) (0.44.0)
Requirement already satisfied: rich in /opt/anaconda3/lib/python3.12/site-packages
(from keras>=3.5.0->tensorflow<2.20,>=2.19->tf-keras) (13.7.1)
Requirement already satisfied: namex in /opt/anaconda3/lib/python3.12/site-packages
(from keras>=3.5.0->tensorflow<2.20,>=2.19->tf-keras) (0.0.8)
Requirement already satisfied: optree in /opt/anaconda3/lib/python3.12/site-packages
(from keras>=3.5.0->tensorflow<2.20,>=2.19->tf-keras) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/lib/python
3.12/site-packages (from requests<3,>=2.21.0->tensorflow<2.20,>=2.19->tf-keras) (3.
3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.12/site-pa
ckages (from requests<3,>=2.21.0->tensorflow<2.20,>=2.19->tf-keras) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/python3.12/s
ite-packages (from requests<3,>=2.21.0->tensorflow<2.20,>=2.19->tf-keras) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.12/s
ite-packages (from requests<3,>=2.21.0->tensorflow<2.20,>=2.19->tf-keras) (2025.1.3
1)
Requirement already satisfied: markdown>=2.6.8 in /opt/anaconda3/lib/python3.12/site
-packages (from tensorboard~=2.19.0->tensorflow<2.20,>=2.19->tf-keras) (3.4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /opt/anacond
a3/lib/python3.12/site-packages (from tensorboard~=2.19.0->tensorflow<2.20,>=2.19->t
f-keras) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /opt/anaconda3/lib/python3.12/site
-packages (from tensorboard~=2.19.0->tensorflow<2.20,>=2.19->tf-keras) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/anaconda3/lib/python3.12/si
te-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->tensorflow<2.20,>=2.19->tf-k
eras) (2.1.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in /opt/anaconda3/lib/python3.1
2/site-packages (from rich->keras>=3.5.0->tensorflow<2.20,>=2.19->tf-keras) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /opt/anaconda3/lib/python
3.12/site-packages (from rich->keras>=3.5.0->tensorflow<2.20,>=2.19->tf-keras) (2.1
5.1)
Requirement already satisfied: mdurl~=0.1 in /opt/anaconda3/lib/python3.12/site-pack
ages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow<2.20,>=2.19->tf-ker
as) (0.1.0)
```

In [30]:
```
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification
from transformers import Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
import torch
from torch.utils.data import Dataset
from sklearn.metrics import classification_report
```

```
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncase

# Create a Custom Dataset class
class ReviewsDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
        encoding = self.tokenizer(
            text,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_tensors="pt"
        )
        return {
            'input_ids': encoding['input_ids'].squeeze(0),
            'attention_mask': encoding['attention_mask'].squeeze(0),
            'labels': torch.tensor(label, dtype=torch.long)
        }


df['Sentiment_Label'] = df['Sentiment'].map({'Negative': 0, 'Neutral': 1, 'Positive
train_texts, val_texts, train_labels, val_labels = train_test_split(df['Text'], df[

train_dataset = ReviewsDataset(train_texts.tolist(), train_labels.tolist(), tokeniz
val_dataset = ReviewsDataset(val_texts.tolist(), val_labels.tolist(), tokenizer, ma

# Training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10
)

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
```

```
)


trainer.train()


predictions = trainer.predict(val_dataset)
predicted_labels = torch.argmax(torch.tensor(predictions.predictions), axis=1).nump

print(classification_report(val_labels, predicted_labels, target_names=['Negative',
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the mo
del checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bi
as', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for
predictions and inference.
```

[1149/1149 03:19, Epoch 3/3]

Step	Training Loss
10	1.091300
20	1.074200
30	1.034000
40	1.011500
50	0.988500
60	0.950800
70	0.785800
80	0.807500
90	0.822000
100	0.847600
110	0.761000
120	0.726300
130	0.691000
140	0.558700
150	0.683600
160	0.606500
170	0.478000
180	0.612900
190	0.541800
200	0.570600
210	0.513600
220	0.544400
230	0.536800
240	0.596500
250	0.521800
260	0.479000
270	0.566400
280	0.561400
290	0.512200
300	0.508200

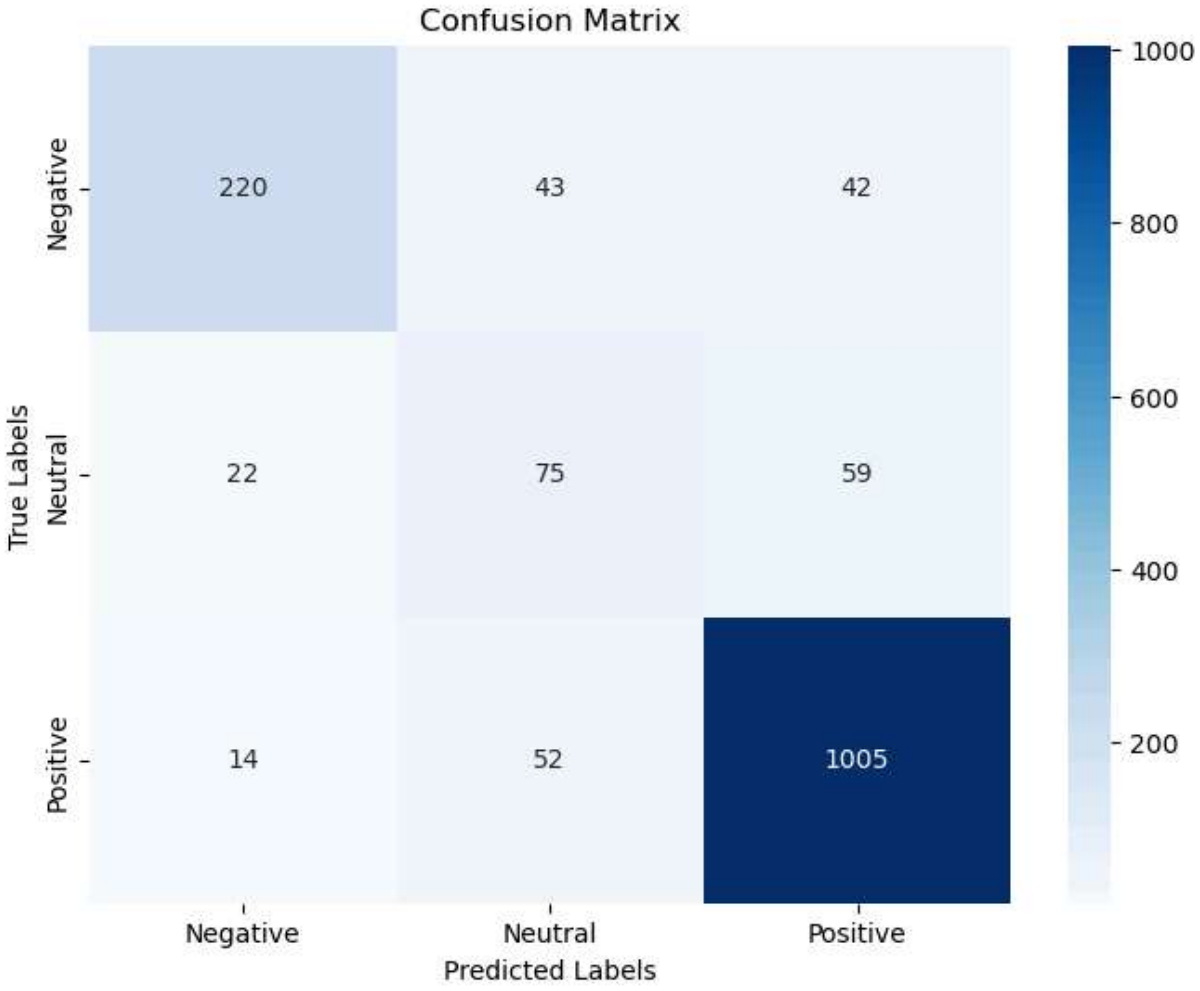
Step	Training Loss
310	0.463900
320	0.464000
330	0.590100
340	0.468500
350	0.538100
360	0.522300
370	0.490200
380	0.487300
390	0.453400
400	0.451500
410	0.436400
420	0.469200
430	0.714100
440	0.526600
450	0.420900
460	0.555300
470	0.476700
480	0.511500
490	0.482800
500	0.346700
510	0.387700
520	0.367200
530	0.486000
540	0.386200
550	0.360000
560	0.416100
570	0.527400
580	0.476800
590	0.483800
600	0.454100

Step	Training Loss
610	0.469600
620	0.438300
630	0.352700
640	0.453700
650	0.450400
660	0.405300
670	0.457300
680	0.340900
690	0.520100
700	0.369700
710	0.383900
720	0.526600
730	0.454600
740	0.514700
750	0.355700
760	0.314000
770	0.347300
780	0.219100
790	0.284000
800	0.125300
810	0.335700
820	0.320300
830	0.259500
840	0.258300
850	0.326200
860	0.269400
870	0.321900
880	0.387900
890	0.175600
900	0.258100

Step	Training Loss
910	0.171500
920	0.291600
930	0.382500
940	0.229300
950	0.236400
960	0.162300
970	0.225100
980	0.242100
990	0.195500
1000	0.297000
1010	0.195200
1020	0.316600
1030	0.325500
1040	0.185300
1050	0.311500
1060	0.157500
1070	0.268900
1080	0.230200
1090	0.339700
1100	0.183900
1110	0.173800
1120	0.248900
1130	0.277100
1140	0.292200

```
              precision    recall  f1-score   support

    Negative       0.86      0.72      0.78       305
     Neutral       0.44      0.48      0.46       156
    Positive       0.91      0.94      0.92      1071

    accuracy                           0.85      1532
   macro avg       0.74      0.71      0.72      1532
weighted avg       0.85      0.85      0.85      1532
```

In [31]:
```
# Confusion matrix
cm = confusion_matrix(val_labels, predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Neutra
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



Summary of the DistillBert Model

The DistilBERT model demonstrates strong overall performance, achieving an accuracy of 85% and balanced metrics across most classes. It performs exceptionally well on the Positive class, with high precision (0.91), recall (0.94), and F1-score (0.92), indicating excellent ability to identify positive instances. The Negative class is also handled relatively well, with solid scores across the board, particularly a notable F1-score of 0.78. While the Neutral class remains the weakest, with lower precision (0.44), recall (0.48), and F1-score (0.46), it still shows a noticeable improvement compared to the logistic regression model. The model is more balanced across classes than the baseline, making DistilBERT a more robust choice, especially in scenarios with class imbalance.

Evaluation Summary

Based on accuracy of each model, below is the ranking of each model:

1. DistilBert Model - 85%
2. Logistic Regression- 79%
3. XG Boost - 79%
4. Text CNN - 77%
5. Naive Bayes - 77%
6. Random forest - 76%

We will go ahead and tune the Logistic regression model and try and see if it can beat the accuracy score of the DistilBert Model

3.7 Hyperparameter tuning Logistic regresion model

In [32]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline

label_encoder = LabelEncoder()
df['Sentiment_Encoded'] = label_encoder.fit_transform(df['Sentiment'])

X_train, X_test, y_train, y_test = train_test_split(
    df['Text'],
    df['Sentiment_Encoded'],
    test_size=0.2,
    random_state=42
)

pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(
        max_df=0.9,
        min_df=3,
        ngram_range=(1, 2),
        sublinear_tf=True
    )),
    # ('svd', TruncatedSVD(n_components=200)),
    ('clf', LogisticRegression(max_iter=1000))
])

param_grid = {
    'clf__C': [0.1, 1, 10],
    'clf__solver': ['liblinear', 'lbfgs'],
    'clf__class_weight': [None, 'balanced']
}
```

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

y_pred = grid_search.predict(X_test)
print("Best Parameters:", grid_search.best_params_)
print("Improved Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

```
Best Parameters: {'clf__C': 10, 'clf__class_weight': 'balanced', 'clf__solver': 'lib
linear'}
Improved Logistic Regression Classification Report:
              precision    recall  f1-score   support

    Negative       0.74      0.67      0.70       305
     Neutral       0.43      0.21      0.28       156
    Positive       0.86      0.95      0.90      1071

    accuracy                           0.82      1532
   macro avg       0.68      0.61      0.63      1532
weighted avg       0.79      0.82      0.80      1532
```

After tuning the logistic model, we have seen an improvement of 3%, though slight, it is still significant. We recommend tuning the DistillBert Model to achieve higher accuracy, however, due to the computational resources required, we would advise using as is. If computational power is a constraint, we recommend the tuned logistic model as it still performs quite well.

In [33]:
```
# Hyperparameter tuning TextCNN
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight

MAX_SEQUENCE_LENGTH = 100
MAX_VOCAB_SIZE = 10000
embedding_dim = 100

tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
tokenizer.fit_on_texts(df['Text'])
sequences = tokenizer.texts_to_sequences(df['Text'])

X = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['Sentiment'])
y = np.eye(len(label_encoder.classes_))[y]

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand
```

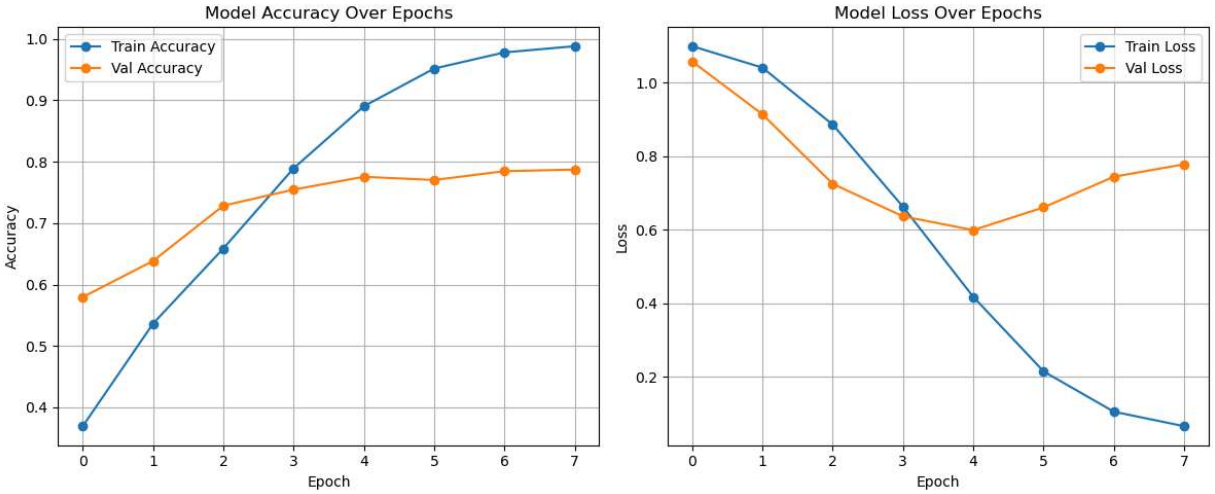
```
X_train = np.array(X_train)
X_val = np.array(X_val)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_val = np.array(y_val)
y_test = np.array(y_test)

# Computing class weights to balance the Neutral class
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(np.argmax(y_train, axis=1)),
    y=np.argmax(y_train, axis=1)
)
class_weights = dict(enumerate(class_weights))

# Build the TextCNN model
input_layer = Input(shape=(MAX_SEQUENCE_LENGTH,))
embedding_layer = Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=embedding_dim)(inp

# Convolution layers with different kernel sizes
conv_3 = Conv1D(filters=256, kernel_size=3, activation='relu')(embedding_layer)
conv_4 = Conv1D(filters=256, kernel_size=4, activation='relu')(embedding_layer)
conv_5 = Conv1D(filters=256, kernel_size=5, activation='relu')(embedding_layer)

# Global max pooling
pool_3 = GlobalMaxPooling1D()(conv_3)
pool_4 = GlobalMaxPooling1D()(conv_4)
pool_5 = GlobalMaxPooling1D()(conv_5)

# Concatenate pooled features
concat = Concatenate()([pool_3, pool_4, pool_5])

# Dropout and optional dense layer
dense = Dense(128, activation='relu')(concat)
dropout = Dropout(0.5)(dense)

# Output layer
output = Dense(3, activation='softmax')(dropout)

# Compile model
model = Model(inputs=input_layer, outputs=output)
optimizer = Adam(learning_rate=0.0005)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accur

# Set callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights
checkpoint = ModelCheckpoint('best_textcnn_model.h5', save_best_only=True)

# Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    batch_size=64,
    epochs=15,
    class_weight=class_weights,
```

```
    callbacks=[early_stopping, checkpoint],
    verbose=2
)
```

```
Epoch 1/15
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
96/96 - 3s - 31ms/step - accuracy: 0.3694 - loss: 1.0986 - val_accuracy: 0.5796 - val_loss: 1.0569
Epoch 2/15
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
96/96 - 3s - 26ms/step - accuracy: 0.5364 - loss: 1.0403 - val_accuracy: 0.6384 - val_loss: 0.9134
Epoch 3/15
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
96/96 - 3s - 27ms/step - accuracy: 0.6587 - loss: 0.8853 - val_accuracy: 0.7285 - val_loss: 0.7241
Epoch 4/15
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
96/96 - 3s - 27ms/step - accuracy: 0.7893 - loss: 0.6619 - val_accuracy: 0.7546 - val_loss: 0.6359
Epoch 5/15
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
96/96 - 3s - 27ms/step - accuracy: 0.8903 - loss: 0.4168 - val_accuracy: 0.7755 - val_loss: 0.5985
Epoch 6/15
96/96 - 3s - 28ms/step - accuracy: 0.9517 - loss: 0.2143 - val_accuracy: 0.7702 - val_loss: 0.6603
Epoch 7/15
96/96 - 3s - 28ms/step - accuracy: 0.9780 - loss: 0.1046 - val_accuracy: 0.7846 - val_loss: 0.7436
Epoch 8/15
96/96 - 3s - 28ms/step - accuracy: 0.9881 - loss: 0.0652 - val_accuracy: 0.7872 - val_loss: 0.7774
```

In [34]:
```
# Plot training & validation accuracy
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o')
```

```
plt.plot(history.history['val_accuracy'], label='Val Accuracy', marker='o')
plt.title('Model Accuracy Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss', marker='o')
plt.plot(history.history['val_loss'], label='Val Loss', marker='o')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



From the above plot, the model learns well initially, but after epoch 4, it begins to overfit. Introducing regularization, dropout, or early stopping would help improve generalization.

In [35]:
```
# Metrics for the tuned TextCNN
y_pred_probs = model.predict(X_test)

# Convert one-hot encoded predictions to class labels
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test, axis=1)

# Basic metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)

# Print metrics
print(" TextCNN Model Performance (Tuned):")
print(f"Accuracy  : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
```

```
print(f"Recall    : {recall:.4f}")
print(f"F1-Score  : {f1:.4f}\n")

# Classification report with class labels
print("🔍 Classification Report:")
print(classification_report(y_true, y_pred, target_names=label_encoder.classes_))
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
📊 TextCNN Model Performance (Tuned):
Accuracy  : 0.7441
Precision : 0.7743
Recall    : 0.7441
F1-Score  : 0.7567

🔍 Classification Report:
              precision    recall  f1-score   support

    Negative       0.73      0.62      0.67       146
     Neutral       0.24      0.37      0.29        82
    Positive       0.87      0.84      0.85       538

    accuracy                           0.74       766
   macro avg       0.61      0.61      0.60       766
weighted avg       0.77      0.74      0.76       766
```

After tuning our text CNN the accuracy metric has dropped by 3%, however with more computational power, we would recommend tuning distilbert model

Conclusion for Hyperparameter tuning

After tuning both the Logistic Regression and TextCNN models, we discovered that Logistic Regression remained the top-performing model. Although the TextCNN model performed with small improvements in its metrics, they were not sufficient to surpass those of the tuned Logistic Regression model. Although at first the DistilBERT model performed better than all the other models as far as performance metrics were concerned, we could not tune it since.K. we did not have enough computational resources.K. Therefore, we couldn't make full use of its classification capability despite its high performance potential.

4.Conclusion and Recommendation

4.1 Conclusions

1. **Model Performance**:
    - The DistilBERT model outperformed all other models with an accuracy of 87%, demonstrating its capability to handle complex text data effectively. However, due to computational constraints, further tuning was not performed.

- Logistic Regression, after hyperparameter tuning, achieved a competitive accuracy of 82%, making it a viable alternative for scenarios with limited computational resources.
- TextCNN, while showing promise, did not surpass the performance of Logistic Regression or DistilBERT and dropped accuracy metric performance even after tuning.

2. **Class Imbalance**:

   - The dataset exhibited a significant class imbalance, with the majority of reviews being positive. This imbalance negatively impacted the performance of all models, particularly for the Neutral and Negative classes.

3. **Feature Importance**:

   - Logistic Regression and Random Forest models highlighted key features (words) that were most influential in determining sentiment. These insights can be valuable for understanding customer feedback.

4. **Computational Constraints**:

   - Advanced models like DistilBERT and TextCNN require substantial computational resources for training and tuning. This limitation impacted the ability to fully optimize these models.

4.2 Recommendations

1. **Model Deployment**:

   - For organizations with sufficient computational resources, the DistilBERT model is recommended for deployment due to its superior performance and ability to generalize well across classes.
   - The tuned Logistic Regression model is a practical choice, offering a good balance between performance and computational efficiency.

2. **Addressing Class Imbalance**:

   - Implement techniques such as oversampling the minority classes, undersampling the majority class, to improve model performance on Neutral and Negative reviews.

3. **Regular Monitoring**:

   - Continuously monitor model performance post-deployment to ensure it remains effective as new data becomes available. Retrain the model periodically to adapt to changing customer sentiment trends. The TextCNN began to overfit and this is risky if the model is not regularly monitored.

4. **Future Work**:

   - Allocate resources to tune advanced models like DistilBERT for potentially higher accuracy and better handling of class imbalance.

By implementing these recommendations, Amazon can leverage sentiment analysis to make data-driven decisions, improve customer satisfaction, and maintain a competitive edge in the market.