American International University-Bangladesh (AIUB)

Faculty of Science & Technology (FST)

Department of Computer Science

Natural Language Processing

Mid-Term Project Report

Summer 2024-2025

Section: A

Group: 03

SL #	Student Name	Student ID	Contribution (%)
1.	Aonyendo Paul Neteish	22-49421-3	25%
2.	Md Ehsanul Haque	22-49370-3	25%
3.	Golam Kibria	22-49507-3	25%
4.	MD Sadman Sakib Shad	22-49501-3	25%

Introduction

This project explores the application of **Natural Language Processing (NLP)** techniques to perform sentiment analysis on text data. The primary goal is to **classify sentiment** by applying the **Multinomial Naïve Bayes algorithm** to a chosen dataset titled **"Spotify User Reviews",** from Kaggle..

The process involves several key steps, beginning with comprehensive data preprocessing, which includes tokenization, case folding, lemmatization, and the removal of punctuation and stop words. Following this, the cleaned text is converted into numerical features using TF-IDF vectorization. A machine learning pipeline is then created to train the Multinomial Naïve Bayes model on the processed data. Finally, the model's performance is rigorously evaluated using standard metrics such as accuracy, F1-score, precision, recall, and a confusion matrix to assess its effectiveness in classifying sentiments.

Dataset Description

The selected dataset titled **"Spotify User Reviews**", is available on Kaggle. The dataset is chosen for a sentiment analysis project in the field of natural language processing (NLP).

Dataset Details

The dataset contains text reviews of the Spotify mobile application, collected from the Google Play Store between January and July 2022. It consists of over 51,000 individual user reviews, each labeled with a sentiment classification.

Key Details:

- Source: Kaggle dataset by alexandrakim2201
- Link: Spotify User Reviews
- Content: User-generated text reviews of Spotify
- Size: Approximately 51,000 entries
- Time Range: January 2022 – July 2022
- Sentiment Labels: Positive or Negative

Reason for Selection (NLP – Sentiment Analysis)

The dataset was selected for sentiment analysis due to the following reasons:

1. Authentic Real-world Text
2. Balanced Sentiment Categories
3. Suitable Dataset Size

4. Domain Relevance
5. Compatibility with Deep Learning Approaches

4. Dataset Attributes

Attribute Name	Description
review	The textual content of the user's review written in free-form natural language.
label	Sentiment classification assigned to each review: either positive or negative.
date range	Reviews collected between January and July 2022.
size	Approximately 51,000 review entries.

The Spotify User Reviews dataset is a comprehensive and relevant data source for sentiment analysis in NLP. Its textual data offers valuable insights into user opinions and emotional tone. Due to its quality, balanced sentiment distribution, and adequate size, it provides a strong foundation for building and evaluating sentiment classification models using machine learning and deep learning techniques.

Project Implementation Details

1. Importing Necessary Libraries

Description: The first step is to import all the required Python libraries for data manipulation, text preprocessing, feature extraction, model training, and evaluation. We use **Pandas** for handling the dataset, **NLTK** and **spaCy** for various NLP tasks like tokenization and lemmatization, **Scikit-learn** for vectorization (TF-IDF) and implementing the Multinomial Naïve Bayes model, and **Matplotlib** for visualizing the results, such as the confusion matrix.

Code:

```
import pandas as pd
import nltk
import string
import re
import spacy
import matplotlib.pyplot as plt
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
print("All neccessary libraries imported")
```

Sample Output:

```
All neccessary libraries imported
```

Code Description: This cell imports essential libraries. pandas is for dataframes, nltk and spacy are for core NLP tasks, sklearn provides tools for machine learning pipelines including vectorization (TfidfVectorizer), model selection (train_test_split), the classification algorithm (MultinomialNB), and performance metrics. matplotlib is used for plotting graphs.

Download important language processing resources

Description: The task here is to **download important language processing resources** using the NLTK library in Python. These resources are needed in order to perform various Natural Language Processing (NLP) tasks such as tokenizing text, removing stopwords, and finding word meanings or lemmas. We solve this task by calling the nltk.download() function with the names of each resource we need.

Code:

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

Sample Output:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Code Description:

- nltk.download('punkt') → downloads tokenizer for splitting text.
- nltk.download('punkt_tab') → another format of tokenizer data.
- nltk.download('wordnet') → downloads dictionary for meanings and lemmatization.
- nltk.download('stopwords') → downloads list of common words to remove.

2. Loading the Dataset

Description: Here, we load the Spotify reviews dataset from a CSV file named DATASET.csv into a Pandas DataFrame. We then display the first few rows and the shape of the DataFrame to get an initial understanding of its structure and size.

Code:

```
df = pd.read_csv('DATASET.csv', encoding='utf-8')
print(df.head())
print(f"Dataset shape: {df.shape}")
```

Sample Output:

```
                                              Review     label
0  Great music service, the audio is high quality...  POSITIVE
1  Please ignore previous negative rating. This a...  POSITIVE
2  This pop-up "Get the best Spotify experience o...  NEGATIVE
3    Really buggy and terrible to use as of recently  NEGATIVE
4  Dear Spotify why do I get songs that I didn't ...  NEGATIVE
Dataset shape: (52702, 2)
```

Code Description: The pd.read_csv() function reads the data into a DataFrame called df. df.head() shows the first 5 rows, and df.shape returns a tuple representing the dimensionality (rows, columns) of the DataFrame.

3. Data Cleaning and Preprocessing

Description: Raw text data is often noisy. This stage involves several sequential steps to clean and standardize the reviews to prepare them for analysis.

3.1. Handling Missing Values & Initial Cleaning

Description: We first remove any rows with missing values (NaN) to prevent errors. Then, a cleaning function is applied to remove non-ASCII characters, numbers, URLs, and user mentions/hashtags, which are generally not useful for sentiment analysis.

Code:

```
df.dropna(how='any', inplace=True)

reviews = df['Review']
sentiments = df['label']
print(f"After dropping NaN: {df.shape}")
```

Sample Output:

```
After dropping NaN: (52686, 2)
```

Code:

```
def cleanText(text):

    text = str(text)

    text = text.encode('ascii', 'ignore').decode('ascii')
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'http\S+|www\S+|https\S+', '', text,
flags=re.MULTILINE)
    text = re.sub(r'@\w+|#\w+', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

reviews = reviews.apply(cleanText)
print(reviews.head())
```

Sample Output:

```
0    Great music service, the audio is high quality...
1    Please ignore previous negative rating. This a...
2    This pop-up "Get the best Spotify experience o...
3      Really buggy and terrible to use as of recently
4    Dear Spotify why do I get songs that I didn't ...
Name: Review, dtype: object
```

Code Description: df.dropna() removes null entries. The cleanText function uses regular expressions (re.sub) to systematically remove unwanted patterns from the text. This function is then applied to every review in the reviews Series.

3.2. Tokenization

Description: Tokenization is the process of breaking down the text into individual words or "tokens". This is a fundamental step for further processing.

Code:

```
def tokenization(text):
```

```
    tokens = word_tokenize(str(text))
    return tokens

reviews = reviews.apply(tokenization)
print(reviews.head(5))
```

Sample Output:

```
0    [Great, music, service, ,, the, audio, is, hig...
1    [Please, ignore, previous, negative, rating, ....
2    [This, pop-up, ``, Get, the, best, Spotify, ex...
3    [Really, buggy, and, terrible, to, use, as, of...
4    [Dear, Spotify, why, do, I, get, songs, that, ...
Name: Review, dtype: object
```

Code Description: We use the word_tokenize function from the NLTK library to split each review string into a list of tokens.

3.3. Case Folding

Description: To ensure that words like "Great" and "great" are treated as the same word, we convert all text to lowercase. This reduces the vocabulary size and simplifies the analysis.

Code:

```
def case_folding(tokens):
    cf = [word.lower() for word in tokens]
    return cf

reviews = reviews.apply(case_folding)
print(reviews.head(5))
```

Sample Output:

```
0    [great, music, service, ,, the, audio, is, hig...
1    [please, ignore, previous, negative, rating, ....
2    [this, pop-up, ``, get, the, best, spotify, ex...
3    [really, buggy, and, terrible, to, use, as, of...
4    [dear, spotify, why, do, i, get, songs, that, ...
Name: Review, dtype: object
```

Code Description: This function iterates through each token in a list and applies the .lower() method to convert it to lowercase.

3.4. Stemming

Description: Stemming reduces words to their root or base form. For example, "running", "ran", and "runner" might all be stemmed to "run". We use the Porter Stemmer for this task.

Code:

```
stemmer = PorterStemmer()

def stemming(tokens):
```

```
    stemmed = [stemmer.stem(words) for words in tokens]
    return stemmed

reviews = reviews.apply(stemming)
print(reviews.head(5))
```

Sample Output:

```
0    [great, music, servic, ,, the, audio, is, high...
1    [pleas, ignor, previou, neg, rate, ., thi, app...
2    [thi, pop-up, ``, get, the, best, spotifi, exp...
3    [realli, buggi, and, terribl, to, use, as, of,...
4    [dear, spotifi, whi, do, i, get, song, that, i...
Name: Review, dtype: object
```

Code Description: An instance of PorterStemmer is created. The stemming function applies the .stem() method to each word in the token list.

3.5. Lemmatization

Description: Lemmatization is a more advanced technique than stemming. It reduces words to their dictionary form, or "lemma", considering the context. For example, "better" becomes "good". We use the spaCy library for this.

Code:

```
nlp = spacy.load("en_core_web_sm")

def lemmatization(tokens):
    doc = nlp(" ".join(tokens))
    lemmatized = [word.lemma_ for word in doc]
    return lemmatized

reviews = reviews.apply(lemmatization)
print(reviews.head(5))
```

Sample Output:

```
0    [great, music, servic, ,, the, audio, be, high...
1    [pleas, ignor, previou, neg, rate, ., thi, app...
2    [thi, pop, -, up, `, `, get, the, good, spotif...
3    [realli, buggi, and, terribl, to, use, as, of,...
4    [dear, spotifi, whi, do, I, get, song, that, I...
Name: Review, dtype: object
```

Code Description: We load a small English model from spaCy. The lemmatization function joins the tokens back into a string, processes it with the spaCy model, and extracts the lemma for each word.

3.6. Punctuation and Stop Words Removal

Description: Punctuation is removed as it typically doesn't carry sentiment. Stop words (common words like "the", "a", "is") are also removed because they appear frequently but add little meaning.

Code:

```
def pr(tokens):
    trans = str.maketrans("", "", string.punctuation)
    text = " ".join(tokens)
    rPunc = text.translate(trans)
    puncRe = rPunc.strip().split()
    return puncRe

reviews = reviews.apply(pr)
print(reviews.head())
```

Sample Output:

```
0    [great, music, servic, the, audio, be, high, q...
1    [pleas, ignor, previou, neg, rate, thi, app, b...
2    [thi, pop, up, get, the, good, spotifi, experi...
3    [realli, buggi, and, terribl, to, use, as, of,...
4    [dear, spotifi, whi, do, I, get, song, that, I...
Name: Review, dtype: object
```

Code:

```
stopWord = set(stopwords.words("english"))
def swr(tokens):
    rStopWord = [word for word in tokens if word not in stopWord and
len(word)>1]
    return rStopWord

reviews = reviews.apply(swr)
print(reviews.head(5))
```

Sample Output:

```
0    [great, music, servic, audio, high, qualiti, a...
1    [pleas, ignor, previou, neg, rate, thi, app, s...
2    [thi, pop, get, good, spotifi, experi, android...
3                [realli, buggi, terribl, use, recent]
4    [dear, spotifi, whi, get, song, put, playlist,...
Name: Review, dtype: object
```

Code Description: The pr function removes punctuation using str.maketrans. The swr function filters out any token that is present in NLTK's English stop words list. We also remove any single-character tokens.

4. Feature Extraction (Vector Semantics)

Description: Machine learning models require numerical input. We convert the cleaned text data into a numerical format using the **Term Frequency-Inverse Document Frequency (TF-IDF)** method. TF-IDF reflects how important a word is to a document in a collection, effectively turning text into a matrix of numerical features.

Code:

```
def joined(tokens):
    joined_tokens = " ".join(tokens)
    return joined_tokens
```

```
reviews = reviews.apply(joined)
print(reviews.head())
print(f"Sample processed review: {reviews.iloc[0]}")
```

Sample Output:

```
0    great music servic audio high qualiti app easi...
1    pleas ignor previou neg rate thi app super gre...
2    thi pop get good spotifi experi android annoy ...
3                      realli buggi terribl use recent
4    dear spotifi whi get song put playlist whi shu...
Name: Review, dtype: object
Sample processed review: great music servic audio high qualiti app easi
use also veri quick friendli support
```

Code:

```
mask = reviews.str.len() > 0

reviews = reviews[mask]
sentiments = sentiments[mask]
print(f"After filtering empty reviews: {len(reviews)}")

tfidf_vector = TfidfVectorizer(smooth_idf=False, max_features=5000,
min_df=2)
tfidf_matrix = tfidf_vector.fit_transform(reviews)

df_tfidf = pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf_vector.get_feature_names_out())
print(df_tfidf.head())
print(df_tfidf.shape)
print(f"Sample feature names: {list(df_tfidf.columns[:10])}")
```

Sample Output:

```
After filtering empty reviews: 52665
    aa  aap  aaron   ab  abandon  abd  abil  abl  abov  abroad  ...
youtub  \
0  0.0  0.0    0.0  0.0      0.0  0.0   0.0  0.0   0.0     0.0  ...
0.0
1  0.0  0.0    0.0  0.0      0.0  0.0   0.0  0.0   0.0     0.0  ...
0.0
2  0.0  0.0    0.0  0.0      0.0  0.0   0.0  0.0   0.0     0.0  ...
0.0
3  0.0  0.0    0.0  0.0      0.0  0.0   0.0  0.0   0.0     0.0  ...
0.0
4  0.0  0.0    0.0  0.0      0.0  0.0   0.0  0.0   0.0     0.0  ...
0.0
```

```
   youtube  youv  ypu   yr   yt  ytube  zero  zone  zoom
0      0.0   0.0  0.0  0.0  0.0    0.0   0.0   0.0   0.0
1      0.0   0.0  0.0  0.0  0.0    0.0   0.0   0.0   0.0
```

```
2      0.0   0.0  0.0  0.0  0.0    0.0   0.0   0.0   0.0
3      0.0   0.0  0.0  0.0  0.0    0.0   0.0   0.0   0.0
4      0.0   0.0  0.0  0.0  0.0    0.0   0.0   0.0   0.0

[5 rows x 5000 columns]
(52665, 5000)
Sample feature names: ['aa', 'aap', 'aaron', 'ab', 'abandon', 'abd',
'abil', 'abl', 'abov', 'abroad']
```

Code Description: First, the lists of tokens are joined back into single strings. We then initialize TfidfVectorizer with a maximum of 5000 features and a minimum document frequency of 2. fit_transform learns the vocabulary and inverse document frequency, and transforms the text data into a TF-IDF matrix.

5. Model Training and Evaluation

Description: With the data prepared, we split it into training (80%) and testing (20%) sets. We then train a **Multinomial Naïve Bayes** classifier on the training data and evaluate its performance on the unseen test data using metrics like accuracy, precision, recall, and F1-score.

Code:

```
x = df_tfidf
y = sentiments

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
model = MultinomialNB()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Precision: ", precision_score(y_test, y_pred,
average='weighted'))
print("Recall: ", recall_score(y_test, y_pred, average='weighted'))
print("F1 Score: ", f1_score(y_test, y_pred, average='weighted'))
```

Sample Output:

```
Accuracy:  0.8598689831956707
Precision:  0.85991707240068
Recall:  0.8598689831956707
F1 Score:  0.8594070013232645
```

Code Description: train_test_split divides the features (x) and labels (y). We create an instance of MultinomialNB and train it with model.fit(). model.predict() makes predictions on the test set. Finally, we print the performance scores by comparing the predicted labels (y_pred) with the actual labels (y_test).

6. Confusion Matrix Visualization

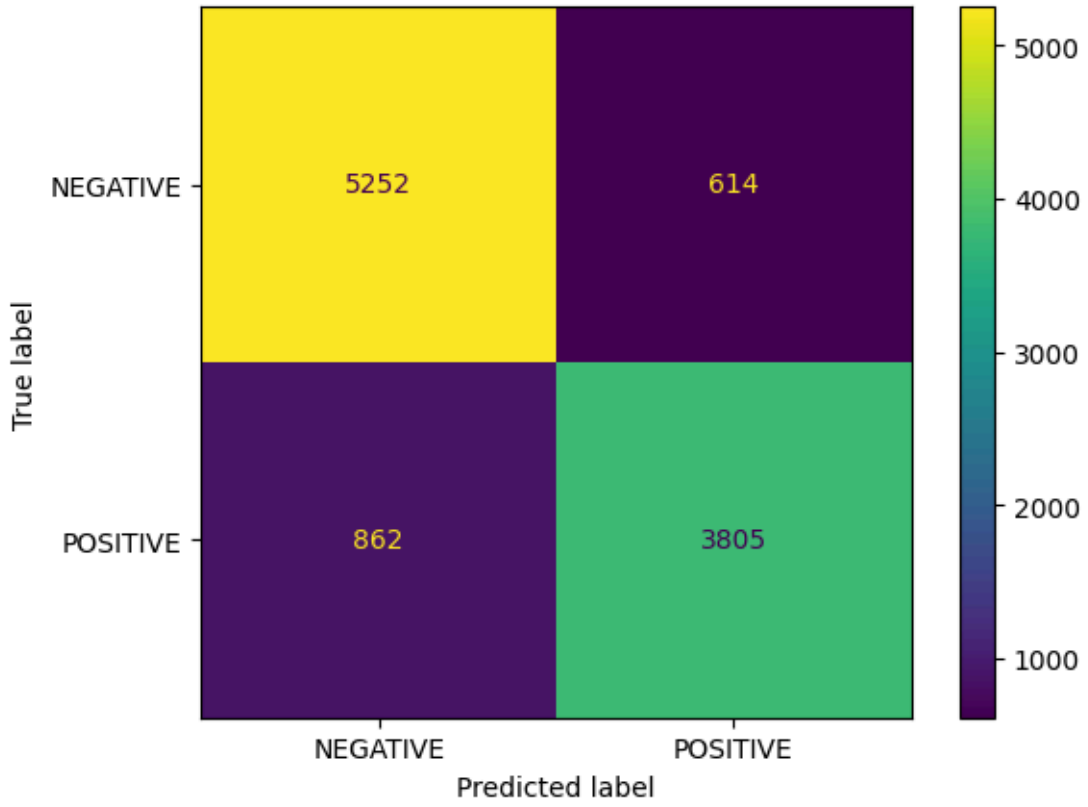
Description: A confusion matrix provides a detailed breakdown of the model's performance, showing the number of correct and incorrect predictions for each class (Positive vs. Negative). We visualize this matrix as a heatmap for better interpretation.

Code:

```
confusion_matrix_result = confusion_matrix(y_test, y_pred)

display =
ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_result,
display_labels=model.classes_)
display.plot()
plt.show()
```

Sample Output:



Code Description: We generate the confusion matrix using confusion_matrix from Scikit-learn. The ConfusionMatrixDisplay object is then used with matplotlib to plot the matrix, clearly showing true positives, true negatives, false positives, and false negatives.

Conclusion

In this project, we successfully developed a sentiment analysis system using Natural Language Processing (NLP) techniques. By collecting review data, preprocessing the text, extracting features using TF-IDF, and training a Multinomial Naive Bayes classifier, we were able to classify reviews into positive or negative sentiments with reliable performance. The evaluation metrics such as accuracy, precision, recall, and F1-score indicate that the model is effective and can serve as a useful tool for automated sentiment detection. Overall, this

project demonstrates how machine learning and NLP can be applied to real-world text data to derive meaningful insights and support decision-making.

Limitations

While the model performed well, there are several limitations that should be addressed in future work:

- **Limited Dataset**: The dataset used was relatively small and may not represent all types of language or contexts found in real-world reviews.
- **Imbalanced Classes**: If the dataset had more positive than negative reviews (or vice versa), it could bias the model's predictions.
- **Basic Preprocessing**: The text preprocessing was simple and did not capture deeper linguistic structures such as sarcasm, idioms, or context-specific meaning.
- **Model Simplicity**: The Naive Bayes classifier assumes word independence and may not fully capture complex relationships in the text compared to more advanced models like LSTM or BERT.
- **Domain Specific**: The model is trained on a specific type of review data and may not generalize well to other domains without retraining.

Project Code

```
# -*- coding: utf-8 -*-

"""spotify_sentiment_analysis_fixed.ipynb

Automatically generated by Colab.

Original file is located at

    https://colab.research.google.com/drive/1oNYSi2t43OX0CzUoFW4VyoP9PJ4q3s7C
"""

import pandas as pd

import nltk

import string

import re

import spacy

import matplotlib.pyplot as plt

from nltk.tokenize import word_tokenize

from nltk.stem import PorterStemmer

from nltk.corpus import stopwords

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
```

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, ConfusionMatrixDisplay


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


df = pd.read_csv('DATASET.csv', encoding='utf-8')
print(df.head())
print(f"Dataset shape: {df.shape}")


df.dropna(how='any', inplace=True)


reviews = df['Review']
sentiments = df['label']
print(f"After dropping NaN: {df.shape}")


def cleanText(text):
    text = str(text)


    text = text.encode('ascii', 'ignore').decode('ascii')
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'@\w+|#\w+', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text


reviews = reviews.apply(cleanText)
```

```
print(reviews.head())

def tokenization(text):
    tokens = word_tokenize(str(text))
    return tokens

reviews = reviews.apply(tokenization)
print(reviews.head(5))

def case_folding(tokens):
    cf = [word.lower() for word in tokens]
    return cf

reviews = reviews.apply(case_folding)
print(reviews.head(5))

stemmer = PorterStemmer()

def stemming(tokens):
    stemmed = [stemmer.stem(words) for words in tokens]
    return stemmed

reviews = reviews.apply(stemming)
print(reviews.head(5))

nlp = spacy.load("en_core_web_sm")

def lemmatization(tokens):
    doc = nlp(" ".join(tokens))
    lemmatized = [word.lemma_ for word in doc]
```

```
    return lemmatized

reviews = reviews.apply(lemmatization)
print(reviews.head(5))

def pr(tokens):
    trans = str.maketrans("", "", string.punctuation)
    text = " ".join(tokens)
    rPunc = text.translate(trans)
    puncRe = rPunc.strip().split()
    return puncRe

reviews = reviews.apply(pr)
print(reviews.head())

stopWord = set(stopwords.words("english"))
def swr(tokens):
    rStopWord = [word for word in tokens if word not in stopWord and len(word)>1]
    return rStopWord

reviews = reviews.apply(swr)
print(reviews.head(5))

def joined(tokens):
    joined_tokens = " ".join(tokens)
    return joined_tokens

reviews = reviews.apply(joined)
print(reviews.head())
print(f"Sample processed review: {reviews.iloc[0]}")
```

```
mask = reviews.str.len() > 0
reviews = reviews[mask]
sentiments = sentiments[mask]
print(f"After filtering empty reviews: {len(reviews)}")

tfidf_vector = TfidfVectorizer(smooth_idf=False, max_features=5000, min_df=2)
tfidf_matrix = tfidf_vector.fit_transform(reviews)

df_tfidf = pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf_vector.get_feature_names_out())
print(df_tfidf.head())
print(df_tfidf.shape)
print(f"Sample feature names: {list(df_tfidf.columns[:10])}")

x = df_tfidf
y = sentiments

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model = MultinomialNB()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Precision: ", precision_score(y_test, y_pred, average='weighted'))
print("Recall: ", recall_score(y_test, y_pred, average='weighted'))
print("F1 Score: ", f1_score(y_test, y_pred, average='weighted'))

confusion_matrix_result = confusion_matrix(y_test, y_pred)
```

```
display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_result,
display_labels=model.classes_)

display.plot()

plt.show()
```