**EC3066D Artificial Intelligence: Theory and Practice**

# NLP ASSIGNMENT

**ABHINAV RAJ V**

**B210759EC**

# Procedure

**1. Data Collection:**

- Obtain Twitter data from nltk, financial reports, emotional data from CSV files, product reviews from text file.
- Extract text data from each dataset, ensuring a wide range of textual content for comprehensive sentiment analysis.

**2. Exploratory Data Analysis (EDA):**

- Visualize the distribution of sentiments across different datasets, along with other relevant attributes such as tweet lengths, word counts, emotional expressions, and product ratings.
- Construct visualizations such as word clouds, bar plots, and histograms to identify common words and sentiments expressed within the dataset.

**3. Data Preprocessing and Feature Engineering:**

- Clean the text data by removing noise such as URLs, special characters, punctuation, and irrelevant symbols.
- Tokenize the text into individual words and remove stopwords to reduce noise and improve model performance.
- Perform part-of-speech tagging to analyze the grammatical structure of the text and extract meaningful features.
- Lemmatize words to reduce them to their base form, aiding in feature extraction and reducing dimensionality.
- Convert the preprocessed text into numerical features using techniques such as TF-IDF vectorization, preparing it for model training.

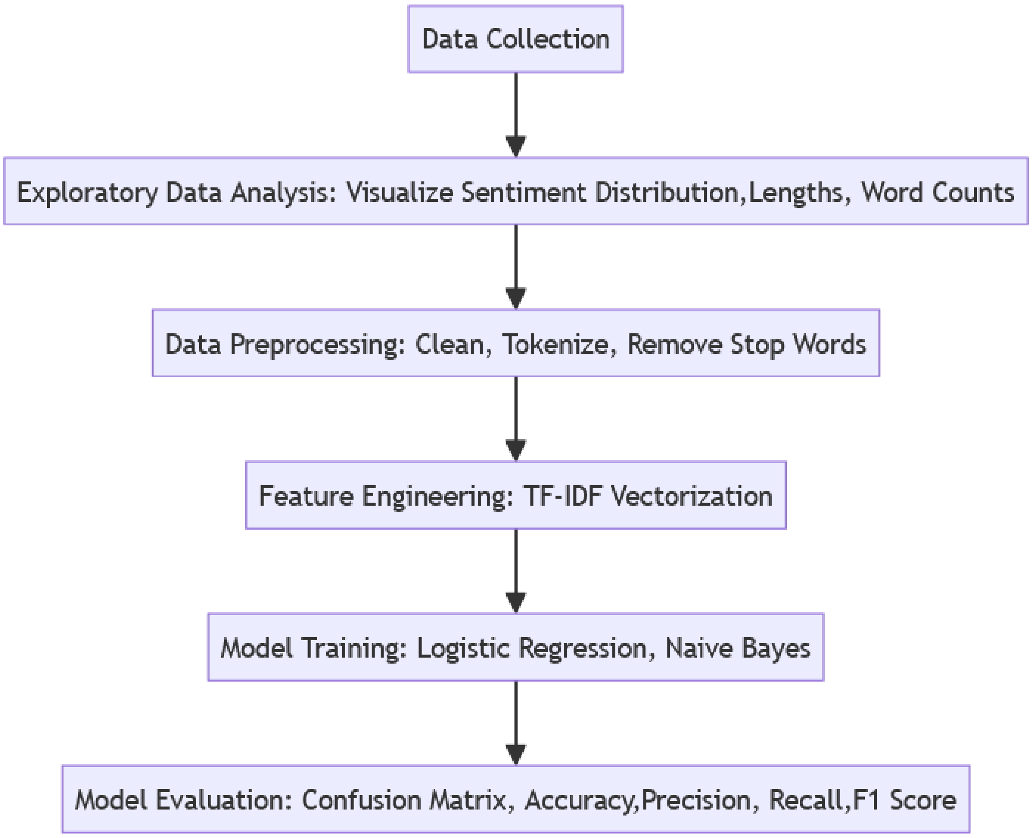
**4. Model Training and Evaluation:**

- Split the dataset into training and testing sets using techniques like train-test split if the data is present as a single file.
- Train the sentiment analysis models Logistic regression, Naive Bayes on the training data.
- Evaluate model performance using a variety of metrics including accuracy, precision, recall, F1-score, and area under the ROC curve (AUC).
- Visualize model evaluation results using confusion matrices, ROC curves, and precision-recall curves to gain insights into model behavior and performance.

## Terminologies

1. **Tokenization**: Tokenization is the process of breaking down a text into smaller units called tokens. These tokens could be words, phrases, or even individual characters, depending on the level of granularity required for analysis. Tokenization is a fundamental step in natural language processing (NLP) tasks as it enables further analysis of text data by treating each token as a separate entity.
2. **Lemmatization**: Lemmatization is the process of reducing words to their base or root form, known as a lemma. Unlike stemming, which simply chops off prefixes or suffixes to derive the root form (stem), lemmatization considers the context of the word and morphological analysis to ensure that the resulting lemma is a valid word. For example, the lemma of "running" is "run", and the lemma of "better" is "good".
3. **POS Tagging (Part-of-Speech Tagging)**: POS tagging is the process of assigning grammatical tags to each word in a sentence based on its part of speech, such as noun, verb, adjective, adverb, etc. These tags provide information about the syntactic role of each word in the sentence, which is essential for many NLP tasks, including parsing, information extraction, and sentiment analysis.
4. **Removing Stop Words**: Stop words are common words that are often filtered out during text preprocessing because they typically do not carry significant meaning for analysis. Examples of stop words include "the", "is", "and", "in", etc. Removing stop words helps reduce noise in the text data and improves the efficiency of downstream NLP tasks by focusing on content-bearing words.
5. **Logistic Regression Model**: Logistic regression is a statistical method used for binary classification tasks, where the outcome variable is categorical and has only two possible outcomes (e.g., yes/no, true/false, spam/not spam). It models the probability that a given input belongs to a particular class using a logistic (sigmoid) function. In the context of sentiment analysis, logistic regression can be used to predict the sentiment (positive/negative) of a piece of text based on its features.
6. **Naive Bayes Model**: Naive Bayes is a probabilistic machine learning model based on Bayes' theorem with the "naive" assumption of independence between features. Despite its simplicity, Naive Bayes often performs well in text classification tasks, including sentiment analysis. It calculates the probability of a class (e.g., positive sentiment) given a set of features (e.g., words in a text) and selects the class with the highest probability as the predicted class.

## Flowchart


Data Collection
Exploratory Data Analysis: Visualize Sentiment Distribution,Lengths, Word Counts
Data Preprocessing: Clean, Tokenize, Remove Stop Words
Feature Engineering: TF-IDF Vectorization
Model Training: Logistic Regression, Naive Bayes
Model Evaluation: Confusion Matrix, Accuracy,Precision, Recall,F1 Score

# Twitter Samples

## Importing Necessary Libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import textblob
import nltk
from nltk.corpus import twitter_samples
import wordcloud
from nltk.probability import FreqDist
```

In [2]:
```
from textblob import TextBlob
from nltk.stem.wordnet import WordNetLemmatizer
import re
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
import string
import emoji
```

In [3]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, fbeta_score
```

## Downloading Required data

In [4]:
```
nltk.download('twitter_samples')
```

```
[nltk_data] Downloading package twitter_samples to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
```

Out[4]: True

In [5]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[5]: True

In [6]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[6]: True

# Preparing the Data Set

In [7]:
```
positive_tweets = twitter_samples.strings('positive_tweets.json')
negative_tweets = twitter_samples.strings('negative_tweets.json')
```

In [8]:
```
print("Positive Tweets:")
for tweet in positive_tweets[:5]:
    print(tweet)
print("\nNegative Tweets:")
for tweet in negative_tweets[:5]:
    print(tweet)
```

```
Positive Tweets:
#FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this week :)
@Lamb2ja Hey James! How odd :/ Please call our Contact Centre on 02392441234 and we will be able to assist you :) Many thanks!
@DespiteOfficial we had a listen last night :) As You Bleed is an amazing track. When are you in Scotland?!
@97sides CONGRATS :)
yeaaaah yippppy!!!  my accnt verified rqst has succeed got a blue tick mark on my fb profile :) in 15 days

Negative Tweets:
hopeless for tmr :(
Everything in the kids section of IKEA is so cute. Shame I'm nearly 19 in 2 months :(
@Hegelbon That heart sliding into the waste basket. :(
"@ketchBurning: I hate Japanese call him "bani" :( :("

Me too
Dang starting next week I have "work" :(
```

In [9]:
```
df_positive = pd.DataFrame(positive_tweets, columns=['tweet'])
df_positive['sentiment'] = 1  # positive sentiment is 1
df_negative = pd.DataFrame(negative_tweets, columns=['tweet'])
df_negative['sentiment'] = 0  # negative sentiment is 0
df = pd.concat([df_positive, df_negative])
df = df.sample(frac=1).reset_index(drop=True)
```

In [10]:
```
df.head(10)
```

Out[10]:

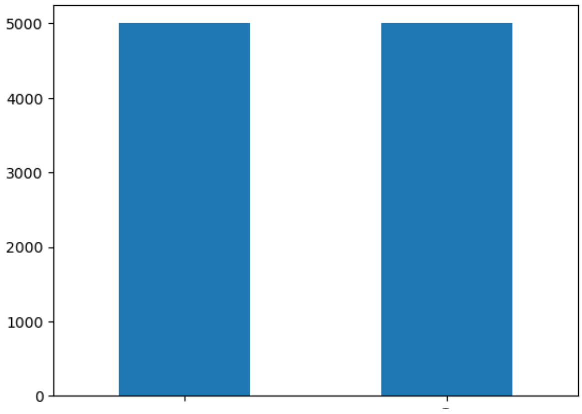
| | tweet | sentiment |
|---|---|---|
| 0 | @KageYashsa Shopping for a bit :p | 1 |
| 1 | my layout doesn't match but it's the closest I... | 0 |
| 2 | Lions agains Otani, 3-0 down already at 3rd bo... | 0 |
| 3 | @AldiUSA love your store!!! Best chocolate sel... | 1 |
| 4 | maroon cocktail dresses http://t.co/loj5YzNRwu... | 1 |
| 5 | Ah Millz askies :( "@_Millzxy3D: so you come b... | 0 |
| 6 | @Inugamikun Wub Cerbchan? :D | 1 |
| 7 | @NiallOfficial gn love u see u in 2 days :) | 1 |
| 8 | Craving for Banana Crumble McFlurry and Fries :( | 0 |
| 9 | @NefariousBella9 @laurenkatebooks @Fallen_Seri... | 1 |

## Exploratory Data Analysis

In [11]:
```
print(df['sentiment'].value_counts())
```

```
1    5000
0    5000
Name: sentiment, dtype: int64
```

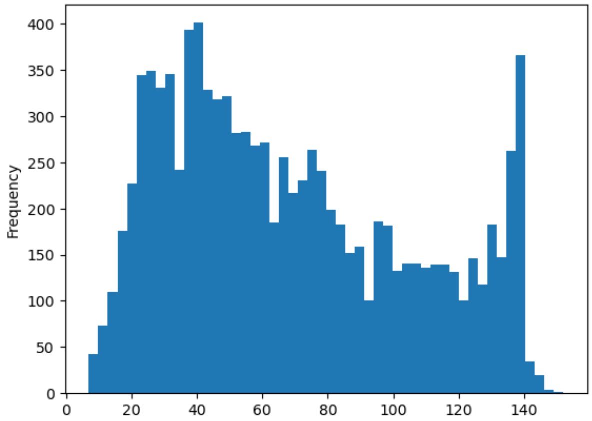
In [12]:
```
df['sentiment'].value_counts().plot(kind='bar')
plt.show()
```



## Tweet Length

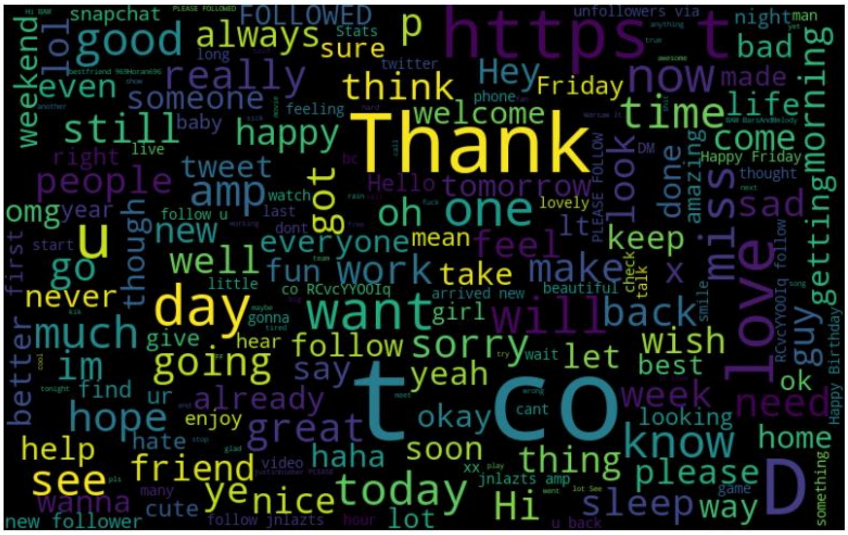
```
In [13]: df['length'] = df['tweet'].apply(len)
         print(df['length'].describe())
         df['length'].plot(kind='hist', bins=50)
         plt.show()
```

```
count    10000.000000
mean        68.537700
std         37.138461
min          7.000000
25%         37.000000
50%         61.000000
75%         97.000000
max        152.000000
Name: length, dtype: float64
```




## Visual Representation of Most Common Words in Tweets

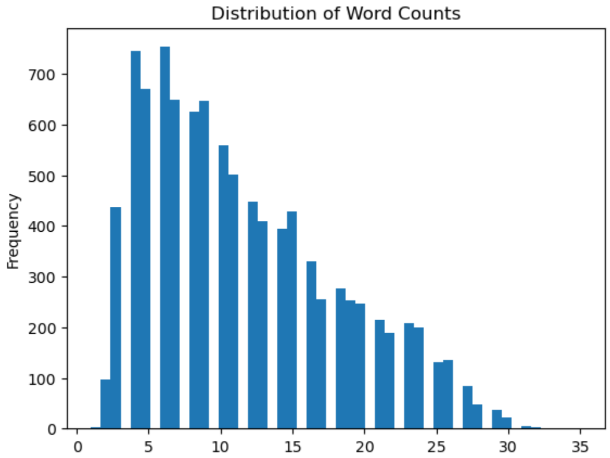
```
In [14]: from wordcloud import WordCloud
         all_words = ' '.join(df['tweet'])
         wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_words)
         plt.figure(figsize=(10, 7))
         plt.imshow(wordcloud, interpolation="bilinear")
         plt.axis('off')
         plt.show()
```



## Word Count in Tweet

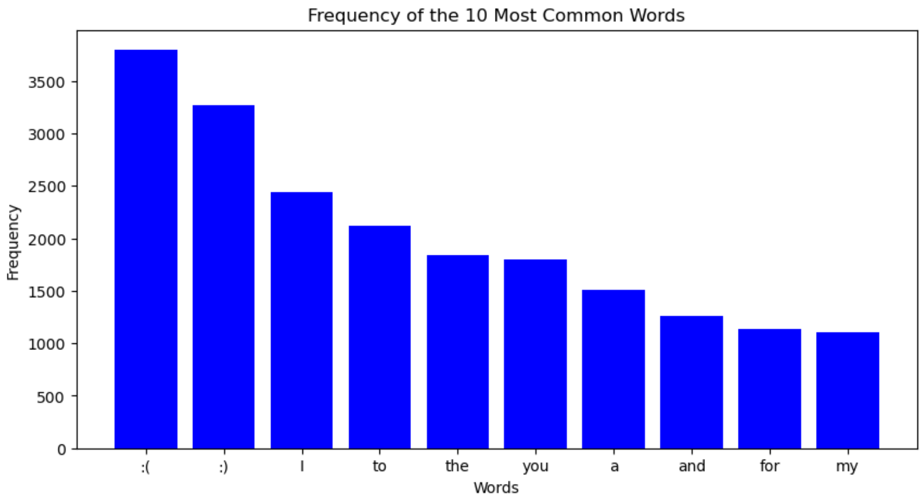
```
In [15]: df['word_count'] = df['tweet'].apply(lambda x: len(str(x).split()))
         print(df['word_count'].describe())
         df['word_count'].plot(kind='hist', bins=50)
         plt.title('Distribution of Word Counts')
         plt.show()
```

```
count    10000.000000
mean        11.639700
std          6.580275
min          1.000000
25%          6.000000
50%         10.000000
75%         16.000000
max         35.000000
Name: word_count, dtype: float64
```




## Most Common Words in Tweets

```
In [16]: from collections import Counter
         all_words = [word for tweet in df['tweet'] for word in tweet.split()]
         word_counts = Counter(all_words)
         common_words = word_counts.most_common(10)
         words, counts = zip(*common_words)
         plt.figure(figsize=(10, 5))
         plt.bar(words, counts, color='b')
         plt.xlabel('Words')
         plt.ylabel('Frequency')
         plt.title('Frequency of the 10 Most Common Words')
         plt.show()
```



## Preprocessing

### Cleaning the Text

In [17]:
```
def clean_text(text):

    text = str(text).lower()

    # Remove Twitter handles
    text = re.sub('@\w+', '', text)

    # Remove text in square brackets
    text = re.sub('\[.*?\]', '', text)

    # Remove URLs
    text = re.sub('https?://\S+|www\.\S+', '', text)

    # Remove HTML tags
    text = re.sub('<.*?>+', '', text)

    # Remove punctuation
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)

    # Remove new line characters
    text = re.sub('\n', '', text)

    # Remove words that contain numbers
    text = re.sub('\w*\d\w*', '', text)

    # Convert emojis to words
    text = emoji.demojize(text, delimiters=(" ", " "))

    return text
```

In [18]:
```
print("Data before preprocessing")
df['tweet'].head(5)
```

```
Data before preprocessing
```

Out[18]:
```
0                      @KageYashsa Shopping for a bit :p
1     my layout doesn't match but it's the closest I...
2     Lions agains Otani, 3-0 down already at 3rd bo...
3     @AldiUSA love your store!!! Best chocolate sel...
4     maroon cocktail dresses http://t.co/Ioj5YzNRwu...
Name: tweet, dtype: object
```

In [19]:
```
df['cleaned_tweet'] = df['tweet'].apply(clean_text)
print("Data after preprocessing")
print(df['cleaned_tweet'].head(5))
```

```
Data after preprocessing
0                                  shopping for a bit p
1     my layout doesnt match but its the closest i c...
2     lions agains otani  down already at  bottom no...
3            love your store best chocolate selections
4           maroon cocktail dresses  mididresses for
Name: cleaned_tweet, dtype: object
```

### Tokenization

In [20]:
```
df['tokenized_tweet']=df['cleaned_tweet'].apply(word_tokenize)
print("Data after tokenization")
df['tokenized_tweet'].head(5)
```

```
Data after tokenization
```

Out[20]:
```
0                            [shopping, for, a, bit, p]
1     [my, layout, doesnt, match, but, its, the, clo...
2     [lions, agains, otani, down, already, at, bott...
3      [love, your, store, best, chocolate, selections]
4         [maroon, cocktail, dresses, mididresses, for]
Name: tokenized_tweet, dtype: object
```

## Removing Stop Words

In [22]:
```
stop_words = set(stopwords.words('english'))
print("The stop words are\n",stop_words)
```

```
The stop words are
 {'are', 'your', 'theirs', 'as', 'being', "mightn't", 'whom', 'myself', 'up', 's', 'itself', 'you', 'themselves', "that'll
", 'some', 'those', 'hasn', 'and', 'her', 'most', 'haven', 'yourself', 'further', 'weren', "wouldn't", 'didn', 'after', 'h
e', 'him', 'ma', 'ours', 'been', 'o', 'can', 'ourselves', 'off', "hadn't", 'through', "hasn't", 'these', 'my', 'too', 'him
self', 'have', 'me', 're', 'our', 'what', 'is', 'why', 'just', 'the', 'isn', 'needn', 'over', 'if', 'wouldn', 'by', 'nor',
'while', 'won', 'doesn', 'same', 'shan', 'which', 'about', 'mightn', 'on', 'each', 'will', 'a', 'an', 'such', 'she', 'must
n', 'against', 'be', 'has', 've', 'out', "needn't", 'yourselves', "should've", "doesn't", "couldn't", 'all', 'in', 'before
', 'couldn', 'own', 'his', 'having', 'down', 'there', 'y', 'to', 'am', "shouldn't", 't', 'between', 'during', "weren't", '
was', "you'd", 'very', "didn't", 'm', 'then', 'but', 'until', 'both', 'wasn', "you've", "aren't", 'any', 'do', 'd', 'had',
"mustn't", 'here', 'hers', 'than', "it's", 'were', 'll', 'yours', 'shouldn', "she's", 'for', 'from', 'aren', 'they', 'hadn
', "you'll", 'did', 'under', "won't", "don't", 'its', "haven't", 'above', 'once', 'them', 'not', 'below', 'that', 'more',
'i', 'only', 'does', 'should', 'when', 'no', 'this', 'few', "isn't", 'of', 'where', 'into', 'now', 'or', "wasn't", 'their'
, 'doing', 'so', 'who', 'again', 'ain', 'herself', "shan't", 'don', 'it', 'other', 'at', 'we', "you're", 'because', 'how',
'with'}
```

In [23]:
```
df['tokenized_tweet'] = df['tokenized_tweet'].apply(lambda x: [word for word in x if word not in stop_words])
print("Data after removing stopwords:")
print(df['tokenized_tweet'].head(5))
```

```
Data after removing stopwords:
0                                    [shopping, bit, p]
1     [layout, doesnt, match, closest, could, find, ...
2     [lions, agains, otani, already, bottom, chance...
3            [love, store, best, chocolate, selections]
4              [maroon, cocktail, dresses, mididresses]
Name: tokenized_tweet, dtype: object
```

## POS Tagging

In [24]:
```
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[24]: True

In [25]:
```
df['pos_tagged_tweet'] = df['tokenized_tweet'].apply(lambda x: nltk.pos_tag(x))
print("Data after POS tagging")
print(df['pos_tagged_tweet'].head(5))
```

```
Data after POS tagging
0                 [(shopping, VBG), (bit, NN), (p, NN)]
1     [(layout, NN), (doesnt, NN), (match, NN), (clo...
2     [(lions, NNS), (agains, VBZ), (otani, JJ), (al...
3     [(love, VB), (store, NN), (best, JJS), (chocol...
4     [(maroon, NN), (cocktail, NN), (dresses, VBZ),...
Name: pos_tagged_tweet, dtype: object
```

## Lemmatization

In [26]:
```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[26]: True

In [27]:
```
lemmatizer = WordNetLemmatizer()

df['lemmatized_tweet'] = df['pos_tagged_tweet'].apply(lambda x: [lemmatizer.lemmatize(word[0]) for word in x])

print("Data after lemmatization:")

df['lemmatized_tweet'].head(5)
```

```
Data after lemmatization:
```

Out[27]:
```
0                                    [shopping, bit, p]
1     [layout, doesnt, match, closest, could, find, ...
2     [lion, agains, otani, already, bottom, chance,...
3             [love, store, best, chocolate, selection]
4                [maroon, cocktail, dress, mididresses]
Name: lemmatized_tweet, dtype: object
```

In [48]:
```
df['lemmatized_tweet']=df['lemmatized_tweet'].dropna()
```

### Splitting of Data into Train and Test Set

In [49]:
```
X_train, X_test, y_train, y_test = train_test_split(df['lemmatized_tweet'], df['sentiment'], test_size=0.2, random_state=42)
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train.apply(' '.join))
X_test = vectorizer.transform(X_test.apply(' '.join))
print("Training set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])
```

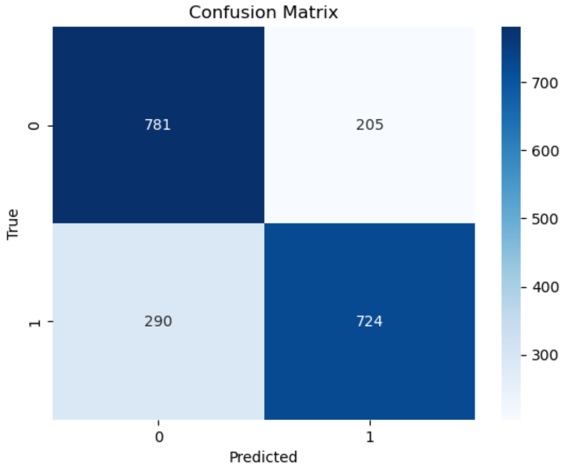
```
Training set size: 8000
Test set size: 2000
```

### Training Logisctic Model

In [50]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.79      0.76       986
           1       0.78      0.71      0.75      1014

    accuracy                           0.75      2000
   macro avg       0.75      0.75      0.75      2000
weighted avg       0.75      0.75      0.75      2000
```

In [51]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

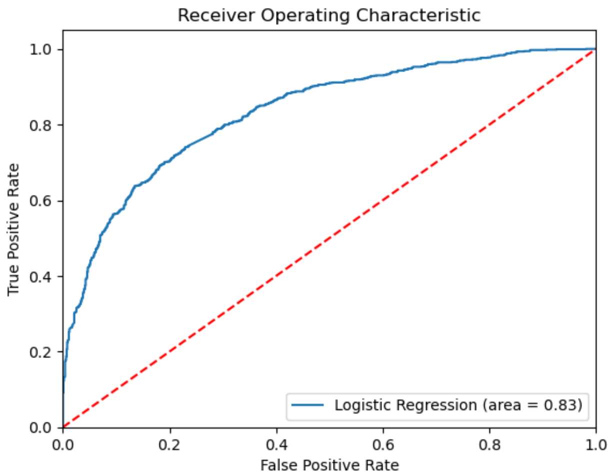



In [52]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
precision = precision_score(y_test, y_pred, average='weighted')
print("Precision:", precision)
recall = recall_score(y_test, y_pred, average='weighted')
print("Recall:", recall)

f2_score = fbeta_score(y_test, y_pred, beta=2, average='weighted')
print("F2 Score:", f2_score)
```

```
Accuracy: 0.7525
Precision: 0.754629572675859
Recall: 0.7525
F2 Score: 0.7520538734848613
```

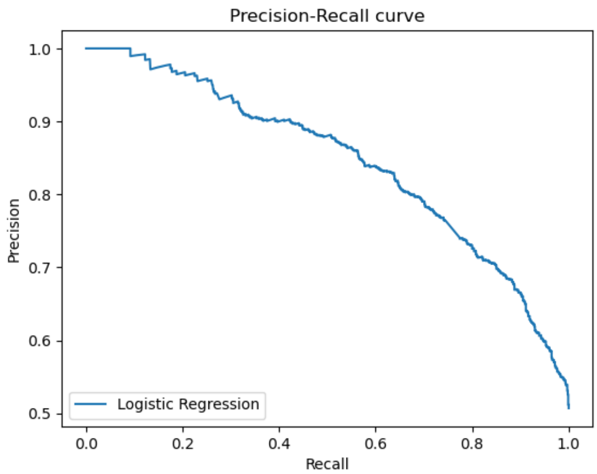
In [32]:

```
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % auc(fpr, tpr))
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```




In [33]:

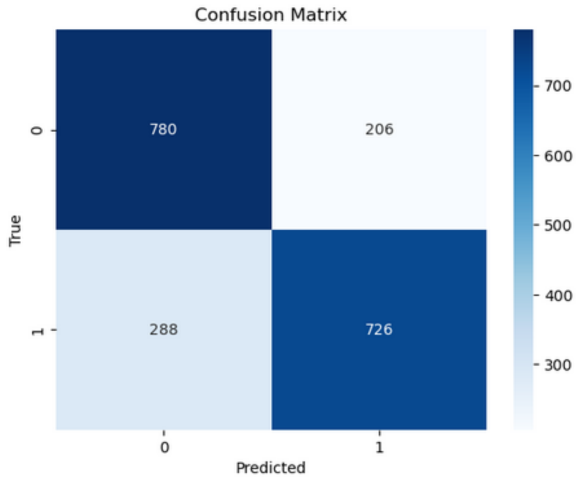
```
precision, recall, _ = precision_recall_curve(y_test, model.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(recall, precision, label='Logistic Regression')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall curve')
plt.legend(loc="lower left")
plt.show()
```



### Training Naive Bias Model

In [34]:
```
from sklearn.naive_bayes import MultinomialNB
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
nb_y_pred = nb_model.predict(X_test)
print(classification_report(y_test, nb_y_pred))
cmnb = confusion_matrix(y_test, nb_y_pred)
sns.heatmap(cmnb, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

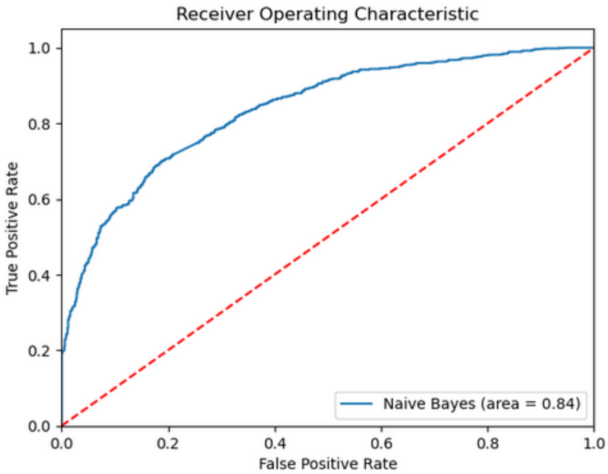
```
              precision    recall  f1-score   support

           0       0.73      0.79      0.76       986
           1       0.78      0.72      0.75      1014

    accuracy                           0.75      2000
   macro avg       0.75      0.75      0.75      2000
weighted avg       0.75      0.75      0.75      2000
```



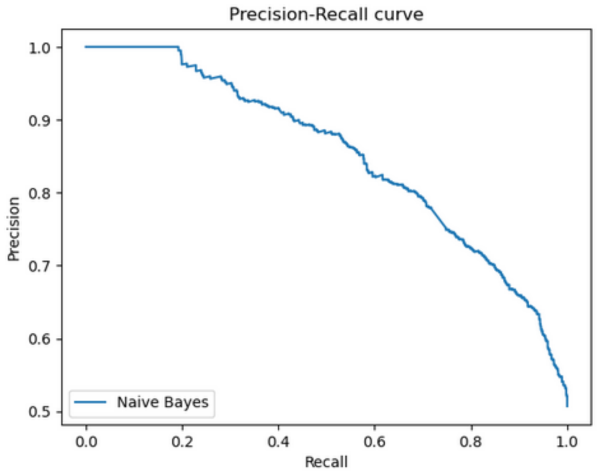

In [35]:
```
accuracy_nb = accuracy_score(y_test, nb_y_pred)
print("Accuracy:", accuracy_nb)
precision_nb = precision_score(y_test, nb_y_pred, average='weighted')
print("Precision:", precision_nb)
recall_nb = recall_score(y_test, nb_y_pred, average='weighted')
print("Recall:", recall_nb)
f2_score_nb = fbeta_score(y_test, nb_y_pred, beta=2, average='weighted')
print("F2 Score:", f2_score_nb)
```

```
Accuracy: 0.753
Precision: 0.7549939480156242
Recall: 0.753
F2 Score: 0.7525868357001737
```

In [36]:
```
fpr, tpr, thresholds = roc_curve(y_test, nb_model.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Naive Bayes (area = %0.2f)' % auc(fpr, tpr))
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```




In [37]:
```
precision, recall, _ = precision_recall_curve(y_test, nb_model.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(recall, precision, label='Naive Bayes')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall curve')
plt.legend(loc="lower left")
plt.show()
```



# Financial Dataset

## Importing Necessary Libraries

In [1]:
```
#Data Analysis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import textblob
import nltk
import wordcloud
from nltk.probability import FreqDist
from wordcloud import WordCloud
from collections import Counter
```

In [2]:
```
from textblob import TextBlob
from nltk.stem.wordnet import WordNetLemmatizer
import re
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
import string
import emoji
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
```

In [3]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, fbeta_score
```

In [4]:
```
import pandas as pd
df = pd.read_csv('C:\\Users\\HP\\Desktop\\AI ASSIGMNET\\New folder\\financial.csv', encoding='latin1',names=['sentimer
df.head(10)
```

Out[4]:

| | sentiment | text |
|---|---|---|
| 0 | neutral | According to Gran , the company has no plans t... |
| 1 | neutral | Technopolis plans to develop in stages an area... |
| 2 | negative | The international electronic industry company ... |
| 3 | positive | With the new production plant the company woul... |
| 4 | positive | According to the company 's updated strategy f... |
| 5 | positive | FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is ag... |
| 6 | positive | For the last quarter of 2010 , Componenta 's n... |
| 7 | positive | In the third quarter of 2010 , net sales incre... |
| 8 | positive | Operating profit rose to EUR 13.1 mn from EUR ... |
| 9 | positive | Operating profit totalled EUR 21.1 mn , up fro... |

### Exploratory Data Analysis

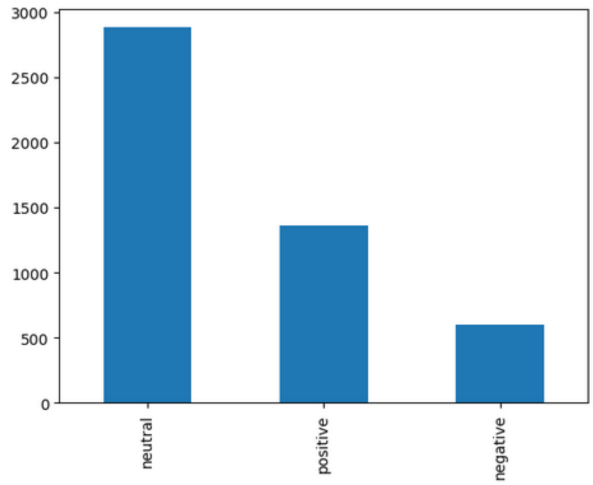
In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4846 entries, 0 to 4845
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   sentiment  4846 non-null   object
 1   text       4846 non-null   object
dtypes: object(2)
memory usage: 75.8+ KB
```

In [6]:
```
print(df['sentiment'].value_counts())
```

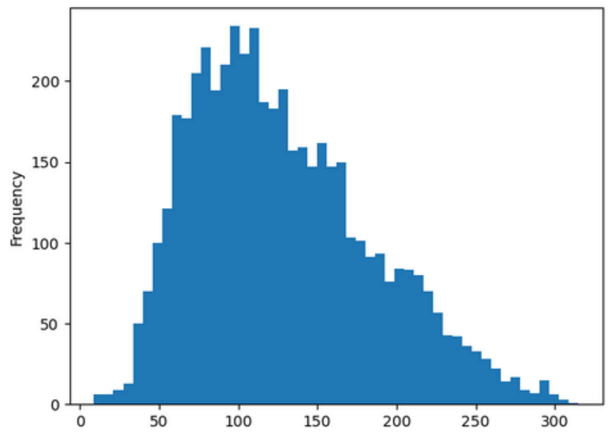
```
neutral     2879
positive    1363
negative     604
Name: sentiment, dtype: int64
```

In [7]:
```
df['sentiment'].value_counts().plot(kind='bar')
plt.show()
```




In [8]:
```
df['length'] = df['text'].apply(len)
print(df['length'].describe())
df['length'].plot(kind='hist', bins=50)
plt.show()
```

```
count    4846.000000
mean      128.132068
std        56.526180
min         9.000000
25%        84.000000
50%       119.000000
75%       163.000000
max       315.000000
Name: length, dtype: float64
```



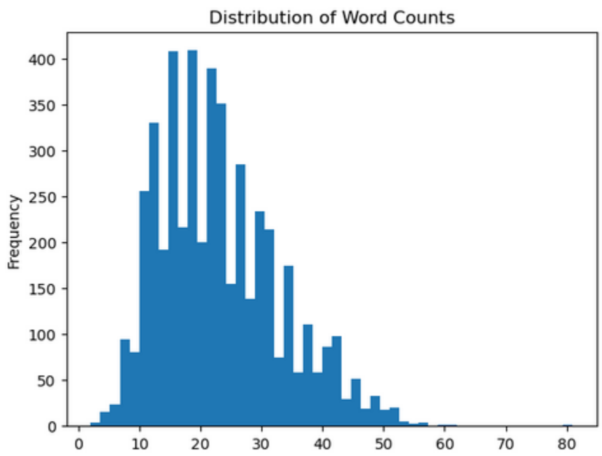
In [9]:

```python
all_words = ' '.join(df['text'])
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```


In [10]:

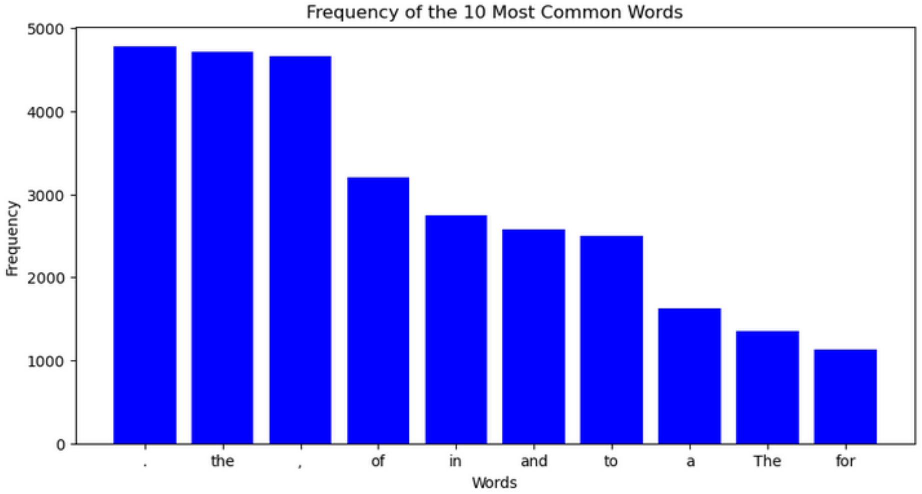
```python
df['word_count'] = df['text'].apply(lambda x: len(str(x).split()))
print(df['word_count'].describe())
df['word_count'].plot(kind='hist', bins=50)
plt.title('Distribution of Word Counts')
plt.show()
```

```
count    4846.000000
mean       23.101114
std         9.958474
min         2.000000
25%        16.000000
50%        21.000000
75%        29.000000
max        81.000000
Name: word_count, dtype: float64
```



In [11]:

```
all_words = [word for tweet in df['text'] for word in tweet.split()]
word_counts = Counter(all_words)
common_words = word_counts.most_common(10)
words, counts = zip(*common_words)
plt.figure(figsize=(10, 5))
plt.bar(words, counts, color='b')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Frequency of the 10 Most Common Words')
plt.show()
```



## Preprocessing

### Cleaning the data

In [12]:
```
def clean_text(text):

    text = str(text).lower()

    # Remove Twitter handles
    text = re.sub('@\w+', '', text)

    # Remove text in square brackets
    text = re.sub('\[.*?\]', '', text)

    # Remove URLs
    text = re.sub('https?://\S+|www\.\S+', '', text)

    # Remove HTML tags
    text = re.sub('<.*?>+', '', text)

    # Remove punctuation
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)

    # Remove new line characters
    text = re.sub('\n', '', text)

    # Remove words that contain numbers
    text = re.sub('\w*\d\w*', '', text)

    # Convert emojis to words
    text = emoji.demojize(text, delimiters=(" ", " "))

    return text
```

In [13]:
```
print("Data before preprocessing")
df['text'].head(5)
```

```
Data before preprocessing
```

Out[13]:
```
0    According to Gran , the company has no plans t...
1    Technopolis plans to develop in stages an area...
2    The international electronic industry company ...
3    With the new production plant the company woul...
4    According to the company 's updated strategy f...
Name: text, dtype: object
```

In [14]:
```
df['cleaned_text'] = df['text'].apply(clean_text)
print("Data after preprocessing")
print(df['cleaned_text'].head(5))
```

```
Data after preprocessing
0    according to gran  the company has no plans to...
1    technopolis plans to develop in stages an area...
2    the international electronic industry company ...
3    with the new production plant the company woul...
4    according to the company s updated strategy fo...
Name: cleaned_text, dtype: object
```

### Tokenization

In [15]:
```
df['tokenized_text']=df['cleaned_text'].apply(word_tokenize)
print("Data after tokenization")
df['tokenized_text'].head(5)
```

```
Data after tokenization
```

Out[15]:
```
0    [according, to, gran, the, company, has, no, p...
1    [technopolis, plans, to, develop, in, stages, ...
2    [the, international, electronic, industry, com...
3    [with, the, new, production, plant, the, compa...
4    [according, to, the, company, s, updated, stra...
Name: tokenized_text, dtype: object
```

### Removing Stop Words

In [16]:
```
stop_words = set(stopwords.words('english'))
print("The stop words are\n",stop_words)
```

```
The stop words are
 {'needn', 'is', 'of', 'now', 'herself', 'through', 'will', 'd', "doesn't", "couldn't", 'while', "won't", "you'd", 'h
ad', 'about', 'her', 'from', 'isn', 'down', "wasn't", 'has', 'myself', 'but', 'having', 'some', 'm', 'won', 'shan', '
me', 'during', 'haven', 'or', "isn't", 'off', 'these', 'wasn', 'were', "you've", 'ours', 'yourself', 'once', 'ourselv
es', "you're", 'our', 'by', "weren't", 'those', 'there', 'itself', 'll', 'have', 'been', 'shouldn', 'not', 'both', 'd
idn', 'into', 'you', 'in', 'to', 'themselves', 'does', 'too', 'hers', 'doesn', 'don', 'mightn', 'how', 'it', "shouldn
't", 'he', 'this', 'under', 'nor', 'here', 'own', 'same', 'and', 'y', "that'll", 'above', 'hadn', 'them', 'who', 'bei
ng', "haven't", 'aren', 'am', 're', "mustn't", 'on', 'against', 'my', "it's", 'why', 'yours', 'should', "needn't", 'a
', 'such', 'than', 's', 'do', "she's", 'their', 've', 'wouldn', 'whom', 'your', 'other', 'ain', 'mustn', 'over', 'be'
, 'before', 'for', 'the', 'just', "wouldn't", 'as', 'which', 'at', 'its', 'ma', 'she', 't', 'hasn', 'theirs', "don't"
, 'each', 'where', 'out', 'most', "didn't", 'after', 'can', "mightn't", 'any', 'weren', "hadn't", 'with', 'we', 'furt
her', 'no', 'they', 'are', "hasn't", 'what', 'was', 'few', 'i', 'then', 'did', "you'll", 'all', 'more', 'his', 'an',
"aren't", 'himself', "should've", 'between', 'only', 'up', 'o', 'when', 'very', 'that', 'if', 'until', 'below', 'agai
n', 'yourselves', 'couldn', "shan't", 'doing', 'him', 'because', 'so'}
```

In [17]:
```
df['tokenized_text'] = df['tokenized_text'].apply(lambda x: [word for word in x if word not in stop_words])
print("Data after removing stopwords:")
print(df['tokenized_text'].head(5))
```

```
Data after removing stopwords:
0    [according, gran, company, plans, move, produc...
1    [technopolis, plans, develop, stages, area, le...
2    [international, electronic, industry, company,...
3    [new, production, plant, company, would, incre...
4    [according, company, updated, strategy, years,...
Name: tokenized_text, dtype: object
```

### POS Tagging

In [18]:
```
df['pos_tagged_text'] = df['tokenized_text'].apply(lambda x: nltk.pos_tag(x))
print("Data after POS tagging")
print(df['pos_tagged_text'].head(5))
```

```
Data after POS tagging
0    [(according, VBG), (gran, NN), (company, NN), ...
1    [(technopolis, NNS), (plans, NNS), (develop, V...
2    [(international, JJ), (electronic, JJ), (indus...
3    [(new, JJ), (production, NN), (plant, NN), (co...
4    [(according, VBG), (company, NN), (updated, VB...
Name: pos_tagged_text, dtype: object
```

### Lemmatization

In [19]:
```
lemmatizer = WordNetLemmatizer()

df['lemmatized_text'] = df['pos_tagged_text'].apply(lambda x: [lemmatizer.lemmatize(word[0]) for word in x])

print("Data after lemmatization:")

df['lemmatized_text'].head(5)
```

```
Data after lemmatization:
```

Out[19]:
```
0    [according, gran, company, plan, move, product...
1    [technopolis, plan, develop, stage, area, le, ...
2    [international, electronic, industry, company,...
3    [new, production, plant, company, would, incre...
4    [according, company, updated, strategy, year, ...
Name: lemmatized_text, dtype: object
```

In [26]:
```
df=df.dropna()
```

### Splitting of Data into Train and Test Set

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(df['lemmatized_text'], df['sentiment'], test_size=0.2, random_stat
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train.apply(' '.join))
X_test = vectorizer.transform(X_test.apply(' '.join))
print("Training set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])
```

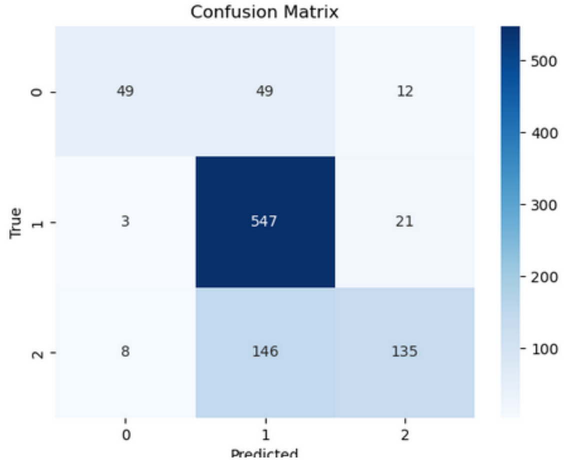
```
Training set size: 3876
Test set size: 970
```

### Training Logistic Model

In [28]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    negative       0.82      0.45      0.58       110
     neutral       0.74      0.96      0.83       571
    positive       0.80      0.47      0.59       289

    accuracy                           0.75       970
   macro avg       0.79      0.62      0.67       970
weighted avg       0.77      0.75      0.73       970
```

In [29]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



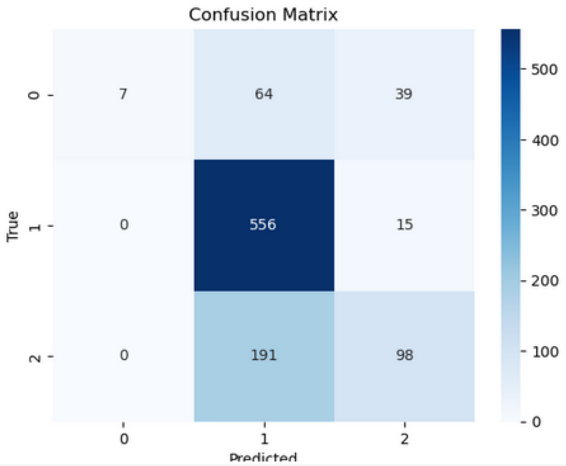

In [30]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
precision = precision_score(y_test, y_pred, average='weighted')
print("Precision:", precision)
recall = recall_score(y_test, y_pred, average='weighted')
print("Recall:", recall)

f2_score = fbeta_score(y_test, y_pred, beta=2, average='weighted')
print("F2 Score:", f2_score)
```

```
Accuracy: 0.7536082474226804
Precision: 0.7659843600929965
Recall: 0.7536082474226804
F2 Score: 0.7395118654904143
```

### Training Naive Bias Model

In [33]:
```
from sklearn.naive_bayes import MultinomialNB
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
nb_y_pred = nb_model.predict(X_test)
print(classification_report(y_test, nb_y_pred))
cmnb = confusion_matrix(y_test, nb_y_pred)
sns.heatmap(cmnb, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

    negative       1.00      0.06      0.12       110
     neutral       0.69      0.97      0.80       571
    positive       0.64      0.34      0.44       289

    accuracy                           0.68       970
   macro avg       0.78      0.46      0.46       970
weighted avg       0.71      0.68      0.62       970
```




In [32]:
```
accuracy_nb = accuracy_score(y_test, nb_y_pred)
print("Accuracy:", accuracy_nb)
precision_nb = precision_score(y_test, nb_y_pred, average='weighted')
print("Precision:", precision_nb)
recall_nb = recall_score(y_test, nb_y_pred, average='weighted')
print("Recall:", recall_nb)
f2_score_nb = fbeta_score(y_test, nb_y_pred, beta=2, average='weighted')
print("F2 Score:", f2_score_nb)
```

```
Accuracy: 0.6814432989690722
Precision: 0.7090632365741536
Recall: 0.6814432989690722
F2 Score: 0.6492400002623985
```

# Emotions Dataset

## Importing Necessary Libraries

In [1]:
```python
#Data Analysis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import textblob
import nltk
import wordcloud
from nltk.probability import FreqDist
from wordcloud import WordCloud
from collections import Counter
```

In [2]:
```python
from textblob import TextBlob
from nltk.stem.wordnet import WordNetLemmatizer
import re
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
import string
import emoji
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
```

In [3]:
```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, fbeta_score
```

In [4]:
```python
import pandas as pd
df = pd.read_csv('C:\\Users\\HP\\Desktop\\AI ASSIGMNET\\New folder\\Emotions\\train.csv', encoding='latin1')
df.head(10)
```

Out[4]:

| | textID | text | selected_text | sentiment | Time of Tweet | Age of User | Country | Population -2020 | Land Area (Km²) | Density (P/Km²) |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | cb774db0d1 | I'd have responded, if I were going | I'd have responded, if I were going | neutral | morning | 0-20 | Afghanistan | 38928346 | 652860.0 | 60 |
| 1 | 549e992a42 | Sooo SAD I will miss you here in San Diego!!! | Sooo SAD | negative | noon | 21-30 | Albania | 2877797 | 27400.0 | 105 |
| 2 | 088c60f138 | my boss is bullying me... | bullying me | negative | night | 31-45 | Algeria | 43851044 | 2381740.0 | 18 |
| 3 | 9642c003ef | what interview! leave me alone | leave me alone | negative | morning | 46-60 | Andorra | 77265 | 470.0 | 164 |
| 4 | 358bd9e861 | Sons of ****, why couldn`t they put them on t... | Sons of ****, | negative | noon | 60-70 | Angola | 32866272 | 1246700.0 | 26 |
| 5 | 28b57f3990 | http://www.dothebouncy.com/smf - some shameles... | http://www.dothebouncy.com/smf - some shameles... | neutral | night | 70-100 | Antigua and Barbuda | 97929 | 440.0 | 223 |
| 6 | 6e0c6d75b1 | 2am feedings for the baby are fun when he is a... | fun | positive | morning | 0-20 | Argentina | 45195774 | 2736690.0 | 17 |
| 7 | 50e14c0bb8 | Soooo high | Soooo high | neutral | noon | 21-30 | Armenia | 2963243 | 28470.0 | 104 |
| 8 | e050245fbd | Both of you | Both of you | neutral | night | 31-45 | Australia | 25499884 | 7682300.0 | 3 |
| 9 | fc2cbefa9d | Journey!? Wow... u just became cooler. hehe.... | Wow... u just became cooler. | positive | morning | 46-60 | Austria | 9006398 | 82400.0 | 109 |

### Dropping unwanted columns

In [5]:
```
def drop_unwanted_columns(df):
    columns_to_drop = ['textID','text', 'Time of Tweet', 'Age of User', 'Country', 'Population -2020', 'Land Area (Km²
    df = df.drop(columns_to_drop, axis=1)
    return df
df = drop_unwanted_columns(df)
df.head(10)
```

Out[5]:

| | selected_text | sentiment |
|---|---|---|
| 0 | I'd have responded, if I were going | neutral |
| 1 | Sooo SAD | negative |
| 2 | bullying me | negative |
| 3 | leave me alone | negative |
| 4 | Sons of ****, | negative |
| 5 | http://www.dothebouncy.com/smf - some shameles... | neutral |
| 6 | fun | positive |
| 7 | Soooo high | neutral |
| 8 | Both of you | neutral |
| 9 | Wow... u just became cooler. | positive |

### Exploratory Data Analysis

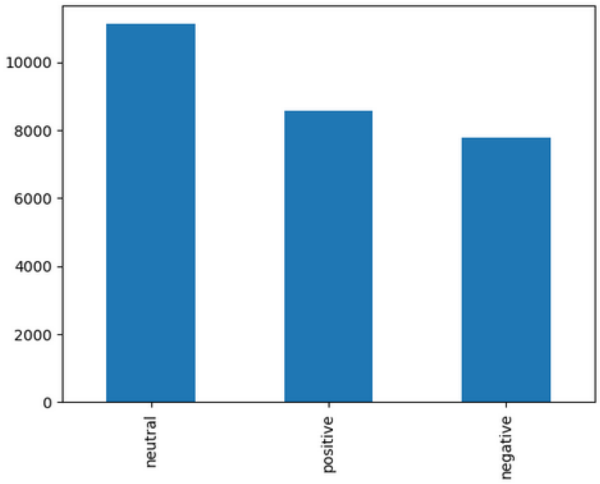
In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27481 entries, 0 to 27480
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   selected_text  27480 non-null  object
 1   sentiment      27481 non-null  object
dtypes: object(2)
memory usage: 429.5+ KB
```

In [7]:
```
print(df['sentiment'].value_counts())
```

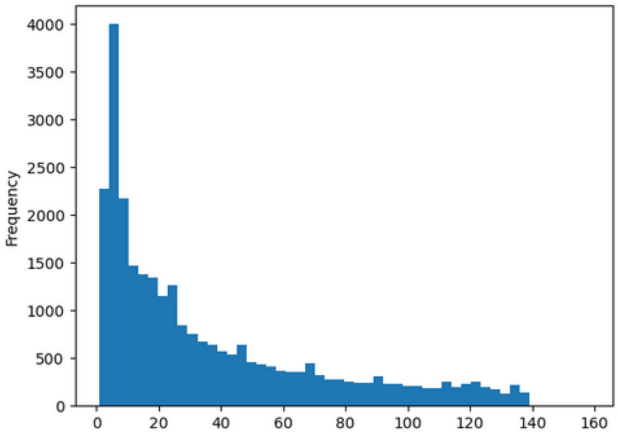
```
neutral     11118
positive     8582
negative     7781
Name: sentiment, dtype: int64
```

In [8]:
```
df['sentiment'].value_counts().plot(kind='bar')
plt.show()
```

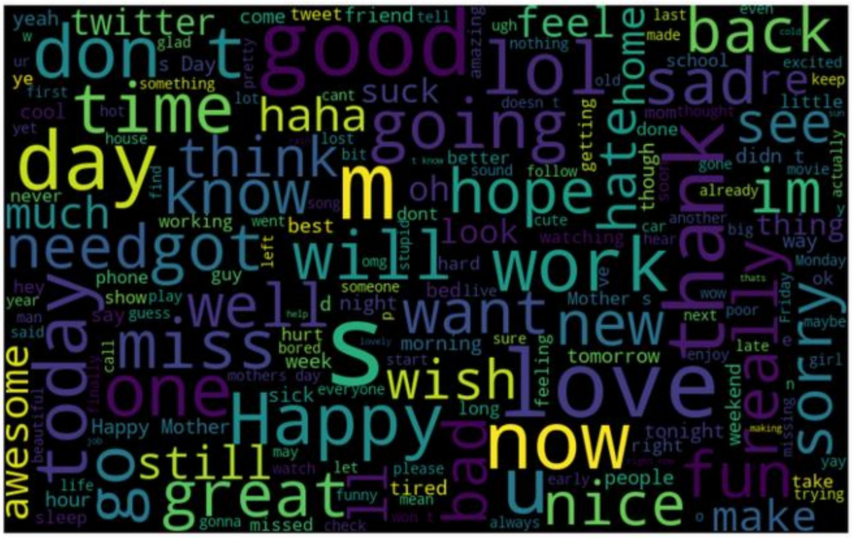


In [9]:
```
df['length'] = df['selected_text'].astype(str).apply(len)
print(df['length'].describe())
df['length'].plot(kind='hist', bins=50)
plt.show()
```

```
count    27481.000000
mean        36.681234
std         35.680581
min          1.000000
25%          8.000000
50%         22.000000
75%         55.000000
max        158.000000
Name: length, dtype: float64
```



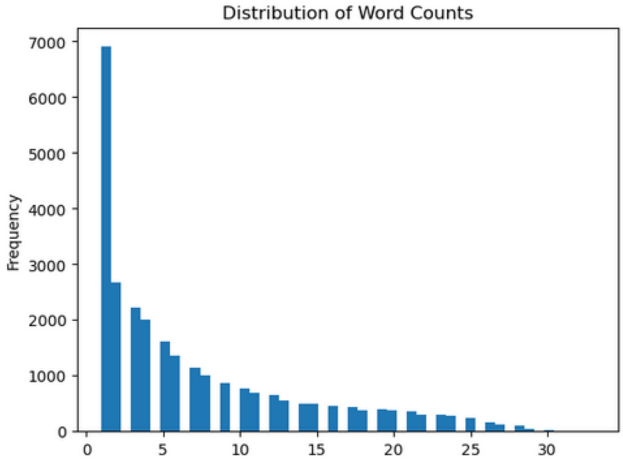

In [10]:
```
all_words = ' '.join(df['selected_text'].astype(str))
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```



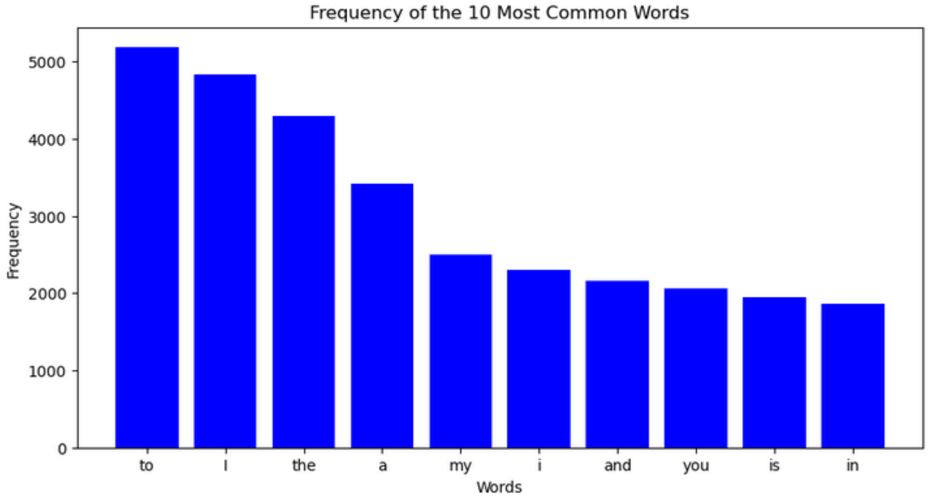
In [11]:

```
df['word_count'] = df['selected_text'].apply(lambda x: len(str(x).split()))
print(df['word_count'].describe())
df['word_count'].plot(kind='hist', bins=50)
plt.title('Distribution of Word Counts')
plt.show()
```

```
count    27481.000000
mean         7.094465
std          6.891356
min          1.000000
25%          1.000000
50%          4.000000
75%         11.000000
max         33.000000
Name: word_count, dtype: float64
```



In [12]:
```python
all_words = [word for tweet in df['selected_text'].astype(str) for word in tweet.split()]
word_counts = Counter(all_words)
common_words = word_counts.most_common(10)
words, counts = zip(*common_words)

plt.figure(figsize=(10, 5))
plt.bar(words, counts, color='b')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Frequency of the 10 Most Common Words')
plt.show()
```




## Preprocessing

### Cleaning the data

In [13]:
```python
def clean_text(text):

    text = str(text).lower()

    # Remove Twitter handles
    text = re.sub('@\w+', '', text)

    # Remove text in square brackets
    text = re.sub('\[.*?\]', '', text)

    # Remove URLs
    text = re.sub('https?://\S+|www\.\S+', '', text)

    # Remove HTML tags
    text = re.sub('<.*?>+', '', text)

    # Remove punctuation
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)

    # Remove new line characters
    text = re.sub('\n', '', text)

    # Remove words that contain numbers
    text = re.sub('\w*\d\w*', '', text)

    # Convert emojis to words
    text = emoji.demojize(text, delimiters=(" ", " "))

    return text
```

In [14]:
```python
print("Data before preprocessing")
df['selected_text'].head(5)
```

```
Data before preprocessing
```

Out[14]:
```
0    I`d have responded, if I were going
1                                Sooo SAD
2                              bullying me
3                           leave me alone
4                            Sons of ****,
Name: selected_text, dtype: object
```

In [15]:
```
df['cleaned_text'] = df['selected_text'].apply(clean_text)
print("Data after preprocessing")
print(df['cleaned_text'].head(5))
```

```
Data after preprocessing
0    id have responded if i were going
1                             sooo sad
2                          bullying me
3                       leave me alone
4                              sons of
Name: cleaned_text, dtype: object
```

### Tokenization

In [16]:
```
def tokenize_text(df, column_name='cleaned_text'):
    df['tokenized_text'] = df[column_name].apply(word_tokenize)
    return df
df = tokenize_text(df, 'cleaned_text')
print("Data after tokenization")
print(df['tokenized_text'].head(5))
```

```
Data after tokenization
0    [id, have, responded, if, i, were, going]
1                                  [sooo, sad]
2                              [bullying, me]
3                          [leave, me, alone]
4                                  [sons, of]
Name: tokenized_text, dtype: object
```

### Removing Stop Words

In [17]:
```
stop_words = set(stopwords.words('english'))
print("The stop words are\n",stop_words)
```

```
The stop words are
 {'and', "shouldn't", "should've", 'd', 'have', 'why', 'is', 'same', 'any', 'before', "hasn't", 'himself', 'having',
'only', 'our', 'very', 'wasn', 'then', 'most', 'hasn', 'your', "don't", 'am', 'while', 'not', "wouldn't", 'ain', 'its
', 'should', 'needn', 'he', 'so', 'between', 'theirs', 'do', 'm', 'against', 'now', 'yourself', 'just', 'through', 'h
is', 'because', 'such', "mightn't", 'ma', 'mustn', 'a', 'from', 'down', 'themselves', 'won', 'does', "couldn't", 'can
', 'was', "wasn't", "isn't", 'her', 'll', 'their', 'these', 'some', 'isn', 'after', 'you', 'been', 'during', 'to', 'a
bout', 'had', 'my', 'o', "won't", "that'll", 'it', 'shan', 've', 'y', 'the', 'which', 'those', 'again', 'how', 'mysel
f', 'weren', 'once', "mustn't", "weren't", 'too', 'are', 'yours', 'own', 'that', 'hadn', "shan't", 'but', 'both', 'ot
her', 'don', 'didn', 'were', 'wouldn', "it's", 'being', 'haven', 'ourselves', 'further', 'here', 'ours', 'out', 'all'
, 'over', 'yourselves', 'what', "you'd", 'doing', 'she', 'above', 'more', 's', "she's", 'we', "you've", 'whom', 'them
', 'shouldn', 'at', 'on', 'until', 'me', 'has', 'this', 'if', 'in', 'under', 'there', 'who', 'by', 'below', 't', "are
n't", 'they', 'each', 'off', 'nor', 'be', 'than', 'for', 'doesn', "needn't", 'itself', 'no', 'or', 'up', 'where', 'hi
m', 'will', 'into', 'with', "you'll", 'did', "didn't", 'few', "hadn't", 'of', 'aren', 'hers', 're', "you're", 'couldn
', 'mightn', 'as', "doesn't", 'when', 'i', 'an', 'herself', "haven't"}
```

In [18]:
```
def remove_stopwords(df, column_name='tokenized_text'):
    df[column_name] = df[column_name].apply(lambda x: [word for word in x if word not in stop_words])
    return df
df = remove_stopwords(df, 'tokenized_text')
print(df['tokenized_text'].head(5))
```

```
0    [id, responded, going]
1                [sooo, sad]
2                 [bullying]
3             [leave, alone]
4                     [sons]
Name: tokenized_text, dtype: object
```

## POS Tagging

In [19]:
```
def pos_tagging(df,colummn_name='tokenized_text'):
    df['pos_tagged_text'] = df['tokenized_text'].apply(lambda x: nltk.pos_tag(x))
    return df
df=pos_tagging(df, 'tokenized_text')
print("Data after POS tagging")
print(df['pos_tagged_text'].head(5))
```

```
Data after POS tagging
0    [(id, NN), (responded, VBD), (going, VBG)]
1                      [(sooo, NN), (sad, NN)]
2                             [(bullying, NN)]
3                  [(leave, VB), (alone, RB)]
4                                [(sons, NNS)]
Name: pos_tagged_text, dtype: object
```

## Lemmatization

In [20]:
```
def lemmatize_text(df):
    lemmatizer = WordNetLemmatizer()
    df['lemmatized_text'] = df['pos_tagged_text'].apply(lambda x: [lemmatizer.lemmatize(word[0]) for word in x])
    return df
df = lemmatize_text(df)
print("Data after Lemmatization")
print(df['lemmatized_text'].head(5))
```

```
Data after Lemmatization
0    [id, responded, going]
1                [sooo, sad]
2                 [bullying]
3             [leave, alone]
4                      [son]
Name: lemmatized_text, dtype: object
```

In [21]:
```
df.head(5)
```

Out[21]:

| | selected_text | sentiment | length | word_count | cleaned_text | tokenized_text | pos_tagged_text | lemmatized_text |
|---|---|---|---|---|---|---|---|---|
| 0 | I'd have responded, if I were going | neutral | 35 | 7 | id have responded if i were going | [id, responded, going] | [(id, NN), (responded, VBD), (going, VBG)] | [id, responded, going] |
| 1 | Sooo SAD | negative | 8 | 2 | sooo sad | [sooo, sad] | [(sooo, NN), (sad, NN)] | [sooo, sad] |
| 2 | bullying me | negative | 11 | 2 | bullying me | [bullying] | [(bullying, NN)] | [bullying] |
| 3 | leave me alone | negative | 14 | 3 | leave me alone | [leave, alone] | [(leave, VB), (alone, RB)] | [leave, alone] |
| 4 | Sons of ****, | negative | 13 | 3 | sons of | [sons] | [(sons, NNS)] | [son] |

## Preparing Test Data

In [22]:
```
df2= pd.read_csv('C:\\Users\\HP\\Desktop\\AI ASSIGMNET\\New folder\\Emotions\\test.csv', encoding='latin1')
df2.head(10)
```

Out[22]:

| | textID | text | sentiment | Time of Tweet | Age of User | Country | Population -2020 | Land Area ($Km^2$) | Density (P/$Km^2$) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | f87dea47db | Last session of the day http://twitpic.com/67ezh | neutral | morning | 0-20 | Afghanistan | 38928346.0 | 652860.0 | 60.0 |
| 1 | 96d74cb729 | Shanghai is also really exciting (precisely -... | positive | noon | 21-30 | Albania | 2877797.0 | 27400.0 | 105.0 |
| 2 | eee518ae67 | Recession hit Veronique Branquinho, she has to... | negative | night | 31-45 | Algeria | 43851044.0 | 2381740.0 | 18.0 |
| 3 | 01082688c6 | happy bday! | positive | morning | 46-60 | Andorra | 77265.0 | 470.0 | 164.0 |
| 4 | 33987a8ee5 | http://twitpic.com/4w75p - I like it!! | positive | noon | 60-70 | Angola | 32866272.0 | 1246700.0 | 26.0 |
| 5 | 726e501993 | that's great!! weee!! visitors! | positive | night | 70-100 | Antigua and Barbuda | 97929.0 | 440.0 | 223.0 |
| 6 | 261932614e | I THINK EVERYONE HATES ME ON HERE lol | negative | morning | 0-20 | Argentina | 45195774.0 | 2736690.0 | 17.0 |
| 7 | afa11da83f | soooooo wish i could, but im in school and my... | negative | noon | 21-30 | Armenia | 2963243.0 | 28470.0 | 104.0 |
| 8 | e64208b4ef | and within a short time of the last clue all ... | neutral | night | 31-45 | Australia | 25499884.0 | 7682300.0 | 3.0 |
| 9 | 37bcad24ca | What did you get? My day is alright.. haven'... | neutral | morning | 46-60 | Austria | 9006398.0 | 82400.0 | 109.0 |

In [23]:
```python
columns_to_drop = ['textID', 'Time of Tweet', 'Age of User', 'Country', 'Population -2020', 'Land Area (Km²)', 'Densit
df2 = df2.drop(columns_to_drop, axis=1)
df2.head(5)
```

Out[23]:

| | text | sentiment |
|---|---|---|
| 0 | Last session of the day http://twitpic.com/67ezh | neutral |
| 1 | Shanghai is also really exciting (precisely -... | positive |
| 2 | Recession hit Veronique Branquinho, she has to... | negative |
| 3 | happy bday! | positive |
| 4 | http://twitpic.com/4w75p - I like it!! | positive |

In [24]:
```python
df2['cleaned_text'] = df2['text'].apply(clean_text)
df2 = tokenize_text(df2, 'cleaned_text')
df2 = remove_stopwords(df2, 'tokenized_text')
df2 = pos_tagging(df2, 'tokenized_text')
df2 = lemmatize_text(df2)
```

In [25]:
```python
df2.head(5)
```

Out[25]:

| | text | sentiment | cleaned_text | tokenized_text | pos_tagged_text | lemmatized_text |
|---|---|---|---|---|---|---|
| 0 | Last session of the day http://twitpic.com/67ezh | neutral | last session of the day | [last, session, day] | [(last, JJ), (session, NN), (day, NN)] | [last, session, day] |
| 1 | Shanghai is also really exciting (precisely -... | positive | shanghai is also really exciting precisely s... | [shanghai, also, really, exciting, precisely, ... | [(shanghai, NN), (also, RB), (really, RB), (ex... | [shanghai, also, really, exciting, precisely, ... |
| 2 | Recession hit Veronique Branquinho, she has to... | negative | recession hit veronique branquinho she has to ... | [recession, hit, veronique, branquinho, quit, ... | [(recession, NN), (hit, VBD), (veronique, JJ),... | [recession, hit, veronique, branquinho, quit, ... |
| 3 | happy bday! | positive | happy bday | [happy, bday] | [(happy, JJ), (bday, NN)] | [happy, bday] |
| 4 | http://twitpic.com/4w75p - I like it!! | positive | i like it | [like] | [(like, IN)] | [like] |

In [26]:
```python
df.head(5)
```

Out[26]:

| | selected_text | sentiment | length | word_count | cleaned_text | tokenized_text | pos_tagged_text | lemmatized_text |
|---|---|---|---|---|---|---|---|---|
| 0 | I'd have responded, if I were going | neutral | 35 | 7 | id have responded if i were going | [id, responded, going] | [(id, NN), (responded, VBD), (going, VBG)] | [id, responded, going] |
| 1 | Sooo SAD | negative | 8 | 2 | sooo sad | [sooo, sad] | [(sooo, NN), (sad, NN)] | [sooo, sad] |
| 2 | bullying me | negative | 11 | 2 | bullying me | [bullying] | [(bullying, NN)] | [bullying] |
| 3 | leave me alone | negative | 14 | 3 | leave me alone | [leave, alone] | [(leave, VB), (alone, RB)] | [leave, alone] |
| 4 | Sons of ****, | negative | 13 | 3 | sons of | [sons] | [(sons, NNS)] | [son] |

## Training Logistic Model

In [27]:
```python
df['lemmatized_text_str'] = df['lemmatized_text'].apply(' '.join)
df2['lemmatized_text_str'] = df2['lemmatized_text'].apply(' '.join)
```

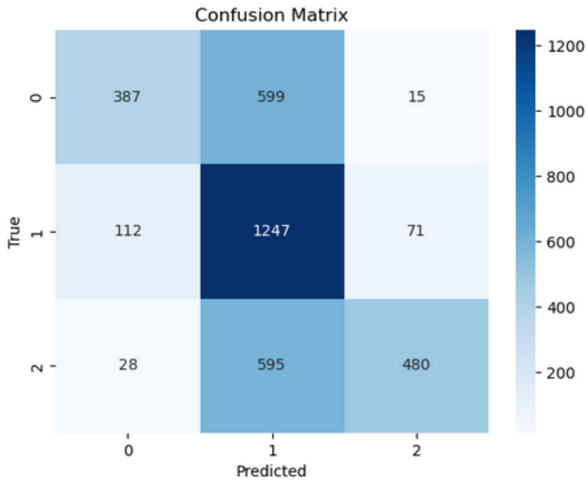
In [29]:
```python
df['sentiment'] = df['sentiment'].astype(str)
df2['sentiment'] = df2['sentiment'].astype(str)
```

In [31]:
```python
df = df.dropna()
df2 = df2.dropna()
```

In [32]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(df['lemmatized_text_str'])
y_train = df['sentiment']
X_test = vectorizer.transform(df2['lemmatized_text_str'])
y_test = df2['sentiment']
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    negative       0.73      0.39      0.51      1001
     neutral       0.51      0.87      0.64      1430
    positive       0.85      0.44      0.58      1103

    accuracy                           0.60      3534
   macro avg       0.70      0.56      0.58      3534
weighted avg       0.68      0.60      0.58      3534
```

In [33]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



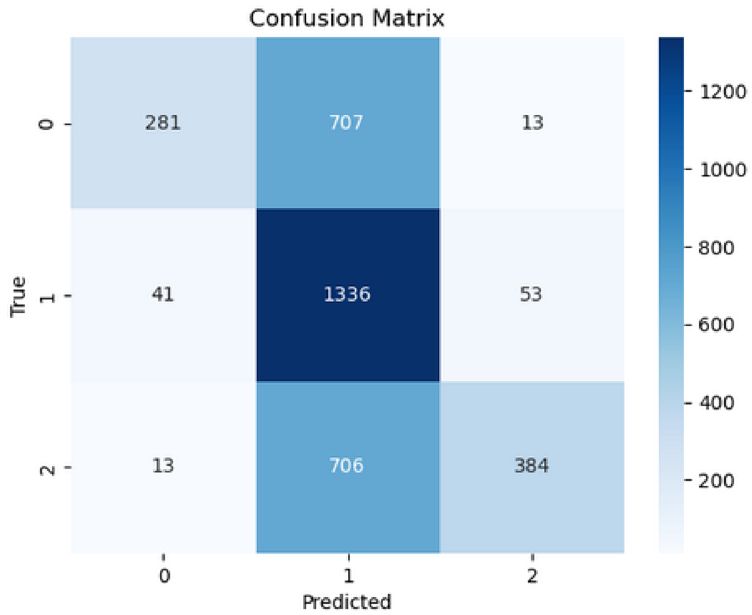

In [34]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
precision = precision_score(y_test, y_pred, average='weighted')
print("Precision:", precision)
recall = recall_score(y_test, y_pred, average='weighted')
print("Recall:", recall)

f2_score = fbeta_score(y_test, y_pred, beta=2, average='weighted')
print("F2 Score:", f2_score)
```

```
Accuracy: 0.5981890209394454
Precision: 0.6794030651032139
Recall: 0.5981890209394454
F2 Score: 0.5805841066749821
```

### Training Naive Bias Model

In [35]:
```
from sklearn.naive_bayes import MultinomialNB
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
nb_y_pred = nb_model.predict(X_test)
print(classification_report(y_test, nb_y_pred))
cmnb = confusion_matrix(y_test, nb_y_pred)
sns.heatmap(cmnb, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

    negative       0.84      0.28      0.42      1001
     neutral       0.49      0.93      0.64      1430
    positive       0.85      0.35      0.49      1103

    accuracy                           0.57      3534
   macro avg       0.73      0.52      0.52      3534
weighted avg       0.70      0.57      0.53      3534
```




In [36]:
```
accuracy_nb = accuracy_score(y_test, nb_y_pred)
print("Accuracy:", accuracy_nb)
precision_nb = precision_score(y_test, nb_y_pred, average='weighted')
print("Precision:", precision_nb)
recall_nb = recall_score(y_test, nb_y_pred, average='weighted')
print("Recall:", recall_nb)
f2_score_nb = fbeta_score(y_test, nb_y_pred, beta=2, average='weighted')
print("F2 Score:", f2_score_nb)
```

```
Accuracy: 0.566213921901528
Precision: 0.7005784267509434
Recall: 0.566213921901528
F2 Score: 0.5341342244001019
```

# Product Review Dataset

## Importing Necessary Libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import textblob
import nltk
import wordcloud
from nltk.probability import FreqDist
from wordcloud import WordCloud
from collections import Counter
```

In [2]:
```
from textblob import TextBlob
from nltk.stem.wordnet import WordNetLemmatizer
import re
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
import string
import emoji
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
```

In [3]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, fbeta_score
```

In [4]:
```
df = pd.read_csv('C:\\Users\\HP\\Desktop\\AI ASSIGMNET\\New folder\\Product review\\dataset.txt', sep="\t", header=Non
df['label'] = df[0].str.split(' ').str[0]
df['review'] = df[0].str.split(' ').str[1:]
df['review'] = df['review'].apply(lambda x: ' '.join(x))
df.drop(columns=[0], inplace=True)
print(df.head(10))
```

```
         label                                             review
0  __label__2  Great CD: My lovely Pat has one of the GREAT v...
1  __label__2  One of the best game music soundtracks - for a...
2  __label__1  Batteries died within a year ...: I bought thi...
3  __label__2  works fine, but Maha Energy is better: Check o...
4  __label__2  Great for the non-audiophile: Reviewed quite a...
5  __label__1  DVD Player crapped out after one year: I also ...
6  __label__1  Incorrect Disc: I love the style of this, but ...
7  __label__1  DVD menu select problems: I cannot scroll thro...
8  __label__2  Unique Weird Orientalia from the 1930's: Exoti...
9  __label__1  Not an "ultimate guide": Firstly,I enjoyed the...
```

In [5]:
```
df['sentiment'] = df['label'].replace({'__label__1': 'negative', '__label__2': 'positive'})
df.head(10)
```

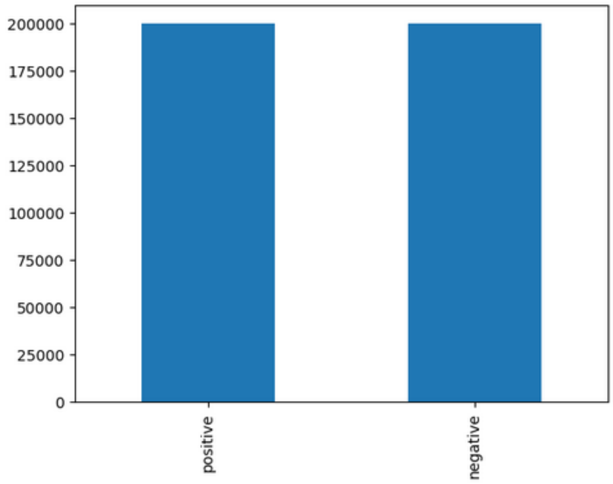
Out[5]:

| | label | review | sentiment |
|---|---|---|---|
| 0 | __label__2 | Great CD: My lovely Pat has one of the GREAT v... | positive |
| 1 | __label__2 | One of the best game music soundtracks - for a... | positive |
| 2 | __label__1 | Batteries died within a year ...: I bought thi... | negative |
| 3 | __label__2 | works fine, but Maha Energy is better: Check o... | positive |
| 4 | __label__2 | Great for the non-audiophile: Reviewed quite a... | positive |
| 5 | __label__1 | DVD Player crapped out after one year: I also ... | negative |
| 6 | __label__1 | Incorrect Disc: I love the style of this, but ... | negative |
| 7 | __label__1 | DVD menu select problems: I cannot scroll thro... | negative |
| 8 | __label__2 | Unique Weird Orientalia from the 1930's: Exoti... | positive |
| 9 | __label__1 | Not an "ultimate guide": Firstly,I enjoyed the... | negative |

In [7]:
```
print(df['sentiment'].value_counts())
```

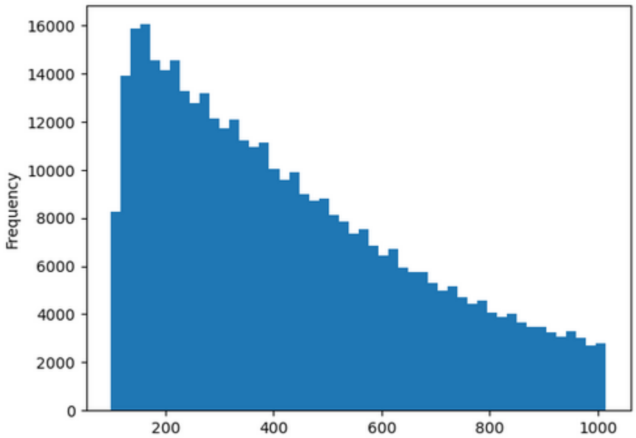
```
positive    200000
negative    200000
Name: sentiment, dtype: int64
```

In [8]:
```
df['sentiment'].value_counts().plot(kind='bar')
plt.show()
```



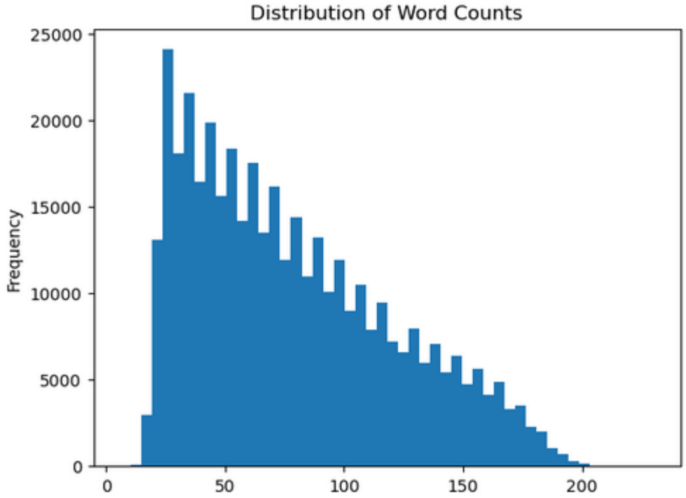

In [9]:
```
df['length'] = df['review'].apply(len)
print(df['length'].describe())
df['length'].plot(kind='hist', bins=50)
plt.show()
```

```
count    400000.000000
mean        431.429630
std         237.435383
min          99.000000
25%         231.000000
50%         383.000000
75%         595.000000
max        1015.000000
Name: length, dtype: float64
```

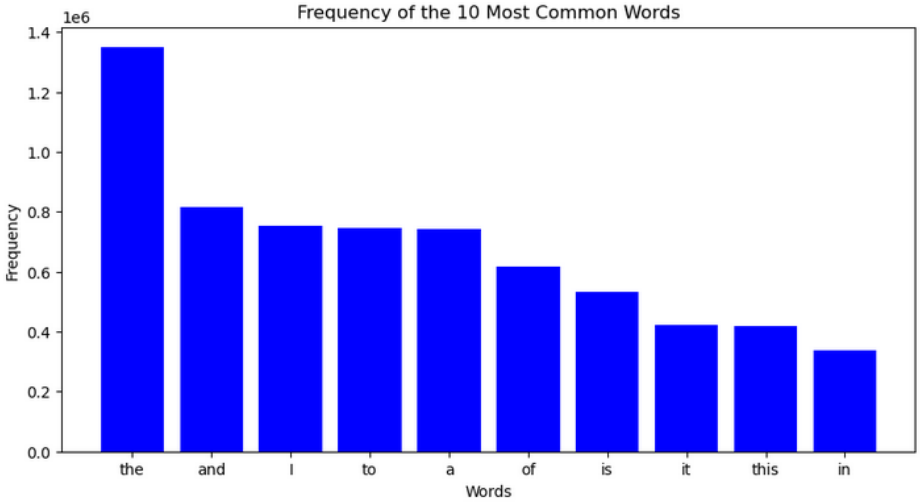


In [10]:
```
df['word_count'] = df['review'].apply(lambda x: len(str(x).split()))
print(df['word_count'].describe())
df['word_count'].plot(kind='hist', bins=50)
plt.title('Distribution of Word Counts')
plt.show()
```

```
count    400000.000000
mean         78.424145
std          42.798609
min           6.000000
25%          42.000000
50%          70.000000
75%         108.000000
max         230.000000
Name: word_count, dtype: float64
```




In [11]:
```
all_words = [word for tweet in df['review'] for word in tweet.split()]
word_counts = Counter(all_words)
common_words = word_counts.most_common(10)
words, counts = zip(*common_words)
plt.figure(figsize=(10, 5))
plt.bar(words, counts, color='b')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Frequency of the 10 Most Common Words')
plt.show()
```



# Preprocessing

## Cleaning the data

In [12]:
```
def clean_text(text):

    text = str(text).lower()

    # Remove Twitter handles
    text = re.sub('@\w+', '', text)

    # Remove text in square brackets
    text = re.sub('\[.*?\]', '', text)

    # Remove URLs
    text = re.sub('https?://\S+|www\.\S+', '', text)

    # Remove HTML tags
    text = re.sub('<.*?>+', '', text)

    # Remove punctuation
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)

    # Remove new line characters
    text = re.sub('\n', '', text)

    # Remove words that contain numbers
    text = re.sub('\w*\d\w*', '', text)

    # Convert emojis to words
    text = emoji.demojize(text, delimiters=(" ", " "))

    return text
```

In [13]:
```
print("Data before preprocessing")
df['review'].head(5)
```

```
Data before preprocessing
```

Out[13]:
```
0    Great CD: My lovely Pat has one of the GREAT v...
1    One of the best game music soundtracks - for a...
2    Batteries died within a year ...: I bought thi...
3    works fine, but Maha Energy is better: Check o...
4    Great for the non-audiophile: Reviewed quite a...
Name: review, dtype: object
```

In [14]:
```
df['cleaned_text'] = df['review'].apply(clean_text)
print("Data after preprocessing")
print(df['cleaned_text'].head(5))
```

```
Data after preprocessing
0    great cd my lovely pat has one of the great vo...
1    one of the best game music soundtracks  for a ...
2    batteries died within a year  i bought this ch...
3    works fine but maha energy is better check out...
4    great for the nonaudiophile reviewed quite a b...
Name: cleaned_text, dtype: object
```

## Tokenization

In [15]:
```
df['tokenized_text']=df['cleaned_text'].apply(word_tokenize)
print("Data after tokenization")
df['tokenized_text'].head(5)
```

```
Data after tokenization
```

Out[15]:
```
0    [great, cd, my, lovely, pat, has, one, of, the...
1    [one, of, the, best, game, music, soundtracks,...
2    [batteries, died, within, a, year, i, bought, ...
3    [works, fine, but, maha, energy, is, better, c...
4    [great, for, the, nonaudiophile, reviewed, qui...
Name: tokenized_text, dtype: object
```

## Removing Stop Words

In [16]:
```
stop_words = set(stopwords.words('english'))
print("The stop words are\n",stop_words)
```

```
The stop words are
 {'did', 'before', 'than', 'further', 'is', 'were', 'those', 'both', 'doing', 'we', 'are', "shouldn't", 'been', 'off'
, 'our', 'if', 'didn', 'or', 'through', 'm', 'while', 'o', 'once', 'needn', 'about', 'myself', 'over', 'ourselves', '
more', 'isn', 'shouldn', "weren't", 'be', 'for', 'am', 'their', "mightn't", "hasn't", 'because', 'again', 'an', 'but'
, "she's", 'ma', "won't", 'as', 'all', "hadn't", 'himself', 'was', 'ain', 'she', 'and', 'not', 'should', 'hasn', 'you
r', 'very', 'themselves', 'to', 'had', 'such', "you'd", "haven't", "should've", 'being', 'then', 'can', 've', 'how',
'you', 'above', 'doesn', "wasn't", "you'll", 'it', 'on', 'where', 'him', 'my', "mustn't", 'yourself', 'most', 'below'
, 'the', 'same', 'which', "you're", 'after', 'in', 'other', "didn't", 'yourselves', 'hers', 'its', 'do', 'no', 'from'
, 'only', "shan't", 'who', 'ours', 'aren', 'between', 'them', 'a', 'i', 'few', 'mustn', 'they', 'what', 'any', "that'
ll", 'won', 'up', 'too', 'having', 'out', "you've", 'that', "isn't", 'here', 're', 'why', 'whom', "needn't", 'her', '
until', 's', 'now', 'this', "aren't", 'his', 'don', "it's", 'theirs', "couldn't", 'under', 'nor', 'by', 'so', 'of', '
he', 'have', 't', 'own', 'wasn', 'has', 'with', 'yours', 'itself', 'against', 'just', 'couldn', 'there', 'hadn', 'her
self', "wouldn't", 'will', 'down', 'during', 'these', 'each', 'y', 'd', 'me', 'haven', 'when', 'weren', 'wouldn', 'mi
ghtn', 'at', 'some', 'into', "doesn't", 'shan', "don't", 'does', 'll'}
```

In [17]:
```
df['tokenized_text'] = df['tokenized_text'].apply(lambda x: [word for word in x if word not in stop_words])
print("Data after removing stopwords:")
print(df['tokenized_text'].head(5))
```

```
Data after removing stopwords:
0    [great, cd, lovely, pat, one, great, voices, g...
1    [one, best, game, music, soundtracks, game, di...
2    [batteries, died, within, year, bought, charge...
3    [works, fine, maha, energy, better, check, mah...
4    [great, nonaudiophile, reviewed, quite, bit, c...
Name: tokenized_text, dtype: object
```

## POS Tagging

In [18]:
```
df['pos_tagged_text'] = df['tokenized_text'].apply(lambda x: nltk.pos_tag(x))
print("Data after POS tagging")
print(df['pos_tagged_text'].head(5))
```

```
Data after POS tagging
0    [(great, JJ), (cd, NN), (lovely, RB), (pat, JJ...
1    [(one, CD), (best, JJS), (game, NN), (music, N...
2    [(batteries, NNS), (died, VBD), (within, IN), ...
3    [(works, NNS), (fine, VBP), (maha, NN), (energ...
4    [(great, JJ), (nonaudiophile, JJ), (reviewed, ...
Name: pos_tagged_text, dtype: object
```

## Lemmatization

In [19]:
```
lemmatizer = WordNetLemmatizer()

df['lemmatized_text'] = df['pos_tagged_text'].apply(lambda x: [lemmatizer.lemmatize(word[0]) for word in x])

print("Data after lemmatization:")

df['lemmatized_text'].head(5)
```

```
Data after lemmatization:
```

Out[19]:
```
0    [great, cd, lovely, pat, one, great, voice, ge...
1    [one, best, game, music, soundtrack, game, did...
2    [battery, died, within, year, bought, charger,...
3    [work, fine, maha, energy, better, check, maha...
4    [great, nonaudiophile, reviewed, quite, bit, c...
Name: lemmatized_text, dtype: object
```

In [20]:
```
df=df.dropna()
```

## Splitting of Data into Train and Test Set

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(df['lemmatized_text'], df['sentiment'], test_size=0.2, random_s
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train.apply(' '.join))
X_test = vectorizer.transform(X_test.apply(' '.join))
print("Training set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])
```

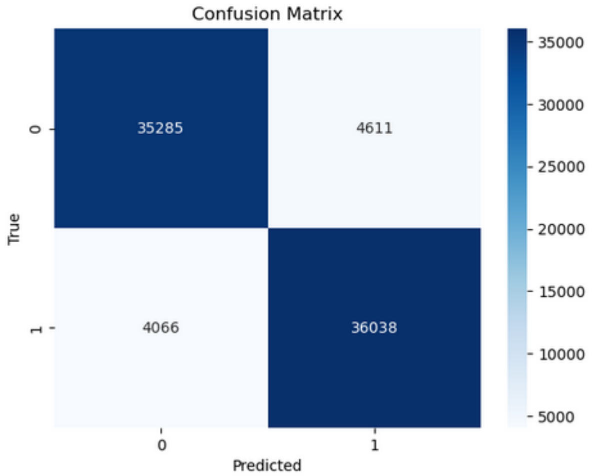
```
Training set size: 320000
Test set size: 80000
```

## Training Logistic Model

In [22]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    negative       0.90      0.88      0.89     39896
    positive       0.89      0.90      0.89     40104

    accuracy                           0.89     80000
   macro avg       0.89      0.89      0.89     80000
weighted avg       0.89      0.89      0.89     80000
```

In [23]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



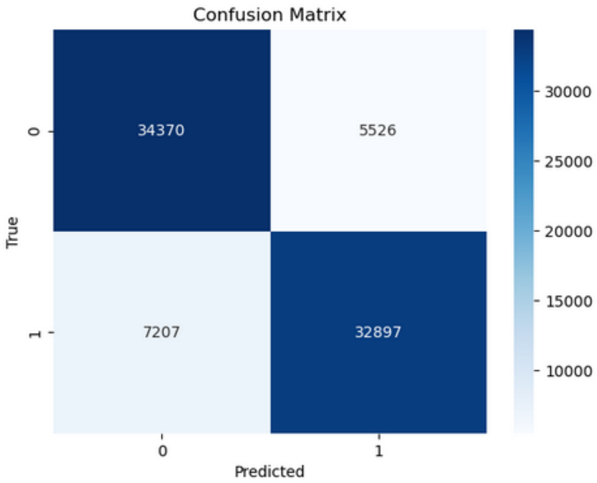

In [24]:
```python
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
precision = precision_score(y_test, y_pred, average='weighted')
print("Precision:", precision)
recall = recall_score(y_test, y_pred, average='weighted')
print("Recall:", recall)

f2_score = fbeta_score(y_test, y_pred, beta=2, average='weighted')
print("F2 Score:", f2_score)
```

```
Accuracy: 0.8915375
Precision: 0.8916063611098787
Recall: 0.8915375
F2 Score: 0.8915247873433607
```

### Training Naive Bias Model

In [25]:
```
from sklearn.naive_bayes import MultinomialNB
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
nb_y_pred = nb_model.predict(X_test)
print(classification_report(y_test, nb_y_pred))
cmnb = confusion_matrix(y_test, nb_y_pred)
sns.heatmap(cmnb, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

    negative       0.83      0.86      0.84     39896
    positive       0.86      0.82      0.84     40104

    accuracy                           0.84     80000
   macro avg       0.84      0.84      0.84     80000
weighted avg       0.84      0.84      0.84     80000
```




In [26]:
```
accuracy_nb = accuracy_score(y_test, nb_y_pred)
print("Accuracy:", accuracy_nb)
precision_nb = precision_score(y_test, nb_y_pred, average='weighted')
print("Precision:", precision_nb)
recall_nb = recall_score(y_test, nb_y_pred, average='weighted')
print("Recall:", recall_nb)
f2_score_nb = fbeta_score(y_test, nb_y_pred, beta=2, average='weighted')
print("F2 Score:", f2_score_nb)
```

```
Accuracy: 0.8408375
Precision: 0.8414578083766426
Recall: 0.8408375
F2 Score: 0.8407237644283192
```