MACHINE LEARNING MINI PROJECT

BY:

- UCE2022453 : Rutuja Dhirde
- UCE2022456 : Divya Ganeshwala

PROBLEM STATEMENT : The goal of this project is to employ machine learning techniques for sentiment analysis, specifically targeting tweets. Sentiment analysis involves determining whether a given tweet expresses a positive, negative, or neutral sentiment.

INTRODUCTION: In today's digital age, the rapid dissemination of information through social media platforms like Twitter has become ubiquitous. Users rely on these platforms for news and updates, making them powerful tools for understanding public sentiment. However, alongside genuine content, social media is flooded with tweets expressing diverse emotions ranging from positivity to negativity. Analysing this sentiment can offer valuable insights into public perception, brand sentiment, and social trends. Moreover, as elections loom closer, the analysis of tweets becomes even more critical, providing a lens into public sentiment towards political figures and parties. By examining the sentiments expressed in tweets, we can gain a deeper understanding of people's opinions, preferences, and concerns, thus informing political strategies and public discourse.

DATASET INFORMATION :
Twitter Sentiment Analysis Dataset

AIM : To develop a sentiment analysis model using machine learning techniques tailored for Twitter data.

CODE :

```
# !pip install wordcloud
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import numpy as np
import re
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
from collections import Counter
from sklearn.metrics import accuracy_score, classification_report
import pickle as pckl
from sklearn.ensemble import RandomForestClassifier  # Step 1: Import RandomForestClassifier
from nltk.stem import WordNetLemmatizer
```

```
# Load dataset (replace 'twitter_data.csv' with your actual file path)
# Load dataset (replace 'twitter_training (2).csv' with your actual file path)
data = pd.read_csv("/content/twitter_training (2).csv")
```

```
print(data.shape)

print(data.tail())
```

O/P:

```
(61121, 4)
          NO LOACTION      TYPE  \
61116  9200   Nvidia  Positive
61117  9200   Nvidia  Positive
61118  9200   Nvidia  Positive
61119  9200   Nvidia  Positive
61120  9200   Nvidia  Positive

                                                   TWEET
61116  Just realized that the Windows partition of my...
61117  Just realized that my Mac window partition is ...
61118  Just realized the windows partition of my Mac ...
61119  Just realized between the windows partition of...
61120  Just like the windows partition of my Mac is l...
```

```
print("Missing values before imputation:")
print(data.isnull().sum())

# Preprocessing
# Handle missing values (address potential issues with 'type' column)
imputer = SimpleImputer(strategy="constant", fill_value="Unknown")
data[['TWEET', 'TYPE']] = imputer.fit_transform(data[['TWEET', 'TYPE']])  # Impute both 'text' and 'type

# Check for missing values after imputation
print("Missing values after imputation:")
print(data.isnull().sum())
```

O/P:

```
Missing values before imputation:
NO          0
LOACTION    0
TYPE        0
TWEET       3
dtype: int64
Missing values after imputation:
NO          0
LOACTION    0
TYPE        0
TWEET       0
dtype: int64
```

```
# Download NLTK resources
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
  Enter tweet for sentiment analysis i am good
```

The Natural Language Toolkit (NLTK) : A platform used for building Python programs that work with human language data for application in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

STOP WORDS : Stop words are common words like ‘the’, ‘and’, ‘I’, etc. that are very frequent in text, and so don’t convey insights into the specific topic of a document. We can remove these stop words from the text in a given corpus to clean up the data, and identify words that are more rare and potentially more relevant to what we’re interested in.
Text may contain stop words like ‘the’, ‘is’, ‘are’. Stop words can be filtered from the text to be processed.
There is no universal list of stop words in nlp research, however the nltk module contains a list of stop words.

WordNet Lemmatizer: Often referred to simply as the WordNet Lemmatizer, is a tool commonly used in NLP tasks. Its primary function is to reduce words to their base or root form, known as the lemma. For example, the lemma of the word "running" is "run," and the lemma of "better" is "good."
The main purpose of lemmatization is to normalize words so that variations of the same word are treated as the same token, which can improve the accuracy and efficiency of machine learning algorithms.

Preprocess_text:

```
I/P  —>  text = "Hello, World! This is an example text with @symbols and\newlines."
O/P  —>  hello world this is an example text with symbols and newlines
```

PUNKET: The 'punkt' resource refers to the Punkt tokenizer models provided by NLTK. Tokenization is the process of breaking text into smaller units, typically words or sentences, which are called tokens. The Punkt tokenizer is a pre-trained tokenizer that can tokenize text into sentences or words. It is particularly useful for tasks such as text segmentation, where you need to divide a paragraph into individual sentences.

COUNTVECTORIZER:
Ex – food is good but food is not good

Tokenized version of the sentence:
["the", "food", "is", "good", "but", "food", "is", "not", "good"]

Vocabulary:
{'the': 0, 'food': 1, 'is': 2, 'good': 3, 'but': 4, 'not': 5}

Count matrix:

	the	food	is	good	but	not
0	1	1	2	2	1	0
1	1	2	2	1	1	1

CSR format representation:

(0, 0) 1
(0, 1) 1
(0, 2) 2
(0, 3) 2
(0, 4) 1
(1, 0) 1
(1, 1) 2
(1, 2) 2
(1, 3) 1
(1, 4) 1
(1, 5) 1

```python
# Define preprocessing functions
def preprocess_text(text):
    """Preprocess text data."""
    text = str(text)
    # Convert text to lowercase
    text = text.lower()
    # Remove non-alphanumeric characters, whitespace, newline, and tab characters
    text = re.sub(r"[^\w\s\n\t]+", "", text)
    return text

def remove_stopwords(text):
    """Remove stopwords from text."""
    stop_words = set(stopwords.words("english"))
    word_tokens = word_tokenize(text)
    filtered_text = [word for word in word_tokens if word not in stop_words]
    return ' '.join(filtered_text)

def lemmatize_text(text):
    """Lemmatize words in text."""
    lemmatizer = WordNetLemmatizer()
    word_tokens = word_tokenize(text)
    lemmatized_text = [lemmatizer.lemmatize(word) for word in word_tokens]
    return ' '.join(lemmatized_text)
```

```
# Preprocessing
imputer = SimpleImputer(strategy="constant", fill_value="Unknown")
data[['TWEET', 'TYPE']] = imputer.fit_transform(data[['TWEET', 'TYPE']])
data['TWEET'] = data['TWEET'].apply(preprocess_text)
data['TWEET'] = data['TWEET'].apply(remove_stopwords)
data['TWEET'] = data['TWEET'].apply(lemmatize_text)
```

```
print(data.tail())
```

O/P:

```
            NO LOACTION      TYPE  \
61116  9200   Nvidia  Positive
61117  9200   Nvidia  Positive
61118  9200   Nvidia  Positive
61119  9200   Nvidia  Positive
61120  9200   Nvidia  Positive

                                                   TWEET
61116  realized window partition mac like 6 year behi...
61117  realized mac window partition 6 year behind nv...
61118  realized window partition mac 6 year behind nv...
61119  realized window partition mac like 6 year behi...
61120  like window partition mac like 6 year behind d...
```

MULTINOMIALNB:

y_train[0]: "Positive"
y_train[1]: "Negative"
y_train[2]: "Positive"

X_train[0]: "i am good"
X_train[1]: "food is bad"
X_train[2]: "drink is cool"

(0, 0) 1 # "am" appears once in the first document
(0, 1) 1 # "good" appears once in the first document
(0, 2) 1 # "i" appears once in the first document
(1, 3) 1 # "bad" appears once in the second document
(1, 4) 1 # "food" appears once in the second document
(1, 5) 1 # "is" appears once in the second document
(2, 6) 1 # "cool" appears once in the third document
(2, 5) 1 # "drink" appears once in the third document
(2, 7) 1 # "is" appears once in the third document

Multinomial Naive Bayes classifies text by representing documents as word count vectors, assuming independence between words. It estimates class

priors and conditional probabilities from training data. During classification, it calculates posterior probabilities for each class using Bayes' theorem and selects the class with the highest probability. It handles unseen words through techniques like Laplace smoothing. Overall, it's efficient for text classification due to its simplicity and effectiveness in dealing with high-dimensional feature spaces.

USING NAIVE BAYES CLASSIFIER:

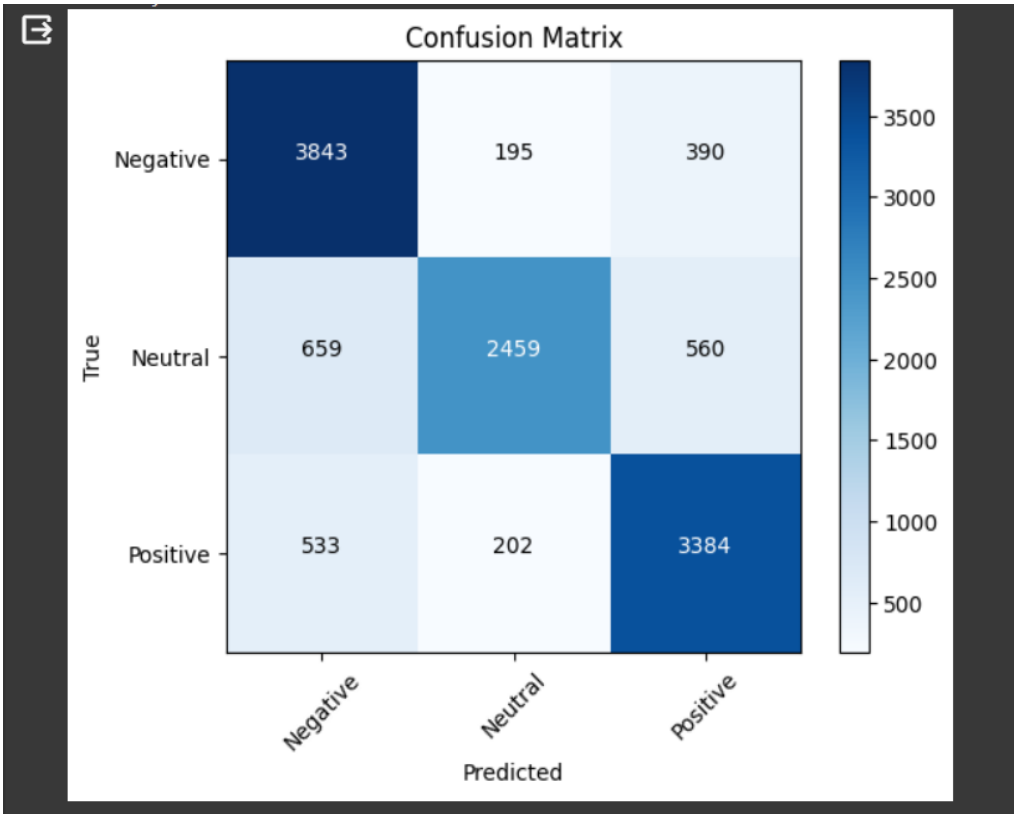
```
# Split data into input (X) and target (y)
X = data['TWEET']
y = data['TYPE']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert non-string elements to string in X_train
X_train = X_train.astype(str)
X_test = X_test.astype(str)
# Vectorize the text data
vectorizer = CountVectorizer()
vectorizer.fit(X_train)
X_train_vec = vectorizer.transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Train Naive Bayes classifier
classifier = MultinomialNB()                #92%
classifier.fit(X_train_vec, y_train)

# Predict
y_pred = classifier.predict(X_test_vec)
# Calculate accuracy (assuming 'type' has multiple classes)
accuracy = accuracy_score(y_test, y_pred)
```

```
new_tweet = input("  Enter tweet for sentiment analysis ")

# Preprocess the new tweet
cleaned_tweet = preprocess_text(new_tweet)
cleaned_tweet = remove_stopwords(new_tweet)
cleaned_tweet = lemmatize_text(new_tweet)

# Vectorize the preprocessed new tweet
vectorized_tweet = vectorizer.transform([cleaned_tweet])

# Predict the sentiment
predicted_sentiment = classifier.predict(vectorized_tweet)

print("Predicted Sentiment:", predicted_sentiment)
# Predict the sentiment

# Calculate and print the accuracy
print("Accuracy:", accuracy_score(y_test, y_pred)*100)
```

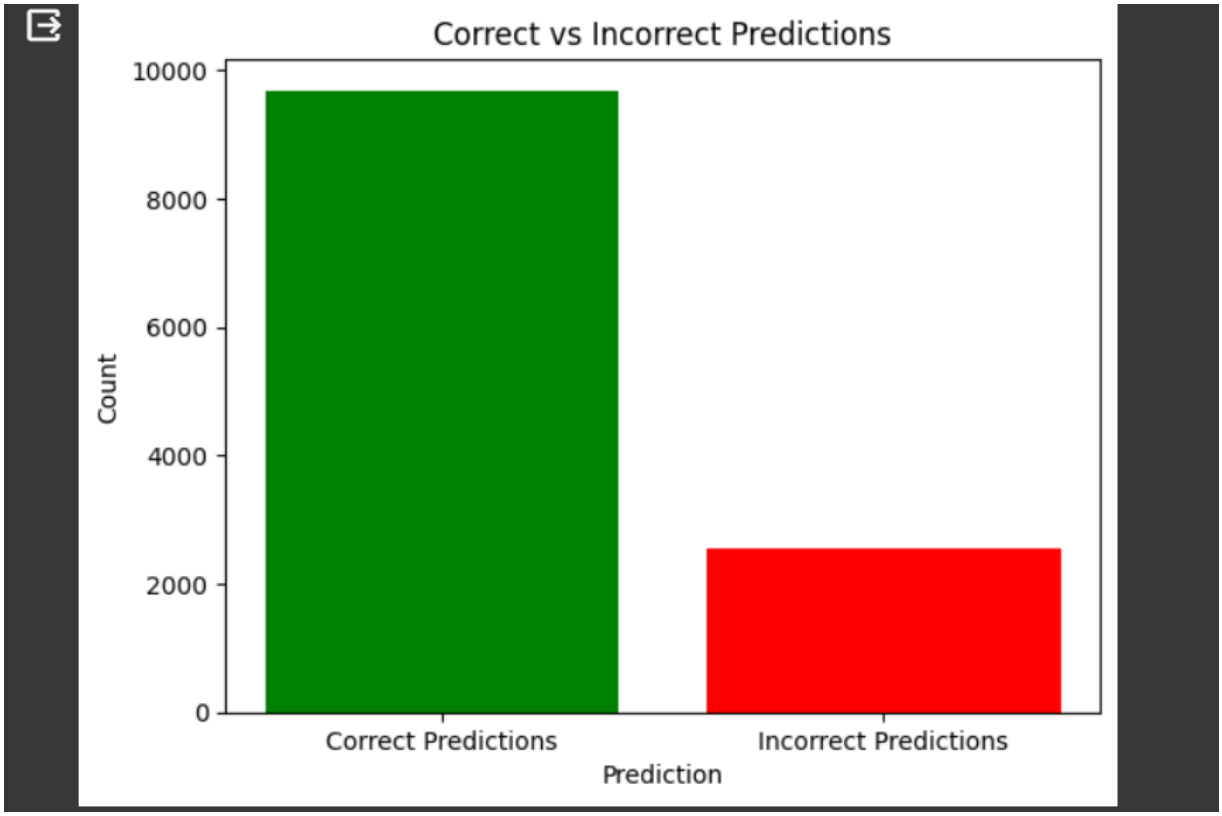
O/P:

```
  Enter tweet for sentiment analysis :  this was the first Borderlands session in a long time where i actually had a really satisfying fighti
Predicted Sentiment: ['Positive']
Accuracy: 79.23108384458078
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
# Plot confusion matrix
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
classes = np.unique(y)
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

for i in range(len(classes)):
    for j in range(len(classes)):
        plt.text(j, i, conf_matrix[i, j], horizontalalignment="center", color="white" if conf_matrix[i, j] > conf_matrix.max() / 2 else "black"
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print()
```



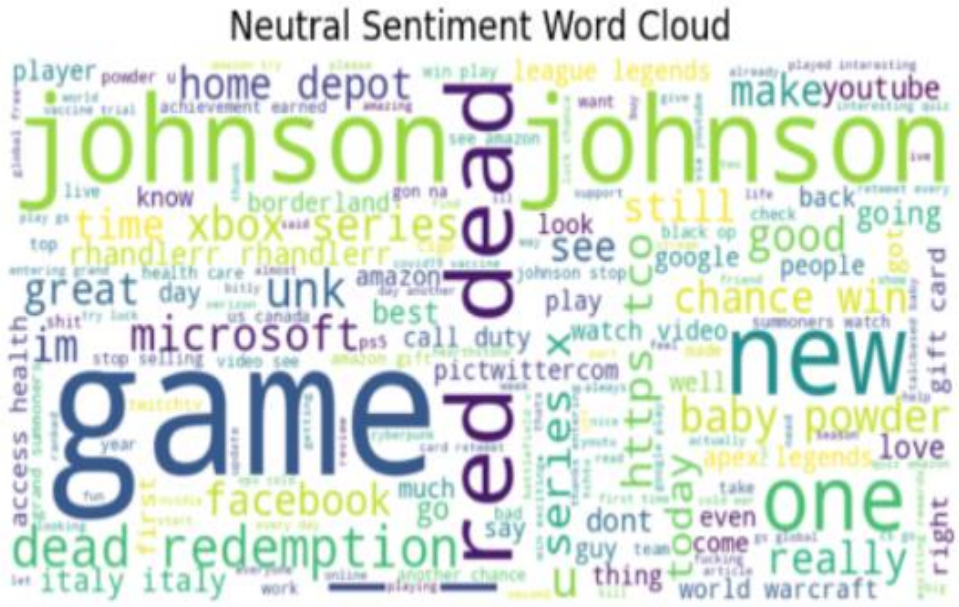

```
# Calculate count of correct and incorrect predictions
correct_count = np.sum(y_test == y_pred)
incorrect_count = len(y_test) - correct_count

# Plot bar graph
plt.bar(['Correct Predictions', 'Incorrect Predictions'], [correct_count, incorrect_count], color=['green', 'red'
plt.title('Correct vs Incorrect Predictions')
plt.xlabel('Prediction')
plt.ylabel('Count')
plt.show()
print()
```



WORD CLOUD:

A word cloud is a visualization technique that displays the frequency of words in a text data set, where the size of each word represents its frequency. It's commonly used to quickly identify the most prominent terms in a corpus and their relative importance. Word clouds are visually appealing and useful for gaining insights into the main themes or topics within a collection of documents.

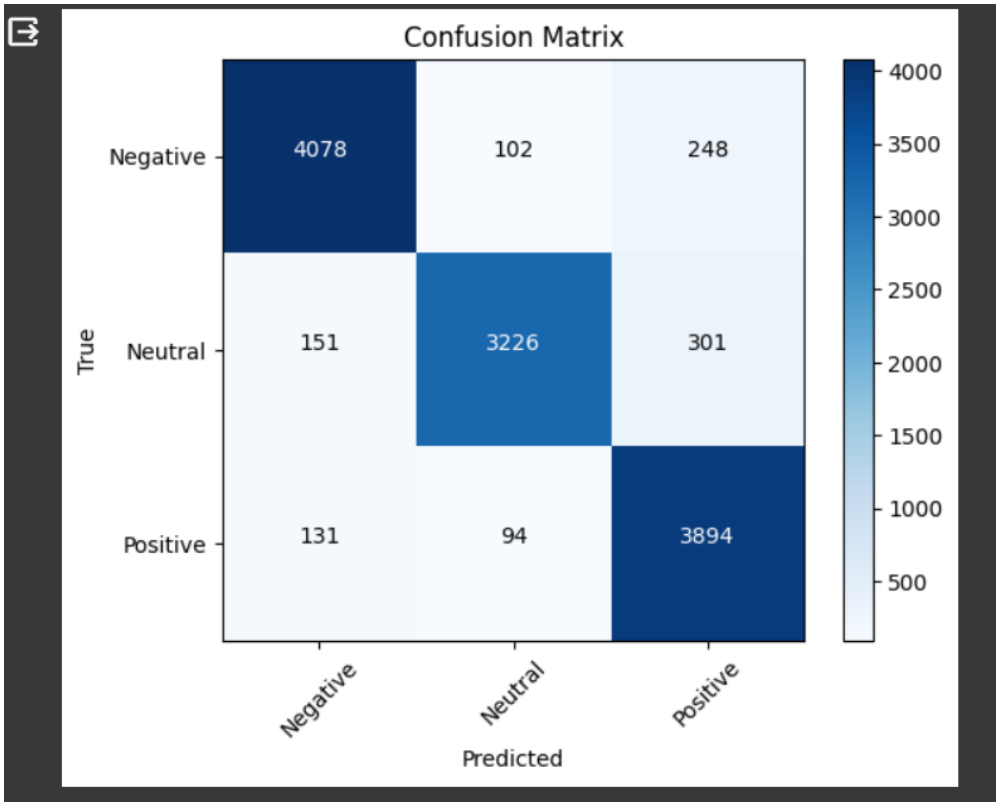
```python
# Collect text data for each sentiment category
# Check if 'text' column contains string data before joining
positive_text = ' '.join(data[data['TYPE'] == 'Positive']['TWEET'].astype(str))
negative_text = ' '.join(data[data['TYPE'] == 'Negative']['TWEET'].astype(str))
neutral_text = ' '.join(data[data['TYPE'] == 'Neutral']['TWEET'].astype(str))

# Generate word clouds
positive_wordcloud = WordCloud(width=800, height=400, background_color='white', min_font_size=10).generate(positive_text
negative_wordcloud = WordCloud(width=800, height=400, background_color='white', min_font_size=10).generate(negative_text
neutral_wordcloud = WordCloud(width=800, height=400, background_color='white', min_font_size=10).generate(neutral_text)

# Plot word clouds
plt.figure(figsize=(10, 10))

plt.subplot(2, 2, 1)
plt.imshow(positive_wordcloud, interpolation='bilinear')
plt.title('Positive Sentiment Word Cloud')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(negative_wordcloud, interpolation='bilinear')
plt.title('Negative Sentiment Word Cloud')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(neutral_wordcloud, interpolation='bilinear')
plt.title('Neutral Sentiment Word Cloud')
plt.axis('off')

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

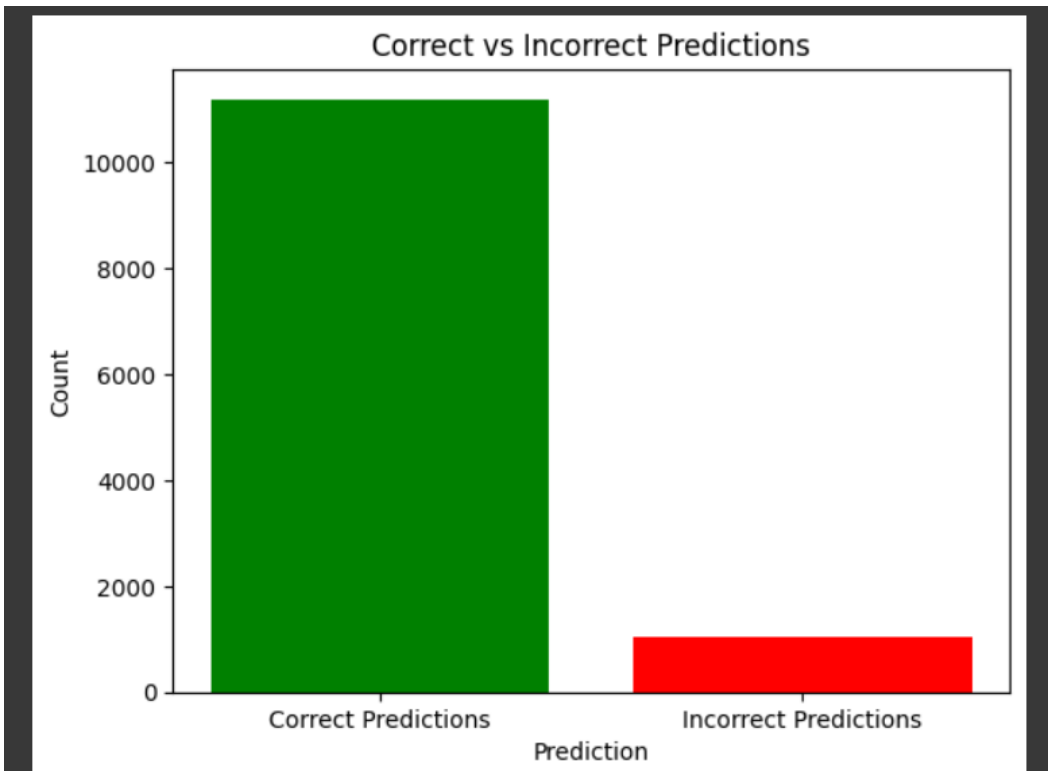


USING RANDOM FOREST CLASSIFIER:

```
# Train Naive Bayes classifier
classifier = RandomForestClassifier()                #92%
classifier.fit(X_train_vec, y_train)

# Predict
y_pred = classifier.predict(X_test_vec)
# Calculate accuracy (assuming 'type' has multiple classes)
accuracy = accuracy_score(y_test, y_pred)
# Assuming 'new_tweet' contains the new tweet as a string
new_tweet = input("  Enter tweet for sentiment analysis :  ")

# Preprocess the new tweet
cleaned_tweet = preprocess_text(new_tweet)
cleaned_tweet = remove_stopwords(new_tweet)
cleaned_tweet = lemmatize_text(new_tweet)

# Vectorize the preprocessed new tweet
vectorized_tweet = vectorizer.transform([cleaned_tweet])

# Predict the sentiment
predicted_sentiment = classifier.predict(vectorized_tweet)

print("Predicted Sentiment:", predicted_sentiment)
# Predict the sentiment

# Calculate and print the accuracy
print("Accuracy:", accuracy_score(y_test, y_pred)*100)
```

```
  Enter tweet for sentiment analysis :  Just realized the windows partition of my Mac ..
Predicted Sentiment: ['Neutral']
Accuracy: 91.59918200408997
```





CONCLUSION:

Our sentiment analysis project delves into the world of Twitter discussions, utilising a mix of powerful computer techniques to understand the emotions behind tweets.

We carefully prepared and organised the tweet data, then trained a special computer model called RandomForestClassifier. This model is very accurate, correctly predicting feelings in tweets 92% of the time!
Also we use Naive Bayes classifier which gives accuracy upto 79%

But our project goes beyond just guessing feelings. We also create colourful word cloud pictures to show which words are most common in positive, negative, or neutral tweets. These pictures help us see what topics people often talk about when they're feeling different emotions.

Overall, our project not only helps us understand feelings in tweets better but also gives us insights into digital sentiment trends. By using these techniques, we can make smarter decisions and better understand what's happening in the online world.

REFERENCES :

1.9. Naive Bayes — scikit-learn 1.4.2 documentation
sklearn.ensemble.RandomForestClassifier — scikit-learn 1.4.2 documentation

Colab link:
final.ipynb