In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv("labeled_data.csv")
```

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

Out[3]:

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24783 entries, 0 to 24782
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Unnamed: 0          24783 non-null  int64 
 1   count               24783 non-null  int64 
 2   hate_speech         24783 non-null  int64 
 3   offensive_language  24783 non-null  int64 
 4   neither             24783 non-null  int64 
 5   class               24783 non-null  int64 
 6   tweet               24783 non-null  object
dtypes: int64(6), object(1)
memory usage: 1.3+ MB
```

In [5]:
```
data = df.drop(columns = ["Unnamed: 0" , "count" , "hate_speech" , "offensive_language" , "neith
```

In [6]:
```
data["class"] = data["class"].astype("category")
```

PRE - PROCESSING

- TOKENIZATION
- LOWER CASING
- REMOVAL OF WHITESPACES
- REMOVAL OF STOP WORDS
- REMOVAL OF PUNCTUATION
- LEMMATIZATION

In [7]:
```
import string
import re
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.corpus import wordnet as wn
from nltk.stem.wordnet import WordNetLemmatizer

def text_cleaning(text):
    text = text.lower()
    exclude = set(string.punctuation) - {"'"}
    text = ''.join(ch for ch in text if ch not in exclude)
    text = re.sub(r"[^a-zA-Z0-9'\s]", "", text)
    return text

def stopword_removal(text):
    words = text.split()
    stop_words = stopwords.words('english')
    filtered_words = [word for word in words if word not in stop_words]
    text = ' '.join(filtered_words)
    return text

def tokenization(text):
    tokens = text.split()
    return tokens

def lemmatization(tokens):
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return lemmatized_tokens
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\chon2\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\chon2\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [8]:
```
def pre_proc(text):
    cleaned_text = text_cleaning(text)
    no_stopwords = stopword_removal(cleaned_text)
    tokens = tokenization(no_stopwords)
    lemmatized_tokens = lemmatization(tokens)

    return lemmatized_tokens
```

PRE - PROCESSING THE TWEET COLUMN

In [9]:
```
data["preproc_tweet"] = data["tweet"].apply(pre_proc)
```

In [10]:
```
data
```

Out[10]:

	class	tweet	preproc_tweet
0	2	!!! RT @mayasolovely: As a woman you shouldn't...	[rt, mayasolovely, woman, complain, cleaning, ...
1	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...	[rt, mleew17, boy, dat, coldtyga, dwn, bad, cu...
2	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...	[rt, urkindofbrand, dawg, rt, 80sbaby4life, ev...
3	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...	[rt, cganderson, vivabased, look, like, tranny]
4	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...	[rt, shenikaroberts, shit, hear, might, true, ...
...	...	...	...
24778	1	you's a muthaf***in lie “@LifeAsKing: @2...	[you's, muthafin, lie, 8220lifeasking, 20pearl...
24779	2	you've gone and broke the wrong heart baby, an...	[gone, broke, wrong, heart, baby, drove, redne...
24780	1	young buck wanna eat!!.. dat nigguh like I ain...	[young, buck, wanna, eat, dat, nigguh, like, a...
24781	1	youu got wild bitches tellin you lies	[youu, got, wild, bitch, tellin, lie]
24782	2	~~Ruffled | Ntac Eileen Dahlia - Beautiful col...	[ruffled, ntac, eileen, dahlia, beautiful, col...

24783 rows × 3 columns

In [11]:
```
data['class'].unique()
```

Out[11]:
```
[2, 1, 0]
Categories (3, int64): [0, 1, 2]
```

In [12]:
```
X = data["preproc_tweet"]
y = data["class"]
```

Word2Vect - CONTINUOUS BAG OF WORDS

STEPS:

- TRAINING THE WORD2VECT - CBOW MODEL
- AVERAGE WORD VECTORS FOR A SENTENCE
- TRANSFORMING TRAINING AND TEST DATA TO AVERAGE WORD VECTORS
- ENCODE LABELS TO NUMERIC VALUES
- CLASSIFICATION USING LOGISTIC REGRESSION
- EVALUATING THE MODEL
  - ACCURACY
  - CONFUSION MATRIX
  - CLASSIFICATION REPORT

In [13]:
```
from gensim.models import Word2Vec
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

w2v_model_cbow = Word2Vec(sentences=X_train, vector_size=100, window=5, min_count=1, workers=4, 
```

In [14]:
```
def average_word_vectors(words, model, num_features):
    feature_vector = np.zeros((num_features,), dtype="float32")
    nwords = 0.

    for word in words:
        if word in model.wv:
            nwords += 1
            feature_vector = np.add(feature_vector, model.wv[word])

    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector
```

In [15]:
```
X_train_cbow = [average_word_vectors(tokens, w2v_model_cbow, 100) for tokens in X_train]
X_test_cbow = [average_word_vectors(tokens, w2v_model_cbow, 100) for tokens in X_test]
```

In [16]:
```
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

In [17]:
```
classifier_cbow = LogisticRegression(max_iter=1000)
classifier_cbow.fit(X_train_cbow, y_train_encoded)
```

Out[17]:
```
LogisticRegression(max_iter=1000)
```

In [18]:
```
y_pred_cbow = classifier_cbow.predict(X_test_cbow)
accuracy_cbow = accuracy_score(y_test_encoded, y_pred_cbow)
print(f"Word2Vec CBOW Model Accuracy: {accuracy_cbow}")
```

```
Word2Vec CBOW Model Accuracy: 0.8365947145450877
```

In [19]:
```
conf_matrix = confusion_matrix(y_test_encoded, y_pred_cbow)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[   0  215   75]
 [   0 3666  166]
 [   0  354  481]]
```

In [20]:
```
class_report = classification_report(y_test_encoded, y_pred_cbow)
print("Classification Report:")
print(class_report)
```

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

           0       0.00      0.00      0.00       290
           1       0.87      0.96      0.91      3832
           2       0.67      0.58      0.62       835

    accuracy                           0.84      4957
   macro avg       0.51      0.51      0.51      4957
weighted avg       0.78      0.84      0.81      4957


C:\Users\chon2\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\chon2\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\chon2\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
```

Word2Vect - SKIP GRAM

STEPS:

- TRAINING THE WORD2VECT - CBOW MODEL
- AVERAGE WORD VECTORS FOR A SENTENCE
- TRANSFORMING TRAINING AND TEST DATA TO AVERAGE WORD VECTORS
- ENCODE LABELS TO NUMERIC VALUES
- CLASSIFICATION USING LOGISTIC REGRESSION
- EVALUATING THE MODEL
  - ACCURACY
  - CONFUSION MATRIX
  - CLASSIFICATION REPORT

In [21]:
```
from gensim.models import Word2Vec
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

w2v_model_sg = Word2Vec(sentences=X_train, vector_size=100, window=5, min_count=1, workers=4, sg
```

In [22]:
```
def average_word_vectors(words, model, num_features):
    feature_vector = np.zeros((num_features,), dtype="float32")
    nwords = 0.

    for word in words:
        if word in model.wv:
            nwords += 1
            feature_vector = np.add(feature_vector, model.wv[word])

    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector
```

In [23]:
```
X_train_sg = [average_word_vectors(tokens, w2v_model_sg, 100) for tokens in X_train]
X_test_sg = [average_word_vectors(tokens, w2v_model_sg, 100) for tokens in X_test]
```

In [24]:
```
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

In [25]:
```
classifier_sg = LogisticRegression(max_iter=1000)
classifier_sg.fit(X_train_sg, y_train_encoded)
```

Out[25]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [26]:
```
y_pred_sg = classifier_sg.predict(X_test_sg)
accuracy_sg = accuracy_score(y_test_encoded, y_pred_sg)
print(f"Word2Vec Skip-gram Model Accuracy: {accuracy_sg}")
```

```
Word2Vec Skip-gram Model Accuracy: 0.8601977002219084
```

In [27]:
```
conf_matrix_sg = confusion_matrix(y_test_encoded, y_pred_sg)
print("Confusion Matrix:")
print(conf_matrix_sg)
```

```
Confusion Matrix:
[[   8  222   60]
 [  19 3676  137]
 [   4  251  580]]
```

In [28]:
```
class_report_sg = classification_report(y_test_encoded, y_pred_sg)
print("Classification Report:")
print(class_report_sg)
```

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

           0       0.26      0.03      0.05       290
           1       0.89      0.96      0.92      3832
           2       0.75      0.69      0.72       835

    accuracy                           0.86      4957
   macro avg       0.63      0.56      0.56      4957
weighted avg       0.83      0.86      0.84      4957
```

FAST TEXT

STEPS:

- LOADING THE FASTTEXT PRE-TRAINED MODEL
- FASTTEXT FOR A SENTENCE
- TRANSFORMING TRAINING AND TEST DATA TO FASTTEXT VECTORS
- ENCODE LABELS TO NUMERIC VALUES
- CLASSIFICATION USING LOGISTIC REGRESSION
- EVALUATING THE MODEL
  - ACCURACY
  - CONFUSION MATRIX
  - CLASSIFICATION REPORT

In [29]:
```
from gensim.models import FastText
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
import numpy as np

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

fasttext_model = FastText.load_fasttext_format('cc.en.300.bin')
```

```
C:\Users\chon2\AppData\Local\Temp\ipykernel_2264\1509263375.py:12: DeprecationWarning: Call to
deprecated `load_fasttext_format` (use load_facebook_vectors (to use pretrained embeddings) or
load_facebook_model (to continue training with the loaded full model, more RAM) instead).
  fasttext_model = FastText.load_fasttext_format('cc.en.300.bin')
```

In [30]:
```
def get_fasttext_vectors(sentences, model):
    vectors = []
    for sentence in sentences:
        vector = np.mean([model.wv[word] for word in sentence if word in model.wv], axis=0)
        vectors.append(vector)
    return vectors
```

In [31]:
```
X_train_fasttext = get_fasttext_vectors(X_train, fasttext_model)
X_test_fasttext = get_fasttext_vectors(X_test, fasttext_model)
```

In [32]:
```
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

In [33]:
```
classifier_fasttext = LogisticRegression(max_iter=1000)
classifier_fasttext.fit(X_train_fasttext, y_train_encoded)
```

Out[33]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [34]:
```
y_pred_fasttext = classifier_fasttext.predict(X_test_fasttext)
accuracy_fasttext = accuracy_score(y_test_encoded, y_pred_fasttext)
print(f"FastText Model Accuracy: {accuracy_fasttext}")
```

```
FastText Model Accuracy: 0.8793625176518055
```

In [35]:
```
conf_matrix_fasttext = confusion_matrix(y_test_encoded, y_pred_fasttext)
print("Confusion Matrix:")
print(conf_matrix_fasttext)
```

```
Confusion Matrix:
[[  33  223   34]
 [  31 3698  103]
 [   7  200  628]]
```

In [36]:
```
class_report_fasttext = classification_report(y_test_encoded, y_pred_fasttext)
print("Classification Report:")
print(class_report_fasttext)
```

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

           0       0.46      0.11      0.18       290
           1       0.90      0.97      0.93      3832
           2       0.82      0.75      0.78       835

    accuracy                           0.88      4957
   macro avg       0.73      0.61      0.63      4957
weighted avg       0.86      0.88      0.86      4957
```

CONVOLUTIONAL NEURAL NETWORK

STEPS:

- TOKENIZE AND PAD SEQUENCES
- ENCODE LABELS TO NUMERIC VALUES
- BUILDING CNN MODEL
- EVALUATE AND PREDICT
- DECODE NUMERIC VALUES TO LABELS
- METRICS
  - ACCURACY
  - CONFUSION MATRIX
  - CLASSIFICATION REPORT

In [37]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index) + 1

max_length = max(len(seq) for seq in X_train_sequences)

X_train_padded = pad_sequences(X_train_sequences, maxlen=max_length)
X_test_padded = pad_sequences(X_test_sequences, maxlen=max_length)
```

In [38]:
```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

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

In [39]:
```
embedding_dim = 100
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length))
model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
```

In [40]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

In [41]:
```
model.fit(X_train_padded, y_train_encoded, epochs=5, batch_size=32, validation_split=0.2)
```

```
Epoch 1/5
496/496 [==============================] - 13s 25ms/step - loss: 3.9180 - accuracy: 0.1940 - va
l_loss: 4.1589 - val_accuracy: 0.0517
Epoch 2/5
496/496 [==============================] - 11s 23ms/step - loss: 4.1589 - accuracy: 0.0590 - va
l_loss: 4.1589 - val_accuracy: 0.0517
Epoch 3/5
496/496 [==============================] - 11s 23ms/step - loss: 4.1589 - accuracy: 0.0590 - va
l_loss: 4.1589 - val_accuracy: 0.0517
Epoch 4/5
496/496 [==============================] - 11s 23ms/step - loss: 4.1589 - accuracy: 0.0590 - va
l_loss: 4.1589 - val_accuracy: 0.0517
Epoch 5/5
496/496 [==============================] - 11s 23ms/step - loss: 4.1589 - accuracy: 0.0590 - va
l_loss: 4.1589 - val_accuracy: 0.0517
```

Out[41]:
```
<keras.callbacks.History at 0x21fdd263ac0>
```

In [42]:
```
loss, accuracy = model.evaluate(X_test_padded, y_test_encoded)
```

```
155/155 [==============================] - 0s 2ms/step - loss: 4.1589 - accuracy: 0.0585
```

In [43]:
```
y_pred_probs = model.predict(X_test_padded)
y_pred = np.argmax(y_pred_probs, axis=1)
```

```
155/155 [==============================] - 0s 2ms/step
```

In [44]:
```
y_test_labels = label_encoder.inverse_transform(y_test_encoded)
y_pred_labels = label_encoder.inverse_transform(y_pred)
```

In [45]:
```
accuracy = accuracy_score(y_test_labels, y_pred_labels)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.05850312689126488
```

In [46]:
```
conf_matrix = confusion_matrix(y_test_labels, y_pred_labels)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[ 290    0    0]
 [3832    0    0]
 [ 835    0    0]]
```

In [47]:

```
class_report = classification_report(y_test_labels, y_pred_labels)
print("Classification Report:")
print(class_report)
```

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

           0       0.06      1.00      0.11       290
           1       0.00      0.00      0.00      3832
           2       0.00      0.00      0.00       835

    accuracy                           0.06      4957
   macro avg       0.02      0.33      0.04      4957
weighted avg       0.00      0.06      0.01      4957


C:\Users\chon2\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\chon2\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\chon2\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classi
fication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
```

RECURRENT NEURAL NETWORK

STEPS:

- TOKENIZE AND PAD SEQUENCES
- ENCODE LABELS TO NUMERIC VALUES
- BUILDING RNN MODEL
- EVALUATE AND PREDICT
- DECODE NUMERIC VALUES TO LABELS
- METRICS
  - ACCURACY
  - CONFUSION MATRIX
  - CLASSIFICATION REPORT

In [48]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index) + 1

max_length = max(len(seq) for seq in X_train_sequences)

X_train_padded = pad_sequences(X_train_sequences, maxlen=max_length)
X_test_padded = pad_sequences(X_test_sequences, maxlen=max_length)
```

In [49]:
```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42
```

In [50]:
```
embedding_dim = 100
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length))
model.add(SimpleRNN(units=64, activation='relu'))
model.add(Dense(64, activation='relu'))
```

In [51]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

In [52]:
```
model.fit(X_train_padded, y_train_encoded, epochs=5, batch_size=32, validation_split=0.2)
```

```
Epoch 1/5
496/496 [==============================] - 13s 23ms/step - loss: 0.7662 - accuracy: 0.7639 - va
l_loss: 0.5860 - val_accuracy: 0.8058
Epoch 2/5
496/496 [==============================] - 11s 23ms/step - loss: 0.4062 - accuracy: 0.8992 - va
l_loss: 0.5590 - val_accuracy: 0.8255
Epoch 3/5
496/496 [==============================] - 11s 23ms/step - loss: 0.2375 - accuracy: 0.9444 - va
l_loss: 0.9007 - val_accuracy: 0.8719
Epoch 4/5
496/496 [==============================] - 11s 23ms/step - loss: 0.1785 - accuracy: 0.9663 - va
l_loss: 1.4074 - val_accuracy: 0.8336
Epoch 5/5
496/496 [==============================] - 11s 23ms/step - loss: 0.8098 - accuracy: 0.8436 - va
l_loss: 3.2249 - val_accuracy: 0.5436
```

Out[52]:
```
<keras.callbacks.History at 0x21fa036df40>
```

In [53]:
```
loss, accuracy = model.evaluate(X_test_padded, y_test_encoded)
```

```
155/155 [==============================] - 0s 2ms/step - loss: 3.0184 - accuracy: 0.5568
```

In [54]:
```
y_pred_probs = model.predict(X_test_padded)
y_pred = np.argmax(y_pred_probs, axis=1)
```

```
155/155 [==============================] - 0s 2ms/step
```

In [55]:
```
y_test_labels = label_encoder.inverse_transform(y_test_encoded)
y_pred_labels = label_encoder.inverse_transform(y_pred)
```

In [56]:
```
accuracy = accuracy_score(y_test_labels, y_pred_labels)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.5567883800685899
```

In [57]:
```
conf_matrix = confusion_matrix(y_test_labels, y_pred_labels)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[ 145   76   69]
 [1451 1956  425]
 [ 113   63  659]]
```

In [58]:
```
class_report = classification_report(y_test_labels, y_pred_labels)
print("Classification Report:")
print(class_report)
```

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

           0       0.08      0.50      0.15       290
           1       0.93      0.51      0.66      3832
           2       0.57      0.79      0.66       835

    accuracy                           0.56      4957
   macro avg       0.53      0.60      0.49      4957
weighted avg       0.82      0.56      0.63      4957
```