Import the required libraries

In [1]:

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.regularizers import l2
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional, SimpleRNN, Dense, Embe
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, roc_
from tensorflow.keras.utils import plot_model
import time
import warnings
warnings.filterwarnings('ignore')
from tensorflow import keras
from tensorflow.keras import layers
from nltk.corpus import stopwords
import nltk
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer, PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
import unicodedata
from num2words import num2words

```

```
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\Nitin\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Data Acquisition

In [2]:

```
data = pd.read_json('News_Category_Dataset_v3.json', lines=True)
data.head()
```

Out[2]:

	link	headline	category	short_description	authors	c
0	https://www.huffpost.com/entry/covid-boosters-...	Over 4 Million Americans Roll Up Sleeves For O...	U.S. NEWS	Health experts said it is too early to predict...	Carla K. Johnson, AP	2(09
1	https://www.huffpost.com/entry/american-airlin...	American Airlines Flyer Charged, Banned For Li...	U.S. NEWS	He was subdued by passengers and crew when he ...	Mary Papenfuss	2(09
2	https://www.huffpost.com/entry/funniest-tweets...	23 Of The Funniest Tweets About Cats And Dogs ...	COMEDY	"Until you have a dog you don't understand wha...	Elyse Wanshel	2(09
3	https://www.huffpost.com/entry/funniest-parent...	The Funniest Tweets From Parents This Week (Se...	PARENTING	"Accidentally put grown-up toothpaste on my to...	Caroline Bologna	2(09
4	https://www.huffpost.com/entry/amy-cooper-lose...	Woman Who Called Cops On Black Bird-Watcher Lo...	U.S. NEWS	Amy Cooper accused investment firm Franklin Te...	Nina Golgowski	2(09

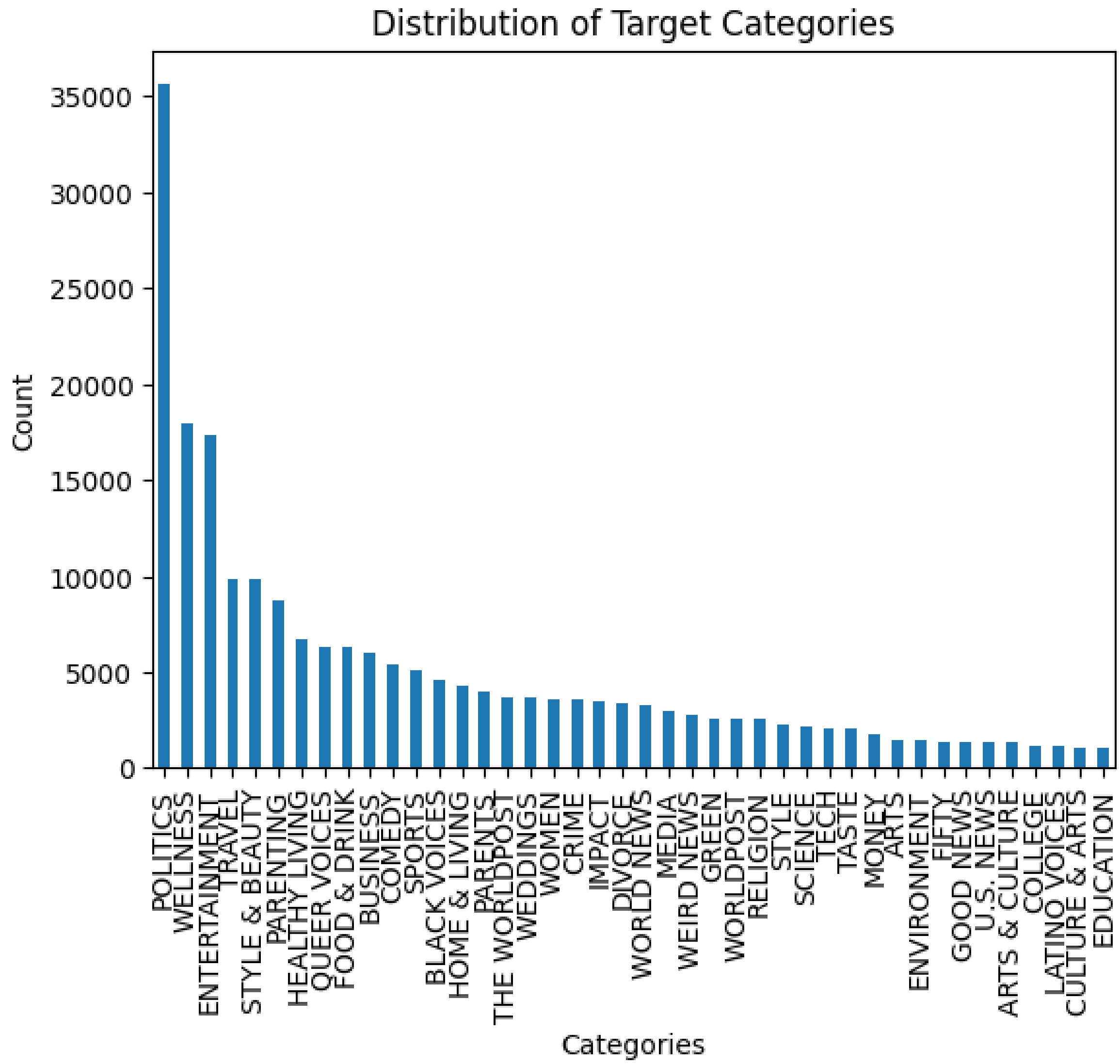
Dataset observations as

- Size of the dataset
- What type of data attributes are there?
- What are you classifying?
- Plot the distribution of the categories of the target / label.

In [3]:

```
# Size of the dataset
dataset_size = data.shape[0]  # Number of rows in the dataset
print("Size of the dataset:", dataset_size)

# Data attributes and their types
data_attributes = data.dtypes
print("\nType of data attributes:")
print(data_attributes)

# Data attributes
data_attributes = list(data.columns)
print("\nData attributes:", data_attributes)

# Target variable
target_variable = 'category'  # Replace with the actual name of the target
target_classes = data[target_variable].unique()

# What are you classifying?
print("\nClassifying:", target_classes)

# Plot the distribution of target categories
data[target_variable].value_counts().plot(kind='bar')
plt.xlabel('Categories')
plt.ylabel('Count')
plt.title('Distribution of Target Categories')
plt.show()
```

```
Size of the dataset: 209527

Type of data attributes:
link                             object
headline                         object
category                         object
short_description                object
authors                          object
date                     datetime64[ns]
dtype: object

Data attributes: ['link', 'headline', 'category', 'short_description', 'autho
rs', 'date']

Classifying: ['U.S. NEWS' 'COMEDY' 'PARENTING' 'WORLD NEWS' 'CULTURE & ARTS'
'TECH'
 'SPORTS' 'ENTERTAINMENT' 'POLITICS' 'WEIRD NEWS' 'ENVIRONMENT'
 'EDUCATION' 'CRIME' 'SCIENCE' 'WELLNESS' 'BUSINESS' 'STYLE & BEAUTY'
 'FOOD & DRINK' 'MEDIA' 'QUEER VOICES' 'HOME & LIVING' 'WOMEN'
 'BLACK VOICES' 'TRAVEL' 'MONEY' 'RELIGION' 'LATINO VOICES' 'IMPACT'
 'WEDDINGS' 'COLLEGE' 'PARENTS' 'ARTS & CULTURE' 'STYLE' 'GREEN' 'TASTE'
 'HEALTHY LIVING' 'THE WORLDPOST' 'GOOD NEWS' 'WORLDPOST' 'FIFTY' 'ARTS'
 'DIVORCE']
```



Plot Obeservation

Above plot shows the data imbalance. To deal with data imbalance we can group by the category

In [4]:

```
categories = data['category'].value_counts().index

def groupper(grouplist,name):
    for ele in categories:
        if ele in grouplist:
            data.loc[data['category'] == ele, 'category'] = name


groupper( grouplist= ['WELLNESS', 'HEALTHY LIVING','HOME & LIVING','STYLE

groupper( grouplist= [ 'PARENTING', 'PARENTS' ,'EDUCATION' ,'COLLEGE'] , 

groupper( grouplist= ['SPORTS','ENTERTAINMENT' , 'COMEDY','WEIRD NEWS','A

groupper( grouplist= ['TRAVEL', 'ARTS & CULTURE','CULTURE & ARTS','FOOD &

groupper( grouplist= ['WOMEN','QUEER VOICES', 'LATINO VOICES', 'BLACK VOI

groupper( grouplist= ['BUSINESS' ,  'MONEY'] , name =  'BUSINESS-MONEY')

groupper( grouplist= ['THE WORLDPOST' , 'WORLDPOST' , 'WORLD NEWS', 'U.S.

groupper( grouplist= ['ENVIRONMENT' ,'GREEN'] , name =  'ENVIRONMENT')

groupper( grouplist= ['TECH', 'SCIENCE'] , name =  'SCIENCE AND TECH')

groupper( grouplist= ['FIFTY' , 'IMPACT' ,'GOOD NEWS','CRIME'] , name = 

groupper( grouplist= ['WEDDINGS', 'DIVORCE',  'RELIGION','MEDIA'] , name 

print("We have a total of {} categories now".format(data['category'].nuni
data['category'].value_counts()

# # Plot the distribution of target categories
# data[target_variable].value_counts().plot(kind='bar')
# plt.xlabel('Categories')
# plt.ylabel('Count')
# plt.title('Distribution of Target Categories')
# plt.show()

```

```
We have a total of 12 categories now
```

Out[4]:
```
category
LIFESTYLE AND WELLNESS          41027
POLITICS                        35602
SPORTS AND ENTERTAINMENT        32125
TRAVEL-TOURISM & ART-CULTURE    20749
EMPOWERED VOICES                15632
PARENTING AND EDUCATION         14904
MISC                            12600
WORLDNEWS                       10919
GENERAL                          9845
BUSINESS-MONEY                   7748
SCIENCE AND TECH                 4310
ENVIRONMENT                      4066
Name: count, dtype: int64
```

Applied pre-processing techiniques as:

- to remove duplicate data
- to impute or remove missing data
- to remove data inconsistencies
- Encode categorical data
- Normalize the data
- Feature Engineering
- Stop word removal, lemmatiation, stemming, vectorization

In [5]:

```
# Check for duplicate data
print("Number of duplicate rows before preprocessing:", data.duplicated()

# Remove duplicate data
data.drop_duplicates(inplace=True)

# Check for missing data
print("Number of missing values before preprocessing:")
print(data.isnull().sum())

# Impute or remove missing data
# Option 1: Remove rows with missing values
data.dropna(inplace=True)

# Option 2: Impute missing values with mean or median
# Uncomment the following lines to use mean imputation
# data.fillna(data.mean(), inplace=True)

# Option 3: Impute missing values with mode (for categorical variables)
# Uncomment the following lines to use mode imputation
# data.fillna(data.mode().iloc[0], inplace=True)

# Check for data inconsistencies and perform necessary corrections
# Option 1: Manual correction based on specific rules
# Uncomment the following lines and modify as per your requirements
# data['column_name'] = data['column_name'].apply(lambda x: x.replace('inc

# Option 2: Use regular expressions for pattern matching and correction
# Uncomment the following lines and modify as per your requirements
# import re
# data['column_name'] = data['column_name'].apply(lambda x: re.sub(r'patte

# Check the preprocessed dataset
print("Number of duplicate rows after preprocessing:", data.duplicated().s
print("Number of missing values after preprocessing:")
print(data.isnull().sum())
```

```
Number of duplicate rows before preprocessing: 13
Number of missing values before preprocessing:
link                 0
headline             0
category             0
short_description    0
authors              0
date                 0
dtype: int64
Number of duplicate rows after preprocessing: 0
Number of missing values after preprocessing:
link                 0
headline             0
category             0
short_description    0
authors              0
date                 0
dtype: int64
```

In [6]:
```
# As per suggetion using two column
data = data[['headline', 'category']]
```

In [7]:
```
# Function for lemmatization
def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in text.spli
    return ' '.join(lemmatized_tokens)

# Function for stemming
def stem_text(text):
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in text.split()]
    return ' '.join(stemmed_tokens)

# Function for vectorization
def vectorize_text(text):
    vectorizer = TfidfVectorizer()
    text_features = vectorizer.fit_transform(text)
    return text_features

# Function for preprocess the text
def preprocess_text(text, lower_case=True, convert_numbers=True, remove_pu
                    remove_accents=True, remove_whitespace=True, expand_al
                    remove_stopwords=True, sparse_terms=[], specific_word

    # Convert all letters to lower or upper case
    if lower_case:
        text = text.lower()
    else:
        text = text.upper()

    # Converting numbers into words or removing numbers
    if convert_numbers:
        # Replace numbers with their word equivalents using num2words libr
        tokens = []
        for word in text.split():
            if word.isdigit():
                word = num2words(int(word))
            tokens.append(word)
        text = ' '.join(tokens)
    else:
        # Remove numbers using regular expression
        text = re.sub(r'\d+', '', text)

    # Removing punctuations, accent marks, and other diacritics
    if remove_punctuation:
        text = re.sub(r'[^\w\s]', '', text)
    if remove_accents:
        text = ''.join(c for c in unicodedata.normalize('NFD', text)
                       if not unicodedata.combining(c))

    # Removing white spaces
    if remove_whitespace:
        text = re.sub(r'\s+', ' ', text.strip())

    # Expanding abbreviations
    if expand_abbreviations:
        abbreviations = {'e.g.': 'for example', 'i.e.': 'that is', 'etc.'
        for abbr, expanded in abbreviations.items():
```

```
            text = text.replace(abbr, expanded)

    # Removing stop words, sparse terms, and specific words
    if remove_stopwords:
        stop_words = set(stopwords.words('english'))
        text = ' '.join(word for word in text.split() if word not in stop_
    text = ' '.join(word for word in text.split() if word not in sparse_t
    text = ' '.join(word for word in text.split() if word not in specific_

    return text


# Apply the preprocessing function to a specific column
text_col = 'headline'  # Specify the text column to preprocess
data['text'] = data[text_col].apply(preprocess_text)

# Apply lemmatization to a specific column
text_col = 'text'  # Specify the text column to preprocess
data[text_col] = data[text_col].apply(lemmatize_text)

# # Apply stemming to a specific column
# text_col = 'text'  # Specify the text column to preprocess
# data[text_col] = data[text_col].apply(stem_text)

# # Apply vectorization to a specific column
# text_col = 'text'  # Specify the text column to preprocess
# text_features = vectorize_text(data[text_col])

# Check the preprocessed dataset and vectorized features
display(data.head())

# display(text_features.toarray())
start_time = time.time()
# # Save the preprocessed dataset
data.to_csv('preprocessed_Dataset.csv', index=False)

end_time = time.time() - start_time
print("Time taken for text preprocessing: ", end_time)
```

	headline	category	text
0	Over 4 Million Americans Roll Up Sleeves For O...	WORLDNEWS	four million american roll sleeve omicrontarge...
1	American Airlines Flyer Charged, Banned For Li...	WORLDNEWS	american airline flyer charged banned life pun...
2	23 Of The Funniest Tweets About Cats And Dogs ...	SPORTS AND ENTERTAINMENT	twentythree funniest tweet cat dog week sept 1723
3	The Funniest Tweets From Parents This Week (Se...	PARENTING AND EDUCATION	funniest tweet parent week sept 1723
4	Woman Who Called Cops On Black Bird-Watcher Lo...	WORLDNEWS	woman called cop black birdwatcher loses lawsu...

```
Time taken for text preprocessing:  0.6760261058807373
```

```
# Identify the target variables.
```

```
Separate the data front the target such that the dataset is in the form of (X,y) or (Features, Label)
Encode the target variable or perform one-hot encoding on the target or any other as and if required.
```

Identify the target variables

- Separate the data front the target such that the dataset is in the form of (X,y) or (Features, Label)
- Discretize / Encode the target variable or perform one-hot encoding on the target or any other as and if required.

In [8]:

```
# Preprocessing: Convert the labels to numeric values
label_encoder = LabelEncoder()
data['category'] = label_encoder.fit_transform(data['category'])
classes = list(label_encoder.classes_)
print(classes)

# Preprocessing: Split the dataset into input features and labels
X = data['text']
y = data['category']
```

```
['BUSINESS-MONEY', 'EMPOWERED VOICES', 'ENVIRONMENT', 'GENERAL', 'LIFESTYLE A
ND WELLNESS', 'MISC', 'PARENTING AND EDUCATION', 'POLITICS', 'SCIENCE AND TEC
H', 'SPORTS AND ENTERTAINMENT', 'TRAVEL-TOURISM & ART-CULTURE', 'WORLDNEWS']
```

Split the data into training set and testing set

In [9]:

```
# Split the data into training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, 

# Check the shapes of the training and testing sets
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (167611,)
y_train shape: (167611,)
X_test shape: (41903,)
y_test shape: (41903,)
```

LSTM Model

In [10]:

```
# Tokenization and vectorization using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()

# Padding sequences
max_sequence_length = X_train.shape[1]

# Reshape the input data to be 3-dimensional (samples, time steps, feature
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Define the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(1, max_sequence_le
model.add(Dropout(0.2))
model.add(LSTM(64))
model.add(Dropout(0.2))
model.add(Dense(len(label_encoder.classes_), activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', me

# Print the model summary
print(model.summary())

# Define early stopping
early_stopping = EarlyStopping(patience=3, restore_best_weights=True)

# Train the model with early stopping
model.fit(X_train, y_train, batch_size=32, epochs=5, validation_data=(X_te

# Generate predictions on the test set
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))


```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 1, 128)            2626048

 dropout (Dropout)           (None, 1, 128)            0

 lstm_1 (LSTM)               (None, 64)                49408

 dropout_1 (Dropout)         (None, 64)                0

 dense (Dense)               (None, 12)                780

=================================================================
Total params: 2,676,236
Trainable params: 2,676,236
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/3
5238/5238 [==============================] - 81s 15ms/step - loss: 1.3865 - a
ccuracy: 0.5699 - val_loss: 1.1455 - val_accuracy: 0.6387
Epoch 2/3
5238/5238 [==============================] - 75s 14ms/step - loss: 1.0996 - a
ccuracy: 0.6553 - val_loss: 1.1137 - val_accuracy: 0.6451
Epoch 3/3
5238/5238 [==============================] - 74s 14ms/step - loss: 1.0352 - a
ccuracy: 0.6692 - val_loss: 1.1066 - val_accuracy: 0.6476
1310/1310 [==============================] - 6s 3ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.46      0.49      1532
           1       0.60      0.45      0.51      3085
           2       0.48      0.40      0.44       815
           3       0.45      0.34      0.38      1952
           4       0.64      0.80      0.71      8330
           5       0.71      0.58      0.64      2564
           6       0.62      0.62      0.62      2933
           7       0.74      0.77      0.76      6992
           8       0.46      0.44      0.45       811
           9       0.64      0.64      0.64      6393
          10       0.70      0.68      0.69      4259
          11       0.64      0.62      0.63      2237

    accuracy                           0.65     41903
   macro avg       0.60      0.57      0.58     41903
weighted avg       0.64      0.65      0.64     41903
```

Bidirectional LSTM Model

In [12]:

```
# Split the data into training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Tokenization and vectorization using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()

# Padding sequences
max_sequence_length = X_train.shape[1]

# Reshape the input data to be 3-dimensional (samples, time steps, feature
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Define the LSTM model with Bidirectional layer
model = Sequential()
model.add(Bidirectional(LSTM(128, return_sequences=True), input_shape=(1,
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.2))
model.add(Dense(len(label_encoder.classes_), activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', me

# Print the model summary
print(model.summary())

# Define early stopping
early_stopping = EarlyStopping(patience=3, restore_best_weights=True)

# Train the model with early stopping
model.fit(X_train, y_train, batch_size=32, epochs=5, validation_data=(X_te

# Generate predictions on the test set
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 bidirectional (Bidirectiona  (None, 1, 256)           5252096   
 l)                                                              
                                                                 
 dropout_2 (Dropout)         (None, 1, 256)            0         
                                                                 
 bidirectional_1 (Bidirectio  (None, 128)              164352    
 nal)                                                            
                                                                 
 dropout_3 (Dropout)         (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 12)                1548      
                                                                 
=================================================================
Total params: 5,417,996
Trainable params: 5,417,996
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
5238/5238 [==============================] - 148s 26ms/step - loss: 1.3114 -
accuracy: 0.5917 - val_loss: 1.1265 - val_accuracy: 0.6405
Epoch 2/5
5238/5238 [==============================] - 129s 25ms/step - loss: 1.0681 -
accuracy: 0.6598 - val_loss: 1.1045 - val_accuracy: 0.6472
Epoch 3/5
5238/5238 [==============================] - 133s 25ms/step - loss: 1.0040 -
accuracy: 0.6746 - val_loss: 1.1066 - val_accuracy: 0.6491
Epoch 4/5
5238/5238 [==============================] - 128s 24ms/step - loss: 0.9539 -
accuracy: 0.6878 - val_loss: 1.1147 - val_accuracy: 0.6474
Epoch 5/5
5238/5238 [==============================] - 129s 25ms/step - loss: 0.9022 -
accuracy: 0.6998 - val_loss: 1.1186 - val_accuracy: 0.6498
1310/1310 [==============================] - 9s 5ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.46      0.49      1532
           1       0.58      0.46      0.51      3085
           2       0.50      0.38      0.43       815
           3       0.46      0.31      0.37      1952
           4       0.67      0.76      0.71      8330
           5       0.70      0.58      0.63      2564
           6       0.60      0.64      0.62      2933
           7       0.73      0.79      0.76      6992
           8       0.54      0.38      0.44       811
           9       0.60      0.68      0.64      6393
          10       0.70      0.69      0.70      4259
          11       0.66      0.60      0.63      2237

    accuracy                           0.65     41903
   macro avg       0.61      0.56      0.58     41903
```

```
weighted avg       0.64      0.65      0.64     41903
```

RNN Model

In [13]:

```
# Split the data into training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Tokenization and vectorization using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()

# Padding sequences
max_sequence_length = X_train.shape[1]

# Reshape the input data to be 3-dimensional (samples, time steps, feature
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Define the RNN model
model = Sequential()
model.add(SimpleRNN(128, return_sequences=True, input_shape=(1, max_sequen
model.add(Dropout(0.2))
model.add(SimpleRNN(64))
model.add(Dropout(0.2))
model.add(Dense(len(label_encoder.classes_), activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', me

# Print the model summary
print(model.summary())

# Define early stopping
early_stopping = EarlyStopping(patience=3, restore_best_weights=True)

# Train the model with early stopping
model.fit(X_train, y_train, batch_size=32, epochs=5, validation_data=(X_te

# Generate predictions on the test set
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 1, 128)            656512

 dropout_4 (Dropout)         (None, 1, 128)            0

 simple_rnn_1 (SimpleRNN)    (None, 64)                12352

 dropout_5 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 12)                780

=================================================================
Total params: 669,644
Trainable params: 669,644
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
5238/5238 [==============================] - 38s 7ms/step - loss: 1.2832 - ac
curacy: 0.6029 - val_loss: 1.1439 - val_accuracy: 0.6391
Epoch 2/5
5238/5238 [==============================] - 37s 7ms/step - loss: 1.1114 - ac
curacy: 0.6511 - val_loss: 1.1326 - val_accuracy: 0.6427
Epoch 3/5
5238/5238 [==============================] - 34s 6ms/step - loss: 1.0751 - ac
curacy: 0.6598 - val_loss: 1.1310 - val_accuracy: 0.6424
Epoch 4/5
5238/5238 [==============================] - 32s 6ms/step - loss: 1.0518 - ac
curacy: 0.6659 - val_loss: 1.1298 - val_accuracy: 0.6436
Epoch 5/5
5238/5238 [==============================] - 35s 7ms/step - loss: 1.0364 - ac
curacy: 0.6701 - val_loss: 1.1331 - val_accuracy: 0.6436
1310/1310 [==============================] - 5s 4ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.43      0.47      1532
           1       0.54      0.49      0.51      3085
           2       0.48      0.40      0.44       815
           3       0.44      0.34      0.39      1952
           4       0.68      0.75      0.71      8330
           5       0.72      0.57      0.63      2564
           6       0.59      0.64      0.61      2933
           7       0.71      0.79      0.75      6992
           8       0.49      0.36      0.41       811
           9       0.63      0.65      0.64      6393
          10       0.69      0.69      0.69      4259
          11       0.65      0.59      0.62      2237

    accuracy                           0.64     41903
   macro avg       0.60      0.56      0.57     41903
weighted avg       0.64      0.64      0.64     41903
```

Bidirectioanl RNN model

In [14]:

```
# Split the data into training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Tokenization and vectorization using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()

# Padding sequences
max_sequence_length = X_train.shape[1]

# Reshape the input data to be 3-dimensional (samples, time steps, feature
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Define the Bidirectional RNN model
model = Sequential()
model.add(Bidirectional(SimpleRNN(128, return_sequences=True), input_shape
model.add(Dropout(0.2))
model.add(Bidirectional(SimpleRNN(64)))
model.add(Dropout(0.2))
model.add(Dense(len(label_encoder.classes_), activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', me

# Print the model summary
print(model.summary())

# Define early stopping
early_stopping = EarlyStopping(patience=3, restore_best_weights=True)

# Train the model with early stopping
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_t

# Generate predictions on the test set
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional_2 (Bidirectio  (None, 1, 256)           1313024
 nal)

 dropout_6 (Dropout)         (None, 1, 256)            0

 bidirectional_3 (Bidirectio  (None, 128)              41088
 nal)

 dropout_7 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 12)                1548

=================================================================
Total params: 1,355,660
Trainable params: 1,355,660
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
5238/5238 [==============================] - 84s 15ms/step - loss: 1.2662 - a
ccuracy: 0.6072 - val_loss: 1.1506 - val_accuracy: 0.6383
Epoch 2/10
5238/5238 [==============================] - 72s 14ms/step - loss: 1.1040 - a
ccuracy: 0.6504 - val_loss: 1.1332 - val_accuracy: 0.6429
Epoch 3/10
5238/5238 [==============================] - 60s 11ms/step - loss: 1.0640 - a
ccuracy: 0.6616 - val_loss: 1.1320 - val_accuracy: 0.6436
Epoch 4/10
5238/5238 [==============================] - 56s 11ms/step - loss: 1.0380 - a
ccuracy: 0.6675 - val_loss: 1.1370 - val_accuracy: 0.6416
Epoch 5/10
5238/5238 [==============================] - 70s 13ms/step - loss: 1.0168 - a
ccuracy: 0.6720 - val_loss: 1.1281 - val_accuracy: 0.6439
Epoch 6/10
5238/5238 [==============================] - 76s 14ms/step - loss: 0.9948 - a
ccuracy: 0.6789 - val_loss: 1.1231 - val_accuracy: 0.6455
Epoch 7/10
5238/5238 [==============================] - 56s 11ms/step - loss: 0.9744 - a
ccuracy: 0.6833 - val_loss: 1.1307 - val_accuracy: 0.6467
Epoch 8/10
5238/5238 [==============================] - 66s 13ms/step - loss: 0.9520 - a
ccuracy: 0.6897 - val_loss: 1.1364 - val_accuracy: 0.6435
Epoch 9/10
5238/5238 [==============================] - 69s 13ms/step - loss: 0.9302 - a
ccuracy: 0.6952 - val_loss: 1.1395 - val_accuracy: 0.6455
1310/1310 [==============================] - 11s 8ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.41      0.47      1532
           1       0.55      0.48      0.51      3085
           2       0.49      0.37      0.42       815
           3       0.46      0.30      0.36      1952
```

```
           4       0.66      0.77      0.71      8330
           5       0.66      0.62      0.64      2564
           6       0.58      0.65      0.61      2933
           7       0.73      0.78      0.75      6992
           8       0.54      0.36      0.43       811
           9       0.64      0.65      0.64      6393
          10       0.69      0.70      0.70      4259
          11       0.66      0.60      0.63      2237

    accuracy                           0.65     41903
   macro avg       0.60      0.56      0.57     41903
weighted avg       0.64      0.65      0.64     41903
```

Insights on the performance of the model and ways to improve it

1. Performance:

- The models are able to classify the news. LSTM and Bidirectional LSTM model accuracy is 65% and RNN and Bidirectional RNN accuracy is 64%
- Generated a classification report on test dataset.

2. Improvements:

- Try different network architectures: Besides the bidirectional LSTM, need to explore other network architectures such as CNN-LSTM or Transformer-based models. These architectures might capture different aspects of the text data and potentially improve the accuracy.
- Utilize word embeddings: Instead of relying solely on TF-IDF vectorization, incorporate pre-trained word embeddings (e.g., Word2Vec, GloVe) to represent the input text. Word embeddings can capture more semantic information and improve the model's understanding of the text.
- Handle class imbalance: If there is a class imbalance in the dataset, consider applying techniques such as oversampling, undersampling, or class weights to balance the classes during training. This can help the model learn from all classes equally and improve its accuracy.
- Capture more contextual information: Depending on the nature of the text data, you can explore techniques such as using n-grams, attention mechanisms, or contextual embeddings (e.g., BERT, GPT) to capture more fine-grained contextual information

By implementing these improvements and fine-tuning the model, you can expect better performance and more accurate classification of News.

In [ ]:

```
1
```