Problem Statement:

Context: The Gurugram-based FlipItNews aims to revolutionize the way Indians perceive finance, business, and capital market investment, by giving it a boost through artificial intelligence (AI) and machine learning (ML). They're on a mission to reinvent financial literacy for Indians, where financial awareness is driven by smart information discovery and engagement with peers. Through their smart content discovery and contextual engagement, the company is simplifying business, finance, and investment for millennials and first-time investors

Objective: The goal of this project is to use a bunch of news articles extracted from the companies' internal database and categorize them into several categories like politics, technology, sports, business and entertainment based on their content. Use natural language processing and create & compare at least three different models.

Attribute Information:

- Article
- Category

The feature names are themselves pretty self-explanatory.

Our Approach:

1. Importing the libraries
2. Loading the dataset

- Mounting the drive
- Reading the data file

3. Data Exploration

- Shape of the dataset
- News articles per category

4. Text Processing

- Removing the non-letters
- Tokenizing the text
- Removing stopwords
- Lemmatization

5. Data Transformation

- Encoding the target variable
- Bag of Words
- TF-IDF
- Train-Test Split

6. Model Training & Evaluation

- Simple Approach

    - Naive Bayes
- Functionalized Code
    - Decision Tree
    - Nearest Neighbors
    - Random Forest

Importing the libraries -

In [1]:
```
# To ignore all warnings
import warnings

# For reading & manipulating the data
import pandas as pd
import numpy as np

# For visualizing the data
!pip install matplotlib --upgrade
import matplotlib.pyplot as plt
import seaborn as sns

# To use Regular Expressions
import re

# To use Natural Language Processing
import nltk

# For tokenization
from nltk.tokenize import word_tokenize
nltk.download('punkt')

# To remove stopwords
from nltk.corpus import stopwords
nltk.download('stopwords')

# For Lemmetization
from nltk import WordNetLemmatizer
nltk.download('wordnet')

# For BoW & TF-IDF
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorize

# For encoding the categorical variable
!pip install category_encoders
import category_encoders as ce

# To try out different ML models
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier

# To perform train-test split
from sklearn.model_selection import train_test_split

# Performace Metrics for evaluating the model
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, precis
from sklearn.metrics import confusion_matrix, classification_report

warnings.simplefilter('ignore')
```

```
Requirement already satisfied: matplotlib in /Users/shivam13juna/Document
s/virtual_envs/appy/lib/python3.9/site-packages (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in /Users/shivam13juna/Doc
uments/virtual_envs/appy/lib/python3.9/site-packages (from matplotlib) (1.
2.1)
Requirement already satisfied: cycler>=0.10 in /Users/shivam13juna/Documen
ts/virtual_envs/appy/lib/python3.9/site-packages (from matplotlib) (0.12.
1)
Requirement already satisfied: fonttools>=4.22.0 in /Users/shivam13juna/Do
cuments/virtual_envs/appy/lib/python3.9/site-packages (from matplotlib)
(4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /Users/shivam13juna/Do
cuments/virtual_envs/appy/lib/python3.9/site-packages (from matplotlib)
(1.4.5)
Requirement already satisfied: numpy>=1.21 in /Users/shivam13juna/Document
s/virtual_envs/appy/lib/python3.9/site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /Users/shivam13juna/Docu
ments/virtual_envs/appy/lib/python3.9/site-packages (from matplotlib) (24.
0)
Requirement already satisfied: pillow>=8 in /Users/shivam13juna/Documents/
virtual_envs/appy/lib/python3.9/site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /Users/shivam13juna/Doc
uments/virtual_envs/appy/lib/python3.9/site-packages (from matplotlib) (3.
1.2)
Requirement already satisfied: python-dateutil>=2.7 in /Users/shivam13jun
a/Documents/virtual_envs/appy/lib/python3.9/site-packages (from matplotli
b) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in /Users/shivam
13juna/Documents/virtual_envs/appy/lib/python3.9/site-packages (from matpl
otlib) (6.4.0)
Requirement already satisfied: zipp>=3.1.0 in /Users/shivam13juna/Document
s/virtual_envs/appy/lib/python3.9/site-packages (from importlib-resources>
=3.2.0->matplotlib) (3.18.1)
Requirement already satisfied: six>=1.5 in /Users/shivam13juna/Documents/v
irtual_envs/appy/lib/python3.9/site-packages (from python-dateutil>=2.7->m
atplotlib) (1.16.0)

[nltk_data] Downloading package punkt to
[nltk_data]     /Users/shivam13juna/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/shivam13juna/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/shivam13juna/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl.metadata (8.0 k
B)
Requirement already satisfied: numpy>=1.14.0 in /Users/shivam13juna/Docume
nts/virtual_envs/appy/lib/python3.9/site-packages (from category_encoders)
(1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /Users/shivam13jun
a/Documents/virtual_envs/appy/lib/python3.9/site-packages (from category_e
ncoders) (1.4.2)
Requirement already satisfied: scipy>=1.0.0 in /Users/shivam13juna/Documen
ts/virtual_envs/appy/lib/python3.9/site-packages (from category_encoders)
(1.13.0)
Collecting statsmodels>=0.9.0 (from category_encoders)
  Downloading statsmodels-0.14.2-cp39-cp39-macosx_11_0_arm64.whl.metadata
(9.2 kB)
Requirement already satisfied: pandas>=1.0.5 in /Users/shivam13juna/Docume
nts/virtual_envs/appy/lib/python3.9/site-packages (from category_encoders)
(2.2.2)
Collecting patsy>=0.5.1 (from category_encoders)
  Downloading patsy-0.5.6-py2.py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/shivam13ju
na/Documents/virtual_envs/appy/lib/python3.9/site-packages (from pandas>=
1.0.5->category_encoders) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /Users/shivam13juna/Documen
ts/virtual_envs/appy/lib/python3.9/site-packages (from pandas>=1.0.5->cate
gory_encoders) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /Users/shivam13juna/Docum
ents/virtual_envs/appy/lib/python3.9/site-packages (from pandas>=1.0.5->ca
tegory_encoders) (2024.1)
Requirement already satisfied: six in /Users/shivam13juna/Documents/virtua
l_envs/appy/lib/python3.9/site-packages (from patsy>=0.5.1->category_encod
ers) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in /Users/shivam13juna/Docume
nts/virtual_envs/appy/lib/python3.9/site-packages (from scikit-learn>=0.2
0.0->category_encoders) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/shivam13jun
a/Documents/virtual_envs/appy/lib/python3.9/site-packages (from scikit-lea
rn>=0.20.0->category_encoders) (3.4.0)
Requirement already satisfied: packaging>=21.3 in /Users/shivam13juna/Docu
ments/virtual_envs/appy/lib/python3.9/site-packages (from statsmodels>=0.
9.0->category_encoders) (24.0)
Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 774.9 kB/s eta 0:
00:00a 0:00:01
Downloading patsy-0.5.6-py2.py3-none-any.whl (233 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 233.9/233.9 kB 2.4 MB/s eta 0:
00:00a 0:00:01
Downloading statsmodels-0.14.2-cp39-cp39-macosx_11_0_arm64.whl (10.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.1/10.1 MB 6.8 MB/s eta 0:0
0:00 0:00:01m
Installing collected packages: patsy, statsmodels, category_encoders
Successfully installed category_encoders-2.6.3 patsy-0.5.6 statsmodels-0.1
4.2
```

Loading the dataset -

Mounting the drive -

In [78]:
```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [79]:
```
link = "https://drive.google.com/file/d/1I3-pQFzbSufhpMrUKAROBLGULXcWiB9u/v

id = link.split("/")[-2]

downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('news-articles.csv')
```

Reading the data file -

In [3]:
```
df = pd.read_csv('flipitnews-data.csv')
df.sample(10)
```

Out[3]:

	Category	Article
1287	Business	ericsson sees earnings improve telecoms equipm...
1329	Sports	irish finish with home game republic of irelan...
1238	Politics	sainsbury s labour election gift science minis...
1702	Business	call to overhaul uk state pension the uk pensi...
766	Entertainment	applegate s charity show closes us musical swe...
1648	Sports	prutton poised for lengthy fa ban southampton ...
323	Politics	blunkett hints at election call ex-home secret...
541	Entertainment	mumbai bombs movie postponed the release of a ...
426	Sports	bortolami predicts dour contest italy skipper ...
1544	Business	jobs go at oracle after takeover oracle has an...

Data Exploration

First, let's check the shape of the dataset that we have.

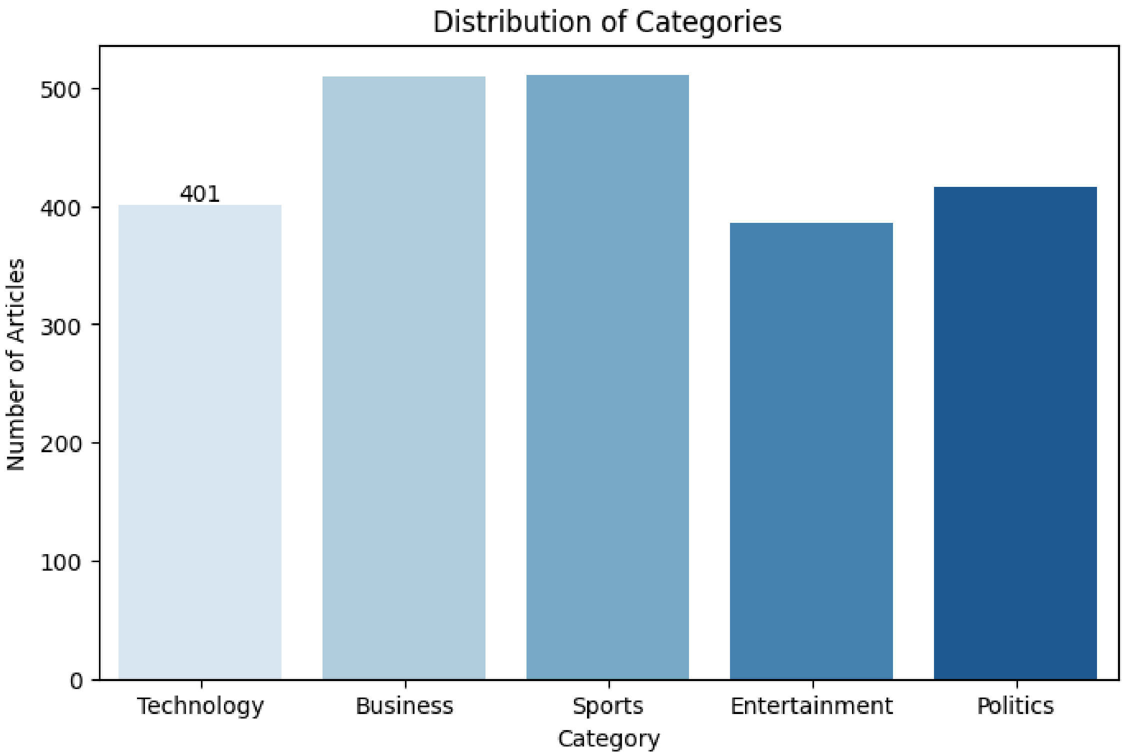
In [4]:
```
print("No. of rows: {}".format(df.shape[0]))
```

```
No. of rows: 2225
```

Observation: There are 2,225 different news articles present in the dataset.

No. of news articles per category -

In [5]:
```
plt.figure(figsize=(8, 5))
ax = sns.countplot(x='Category', data=df, palette='Blues')

ax.bar_label(ax.containers[0])

ax.set_title('Distribution of Categories')
ax.set_xlabel('Category')
ax.set_ylabel('Number of Articles')

plt.show()
```



Observation: Most of the news articles in the dataset are from Business & Sports category.

Text Processing

Before processing -

In [6]:
```
df['Article'][1]
```

Out[6]:
```
'worldcom boss  left books alone  former worldcom boss bernie ebbers  who
is accused of overseeing an $11bn (£5.8bn) fraud  never made accounting de
cisions  a witness has told jurors.  david myers made the comments under q
uestioning by defence lawyers who have been arguing that mr ebbers was not
responsible for worldcom s problems. the phone company collapsed in 2002 a
nd prosecutors claim that losses were hidden to protect the firm s shares.
mr myers has already pleaded guilty to fraud and is assisting prosecutors.
on monday  defence lawyer reid weingarten tried to distance his client fro
m the allegations. during cross examination  he asked mr myers if he ever
knew mr ebbers  make an accounting decision  .  not that i am aware of   m
r myers replied.  did you ever know mr ebbers to make an accounting entry
into worldcom books   mr weingarten pressed.  no   replied the witness. mr
myers has admitted that he ordered false accounting entries at the request
of former worldcom chief financial officer scott sullivan. defence lawyers
have been trying to paint mr sullivan  who has admitted fraud and will tes
tify later in the trial  as the mastermind behind worldcom s accounting ho
use of cards.  mr ebbers  team  meanwhile  are looking to portray him as a
n affable boss  who by his own admission is more pe graduate than economis
t. whatever his abilities  mr ebbers transformed worldcom from a relative
unknown into a $160bn telecoms giant and investor darling of the late 1990
s. worldcom s problems mounted  however  as competition increased and the
telecoms boom petered out. when the firm finally collapsed  shareholders l
ost about $180bn and 20 000 workers lost their jobs. mr ebbers  trial is e
xpected to last two months and if found guilty the former ceo faces a subs
tantial jail sentence. he has firmly declared his innocence.'
```

This is how a single news article in our dataset looks before processing.
We can see that everything is already in lower case so we don't need to do that explicitly.

In [7]:
```
stop_words = list(stopwords.words("english"))

def text_process(sent):
  # Removing non-letters
  sent = re.sub('[^a-zA-Z]', ' ', sent)

  # Word tokenizing the text
  words = nltk.word_tokenize(sent)

  # Removing stopwords
  filtered_sent = [w for w in words if not w in stop_words]

  # Lemmatization
  lemmatizer = WordNetLemmatizer()
  new_txt = [lemmatizer.lemmatize(word) for word in filtered_sent]
  new_txt = " ".join(new_txt)

  return new_txt

df['Article'] = df['Article'].apply(text_process)
```

After processing -

In [8]:
```
df['Article'][1]
```

Out[8]:
```
'worldcom bos left book alone former worldcom bos bernie ebbers accused ov
erseeing bn bn fraud never made accounting decision witness told juror dav
id myers made comment questioning defence lawyer arguing mr ebbers respons
ible worldcom problem phone company collapsed prosecutor claim loss hidden
protect firm share mr myers already pleaded guilty fraud assisting prosecu
tor monday defence lawyer reid weingarten tried distance client allegation
cross examination asked mr myers ever knew mr ebbers make accounting decis
ion aware mr myers replied ever know mr ebbers make accounting entry world
com book mr weingarten pressed replied witness mr myers admitted ordered f
alse accounting entry request former worldcom chief financial officer scot
t sullivan defence lawyer trying paint mr sullivan admitted fraud testify
later trial mastermind behind worldcom accounting house card mr ebbers tea
m meanwhile looking portray affable bos admission pe graduate economist wh
atever ability mr ebbers transformed worldcom relative unknown bn telecom
giant investor darling late worldcom problem mounted however competition i
ncreased telecom boom petered firm finally collapsed shareholder lost bn w
orker lost job mr ebbers trial expected last two month found guilty former
ceo face substantial jail sentence firmly declared innocence'
```

This is what an article obtained after text processing looks like.

Data Transformation

Encoding the target variable -

We will be using the OrdinalEncoder from category_encoders.

It encodes categorical features as ordinal, in one ordered feature. Ordinal encoding uses a single column of integers to represent the classes.

For more details you can refer to this link: [https://contrib.scikit-learn.org/category_encoders/ (https://contrib.scikit-learn.org/category_encoders/)](https://contrib.scikit-learn.org/category_encoders/)

In [9]:
```
encode = ce.OrdinalEncoder(cols=['Category'])
df = encode.fit_transform(df)
```

Outcome labels after encoding -

Category:

1 - Technology

2 - Business

3 - Sports

4 - Entertainment

5 - Politics

Bag of Words / TF-IDF

We've given the user a choice to select one of the following techniques for vectorizing the data -

- BoW
- TF-IDF

In [10]:
```
df
```

Out[10]:

	Category	Article
0	1	tv future hand viewer home theatre system plas...
1	2	worldcom bos left book alone former worldcom b...
2	3	tiger wary farrell gamble leicester say rushed...
3	3	yeading face newcastle fa cup premiership side...
4	4	ocean twelve raid box office ocean twelve crim...
...	...	...
2220	2	car pull u retail figure u retail sale fell ja...
2221	5	kilroy unveils immigration policy ex chatshow ...
2222	4	rem announce new glasgow concert u band rem an...
2223	5	political squabble snowball become commonplace...
2224	3	souness delight euro progress bos graeme soune...

2225 rows × 2 columns

In [12]:
```
If you want to use Bag of Words \n (2) If you want to use TF-IDF \n Choice:


s=5000) #BOW  #  max_features=5000 specifies that only the top 5000 most fre
).toarray()   #This transforms the text data into a document-term matrix, wh
es)


DF
tle).toarray()
es)
```

Performing train-test split -

In [13]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25,
                                                  shuffle=True, stratify=y,
                                                  random_state=42)
```

Final shape of the train & test set.

In [14]:
```
print("No. of rows in train set is {}.".format(X_train.shape[0]))
print("No. of rows in test set is {}.".format(X_val.shape[0]))
```

```
No. of rows in train set is 1668.
No. of rows in test set is 557.
```

Simple Approach -

First, we'll follow a basic approach to create a model for this multi-class classification problem.

####Naive Bayes Classifier

The very first ML algorithm that we'll be trying is Naive Bayes Classifier.

In [15]:
```
# Training the model -
nb = MultinomialNB()
nb.fit(X_train, y_train)
```

Out[15]:
```
MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [16]:
```
# Calculating the train & test accuracy -
nb_train = accuracy_score(y_train, nb.predict(X_train))
nb_test = accuracy_score(y_val, nb.predict(X_val))

print("Train accuracy :{:.3f}".format(nb_train))
print("Test accuracy :{:.3f}".format(nb_test))
```

```
Train accuracy :0.988
Test accuracy :0.977
```

In [17]:
```
# Making predictions on the test set -
y_pred_nb = nb.predict(X_val)
y_pred_proba_nb = nb.predict_proba(X_val)
```

In [18]:
```
# Computing the ROC AUC score -
print("ROC AUC Score: {:.3f}".format(roc_auc_score(y_val, y_pred_proba_nb,
```

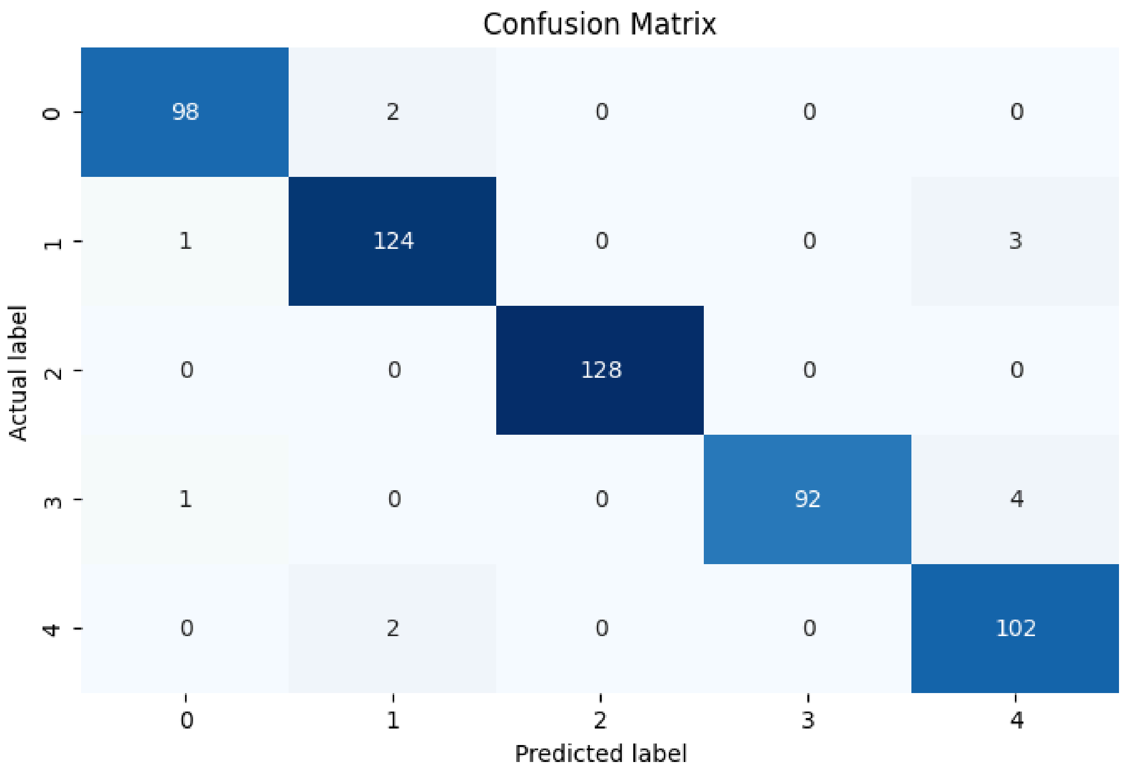
```
ROC AUC Score: 0.999
```

In [19]:
```
# Computing the precision, recall & f1 score -
precision = precision_score(y_val, y_pred_nb, average='weighted')
recall = recall_score(y_val, y_pred_nb, average='weighted')
f1 = f1_score(y_val, y_pred_nb, average='weighted')

print("Precision: {:.3f}".format(precision))
print("Recall: {:.3f}".format(recall))
print("F1 Score: {:.3f}".format(f1))
```

```
Precision: 0.977
Recall: 0.977
F1 Score: 0.977
```

Plotting the Confusion Matrix -

In [20]:
```
cm = confusion_matrix(y_val, y_pred_nb)

plt.figure(figsize = (8, 5))
sns.heatmap(cm, annot=True, fmt='d', cbar=False, cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('Actual label')
plt.show()
```



Printing the Classification Report -

In [21]:
```
print(classification_report(y_val, y_pred_nb))
```

```
              precision    recall  f1-score   support

           1       0.98      0.98      0.98       100
           2       0.97      0.97      0.97       128
           3       1.00      1.00      1.00       128
           4       1.00      0.95      0.97        97
           5       0.94      0.98      0.96       104

    accuracy                           0.98       557
   macro avg       0.98      0.98      0.98       557
weighted avg       0.98      0.98      0.98       557
```

Functionalized Code -

Now, we'll try to functionalize the above code so that we can use it for a few more different models.

Model Training

In [22]:
```
def model_train(obj):
  obj.fit(X_train, y_train) # Training the model
  y_pred = obj.predict(X_val) # Making predictions
  y_pred_proba = obj.predict_proba(X_val)
  return y_pred, y_pred_proba
```

Model Evaluation

In [23]:
```
def model_eval(obj, y_pred, y_pred_proba):
  print("-------------------------")

  # Calculating the train & test accuracy
  train_acc = accuracy_score(y_train, obj.predict(X_train))
  test_acc = accuracy_score(y_val, obj.predict(X_val))

  print("Train Accuracy: {:.3f}".format(train_acc))
  print("Test Accuracy: {:.3f}\n".format(test_acc))

  # Computing the ROC AUC score
  print("ROC AUC Score: {:.3f}\n".format(roc_auc_score(y_val, y_pred_proba,

  # Computing the precision, recall & f1 score
  precision = precision_score(y_val, y_pred, average='weighted')
  recall = recall_score(y_val, y_pred, average='weighted')
  f1 = f1_score(y_val, y_pred, average='weighted')

  print("Precision: {:.3f}".format(precision))
  print("Recall: {:.3f}".format(recall))
  print("F1 Score: {:.3f}".format(f1))

  print("-------------------------")
```

Now, let us try out a few more different ML algorithm to see how they perform for this problem, on this dataset.

Decision Tree Classifer

In [24]:
```
# Creating the model object -
dt = DecisionTreeClassifier()

# Training the model -
y_pred_dt, y_pred_proba_dt = model_train(dt)

# Evaluatong the model -
model_eval(dt, y_pred_dt, y_pred_proba_dt)
```

```
-------------------------
Train Accuracy: 1.000
Test Accuracy: 0.862

ROC AUC Score: 0.912

Precision: 0.863
Recall: 0.862
F1 Score: 0.862
-------------------------
```

Nearest Neighbors Classifier

In [25]:
```
# Creating the model object -
knn = KNeighborsClassifier(n_neighbors=5)

# Training the model -
y_pred_knn, y_pred_proba_knn = model_train(knn)

# Evaluatong the model -
model_eval(knn, y_pred_knn, y_pred_proba_knn)
```

```
-------------------------
Train Accuracy: 0.965
Test Accuracy: 0.934

ROC AUC Score: 0.988

Precision: 0.935
Recall: 0.934
F1 Score: 0.933
-------------------------
```

Random Forest Classifier

In [26]:
```
# Creating the model object -
rf = RandomForestClassifier()

# Training the model -
y_pred_rf, y_pred_proba_rf = model_train(rf)

# Evaluatong the model -
model_eval(rf, y_pred_rf, y_pred_proba_rf)
```

```
-------------------------
Train Accuracy: 1.000
Test Accuracy: 0.955

ROC AUC Score: 0.998

Precision: 0.956
Recall: 0.955
F1 Score: 0.955
-------------------------
```

Observation: Out of all the models tested till now, Naive Bayes Classifier seems to be the best performing one since it gives good train & test accuracy, more than satisfactory precision & recall and almost non-significant overfitting.

Let's train LSTM Model

In [21]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding, GRU, SimpleRNN
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder



df = pd.read_csv('flipitnews-data.csv')
df.head()
```

Out[21]:

	Category	Article
0	Technology	tv future in the hands of viewers with home th...
1	Business	worldcom boss left books alone former worldc...
2	Sports	tigers wary of farrell gamble leicester say ...
3	Sports	yeading face newcastle in fa cup premiership s...
4	Entertainment	ocean s twelve raids box office ocean s twelve...

In [25]:

```
# Parameters
max_features = 5000  #  max_features=5000 specifies that only the top 5000
maxlen = 100  # Cuts off the text after this number of words # This variabl
embedding_size = 100  # Dimensionality of the GloVe embeddings
batch_size = 1000
epochs = 100

# Preprocessing
def preprocess_text(df, text_column):
    df[text_column] = df[text_column].apply(lambda x: x.lower())  # Lowerca
    return df

df = preprocess_text(df, 'Article')

# Tokenization
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(df['Article'])    # This updates the internal vocabu
sequences = tokenizer.texts_to_sequences(df['Article'])  #: This converts e
data = pad_sequences(sequences, maxlen=maxlen)   # This pads each sequence

# Label encoding
le = LabelEncoder()
labels = le.fit_transform(df['Category'])
labels = tf.keras.utils.to_categorical(labels)

# Split data
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size


# Load GloVe embeddings
def load_glove_embeddings(embedding_path, embedding_dim, tokenizer, max_fea
    embeddings_index = {}
    with open(embedding_path, 'r', encoding='utf8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs

    # Only use the top max_features items from word_index
    limited_word_index = {word: index for word, index in tokenizer.word_ind

    # Note: We use min(max_features + 1, len(limited_word_index) + 1) to ha
    embedding_matrix = np.zeros((min(max_features + 1, len(limited_word_ind
    for word, i in limited_word_index.items():
        if i > max_features:  # Safety check to ensure we do not exceed max_
            continue
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    return embedding_matrix

embedding_matrix = load_glove_embeddings('glove.6B.100d.txt', embedding_siz
#print(embedding_matrix.shape) #(5000,100)

# Model building
model = Sequential([
    Embedding(max_features, embedding_size, weights=[embedding_matrix], inp
    LSTM(100, dropout=0.2, recurrent_dropout=0.2),
    Dense(len(np.unique(df['Category'])), activation='softmax')
```

```
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['

# Use EarlyStopping

from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Model training
model.fit(X_train, y_train,
batch_size=batch_size, epochs=epochs, validation_data=(X_test, y_test), ver
callbacks=[early_stopping])

# Evaluate

from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
y_test_argmax = np.argmax(y_test, axis=1)

print(classification_report(y_test_argmax, y_pred))
```

```
Epoch 1/100

/Users/shivam13juna/Documents/virtual_envs/appy/lib/python3.9/site-package
s/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_leng
th` is deprecated. Just remove it.
  warnings.warn(
```

```
2/2 - 11s - 6s/step - accuracy: 0.2461 - loss: 1.5951 - val_accuracy: 0.35
06 - val_loss: 1.5337
Epoch 2/100
2/2 - 9s - 5s/step - accuracy: 0.3567 - loss: 1.5288 - val_accuracy: 0.438
2 - val_loss: 1.4676
Epoch 3/100
2/2 - 9s - 5s/step - accuracy: 0.4270 - loss: 1.4652 - val_accuracy: 0.498
9 - val_loss: 1.3978
Epoch 4/100
2/2 - 10s - 5s/step - accuracy: 0.4725 - loss: 1.4048 - val_accuracy: 0.50
34 - val_loss: 1.3167
Epoch 5/100
2/2 - 10s - 5s/step - accuracy: 0.5107 - loss: 1.3261 - val_accuracy: 0.51
46 - val_loss: 1.2182
Epoch 6/100
2/2 - 10s - 5s/step - accuracy: 0.5337 - loss: 1.2371 - val_accuracy: 0.55
51 - val_loss: 1.1133
Epoch 7/100
2/2 - 10s - 5s/step - accuracy: 0.5326 - loss: 1.1527 - val_accuracy: 0.53
48 - val_loss: 1.1023
Epoch 8/100
2/2 - 10s - 5s/step - accuracy: 0.5449 - loss: 1.1180 - val_accuracy: 0.57
75 - val_loss: 0.9930
Epoch 9/100
2/2 - 9s - 5s/step - accuracy: 0.5719 - loss: 1.0715 - val_accuracy: 0.683
1 - val_loss: 0.8749
Epoch 10/100
2/2 - 9s - 5s/step - accuracy: 0.6354 - loss: 0.9994 - val_accuracy: 0.714
6 - val_loss: 0.8681
Epoch 11/100
2/2 - 9s - 5s/step - accuracy: 0.6489 - loss: 0.9748 - val_accuracy: 0.759
6 - val_loss: 0.7541
Epoch 12/100
2/2 - 9s - 5s/step - accuracy: 0.6719 - loss: 0.9303 - val_accuracy: 0.752
8 - val_loss: 0.7245
Epoch 13/100
2/2 - 9s - 4s/step - accuracy: 0.6882 - loss: 0.8686 - val_accuracy: 0.716
9 - val_loss: 0.7416
Epoch 14/100
2/2 - 9s - 5s/step - accuracy: 0.6944 - loss: 0.8272 - val_accuracy: 0.748
3 - val_loss: 0.6607
Epoch 15/100
2/2 - 10s - 5s/step - accuracy: 0.7090 - loss: 0.8076 - val_accuracy: 0.79
33 - val_loss: 0.5939
Epoch 16/100
2/2 - 9s - 5s/step - accuracy: 0.7236 - loss: 0.7810 - val_accuracy: 0.802
2 - val_loss: 0.5861
Epoch 17/100
2/2 - 9s - 5s/step - accuracy: 0.7472 - loss: 0.7347 - val_accuracy: 0.811
2 - val_loss: 0.5828
Epoch 18/100
2/2 - 10s - 5s/step - accuracy: 0.7298 - loss: 0.7568 - val_accuracy: 0.79
55 - val_loss: 0.5886
Epoch 19/100
2/2 - 10s - 5s/step - accuracy: 0.7567 - loss: 0.6854 - val_accuracy: 0.82
02 - val_loss: 0.5465
Epoch 20/100
2/2 - 12s - 6s/step - accuracy: 0.7612 - loss: 0.6789 - val_accuracy: 0.82
47 - val_loss: 0.4866
Epoch 21/100
2/2 - 15s - 7s/step - accuracy: 0.7719 - loss: 0.6477 - val_accuracy: 0.84
```

```
27 - val_loss: 0.4781
Epoch 22/100
2/2 - 22s - 11s/step - accuracy: 0.7764 - loss: 0.6341 - val_accuracy: 0.8
270 - val_loss: 0.4845
Epoch 23/100
2/2 - 24s - 12s/step - accuracy: 0.7865 - loss: 0.6221 - val_accuracy: 0.8
449 - val_loss: 0.4885
Epoch 24/100
2/2 - 25s - 12s/step - accuracy: 0.8000 - loss: 0.5780 - val_accuracy: 0.8
562 - val_loss: 0.4293
Epoch 25/100
2/2 - 18s - 9s/step - accuracy: 0.8096 - loss: 0.5633 - val_accuracy: 0.85
39 - val_loss: 0.4424
Epoch 26/100
2/2 - 16s - 8s/step - accuracy: 0.8051 - loss: 0.5730 - val_accuracy: 0.86
07 - val_loss: 0.4261
Epoch 27/100
2/2 - 15s - 7s/step - accuracy: 0.8180 - loss: 0.5460 - val_accuracy: 0.86
97 - val_loss: 0.4121
Epoch 28/100
2/2 - 13s - 6s/step - accuracy: 0.8112 - loss: 0.5470 - val_accuracy: 0.87
87 - val_loss: 0.3927
Epoch 29/100
2/2 - 12s - 6s/step - accuracy: 0.8258 - loss: 0.5307 - val_accuracy: 0.88
09 - val_loss: 0.3804
Epoch 30/100
2/2 - 12s - 6s/step - accuracy: 0.8416 - loss: 0.4935 - val_accuracy: 0.86
74 - val_loss: 0.4049
Epoch 31/100
2/2 - 12s - 6s/step - accuracy: 0.8315 - loss: 0.5154 - val_accuracy: 0.88
31 - val_loss: 0.3799
Epoch 32/100
2/2 - 12s - 6s/step - accuracy: 0.8455 - loss: 0.4877 - val_accuracy: 0.87
87 - val_loss: 0.3788
Epoch 33/100
2/2 - 11s - 6s/step - accuracy: 0.8461 - loss: 0.4825 - val_accuracy: 0.89
44 - val_loss: 0.3462
Epoch 34/100
2/2 - 11s - 6s/step - accuracy: 0.8478 - loss: 0.4708 - val_accuracy: 0.88
54 - val_loss: 0.3655
Epoch 35/100
2/2 - 12s - 6s/step - accuracy: 0.8680 - loss: 0.4255 - val_accuracy: 0.87
64 - val_loss: 0.3889
Epoch 36/100
2/2 - 12s - 6s/step - accuracy: 0.8691 - loss: 0.4270 - val_accuracy: 0.88
31 - val_loss: 0.3674
Epoch 37/100
2/2 - 11s - 6s/step - accuracy: 0.8652 - loss: 0.4087 - val_accuracy: 0.88
31 - val_loss: 0.3632
Epoch 38/100
2/2 - 12s - 6s/step - accuracy: 0.8612 - loss: 0.4432 - val_accuracy: 0.88
31 - val_loss: 0.3600
14/14 ━━━━━━━━━━━━━━━━━━━━ 4s 246ms/step
              precision    recall  f1-score   support

           0       0.88      0.84      0.86       101
           1       0.89      0.86      0.88        81
           2       0.78      0.93      0.85        83
           3       0.95      0.94      0.94        98
           4       0.95      0.84      0.89        82
```

```
    accuracy                           0.88       445
   macro avg       0.89      0.88      0.88       445
weighted avg       0.89      0.88      0.88       445
```

In [20]:
```
# Model building
model = Sequential([
    Embedding(max_features, embedding_size, weights=[embedding_matrix], inp
    GRU(100, dropout=0.2, recurrent_dropout=0.2),
    Dense(len(np.unique(df['Category'])), activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['

# Use EarlyStopping

from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Model training
model.fit(X_train, y_train,
batch_size=batch_size, epochs=epochs, validation_data=(X_test, y_test), ver
callbacks=[early_stopping])

# Evaluate

# Model building
model = Sequential([
    Embedding(max_features, embedding_size, weights=[embedding_matrix], inp
    GRU(100, dropout=0.2, recurrent_dropout=0.2),
    Dense(len(np.unique(df['Category'])), activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['

# Use EarlyStopping

from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Model training
model.fit(X_train, y_train,
batch_size=batch_size, epochs=epochs, validation_data=(X_test, y_test), ver
callbacks=[early_stopping])

# Evaluate

from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
y_test_argmax = np.argmax(y_test, axis=1)

print(classification_report(y_test_argmax, y_pred))
```

```
4/4 - 19s - 5s/step - accuracy: 0.3388 - loss: 1.5428 - val_accuracy:
0.3753 - val_loss: 1.4874
Epoch 84/100
4/4 - 18s - 5s/step - accuracy: 0.3466 - loss: 1.4898 - val_accuracy:
0.3798 - val_loss: 1.4862
Epoch 85/100
4/4 - 19s - 5s/step - accuracy: 0.3596 - loss: 1.4881 - val_accuracy:
0.3820 - val_loss: 1.4852
Epoch 86/100
4/4 - 18s - 5s/step - accuracy: 0.3466 - loss: 1.4859 - val_accuracy:
0.3820 - val_loss: 1.4841
```

In [ ]:

```
# Model building
model = Sequential([
    Embedding(max_features, embedding_size, weights=[embedding_matrix], inp
    SimpleRNN(100, dropout=0.2, recurrent_dropout=0.2),
    Dense(len(np.unique(df['Category'])), activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['

# Use EarlyStopping

from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Model training
model.fit(X_train, y_train,
batch_size=batch_size, epochs=epochs, validation_data=(X_test, y_test), ver
callbacks=[early_stopping])

# Evaluate

from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
y_test_argmax = np.argmax(y_test, axis=1)

print(classification_report(y_test_argmax, y_pred))
```

Inference

In [35]:
```
def predict_category(text, tokenizer, model, label_encoder, max_len):
    # Preprocess the text: lowercasing
    text = text.lower()

    # Tokenize text
    seq = tokenizer.texts_to_sequences([text])
    # Pad sequence to be of the same length as training data
    padded_seq = pad_sequences(seq, maxlen=max_len)

    # Predict using the model
    pred = model.predict(padded_seq)
    print("This was pred: ", pred)
    # Convert prediction to category label
    pred_label_index = np.argmax(pred, axis=1)
    pred_label = label_encoder.inverse_transform(pred_label_index)

    return pred_label[0]

# Example usage
input_text = "I need to create a better algorithm for prediting the stock m
predicted_category = predict_category(input_text, tokenizer, model, le, max
print("Predicted Category:", predicted_category)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
This was pred:  [[0.55646396 0.02896875 0.07845788 0.01739557 0.3187139 ]]
Predicted Category: Business
```