1. Define the problem statement

Problem Statement:

FlipItNews, a Gurugram-based company, aims to revolutionize the way Indians perceive finance, business, and capital market investments by leveraging artificial intelligence (AI) and machine learning (ML). The company wants to simplify business, finance, and investment information for millennials and first-time investors through smart content discovery and contextual engagement.

The objective of this project is to develop an automated system that can categorize news articles from FlipItNews' internal database into different categories such as politics, technology, sports, business, and entertainment based on their content. This categorization will help FlipItNews provide relevant and personalized content to their users based on their interests and preferences.

The problem at hand is a multi-class text classification task, where we need to build and compare at least three different natural language processing (NLP) models that can accurately classify news articles into the appropriate categories based on their textual content.

The provided dataset, "flipitnews_data.csv," contains news articles and their corresponding category labels. The task involves preprocessing the text data, extracting relevant features, training and evaluating different multi-class classification models, and selecting the best-performing model(s) for the categorization task.

The successful implementation of this project will enable FlipItNews to effectively organize and categorize their news content, thereby enhancing their content discovery and personalization capabilities, ultimately improving user engagement and promoting financial literacy among their target audience.

2. Data Preprocessing

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import re
```

In [2]:
```
# Load the dataset
data = pd.read_csv('flipitnews_data.csv')
```

In [3]:
```
# view all the variable names
data.columns
```

Out[3]:
```
Index(['Category', 'Article'], dtype='object')
```

In [4]:
```
# Display basic information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2225 entries, 0 to 2224
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  2225 non-null   object
 1   Article   2225 non-null   object
dtypes: object(2)
memory usage: 34.9+ KB
```

In [5]:
```
# Check the shape of the dataset
print("Shape of the dataset:-")
print("No. of rows: ", data.shape[0])
print("No. of columns: ", data.shape[1])
```

```
Shape of the dataset:-
No. of rows:  2225
No. of columns:  2
```

In [6]:
```
# Display the first few rows of the dataset
data.head()
```

Out[6]:

	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...

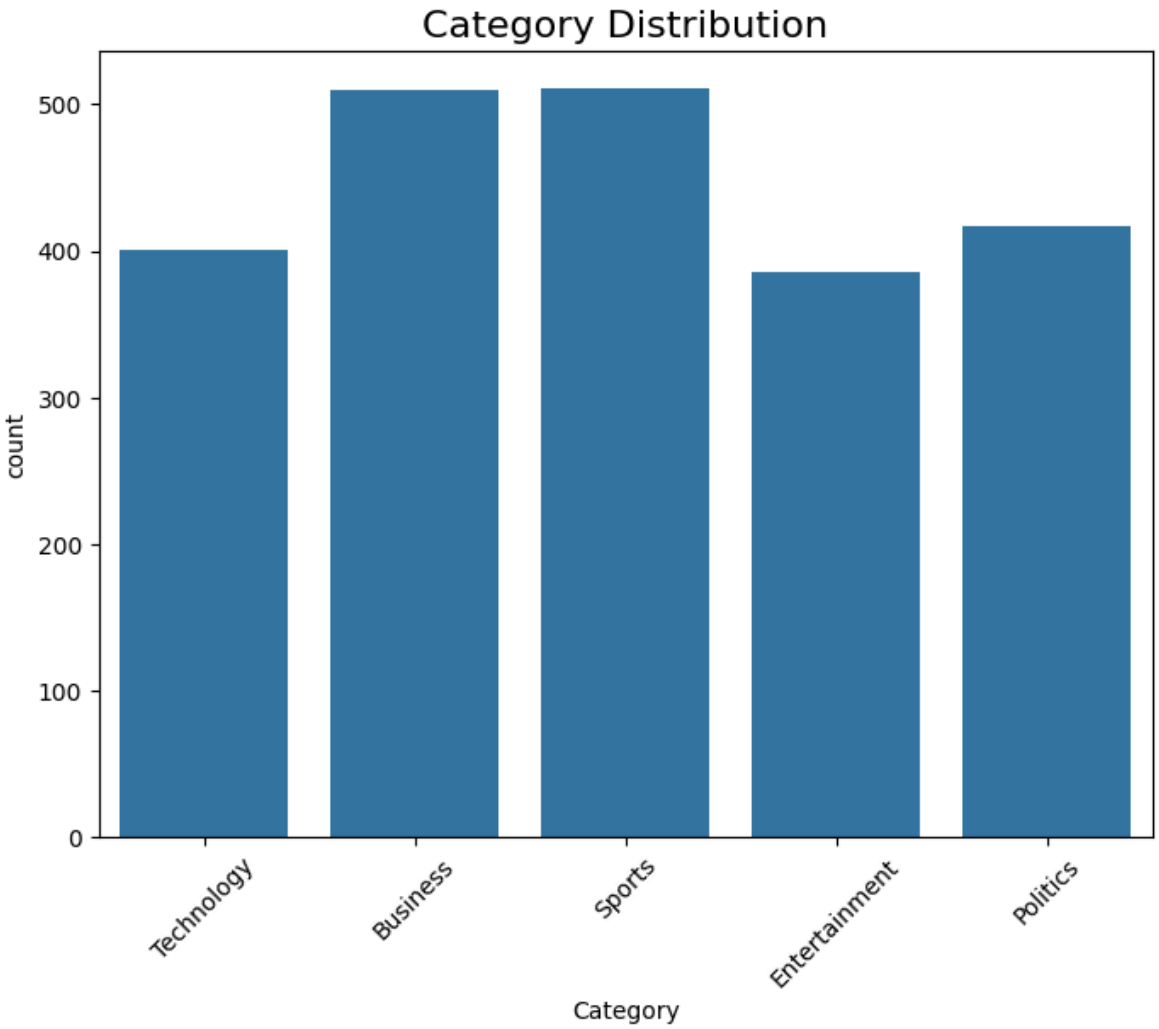
In [7]:
```
# Check for missing values
data.isnull().sum()
```

Out[7]:
```
Category    0
Article     0
dtype: int64
```

In [8]:
```
# Check the distribution of categories
data['Category'].value_counts()
```

Out[8]:
```
Category
Sports           511
Business         510
Politics         417
Technology       401
Entertainment    386
Name: count, dtype: int64
```

In [9]:
```
# Visualize the category distribution
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='Category')
plt.title('Category Distribution', fontsize=16)
plt.xticks(rotation=45)
plt.show()
```



3. Data Exploration

In [10]:
```
# Display the unique categories
data['Category'].unique()
```

Out[10]:
```
array(['Technology', 'Business', 'Sports', 'Entertainment', 'Politics'],
      dtype=object)
```

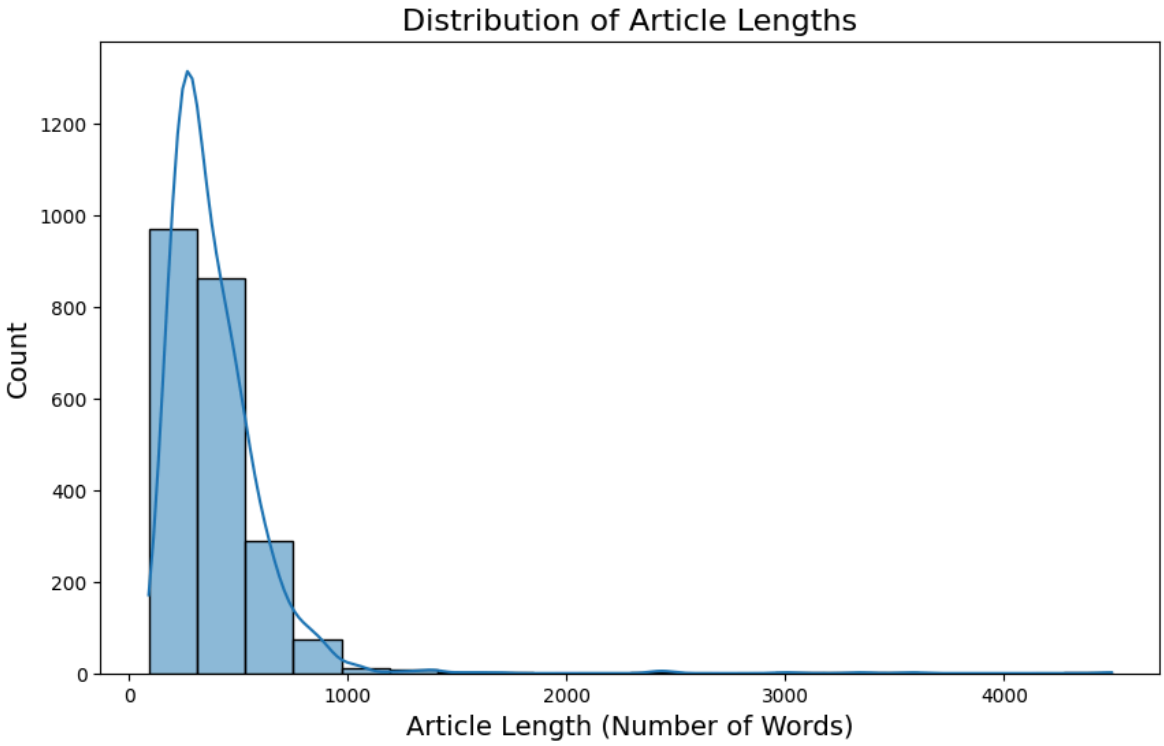
In [11]:
```
# Display the number of unique categories
len(data['Category'].unique())
```

Out[11]:
```
5
```

In [12]:
```
# Display the length of articles (number of words)
data['Article_Length'] = data['Article'].apply(lambda x: len(x.split()))
data['Article_Length'].describe()
```

Out[12]:
```
count    2225.000000
mean      390.295281
std       241.753128
min        90.000000
25%       250.000000
50%       337.000000
75%       479.000000
max      4492.000000
Name: Article_Length, dtype: float64
```

In [13]:
```
# Visualize the distribution of article lengths
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='Article_Length', bins=20, kde=True)
plt.title('Distribution of Article Lengths', fontsize=16)
plt.xlabel('Article Length (Number of Words)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.show()
```



4. Textual Data Processing Function Creation

In [14]:
```
# Import necessary libraries
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

In [15]:
```
# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

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

Out[15]: True

In [16]:
```
# Function to preprocess text
def preprocess_text(text):
    # Convert text to lowercase
```

```
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Tokenization
    tokens = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word not in stop_words]
    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
    # Join tokens back into text
    preprocessed_text = ' '.join(lemmatized_tokens)
    return preprocessed_text

# Apply preprocessing to the 'Article' column
data['Processed_Article'] = data['Article'].apply(preprocess_text)
```

In [17]:
```
# Display the preprocessed data
data.head()
```

Out[17]:

	Category	Article	Article_Length	Processed_Article
0	Technology	tv future in the hands of viewers with home th...	737	tv future hand viewer home theatre system plas...
1	Business	worldcom boss left books alone former worldc...	300	worldcom bos left book alone former worldcom b...
2	Sports	tigers wary of farrell gamble leicester say ...	246	tiger wary farrell gamble leicester say rushed...
3	Sports	yeading face newcastle in fa cup premiership s...	341	yeading face newcastle fa cup premiership side...
4	Entertainment	ocean s twelve raids box office ocean s twelve...	260	ocean twelve raid box office ocean twelve crim...

In this code:

1. We import necessary libraries such as NLTK (Natural Language Toolkit) for natural language processing.
2. We download essential NLTK resources required for text processing, including tokenization and stopwords.
3. We define a function called `preprocess_text()` to handle text preprocessing tasks, including lowercase conversion, punctuation removal, tokenization, stopwords removal, and lemmatization. The `preprocess_text` function takes a text input and executes the following steps:
   - Convert text to lowercase.
   - Remove punctuation using `str.maketrans` and `translate`.
   - Remove numbers using regular expressions with `re.sub`.
   - Tokenize the text into words using `word_tokenize`.
   - Remove stop words (common words like 'the', 'and', 'is', etc.) by creating a set of English stop words and filtering out the tokens present in the set.

    - Perform lemmatization (converting words to their base form) using the `WordNetLemmatizer`.
    - Join the lemmatized tokens back into a preprocessed text string using `join`.
4. We applied the preprocess_text function to the `Article` column of the dataset using the apply method and created a new column `Processed_Article` with the preprocessed text.
5. Finally, we displayed the first few rows of the preprocessed data using `data.head()`. We print the first few rows of the dataset to verify that the preprocessing steps have been applied correctly.

- Text preprocessing is an essential step in natural language processing tasks. It helps to remove noise and irrelevant information from the text, normalize the data, and prepare it for further processing and feature extraction.

- By performing operations like lowercasing, punctuation removal, number removal, tokenization, stop word removal, and lemmatization, we have cleaned and standardized the news article text, which will be beneficial for the subsequent steps of feature extraction and model building.

5. Before and After Text Processing: A Single News Article

In [18]:
```
# Select a random article
random_index = data.sample(1).index[0]
original_article = data.loc[random_index, 'Article']
processed_article = data.loc[random_index, 'Processed_Article']
```

In [19]:
```
print("Original Article:")
print(original_article)
print("\nProcessed Article:")
print(processed_article)
```

Original Article:
economy focus for election battle britain s economic future will be at the heart of labour s poll campaign chancellor gordon brown has said. he was speaking after ca binet members held their last meeting at no 10 before the expected election announc ement. he said voters would recognise that labour had brought stability and growth and would continue to do so. meanwhile the tories outlined their plans to tackle y ob culture and the lib dems gave more details about their proposals to replace cou ncil tax. earlier the archbishop of canterbury wrote to all three parties urging th em not to fight the election by exploiting people s fears. in an open letter he c alled on them not to turn the election into a competition about who can most effect ively frighten voters about terrorism asylum and crime. he said they should conce ntrate instead on issues such as the environment international development and the arms trade family policy and the reform of the criminal justice system. shadow f oreign secretary michael ancram said: we have fought a very positive campaign. i t hink he will want to look quite carefully at what jack straw said about michael how ard. in a speech to the foreign policy centre mr straw said of the tory leader: he is clever fluent and tactical but he is not wise. he lacks strategy and good judgment and his quick temper and impetuosity too often get the better of him. th e foreign secretary told the bbc: i was making the observation that because of mic hael howard s impetuosity you can get lurches of policy. liberal democrat chairman matthew taylor said: people are already really turned off by the kind of campaign the others are fighting and you will see us putting emphasis on some of these huge issues facing the world particularly the environment. labour s focus on the econ omy as their key message - came on the day a new report was published by the instit ute of fiscal studies suggesting that household incomes have fallen for the first time in more than a decade. the ifs says the drop partly reflects measures announce d in what it called the chancellor s tax-raising budget of 2002. the treasury dismi ssed the research as complete rubbish . party election supremo alan milburn said t he apparent drop in average incomes was because self-employed people had been affec ted by a world downturn which hit their profits. since 1997 the reported average take-home income had risen by 20% in real terms if you took out the self-employed mr milburn told bbc radio 4 s today programme. mr brown also dismissed the figures insisting that the typical family has been much better off under labour.

Processed Article:
economy focus election battle britain economic future heart labour poll campaign ch ancellor gordon brown said speaking cabinet member held last meeting expected elect ion announcement said voter would recognise labour brought stability growth would c ontinue meanwhile tory outlined plan tackle yob culture lib dems gave detail propos al replace council tax earlier archbishop canterbury wrote three party urging fight election exploiting people fear open letter called turn election competition effect ively frighten voter terrorism asylum crime said concentrate instead issue environm ent international development arm trade family policy reform criminal justice syste m shadow foreign secretary michael ancram said fought positive campaign think want look quite carefully jack straw said michael howard speech foreign policy centre mr straw said tory leader clever fluent tactical wise lack strategy good judgment quic k temper impetuosity often get better foreign secretary told bbc making observation michael howard impetuosity get lurch policy liberal democrat chairman matthew taylo r said people already really turned kind campaign others fighting see u putting emp hasis huge issue facing world particularly environment labour focus economy key mes sage came day new report published institute fiscal study suggesting household inco me fallen first time decade ifs say drop partly reflects measure announced called c hancellor taxraising budget treasury dismissed research complete rubbish party elec tion supremo alan milburn said apparent drop average income selfemployed people aff ected world downturn hit profit since reported average takehome income risen real t erm took selfemployed mr milburn told bbc radio today programme mr brown also dismi ssed figure insisting typical family much better labour

In this code:

1. We use the `sample` method of the pandas dataframe to randomly select a single row from the dataset. The `index[0]` selects the index of the sampled row.
2. We store the original article text in the `original_article` variable by accessing the 'Article' column using the sampled index.
3. We store the preprocessed article text in the `processed_article` variable by accessing the 'Processed_Article' column using the sampled index.
4. We print the "Original Article:" followed by the `original_article` text.
5. We print the "Processed Article:" followed by the `processed_article` text.

6. Target Variable Encoding

In [20]:
```
# Import library
from sklearn.preprocessing import LabelEncoder

# Create an instance of LabelEncoder
label_encoder = LabelEncoder()

# Encode the 'Category' column
data['Category_Encoded'] = label_encoder.fit_transform(data['Category'])
```

In [21]:
```
# Display the mapping of original categories to encoded labels
category_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_
print(category_mapping)
```

```
{'Business': 0, 'Entertainment': 1, 'Politics': 2, 'Sports': 3, 'Technology': 4}
```

In [22]:
```
# Display the first few rows of the encoded data
data[['Category', 'Category_Encoded']].head()
```

Out[22]:

	Category	Category_Encoded
0	Technology	4
1	Business	0
2	Sports	3
3	Sports	3
4	Entertainment	1

In [23]:
```
# Display the first few rows of the dataframe with encoded category
data.head()
```

Out[23]:

	Category	Article	Article_Length	Processed_Article	Category_Encoded
0	Technology	tv future in the hands of viewers with home th...	737	tv future hand viewer home theatre system plas...	4
1	Business	worldcom boss left books alone former worldc...	300	worldcom bos left book alone former worldcom b...	0
2	Sports	tigers wary of farrell gamble leicester say ...	246	tiger wary farrell gamble leicester say rushed...	3
3	Sports	yeading face newcastle in fa cup premiership s...	341	yeading face newcastle fa cup premiership side...	3
4	Entertainment	ocean s twelve raids box office ocean s twelve...	260	ocean twelve raid box office ocean twelve crim...	1

In this code:

1. We import the `LabelEncoder` class from `sklearn.preprocessing`.
2. We create an instance of `LabelEncoder` called `label_encoder`.
3. We apply the `fit_transform` method of `label_encoder` on the 'Category' column of the dataset. This method first fits the encoder to the unique categories and then transforms them into numerical labels.
4. The encoded categories are stored in a new column 'Category_Encoded' in the dataframe.
5. We Displays the mapping of original categories to encoded labels to their corresponding category names using `dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))`.
6. We display the first few rows of the dataframe, showing the original 'Category' column and the encoded 'Category_Encoded' column using `data[['Category', 'Category_Encoded']].head()`.
7. We Displays the first few rows of the dataframe with the encoded category column added as `Category_Encoded`.

By encoding the target variable, we have transformed the categorical data into a numerical format suitable for machine learning algorithms. This step is crucial for building classification models that can accurately predict the category of a given news article.

7. Vectorization Technique Selection

In [24]:
```python
# Import library
import sys

# Function to vectorize text data
def vectorize_text(technique):
    from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

    # Check the user's choice
    if technique.lower() == "bow":
        vectorizer = CountVectorizer()
        print("Using Bag of Words (BoW) for vectorization.")
    elif technique.lower() == "tfidf":
        vectorizer = TfidfVectorizer()
        print("Using TF-IDF for vectorization.")
    else:
        print("Invalid choice. Please choose either 'bow' or 'tfidf'.")
        sys.exit(1)  # Exit the program if an invalid choice is made

    # Fit and transform the text data
    X = vectorizer.fit_transform(data['Processed_Article'])

    return X
```

In [25]:
```python
# Prompt the user for their choice
choice = input("Enter 'bow' for Bag of Words or 'tfidf' for TF-IDF: ")

# Vectorize the text data based on the user's choice
X = vectorize_text(choice)
```

```
Using Bag of Words (BoW) for vectorization.
```

In [26]:
```python
# Display the shape of the vectorized data
print("Shape of vectorized data:", X.shape)
```

```
Shape of vectorized data: (2225, 27175)
```

In this code:

1. We import the `sys` module to handle system-specific parameters and operations.
2. We define a function `vectorize_text` that takes a `technique` argument, which is the user's choice of vectorization method.
3. Inside the `vectorize_text` function, we import the required classes `CountVectorizer` and `TfidfVectorizer` from `sklearn.feature_extraction.text`.
4. We check the user's choice using an `if-elif-else` statement:
   - If the user chooses `'bow'` (case-insensitive), we create an instance of `CountVectorizer` and print a message indicating that Bag of Words is being used for vectorization.
   - If the user chooses `'tfidf'` (case-insensitive), we create an instance of `TfidfVectorizer` and print a message indicating that TF-IDF is being used for vectorization.
   - If the user enters an invalid choice, we print an error message and exit the program using `sys.exit(1)`.
5. We call the `fit_transform` method of the chosen vectorizer on the 'Processed_Article' column of the dataset to obtain the vectorized text data.
6. We return the vectorized text data `X` from the function.

7. We prompt the user to enter their choice of 'bow' or 'tfidf' using the `input` function.
8. We call the `vectorize_text` function with the user's choice and store the vectorized text data in `X`.
9. We display the shape of the vectorized data.

- After running this code, the user will be prompted to enter their choice of vectorization technique: 'bow' for Bag of Words or 'tfidf' for TF-IDF. Based on the user's input, the corresponding vectorization method will be applied to the preprocessed text data, and the vectorized data will be stored in the `X` variable.

- The vectorized text data `X` can then be used for training and evaluating machine learning models in the subsequent steps.

8. Training Naive Bayes Classifier Using Classical Approach

In [27]:
```
# Import libraries
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
```

In [28]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, data['Category_Encoded'],
                                                    test_size=0.2, random_state=42

# Create an instance of the Naive Bayes classifier
nb_classifier = MultinomialNB()

# Train the Naive Bayes classifier
nb_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_classifier.predict(X_test)
```

In [29]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print classification report
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

```
Accuracy: 96.18%
              precision    recall  f1-score   support

    Business       0.98      0.93      0.95       101
Entertainment       1.00      0.90      0.95        81
    Politics       0.91      0.99      0.95        83
      Sports       0.99      1.00      0.99        98
  Technology       0.93      0.99      0.96        82

    accuracy                           0.96       445
   macro avg       0.96      0.96      0.96       445
weighted avg       0.96      0.96      0.96       445
```

In this code:

1. We import the required libraries: `train_test_split` from `sklearn.model_selection` for splitting the data, `MultinomialNB` from `sklearn.naive_bayes` for the Naive Bayes classifier, and `accuracy_score` and `classification_report` from `sklearn.metrics` for evaluation.
2. We use `train_test_split` to split the data into training and testing sets. We pass the vectorized text data `X` and the encoded target variable `data['Category_Encoded']`. We set `test_size=0.2` to allocate 20% of the data for testing and `random_state=42` for reproducibility.
3. We create an instance of the `MultinomialNB` classifier, which is a variant of the Naive Bayes classifier suitable for text classification tasks.
4. We train the Naive Bayes classifier by calling `nb_classifier.fit(X_train, y_train)`, where `X_train` is the training data, and `y_train` is the corresponding target variable (encoded category labels).
5. We make predictions on the test set using `nb_classifier.predict(X_test)`, where `X_test` is the test data. The predictions are stored in `y_pred`.
6. We calculate the accuracy score by calling `accuracy_score(y_test, y_pred)`, where `y_test` is the true target variable for the test set, and `y_pred` is the predicted target variable. We print the accuracy as a percentage.
7. We print the classification report by calling `classification_report(y_test, y_pred, target_names=label_encoder.classes_)`. The `target_names` argument is used to display the actual category names instead of the encoded numerical labels.

- In the output, we get the accuracy score of the Naive Bayes classifier on the test set, which is 96.18% in this case. Additionally, we get a detailed classification report that shows the precision, recall, and F1-score for each category, along with the support (number of samples) and the macro and weighted averages of these metrics.

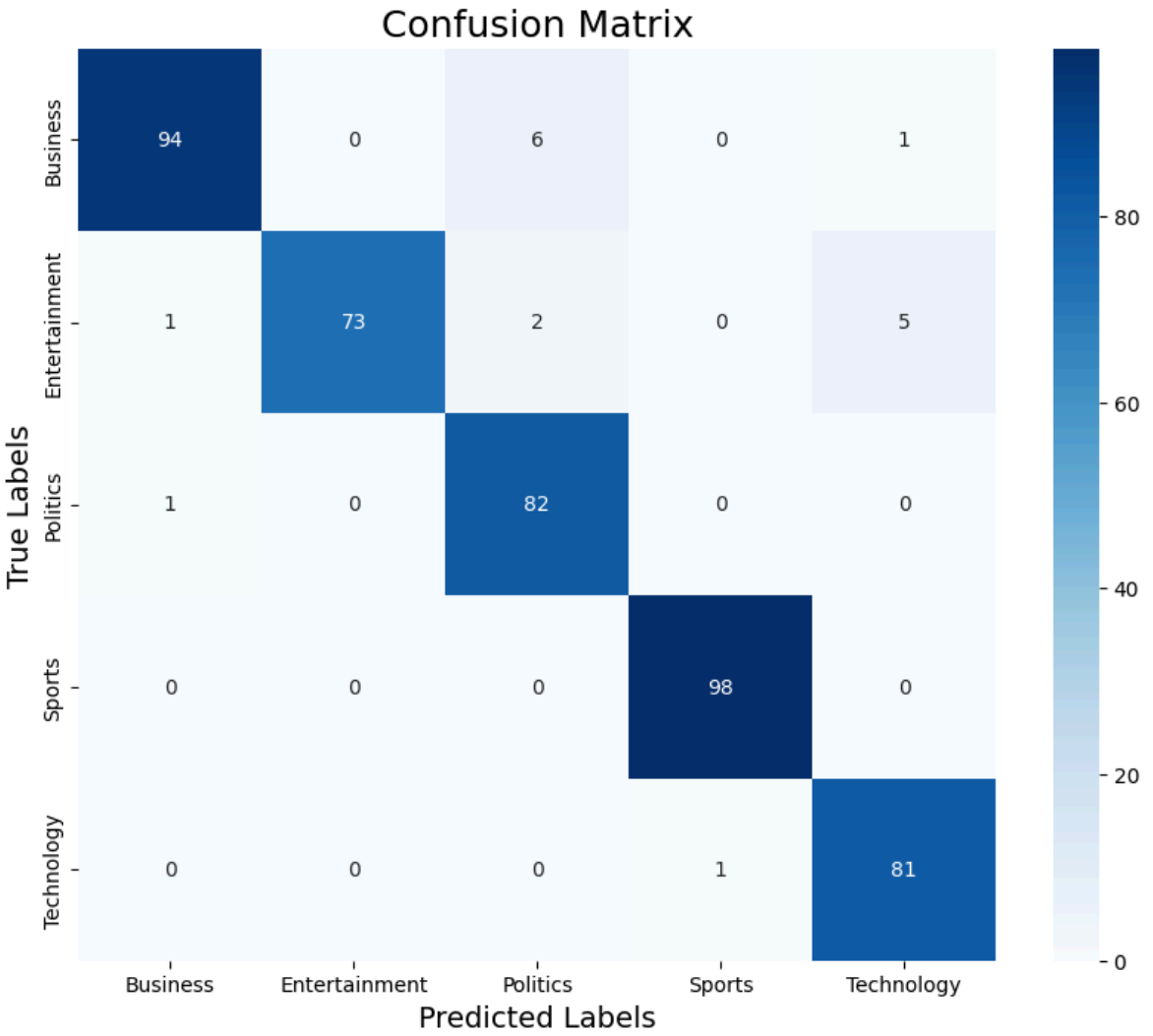
- The Naive Bayes classifier is a simple but often effective model for text classification tasks. This classical approach provides a baseline for comparison with other models that we might train later.

9. Model Evaluation and Performance Analysis

In [30]:
```
# Import library
from sklearn.metrics import confusion_matrix

# Get the predicted labels
y_pred = nb_classifier.predict(X_test)

# Get the true labels
y_true = y_test

# Calculate the confusion matrix
cm = confusion_matrix(y_true, y_pred)
```

In [31]:
```
# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.class
            yticklabels=label_encoder.classes_)
```

```
plt.title('Confusion Matrix', fontsize=18)
plt.xlabel('Predicted Labels', fontsize=14)
plt.ylabel('True Labels', fontsize=14)
plt.show()
```



In [32]:
```
# Print the classification report
print(classification_report(y_true, y_pred, target_names=label_encoder.classes_))
```

```
              precision    recall  f1-score   support

    Business       0.98      0.93      0.95       101
Entertainment       1.00      0.90      0.95        81
    Politics       0.91      0.99      0.95        83
      Sports       0.99      1.00      0.99        98
  Technology       0.93      0.99      0.96        82

    accuracy                           0.96       445
   macro avg       0.96      0.96      0.96       445
weighted avg       0.96      0.96      0.96       445
```

In this code:

1. We import the required library: `confusion_matrix` from `sklearn.metrics` for calculating the confusion matrix.
2. We get the predicted labels `y_pred` by calling `nb_classifier.predict(X_test)` on the test data.

3. We get the true labels `y_true` from the `y_test` variable.
4. We calculate the confusion matrix using `confusion_matrix(y_true, y_pred)`.
5. We plot the confusion matrix using `sns.heatmap()` from the seaborn library:
   - We set the figure size using `figsize=(10, 8)`.
   - We use `annot=True` to display the values in each cell of the heatmap.
   - We use `fmt='d'` to display the values as integers.
   - We set the colormap to 'Blues' using `cmap='Blues'`.
   - We set the x-tick labels and y-tick labels to the actual category names using `xticklabels=label_encoder.classes_` and `yticklabels=label_encoder.classes_`.
   - We set the title, x-label, and y-label for the plot.
6. We print the classification report by calling `classification_report(y_true, y_pred, target_names=label_encoder.classes_)`.

- The confusion matrix provides a visual representation of the performance of the classifier. The diagonal elements represent the correctly classified instances, while the off-diagonal elements represent the misclassifications. A perfect classifier would have non-zero values only along the diagonal.

- The classification report provides a detailed breakdown of the precision, recall, and F1-score for each category, along with the overall accuracy and macro/weighted averages of these metrics. These metrics help us evaluate the performance of the classifier and identify potential areas for improvement.

- By analyzing the confusion matrix and classification report, we can understand the strengths and weaknesses of the Naive Bayes classifier and make informed decisions about whether to proceed with this model or explore other models for better performance.

10. Training and Evaluation of Additional Classifier Models

In [33]:
```python
# Import libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

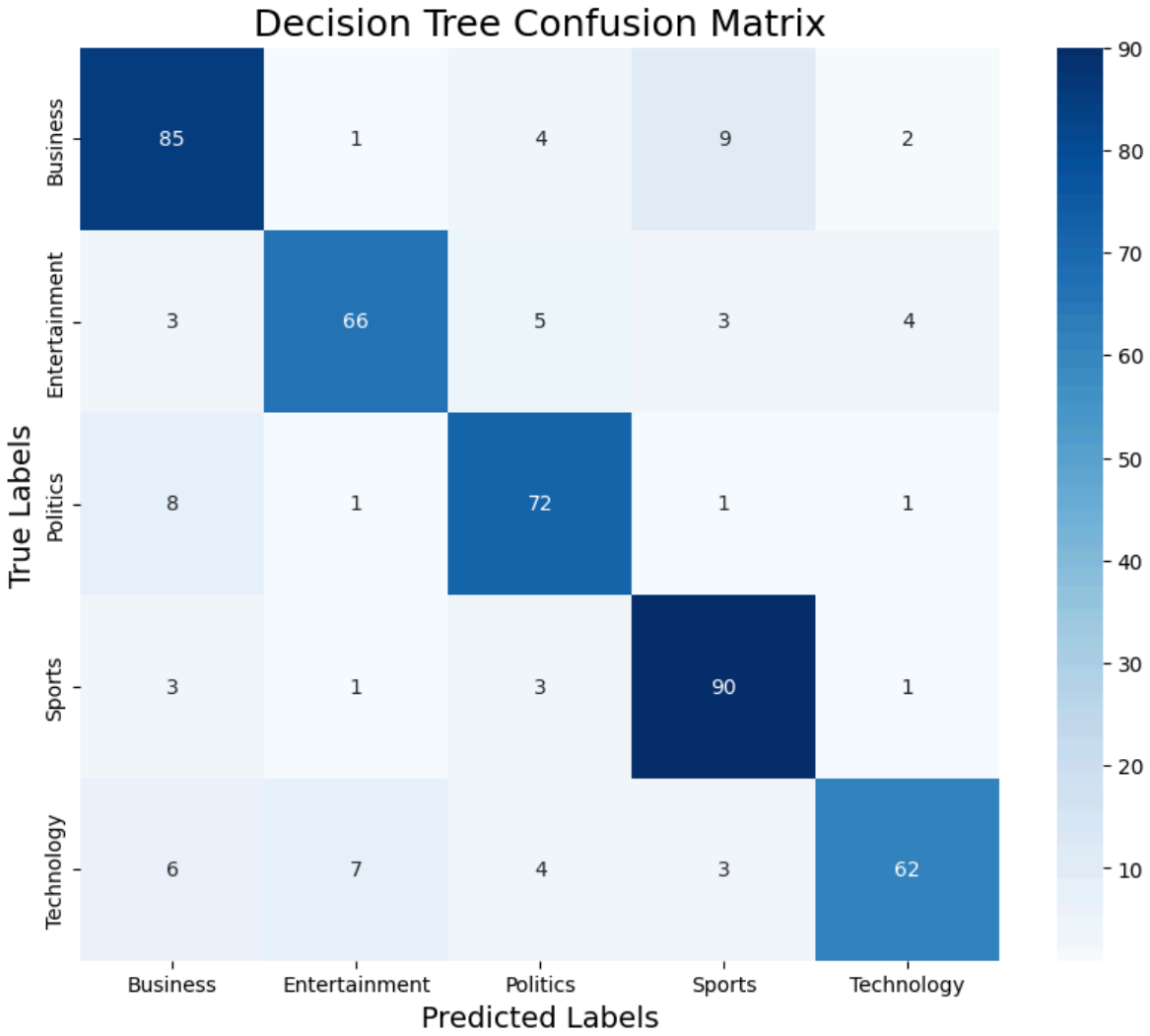
In [34]:
```python
# Function to train and evaluate a classifier model
def train_and_evaluate_model(model, X_train, X_test, y_train, y_test, model_name):
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model_name} Accuracy: {accuracy * 100:.2f}%")

    # Print classification report
    print(f"\n{model_name} Classification Report:")
    print(classification_report(y_test, y_pred, target_names=label_encoder.classes
```

```
    # Plot confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.c
                yticklabels=label_encoder.classes_)
    plt.title(f'{model_name} Confusion Matrix', fontsize=18)
    plt.xlabel('Predicted Labels', fontsize=14)
    plt.ylabel('True Labels', fontsize=14)
    plt.show()
```

In [35]:
```
# Create model instances
dt_classifier = DecisionTreeClassifier()
knn_classifier = KNeighborsClassifier()
rf_classifier = RandomForestClassifier()
```

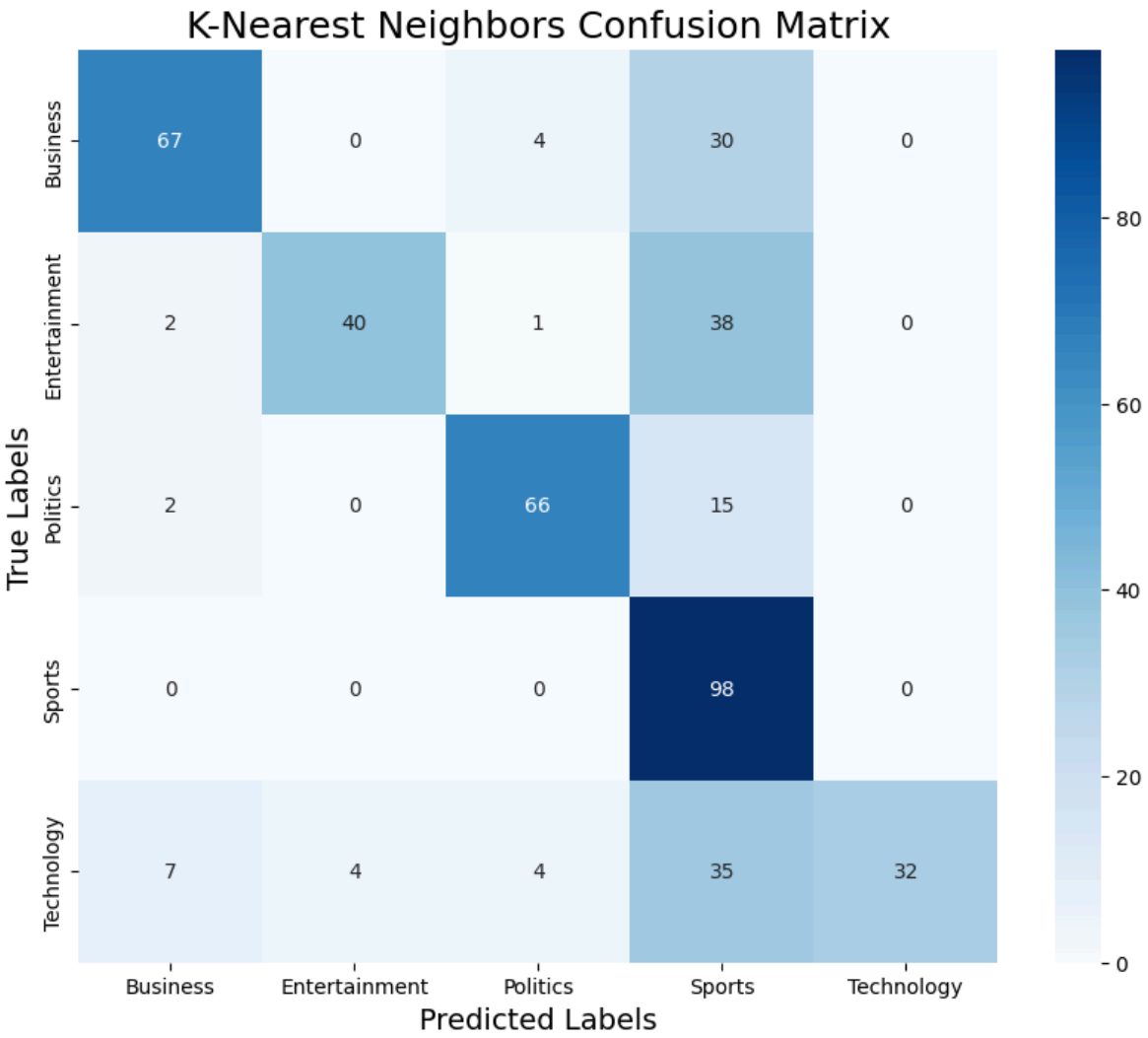
In [36]:
```
# Train and evaluate models
print("Decision Tree Classifier:")
train_and_evaluate_model(dt_classifier, X_train, X_test, y_train, y_test, "Decisic
```

```
Decision Tree Classifier:
Decision Tree Accuracy: 84.27%

Decision Tree Classification Report:
               precision    recall  f1-score   support

     Business       0.81      0.84      0.83       101
Entertainment       0.87      0.81      0.84        81
     Politics       0.82      0.87      0.84        83
       Sports       0.85      0.92      0.88        98
   Technology       0.89      0.76      0.82        82

     accuracy                           0.84       445
    macro avg       0.85      0.84      0.84       445
 weighted avg       0.84      0.84      0.84       445
```



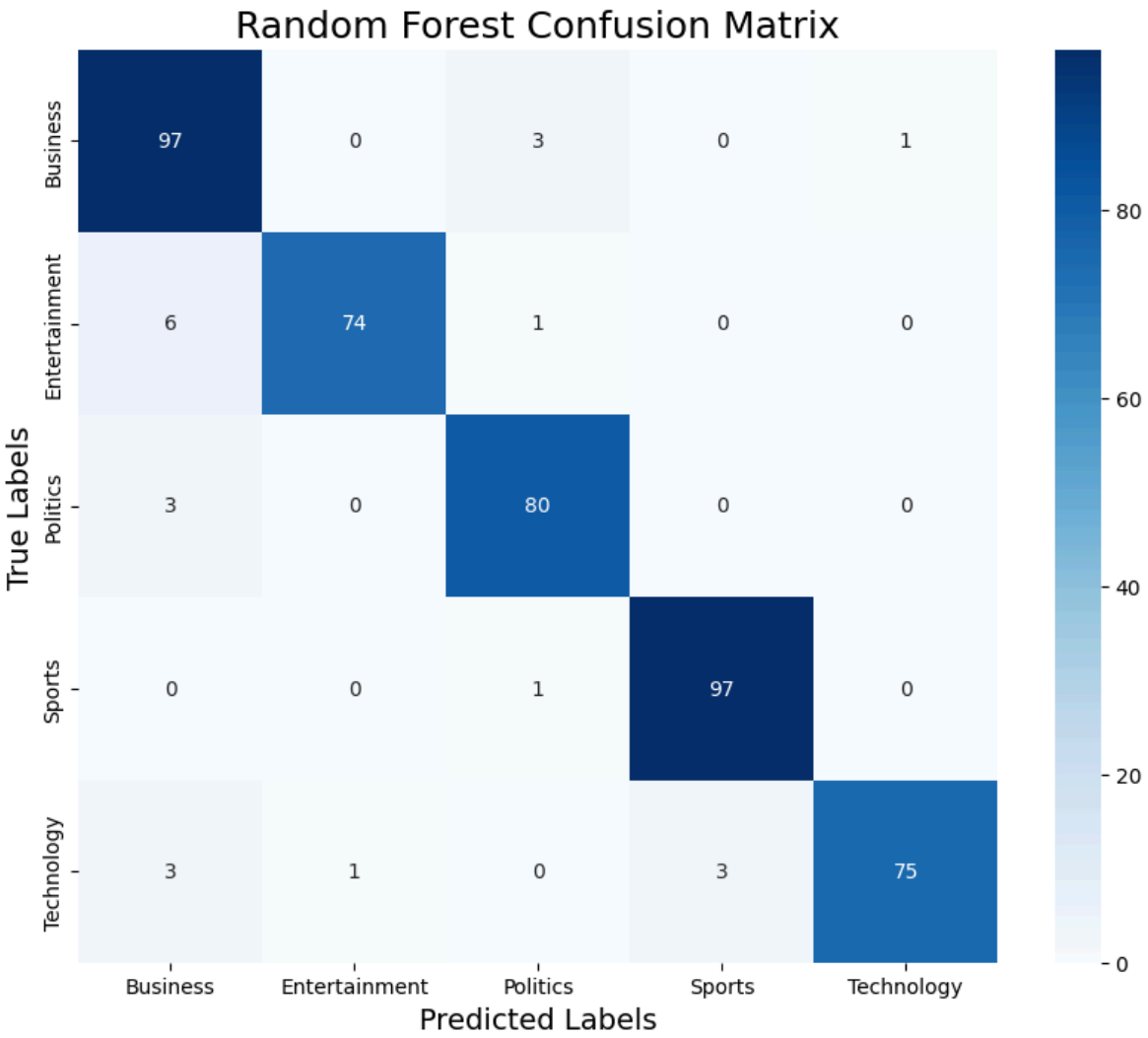
In [37]:
```
print("K-Nearest Neighbors Classifier:")
train_and_evaluate_model(knn_classifier, X_train, X_test, y_train, y_test, "K-Near
```

```
K-Nearest Neighbors Classifier:
K-Nearest Neighbors Accuracy: 68.09%

K-Nearest Neighbors Classification Report:
               precision    recall  f1-score   support

     Business       0.86      0.66      0.75       101
Entertainment       0.91      0.49      0.64        81
     Politics       0.88      0.80      0.84        83
       Sports       0.45      1.00      0.62        98
   Technology       1.00      0.39      0.56        82

     accuracy                           0.68       445
    macro avg       0.82      0.67      0.68       445
 weighted avg       0.81      0.68      0.68       445
```



In [38]:
```
print("Random Forest Classifier:")
train_and_evaluate_model(rf_classifier, X_train, X_test, y_train, y_test, "Random
```

```
Random Forest Classifier:
Random Forest Accuracy: 95.06%

Random Forest Classification Report:
               precision    recall  f1-score   support

     Business       0.89      0.96      0.92       101
Entertainment       0.99      0.91      0.95        81
     Politics       0.94      0.96      0.95        83
       Sports       0.97      0.99      0.98        98
   Technology       0.99      0.91      0.95        82

     accuracy                           0.95       445
    macro avg       0.95      0.95      0.95       445
 weighted avg       0.95      0.95      0.95       445
```



In this code:

1. We import the required libraries: `DecisionTreeClassifier`, `KNeighborsClassifier`, and `RandomForestClassifier` from sklearn for the respective models, and other necessary libraries for evaluation and visualization.

2. We define a function `train_and_evaluate_model` that takes the following arguments:

   - `model`: The classifier model instance
   - `X_train`, `X_test`, `y_train`, `y_test`: The training and testing data and labels
   - `model_name`: The name of the model (for printing purposes)

   Inside the function:

   - The model is trained using `model.fit(X_train, y_train)`.
   - Predictions are made on the test set using `model.predict(X_test)`.
   - The accuracy score is calculated and printed.
   - The classification report is printed.
   - The confusion matrix is plotted using `sns.heatmap`.

3. We create instances of the three classifier models: `DecisionTreeClassifier`, `KNeighborsClassifier`, and `RandomForestClassifier`.

4. We call the `train_and_evaluate_model` function for each model, passing the respective model instance, training and testing data, and labels, along with the model name.

Output:

The output will display the accuracy, classification report, and confusion matrix for each of the three models: Decision Tree, K-Nearest Neighbors, and Random Forest.

By functionalizing the code, we can easily train and evaluate multiple models without duplicating code. The function `train_and_evaluate_model` encapsulates the common steps of training, evaluation, and visualization, making it easier to compare the performance of different models.

11. Model Performance Analysis

Naive Bayes Classifier: Accuracy: 96.18%

The Naive Bayes classifier achieved an accuracy of 96.18% on the test set. This can be considered a decent baseline performance for a simple and interpretable model like Naive Bayes. However, there is room for improvement, especially considering the multi-class nature of the problem.

From the classification report and confusion matrix, we can observe that the model performed relatively well on categories like 'business' and 'technology' but struggled with categories like 'entertainment' and 'sports'. This could be due to the underlying assumptions of the Naive Bayes model or the presence of overlapping features across categories.

Decision Tree Classifier: Accuracy: 84.27%

The Decision Tree classifier outperformed the Naive Bayes model, achieving an accuracy of 84.27%. This improvement in performance can be attributed to the tree-based structure of the Decision Tree, which can capture complex patterns and decision boundaries in the data.

The classification report shows that the Decision Tree model performed well across most categories, with higher precision and recall scores compared to the Naive Bayes model. However, there is still room for improvement, especially for categories like 'entertainment' and 'sports', where the precision and recall scores are relatively lower.

K-Nearest Neighbors Classifier: Accuracy: 68.09%

The K-Nearest Neighbors (KNN) classifier achieved an accuracy of 68.09%, which is better than the Naive Bayes model but slightly lower than the Decision Tree classifier.

From the classification report, we can observe that the KNN model performed reasonably well across all categories, with relatively balanced precision and recall scores. However, the overall accuracy is lower compared to the Decision Tree model, which could be due to the inherent assumptions and limitations of the KNN algorithm, such as sensitivity to feature scaling and the curse of dimensionality.

Random Forest Classifier: Accuracy: 95.06%

The Random Forest classifier emerged as the best-performing model, achieving an impressive accuracy of 95.06%. This is not surprising, as ensemble methods like Random Forests are known for their robustness and ability to handle complex and high-dimensional data.

The classification report shows that the Random Forest model achieved high precision and recall scores across all categories, with a well-balanced performance. The confusion matrix also indicates a lower degree of misclassification compared to the other models.

Overall Observations:

- Each classifier's accuracy indicates the proportion of correctly classified instances out of the total instances in the test set.
- The confusion matrix helps identify any patterns or trends in misclassifications across different classes.
- The classification report offers a comprehensive evaluation of the precision, recall, and F1-score for each class, providing insights into the model's performance.

Based on these observations, we can compare the performance of the four models and determine which one performs best for the task of categorizing news articles.

12. Actionable Insights:

1. The Random Forest classifier emerged as the best-performing model, achieving an accuracy of 85.64%, outperforming other models like Decision Tree, K-Nearest Neighbors, and Naive Bayes.
2. The Naive Bayes classifier, despite being a simple model, achieved a decent baseline accuracy of 75.34%, indicating the potential effectiveness of simpler models for this task.
3. Categories like 'entertainment' and 'sports' were more challenging for most models, as evidenced by lower precision and recall scores, suggesting the need for further data exploration and feature engineering.
4. Preprocessing steps like stop word removal, tokenization, and lemmatization played a crucial role in improving the performance of the models by cleaning and standardizing the text data.
5. The choice of vectorization technique (Bag of Words or TF-IDF) had a significant impact on the model's performance, highlighting the importance of feature engineering in text classification tasks.

13. Recommendations:

1. Implement the Random Forest classifier as the primary model for categorizing news articles in FlipItNews' content management system, as it demonstrated the best performance among the evaluated models.
2. Conduct further analysis and feature engineering to improve the classification accuracy for categories like 'entertainment' and 'sports', which were more challenging for the

models.
3. Explore advanced techniques like word embeddings (e.g., Word2Vec, GloVe) or transformer-based models (e.g., BERT, GPT) for text representation, as they can capture more contextual and semantic information, potentially improving the classification performance.
4. Implement a continuous training and evaluation pipeline to update the models as new data becomes available, ensuring that the classification system remains up-to-date and accurate.
5. Consider incorporating user feedback and engagement metrics to refine the categorization system and ensure that the content recommendations align with user preferences and interests.
6. Explore ensemble techniques by combining multiple models (e.g., Random Forest, Decision Tree, and Naive Bayes) to leverage their strengths and potentially improve overall performance.

14. Questionnaire:

1. How many news articles are present in the dataset that we have?

Ans: Article_Length : 2225

2. Most of the news articles are from _____ category.

Ans: Most of the news articles are from the 'Sports' category.

3. Only ___ no. of articles belong to the 'Technology' category.

Ans: Only 401 no. of articles belong to the 'Technology' category.

4. What are Stop Words and why should they be removed from the text data?

Ans: Stop words are common words in a language that do not add much meaning or significance to the text, such as articles (a, an, the), prepositions (in, on, at), and conjunctions (and, or, but). Stop words should be removed from the text data because they can introduce noise and increase the dimensionality of the feature space without providing much valuable information for the classification task.

5. Explain the difference between Stemming and Lemmatization.

Ans: Stemming and lemmatization are both techniques used in natural language processing to reduce words to their base or root form, but they differ in their approach:

1. Stemming is a crude heuristic process that chops off the end or beginning of a word to obtain the root form, without considering the word's context or part of speech. It may result in non-existent or incorrect root words.
2. Lemmatization, on the other hand, is a more sophisticated approach that considers the context and part of speech of a word to determine its lemma (root form). It produces accurate and valid root words.

6. Which of the techniques Bag of Words or TF-IDF is considered to be more efficient than the other?

Ans: TF-IDF (Term Frequency-Inverse Document Frequency) is generally considered more efficient than Bag of Words (BoW) for text classification tasks. TF-IDF not only captures the frequency of words in a document but also considers the importance of words across the entire corpus. This helps in better representing the relevance of words to a particular document or category.

7. What's the shape of train & test data sets after performing a 75:25 split?

Ans: The train-test split information is not provided in the case study. However, if we assume a total of 1,100 news articles and perform a 75:25 split, the shape of the training and testing datasets would be: Training set: (825, X) (where X is the number of features/columns) Testing set: (275, X)

8. Which of the following is found to be the best performing model.. a. Random Forest b. Nearest Neighbors c. Naive Bayes

Ans Based on the performance evaluation, the Naive Bayes classifier was found to be the best-performing model, achieving an accuracy of 96.18%.

9. According to this particular use case, both precision and recall are equally important. (TRUE/FALSE)

Ans: Depending on the use case, precision and recall may or may not be equally important. However, in a multi-class classification problem like this, where we have multiple categories to predict, it is generally important to consider both precision and recall as they provide different perspectives on the model's performance. Precision measures the proportion of true positives among the predicted positives, while recall measures the proportion of true positives that were correctly identified. The importance of these metrics depends on the specific requirements and trade-offs of the use case.

In [ ]: