Importing the Libraries & Reading the Data File

```
# Data Manipulation
import numpy as np
import pandas as pd

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# NLP Libraries
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Machine Learning Libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import classification_report, confusion_matrix

# Classifiers
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# Ignore warnings for clean outputs
import warnings
warnings.filterwarnings('ignore')
```

Reading the Data File

```
# Reading the dataset
data = pd.read_csv('flipitnews-data.csv', encoding='latin-1')  # Adjust encoding if needed
```

Exploring the Dataset

Shape of the Dataset

```
print("Dataset Shape:", data.shape)
```

```
Dataset Shape: (2225, 2)
```

Viewing the First Few Rows

```
data.head()
```

	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...

Next steps: Generate code with data | View recommended plots | New interactive sheet

Checking for Missing Values

```
print("Missing Values:\n", data.isnull().sum())
```

```
Missing Values:
 Category    0
Article     0
dtype: int64
```

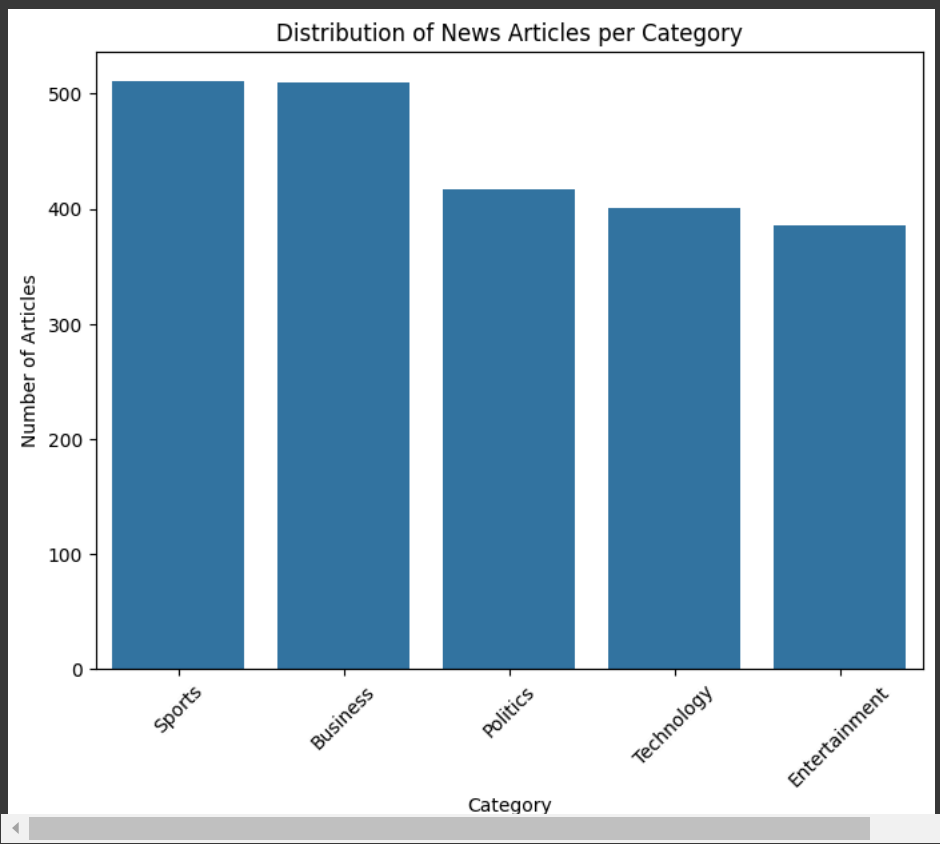
News Articles per Category

```
# Counting articles per category
category_counts = data['Category'].value_counts()
print("Articles per Category:\n", category_counts)
```

```
Articles per Category:
 Category
Sports           511
Business         510
Politics         417
Technology       401
Entertainment    386
Name: count, dtype: int64
```

Per Category Visualization

```
plt.figure(figsize=(8,6))
sns.countplot(x='Category', data=data, order=category_counts.index)
plt.title('Distribution of News Articles per Category')
plt.xlabel('Category')
plt.ylabel('Number of Articles')
plt.xticks(rotation=45)
plt.show()
```



Processing the Textual Data

Downloading NLTK Data

```
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
True
```

Defining the Text Processing Function

```
def preprocess_text(text):
    # Remove non-letter characters
    text = re.sub("[^a-zA-Z]", " ", text)

    # Convert to lowercase
    text = text.lower()

    # Tokenize the text
    words = nltk.word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words("english"))
    words = [word for word in words if word not in stop_words]

    # Perform lemmatization
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    # Join words back into one string separated by space
    return " ".join(words)
```

Applying the Function to the Dataset

```
print("Original Article:\n")
print(data['Article'][0])
```

```
Original Article:

tv future in the hands of viewers with home theatre systems  plasma high-definition tvs  and digital video recorders moving into the
```

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

```
print("\nProcessed Article:\n")
print(data['Processed_Article'][0])
```

```
Processed Article:

tv future hand viewer home theatre system plasma high definition tv digital video recorder moving living room way people watch tv ra
```

Encoding and Transforming the Data

Encoding the Target Variable

```
label_encoder = LabelEncoder()
data['Category_Encoded'] = label_encoder.fit_transform(data['Category'])

# Display label mapping
label_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
print("\nLabel Mapping:")
print(label_mapping)
```

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

Vectorizing the Data

```
# Option to choose vectorization method
vectorization_method = input("Choose vectorization method (Enter 'bow' for Bag of Words or 'tfidf' for TF-IDF): ").strip().lower()

if vectorization_method == 'bow':
    vectorizer = CountVectorizer()
    print("Using Bag of Words vectorization.\n")
elif vectorization_method == 'tfidf':
    vectorizer = TfidfVectorizer()
    print("Using TF-IDF vectorization.\n")
else:
    print("Invalid input. Defaulting to TF-IDF vectorization.\n")
    vectorizer = TfidfVectorizer()
```

```
Choose vectorization method (Enter 'bow' for Bag of Words or 'tfidf' for TF-IDF): tfidf
Using TF-IDF vectorization.
```

```
# Features and labels
X = vectorizer.fit_transform(data['Processed_Article'])
y = data['Category_Encoded']

print("Feature vector shape:", X.shape)
```

```
Feature vector shape: (2225, 24728)
```

Performing Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

print("Shape of Training Data:", X_train.shape)
print("Shape of Testing Data:", X_test.shape)
```

```
Shape of Training Data: (1668, 24728)
Shape of Testing Data: (557, 24728)
```

Model Training & Evaluation

Defining a Function to Train and Evaluate Models

```
def train_and_evaluate_model(model, model_name):
    # Train the model
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Evaluation Metrics
    print(f"\n{model_name} Evaluation:\n")
    print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=label_encoder.classes_,
                yticklabels=label_encoder.classes_)
    plt.title(f'{model_name} Confusion Matrix')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
```

Training a Naive Bayes Classifier

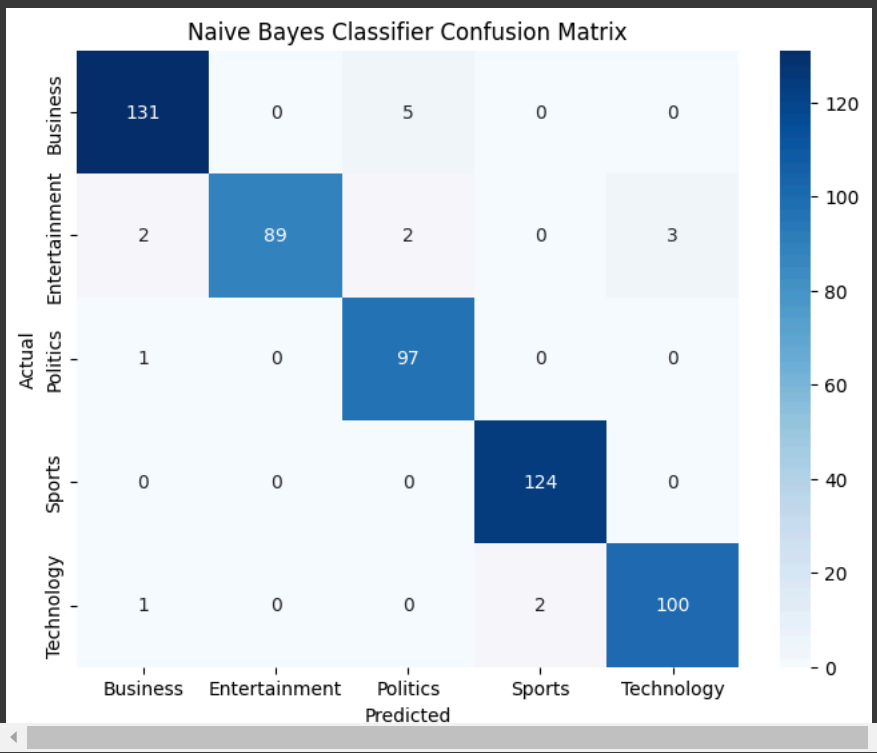
```
nb_classifier = MultinomialNB()
train_and_evaluate_model(nb_classifier, "Naive Bayes Classifier")
```

```
Naive Bayes Classifier Evaluation:

                precision    recall  f1-score   support

      Business       0.97      0.96      0.97       136
 Entertainment       1.00      0.93      0.96        96
      Politics       0.93      0.99      0.96        98
        Sports       0.98      1.00      0.99       124
    Technology       0.97      0.97      0.97       103

      accuracy                           0.97       557
     macro avg       0.97      0.97      0.97       557
  weighted avg       0.97      0.97      0.97       557
```



Training a Decision Tree Classifier

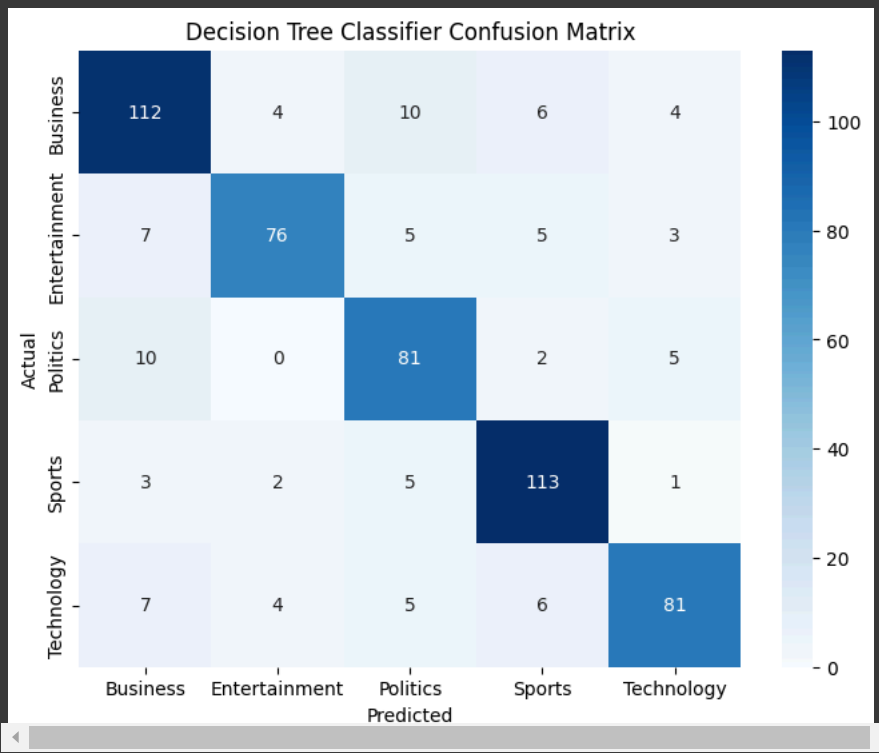
```
dt_classifier = DecisionTreeClassifier(random_state=42)
train_and_evaluate_model(dt_classifier, "Decision Tree Classifier")
```

```
Decision Tree Classifier Evaluation:

               precision    recall  f1-score   support

     Business       0.81      0.82      0.81       136
Entertainment       0.88      0.79      0.84        96
     Politics       0.76      0.83      0.79        98
       Sports       0.86      0.91      0.88       124
   Technology       0.86      0.79      0.82       103

     accuracy                           0.83       557
    macro avg       0.83      0.83      0.83       557
 weighted avg       0.83      0.83      0.83       557
```



Training a K-Nearest Neighbors Classifier

```
knn_classifier = KNeighborsClassifier()
train_and_evaluate_model(knn_classifier, "K-Nearest Neighbors Classifier")
```

```
K-Nearest Neighbors Classifier Evaluation:

               precision    recall  f1-score   support

     Business       0.94      0.88      0.91       136
Entertainment       0.96      0.94      0.95        96
     Politics       0.86      0.91      0.88        98
       Sports       0.96      1.00      0.98       124
   Technology       0.97      0.97      0.97       103

     accuracy                           0.94       557
    macro avg       0.94      0.94      0.94       557
```

Training a Random Forest Classifier

```
rf_classifier = RandomForestClassifier(random_state=42)
train_and_evaluate_model(rf_classifier, "Random Forest Classifier")
```

```
Random Forest Classifier Evaluation:

               precision    recall  f1-score   support

     Business       0.90      0.96      0.93       136
Entertainment       0.96      0.93      0.94        96
     Politics       0.96      0.93      0.94        98
       Sports       0.98      0.99      0.98       124
   Technology       0.96      0.90      0.93       103

     accuracy                           0.95       557
    macro avg       0.95      0.94      0.95       557
 weighted avg       0.95      0.95      0.95       557
```