News Data Analysis

```
[1] import pandas as pd
    import numpy as np
    import nltk
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
[2] # Loaddataset
    df = pd.read_csv('/content/data_news - data_news.csv')
```

```
[3] df
```

	category	headline	links	short_description	keywords
0	WELLNESS	143 Miles in 35 Days: Lessons Learned	https://www.huffingtonpost.com/entry/running-l...	Resting is part of training. I've confirmed wh...	running-lessons
1	WELLNESS	Talking to Yourself: Crazy or Crazy Helpful?	https://www.huffingtonpost.com/entry/talking-t...	Think of talking to yourself as a tool to coac...	talking-to-yourself-crazy
2	WELLNESS	Crenezumab: Trial Will Gauge Whether Alzheimer...	https://www.huffingtonpost.com/entry/crenezuma...	The clock is ticking for the United States to ...	crenezumab-alzheimers-disease-drug
3	WELLNESS	Oh, What a Difference She Made	https://www.huffingtonpost.com/entry/meaningfu...	If you want to be busy, keep trying to be perf...	meaningful-life
4	WELLNESS	Green Superfoods	https://www.huffingtonpost.com/entry/green-sup...	First, the bad news: Soda bread, corned beef a...	green-superfoods
...	...	...	...	...	...

0s completed at 16:15

```
[ ] print(df.isnull().sum())
```

```
category                0
headline                0
links                   0
short_description       0
keywords             2668
dtype: int64
```

```
[6] df['keywords'].fillna('Unknown', inplace=True)
```

```
[5] print(df.duplicated().sum())
```

```
4251
```

```
[ ] df.drop_duplicates(inplace=True)
```

```
[ ] df = df.drop(columns=['keywords'])
```

```
df
```

	category	headline	links	short_description
0	WELLNESS	143 Miles in 35 Days: Lessons Learned	https://www.huffingtonpost.com/entry/running-l...	Resting is part of training. I've confirmed wh...
1	WELLNESS	Talking to Yourself: Crazy or Crazy Helpful?	https://www.huffingtonpost.com/entry/talking-t...	Think of talking to yourself as a tool to coac...

```
[7] nltk.download('stopwords')
    nltk.download('punkt')
    nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
[8] from nltk.corpus import stopwords
```

```
from nltk.tokenize import word_tokenize

def remove_stopwords(text):
    tokens = word_tokenize(text)
    tokens=[token.strip()for token in tokens]

    filtered_tokens=[]
    for token in tokens:
      if token.lower() not in stopwords.words('english'):
        filtered_tokens.append(token)

    return ' '.join(filtered_tokens)
```

```
[10] bookmark=df.copy()
```

```
[11] df.iloc[:, 1:] = df.iloc[:, 1:].apply(lambda x: x.map(remove_stopwords))
```

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()

def porter_stemmer(text):
    if not isinstance(text, str):  # Ensure input is a string
        return text

    stemmed_words = [porter.stem(word) for word in word_tokenize(text)]

    return ' '.join(stemmed_words)
```

+ Code + Text

```
[14] df.iloc[:, 1:] = df.iloc[:, 1:].apply(lambda x: x.map(porter_stemmer))
```

```
[15]
from sklearn.preprocessing import LabelBinarizer

lb=LabelBinarizer()

df['category']=lb.fit_transform(df['category'])
```

```
[17] from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer= TfidfVectorizer()
```

```
[24] X_text = df[['headline', 'links', 'short_description', 'keywords']]
y = df['category']
```

```
[18] '''#alloting X and Y
X= df.iloc[:, 1:]
y= df.iloc[:,0]
```

```
# Convert text columns to TF-IDF vectors
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
```

```
[26] # Applying vectorization separately
X_headline = vectorizer.fit_transform(X_text['headline'].astype(str))
X_links = vectorizer.fit_transform(X_text['links'].astype(str))
X_short_desc = vectorizer.fit_transform(X_text['short_description'].astype(str))
X_keywords = vectorizer.fit_transform(X_text['keywords'].astype(str))
```

```
[28] # Combining all text feature vectors
from scipy.sparse import hstack

X = hstack([X_headline, X_links, X_short_desc, X_keywords])
```

```
[29] # Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[30] # Initialize and train logistic regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

```
[31] y_pred = lr.predict(X_test)
```

```
[32] # Print predictions
print(y_pred[:10])
```

```
[0 0 0 0 0 1 0 0 0 0]
```

```
[36] # Evaluate accuracy
from sklearn.metrics import accuracy_score, classification_report,f1_score,precision_score,recall_score

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
f1 = f1_score(y_test, y_pred, average='macro')
print(f"\nMacro F1-Score: {f1:.4f}")
precision = precision_score(y_test, y_pred, average='macro')
print(f"Macro Precision: {precision:.4f}")
recall = recall_score(y_test, y_pred, average='macro')
print(f"Macro Recall: {recall:.4f}")

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9607

Macro F1-Score: 0.8745
Macro Precision: 0.9194
Macro Recall: 0.8401

Classification Report:

              precision    recall  f1-score   support

           0       0.97      0.99      0.98      9045
           1       0.87      0.69      0.77       955

    accuracy                           0.96     10000
   macro avg       0.92      0.84      0.87     10000
weighted avg       0.96      0.96      0.96     10000
```

Visualtization

```
[38] # Count each category
category_counts = df['category'].value_counts()
category_counts
```

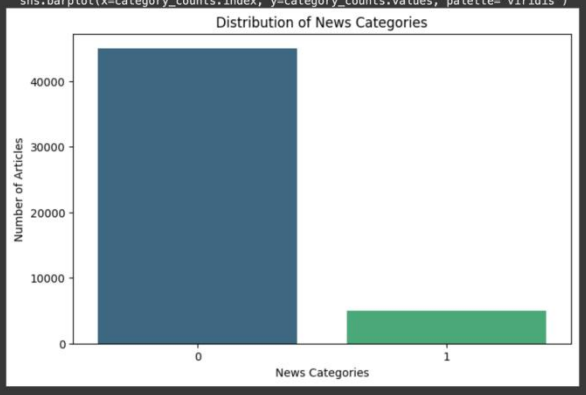
category	count
0	45000
1	5000

dtype: int64

```
[39] bookmark2=df.copy()
```

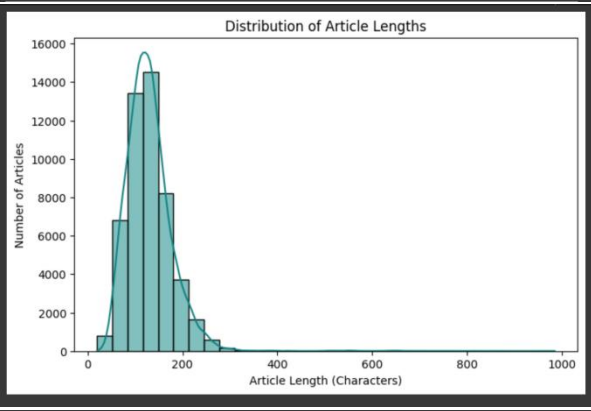
```
plt.figure(figsize=(8, 5))
sns.barplot(x=category_counts.index, y=category_counts.values, palette='viridis')
plt.xlabel("News Categories")
plt.ylabel("Number of Articles")
plt.title("Distribution of News Categories")
plt.show()
```

```
sns.barplot(x=category_counts.index, y=category_counts.values, palette='viridis')
```




```
# Calculate text length

df['text_length'] = df['headline'].astype(str).apply(len) + df['short_description'].astype(str).apply(len)
plt.figure(figsize=(8, 5))
sns.histplot(df['text_length'], bins=30, kde=True, color='teal')
plt.xlabel("Article Length (Characters)")
plt.ylabel("Number of Articles")
plt.title("Distribution of Article Lengths")
plt.show()
```




```
# Create DataFrame of actual vs. predicted categories
results_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
comparison_counts = results_df.groupby(['Actual', 'Predicted']).size().unstack().fillna(0)

#chart
comparison_counts.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='viridis')
plt.xlabel("Actual Category")
plt.ylabel("Number of Predictions")
plt.title("Predicted vs Actual Categories")
plt.legend(title="Predicted Category")
plt.xticks(rotation=45)
plt.show()
```