news_articlesNLP_finalv

May 18, 2025

[183]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download("stopwords")
nltk.download("punkt")
nltk.download("wordnet")
```

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

[183]:
```
True
```

[185]:
```
df = pd.read_csv("data_news - data_news.csv")
```

[187]:
```
df.head()
```

[187]:

	category	headline	\
0	WELLNESS	143 Miles in 35 Days: Lessons Learned	
1	WELLNESS	Talking to Yourself: Crazy or Crazy Helpful?	
2	WELLNESS	Crenezumab: Trial Will Gauge Whether Alzheimer…	
3	WELLNESS	Oh, What a Difference She Made	
4	WELLNESS	Green Superfoods	

	links	\

```
0  https://www.huffingtonpost.com/entry/running-l…
1  https://www.huffingtonpost.com/entry/talking-t…
2  https://www.huffingtonpost.com/entry/crenezuma…
3  https://www.huffingtonpost.com/entry/meaningfu…
4  https://www.huffingtonpost.com/entry/green-sup…

                                   short_description  \
0  Resting is part of training. I've confirmed wh…
1  Think of talking to yourself as a tool to coac…
2  The clock is ticking for the United States to …
3  If you want to be busy, keep trying to be perf…
4  First, the bad news: Soda bread, corned beef a…

                            keywords
0                    running-lessons
1           talking-to-yourself-crazy
2  crenezumab-alzheimers-disease-drug
3                    meaningful-life
4                   green-superfoods
```

[191]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   category           50000 non-null  object
 1   headline           50000 non-null  object
 2   links              50000 non-null  object
 3   short_description  50000 non-null  object
 4   keywords           47332 non-null  object
dtypes: object(5)
memory usage: 1.9+ MB
```

[195]:
```
df.describe()
```

[195]:
```
        category        headline  \
count      50000           50000
unique        10           45577
top     WELLNESS  Sunday Roundup
freq        5000              22

                                                   links  \
count                                              50000
unique                                             45745
top     https://www.huffingtonpost.com/entry/bryce-har…
freq                                                   8
```

```
                                                short_description keywords
count                                                       50000    47332
unique                                                      45743    41558
top     Along with his fists, the star Nationals outfi…     post
freq                                                            8       85
```

[197]:
```
df.isnull().sum()
```

[197]:
```
category               0
headline               0
links                  0
short_description      0
keywords            2668
dtype: int64
```

[199]:
```
# Dropping null values from the dataset
df.dropna(inplace = True)
```

[201]:
```
df.isnull().sum()
```

[201]:
```
category            0
headline            0
links               0
short_description   0
keywords            0
dtype: int64
```

[206]:
```
# Dropping "links" column as it is not necessary for analysis
df = df.drop(columns = ["links"])
```

[208]:
```
df.shape
```

[208]:
```
(47332, 4)
```

[210]:
```
# Initialize Lemmatizer
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words("english"))

# Preprocessing function
def preprocess_text(text):
    text = text.lower() # Convert to lowercase
    text = re.sub(r"[^\w\s]", "", text) # Remove punctuation
    text = re.sub(r"\d+", "", text) # Remove numbers
    words = word_tokenize(text) # Tokenization
    words = [word for word in words if word not in stop_words] # Remove␣
 ↪stopwords
    words = [lemmatizer.lemmatize(word) for word in words] # Lemmatization
```

```
    return " ".join(words) # Convert list back to string
# Apply the function to text columns

text_columns = ["headline", "short_description", "keywords"]
for col in text_columns:
    df[col] = df[col].apply(preprocess_text)
```

[212]:
```
df.head()
```

[212]:
```
   category                                           headline  \
0  WELLNESS                            mile day lesson learned
1  WELLNESS                          talking crazy crazy helpful
2  WELLNESS  crenezumab trial gauge whether alzheimers drug…
3  WELLNESS                                    oh difference made
4  WELLNESS                                      green superfoods

                                  short_description  \
0  resting part training ive confirmed sort alrea…
1  think talking tool coach challenge narrate exp…
2  clock ticking united state find cure team work…
3  want busy keep trying perfect want happy focus…
4  first bad news soda bread corned beef beer hig…

                         keywords
0                 runninglessons
1          talkingtoyourselfcrazy
2  crenezumabalzheimersdiseasedrug
3                 meaningfullife
4                greensuperfoods
```

[220]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Combine headline, short description, and keywords
df["combined_text"] = df["headline"] + " " + df["short_description"] + " " +␣
 ↪df["keywords"]

# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)

# Fit and transform the combined text data
X_tfidf = tfidf_vectorizer.fit_transform(df["combined_text"])

# Convert to a DataFrame for better understanding
tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=tfidf_vectorizer.
 ↪get_feature_names_out())

# Display shape and first few rows
```

```
print("TF-IDF Feature Shape:", tfidf_df.shape)
tfidf_df.head()
```

```
TF-IDF Feature Shape: (47332, 5000)
```

[220]:

	aaron	abandoned	abc	ability	able	aboard	abortion	abroad	absence	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	absolute	…	youre	youth	youtube	youve	zealand	zen	zero	zika	zoe	\
0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	zone
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

```
[5 rows x 5000 columns]
```

[224]:
```
# Count articles per category
category_counts = df["category"].value_counts()

# Display category distribution
category_counts
```

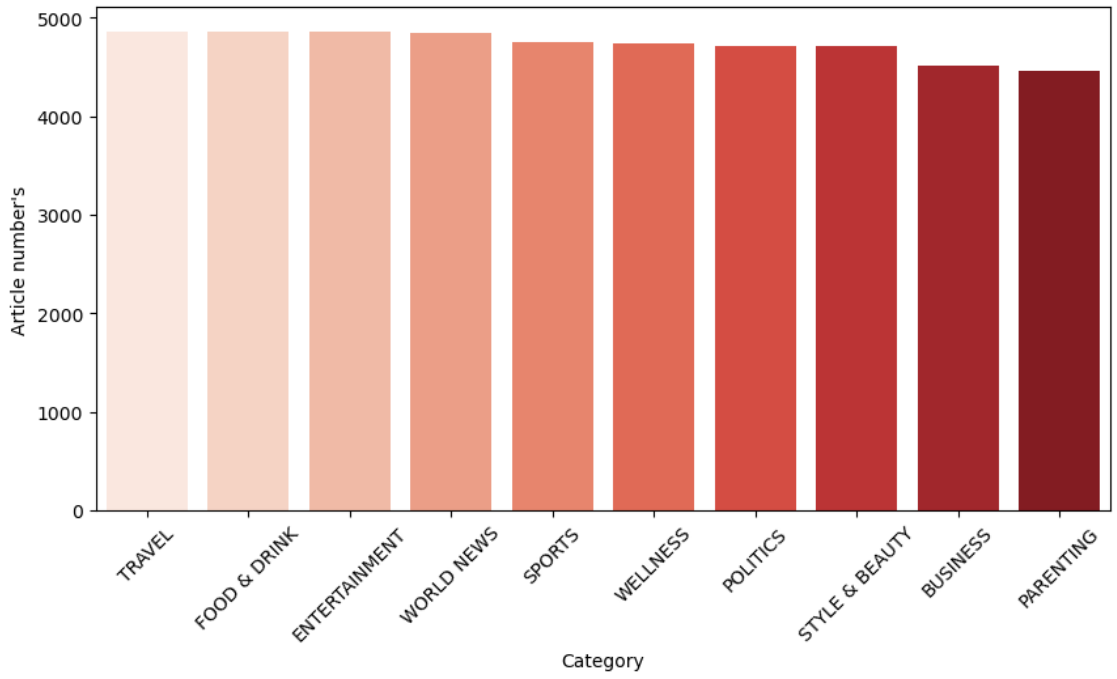
[224]:
```
category
TRAVEL            4865
FOOD & DRINK      4863
ENTERTAINMENT     4855
WORLD NEWS        4851
SPORTS            4759
WELLNESS          4741
POLITICS          4712
STYLE & BEAUTY    4708
BUSINESS          4512
PARENTING         4466
Name: count, dtype: int64
```

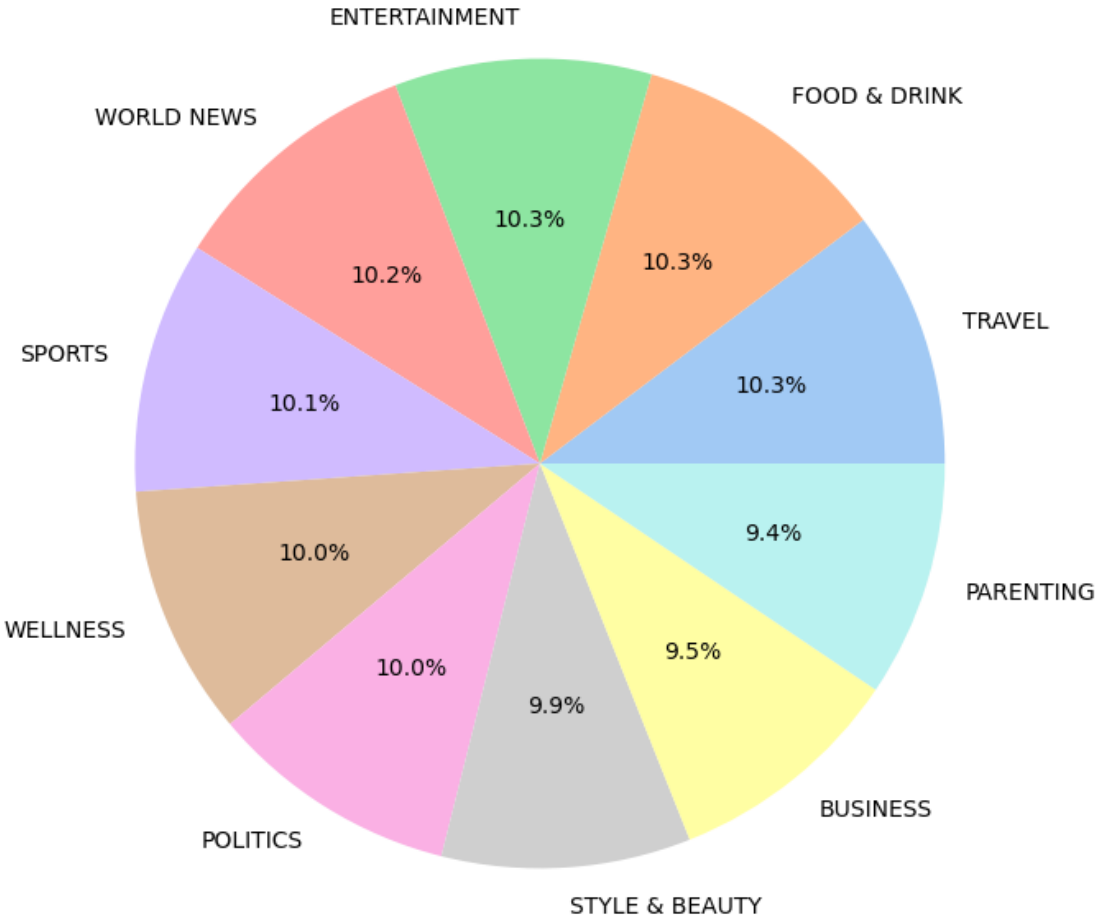
[444]:
```
# plotting Bar Plot
plt.figure(figsize=(10, 5))
sns.barplot(x=category_counts.index, y=category_counts.values, palette="Reds")
plt.xticks(rotation=45)
plt.xlabel("Category")
plt.ylabel("Article number's")
plt.show()
# plotting Pie Chart
plt.figure(figsize=(8, 8))
plt.pie(category_counts, labels=category_counts.index, autopct="%1.1f%%",␣
 ↪colors=sns.color_palette("pastel"))
plt.show()
```

```
C:\Users\numan\AppData\Local\Temp\ipykernel_17456\2194347530.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=category_counts.index, y=category_counts.values, palette="Reds")
```





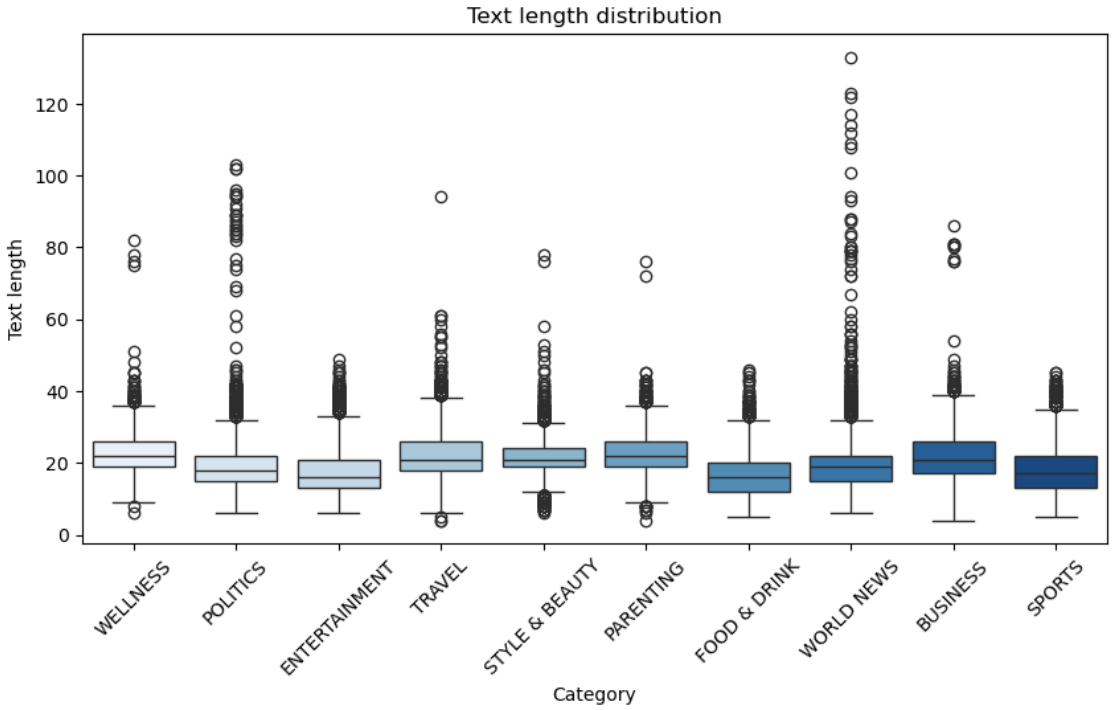
[437]:
```
# Create a new column for text length
df["text_length"] = df["combined_text"].apply(lambda x: len(x.split()))

# Boxplot to compare text lengths per category
plt.figure(figsize=(10, 5))
sns.boxplot(x="category", y="text_length", data=df, palette="Blues")
plt.xticks(rotation=45)
plt.xlabel("Category")
plt.ylabel("Text length")
plt.title("Text length distribution")
plt.show()
```

```
C:\Users\numan\AppData\Local\Temp\ipykernel_17456\3078688946.py:6:
FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

```
  sns.boxplot(x="category", y="text_length", data=df, palette="Blues")
```



0.1 Modelling

[310]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
```

[257]:
```
# Encoding the category labels into numbers
le = LabelEncoder()
df["encoded_cat"] = le.fit_transform(df["category"])
```

[261]:
```
# splitting into train and testing
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, df["encoded_cat"],␣
 ↪test_size = 0.2, random_state = 42)
```

[265]:
```
X_train.shape, X_test.shape
```

[265]:
```
((37865, 5000), (9467, 5000))
```

[267]:
```
# Initialize models
lr = LogisticRegression(max_iter = 1000)
nb = MultinomialNB()
```

[440]:
```
# Training models
lr.fit(X_train, y_train)
nb.fit(X_train, y_train)
```

[440]:
```
MultinomialNB()
```

[271]:
```
# Predicting
y_pred_lr = lr.predict(X_test)
y_pred_nb = nb.predict(X_test)
```

[283]:
```
# Tune Logistic Regression
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid_log = {"C": [0.1, 1, 10, 100]}

# Perform GridSearchCV
grid_log = GridSearchCV(LogisticRegression(max_iter=1000), param_grid_log,␣
 ↪cv=5, scoring="accuracy", n_jobs=-1)
grid_log.fit(X_train, y_train)

# Best parameters
print("Best Parameters (Logistic Regression):", grid_log.best_params_)

# Train optimized model
best_log = grid_log.best_estimator_
```

```
Best Parameters (Logistic Regression): {'C': 1}
```

[295]:
```
# Tune Naive Bayes

# Define parameter grid
param_grid_nb = {"alpha": [0.1, 0.5, 1, 5, 10]}

# Perform GridSearchCV
grid_nb = GridSearchCV(MultinomialNB(), param_grid_nb, cv=5,␣
 ↪scoring="accuracy", n_jobs=-1)
grid_nb.fit(X_train, y_train)

# Best parameters
print("Best Parameters (Naive Bayes):", grid_nb.best_params_)

# Train optimized model
best_nb = grid_nb.best_estimator_
```

```
Best Parameters (Naive Bayes): {'alpha': 1}
```

[312]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
```

[326]:
```
# Define the number of folds

cv_folds = 5
# Logistic Regression Cross-Validation

log_cv_scores = cross_val_score(best_log, X_train, y_train, cv=cv_folds,␣
 ↪scoring="accuracy")
print("Logistic Regression Cross-Validation Accuracy:", log_cv_scores.mean())

# Naive Bayes Cross-Validation
nb_cv_scores = cross_val_score(best_nb, X_train, y_train, cv=cv_folds,␣
 ↪scoring="accuracy")
print("Naive Bayes Cross-Validation Accuracy:", nb_cv_scores.mean())
```

```
Logistic Regression Cross-Validation Accuracy: 0.7922355737488446
Naive Bayes Cross-Validation Accuracy: 0.7796645979136405
```

[342]:
```
# Function to evaluate and print model performance

def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)
    print(f"\n Model: {model_name}")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Evaluate Logistic Regression
evaluate_model(best_log, X_test, y_test, "Logistic Regression")

# Evaluate Naive Bayes
evaluate_model(best_nb, X_test, y_test, "Naive Bayes")
```

```
 Model: Logistic Regression
Accuracy: 0.8055350163726629
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.76      0.76       908
           1       0.77      0.79      0.78       923
           2       0.85      0.85      0.85       987
           3       0.79      0.77      0.78       900
           4       0.78      0.74      0.76       955
```

```
           5       0.88      0.90      0.89       968
           6       0.87      0.85      0.86       903
           7       0.81      0.80      0.81       958
           8       0.73      0.79      0.76       969
           9       0.82      0.81      0.81       996

    accuracy                           0.81      9467
   macro avg       0.81      0.81      0.81      9467
weighted avg       0.81      0.81      0.81      9467

Confusion Matrix:
 [[689  13  22  17  50  14   6  22  44  31]
 [ 14 730   7  33  18  32  30  24  22  13]
 [ 13  12 838  17   3   9  20  37  33   5]
 [ 19  27  16 697  12  10  21  14  79   5]
 [ 62  24   3  20 708  16   6  18  25  73]
 [  3  38   1  11  14 867   6  10   9   9]
 [ 13  46  10  16   7   3 764  15  26   3]
 [ 21  26  43  12  14  12  10 767  26  27]
 [ 35  19  42  55  13  13  10  12 762   8]
 [ 40  17   0   9  65  14   3  25  19 804]]

 Model: Naive Bayes
Accuracy: 0.7913805851906623
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.69      0.73       908
           1       0.79      0.76      0.77       923
           2       0.83      0.86      0.84       987
           3       0.72      0.78      0.74       900
           4       0.79      0.74      0.76       955
           5       0.88      0.85      0.86       968
           6       0.85      0.82      0.83       903
           7       0.79      0.82      0.81       958
           8       0.71      0.77      0.74       969
           9       0.81      0.82      0.81       996

    accuracy                           0.79      9467
   macro avg       0.79      0.79      0.79      9467
weighted avg       0.79      0.79      0.79      9467

Confusion Matrix:
 [[630  16  22  34  54  11  12  27  62  40]
 [ 12 704  13  30  17  33  50  22  28  14]
 [ 13   5 844  20   2  10  14  49  30   0]
 [ 13  25  20 698  17   6  14  16  90   1]
 [ 56  25   4  21 708  15   8  16  23  79]
```

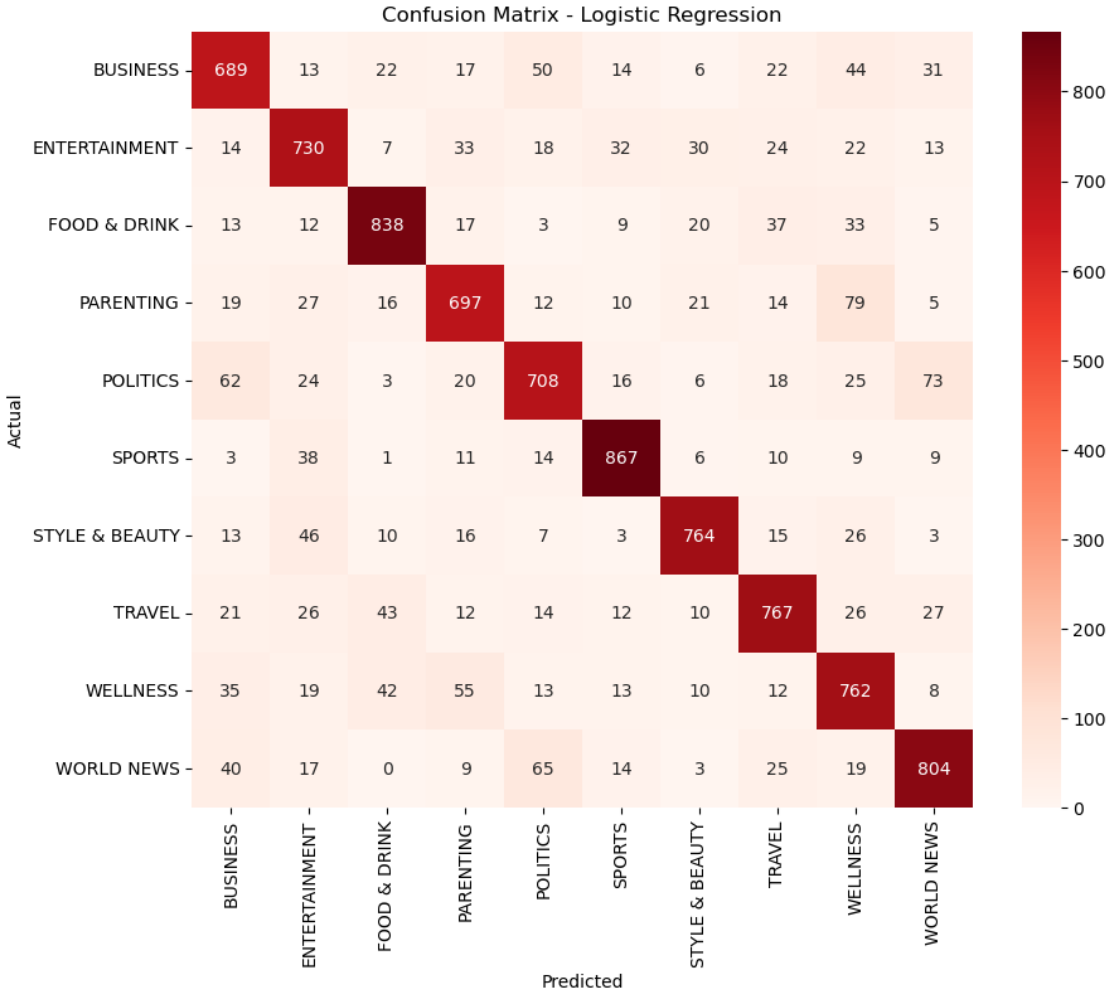
```
[  6  43   5  17  18 823  10  12  16  18]
[ 15  40  21  27   6   2 737  24  26   5]
[ 17  16  42  25   7  17  10 787  14  23]
[ 24   7  51  90  12   8   8  14 748   7]
[ 42  14   1  14  59  12   3  25  13 813]]
```

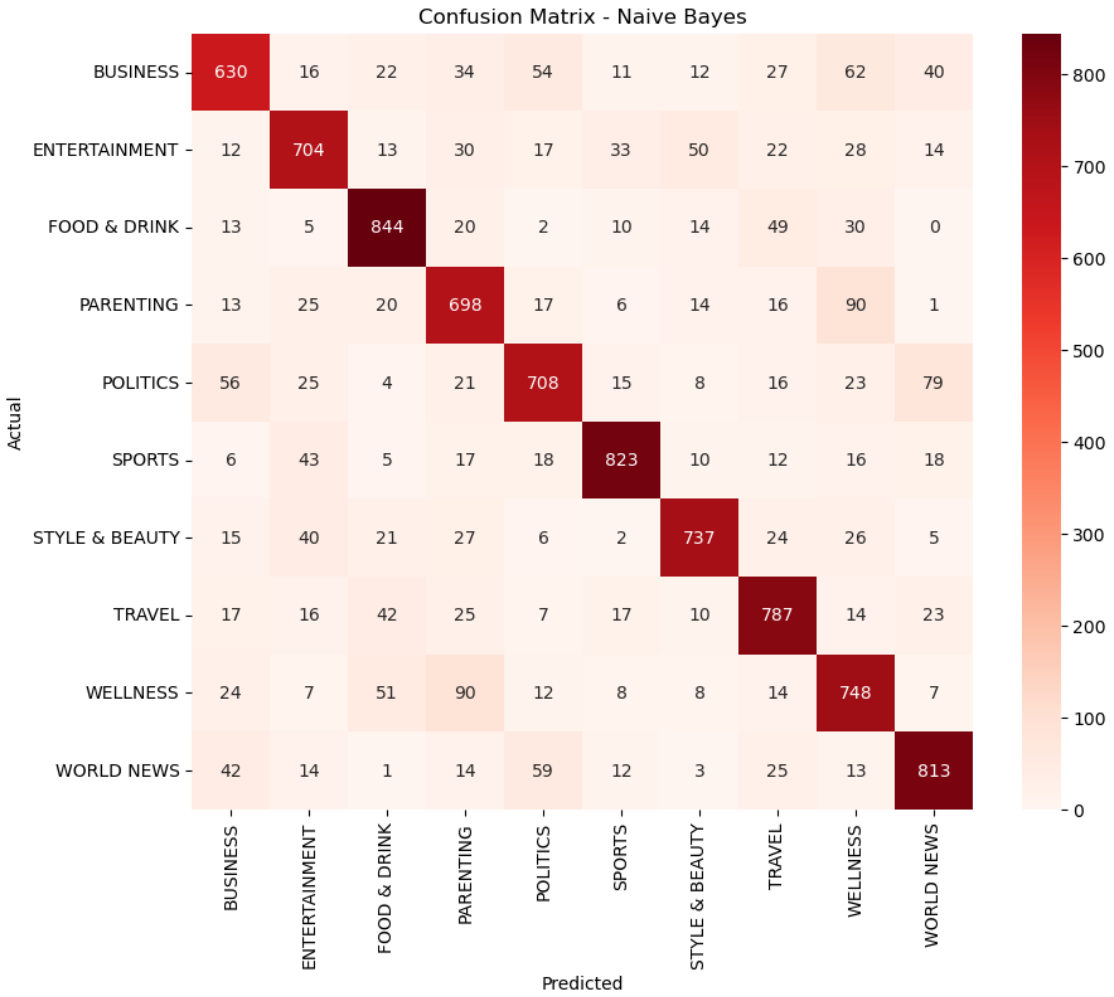
[386]:
```
# Function to plot confusion matrix

def plot_confusion_matrix(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10,8))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Reds", xticklabels=le.classes_,␣
 ↪yticklabels=le.classes_)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()

# Plot for Logistic Regression
plot_confusion_matrix(best_log, X_test, y_test, "Logistic Regression")

# Plot for Naive Bayes
plot_confusion_matrix(best_nb, X_test, y_test, "Naive Bayes")
```


Confusion Matrix - Logistic Regression
BUSINESS 689 13 22 17 50 14 6 22 44 31
ENTERTAINMENT 14 730 7 33 18 32 30 24 22 13
FOOD & DRINK 13 12 838 17 3 9 20 37 33 5
PARENTING 19 27 16 697 12 10 21 14 79 5
POLITICS 62 24 3 20 708 16 6 18 25 73
SPORTS 3 38 1 11 14 867 6 10 9 9
STYLE & BEAUTY 13 46 10 16 7 3 764 15 26 3
TRAVEL 21 26 43 12 14 12 10 767 26 27
WELLNESS 35 19 42 55 13 13 10 12 762 8
WORLD NEWS 40 17 0 9 65 14 3 25 19 804
Actual
BUSINESS
ENTERTAINMENT
FOOD & DRINK
PARENTING
POLITICS
SPORTS
STYLE & BEAUTY
TRAVEL
WELLNESS
WORLD NEWS
Predicted
800
700
600
500
400
300
200
100
0



[417]:
```python
# Function to get model evaluation metrics
def get_model_metrics(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    f1_score = report["weighted avg"]["f1-score"]
    precision = report["weighted avg"]["precision"]
    recall = report["weighted avg"]["recall"]
    return [accuracy, precision, recall, f1_score]

# Store results in a DataFrame
models = {
"Logistic Regression": best_log,
"Naive Bayes": best_nb
}
```

```
# Store results in a dictionary
results = {name: get_model_metrics(model, X_test, y_test) for name, model in␣
 ↪models.items()}
df_results = pd.DataFrame(results, index=["Accuracy", "Precision",␣
 ↪"Recall","F1-Score"]).T # .T --> transposes the dataframe

# Display results
print("\n Model Comparison Table:-------------------------------------")
print(df_results)
```

```
 Model Comparison Table:-------------------------------------
                     Accuracy  Precision    Recall  F1-Score
Logistic Regression  0.805535   0.806218  0.805535  0.805644
Naive Bayes          0.791381   0.792743  0.791381  0.791478
```

[446]:
```
# Select the best model based on highest F1-score

best_model = df_results["F1-Score"].idxmax()
print(f"\n The best model for News_Classification: {best_model}")
```

```
 The best model for News_Classification: Logistic Regression
```

0.2 The Report

https://docs.google.com/document/d/1oMU_DmRw8DlozbNd0QOsoCc-MNsdKMtu/edit?usp=sharing&ouid=101091923509442736382&rtpof=true&sd=true

the video explanation link

https://drive.google.com/file/d/1e_HFK0Hbxj-B94CGyiuhuNRbZwUg25_y/view?usp=sharing