1. Data Collection and Preprocessing (5 Marks):

- Collect a dataset of labeled news articles (sports, politics, technology etc).
- Cleanand preprocess the text data.
- Handle missing data, if any, and ensure the text is ready for feature extraction.

```
# Install necessary packages
!pip install nltk --quiet

# Import libraries
import pandas as pd
import numpy as np
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re

# Download NLTK data
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt 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!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

True

!pip install --upgrade --no-cache-dir numpy==1.23.5 scipy==1.10.1
pandas==1.5.3 scikit-learn==1.2.2 gensim==4.3.1

Collecting numpy==1.23.5
  Downloading numpy-1.23.5-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.3 kB)
Collecting scipy==1.10.1
  Downloading scipy-1.10.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (58 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.9/58.9 kB 3.6 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting scikit-learn==1.2.2
  Downloading scikit_learn-1.2.2-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting gensim==4.3.1
  Downloading gensim-4.3.1-cp311-cp311-
```

```
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.4 kB)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.11/dist-packages (from pandas==1.5.3)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas==1.5.3) (2025.2)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn==1.2.2)
(1.5.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn==1.2.2)
(3.6.0)
Requirement already satisfied: smart-open>=1.8.1 in
/usr/local/lib/python3.11/dist-packages (from gensim==4.3.1) (7.1.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.1-
>pandas==1.5.3) (1.17.0)
Requirement already satisfied: wrapt in
/usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1-
>gensim==4.3.1) (1.17.2)
Downloading numpy-1.23.5-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.1/17.1 MB 250.3 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (34.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.1/34.1 MB 264.4 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.0/12.0 MB 264.9 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (9.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.6/9.6 MB 250.7 MB/s eta
0:00:00
-4.3.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(26.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.6/26.6 MB 159.7 MB/s eta
0:00:00
py, scipy, pandas, scikit-learn, gensim
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: scipy
    Found existing installation: scipy 1.15.3
    Uninstalling scipy-1.15.3:
      Successfully uninstalled scipy-1.15.3
  Attempting uninstall: pandas
    Found existing installation: pandas 2.2.2
    Uninstalling pandas-2.2.2:
```

```
      Successfully uninstalled pandas-2.2.2
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.6.1
    Uninstalling scikit-learn-1.6.1:
      Successfully uninstalled scikit-learn-1.6.1
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 1.5.3
which is incompatible.
cvxpy 1.6.5 requires scipy>=1.11.0, but you have scipy 1.10.1 which is
incompatible.
jaxlib 0.5.1 requires numpy>=1.25, but you have numpy 1.23.5 which is
incompatible.
jaxlib 0.5.1 requires scipy>=1.11.1, but you have scipy 1.10.1 which
is incompatible.
jax 0.5.2 requires numpy>=1.25, but you have numpy 1.23.5 which is
incompatible.
jax 0.5.2 requires scipy>=1.11.1, but you have scipy 1.10.1 which is
incompatible.
mizani 0.13.5 requires pandas>=2.2.0, but you have pandas 1.5.3 which
is incompatible.
xarray 2025.3.1 requires numpy>=1.24, but you have numpy 1.23.5 which
is incompatible.
xarray 2025.3.1 requires pandas>=2.1, but you have pandas 1.5.3 which
is incompatible.
mlxtend 0.23.4 requires scikit-learn>=1.3.1, but you have scikit-learn
1.2.2 which is incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy
1.23.5 which is incompatible.
bigframes 2.4.0 requires numpy>=1.24.0, but you have numpy 1.23.5
which is incompatible.
blosc2 3.3.2 requires numpy>=1.26, but you have numpy 1.23.5 which is
incompatible.
chex 0.1.89 requires numpy>=1.24.1, but you have numpy 1.23.5 which is
incompatible.
dask-expr 1.1.21 requires pandas>=2, but you have pandas 1.5.3 which
is incompatible.
treescope 0.1.9 requires numpy>=1.25.2, but you have numpy 1.23.5
which is incompatible.
scikit-image 0.25.2 requires numpy>=1.24, but you have numpy 1.23.5
which is incompatible.
scikit-image 0.25.2 requires scipy>=1.11.4, but you have scipy 1.10.1
which is incompatible.
cudf-cu12 25.2.1 requires pandas<2.2.4dev0,>=2.0, but you have pandas
1.5.3 which is incompatible.
pymc 5.22.0 requires numpy>=1.25.0, but you have numpy 1.23.5 which is
incompatible.
dask-cudf-cu12 25.2.2 requires pandas<2.2.4dev0,>=2.0, but you have
```

```
pandas 1.5.3 which is incompatible.
imbalanced-learn 0.13.0 requires numpy<3,>=1.24.3, but you have numpy
1.23.5 which is incompatible.
imbalanced-learn 0.13.0 requires scikit-learn<2,>=1.3.2, but you have
scikit-learn 1.2.2 which is incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.23.5
which is incompatible.
albumentations 2.0.6 requires numpy>=1.24.4, but you have numpy 1.23.5
which is incompatible.
plotnine 0.14.5 requires pandas>=2.2.0, but you have pandas 1.5.3
which is incompatible.
tsfresh 0.21.0 requires scipy>=1.14.0; python_version >= "3.10", but
you have scipy 1.10.1 which is incompatible.
albucore 0.0.24 requires numpy>=1.24.4, but you have numpy 1.23.5
which is incompatible.
db-dtypes 1.4.3 requires numpy>=1.24.0, but you have numpy 1.23.5
which is incompatible.
Successfully installed gensim-4.3.1 numpy-1.23.5 pandas-1.5.3 scikit-
learn-1.2.2 scipy-1.10.1
```

{"id":"8b6f60c41f474564b234b3e66fbb0964","pip_warning":{"packages":
["numpy","pandas","sklearn"]}}

```
# Upload the dataset.
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving data_news.csv to data_news.csv
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv('data_news.csv')

# Display basic info

print("First 5 rows:")
display(df.head())

print("\nDataset Info:")
print(df.info())

print("\nClass Distribution:")
print(df['category'].value_counts())
```

```
First 5 rows:
```

{"summary":"{\n \"name\": \"print(df['category']\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"category\",\n

```
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"WELLNESS\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"headline\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"Talking to Yourself: Crazy or Crazy Helpful?\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"links\",\n      \"properties\":
{\n        \"dtype\": \"string\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n
\"https://www.huffingtonpost.com/entry/talking-to-yourself-
crazy_us_5b9def86e4b03a1dcc8f142c\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"short_description\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"Think
of talking to yourself as a tool to coach yourself through a
challenge, or to narrate your own experiences to yourself. In any
case, treat yourself with respect and you just may find you enjoy your
own company.\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"keywords\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"talking-to-yourself-crazy\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
Dataset 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
None

Class Distribution:
category
WELLNESS         5000
POLITICS         5000
ENTERTAINMENT    5000
TRAVEL           5000
```

```
STYLE & BEAUTY    5000
PARENTING         5000
FOOD & DRINK      5000
WORLD NEWS        5000
BUSINESS          5000
SPORTS            5000
Name: count, dtype: int64
```

```
# Check for missing values
print("\nMissing values:")
print(df.isnull().sum())
```

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

```
import nltk
import string
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download required NLTK data
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')

# Create a combined text column
df['text'] = df['headline'] + ' ' + df['short_description']

# Clean text function
def clean_text(text):
    text = text.lower()  # lowercase
    text = re.sub(r'\d+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
# remove punctuation
    text = re.sub(r'\s+', ' ', text).strip()  # remove extra spaces
    return text

# Apply basic cleaning
df['clean_text'] = df['text'].apply(clean_text)

# Remove stopwords
stop_words = set(stopwords.words('english'))

def remove_stopwords(text):
```

```
    tokens = word_tokenize(text)
    filtered = [word for word in tokens if word not in stop_words]
    return " ".join(filtered)

df['clean_text'] = df['clean_text'].apply(remove_stopwords)

# Preview cleaned data
print("\nCleaned Data Sample:")
print(df[['text', 'clean_text']].head())
```

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


Cleaned Data Sample:
                                                text  \
0  143 Miles in 35 Days: Lessons Learned Resting ...
1  Talking to Yourself: Crazy or Crazy Helpful? T...
2  Crenezumab: Trial Will Gauge Whether Alzheimer...
3  Oh, What a Difference She Made If you want to ...
4  Green Superfoods First, the bad news: Soda bre...

                                          clean_text
0  miles days lessons learned resting part traini...
1  talking crazy crazy helpful think talking tool...
2  crenezumab trial gauge whether alzheimers drug...
3  oh difference made want busy keep trying perfe...
4  green superfoods first bad news soda bread cor...
```

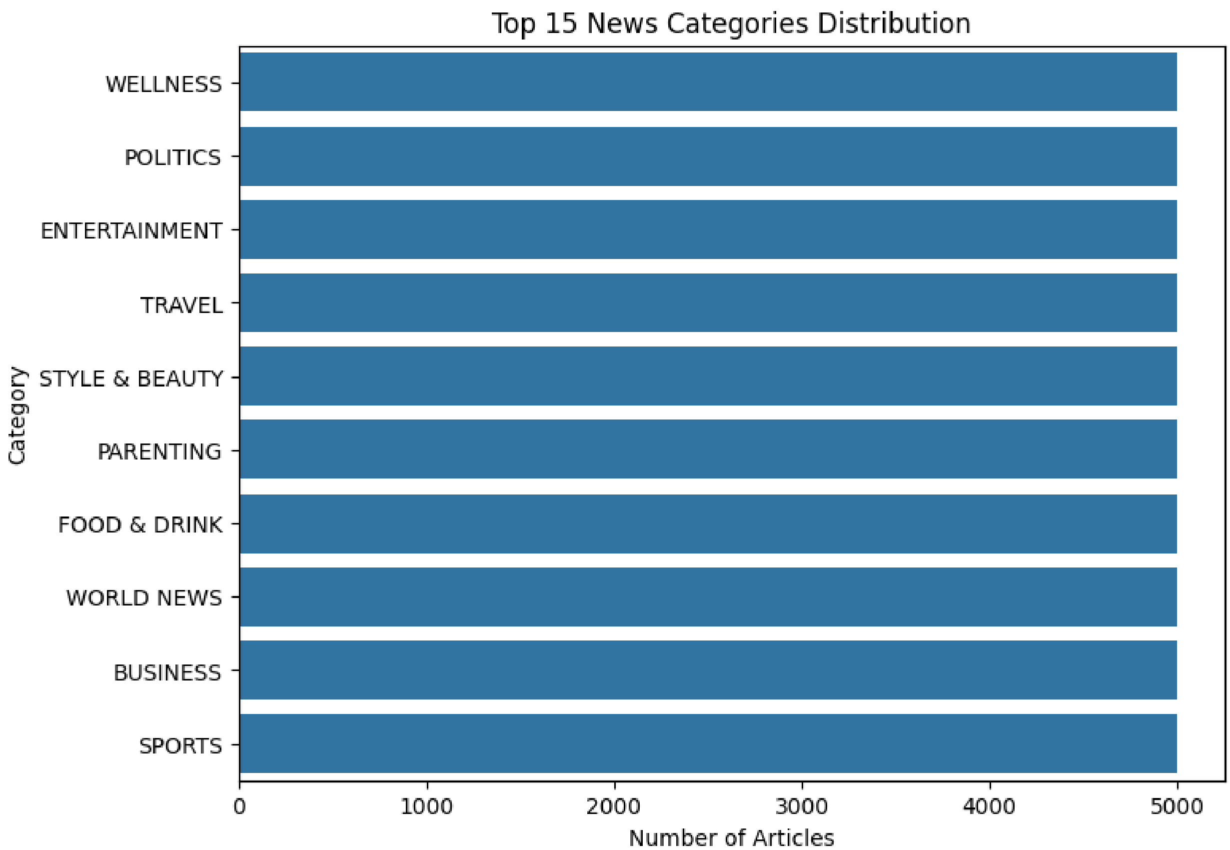
1. Feature Extraction (10 Marks):

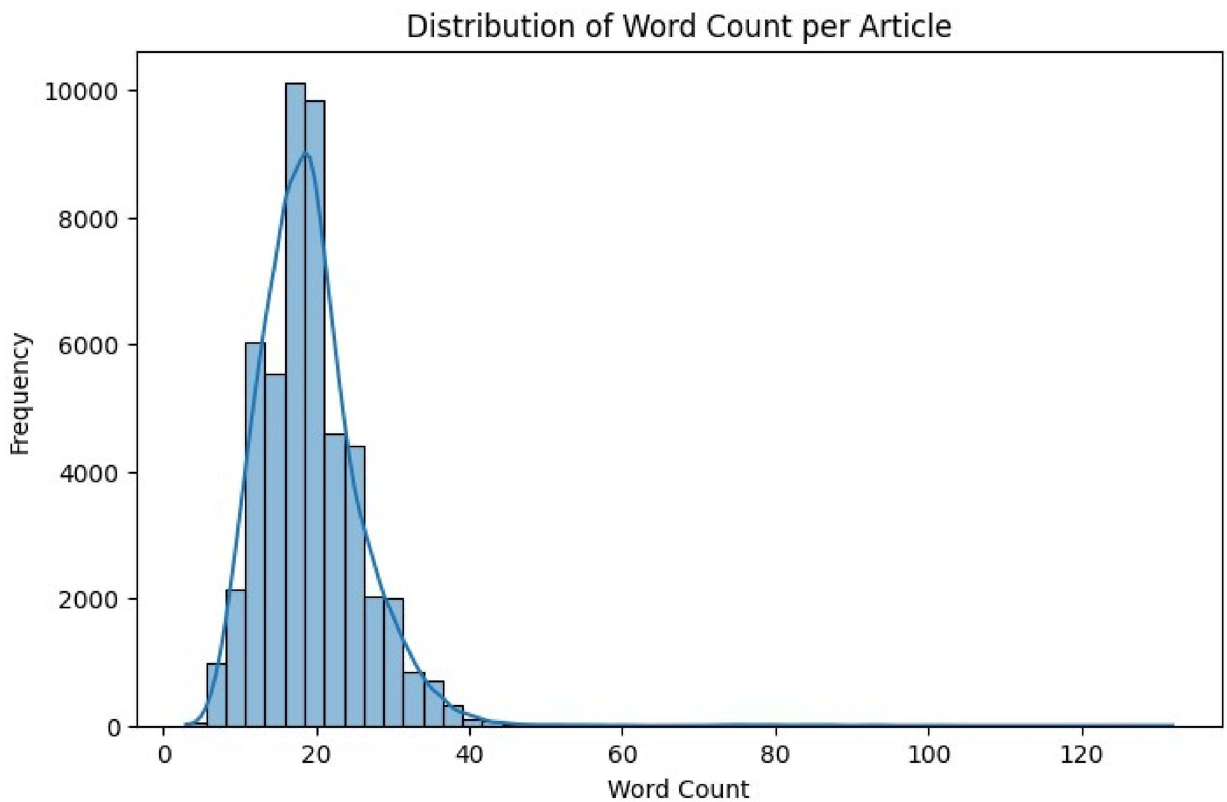
● Use methods like TF-IDF, word embeddings (e.g., Word2Vec, GloVe), or bag-of-words to convert text data into numerical features.

● Perform exploratory data analysis (EDA) to understand the distribution of different categories.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Category distribution
plt.figure(figsize=(8,6))
sns.countplot(data=df, y='category',
order=df['category'].value_counts().index[:15])
plt.title("Top 15 News Categories Distribution")
plt.xlabel("Number of Articles")
plt.ylabel("Category")
```

```
plt.show()

# Word count per article
df['word_count'] = df['clean_text'].apply(lambda x: len(x.split()))
plt.figure(figsize=(8,5))
sns.histplot(df['word_count'], bins=50, kde=True)
plt.title("Distribution of Word Count per Article")
plt.xlabel("Word Count")
plt.ylabel("Frequency")
plt.show()
```






```
from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF Vectorization
tfidf = TfidfVectorizer(max_features=2000)
X = tfidf.fit_transform(df['clean_text'])

# Target variable
y = df['category']

# See a few TF-IDF features
print(tfidf.get_feature_names_out()[:20])

['ability' 'able' 'absolutely' 'abuse' 'academy' 'access' 'according'
 'accused' 'achieve' 'across' 'act' 'action' 'actions' 'active'
 'activities' 'activity' 'actor' 'actress' 'actually' 'ad']
```

1. Model Development and Training (20 Marks):

● Build classification models using algorithms like Logistic Regression, Naive Bayes, Support Vector Machines (SVM).

● Train the models on the preprocessed text data, tuning hyperparameters as necessary.

● Use cross-validation to ensure robust evaluation of model performance.

```python
from sklearn.model_selection import train_test_split, cross_val_score,
GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, accuracy_score

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# 1. Logistic Regression

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred_lr = log_reg.predict(X_test)
print("\nLogistic Regression Results:")
print(classification_report(y_test, y_pred_lr))
print("Cross-Validation Accuracy:", cross_val_score(log_reg, X, y,
cv=5).mean())


# 2. Naive Bayes

nb = MultinomialNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
print("\nNaive Bayes Results:")
print(classification_report(y_test, y_pred_nb))
print("Cross-Validation Accuracy:", cross_val_score(nb, X, y,
cv=5).mean())


# 3. Support Vector Machine (LinearSVC)

svm = LinearSVC()
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
print("\nSupport Vector Machine Results:")
print(classification_report(y_test, y_pred_svm))
print("Cross-Validation Accuracy:", cross_val_score(svm, X, y,
cv=5).mean())
```

```
Logistic Regression Results:
                precision    recall  f1-score   support

     BUSINESS       0.67      0.71      0.68       955
ENTERTAINMENT       0.68      0.69      0.68       985
 FOOD & DRINK       0.81      0.77      0.79      1021
```

```
     PARENTING       0.75      0.74      0.75      1030
      POLITICS       0.72      0.68      0.70      1034
        SPORTS       0.78      0.79      0.78       995
STYLE & BEAUTY       0.83      0.79      0.81       986
        TRAVEL       0.76      0.73      0.74      1008
      WELLNESS       0.66      0.71      0.68      1009
    WORLD NEWS       0.72      0.75      0.73       977

      accuracy                           0.74     10000
     macro avg       0.74      0.74      0.74     10000
  weighted avg       0.74      0.74      0.74     10000

Cross-Validation Accuracy: 0.7249599999999999

Naive Bayes Results:
                precision    recall  f1-score   support

      BUSINESS       0.66      0.65      0.66       955
 ENTERTAINMENT       0.70      0.66      0.68       985
  FOOD & DRINK       0.79      0.78      0.78      1021
     PARENTING       0.66      0.72      0.69      1030
      POLITICS       0.72      0.67      0.69      1034
        SPORTS       0.79      0.76      0.78       995
STYLE & BEAUTY       0.79      0.79      0.79       986
        TRAVEL       0.74      0.74      0.74      1008
      WELLNESS       0.65      0.70      0.68      1009
    WORLD NEWS       0.72      0.75      0.73       977

      accuracy                           0.72     10000
     macro avg       0.72      0.72      0.72     10000
  weighted avg       0.72      0.72      0.72     10000

Cross-Validation Accuracy: 0.7163999999999999

Support Vector Machine Results:
                precision    recall  f1-score   support

      BUSINESS       0.68      0.71      0.70       955
 ENTERTAINMENT       0.71      0.67      0.69       985
  FOOD & DRINK       0.80      0.79      0.79      1021
     PARENTING       0.75      0.74      0.75      1030
      POLITICS       0.73      0.66      0.69      1034
        SPORTS       0.76      0.82      0.79       995
STYLE & BEAUTY       0.80      0.81      0.81       986
        TRAVEL       0.75      0.72      0.73      1008
      WELLNESS       0.66      0.69      0.68      1009
    WORLD NEWS       0.72      0.74      0.73       977

      accuracy                           0.74     10000
     macro avg       0.74      0.74      0.74     10000
```

```
 weighted avg       0.74      0.74      0.74     10000

Cross-Validation Accuracy: 0.71678
```

1. Model Evaluation (5 Marks):

- Evaluate the models using appropriate metrics.
- Comparethe performance of different models and select the best one for classification.

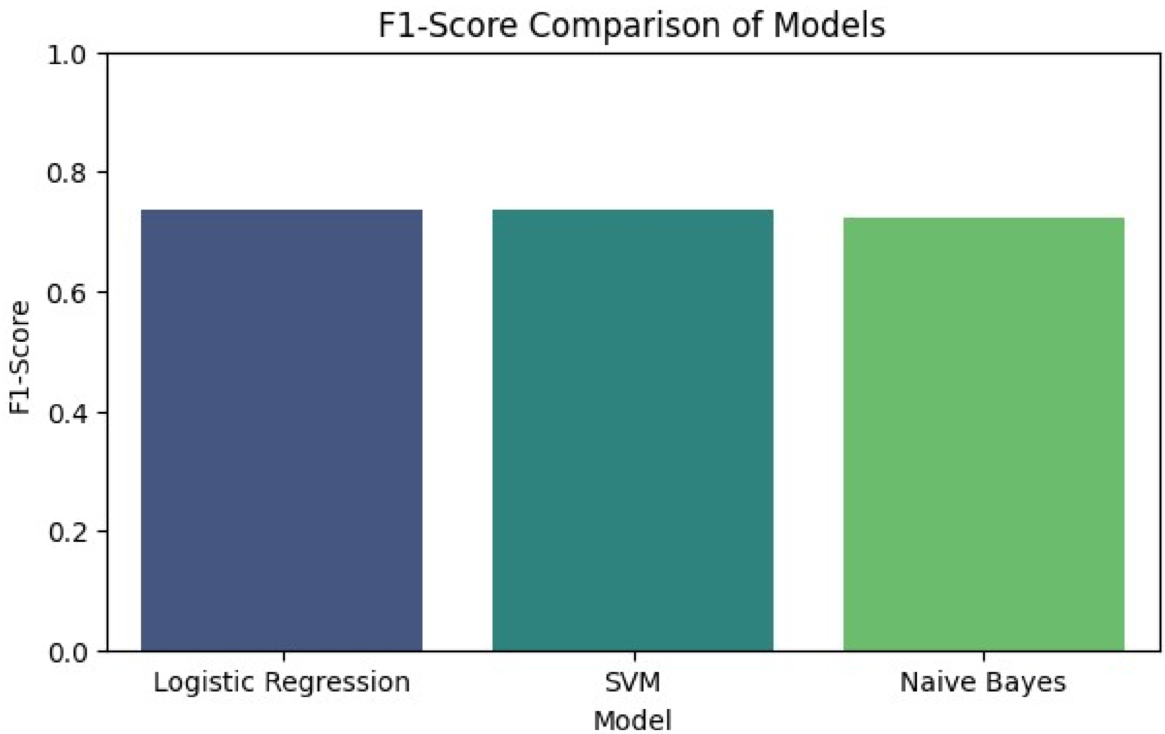
```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# Define a function to compute all metrics
def evaluate_model(name, y_true, y_pred):
    return {
        'Model': name,
        'Accuracy': accuracy_score(y_true, y_pred),
        'Precision': precision_score(y_true, y_pred,
average='weighted'),
        'Recall': recall_score(y_true, y_pred, average='weighted'),
        'F1-Score': f1_score(y_true, y_pred, average='weighted')
    }

# Evaluate all three models
results = []
results.append(evaluate_model("Logistic Regression", y_test,
y_pred_lr))
results.append(evaluate_model("Naive Bayes", y_test, y_pred_nb))
results.append(evaluate_model("SVM", y_test, y_pred_svm))

# Create a DataFrame for comparison
results_df = pd.DataFrame(results)
results_df.sort_values(by='F1-Score', ascending=False, inplace=True)

# Display comparison
print("\nModel Comparison:")
display(results_df)


Model Comparison:
```


```
{"summary":"{\n  \"name\": \"results_df\",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Logistic Regression\",\n          \"SVM\",\n          \"Naive
Bayes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Accuracy\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 0.008150460109711568,\n        \"min\":
0.7216,\n        \"max\": 0.7361,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.7353,\n          0.7361,\n
0.7216\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Precision\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.007976769671077416,\n        \"min\":
0.7229691512243306,\n        \"max\": 0.7372241836760877,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.7372241836760877,\n          0.7363000617231678,\n
0.7229691512243306\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Recall\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.008150460109711568,\n        \"min\": 0.7216,\n
\"max\": 0.7361,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.7353,\n          0.7361,\n
0.7216\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"F1-
Score\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.008093471878327404,\n        \"min\": 0.7218266402553863,\n
\"max\": 0.7358876389109646,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.7358876389109646,\n
0.7358018569094386,\n          0.7218266402553863\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"results_df"}
```

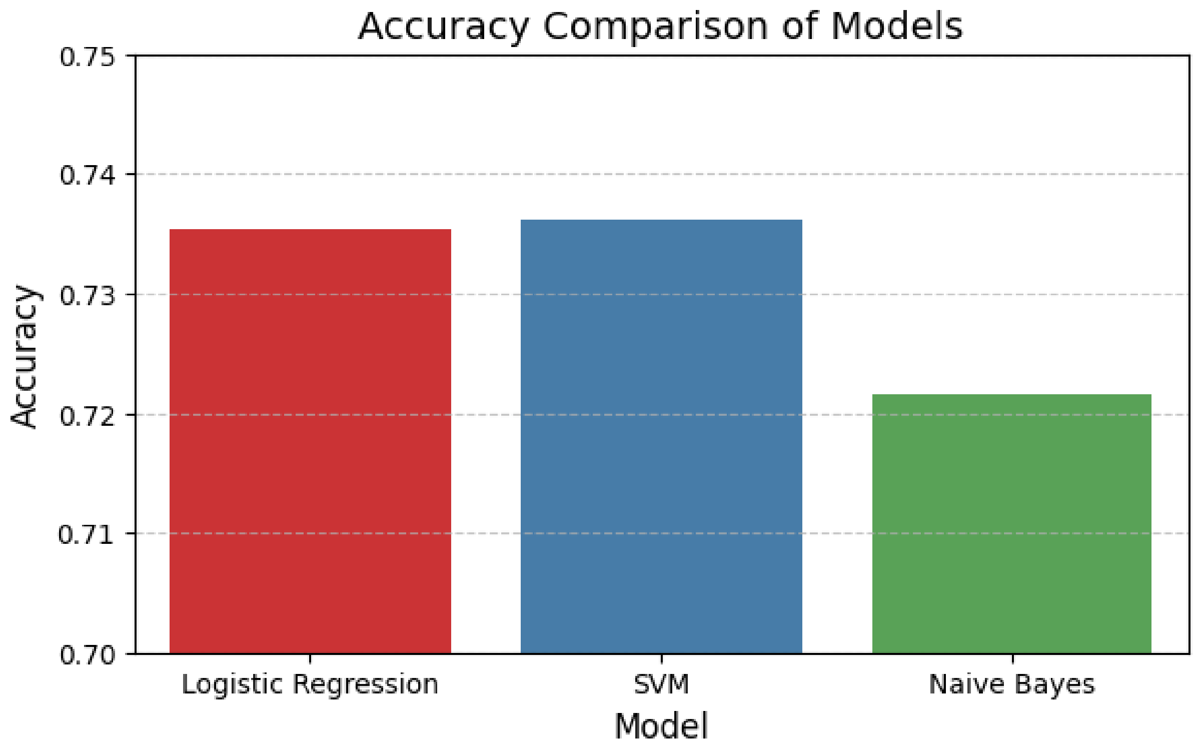

```python
# Let's visualize the F1-score comparisons

import matplotlib.pyplot as plt

# Plot F1-score comparison
plt.figure(figsize=(7,4))
sns.barplot(data=results_df, x='Model', y='F1-Score',
palette='viridis')
plt.title('F1-Score Comparison of Models')
plt.ylabel('F1-Score')
plt.xlabel('Model')
plt.ylim(0, 1)
plt.show()
```

```
<ipython-input-21-da22607d735e>:7: 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(data=results_df, x='Model', y='F1-Score',
palette='viridis')
```




```
# Let's visualize the accuracy comparison

import matplotlib.pyplot as plt
import seaborn as sns

# Accuracy comparison plot
plt.figure(figsize=(7,4))
sns.barplot(data=results_df, x='Model', y='Accuracy', palette='Set1')
plt.title('Accuracy Comparison of Models', fontsize=14)
plt.xlabel('Model', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.ylim(0.70, 0.75)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
<ipython-input-23-dcdf56c1ebfb>:8: 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(data=results_df, x='Model', y='Accuracy',
palette='Set1')
```



*** Imp Note: Best Model: SVM (Support Vector Machine)***

It has the highest accuracy (0.7361)

Comparable F1-score to Logistic Regression

Consistent balance between precision, recall, and F1-score

Performs well with high-dimensional TF-IDF vectors

Among all models, SVM emerged as the top performer, achieving the highest accuracy and maintaining a strong balance across all metrics. While Logistic Regression was a close second, SVM slightly edged it out in terms of consistency.

Therefore, we will select SVM as our final model for news category classification.