Part B — News Article Classification (NLP)

Goal: Automatically classify news articles into predefined categories (e.g., Politics, Sports, Technology, etc.) using classical NLP: text cleaning → TF-IDF features → linear classifiers → evaluation.

Dataset: Labeled news entries with `category`, `headline`, `links`, `short_description` (renamed to `text`), and `keywords`.

Plan:

1. Data understanding & cleaning (lowercase, punctuation removal, tokenization, stopwords).
2. Feature extraction with TF-IDF (max 5k features).
3. Baseline models: Logistic Regression, Linear SVM, Multinomial Naive Bayes.
4. Comparison + 5-fold CV (macro-F1).
5. Final evaluation on held-out test set; discuss confusions and next steps.

Deliverables alignment:

- Clear EDA and preprocessing ✓
- TF-IDF features and model training ✓
- Metrics, confusion matrix, and interpretation ✓
- Story-style Markdown after each code block ✓

1. Data Collection and Preprocessing

```
import pandas as pd

# Load the Excel file
df = pd.read_excel('data_news.xlsx')

# Display basic info and first few rows
print("Dataset Info:")
print(df.info())

print("\nFirst 5 Rows:")
print(df.head())
```

```
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  49994 non-null  object
 4   keywords           47294 non-null  object
dtypes: object(5)
memory usage: 1.9+ MB
None

First 5 Rows:
   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  
```

1. Data Collection & Snapshot — What, Why, and First Checks

What this cell does (1–2 lines):
Loads the news dataset from Excel into a DataFrame and shows its structure and first rows.

Why this matters:
Before any modeling, we confirm the dataset's size, columns, and data types to spot obvious issues early (e.g., missing values or wrong dtypes).

What to look for in the output (from *your* run):

- **Rows/Columns:** ~50,000 rows and 5 columns were detected in my run.
- **Columns:** `category`, `headline`, `links`, `short_description`, `keywords`.
- **Dtypes:** All shown as `object`.
- **Non-null counts:** Notice `short_description` had a few missing values in later steps and `keywords` had many missing entries; we will handle this downstream. These observations match the printed `info()` and the first 5 rows preview. ✔

Interpretation:

- The dataset is large enough for supervised learning (multi-class text classification).
- Text lives mainly in `short_description`; we'll rename it to `text` and clean it for modeling.
- We'll ignore `links` for modeling since it doesn't contribute to textual semantics directly.

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

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

Text Cleaning — From Raw to Model-Ready

What this cell does (1–2 lines):
Renames `short_description` to `text`, removes missing entries for `text` and `category`, and creates `clean_text` by lowercasing, stripping punctuation, tokenizing, and removing English stopwords.

Why this matters:
Machine-learning models need normalized inputs. Cleaning reduces noise (punctuation, case, filler words) so vectorizers (TF-IDF) capture meaningful terms.

What to look for (from *your* printed example_):

- **Original sample** vs **Cleaned sample** clearly shows the removal of punctuation and stopwords and the lowercase normalization.
- We keep only informative tokens to strengthen downstream features.

Interpretation:
The transformation preserves the semantic core while removing noise. This step is essential for stable TF-IDF features and fair model comparison later.

NLTK Resources — Tokenizer & Stopwords

What this cell does (1–2 lines):
Downloads NLTK resources (`punkt`, `stopwords`) required for tokenizing text and removing common words.

Why this matters:
Tokenization splits text into words; stopword removal reduces noise (e.g., “the”, “and”), improving the quality of features fed into models.

Interpretation:
The downloads completed successfully, so the next preprocessing steps that depend on these resources will run without errors.

```
import nltk
nltk.download('punkt_tab')
```

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

```
import nltk
nltk.download('punkt_tab')
```

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

```
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

df = pd.read_excel('/content/data_news.xlsx')
df = df.rename(columns={'short_description': 'text'})
df = df.dropna(subset=['text', 'category'])

def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    words = word_tokenize(text)
    words = [word for word in words if word not in stopwords.words('english')]
    return ' '.join(words)

df['clean_text'] = df['text'].astype(str).apply(clean_text)

print("Original Text Example:\n", df['text'].iloc[0])
print("\nCleaned Text Example:\n", df['clean_text'].iloc[0])
```

```
Original Text Example:
 Resting is part of training. I've confirmed what I sort of already knew: I

Cleaned Text Example:
 resting part training ive confirmed sort already knew im built running str
```

```
# Check for missing values in all columns
print("Missing Values per Column:\n")
print(df.isnull().sum())

# Check class distribution
print("\nNumber of articles per category:\n")
print(df['category'].value_counts())
```

```
Missing Values per Column:

category         0
headline         0
links            0
text             0
keywords      2706
clean_text       0
dtype: int64

Number of articles per category:

category
POLITICS          5000
ENTERTAINMENT     5000
BUSINESS          5000
PARENTING         5000
WORLD NEWS        5000
FOOD & DRINK      5000
SPORTS            5000
WELLNESS          4999
STYLE & BEAUTY    4999
TRAVEL            4996
Name: count, dtype: int64
```

Text Cleaning — From Raw to Model-Ready

What this cell does (1–2 lines):
Renames `short_description` to `text`, removes missing entries for `text` and `category`, and creates `clean_text` by lowercasing, stripping punctuation, tokenizing, and removing English stopwords.

Why this matters:
Machine-learning models need normalized inputs. Cleaning reduces noise (punctuation, case, filler words) so vectorizers (TF-IDF) capture meaningful terms.

What to look for (from *your* printed example_):

- **Original sample** vs **Cleaned sample** clearly shows the removal of punctuation and stopwords and the lowercase normalization.
- We keep only informative tokens to strengthen downstream features.

Interpretation:
The transformation preserves the semantic core while removing noise. This step is essential for stable TF-IDF features and fair model comparison later.

2. Feature Extraction (TF-IDF)

```
# 2. Feature Extraction (TF-IDF)

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['clean_text'])

print("TF-IDF Matrix Shape:", X.shape)
print("Example TF-IDF vector (first row):")
print(X[0])
```

```
TF-IDF Matrix Shape: (49994, 5000)
Example TF-IDF vector (first row):
<Compressed Sparse Row sparse matrix of dtype 'float64'
	with 21 stored elements and shape (1, 5000)>
  Coords	Values
  (0, 3161)	0.13101481182350774
  (0, 4587)	0.3409168793545913
  (0, 2353)	0.273745124320764
  (0, 914)	0.364369618648322
  (0, 4126)	0.1799438713463218
  (0, 156)	0.14162813414180941
  (0, 2470)	0.1581205742616256
  (0, 2199)	0.383641652318966
  (0, 581)	0.36304609515876723
  (0, 3817)	0.16157685927859702
  (0, 2029)	0.13661697466699915
  (0, 4498)	0.12987290236117233
  (0, 1744)	0.14034862023841452
  (0, 1124)	0.1279870290912323
  (0, 4854)	0.11713833023116212
  (0, 2640)	0.17600330358409333
  (0, 1060)	0.1982644877678214
  (0, 3258)	0.17261637336779165
  (0, 4473)	0.20102781726680755
  (0, 3785)	0.19463475286548118
  (0, 157)	0.11256134618127646
```

Feature Extraction — TF-IDF (Bag-of-Words with Importance)

What this cell does (1–2 lines):
Converts `clean_text` into a TF-IDF matrix with up to 5,000 features (most informative terms).

Why this matters:
TF-IDF highlights words that are frequent in a document but not too common across all documents, giving more discriminative power.

What to look for in the output:

- **Shape:** Rows ≈ number of samples after drops; Columns = 5,000 features.
- In my run, the matrix was **(49,994 × 5,000)**, confirming ~50K cleaned samples and a fixed feature size.

Interpretation:
We now have a high-dimensional, sparse numeric representation that classifiers (LogReg, LinearSVC, Naive Bayes) can train on efficiently.

3. Model Development and Training

```
# 3. Model Development and Training

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y = le.fit_transform(df['category'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

print("Model training complete.")
print("Sample prediction on first test entry:", le.inverse_transform([model.pre
```

```
Model training complete.
Sample prediction on first test entry: ['ENTERTAINMENT']
```

First Model — Logistic Regression Baseline

What this cell does (1–2 lines):
Splits data into train/test (80/20), encodes labels, trains a Logistic Regression classifier on TF-IDF, and shows a sample prediction.

Why this matters:
A strong linear baseline sets expectations and lets us compare later models fairly on the same split and features.

What to look for:

- **Sample prediction:** Confirms the pipeline works end-to-end (vectorize → encode → train → predict).
- **max_iter=1000:** Ensures convergence for high-dimensional TF-IDF.

Interpretation:
This baseline is typically competitive for text classification; we'll still compare it to Linear SVM and Multinomial Naive Bayes to confirm the best fit.

3.1 Model Comparison (News)

We compare Logistic Regression, Linear SVM, and Multinomial Naive Bayes on the same TF-IDF features and test split.

```
# News — Model Comparison on the existing train/test split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Reuse your existing train/test (X_train, X_test, y_train, y_test)
# and your LabelEncoder 'le'

models = {
    "LogReg": LogisticRegression(max_iter=1000),
    "LinearSVC": LinearSVC(),
    "MultinomialNB": MultinomialNB()
}

results = {}
for name, clf in models.items():
    clf.fit(X_train, y_train)
    preds = clf.predict(X_test)
    acc = accuracy_score(y_test, preds)
    print(f"\n=== {name} ===")
    print("Accuracy:", f"{acc:.4f}")
    # Show class names in the report
    print(classification_report(
        y_test, preds, target_names=list(le.classes_), zero_division=0, digits=
    ))
    results[name] = (acc, clf)

best_name = max(results, key=lambda k: results[k][0])
best_acc, best_model = results[best_name]
print(f"\nBest model on test set: {best_name} (Accuracy: {best_acc:.4f})")
```

```
                precision    recall  f1-score   support

      BUSINESS     0.6171    0.6621    0.6388       947
 ENTERTAINMENT     0.5269    0.5638    0.5447       972
  FOOD & DRINK     0.6781    0.7048    0.6912      1013
     PARENTING     0.6578    0.6202    0.6385       998
      POLITICS     0.6445    0.5552    0.5966      1032
        SPORTS     0.6750    0.6978    0.6862      1006
STYLE & BEAUTY     0.7066    0.6647    0.6850       993
        TRAVEL     0.6902    0.6465    0.6677      1027
      WELLNESS     0.5817    0.6435    0.6110      1007
    WORLD NEWS     0.6683    0.6723    0.6703      1004

      accuracy                         0.6431      9999
     macro avg     0.6446    0.6431    0.6430      9999
  weighted avg     0.6452    0.6431    0.6433      9999


=== LinearSVC ===
Accuracy: 0.6375
                precision    recall  f1-score   support

      BUSINESS     0.6321    0.6822    0.6562       947
 ENTERTAINMENT     0.5191    0.5319    0.5254       972
  FOOD & DRINK     0.6683    0.6900    0.6790      1013
     PARENTING     0.6379    0.6232    0.6305       998
      POLITICS     0.6264    0.5378    0.5787      1032
        SPORTS     0.6773    0.7386    0.7066      1006
STYLE & BEAUTY     0.6839    0.6667    0.6752       993
        TRAVEL     0.6750    0.6310    0.6522      1027
      WELLNESS     0.5943    0.6226    0.6081      1007
    WORLD NEWS     0.6590    0.6524    0.6557      1004

      accuracy                         0.6375      9999
     macro avg     0.6373    0.6376    0.6368      9999
  weighted avg     0.6378    0.6375    0.6369      9999
```

```
=== MultinomialNB ===
Accuracy: 0.6292
                precision    recall  f1-score   support

      BUSINESS     0.5915    0.6315    0.6108       947
 ENTERTAINMENT     0.5703    0.5216    0.5449       972
  FOOD & DRINK     0.6555    0.7177    0.6852      1013
     PARENTING     0.5329    0.6242    0.5750       998
      POLITICS     0.6703    0.5378    0.5968      1032
        SPORTS     0.7294    0.6431    0.6836      1006
STYLE & BEAUTY     0.6845    0.6445    0.6639       993
        TRAVEL     0.6694    0.6310    0.6496      1027
      WELLNESS     0.5550    0.6465    0.5972      1007
    WORLD NEWS     0.6748    0.6922    0.6834      1004

      accuracy                         0.6292      9999
     macro avg     0.6334    0.6290    0.6290      9999
  weighted avg     0.6340    0.6292    0.6294      9999


Best model on test set: LogReg (Accuracy: 0.6431)
```

Model Comparison — Which Classifier Fits Best?

What this cell does (1–2 lines):
Trains **Logistic Regression**, **Linear SVM**, and **Multinomial Naive Bayes** on the same TF-IDF features and test split; prints metrics; selects the best by accuracy.

Why this matters:
Comparing strong linear baselines helps identify the simplest model that generalizes best before trying heavier approaches (e.g., deep learning).

Results from my run (test set):

- **LogReg:** Accuracy ≈ **0.643** (best)
- **LinearSVC:** Accuracy ≈ 0.638
- **MultinomialNB:** Accuracy ≈ 0.629

Interpretation:

- **Logistic Regression** slightly outperforms the others overall, so we'll treat it as the reference.
- Differences are modest, which is common for well-engineered linear text features.
- Per-class reports show some categories (e.g., SPORTS, FOOD & DRINK) are easier than others (e.g., ENTERTAINMENT) — likely due to more distinctive vocabulary.

3.2 5-Fold Cross-Validation (News)

We run 5-fold Stratified CV (macro-F1) on the training set for each model to report mean ± std.

```
# News — 5-fold CV on training set (macro-F1)
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import make_scorer, f1_score

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scorer = make_scorer(f1_score, average='macro')

for name, clf in models.items():
    scores = cross_val_score(clf, X_train, y_train, cv=cv, scoring=scorer, n_jo
    print(f"{name} macro-F1 (5-fold CV on training): {scores.mean():.3f} ± {sco
```

```
LogReg macro-F1 (5-fold CV on training): 0.641 ± 0.009
LinearSVC macro-F1 (5-fold CV on training): 0.633 ± 0.007
MultinomialNB macro-F1 (5-fold CV on training): 0.626 ± 0.009
```

Robustness Check — 5-Fold Stratified CV (Macro-F1)

What this cell does (1–2 lines):
Runs 5-fold cross-validation on the **training** set to estimate how each model might generalize to unseen data, using **macro-F1** (treats all classes equally).

Why this matters:
Macro-F1 avoids dominance by frequent classes and is ideal for multi-class tasks with near-balanced labels.

Results from my run (training CV):

- **LogReg:** Macro-F1 ≈ **0.641 ± 0.009**
- **LinearSVC:** ≈ 0.633 ± 0.007
- **MultinomialNB:** ≈ 0.626 ± 0.009

Interpretation:
The CV ranking mirrors the test accuracy: **LogReg** remains best. Tight standard deviations suggest stable performance across folds.

4. Model Evaluation

```
# 4. Model Evaluation

from sklearn.metrics import accuracy_score, classification_report, confusion_ma

y_pred = model.predict(X_test)

print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, targe
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy Score: 0.643064306430643

Classification Report:
                  precision    recall  f1-score   support

      BUSINESS       0.62      0.66      0.64       947
 ENTERTAINMENT       0.53      0.56      0.54       972
  FOOD & DRINK       0.68      0.70      0.69      1013
     PARENTING       0.66      0.62      0.64       998
      POLITICS       0.64      0.56      0.60      1032
        SPORTS       0.68      0.70      0.69      1006
STYLE & BEAUTY       0.71      0.66      0.69       993
        TRAVEL       0.69      0.65      0.67      1027
      WELLNESS       0.58      0.64      0.61      1007
    WORLD NEWS       0.67      0.67      0.67      1004

      accuracy                           0.64      9999
     macro avg       0.64      0.64      0.64      9999
  weighted avg       0.65      0.64      0.64      9999


Confusion Matrix:
 [[627  27  20  29  61  27  16  28  67  45]
 [ 36 548  45  44  46  90  68  25  36  34]
 [ 21  51 714  32  14  27  35  55  54  10]
 [ 36  47  44 619  23  26  36  30 124  13]
 [ 86  54  21  35 573  50  19  25  45 124]
 [ 18 106  27  16  34 702  16  28  21  38]
 [ 36  88  44  41  13  30 660  31  41   9]
 [ 31  54  71  36  20  24  42 664  48  37]
 [ 58  36  59  63  24  29  27  38 648  25]
 [ 67  29   8  26  81  35  15  38  30 675]]
```

Final Evaluation — Accuracy, Reports, and Confusions

What this cell does (1–2 lines):
Evaluates the chosen model on the **held-out test set** and prints overall accuracy, a per-class classification report, and the confusion matrix.

Why this matters:
This is the **official** performance we would report. The confusion matrix reveals where the model confuses categories and why.

Key results from my run:

- **Accuracy:** ≈ **0.643** on 10 classes.
- **Per-class precision/recall/F1:** Stronger on categories with distinctive vocab (e.g., SPORTS, FOOD & DRINK) and slightly weaker on more overlapping themes (e.g., ENTERTAINMENT).
- **Notable confusions (examples observed):**
    - **POLITICS ↔ WORLD NEWS:** Frequent cross-predictions (e.g., many POLITICS labeled as WORLD NEWS), suggesting overlapping geopolitical terms.
    - **ENTERTAINMENT ↔ SPORTS/STYLE:** Some headlines share celebrity names or lifestyle terms, causing spillover.

Interpretation and next steps:

- The ~0.64 accuracy is reasonable for a lightweight linear baseline on short descriptions.
- **Improvements to try:**
    1. Add `headline` text to `text` (concatenate) to give more context.
    2. Tune TF-IDF (e.g., `ngram_range=(1,2)`, `min_df`, `max_df`).
    3. Calibrate class weights or try **LinearSVC** with probability calibration for different metrics.
    4. Try more expressive models (e.g., **Logistic Regression + n-grams**, **SGDClassifier**, or transformer embeddings) if resources allow.
    5. Visualize the confusion matrix as a heatmap and add top-words per class to explain model decisions for the report.

One-line takeaway (for the report):
A TF-IDF + Logistic Regression baseline achieves ~0.64 accuracy on 10 news categories; errors concentrate on semantically adjacent topics (e.g., POLITICS vs WORLD NEWS), pointing to the value of adding more context and richer features.

Google Drive Link to Codes, reports, presentations and video explanations for the project: https://drive.google.com/drive/folders/1gULvQYdkDMaHAtvwXY2bkE0FrExAVH3D?usp=sharing

By, Utkarsh Anand