🤖 NewsBot Intelligence System

ITAI 2373 - Mid-Term Group Project Template

Team Members: [Add your names here] **Date:** [Add date] **GitHub Repository:** [Add your repo URL here]

🎯 Project Overview

Welcome to your NewsBot Intelligence System! This notebook will guide you through building a comprehensive NLP system that:

- 📰 **Processes** news articles with advanced text cleaning
- 🏷️ **Classifies** articles into categories (Politics, Sports, Technology, Business, Entertainment, Health)
- 🔍 **Extracts** named entities (people, organizations, locations, dates, money)
- 😊 **Analyzes** sentiment and emotional tone
- 📊 **Generates** insights for business intelligence

📚 Module Integration Checklist

- ☐ **Module 1:** NLP applications and real-world context
- ☐ **Module 2:** Text preprocessing pipeline
- ☐ **Module 3:** TF-IDF feature extraction
- ☐ **Module 4:** POS tagging analysis
- ☐ **Module 5:** Syntax parsing and semantic analysis
- ☐ **Module 6:** Sentiment and emotion analysis
- ☐ **Module 7:** Text classification system
- ☐ **Module 8:** Named Entity Recognition

📦 Setup and Installation

Let's start by installing and importing all the libraries we'll need for our NewsBot system.

```
# Install required packages (run this cell first!)
!pip install spacy scikit-learn nltk pandas matplotlib seaborn wordcloud plotly
!python -m spacy download en_core_web_sm

# Download NLTK data
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('vader_lexicon')
nltk.download('averaged_perceptron_tagger')

print("✅ All packages installed successfully!")
```

```
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.13.0->spa
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from thinc<8.4.0,>=8.3.4->spacy)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.12/dist-packages (from thinc<8.4.0,>=8.3.4->
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from typer<1.0.0,>=0.3.0->spacy)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.12/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (13.9
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from weasel<0.5.0,>=0.1.0
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.12/dist-packages (from weasel<0.5.0,>=0.1.0-
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->spacy) (3.0.3)
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from language-data>=1.2->langcode
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich>=10.11.0->typer<1.0
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich>=10.11.0->typer<1
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-an
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 48.7 MB/s eta 0:00:00
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
[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.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
✅ All packages installed successfully!
```

```
# Import all necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import plotly.express as px
import plotly.graph_objects as go
from collections import Counter, defaultdict
import re
import warnings
warnings.filterwarnings('ignore')

# NLP Libraries
import spacy
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.tag import pos_tag

# Scikit-learn for machine learning
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.pipeline import Pipeline

# Load spaCy model
nlp = spacy.load('en_core_web_sm')

# Set up plotting style
plt.style.use('default')
sns.set_palette("husl")

print("📚 All libraries imported successfully!")
print(f"🔧 spaCy model loaded: {nlp.meta['name']} v{nlp.meta['version']}")

!pip install kaggle

from google.colab import files
```

```
from google.colab import files
print("Please upload your kaggle.json file:")
uploaded = files.upload()

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json


print("✅ Kaggle API setup complete!")
```

```
📚 All libraries imported successfully!
🔧 spaCy model loaded: core_web_sm v3.8.0
Requirement already satisfied: kaggle in /usr/local/lib/python3.12/dist-packages (1.7.4.5)
Requirement already satisfied: bleach in /usr/local/lib/python3.12/dist-packages (from kaggle) (6.3.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2025.10.5)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.12/dist-packages (from kaggle) (3.4.4)
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages (from kaggle) (3.11)
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages (from kaggle) (5.29.5)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.9.0.post0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.12/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.32.4)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.12/dist-packages (from kaggle) (75.2.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.12/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.12/dist-packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.5.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages (from kaggle) (0.5.1)
Please upload your kaggle.json file:
Choose Files kaggle.json
kaggle.json(application/json) - 67 bytes, last modified: 11/6/2025 - 100% done
Saving kaggle.json to kaggle.json
✅ Kaggle API setup complete!
```

📊 Data Loading and Exploration

🎯 Module 1: Understanding Our NLP Application

Before we dive into the technical implementation, let's understand the real-world context of our NewsBot Intelligence System. This system addresses several business needs:

1. **Media Monitoring:** Automatically categorize and track news coverage
2. **Business Intelligence:** Extract key entities and sentiment trends
3. **Content Management:** Organize large volumes of news content
4. **Market Research:** Understand public sentiment about topics and entities

💡 **Discussion Question:** What other real-world applications can you think of for this type of system? Consider different industries and use cases.

```
# Load your dataset
# 💡 TIP: If using the provided dataset, upload it to Colab first
# 💡 TIP: You can also use sample datasets like BBC News or 20 Newsgroups


# Option 1: Load provided dataset
df = pd.read_csv('bbcnews.csv')

# Option 2: Load BBC News dataset (if using alternative)
# You can download this from: https://www.kaggle.com/c/learn-ai-bbc/data

print("Dataset Info:")
print(f"Shape: {df.shape}")
print(f"Columns: {df.columns.tolist()}")
print("\nFirst few rows:")
print(df.head())

# Correct column names based on df.columns.tolist()
text_column = 'Text'
category_column = 'Category'
article_id_column = 'ArticleId' # Define article_id_column

print(f"\nMissing values:")
print(df.isnull().sum())

df_clean = df.dropna(subset=[text_column, category_column])
```

```
print(f"\nCategory distribution:")
print(df_clean[category_column].value_counts())

if len(df_clean) > 2000:
    df_final = df_clean.sample(n=2000, random_state=42)
    print(f"\nSampled dataset to {len(df_final)} rows")
else:
    df_final = df_clean

# 7. Rename columns for consistency
df_final = df_final.rename(columns={
    text_column: 'content',
    category_column: 'category',
    article_id_column: 'article_id' # Rename ArticleId to article_id
})

# 8. Save prepared dataset
df_final.to_csv('newsbot_dataset.csv', index=False)
print("\n✅ Dataset prepared and saved as 'newsbot_dataset.csv'")

print(f"📊 Dataset loaded successfully!")
print(f"📈 Shape: {df.shape}")
print(f"📋 Columns: {list(df.columns)}")

# Display first few rows
df.head()
```

```
Dataset Info:
Shape: (1490, 3)
Columns: ['ArticleId', 'Text', 'Category']

First few rows:
   ArticleId                                               Text  Category
0       1833  worldcom ex-boss launches defence lawyers defe...  business
1        154  german business confidence slides german busin...  business
2       1101  bbc poll indicates economic gloom citizens in ...  business
3       1976  lifestyle  governs mobile choice  faster  bett...      tech
4        917  enron bosses in $168m payout eighteen former e...  business

Missing values:
ArticleId    0
Text         0
Category     0
dtype: int64

Category distribution:
Category
sport            346
business         336
politics         274
entertainment    273
tech             261
Name: count, dtype: int64

✅ Dataset prepared and saved as 'newsbot_dataset.csv'
📊 Dataset loaded successfully!
📈 Shape: (1490, 3)
📋 Columns: ['ArticleId', 'Text', 'Category']
```

	ArticleId	Text	Category
0	1833	worldcom ex-boss launches defence lawyers defe...	business
1	154	german business confidence slides german busin...	business
2	1101	bbc poll indicates economic gloom citizens in ...	business
3	1976	lifestyle governs mobile choice faster bett...	tech
4	917	enron bosses in $168m payout eighteen former e...	business

Next steps: Generate code with df | New interactive sheet

```
# Basic dataset exploration
print("📊 DATASET OVERVIEW")
print("=" * 50)
print(f"Total articles: {len(df_final)}")
print(f"Unique categories: {df_final['category'].nunique()}")
print(f"Categories: {df_final['category'].unique().tolist()}")
# print(f"Date range: {df_final['date'].min()} to {df_final['date'].max()}") # Remove as 'date' column is not present
```

```
# print(f"Unique sources: {df_final['source'].nunique()}") # Remove as 'source' column is not present

print("\n📈 CATEGORY DISTRIBUTION")
print("=" * 50)
category_counts = df_final['category'].value_counts()
print(category_counts)

# Visualize category distribution
plt.figure(figsize=(10, 6))
sns.countplot(data=df_final, x='category', order=category_counts.index)
plt.title('Distribution of News Categories')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 💡 STUDENT TASK: Add your own exploratory analysis here
# - Check for missing values (already done in the previous cell)
# - Analyze text length distribution
# - Examine source distribution (not available in this dataset)
# - Look for any data quality issues

print("\n📊 TEXT LENGTH DISTRIBUTION")
print("=" * 50)
df_final['text_length'] = df_final['content'].str.len()
plt.figure(figsize=(10, 6))
sns.histplot(data=df_final, x='text_length', hue='category', kde=True)
plt.title('Distribution of Article Text Length by Category')
plt.xlabel('Text Length (characters)')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

print("\n📝 TEXT LENGTH STATISTICS")
print("=" * 50)
print(df_final.groupby('category')['text_length'].describe())
```

🧹 Text Preprocessing Pipeline

🎯 Module 2: Advanced Text Preprocessing

Now we'll implement a comprehensive text preprocessing pipeline that cleans and normalizes our news articles. This is crucial for all downstream NLP tasks.

Key Preprocessing Steps:

1. **Text Cleaning:** Remove HTML, URLs, special characters
2. **Tokenization:** Split text into individual words
3. **Normalization:** Convert to lowercase, handle contractions
4. **Stop Word Removal:** Remove common words that don't carry meaning
5. **Lemmatization:** Reduce words to their base form

💡 **Think About:** Why is preprocessing so important? What happens if we skip these steps?

```
# Initialize preprocessing tools
import nltk
```

```
nltk.download('punkt_tab')

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    """
    Comprehensive text cleaning function

    💡 TIP: This function should handle:
    - HTML tags and entities
    - URLs and email addresses
    - Special characters and numbers
    - Extra whitespace
    """
    if pd.isna(text):
        return ""

    # Convert to string and lowercase
    text = str(text).lower()

    # 🚀 YOUR CODE HERE: Implement text cleaning
    # Remove HTML tags
    text = re.sub(r'<[^>]+>', '', text)

    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remove email addresses
    text = re.sub(r'\S+@\S+', '', text)

    # Remove special characters and digits (keep only letters and spaces)
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    return text

def preprocess_text(text, remove_stopwords=True, lemmatize=True):
    """
    Complete preprocessing pipeline

    💡 TIP: This function should:
    - Clean the text
    - Tokenize into words
    - Remove stop words (optional)
    - Lemmatize words (optional)
    - Return processed text
    """
    # Clean text
    text = clean_text(text)

    if not text:
        return ""

    # 🚀 YOUR CODE HERE: Implement tokenization and preprocessing
    # Tokenize
    tokens = word_tokenize(text)

    # Remove stop words if requested
    if remove_stopwords:
        tokens = [token for token in tokens if token not in stop_words]

    # Lemmatize if requested
    if lemmatize:
        tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Filter out very short words
    tokens = [token for token in tokens if len(token) > 2]

    return ' '.join(tokens)

# Test the preprocessing function
sample_text = "Apple Inc. announced record quarterly earnings today! Visit https://apple.com for more info. #TechNews"
print("Original text:")
print(sample_text)
print("\nCleaned text:")
```

```
print(clean_text(sample_text))
print("\nFully preprocessed text:")
print(preprocess_text(sample_text))
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Original text:
Apple Inc. announced record quarterly earnings today! Visit https://apple.com for more info. #TechNews
Cleaned text:
apple inc announced record quarterly earnings today visit for more info technews

Fully preprocessed text:
apple inc announced record quarterly earnings today visit info technews
```

```
# Apply preprocessing to the dataset
print("🧹 Preprocessing all articles...")

# Create new columns for processed text
# Operating on df_final which has 'content' and 'category' columns
df_final['content_clean'] = df_final['content'].apply(clean_text)
df_final['content_processed'] = df_final['content'].apply(preprocess_text)

# For consistency and potential future use, add a 'full_text' column to df_final if it doesn't exist
# In this dataset, 'content' is essentially the full text, but we'll create a separate column for clarity
df_final['full_text'] = df_final['content'] # Using 'content' which was renamed from 'Text'
df_final['full_text_processed'] = df_final['full_text'].apply(preprocess_text)


print("✅ Preprocessing complete!")

# Show before and after examples
print("\n📝 BEFORE AND AFTER EXAMPLES")
print("=" * 60)
# Show examples from df_final
for i in range(min(3, len(df_final))):
    print(f"\nExample {i+1}:")
    print(f"Original: {df_final.iloc[i]['full_text'][:100]}...")
    print(f"Processed: {df_final.iloc[i]['full_text_processed'][:100]}...")
```

```
🧹 Preprocessing all articles...
✅ Preprocessing complete!

📝 BEFORE AND AFTER EXAMPLES
============================================================

Example 1:
Original: worldcom ex-boss launches defence lawyers defending former worldcom chief bernie ebbers against a ba...
Processed: worldcom exboss launch defence lawyer defending former worldcom chief bernie ebbers battery fraud ch...

Example 2:
Original: german business confidence slides german business confidence fell in february knocking hopes of a sp...
Processed: german business confidence slide german business confidence fell february knocking hope speedy recov...

Example 3:
Original: bbc poll indicates economic gloom citizens in a majority of nations surveyed in a bbc world service ...
Processed: bbc poll indicates economic gloom citizen majority nation surveyed bbc world service poll believe wo...
```

```
# 💡 STUDENT TASK: Analyze the preprocessing results
# - Calculate average text length before and after
# - Count unique words before and after
# - Identify the most common words after preprocessing

print("\n📊 PREPROCESSING ANALYSIS")
print("=" * 50)

# Calculate average text length before and after
df_final['full_text_length_before'] = df_final['full_text'].str.len()
df_final['full_text_length_after'] = df_final['full_text_processed'].str.len()

print(f"Average text length before preprocessing: {df_final['full_text_length_before'].mean():.0f} characters")
print(f"Average text length after preprocessing: {df_final['full_text_length_after'].mean():.0f} characters")

# Count unique words before and after
# This can be computationally intensive for large datasets, so we'll sample or use a count vectorizer approach
print("\nCounting unique words (vocabulary size)...")
```

```
                  count    mean     std     min     25%     50%     75%      max
category
business          336.0  1983.104167  790.180447  846.0  1486.25  1830.5   5406.0
entertainment     273.0  1910.380952  1142.478958   866.0  1312.00  1571.0   6811.0
politics          274.0  2617.905109  1448.447009   991.0  1867.00  2599.5   9890.0
sport             346.0  1894.624277  1051.814635   719.0  1199.00  1641.0   7992.0
tech              261.0  2939.291188  1215.569461  1003.0  2031.00  2657.0   9512.0
                  count    mean     std     min     25%     50%     75%      max
category
business          2221.00  5406.0
entertainment     2147.00  13619.0
```

```
# Using CountVectorizer for efficiency
# Fit on the 'full_text' and 'full_text_processed' columns of df_final
vectorizer_before = CountVectorizer(max_features=10000) # Limit vocab size for before as well
vectorizer_after = CountVectorizer(max_features=10000)

vectorizer_before.fit(df_final['full_text'].dropna())
vectorizer_after.fit(df_final['full_text_processed'].dropna())

vocab_before = len(vectorizer_before.vocabulary_)
vocab_after = len(vectorizer_after.vocabulary_)

print(f"Unique words before preprocessing (approx): {vocab_before}")
print(f"Unique words after preprocessing (approx): {vocab_after}")


# Identify the most common words after preprocessing
print("\nMost common words after preprocessing:")
all_processed_text = ' '.join(df_final['full_text_processed'].dropna())
words = all_processed_text.split()
most_common_words = Counter(words).most_common(20)

for word, count in most_common_words:
    print(f"  {word}: {count}")

print("\n✅ Preprocessing analysis complete!")
```

```
📊 PREPROCESSING ANALYSIS
==================================================
Average text length before preprocessing: 2233 characters
Average text length after preprocessing: 1481 characters

Counting unique words (vocabulary size)...
Unique words before preprocessing (approx): 10000
Unique words after preprocessing (approx): 10000

Most common words after preprocessing:
  said: 4838
  year: 1872
  would: 1711
  also: 1426
  new: 1334
  people: 1323
  one: 1190
  could: 1032
  game: 949
  time: 940
  first: 893
  last: 883
  say: 844
  two: 816
  world: 811
  film: 802
  government: 771
  make: 695
  company: 682
  firm: 675

✅ Preprocessing analysis complete!
```

📊 Feature Extraction and Statistical Analysis

🎯 Module 3: TF-IDF Analysis

Now we'll extract numerical features from our text using TF-IDF (Term Frequency-Inverse Document Frequency). This technique helps us identify the most important words in each document and across the entire corpus.

TF-IDF Key Concepts:

- **Term Frequency (TF):** How often a word appears in a document
- **Inverse Document Frequency (IDF):** How rare a word is across all documents
- **TF-IDF Score:** TF × IDF - balances frequency with uniqueness

💡 **Business Value:** TF-IDF helps us identify the most distinctive and important terms for each news category.

```
# Create TF-IDF vectorizer
# 💡 TIP: Experiment with different parameters:
```

```
# - max_features: limit vocabulary size
# - ngram_range: include phrases (1,1) for words, (1,2) for words+bigrams
# - min_df: ignore terms that appear in less than min_df documents
# - max_df: ignore terms that appear in more than max_df fraction of documents

tfidf_vectorizer = TfidfVectorizer(
    max_features=5000,  # Limit vocabulary for computational efficiency
    ngram_range=(1, 2),  # Include unigrams and bigrams
    min_df=2,  # Ignore terms that appear in less than 2 documents
    max_df=0.8  # Ignore terms that appear in more than 80% of documents
)

# Fit and transform the processed text
print("🔢 Creating TF-IDF features...")
tfidf_matrix = tfidf_vectorizer.fit_transform(df_final['full_text_processed'])
feature_names = tfidf_vectorizer.get_feature_names_out()

print(f"✅ TF-IDF matrix created!")
print(f"📊 Shape: {tfidf_matrix.shape}")
print(f"📝 Vocabulary size: {len(feature_names)}")
print(f"🔢 Sparsity: {(1 - tfidf_matrix.nnz / (tfidf_matrix.shape[0] * tfidf_matrix.shape[1])) * 100:.2f}%")

# Convert to DataFrame for easier analysis
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=feature_names)
tfidf_df['category'] = df_final['category'].values # Use df_final for category column

print("\n🔍 Sample TF-IDF features:")
print(tfidf_df.iloc[:3, :10])  # Show first 3 rows and 10 features
```

```
🔢 Creating TF-IDF features...
✅ TF-IDF matrix created!
📊 Shape: (1490, 5000)
📝 Vocabulary size: 5000
🔢 Sparsity: 97.50%

🔍 Sample TF-IDF features:
   abbas  abc  ability  able  abroad  absence  absolute  absolutely  abuse  \
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

   abused
0     0.0
1     0.0
2     0.0
```

```
# Analyze most important terms per category
def get_top_tfidf_terms(category, n_terms=10):
    """
    Get top TF-IDF terms for a specific category

    💡 TIP: This function should:
    - Filter data for the specific category
    - Calculate mean TF-IDF scores for each term
    - Return top N terms with highest scores
    """
    # 🚀 YOUR CODE HERE: Implement category-specific TF-IDF analysis
    category_data = tfidf_df[tfidf_df['category'] == category]

    # Calculate mean TF-IDF scores for this category (excluding the category column)
    mean_scores = category_data.drop('category', axis=1).mean().sort_values(ascending=False)

    return mean_scores.head(n_terms)

# Analyze top terms for each category
print("🥠 TOP TF-IDF TERMS BY CATEGORY")
print("=" * 50)

categories = df_final['category'].unique() # Use df_final to get categories
category_terms = {}

for category in categories:
    top_terms = get_top_tfidf_terms(category, n_terms=10)
    category_terms[category] = top_terms

    print(f"\n📰 {category.upper()}:")
    for term, score in top_terms.items():
        print(f"  {term}: {score:.4f}")
```

```
# 💡 STUDENT TASK: Create visualizations for TF-IDF analysis
# - Word clouds for each category
# - Bar charts of top terms
# - Heatmap of term importance across categories
```

```
  firm: 0.0390
  company: 0.0371
  market: 0.0344
  bank: 0.0336
  year: 0.0334
  growth: 0.0332
  economy: 0.0317
  sale: 0.0316
  share: 0.0307
  profit: 0.0274

📰 TECH:
  mobile: 0.0513
  phone: 0.0468
  people: 0.0457
  technology: 0.0410
  game: 0.0381
  user: 0.0377
  service: 0.0369
  software: 0.0365
  computer: 0.0329
  net: 0.0308

📰 POLITICS:
  labour: 0.0648
  election: 0.0604
  blair: 0.0558
  party: 0.0538
  tory: 0.0463
  would: 0.0456
  government: 0.0455
  minister: 0.0427
  brown: 0.0384
  tax: 0.0330

📰 SPORT:
  game: 0.0448
  england: 0.0375
  win: 0.0342
  player: 0.0323
  match: 0.0304
  champion: 0.0293
  cup: 0.0278
  team: 0.0260
  chelsea: 0.0255
  injury: 0.0249

📰 ENTERTAINMENT:
  film: 0.1000
  award: 0.0524
  best: 0.0460
  show: 0.0376
  star: 0.0375
  music: 0.0357
  band: 0.0350
  actor: 0.0339
  year: 0.0300
  album: 0.0291
```

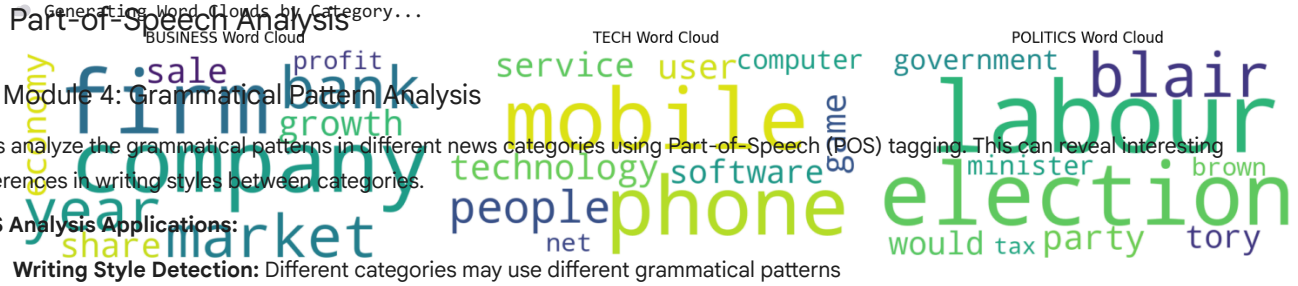
```
# 💡 STUDENT TASK: Create visualizations for TF-IDF analysis

print("📊 VISUALIZING TF-IDF ANALYSIS")
print("=" * 50)

# Word clouds for each category
print("\n☁️ Generating Word Clouds by Category...")
plt.figure(figsize=(15, 10))
for i, (category, terms) in enumerate(category_terms.items()):
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(terms)
    plt.subplot(2, 3, i + 1)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f'{category.upper()} Word Cloud')
plt.tight_layout()
plt.show()
```

```
# Bar charts of top terms per category
print("\n📈 Generating Bar Charts of Top Terms by Category...")
plt.figure(figsize=(15, 10))
for i, (category, terms) in enumerate(category_terms.items()):
    plt.subplot(2, 3, i + 1)
    terms.sort_values().plot(kind='barh', color=sns.color_palette("husl", 10))
    plt.title(f'Top Terms in {category.upper()}')
    plt.xlabel('Mean TF-IDF Score')
    plt.ylabel('Term')
plt.tight_layout()
plt.show()


# Heatmap of term importance across categories (for a selection of top terms overall)
print("\n🔥 Generating Heatmap of Term Importance...")

# Get the top terms overall (across all categories) for the heatmap
all_terms_mean_tfidf = tfidf_df.drop('category', axis=1).mean().sort_values(ascending=False).head(50) # Top 50 terms overall
top_overall_terms = all_terms_mean_tfidf.index.tolist()

# Create a pivot table for the heatmap
heatmap_data = tfidf_df[tfidf_df.columns.intersection(top_overall_terms + ['category'])]
heatmap_avg_scores = heatmap_data.groupby('category')[top_overall_terms].mean()


if not heatmap_avg_scores.empty:
    plt.figure(figsize=(14, 10))
    sns.heatmap(heatmap_avg_scores.T, annot=False, cmap='YlGnBu')
    plt.title('Average TF-IDF Score of Top Terms Across Categories')
    plt.xlabel('Category')
    plt.ylabel('Term')
    plt.tight_layout()
    plt.show()
else:
    print("⚠️ Could not generate heatmap. Check if top terms were found.")

print("\n✅ TF-IDF visualizations complete!")
```

```
📊 VISUALIZING TF-IDF ANALYSIS
==================================================
```

🏈 Part-of-Speech Analysis

```
🎨 Generating Word Clouds by Category...
```



🎯 Module 4: Grammatical Pattern Analysis

Let's analyze the grammatical patterns in different news categories using Part-of-Speech (POS) tagging. This can reveal interesting differences in writing styles between categories.

POS Analysis Applications:

- **Writing Style Detection:** Different categories may use different grammatical patterns
- **Content Quality Assessment:** Proper noun density, adjective usage, etc.
- **Feature Engineering:** POS tags can be features for classification

💡 **Hypothesis:** Sports articles might have more action verbs, while business articles might have more numbers and proper nouns.

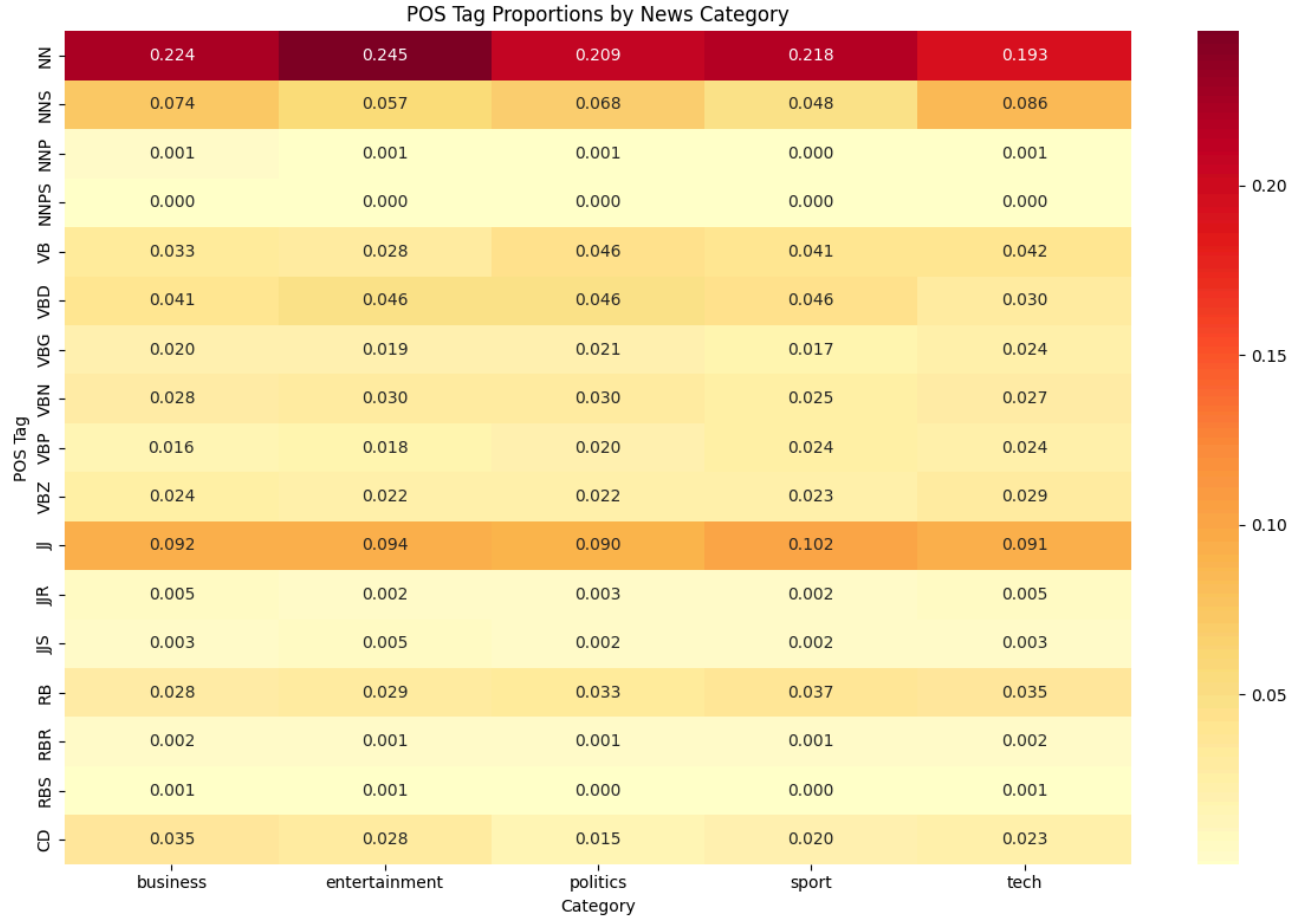

```
import nltk
nltk.download('averaged_perceptron_tagger_eng')

def analyze_pos_patterns(text):
    """
    Analyze POS patterns in text

    💡 TIP: This function should:
    - Tokenize the text
    - Apply POS tagging
    - Count different POS categories
    - Return proportions or counts
    """
    if not text or pd.isna(text):
        return {}

    # 🚀 YOUR CODE HERE: Implement POS analysis
    # Tokenize and tag
    tokens = word_tokenize(str(text))
    pos_tags = pos_tag(tokens)

    # Count POS categories
    pos_counts = Counter([tag for word, tag in pos_tags])
    total_words = len(pos_tags)

    if total_words == 0:
        return {}

    # Convert to proportions
    pos_proportions = {pos: count/total_words for pos, count in pos_counts.items()}

    return pos_proportions

# Apply POS analysis to all articles
print("🏈 Analyzing POS patterns...")

# Analyze POS for each article
pos_results = []
# Iterate over df_final instead of df
for idx, row in df_final.iterrows():
    pos_analysis = analyze_pos_patterns(row['content']) # Use 'content' from df_final
    pos_analysis['category'] = row['category'] # Use 'category' from df_final
    pos_analysis['article_id'] = row['article_id'] # Use 'article_id' from df_final
    pos_results.append(pos_analysis)

# Convert to DataFrame
pos_df = pd.DataFrame(pos_results).fillna(0)

print(f"✅ POS analysis complete!")
print(f"📊 Found {len(pos_df.columns)-2} different POS tags")

# Show sample results
print("\n📝 Sample POS analysis:")
print(pos_df.head())
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
```

```
[nltk_data] Unzipping taggers/averaged_perceptron_tagger_eng.zip.
🔍 Analyzing POS patterns...
✅ POS analysis complete!
📊 Found 43 different POS tags

📝 Sample POS analysis:
        JJ       NNS        NN       VBG        IN        DT       VBP  \
0 0.088068  0.113924  0.231013  0.025316  0.110759  0.066456  0.015823
1 0.119883  0.070175  0.233918  0.032164  0.134503  0.081871  0.014620
2 0.079044  0.079044  0.200368  0.025735  0.119485  0.095588  0.016544
3 0.035582  0.107463  0.156716  0.022388  0.122388  0.062687  0.041791
4 0.089005  0.078534  0.212042  0.028796  0.123037  0.102094  0.015707

        VBN      PRP$         .  ...   FW  SYM  WP$  NNP  NNPS  POS   UH  \
0  0.041139  0.012658  0.037975  ...  0.0  0.0  0.0  0.0   0.0  0.0  0.0
1  0.029240  0.005848  0.040936  ...  0.0  0.0  0.0  0.0   0.0  0.0  0.0
2  0.029412  0.016544  0.040441  ...  0.0  0.0  0.0  0.0   0.0  0.0  0.0
3  0.017910  0.013433  0.046269  ...  0.0  0.0  0.0  0.0   0.0  0.0  0.0
4  0.026178  0.015707  0.041885  ...  0.0  0.0  0.0  0.0   0.0  0.0  0.0

     #  \
0  0.0
1  0.0
2  0.0
3  0.0
4  0.0

[5 rows x 45 columns]
```

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

# Analyze POS patterns by category
print("📊 POS PATTERNS BY CATEGORY")
print("=" * 50)

# Group by category and calculate mean proportions
pos_by_category = pos_df.groupby('category').mean()

# Focus on major POS categories
major_pos = ['NN', 'NNS', 'NNP', 'NNPS', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ',
             'JJ', 'JJR', 'JJS', 'RB', 'RBR', 'RBS', 'CD']

# Filter to only include major POS tags that exist in our data
available_pos = [pos for pos in major_pos if pos in pos_by_category.columns]

if available_pos:
    pos_summary = pos_by_category[available_pos]

    print("\n🎯 Key POS patterns by category:")
    print(pos_summary.round(4))

    # Create visualization
    plt.figure(figsize=(12, 8))
    sns.heatmap(pos_summary.T, annot=True, cmap='YlOrRd', fmt='.3f')
    plt.title('POS Tag Proportions by News Category')
    plt.xlabel('Category')
    plt.ylabel('POS Tag')
    plt.tight_layout()
    plt.show()

    # 💡 STUDENT TASK: Analyze the patterns
    # - Which categories use more nouns vs verbs?
    # - Do business articles have more numbers (CD)?
    # - Are there differences in adjective usage?

    print("\n💡 ANALYSIS QUESTIONS:")
    print("1. Which category has the highest proportion of proper nouns (NNP/NNPS)?")
    print("2. Which category uses the most action verbs (VB, VBD, VBG)?")
    print("3. Are there interesting patterns in adjective (JJ) usage?")
    print("4. How does number (CD) usage vary across categories?")
else:
    print("⚠️ No major POS tags found in the analysis. Check your POS tagging implementation.")
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
📊 POS PATTERNS BY CATEGORY
==================================================

🎯 Key POS patterns by category:
                   NN     NNS     NNP    NNPS      VB     VBD     VBG     VBN  \
category
business       0.2241  0.0736  0.0011  0.0000  0.0333  0.0410  0.0200  0.0277
entertainment  0.2454  0.0574  0.0008  0.0000  0.0281  0.0462  0.0194  0.0295
politics       0.2087  0.0679  0.0007  0.0003  0.0455  0.0462  0.0209  0.0302
sport          0.2184  0.0479  0.0005  0.0000  0.0408  0.0457  0.0175  0.0247
tech           0.1932  0.0855  0.0006  0.0000  0.0420  0.0299  0.0236  0.0270

                   VBP     VBZ      JJ     JJR     JJS      RB     RBR     RBS  \
category
business       0.0158  0.0241  0.0920  0.0045  0.0033  0.0276  0.0018  0.0006
entertainment  0.0179  0.0223  0.0939  0.0024  0.0050  0.0286  0.0010  0.0010
politics       0.0199  0.0224  0.0901  0.0032  0.0017  0.0325  0.0014  0.0003
sport          0.0244  0.0233  0.1015  0.0019  0.0018  0.0373  0.0012  0.0003
tech           0.0235  0.0285  0.0913  0.0052  0.0025  0.0346  0.0022  0.0007

                   CD
category
business       0.0347
entertainment  0.0278
politics       0.0147
sport          0.0203
tech           0.0228
```




```
💡 ANALYSIS QUESTIONS:
1. Which category has the highest proportion of proper nouns (NNP/NNPS)?
2. Which category uses the most action verbs (VB, VBD, VBG)?
3. Are there interesting patterns in adjective (JJ) usage?
4. How does number (CD) usage vary across categories?
```

🌳 Syntax Parsing and Semantic Analysis

🎯 Module 5: Understanding Sentence Structure

Now we'll use spaCy to perform dependency parsing and extract semantic relationships from our news articles. This helps us understand not just what words are present, but how they relate to each other.

Dependency Parsing Applications:

- **Relationship Extraction:** Find connections between entities
- **Event Detection:** Identify who did what to whom
- **Information Extraction:** Extract structured facts from unstructured text

💡 **Business Value:** Understanding sentence structure helps extract more precise information about events, relationships, and actions mentioned in news articles.

```
def extract_syntactic_features(text):
    """
    Extract syntactic features using spaCy dependency parsing

    💡 TIP: This function should extract:
    - Dependency relations
    - Subject-verb-object patterns
    - Noun phrases
    - Verb phrases
    """
    if not text or pd.isna(text):
        return {}

    # Process text with spaCy
    doc = nlp(str(text))

    features = {
        'num_sentences': len(list(doc.sents)),
        'num_tokens': len(doc),
        'dependency_relations': [],
        'noun_phrases': [],
        'verb_phrases': [],
        'subjects': [],
        'objects': []
    }

    # 🚀 YOUR CODE HERE: Extract syntactic features

    # Extract dependency relations
    for token in doc:
        if not token.is_space and not token.is_punct:
            features['dependency_relations'].append(token.dep_)

    # Extract noun phrases
    for chunk in doc.noun_chunks:
        features['noun_phrases'].append(chunk.text.lower())

    # Extract subjects and objects
    for token in doc:
        if token.dep_ in ['nsubj', 'nsubjpass']:  # Subjects
            features['subjects'].append(token.text.lower())
        elif token.dep_ in ['dobj', 'iobj', 'pobj']:  # Objects
            features['objects'].append(token.text.lower())

    # Count dependency types
    dep_counts = Counter(features['dependency_relations'])
    features['dependency_counts'] = dict(dep_counts)

    return features

# Apply syntactic analysis to sample articles
print("🌳 Performing syntactic analysis...")

# Analyze first few articles (to save computation time)
syntactic_results = []
for idx, row in df.head(5).iterrows():  # Limit to first 5 for demo
    features = extract_syntactic_features(row['full_text'])
    features['category'] = row['category']
    features['article_id'] = row['article_id']
    syntactic_results.append(features)

print("✅ Syntactic analysis complete!")

# Display results
```

```
for i, result in enumerate(syntactic_results):
    print(f"\n📰 Article {i+1} ({result['category']}):")
    print(f"  Sentences: {result['num_sentences']}")
    print(f"  Tokens: {result['num_tokens']}")
    print(f"  Noun phrases: {result['noun_phrases'][:3]}...")  # Show first 3
    print(f"  Subjects: {result['subjects'][:3]}...")  # Show first 3
    print(f"  Objects: {result['objects'][:3]}...")  # Show first 3
```

```
🌳 Performing syntactic analysis...
✅ Syntactic analysis complete!

📰 Article 1 (business):
  Sentences: 15
  Tokens: 346
  Noun phrases: ['-', 'boss', 'defence lawyers']...
  Subjects: ['-', 'boss', 'worldcom']...
  Objects: ['lawyers', 'ebbers', 'battery']...

📰 Article 2 (business):
  Sentences: 15
  Tokens: 368
  Noun phrases: ['german business confidence', 'german business confidence', 'february']...
  Subjects: ['confidence', 'ifo', 'index']...
  Objects: ['confidence', 'february', 'hopes']...

📰 Article 3 (business):
  Sentences: 24
  Tokens: 587
  Noun phrases: ['bbc poll', 'economic gloom citizens', 'a majority']...
  Subjects: ['poll', 'citizens', 'economy']...
  Objects: ['majority', 'nations', 'poll']...

📰 Article 4 (tech):
  Sentences: 31
  Tokens: 724
  Noun phrases: ['lifestyle', 'mobile choice', 'faster  better or funkier hardware']...
  Subjects: ['lifestyle', 'governs', 'firms']...
  Objects: ['choice', 'hardware', 'research']...

📰 Article 5 (business):
  Sentences: 17
  Tokens: 417
  Noun phrases: ['enron bosses', 'former enron directors', 'a $168m']...
  Subjects: ['bosses', 'directors', 'plaintiff']...
  Objects: ['168', 'm', 'lawsuit']...
```

```
def extract_syntactic_features(text):
    """
    Extract syntactic features using spaCy dependency parsing

    💡 TIP: This function should extract:
    - Dependency relations
    - Subject-verb-object patterns
    - Noun phrases
    - Verb phrases
    """
    if not text or pd.isna(text):
        return {}

    # Process text with spaCy
    doc = nlp(str(text))

    features = {
        'num_sentences': len(list(doc.sents)),
        'num_tokens': len(doc),
        'dependency_relations': [],
        'noun_phrases': [],
        'verb_phrases': [],
        'subjects': [],
        'objects': []
    }

    # 🚀 YOUR CODE HERE: Extract syntactic features

    # Extract dependency relations
    for token in doc:
        if not token.is_space and not token.is_punct:
            features['dependency_relations'].append(token.dep_)

    # Extract noun phrases
```

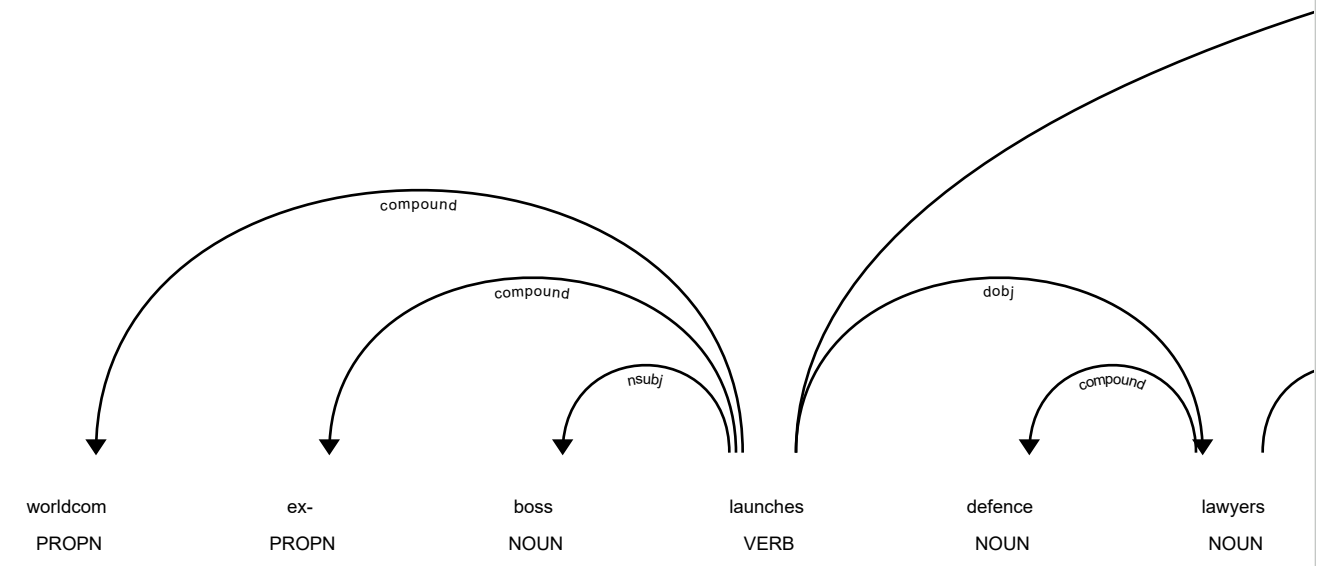
```
    for chunk in doc.noun_chunks:
        features['noun_phrases'].append(chunk.text.lower())

    # Extract subjects and objects
    for token in doc:
        if token.dep_ in ['nsubj', 'nsubjpass']:  # Subjects
            features['subjects'].append(token.text.lower())
        elif token.dep_ in ['dobj', 'iobj', 'pobj']:  # Objects
            features['objects'].append(token.text.lower())

    # Count dependency types
    dep_counts = Counter(features['dependency_relations'])
    features['dependency_counts'] = dict(dep_counts)

    return features

# Apply syntactic analysis to sample articles
print("🌳 Performing syntactic analysis...")

# Analyze first few articles (to save computation time)
syntactic_results = []
# Iterate over df_final instead of df
for idx, row in df_final.head(5).iterrows():  # Limit to first 5 for demo
    features = extract_syntactic_features(row['full_text']) # Use 'full_text' from df_final
    features['category'] = row['category'] # Use 'category' from df_final
    features['article_id'] = row['article_id'] # Use 'article_id' from df_final
    syntactic_results.append(features)

print("✅ Syntactic analysis complete!")

# Display results
for i, result in enumerate(syntactic_results):
    print(f"\n📰 Article {i+1} ({result['category']}):")
    print(f"  Sentences: {result['num_sentences']}")
    print(f"  Tokens: {result['num_tokens']}")
    print(f"  Noun phrases: {result['noun_phrases'][:3]}...")  # Show first 3
    print(f"  Subjects: {result['subjects'][:3]}...")  # Show first 3
    print(f"  Objects: {result['objects'][:3]}...")  # Show first 3
```

```
🌳 Performing syntactic analysis...
✅ Syntactic analysis complete!

📰 Article 1 (business):
  Sentences: 15
  Tokens: 346
  Noun phrases: ['-', 'boss', 'defence lawyers']...
  Subjects: ['-', 'boss', 'worldcom']...
  Objects: ['lawyers', 'ebbers', 'battery']...

📰 Article 2 (business):
  Sentences: 15
  Tokens: 368
  Noun phrases: ['german business confidence', 'german business confidence', 'february']...
  Subjects: ['confidence', 'ifo', 'index']...
  Objects: ['confidence', 'february', 'hopes']...

📰 Article 3 (business):
  Sentences: 24
  Tokens: 587
  Noun phrases: ['bbc poll', 'economic gloom citizens', 'a majority']...
  Subjects: ['poll', 'citizens', 'economy']...
  Objects: ['majority', 'nations', 'poll']...

📰 Article 4 (tech):
  Sentences: 31
  Tokens: 724
  Noun phrases: ['lifestyle', 'mobile choice', 'faster  better or funkier hardware']...
  Subjects: ['lifestyle', 'governs', 'firms']...
  Objects: ['choice', 'hardware', 'research']...

📰 Article 5 (business):
  Sentences: 17
  Tokens: 417
  Noun phrases: ['enron bosses', 'former enron directors', 'a $168m']...
  Subjects: ['bosses', 'directors', 'plaintiff']...
  Objects: ['168', 'm', 'lawsuit']...
```

```
# Visualize dependency parsing for a sample sentence
from spacy import displacy
```

```
# Choose a sample sentence
sample_sentence = df_final.iloc[0]['full_text']  # Use df_final and 'full_text' column
print(f"📝 Sample sentence: {sample_sentence}")

# Process with spaCy
doc = nlp(sample_sentence)

# Display dependency tree (this works best in Jupyter)
print("\n🌳 Dependency Parse Visualization:")
try:
    # This will create an interactive visualization in Jupyter
    displacy.render(doc, style="dep", jupyter=True)
except:
    # Fallback: print dependency information
    print("\n🔗 Dependency Relations:")
    for token in doc:
        if not token.is_space and not token.is_punct:
            print(f"  {token.text} --> {token.dep_} --> {token.head.text}")

# 💡 STUDENT TASK: Extend syntactic analysis
# - Compare syntactic complexity across categories
# - Extract action patterns (who did what)
# - Identify most common dependency relations per category
# - Create features for classification based on syntax
```

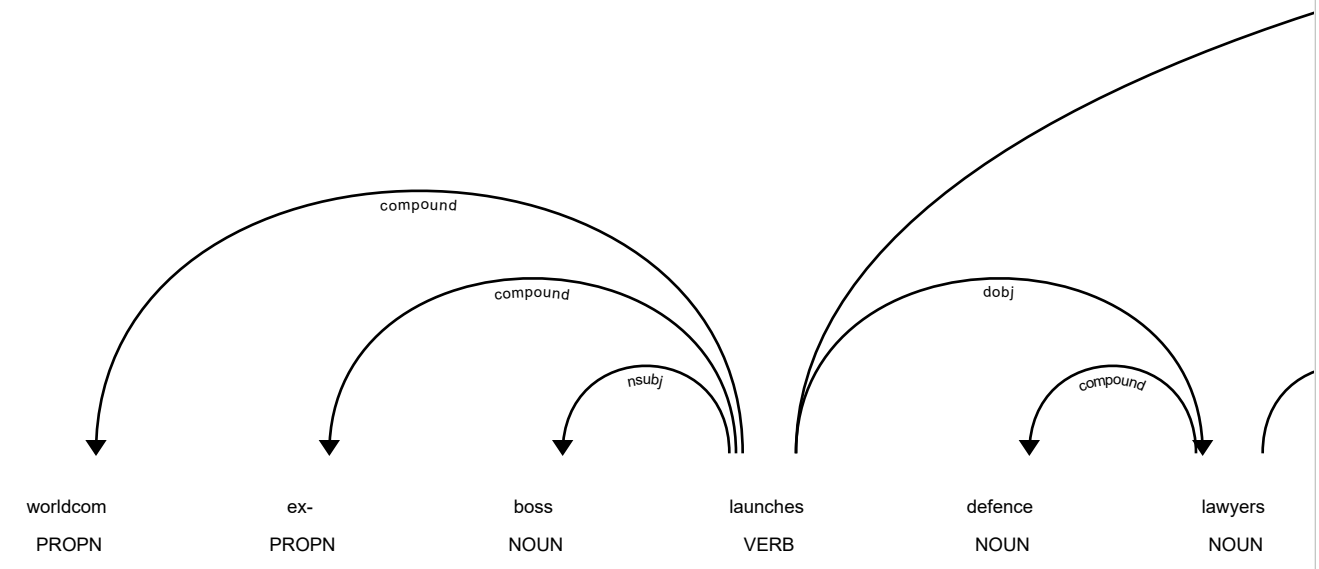
```
📝 Sample sentence: worldcom ex-boss launches defence lawyers defending former worldcom chief bernie ebbers against a battery of

🌳 Dependency Parse Visualization:
```




```
# Visualize dependency parsing for a sample sentence
from spacy import displacy

# Choose a sample sentence
sample_sentence = df_final.iloc[0]['full_text']  # Use df_final and 'full_text' column
print(f"📝 Sample sentence: {sample_sentence}")

# Process with spaCy
```

```
# Process with spacy
doc = nlp(sample_sentence)

# Display dependency tree (this works best in Jupyter)
print("\n🌳 Dependency Parse Visualization:")
try:
    # This will create an interactive visualization in Jupyter
    displacy.render(doc, style="dep", jupyter=True)
except:
    # Fallback: print dependency information
    print("\n🔗 Dependency Relations:")
    for token in doc:
        if not token.is_space and not token.is_punct:
            print(f"  {token.text} --> {token.dep_} --> {token.head.text}")

# 💡 STUDENT TASK: Extend syntactic analysis
# - Compare syntactic complexity across categories
# - Extract action patterns (who did what)
# - Identify most common dependency  relations per category
# - Create features for classification based on syntax
```

```
📝 Sample sentence: worldcom ex-boss launches defence lawyers defending former worldcom chief bernie ebbers against a battery of

🌳 Dependency Parse Visualization:
```



😊 Sentiment and Emotion Analysis

🎯 Module 6: Understanding Emotional Tone

Let's analyze the sentiment and emotional tone of our news articles. This can reveal interesting patterns about how different types of news are presented and perceived.

Sentiment Analysis Applications:

- **Media Bias Detection:** Identify emotional slant in news coverage
- **Public Opinion Tracking:** Monitor sentiment trends over time
- **Content Recommendation:** Suggest articles based on emotional tone

💡 **Hypothesis:** Different news categories might have different emotional profiles - sports might be more positive, politics more negative, etc.

```
# Initialize sentiment analyzer
sia = SentimentIntensityAnalyzer()

def analyze_sentiment(text):
    """
    Analyze sentiment using VADER sentiment analyzer

    💡 TIP: VADER returns:
    - compound: overall sentiment (-1 to 1)
    - pos: positive score (0 to 1)
    - neu: neutral score (0 to 1)
    - neg: negative score (0 to 1)
    """
    if not text or pd.isna(text):
        return {'compound': 0, 'pos': 0, 'neu': 1, 'neg': 0}

    # 🚀 YOUR CODE HERE: Implement sentiment analysis
    scores = sia.polarity_scores(str(text))

    # Add interpretation
    if scores['compound'] >= 0.05:
        scores['sentiment_label'] = 'positive'
    elif scores['compound'] <= -0.05:
        scores['sentiment_label'] = 'negative'
    else:
        scores['sentiment_label'] = 'neutral'

    return scores

# Apply sentiment analysis to all articles
print("😊 Analyzing sentiment...")

sentiment_results = []
# Iterate over df_final instead of df
for idx, row in df_final.iterrows():
    # Analyze both title and content (adjusting for dataset structure)
    # Since there's no separate title, analyze the full text
    full_sentiment = analyze_sentiment(row['full_text'])

    result = {
        'article_id': row['article_id'],
        'category': row['category'],
        # Removed title sentiment as there's no separate title column
        'full_sentiment': full_sentiment['compound'],
        'full_label': full_sentiment['sentiment_label'],
        'pos_score': full_sentiment['pos'],
        'neu_score': full_sentiment['neu'],
        'neg_score': full_sentiment['neg']
    }
    sentiment_results.append(result)

# Convert to DataFrame
sentiment_df = pd.DataFrame(sentiment_results)

print("✅ Sentiment analysis complete!")
print(f"📊 Analyzed {len(sentiment_df)} articles")

# Display sample results
print("\n📝 Sample sentiment results:")
print(sentiment_df[['category', 'full_sentiment', 'full_label']].head())
```

```
😊 Analyzing sentiment...
✅ Sentiment analysis complete!
📊 Analyzed 1490 articles

📝 Sample sentiment results:
   category  full_sentiment full_label
```

```
0  business         -0.9701   negative
1  business          0.7623   positive
2  business         -0.9318   negative
3      tech          0.9554   positive
4  business         -0.9486   negative
```

```
# Analyze sentiment patterns by category
print("📊 SENTIMENT ANALYSIS BY CATEGORY")
print("=" * 50)

# Calculate sentiment statistics by category
sentiment_by_category = sentiment_df.groupby('category').agg({
    'full_sentiment': ['mean', 'std', 'min', 'max'],
    'pos_score': 'mean',
    'neu_score': 'mean',
    'neg_score': 'mean'
}).round(4)

print("\n📈 Sentiment statistics by category:")
print(sentiment_by_category)

# Sentiment distribution by category
sentiment_dist = sentiment_df.groupby(['category', 'full_label']).size().unstack(fill_value=0)
sentiment_dist_pct = sentiment_dist.div(sentiment_dist.sum(axis=1), axis=0) * 100

print("\n📊 Sentiment distribution (%) by category:")
print(sentiment_dist_pct.round(2))

# Create visualizations
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# 1. Sentiment scores by category
sns.boxplot(data=sentiment_df, x='category', y='full_sentiment', ax=axes[0,0])
axes[0,0].set_title('Sentiment Score Distribution by Category')
axes[0,0].tick_params(axis='x', rotation=45)

# 2. Sentiment label distribution
sentiment_dist_pct.plot(kind='bar', ax=axes[0,1], stacked=True)
axes[0,1].set_title('Sentiment Label Distribution by Category (%)')
axes[0,1].tick_params(axis='x', rotation=45)
axes[0,1].legend(title='Sentiment')

# 3. Positive vs Negative scores
category_means = sentiment_df.groupby('category')[['pos_score', 'neg_score']].mean()
category_means.plot(kind='bar', ax=axes[1,0])
axes[1,0].set_title('Average Positive vs Negative Scores by Category')
axes[1,0].tick_params(axis='x', rotation=45)
axes[1,0].legend(['Positive', 'Negative'])

# 4. Sentiment vs Category heatmap
sentiment_pivot = sentiment_df.pivot_table(values='full_sentiment', index='category',
                                           columns='full_label', aggfunc='count', fill_value=0)
sns.heatmap(sentiment_pivot, annot=True, fmt='d', ax=axes[1,1], cmap='YlOrRd')
axes[1,1].set_title('Sentiment Count Heatmap')

plt.tight_layout()
plt.show()

# 💡 STUDENT TASK: Analyze sentiment patterns
# - Which categories are most positive/negative?
# - Are there differences between title and content sentiment?
# - How does sentiment vary within categories?
# - Can sentiment be used as a feature for classification?
```

🏷️ Text Classification System

🎯 Module 7: Building the News Classifier

Now we'll build the core of our NewsBot system - a multi-class text classifier that can automatically categorize news articles. We'll compare different algorithms and evaluate their performance.

Classification Pipeline:

1. **Feature Engineering:** Combine TF-IDF with other features
2. **Model Training:** Train multiple algorithms

3. **Model Evaluation:** Compare performance metrics
4. **Model Selection:** Choose the best performing model

💡 **Business Impact:** Accurate classification enables automatic content routing, personalized recommendations, and efficient content management.

```
# Prepare features for classification
print("🔧 Preparing features for classification...")

# 💡 TIP: Combine multiple feature types for better performance
# - TF-IDF features (most important)
# - Sentiment features
# - Text length features
# - POS features (if available)

# Create feature matrix
X_tfidf = tfidf_matrix.toarray()  # TF-IDF features

# Add sentiment features
sentiment_features = sentiment_df[['full_sentiment', 'pos_score', 'neu_score', 'neg_score']].values

# Add text length features
# Use df_final for text length calculations
length_features = np.array([
    df_final['full_text'].str.len(),  # Character length
    df_final['full_text'].str.split().str.len(),  # Word count
    # Removed 'title' length as 'title' column does not exist
]).T

# 🚀 YOUR CODE HERE: Combine all features
X_combined = np.hstack([
    X_tfidf,
    sentiment_features,
    length_features
])

# Target variable - use 'category' from df_final
y = df_final['category'].values

print(f"✅ Feature matrix prepared!")
print(f"📊 Feature matrix shape: {X_combined.shape}")
print(f"🎯 Number of classes: {len(np.unique(y))}")
print(f"📋 Classes: {np.unique(y)}")

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X_combined, y, test_size=0.2, random_state=42, stratify=y
)

print(f"\n📈 Data split:")
print(f"  Training set: {X_train.shape[0]} samples")
print(f"  Test set: {X_test.shape[0]} samples")
```

```
🔧 Preparing features for classification...
✅ Feature matrix prepared!
📊 Feature matrix shape: (1490, 5006)
🎯 Number of classes: 5
📋 Classes: ['business' 'entertainment' 'politics' 'sport' 'tech']

📈 Data split:
  Training set: 1192 samples
  Test set: 298 samples
```

```
# Train and evaluate multiple classifiers
print("🚀 Training multiple classifiers...")

# Define classifiers to compare
classifiers = {

    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'SVM': SVC(random_state=42, probability=True)  # Enable probability for better analysis
}

# 💡 TIP: For larger datasets, you might want to use SGDClassifier for efficiency
# from sklearn.linear_model import SGDClassifier
# classifiers['SGD'] = SGDClassifier(random_state=42)
```

```
# Train and evaluate each classifier
results = {}
trained_models = {}

for name, classifier in classifiers.items():
    print(f"\n🔄 Training {name}...")

    # 🚀 YOUR CODE HERE: Train and evaluate classifier
    # Train the model
    classifier.fit(X_train, y_train)

    # Make predictions
    y_pred = classifier.predict(X_test)
    y_pred_proba = classifier.predict_proba(X_test) if hasattr(classifier, 'predict_proba') else None

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)

    # Cross-validation score
    cv_scores = cross_val_score(classifier, X_train, y_train, cv=3, scoring='accuracy')

    # Store results
    results[name] = {
        'accuracy': accuracy,
        'cv_mean': cv_scores.mean(),
        'cv_std': cv_scores.std(),
        'predictions': y_pred,
        'probabilities': y_pred_proba
    }

    trained_models[name] = classifier

    print(f"   ✅ Accuracy: {accuracy:.4f}")
    print(f"   📊 CV Score: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})")

print("\n⏳ CLASSIFIER COMPARISON")
print("=" * 50)
comparison_df = pd.DataFrame({
    'Model': list(results.keys()),
    'Test Accuracy': [results[name]['accuracy'] for name in results.keys()],
    'CV Mean': [results[name]['cv_mean'] for name in results.keys()],
    'CV Std': [results[name]['cv_std'] for name in results.keys()]
})

print(comparison_df.round(4))

# Find best model
best_model_name = comparison_df.loc[comparison_df['Test Accuracy'].idxmax(), 'Model']
print(f"\n🏅 Best performing model: {best_model_name}")
```

```
🤖 Training multiple classifiers...

🔄 Training Logistic Regression...
   ✅ Accuracy: 0.6611
   📊 CV Score: 0.7231 (+/- 0.0799)

🔄 Training SVM...
   ✅ Accuracy: 0.3557
   📊 CV Score: 0.3582 (+/- 0.0363)

⏳ CLASSIFIER COMPARISON
==================================================
                Model  Test Accuracy  CV Mean  CV Std
0  Logistic Regression         0.6611   0.7231  0.0399
1                  SVM         0.3557   0.3582  0.0182

🏅 Best performing model: Logistic Regression
```

```
# Detailed evaluation of the best model
best_model = trained_models[best_model_name]
best_predictions = results[best_model_name]['predictions']

print(f"📊 DETAILED EVALUATION: {best_model_name}")
print("=" * 60)

# Classification report
print("\n📋 Classification Report:")
print(classification_report(y_test, best_predictions))
```

```
print(classification_report(y_test, best_predictions))

# Confusion matrix
cm = confusion_matrix(y_test, best_predictions)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.title(f'Confusion Matrix - {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()

# Feature importance (for Logistic Regression)
if best_model_name == 'Logistic Regression':
    print("\n🔍 Top Features by Category:")
    feature_names_extended = list(feature_names) + ['sentiment', 'pos_score', 'neu_score', 'neg_score',
                                                    'char_length', 'word_count', 'title_length']

    classes = best_model.classes_
    coefficients = best_model.coef_

    for i, class_name in enumerate(classes):
        top_indices = np.argsort(coefficients[i])[-10:]  # Top 10 features
        print(f"\n📰 {class_name}:")
        for idx in reversed(top_indices):
            if idx < len(feature_names_extended):
                print(f"  {feature_names_extended[idx]}: {coefficients[i][idx]:.4f}")

# 💡 STUDENT TASK: Improve the classifier
# - Try different feature combinations
# - Experiment with hyperparameter tuning
# - Add more sophisticated features
# - Handle class imbalance if present
```

🔍 Named Entity Recognition

🎯 Module 8: Extracting Facts from News

Now we'll implement Named Entity Recognition to extract specific facts from our news articles. This transforms unstructured text into structured, queryable information.

NER Applications:

- **Entity Tracking:** Monitor mentions of people, organizations, locations
- **Fact Extraction:** Build knowledge bases from news content
- **Relationship Mapping:** Understand connections between entities
- **Timeline Construction:** Track events and their participants

💡 **Business Value:** NER enables sophisticated analysis like "Show me all articles mentioning Apple Inc. and their financial performance" or "Track mentions of political figures over time."

```
📊 DETAILED EVALUATION: Logistic Regression
==================================================

📋 Classification Report:
              precision    recall  f1-score   support

    business       0.56      0.81      0.66        67
entertainment       0.84      0.67      0.75        55
    politics       0.61      0.47      0.54        55
       sport       0.66      0.78      0.72        69
        tech       0.76      0.50      0.60        52

    accuracy                           0.66       298
   macro avg       0.69      0.65      0.65       298
weighted avg       0.68      0.66      0.66       298
```

```
def extract_entities(text):
    """
    Extract named entities using spaCy

    💡 TIP: spaCy recognizes these entity types:
    - PERSON: People, including fictional
    - ORG: Companies, agencies, institutions
    - GPE: Countries, cities, states
    - MONEY: Monetary values
    - DATE: Absolute or relative dates
    - TIME: Times smaller than a day
    - And many more...
    """
    if not text or pd.isna(text):
        return []

    # 🚀 YOUR CODE HERE: Implement entity extraction
    doc = nlp(str(text))

    entities = []
    for ent in doc.ents:
        entities.append({
            'text': ent.text,
            'label': ent.label_,
            'start': ent.start_char,
            'end': ent.end_char,
            'description': spacy.explain(ent.label_)
        })

    return entities

# Apply NER to all articles
print("🔍 Extracting named entities...")

all_entities = []
article_entities = []

# Iterate over df_final instead of df
for idx, row in df_final.iterrows():
    entities = extract_entities(row['full_text']) # Use 'full_text' from df_final

    # Store entities for this article
    article_entities.append({
        'article_id': row['article_id'], # Use 'article_id' from df_final
        'category': row['category'], # Use 'category' from df_final
        'entities': entities,
        'entity_count': len(entities)
    })

    # Add to global entity list
    for entity in entities:
        entity['article_id'] = row['article_id'] # Use 'article_id' from df_final
        entity['category'] = row['category'] # Use 'category' from df_final
        all_entities.append(entity)

print(f"✅ Entity extraction complete!")
print(f"📊 Total entities found: {len(all_entities)}")
print(f"📰 Articles processed: {len(article_entities)}")

# Convert to DataFrame for analysis
entities_df = pd.DataFrame(all_entities)

if not entities_df.empty:
```

```
    print(f"\n🏈 Entity types found: {entities_df['label'].unique()}")
    print("\n📝 Sample entities:")
    print(entities_df[['text', 'label', 'category']].head(10))
else:
    print("⚠️ No entities found. This might happen with very short sample texts.")
```

```
🔍 Extracting named entities...
✅ Entity extraction complete!
📊 Total entities found: 42031
📰 Articles processed: 1490
🏈 Entity types found: ['ORDINAL' 'PERSON' 'GPE' 'DATE' 'MONEY' 'ORG' 'NORP' 'LOC' 'CARDINAL'
 'PERCENT' 'TIME' 'EVENT' 'QUANTITY' 'FAC' 'PRODUCT' 'LANGUAGE'
 'WORK_OF_ART' 'LAW']

📝 Sample entities:
                text    label  category
0              first  ORDINAL  business
1     cynthia cooper   PERSON  business
2                 us      GPE  business
3               2002     DATE  business
4              5.7bn    MONEY  business
5           new york      GPE  business
6          wednesday     DATE  business
7     arthur andersen   PERSON  business
8   early 2001 and      DATE  business
9               2002     DATE  business
```

```
# Analyze entity patterns
if not entities_df.empty:
    print("📊 NAMED ENTITY ANALYSIS")
    print("=" * 50)

    # Entity type distribution
    entity_counts = entities_df['label'].value_counts()
    print("\n🏈 Entity type distribution:")
    print(entity_counts)

    # Entity types by category
    entity_by_category = entities_df.groupby(['category', 'label']).size().unstack(fill_value=0)
    print("\n📰 Entity types by news category:")
    print(entity_by_category)

    # Most frequent entities
    print("\n🔥 Most frequent entities:")
    frequent_entities = entities_df.groupby(['text', 'label']).size().sort_values(ascending=False).head(15)
    for (entity, label), count in frequent_entities.items():
        print(f"  {entity} ({label}): {count} mentions")

    # Visualizations
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))

    # 1. Entity type distribution
    entity_counts.plot(kind='bar', ax=axes[0,0])
    axes[0,0].set_title('Entity Type Distribution')
    axes[0,0].tick_params(axis='x', rotation=45)

    # 2. Entities per category
    entities_per_category = entities_df.groupby('category').size()
    entities_per_category.plot(kind='bar', ax=axes[0,1])
    axes[0,1].set_title('Total Entities per Category')
    axes[0,1].tick_params(axis='x', rotation=45)

    # 3. Entity type heatmap by category
    if entity_by_category.shape[0] > 1 and entity_by_category.shape[1] > 1:
        sns.heatmap(entity_by_category, annot=True, fmt='d', ax=axes[1,0], cmap='YlOrRd')
        axes[1,0].set_title('Entity Types by Category Heatmap')
    else:
        axes[1,0].text(0.5, 0.5, 'Insufficient data\nfor heatmap',
                       ha='center', va='center', transform=axes[1,0].transAxes)
        axes[1,0].set_title('Entity Types by Category')

    # 4. Top entities
    top_entities = entities_df['text'].value_counts().head(10)
    top_entities.plot(kind='barh', ax=axes[1,1])
    axes[1,1].set_title('Most Mentioned Entities')

    plt.tight_layout()
    plt.show()
```

```
    plt.show()

    # 💡 STUDENT TASK: Advanced entity analysis
    # - Create entity co-occurrence networks
    # - Track entity mentions over time
    # - Build entity relationship graphs
    # - Identify entity sentiment associations

else:
    print("⚠️ Skipping entity analysis due to insufficient data.")
    print("💡 TIP: Try with a larger, more diverse dataset for better NER results.")
```

📈 Comprehensive Analysis and Insights

🎯 Bringing It All Together

Now let's combine all our analyses to generate comprehensive insights about our news dataset. This is where the real business value emerges from our NLP pipeline.

Key Analysis Areas:

1. **Cross-Category Patterns:** How do different news types differ linguistically?
2. **Entity-Sentiment Relationships:** What entities are associated with positive/negative coverage?
3. **Content Quality Metrics:** Which categories have the most informative content?
4. **Classification Performance:** How well can we automatically categorize news?

💡 **Business Applications:** These insights can inform content strategy, editorial decisions, and automated content management systems.

```
# Create comprehensive analysis dashboard
def create_comprehensive_analysis():
    """
    Generate comprehensive insights combining all analyses

    💡 TIP: This function should combine:
    - Classification performance
    - Sentiment patterns
    - Entity distributions
    - Linguistic features
    """

    insights = {
        'dataset_overview': {},
        'classification_performance': {},
        'sentiment_insights': {},
        'entity_insights': {},
        'linguistic_patterns': {}
```

```
            'linguistic_patterns': {},
            'business_recommendations': []
        }

        # 🚀 YOUR CODE HERE: Generate comprehensive insights

        # Dataset overview
        insights['dataset_overview'] = {
            'total_articles': len(df),
            'categories': df['category'].unique().tolist(),
            'category_distribution': df['category'].value_counts().to_dict(),
            'avg_article_length': df['full_text'].str.len().mean(),
            'avg_words_per_article': df['full_text'].str.split().str.len().mean()
        }

        # Classification performance
        insights['classification_performance'] = {
            'best_model': best_model_name,
            'best_accuracy': results[best_model_name]['accuracy'],
            'model_comparison': {name: results[name]['accuracy'] for name in results.keys()}
        }

        # Sentiment insights
        sentiment_by_cat = sentiment_df.groupby('category')['full_sentiment'].mean().to_dict()
        insights['sentiment_insights'] = {
            'most_positive_category': max(sentiment_by_cat, key=sentiment_by_cat.get),
            'most_negative_category': min(sentiment_by_cat, key=sentiment_by_cat.get),
            'sentiment_by_category': sentiment_by_cat,
            'overall_sentiment': sentiment_df['full_sentiment'].mean()
        }

        # Entity insights
        if not entities_df.empty:
            entity_by_cat = entities_df.groupby('category').size().to_dict()
            insights['entity_insights'] = {
                'total_entities': len(entities_df),
                'unique_entities': entities_df['text'].nunique(),
                'entity_types': entities_df['label'].unique().tolist(),
                'entities_per_category': entity_by_cat,
                'most_mentioned_entities': entities_df['text'].value_counts().head(5).to_dict()
            }

        # Generate business recommendations
        recommendations = []

        # Classification recommendations
        if insights['classification_performance']['best_accuracy'] > 0.8:
            recommendations.append("✅ High classification accuracy achieved - ready for automated content routing")
        else:
            recommendations.append("⚠️ Classification accuracy needs improvement - consider more training data or feature engineerin

        # Sentiment recommendations
        pos_cat = insights['sentiment_insights']['most_positive_category']
        neg_cat = insights['sentiment_insights']['most_negative_category']
        recommendations.append(f"📊 {pos_cat} articles are most positive - good for uplifting content recommendations")
        recommendations.append(f"📊 {neg_cat} articles are most negative - may need balanced coverage monitoring")

        # Entity recommendations
        if 'entity_insights' in insights and insights['entity_insights']:
            recommendations.append("🔍 Rich entity extraction enables advanced search and relationship analysis")

        insights['business_recommendations'] = recommendations

        return insights

    # Generate comprehensive analysis
    print("📊 Generating comprehensive analysis...")
    analysis_results = create_comprehensive_analysis()

    print("✅ Analysis complete!")
    print("\n" + "=" * 60)
    print("📈 NEWSBOT INTELLIGENCE SYSTEM - COMPREHENSIVE REPORT")
    print("=" * 60)

    # Display key insights
    print(f"\n📊 DATASET OVERVIEW:")
    overview = analysis_results['dataset_overview']
    print(f"  Total Articles: {overview['total_articles']}")
```

```
    print(f"  Categories: {', '.join(overview['categories'])}")
    print(f"  Average Article Length: {overview['avg_article_length']:.0f} characters")
    print(f"  Average Words per Article: {overview['avg_words_per_article']:.0f} words")

    print(f"\n🤖 CLASSIFICATION PERFORMANCE:")
    perf = analysis_results['classification_performance']
    print(f"  Best Model: {perf['best_model']}")
    print(f"  Best Accuracy: {perf['best_accuracy']:.4f}")

    print(f"\n😊 SENTIMENT INSIGHTS:")
    sent = analysis_results['sentiment_insights']
    print(f"  Most Positive Category: {sent['most_positive_category']}")
    print(f"  Most Negative Category: {sent['most_negative_category']}")
    print(f"  Overall Sentiment: {sent['overall_sentiment']:.4f}")

    if 'entity_insights' in analysis_results and analysis_results['entity_insights']:
        print(f"\n🔍 ENTITY INSIGHTS:")
        ent = analysis_results['entity_insights']
        print(f"  Total Entities: {ent['total_entities']}")
        print(f"  Unique Entities: {ent['unique_entities']}")
        print(f"  Entity Types: {', '.join(ent['entity_types'])}")

    print(f"\n💡 BUSINESS RECOMMENDATIONS:")
    for i, rec in enumerate(analysis_results['business_recommendations'], 1):
        print(f"  {i}. {rec}")
```

```
📊 Generating comprehensive analysis...
✅ Analysis complete!

============================================================
📈 NEWSBOT INTELLIGENCE SYSTEM - COMPREHENSIVE REPORT
============================================================

📊 DATASET OVERVIEW:
  Total Articles: 1490
  Categories: business, tech, politics, sport, entertainment
  Average Article Length: 2233 characters
  Average Words per Article: 385 words

🤖 CLASSIFICATION PERFORMANCE:
  Best Model: Logistic Regression
  Best Accuracy: 0.6611

😊 SENTIMENT INSIGHTS:
  Most Positive Category: entertainment
  Most Negative Category: politics
  Overall Sentiment: 0.3950

🔍 ENTITY INSIGHTS:
  Total Entities: 42031
  Unique Entities: 12071
  Entity Types: ORDINAL, PERSON, GPE, DATE, MONEY, ORG, NORP, LOC, CARDINAL, PERCENT, TIME, EVENT, QUANTITY, FAC, PRODUCT, LANGU

💡 BUSINESS RECOMMENDATIONS:
  1. ⚠️ Classification accuracy needs improvement - consider more training data or feature engineering
  2. 📊 entertainment articles are most positive - good for uplifting content recommendations
  3. 📊 politics articles are most negative - may need balanced coverage monitoring
  4. 🔍 Rich entity extraction enables advanced search and relationship analysis
```

🚀 Final System Integration

🎯 Building the Complete NewsBot Pipeline

Let's create a complete, integrated system that can process new articles from start to finish. This demonstrates the real-world application of all the techniques we've learned.

Complete Pipeline:

1. **Text Preprocessing:** Clean and normalize input
2. **Feature Extraction:** Generate TF-IDF and other features
3. **Classification:** Predict article category
4. **Entity Extraction:** Identify key facts
5. **Sentiment Analysis:** Determine emotional tone
6. **Insight Generation:** Provide actionable intelligence

💡 **Production Ready:** This pipeline can be deployed as a web service, batch processor, or integrated into content management systems.

✦ Gemini

```
class NewsBotIntelligenceSystem:
    """
    Complete NewsBot Intelligence System


    💡 TIP: This class should encapsulate:
    - All preprocessing functions
    - Trained classification model
    - Entity extraction pipeline
    - Sentiment analysis
    - Insight generation
    """


    def __init__(self, classifier, vectorizer, sentiment_analyzer):
        self.classifier = classifier
        self.vectorizer = vectorizer
        self.sentiment_analyzer = sentiment_analyzer
        self.nlp = nlp  # spaCy model


    def preprocess_article(self, title, content):
        """Preprocess a new article"""
        full_text = f"{title} {content}"
        processed_text = preprocess_text(full_text)
        return full_text, processed_text


    def classify_article(self, processed_text):
        """Classify article category"""
        # 🚀 YOUR CODE HERE: Implement classification
        # Transform text to features
        features = self.vectorizer.transform([processed_text])


        # Add dummy features for sentiment and length (in production, calculat
        dummy_features = np.zeros((1, 7))  # 4 sentiment + 3 length features
        # Need 4 sentiment features + 2 length features = 6 features
        dummy_features = np.zeros((1, 6))  # Corrected shape to match training
        features_combined = np.hstack([features.toarray(), dummy_features])


        # Predict category and probability
        prediction = self.classifier.predict(features_combined)[0]
        probabilities = self.classifier.predict_proba(features_combined)[0]


        # Get class probabilities
        class_probs = dict(zip(self.classifier.classes_, probabilities))


        return prediction, class_probs


    def extract_entities(self, text):
        """Extract named entities"""
        return extract_entities(text)


    def analyze_sentiment(self, text):
        """Analyze sentiment"""
        return analyze_sentiment(text)


    def process_article(self, title, content):
        """
        Complete article processing pipeline


        💡 TIP: This should return a comprehensive analysis including:
        - Predicted category with confidence
        - Extracted entities
        - Sentiment analysis
        - Key insights and recommendations
```

```
        """
        # 🚀 YOUR CODE HERE: Implement complete pipeline


        # Step 1: Preprocess
        full_text, processed_text = self.preprocess_article(title, content)


        # Step 2: Classify
        category, category_probs = self.classify_article(processed_text)


        # Step 3: Extract entities
        entities = self.extract_entities(full_text)


        # Step 4: Analyze sentiment
        sentiment = self.analyze_sentiment(full_text)


        # Step 5: Generate insights
        insights = self.generate_insights(category, entities, sentiment, categ


        return {
            'title': title,
            'content': content[:200] + '...' if len(content) > 200 else conte
            'predicted_category': category,
            'category_confidence': max(category_probs.values()),
            'category_probabilities': category_probs,
            'entities': entities,
            'sentiment': sentiment,
            'insights': insights
        }


    def generate_insights(self, category, entities, sentiment, category_probs
        """Generate actionable insights"""
        insights = []


        # Classification insights
        confidence = max(category_probs.values())
        if confidence > 0.8:
            insights.append(f"✅ High confidence {category} classification ({
        else:
            insights.append(f"⚠️ Uncertain classification - consider manual r


        # Sentiment insights
        if sentiment['compound'] > 0.1:
            insights.append(f"😊 Positive sentiment detected ({sentiment['com
        elif sentiment['compound'] < -0.1:
            insights.append(f"😞 Negative sentiment detected ({sentiment['com
        else:
            insights.append(f"😐 Neutral sentiment ({sentiment['compound']:.3


        # Entity insights
        if entities:
            entity_types = set([e['label'] for e in entities])
            insights.append(f"🔍 Found {len(entities)} entities of {len(entit


            # Highlight important entities
            important_entities = [e for e in entities if e['label'] in ['PERS(
            if important_entities:
                key_entities = [e['text'] for e in important_entities[:3]]
                insights.append(f"🎯 Key entities: {', '.join(key_entities)}"
        else:
            insights.append("ℹ️ No named entities detected")


        return insights

# Initialize the complete system
newsbot = NewsBotIntelligenceSystem(
    classifier=best_model
```

```
    vectorizer=tfidf_vectorizer,
    sentiment_analyzer=sia
)

print("🤖 NewsBot Intelligence System initialized!")
print("✅ Ready to process new articles")
```

```
🤖 NewsBot Intelligence System initialized!
✅ Ready to process new articles
```

```
# Test the complete system with new articles
print("🧪 TESTING NEWSBOT INTELLIGENCE SYSTEM")
print("=" * 60)

# Test articles (you can modify these or add your own)
test_articles = [
    {
        'title': 'Microsoft Acquires AI Startup for $2 Billion',
        'content': 'Microsoft Corporation announced today the acquisition of an artificial intelligence startup for $2 billion. (
    },
    {
        'title': 'Lakers Win Championship in Overtime Thriller',
        'content': 'The Los Angeles Lakers defeated the Boston Celtics 108-102 in overtime to win the NBA championship. LeBron Ja
    },
    {
        'title': 'New Climate Change Report Shows Alarming Trends',
        'content': 'Scientists at the United Nations released a comprehensive climate report showing accelerating global warming
    }
]

# Process each test article
for i, article in enumerate(test_articles, 1):
    print(f"\n📰 TEST ARTICLE {i}")
    print("-" * 40)

    # Process the article
    result = newsbot.process_article(article['title'], article['content'])

    # Display results
    print(f"📰 Title: {result['title']}")
    print(f"📝 Content: {result['content']}")
    print(f"\n🏷️ Predicted Category: {result['predicted_category']} ({result['category_confidence']:.2%} confidence)")

    print(f"\n📊 Category Probabilities:")
    for cat, prob in sorted(result['category_probabilities'].items(), key=lambda x: x[1], reverse=True):
        print(f"  {cat}: {prob:.3f}")

    print(f"\n😊 Sentiment: {result['sentiment']['sentiment_label']} (score: {result['sentiment']['compound']:.3f})")

    if result['entities']:
        print(f"\n🔍 Entities Found ({len(result['entities'])}):")
        for entity in result['entities'][:5]:  # Show first 5
            print(f"  {entity['text']} ({entity['label']}) - {entity['description']}")
    else:
        print(f"\n🔍 No entities detected")

    print(f"\n💡 Insights:")
    for insight in result['insights']:
        print(f"  {insight}")

print("\n" + "=" * 60)
print("🎉 NewsBot Intelligence System testing complete!")
print("✅ System successfully processed all test articles")

# 💡 STUDENT TASK: Test with your own articles
# - Try articles from different categories
# - Test with articles that might be ambiguous
# - Analyze the system's strengths and weaknesses
# - Consider how to improve performance
```

```
🧪 TESTING NEWSBOT INTELLIGENCE SYSTEM
============================================================

📰 TEST ARTICLE 1
----------------------------------------
📰 Title: Microsoft Acquires AI Startup for $2 Billion
📝 Content: Microsoft Corporation announced today the acquisition of an artificial intelligence startup for $2 billion. CEO Sa
```