🤖 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.

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("gpreda/bbc-news")

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/gpreda/bbc-news?dataset_version_number=1007...
100%|██████████| 3.64M/3.64M [00:00<00:00, 63.5MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/gpreda/bbc-news/versions/1007
```

```
# 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: spacy in /usr/local/lib/python3.11/dist-packages (3.8.7)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: wordcloud in /usr/local/lib/python3.11/dist-packages (1.9.4)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (5.24.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.12)
```

```
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.13)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.11)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.10)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (8.3.6)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.5.1)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.16.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (4.67.1)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.11.7)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from spacy) (75.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (25.0)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.5.0)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.16.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly) (8.5.0)
Requirement already satisfied: language-data>=1.2 in /usr/local/lib/python3.11/dist-packages (from langcodes<4.0.0,>=3.2.0->spacy) (1.
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spac
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (20
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (1.3.0)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (13.9.4)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spacy
```

```
# 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']}")
```

```
📚 All libraries imported successfully!
🔧 spaCy model loaded: core_web_sm v3.8.0
```

📊 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.

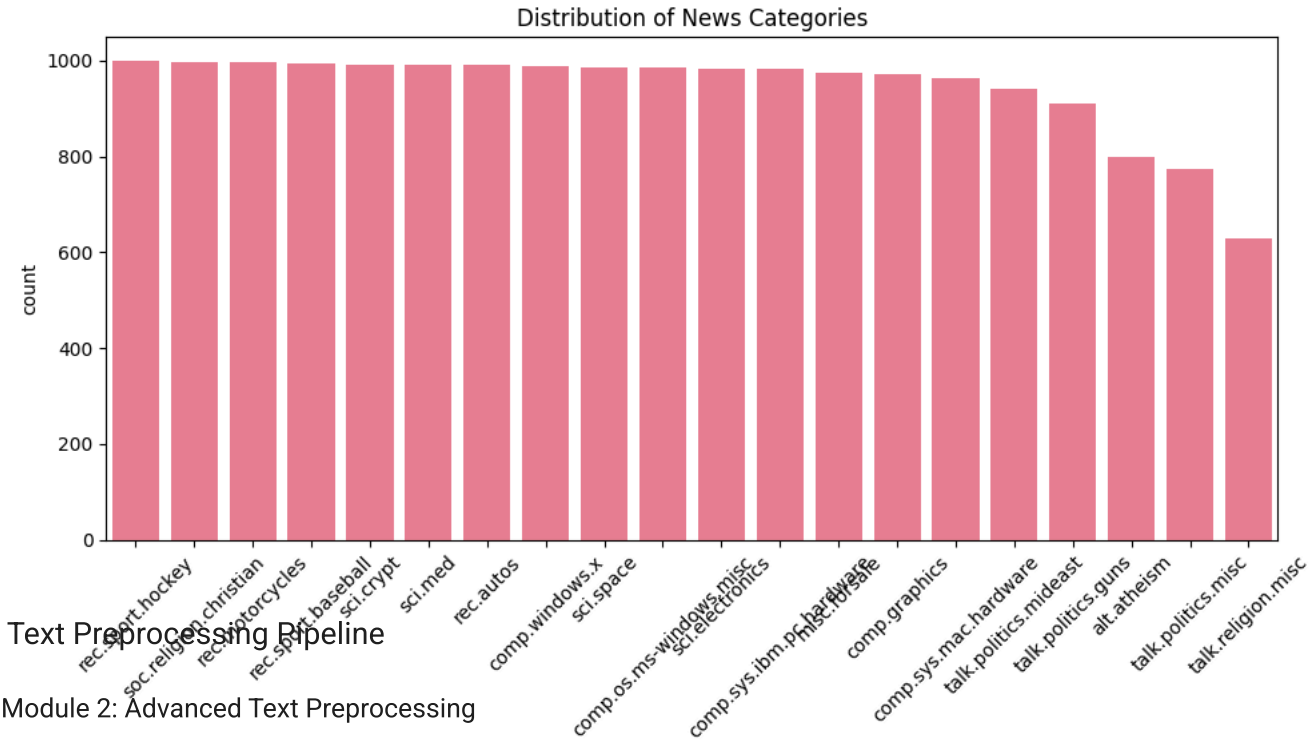
```
# Step 1: Install dependencies
!pip install scikit-learn --quiet

# Step 2: Load 20 Newsgroups Dataset from sklearn
from sklearn.datasets import fetch_20newsgroups
import pandas as pd

# Fetch both train and test to maximize data size
newsgroups_train = fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'))
newsgroups_test = fetch_20newsgroups(subset='test', remove=('headers', 'footers', 'quotes'))

# Combine train and test
data = newsgroups_train.data + newsgroups_test.data
targets = newsgroups_train.target.tolist() + newsgroups_test.target.tolist()
target_names = newsgroups_train.target_names

# Build DataFrame
df = pd.DataFrame({
    'article_id': range(1, len(data) + 1),
    'title': [f"Article {i}" for i in range(1, len(data) + 1)],
    'content': data,
    'category': [target_names[i] for i in targets],
    'date': ['2025-01-01'] * len(data),
    'source': ['20Newsgroups'] * len(data)
})

# Only keep relevant columns if needed
df = df[['article_id', 'title', 'content', 'category', 'date', 'source']]

# Show summary
print("✅ Dataset loaded successfully!")
print(f"📈 Shape: {df.shape}")
print(f"📋 Categories: {df['category'].nunique()} unique categories")
print(f"🔢 Example categories: {df['category'].unique()[:5]}")

# Preview
df.head()
```

```
✅ Dataset loaded successfully!
📈 Shape: (18846, 6)
📋 Categories: 20 unique categories
🔢 Example categories: ['rec.autos' 'comp.sys.mac.hardware' 'comp.graphics' 'sci.space'
 'talk.politics.guns']
```

	article_id	title	content	category	date	source
0	1	Article 1	I was wondering if anyone out there could enli...	rec.autos	2025-01-01	20Newsgroups
1	2	Article 2	A fair number of brave souls who upgraded thei...	comp.sys.mac.hardware	2025-01-01	20Newsgroups
2	3	Article 3	well folks, my mac plus finally gave up the gh...	comp.sys.mac.hardware	2025-01-01	20Newsgroups
3	4	Article 4	\nDo you have Weitek's address/phone number? ...	comp.graphics	2025-01-01	20Newsgroups
4	5	Article 5	From article <C5owCB.n3p@world.std.com>, by to...	sci.space	2025-01-01	20Newsgroups

```
import matplotlib.pyplot as plt
import seaborn as sns
# Basic dataset exploration
print("📊 DATASET OVERVIEW")
print("=" * 50)
print(f"Total articles: {len(df)}")
print(f"Unique categories: {df['category'].nunique()}")
print(f"Categories: {df['category'].unique().tolist()}")
print(f"Date range: {df['date'].min()} to {df['date'].max()}")
print(f"Unique sources: {df['source'].nunique()}")

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

# Visualize category distribution
plt.figure(figsize=(10, 6))
sns.countplot(data=df, 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
# - Analyze text length distribution
# - Examine source distribution
# - Look for any data quality issues
```

```
📊 DATASET OVERVIEW
==================================================
Total articles: 18846
Unique categories: 20
Categories: ['rec.autos', 'comp.sys.mac.hardware', 'comp.graphics', 'sci.space', 'talk.politics.guns', 'sci.med', 'comp.sys.ibm.pc.hardw
Date range: 2025-01-01 to 2025-01-01
Unique sources: 1

📈 CATEGORY DISTRIBUTION
==================================================
category
rec.sport.hockey            999
soc.religion.christian      997
rec.motorcycles             996
rec.sport.baseball          994
sci.crypt                   991
sci.med                     990
rec.autos                   990
comp.windows.x              988
sci.space                   987
comp.os.ms-windows.misc     985
sci.electronics             984
comp.sys.ibm.pc.hardware    982
misc.forsale                975
comp.graphics               973
comp.sys.mac.hardware       963
talk.politics.mideast       940
talk.politics.guns          910
alt.atheism                 799
talk.politics.misc          775
talk.religion.misc          628
Name: count, dtype: int64
```



🖌️ 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?

```
# Step 1: Install dependencies
!pip install scikit-learn --quiet
```

```
# Step 2: Download NLTK data
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Step 3: Import required libraries
from sklearn.datasets import fetch_20newsgroups
import pandas as pd
import re
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

# Step 4: Load 20 Newsgroups dataset
newsgroups_train = fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'))
newsgroups_test = fetch_20newsgroups(subset='test', remove=('headers', 'footers', 'quotes'))

# Combine train and test data
all_data = newsgroups_train.data + newsgroups_test.data
all_targets = newsgroups_train.target.tolist() + newsgroups_test.target.tolist()
target_names = newsgroups_train.target_names

# Step 5: Create a DataFrame
df = pd.DataFrame({
    'article_id': range(1, len(all_data) + 1),
    'title': [f"Article {i}" for i in range(1, len(all_data) + 1)],
    'content': all_data,
    'category': [target_names[i] for i in all_targets],
    'date': ['2025-01-01'] * len(all_data),
    'source': ['20Newsgroups'] * len(all_data)
})

# Optional: Just keep key columns
df = df[['article_id', 'title', 'content', 'category', 'date', 'source']]

print(f"✅ Loaded dataset with {len(df)} articles")
print(f"📋 Categories: {df['category'].nunique()} | Example: {df['category'].unique()[:5]}")
df.head()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
✅ Loaded dataset with 18846 articles
📋 Categories: 20 | Example: ['rec.autos' 'comp.sys.mac.hardware' 'comp.graphics' 'sci.space'
 'talk.politics.guns']
```

	article_id	title	content	category	date	source
0	1	Article 1	I was wondering if anyone out there could enli...	rec.autos	2025-01-01	20Newsgroups
1	2	Article 2	A fair number of brave souls who upgraded thei...	comp.sys.mac.hardware	2025-01-01	20Newsgroups
2	3	Article 3	well folks, my mac plus finally gave up the gh...	comp.sys.mac.hardware	2025-01-01	20Newsgroups
3	4	Article 4	\nDo you have Weitek's address/phone number? ...	comp.graphics	2025-01-01	20Newsgroups
4	5	Article 5	From article <C5owCB.n3p@world.std.com>, by to...	sci.space	2025-01-01	20Newsgroups

```
# Step 6: Set up text preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Text cleaning function
def clean_text(text):
    if pd.isna(text):
        return ""
    text = str(text).lower()
    text = re.sub(r'<[^>]+>', '', text)  # Remove HTML
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)  # Remove URLs
    text = re.sub(r'\S+@\S+', '', text)  # Remove emails
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters & digits
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces
    return text
```

```
# Full preprocessing function
def preprocess_text(text, remove_stopwords=True, lemmatize=True):
    text = clean_text(text)
    if not text:
        return ""
    tokens = word_tokenize(text)
    if remove_stopwords:
        tokens = [token for token in tokens if token not in stop_words]
    if lemmatize:
        tokens = [lemmatizer.lemmatize(token) for token in tokens]
    tokens = [token for token in tokens if len(token) > 2]
    return ' '.join(tokens)



# Step 7: Apply the preprocessing pipeline to the content column
import spacy # Import spacy
nlp = spacy.load('en_core_web_sm') # Load the spacy model and assign to nlp

# Include necessary imports and function definitions from jxYNCg7cwmH2 for preprocess_text
import nltk
import re
nltk.download('stopwords')
nltk.download('punkt')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

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

def clean_text(text):
    """
    Comprehensive text cleaning function
    """
    if pd.isna(text):
        return ""
    text = str(text).lower()
    text = re.sub(r'<[^>]+>', '', text)
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'\S+@\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def preprocess_text(text, remove_stopwords=True, lemmatize=True):
    """
    Complete preprocessing pipeline using spaCy for tokenization
    """
    text = clean_text(text)
    if not text:
        return ""
    # Use spaCy for tokenization
    doc = nlp(text) # Use the nlp object passed or defined in this cell
    tokens = [token.text for token in doc]
    if remove_stopwords:
        tokens = [token for token in tokens if token.lower() not in stop_words]
    if lemmatize:
        tokens = [lemmatizer.lemmatize(token) for token in tokens]
    tokens = [token for token in tokens if len(token) > 2]
    return ' '.join(tokens)


print("⚙️ Preprocessing content... This may take 1–2 minutes.")
# Pass the spaCy nlp object to the preprocess_text function
df['processed_content'] = df['content'].apply(lambda x: preprocess_text(x, nlp))


# Step 8: Preview the result
print("✅ Preprocessing complete.")
df[['content', 'processed_content']].head(3)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
⚙️ Preprocessing content... This may take 1–2 minutes.
✅ Preprocessing complete.
```

	content	processed_content
0	I was wondering if anyone out there could enli...	wondering anyone could enlighten car saw day d...
1	A fair number of brave souls who upgraded thei...	fair number brave soul upgraded clock oscillat...
2	well folks, my mac plus finally gave up the gh...	well folk mac plus finally gave ghost weekend ...

```
# Initialize preprocessing tools
import nltk
import re # Import the re module
nltk.download('stopwords') # Ensure stopwords are downloaded (still needed for stop words)
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
# Removed nltk.tokenize import and nltk.download('punkt') - Will use spaCy for tokenization
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Text cleaning function (remains the same)
def clean_text(text):
    """
    Comprehensive text cleaning function
    """
    if pd.isna(text):
        return ""

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

    # 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

# Full preprocessing function - now accepts spaCy nlp object for tokenization
def preprocess_text(text, nlp_model, remove_stopwords=True, lemmatize=True):
    """
    Complete preprocessing pipeline using spaCy for tokenization
    """
    # Clean text
    text = clean_text(text)

    if not text:
        return ""

    # Use spaCy for tokenization
    doc = nlp_model(text)
    tokens = [token.text for token in doc] # Get tokens from spaCy doc

    # Remove stop words if requested
    if remove_stopwords:
        # Use spaCy's is_stop attribute for efficiency
        tokens = [token.text for token in doc if not token.is_stop and not token.is_punct]


    # Lemmatize if requested
    if lemmatize:
        # Use spaCy's lemma_ attribute
        tokens = [token.lemma_ for token in doc if not token.is_stop and not token.is_punct]
```

```
            # After lemmatization, filter out stop words again as lemma_ might be a stop word
            tokens = [token for token in tokens if token not in stop_words]


        # Filter out very short words and empty strings that might result from cleaning/lemmatization
        tokens = [token for token in tokens if len(token) > 2]


        return ' '.join(tokens)

# Test the preprocessing function (requires passing the nlp object)
# Note: Need to have nlp object loaded from cell 6MrgHCeqwmH0
# sample_text = "Apple Inc. announced record quarterly earnings today! Visit https://apple.com for more info. #TechNews"
# print("Original text:")
# print(sample_text)
# # To test this function, you would need to load nlp first, e.g.:
# # import spacy
# # nlp = spacy.load('en_core_web_sm')
# # print("\nCleaned text:")
# # print(clean_text(sample_text))
# # print("\nFully preprocessed text:")
# # print(preprocess_text(sample_text, nlp))

print("✅ Preprocessing functions defined (using spaCy for tokenization)")
```

```
✅ Preprocessing functions defined (using spaCy for tokenization)
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# Apply preprocessing to the dataset
import spacy # Import spacy
nlp = spacy.load('en_core_web_sm') # Load the spacy model and assign to nlp

# Include necessary imports and function definitions for preprocessing
import nltk
import re
nltk.download('stopwords')
nltk.download('punkt')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

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

def clean_text(text):
    """
    Comprehensive text cleaning function
    """
    if pd.isna(text):
        return ""
    text = str(text).lower()
    text = re.sub(r'<[^>]+>', '', text)
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'\S+@\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def preprocess_text(text, nlp_model, remove_stopwords=True, lemmatize=True):
    """
    Complete preprocessing pipeline using spaCy for tokenization
    """
    text = clean_text(text)
    if not text:
        return ""
    # Use spaCy for tokenization
    doc = nlp_model(text)
    tokens = [token.text for token in doc]
    if remove_stopwords:
        tokens = [token for token in tokens if token.lower() not in stop_words]
    if lemmatize:
        tokens = [lemmatizer.lemmatize(token) for token in tokens]
    tokens = [token for token in tokens if len(token) > 2]
    return ' '.join(tokens)
```

```
print("🧹 Preprocessing all articles...")

# Create new columns for processed text, passing the nlp object
df['title_clean'] = df['title'].apply(clean_text)
df['content_clean'] = df['content'].apply(clean_text)
df['title_processed'] = df['title'].apply(lambda x: preprocess_text(x, nlp))
df['content_processed'] = df['content'].apply(lambda x: preprocess_text(x, nlp))

# Combine title and content for full article analysis
df['full_text'] = df['title'] + ' ' + df['content']
df['full_text_processed'] = df['full_text'].apply(lambda x: preprocess_text(x, nlp))

print("✅ Preprocessing complete!")

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

# 💡 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
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
🧹 Preprocessing all articles...
✅ Preprocessing complete!

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

Example 1:
Original: Article 1 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It...
Processed: article wondering anyone could enlighten car saw day door sport car looked late early called brickli...

Example 2:
Original: Article 2 A fair number of brave souls who upgraded their SI clock oscillator have
shared their expe...
Processed: article fair number brave soul upgraded clock oscillator shared experience poll please send brief me...

Example 3:
Original: Article 3 well folks, my mac plus finally gave up the ghost this weekend after
starting life as a 51...
Processed: article well folk mac plus finally gave ghost weekend starting life way back sooo market new machine...
```

📊 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.

```
from sklearn.feature_extraction.text import TfidfVectorizer # Import TfidfVectorizer

# 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['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['category'].values

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: (18846, 5000)
📝 Vocabulary size: 5000
🔢 Sparsity: 99.05%

🔍 Sample TF-IDF features:
   aaron  abc  ability  able  able get  abortion  absence  absolute  \
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
2    0.0  0.0      0.0   0.0       0.0       0.0      0.0       0.0

   absolutely  abstract
0         0.0       0.0
1         0.0       0.0
2         0.0       0.0
```

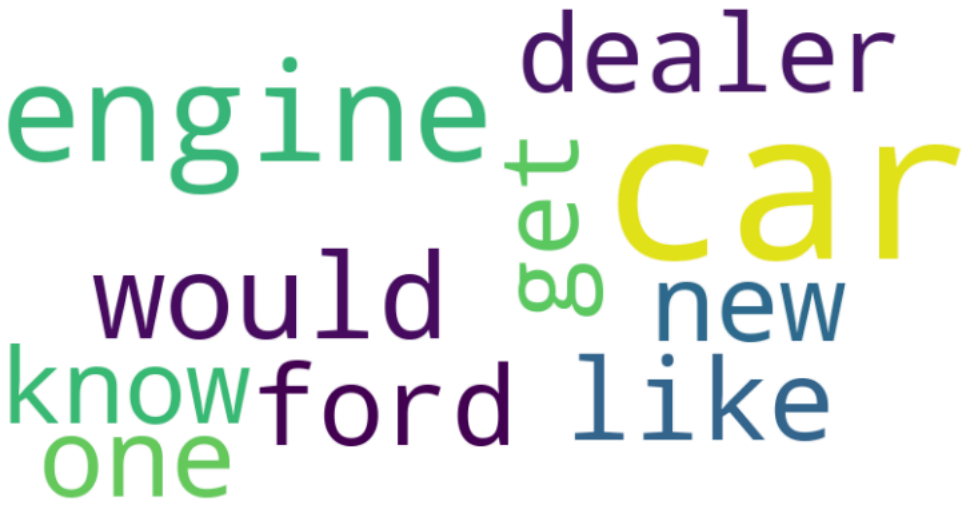
```
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import seaborn as sns

# Define the function to get top TF-IDF terms per category
def get_top_tfidf_terms(category, n_terms=10):
    """
    Get top TF-IDF terms for a specific category
    """
    category_data = tfidf_df[tfidf_df['category'] == category]

    # Drop the 'category' column before calculating the mean
    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['category'].unique()
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}")
```

```
baseball: 0.0295
```

```
          baseball: 0.0295
          pitcher: 0.0266
          hit: 0.0254
          run: 0.0220
          would: 0.0220
          last: 0.0215

        SOC.RELIGION.CHRISTIAN:
          god: 0.0834
          christian: 0.0484
          church: 0.0419
          jesus: 0.0388
          would: 0.0316
          christ: 0.0301
          one: 0.0300
          sin: 0.0289
          people: 0.0268
          bible: 0.0268

        TALK.POLITICS.MIDEAST:
          israel: 0.0574
          armenian: 0.0569
          arab: 0.0440
          israeli: 0.0426
          jew: 0.0407
          people: 0.0307
          muslim: 0.0298
          turkish: 0.0237
          palestinian: 0.0229
          jewish: 0.0229

        TALK.POLITICS.MISC:
          people: 0.0361
          would: 0.0284
          homosexual: 0.0265
          government: 0.0250
          one: 0.0209
          state: 0.0207
          tax: 0.0193
          gay: 0.0189
          right: 0.0181
          law: 0.0180

        SCI.CRYPT:
          key: 0.0723
          chip: 0.0450
          encryption: 0.0421
          clipper: 0.0406
          government: 0.0357
          would: 0.0319
          phone: 0.0284
          nsa: 0.0268
          algorithm: 0.0251
          system: 0.0242
```

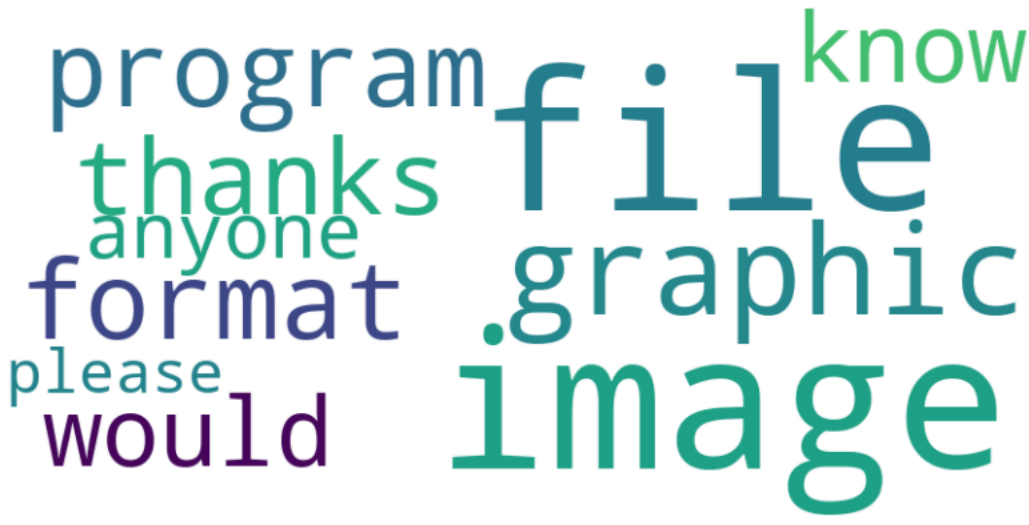
```
for category, terms in category_terms.items():
    wc = WordCloud(background_color='white', width=800, height=400)
    wc.generate_from_frequencies(terms)

    plt.figure(figsize=(10, 5))
    plt.imshow(wc, interpolation='bilinear')
    plt.axis('off')
    plt.title(f"Word Cloud for {category.upper()}")
    plt.show()
```

Word Cloud for REC.AUTOS



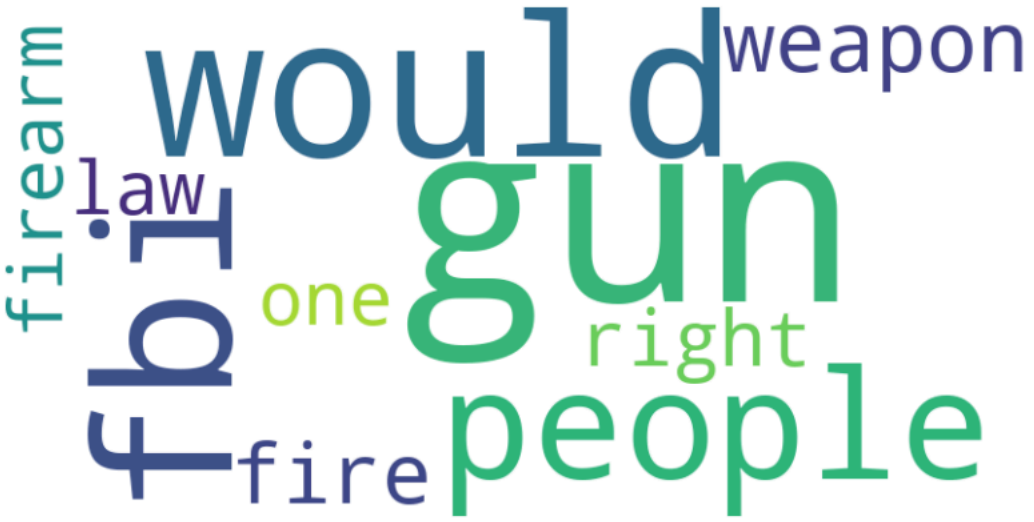
Word Cloud for COMP.SYS.MAC.HARDWARE

Word Cloud for COMP.GRAPHICS



Word Cloud for SCI.SPACE

Word Cloud for TALK.POLITICS.GUNS

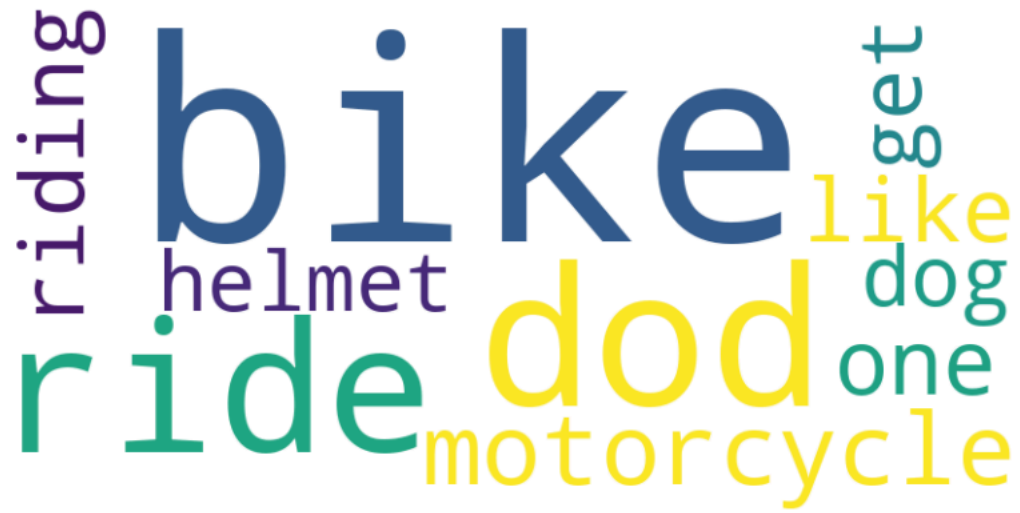


Word Cloud for SCI.MED

Word Cloud for COMP.SYS.IBM.PC.HARDWARE

Word Cloud for COMP.OS.MS-WINDOWS.MISC

Word Cloud for REC.MOTORCYCLES

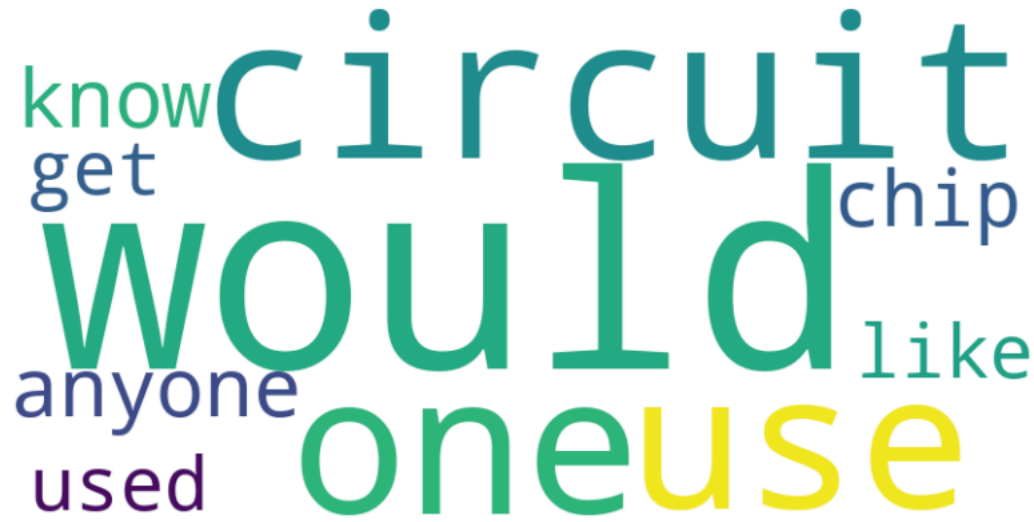


Word Cloud for TALK.RELIGION.MISC

Word Cloud for MISC.FORSALE

Word Cloud for ALT.ATHEISM

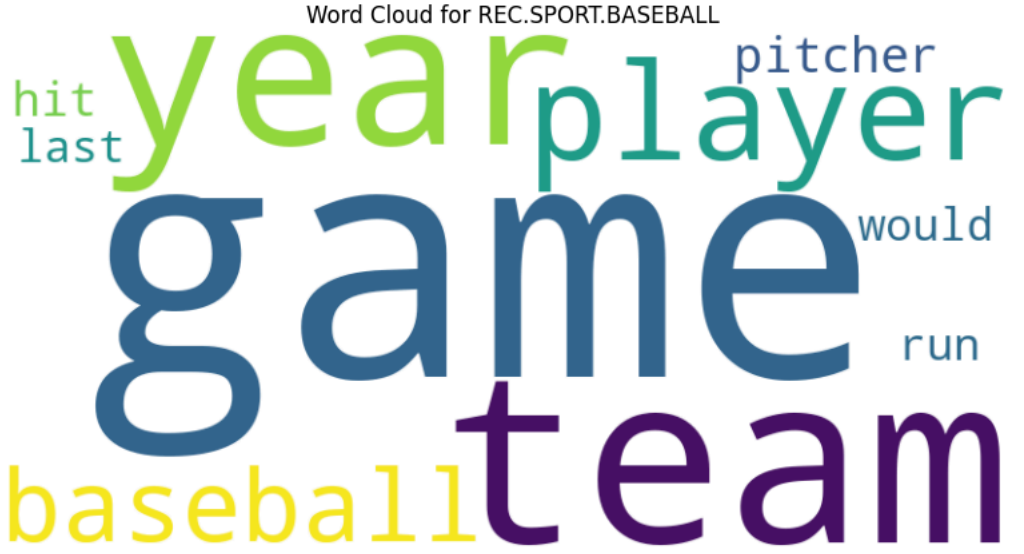
Word Cloud for SCI.ELECTRONICS



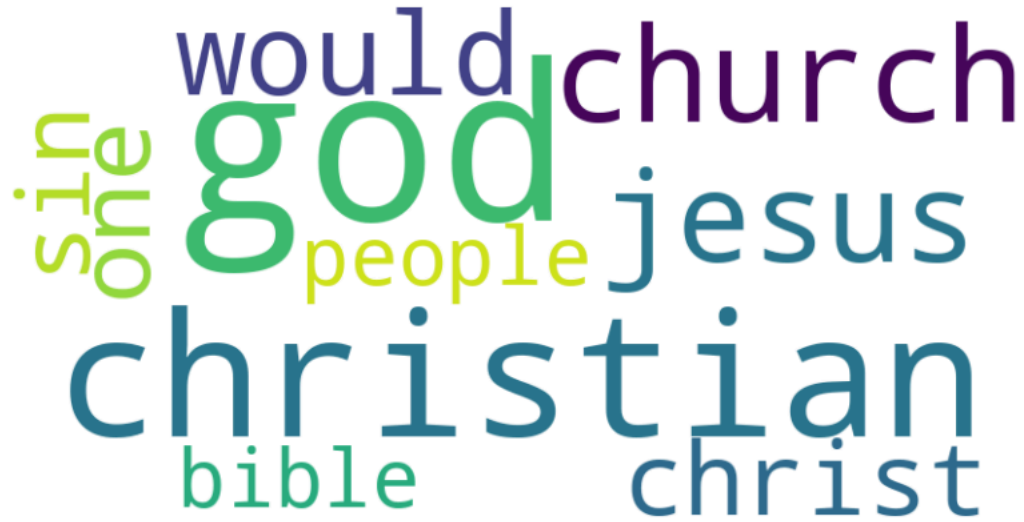
Word Cloud for COMP.WINDOWS.X

Word Cloud for REC.SPORT.HOCKEY

Word Cloud for REC.SPORT.BASEBALL



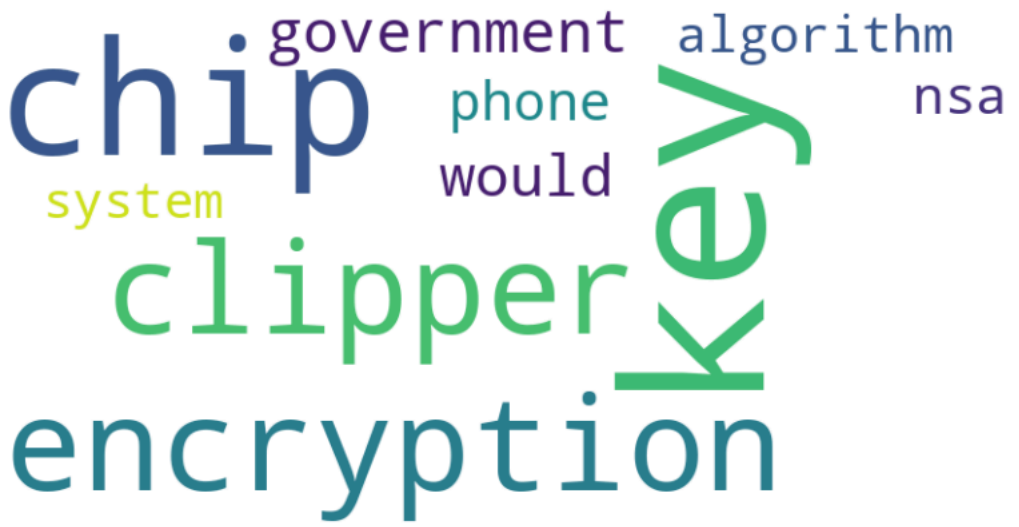
Word Cloud for SOC.RELIGION.CHRISTIAN



Word Cloud for TALK.POLITICS.MIDEAST

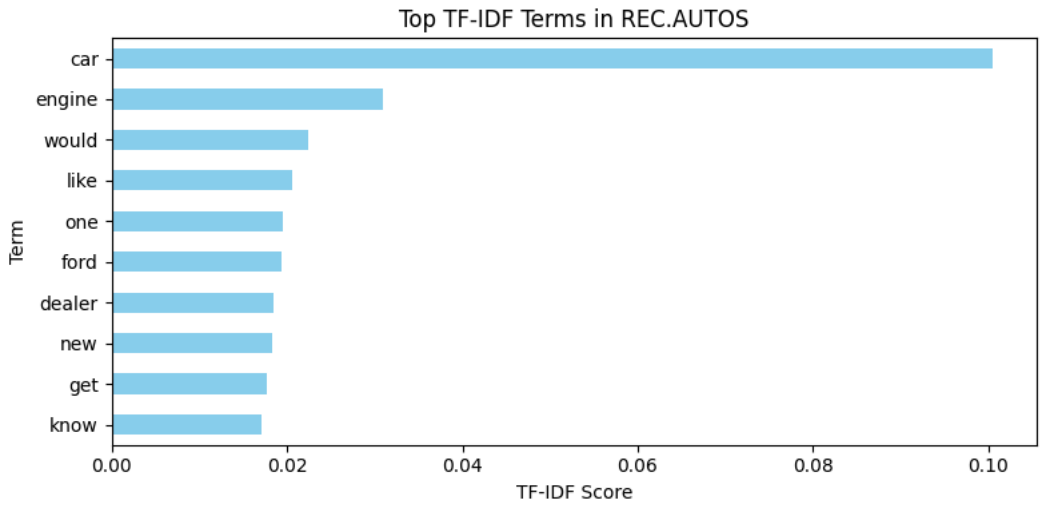
Word Cloud for TALK.POLITICS.MISC

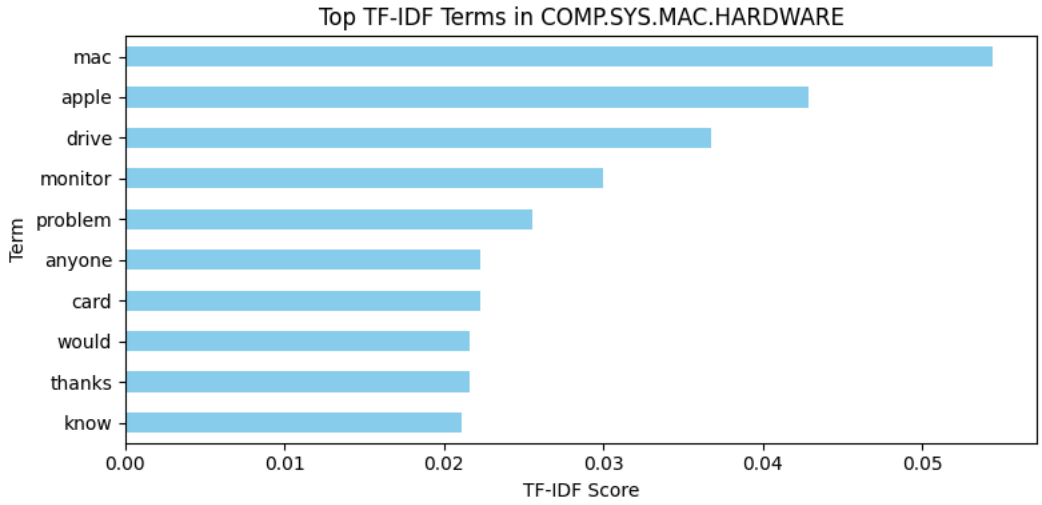
Word Cloud for SCI.CRYPT

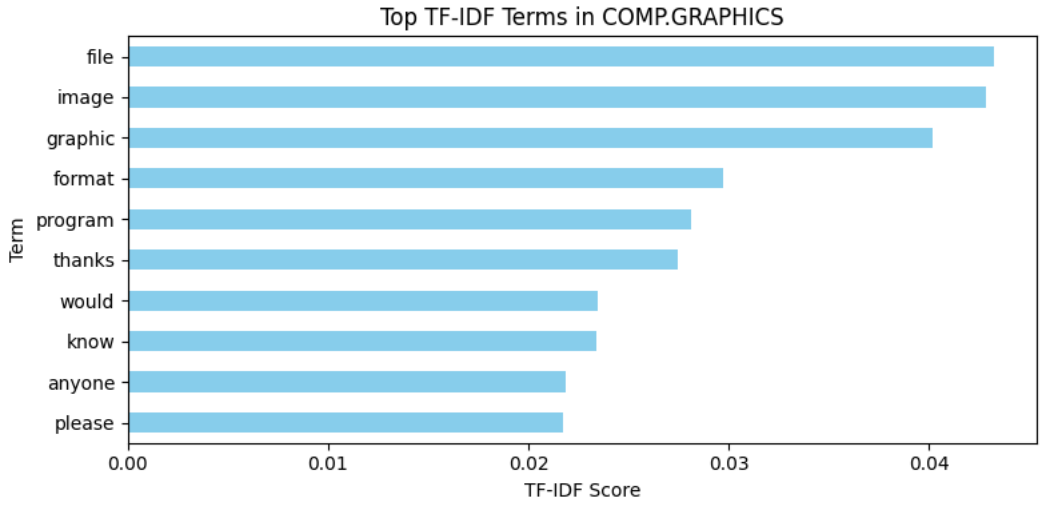


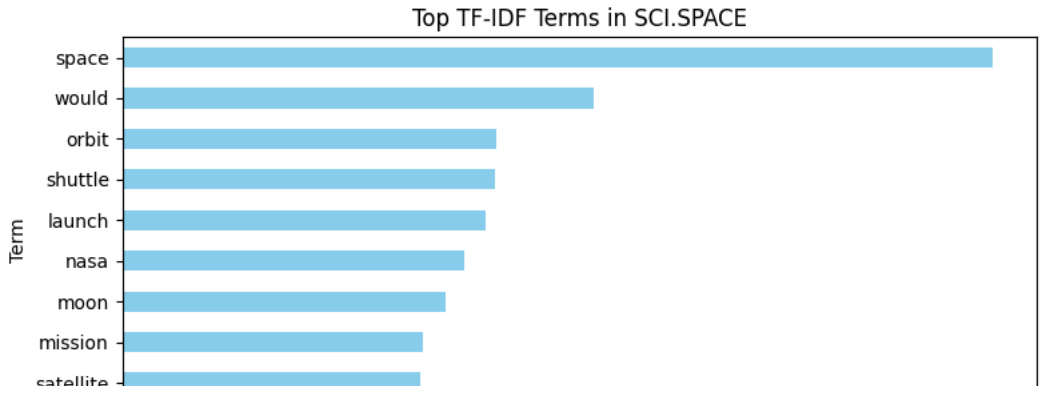
```
    for category, terms in category_terms.items():
        plt.figure(figsize=(8, 4))
        terms.sort_values().plot(kind='barh', color='skyblue')
        plt.title(f"Top TF-IDF Terms in {category.upper()}")
        plt.xlabel("TF-IDF Score")
        plt.ylabel("Term")
        plt.tight_layout()
        plt.show()
```

[+]

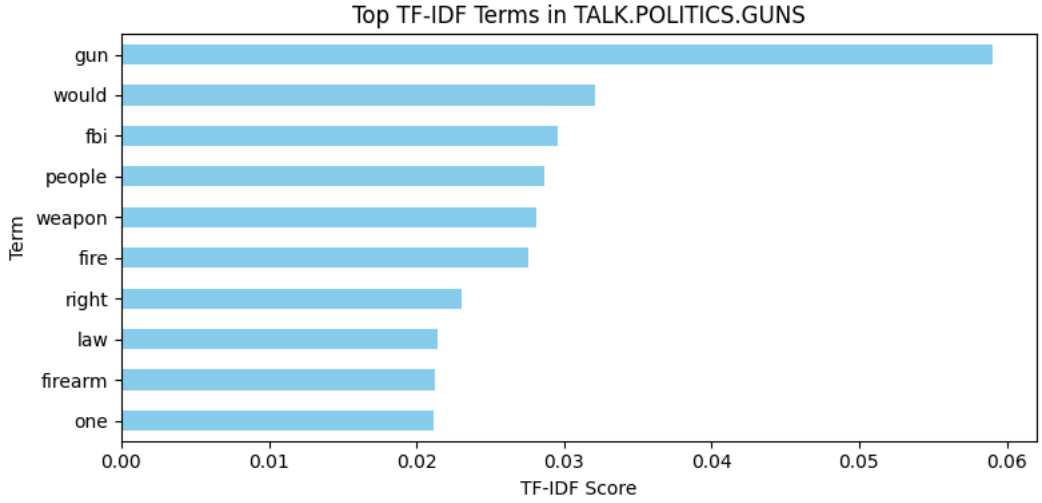

Top TF-IDF Terms in REC.AUTOS
car
engine
would
like
one
ford
dealer
new
get
know
Term
0.00
0.02
0.04
0.06
0.08
0.10
TF-IDF Score

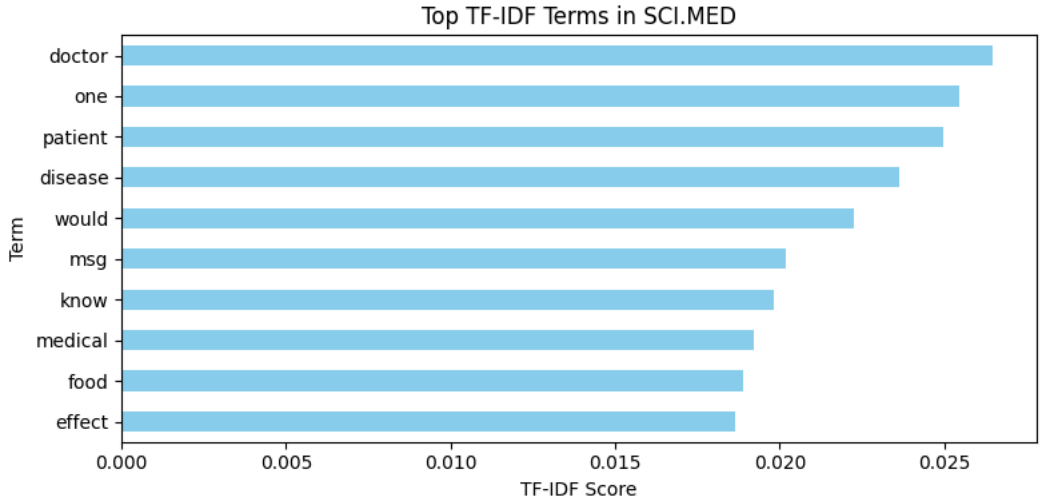

Top TF-IDF Terms in COMP.SYS.MAC.HARDWARE
mac
apple
drive
monitor
problem
anyone
card
would
thanks
know
Term
0.00
0.01
0.02
0.03
0.04
0.05
TF-IDF Score

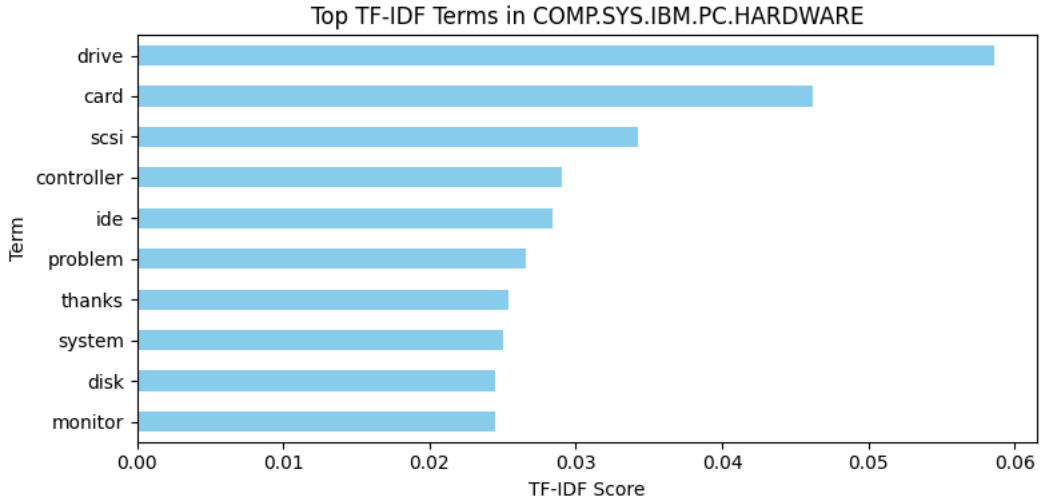

Top TF-IDF Terms in COMP.GRAPHICS
file
image
graphic
format
program
thanks
would
know
anyone
please
Term
0.00
0.01
0.02
0.03
0.04
TF-IDF Score

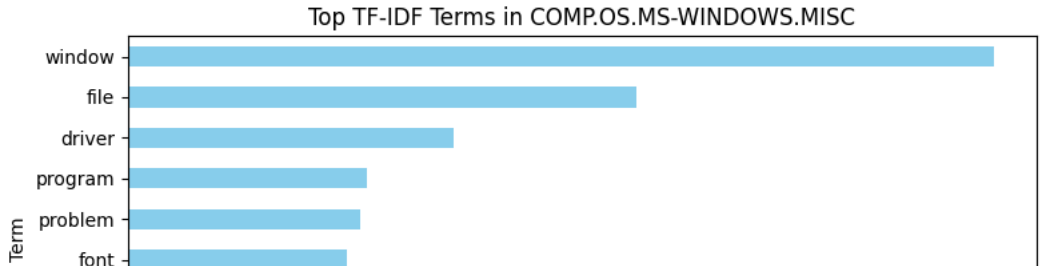

Top TF-IDF Terms in SCI.SPACE
space
would
orbit
shuttle
launch
nasa
moon
mission
satellite
Term

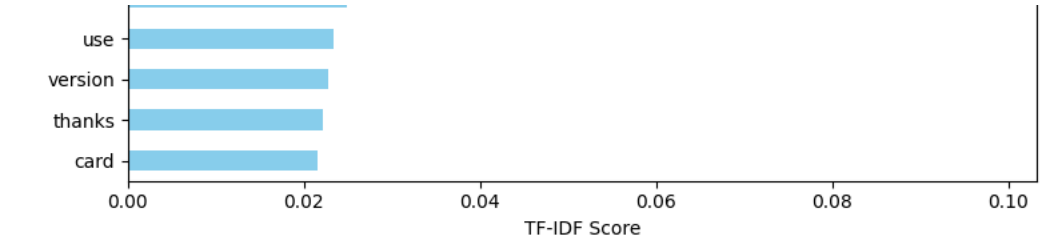

like
0.00
0.01
0.02
0.03
0.04
0.05
TF-IDF Score

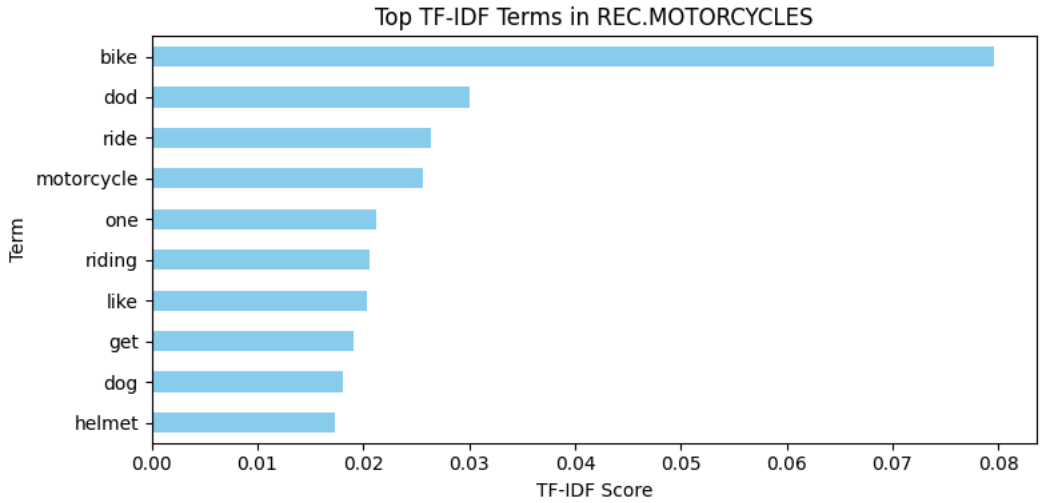

Top TF-IDF Terms in TALK.POLITICS.GUNS
gun
would
fbi
people
weapon
fire
right
law
firearm
one
Term
0.00
0.01
0.02
0.03
0.04
0.05
0.06
TF-IDF Score

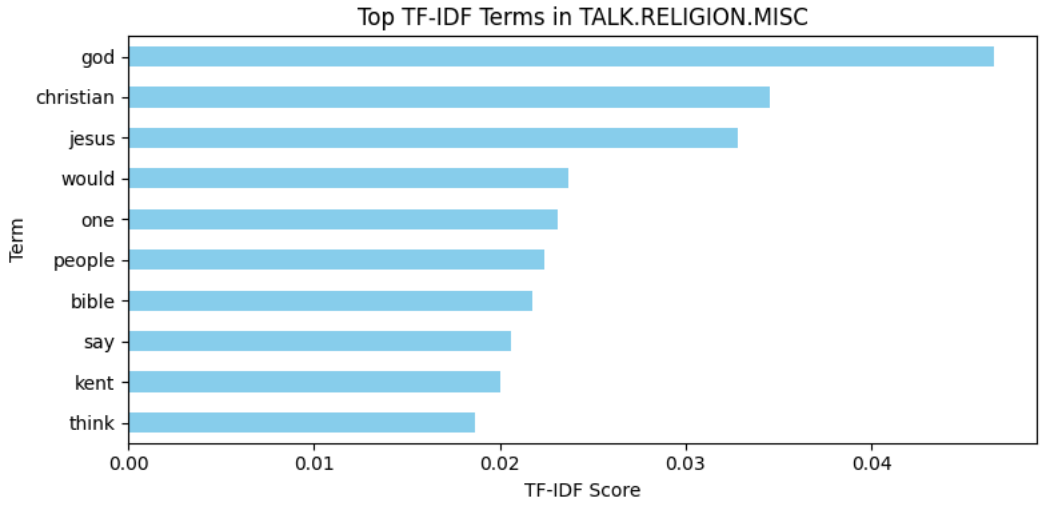

Top TF-IDF Terms in SCI.MED
doctor
one
patient
disease
would
msg
know
medical
food
effect
Term
0.000
0.005
0.010
0.015
0.020
0.025
TF-IDF Score

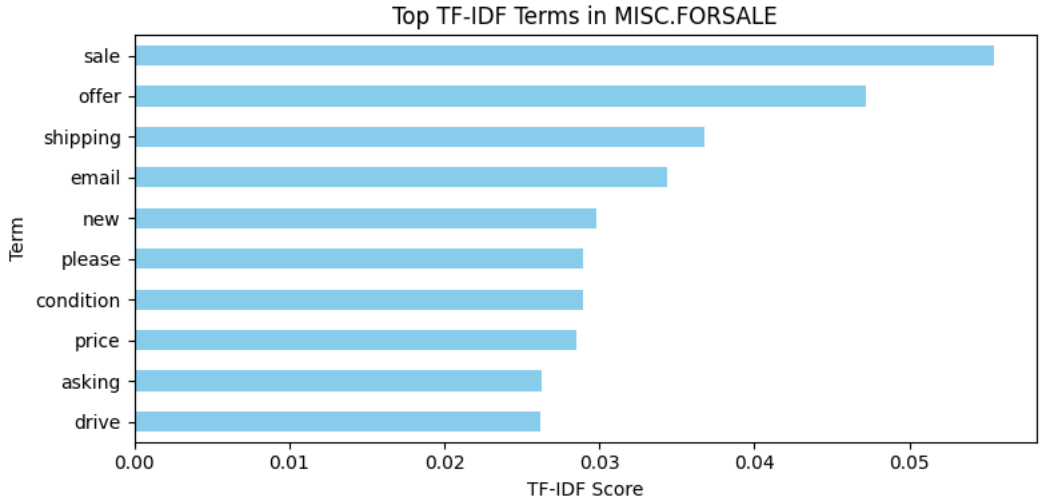

Top TF-IDF Terms in COMP.SYS.IBM.PC.HARDWARE
drive
card
scsi
controller
ide
problem
thanks
system
disk
monitor
Term
0.00
0.01
0.02
0.03
0.04
0.05
0.06
TF-IDF Score

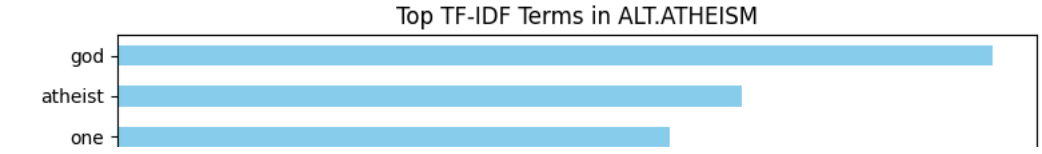

Top TF-IDF Terms in COMP.OS.MS-WINDOWS.MISC
window
file
driver
program
problem
font
Term

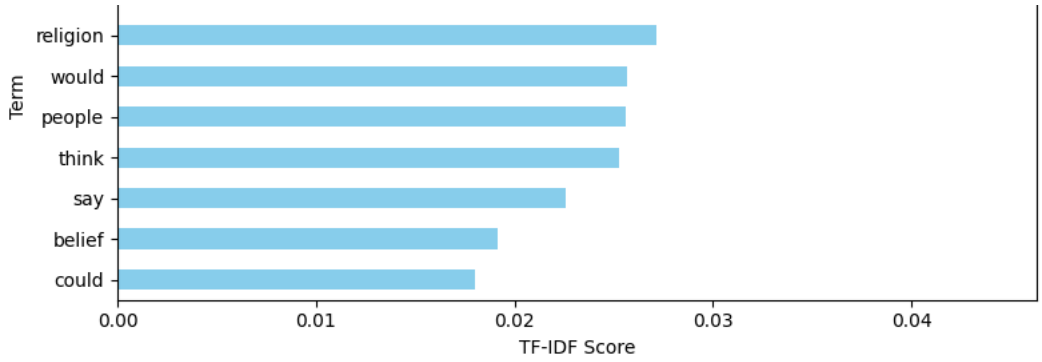

use
version
thanks
card
0.00
0.02
0.04
0.06
0.08
0.10
TF-IDF Score


Top TF-IDF Terms in REC.MOTORCYCLES
bike
dod
ride
motorcycle
one
riding
like
get
dog
helmet
Term
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
TF-IDF Score

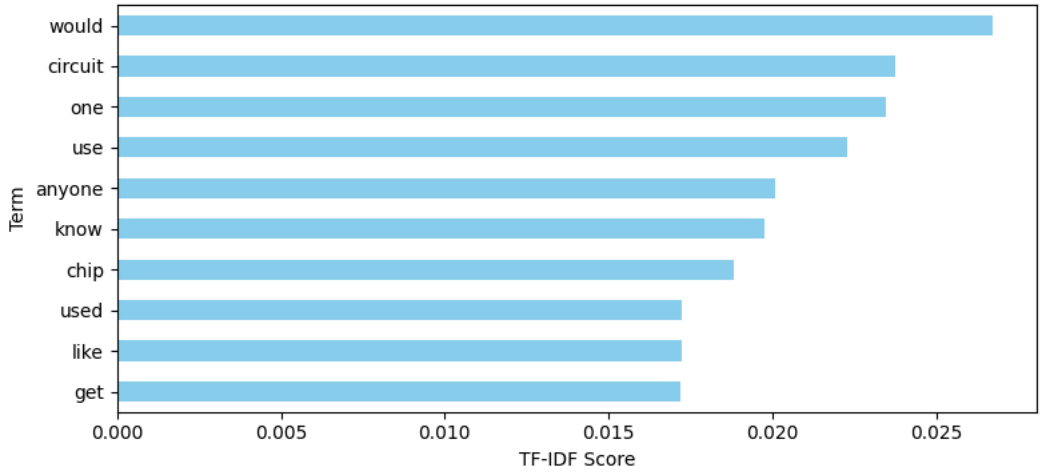

Top TF-IDF Terms in TALK.RELIGION.MISC
god
christian
jesus
would
one
people
bible
say
kent
think
Term
0.00
0.01
0.02
0.03
0.04
TF-IDF Score


Top TF-IDF Terms in MISC.FORSALE
sale
offer
shipping
email
new
please
condition
price
asking
drive
Term
0.00
0.01
0.02
0.03
0.04
0.05
TF-IDF Score

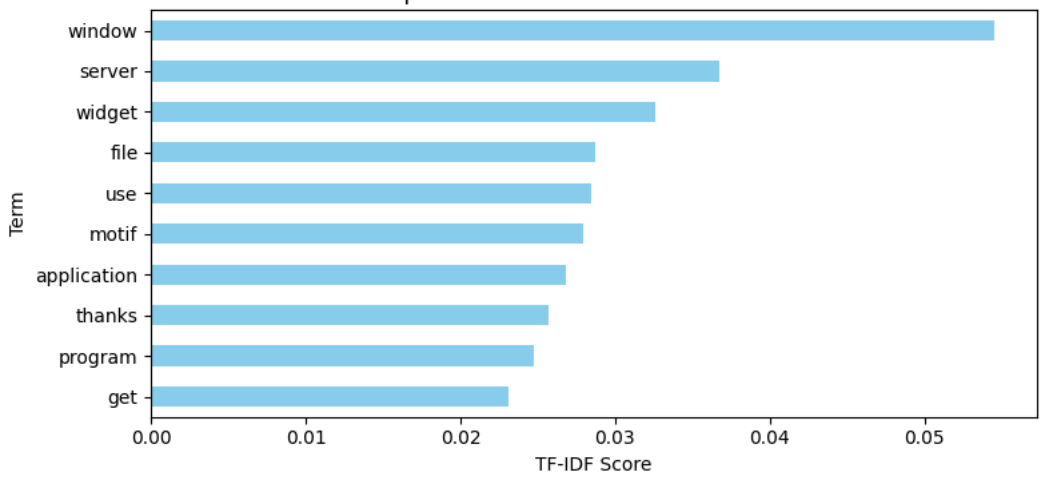

Top TF-IDF Terms in ALT.ATHEISM
god
atheist
one



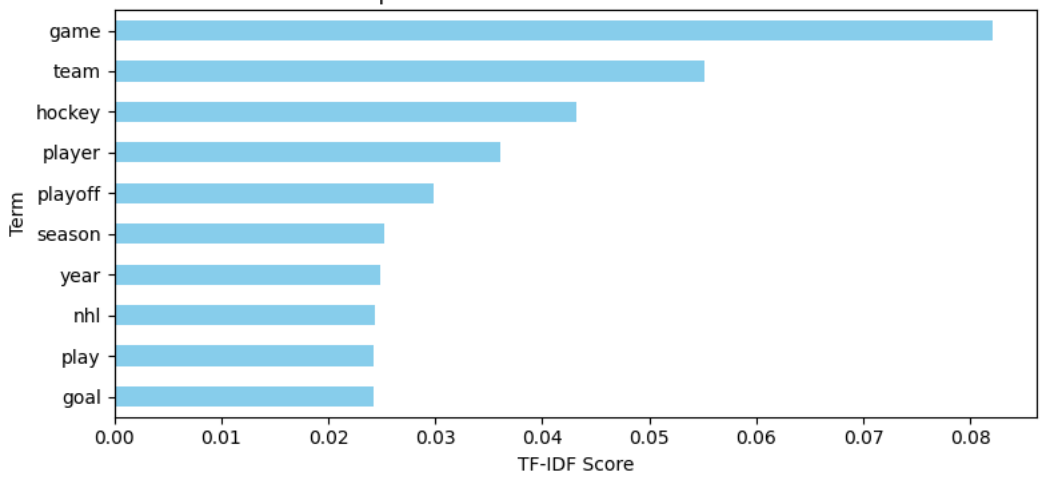
Top TF-IDF Terms in SCI.ELECTRONICS



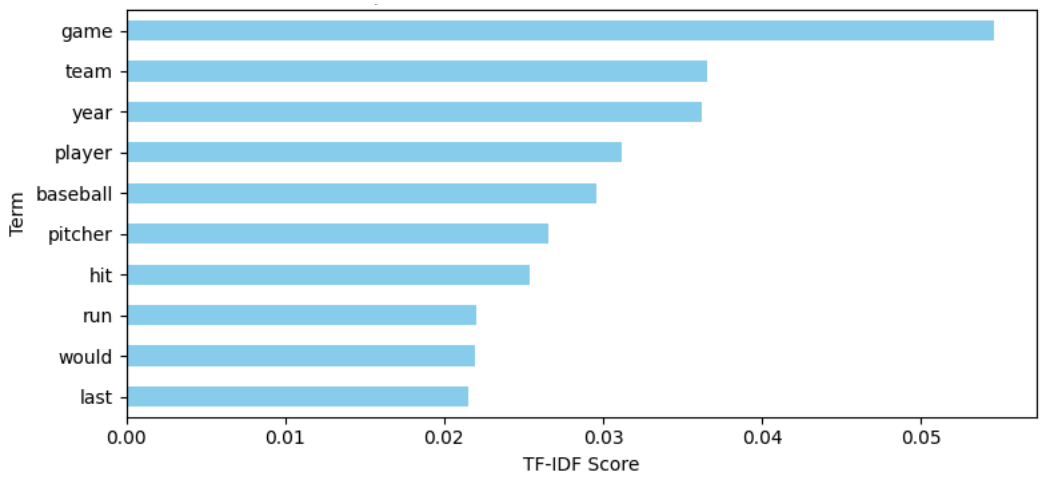
Top TF-IDF Terms in COMP.WINDOWS.X



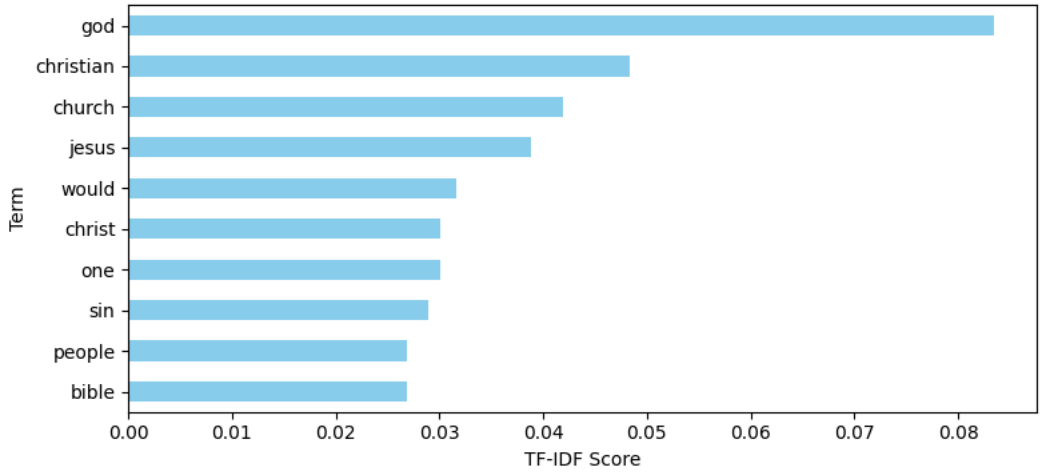
Top TF-IDF Terms in REC.SPORT.HOCKEY



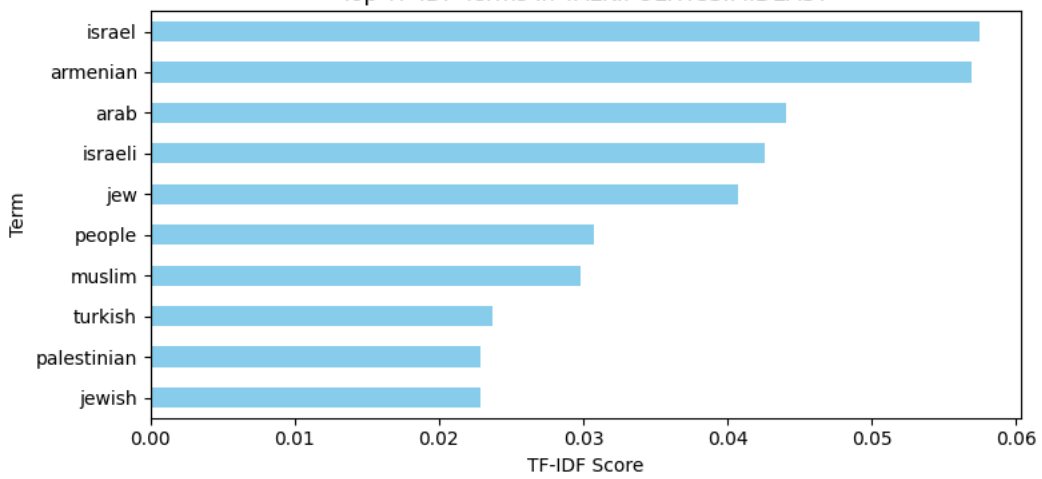
Top TF-IDF Terms in REC.SPORT.BASEBALL



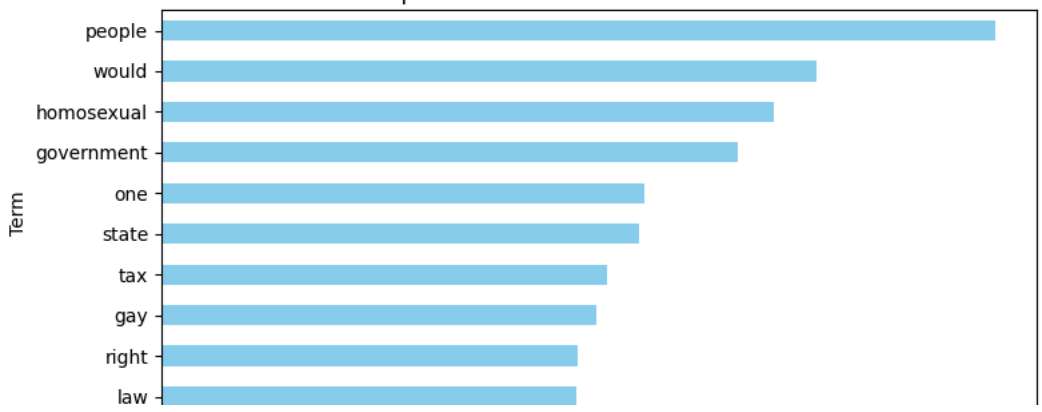
Top TF-IDF Terms in SOC.RELIGION.CHRISTIAN

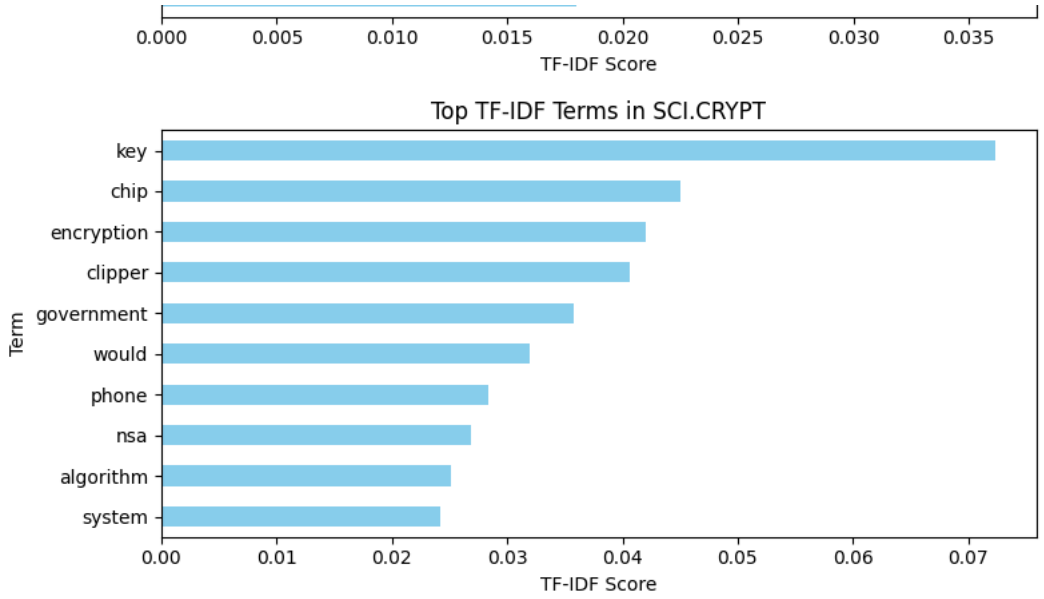


Top TF-IDF Terms in TALK.POLITICS.MIDEAST

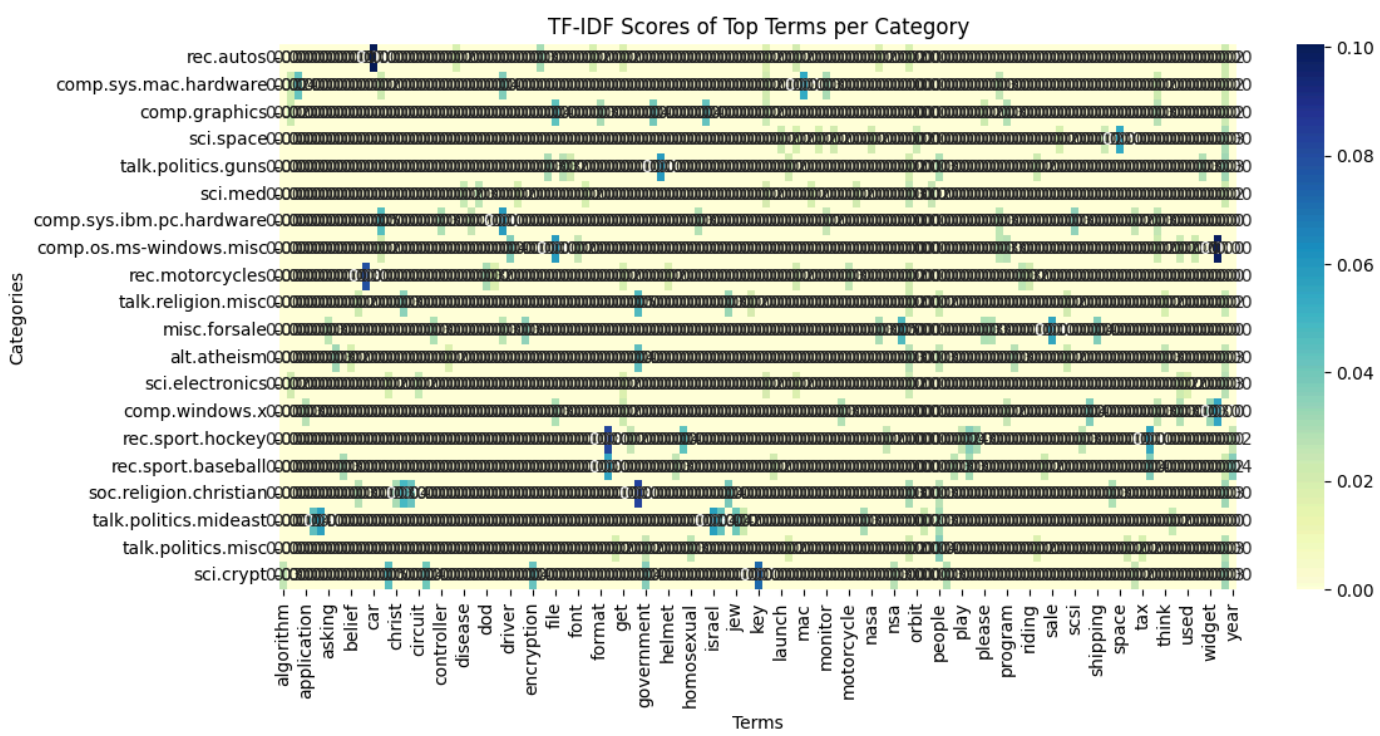


Top TF-IDF Terms in TALK.POLITICS.MISC




0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
TF-IDF Score
Top TF-IDF Terms in SCI.CRYPT
key
chip
encryption
clipper
government
would
phone
nsa
algorithm
system
Term
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
TF-IDF Score

```
# Build a DataFrame where rows = categories, columns = terms
heatmap_data = pd.DataFrame(category_terms).T.fillna(0)

plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, annot=True, fmt=".2f", cmap="YlGnBu")
plt.title("TF-IDF Scores of Top Terms per Category")
plt.xlabel("Terms")
plt.ylabel("Categories")
plt.tight_layout()
plt.show()
```




```
from sklearn.feature_extraction.text import TfidfVectorizer

# Assume df has 'content' and 'category'
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_tfidf = vectorizer.fit_transform(df['content'])

tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=vectorizer.get_feature_names_out())
tfidf_df['category'] = df['category']



# 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['category'].unique()
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
```

```
    game: 0.0360
    year: 0.0348
    baseball: 0.0337
    team: 0.0308
    games: 0.0289
    hit: 0.0241
    think: 0.0223
    players: 0.0211
    don: 0.0195
    braves: 0.0184

 SOC.RELIGION.CHRISTIAN:
    god: 0.0886
    jesus: 0.0417
    church: 0.0393
    christian: 0.0325
    christ: 0.0321
    christians: 0.0287
    people: 0.0285
    bible: 0.0283
    sin: 0.0255
    faith: 0.0247

 TALK.POLITICS.MIDEAST:
    israel: 0.0602
    jews: 0.0396
    israeli: 0.0359
    armenian: 0.0339
    people: 0.0314
    arab: 0.0303
    armenians: 0.0297
    jewish: 0.0265
    turkish: 0.0256
    arabs: 0.0225

 TALK.POLITICS.MISC:
    people: 0.0378
    government: 0.0257
    don: 0.0223
    just: 0.0195
    think: 0.0179
    gay: 0.0177
    clinton: 0.0173
    men: 0.0171
    like: 0.0169
    homosexual: 0.0161

 SCI.CRYPT:
    key: 0.0568
    clipper: 0.0458
    encryption: 0.0441
    chip: 0.0391
    government: 0.0373
    keys: 0.0303
    nsa: 0.0297
    use: 0.0231
    algorithm: 0.0228
    escrow: 0.0226
```

🏷️ Part-of-Speech Analysis

🎯 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.

```
# Step 1: Install spaCy and download the English model
!pip install spacy --quiet
!python -m spacy download en_core_web_sm

# Step 2: Import dependencies
import spacy
from collections import Counter

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

# Step 3: Define POS analysis function
def analyze_pos_patterns(text):
    """
    Analyze POS patterns in text using spaCy
    """
    if not text or pd.isna(text):
        return {}

    doc = nlp(str(text))

    # Count POS tags excluding punctuation and spaces
    pos_counts = Counter([token.pos_ for token in doc if not token.is_punct and not token.is_space])
    total_words = sum(pos_counts.values())

    if total_words == 0:
        return {}

    # Return proportions (you can switch to counts if needed)
    pos_proportions = {pos: count / total_words for pos, count in pos_counts.items()}
    return pos_proportions
```

```
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-any.whl (12
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 63.4 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.
```

```
# Step 4: Apply POS analysis to all articles
print("🗞️ Analyzing POS patterns...")

pos_results = []

for idx, row in df.iterrows():
    # Use 'processed_content' (cleaned text) for analysis
    pos_analysis = analyze_pos_patterns(row['processed_content'])

    # Add metadata
    pos_analysis['category'] = row['category']
    pos_analysis['article_id'] = row['article_id']

    # Collect the result
    pos_results.append(pos_analysis)

# Step 5: Convert list of POS dicts to DataFrame
pos_df = pd.DataFrame(pos_results).fillna(0)

# Step 6: View results
print("✅ POS analysis complete!")
print(f"📊 Found {len(pos_df.columns) - 2} different POS tags")
```

```
# ✅ Fix: Remove the extra period after head()
pos_df.head()
```

```
🔍 Analyzing POS patterns...
✅ POS analysis complete!
📊 Found 17 different POS tags
```

	VERB	PRON	AUX	NOUN	ADV	PROPN	ADJ	DET	INTJ	category	article_id	ADP	
0	0.209302	0.046512	0.023256	0.465116	0.069767	0.046512	0.093023	0.023256	0.023256	rec.autos	1	0.000000	0.000
1	0.226415	0.000000	0.000000	0.528302	0.018868	0.000000	0.150943	0.000000	0.037736	comp.sys.mac.hardware	2	0.018868	0.018
2	0.230216	0.035971	0.028777	0.352518	0.115108	0.043165	0.129496	0.000000	0.021583	comp.sys.mac.hardware	3	0.021583	0.007
3	0.285714	0.000000	0.000000	0.571429	0.000000	0.000000	0.000000	0.000000	0.000000	comp.graphics	4	0.142857	0.000
4	0.235294	0.000000	0.000000	0.441176	0.117647	0.117647	0.058824	0.000000	0.000000	sci.space	5	0.000000	0.000

Next steps: Generate code with pos_df | View recommended plots | New interactive sheet

```
!pip install -q spacy seaborn matplotlib
!python -m spacy download en_core_web_sm
```

```
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-any.whl (12
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 66.9 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.
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import spacy
from collections import Counter
from tqdm import tqdm

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

```
# Create 'processed_content' if not present
if 'processed_content' not in df.columns:
    print("🔄 Creating cleaned version of content...")
    df['processed_content'] = df['content'].str.lower().str.replace(r'[^a-z\s]', '', regex=True)
```

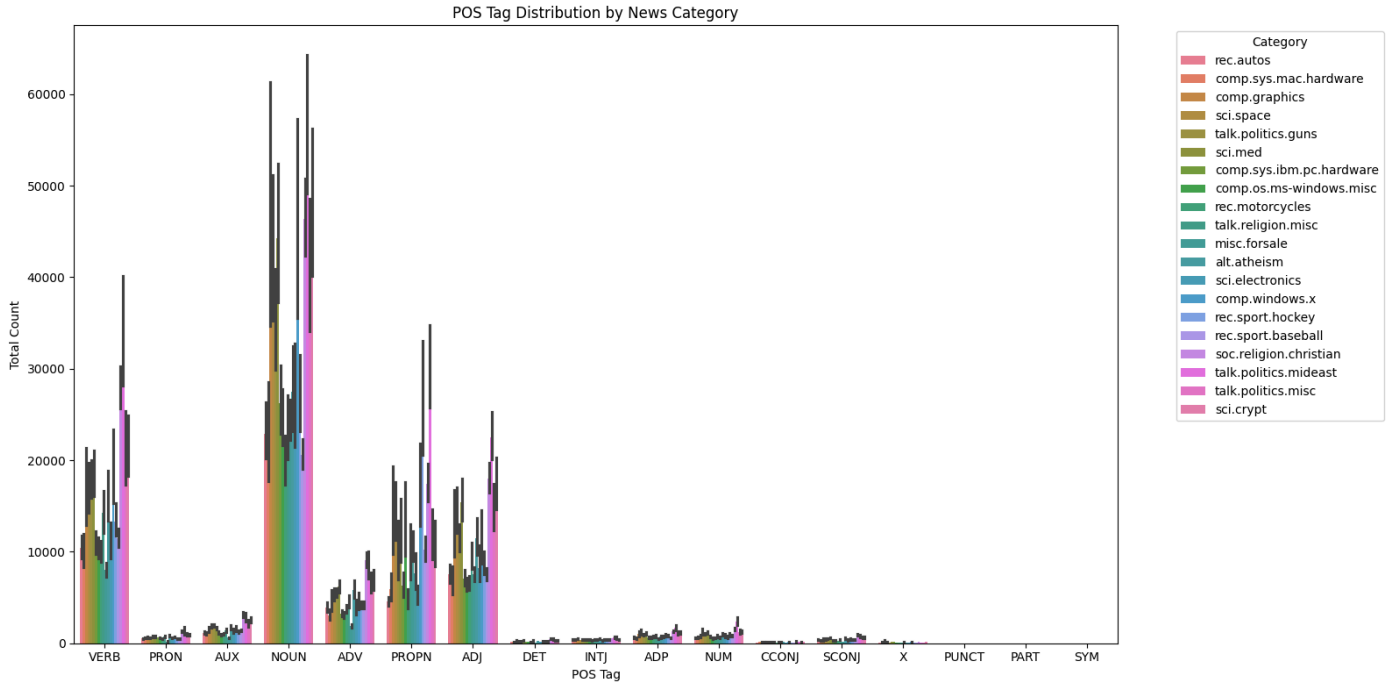
```
# Function to analyze POS tags for one article
def analyze_pos_patterns(text):
    doc = nlp(text)
    pos_counts = Counter([token.pos_ for token in doc if token.is_alpha])
    return dict(pos_counts)
```

```
print("🔍 Analyzing POS patterns...")

pos_results = []
for idx, row in tqdm(df.iterrows(), total=len(df)):
    pos_analysis = analyze_pos_patterns(row['processed_content'])  # or 'content'
    pos_analysis['category'] = row['category']
    pos_analysis['article_id'] = row['article_id']
    pos_results.append(pos_analysis)

# Create POS DataFrame
pos_df = pd.DataFrame(pos_results).fillna(0)

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

```
🔍 Analyzing POS patterns...
100%|██████████| 18846/18846 [06:02<00:00, 52.01it/s]
✅ POS analysis complete! Found 17 POS tags.
```

```
print("\n📊 POS PATTERNS BY CATEGORY\n" + "="*50)

# Melt for plotting
melted = pos_df.melt(id_vars=["article_id", "category"], var_name="pos_tag", value_name="count")

# Filter out tags with zero count
melted = melted[melted["count"] > 0]

# Plot
plt.figure(figsize=(16, 8))
sns.barplot(data=melted, x="pos_tag", y="count", hue="category", estimator=sum)
plt.title("POS Tag Distribution by News Category")
plt.xlabel("POS Tag")
plt.ylabel("Total Count")
plt.legend(title="Category", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

```
📊 POS PATTERNS BY CATEGORY
==================================================
```



🌳 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.

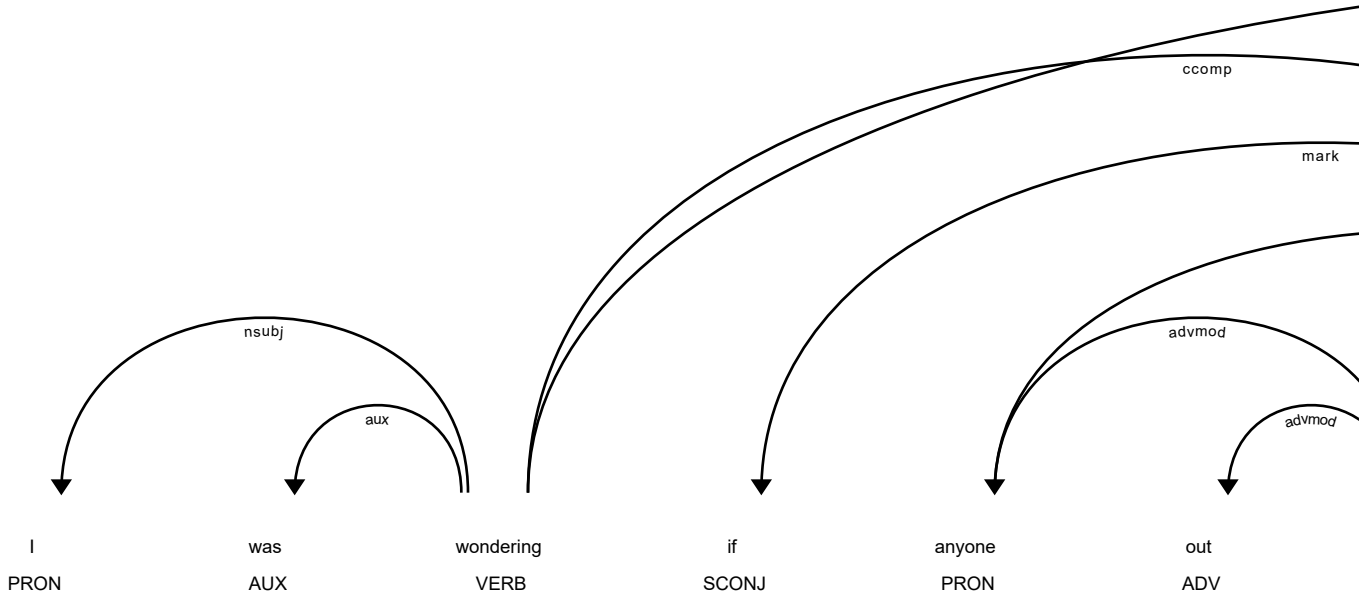
```
def extract_syntactic_features(text):
    """
    Extract syntactic features using spaCy dependency parsing

    💡 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 (rec.autos):
  Sentences: 7
  Tokens: 118
  Noun phrases: ['i', 'anyone', 'me']...
  Subjects: ['i', 'anyone', 'i']...
  Objects: ['me', 'car', 'day']...

📰 Article 2 (comp.sys.mac.hardware):
  Sentences: 6
  Tokens: 111
  Noun phrases: ['article', 'a fair number', 'brave souls']...
  Subjects: ['number', 'who', 'speed']...
  Objects: ['souls', 'oscillator', 'experiences']...

📰 Article 3 (comp.sys.mac.hardware):
  Sentences: 12
  Tokens: 393
  Noun phrases: ['article', 'well folks', 'my mac']...
  Subjects: ['folks', 'i', 'i']...
  Objects: ['ghost', 'life', 'way']...

📰 Article 4 (comp.graphics):
  Sentences: 2
  Tokens: 27
  Noun phrases: ['you', "weitek's address/phone number", 'i']...
  Subjects: ['you', 'i']...
  Objects: ['number', 'information', 'chip']...

📰 Article 5 (sci.space):
  Sentences: 2
  Tokens: 98
  Noun phrases: ['article', 'tombaker@world.std.com', 'tom a baker']...
  Subjects: ['understanding', 'errors', 'bugs']...
  Objects: ['article', 'tombaker@world.std.com', 'software']...
```

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

# Choose a sample sentence
sample_sentence = df.iloc[0]['content']  # First article's content
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: I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

🌳 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

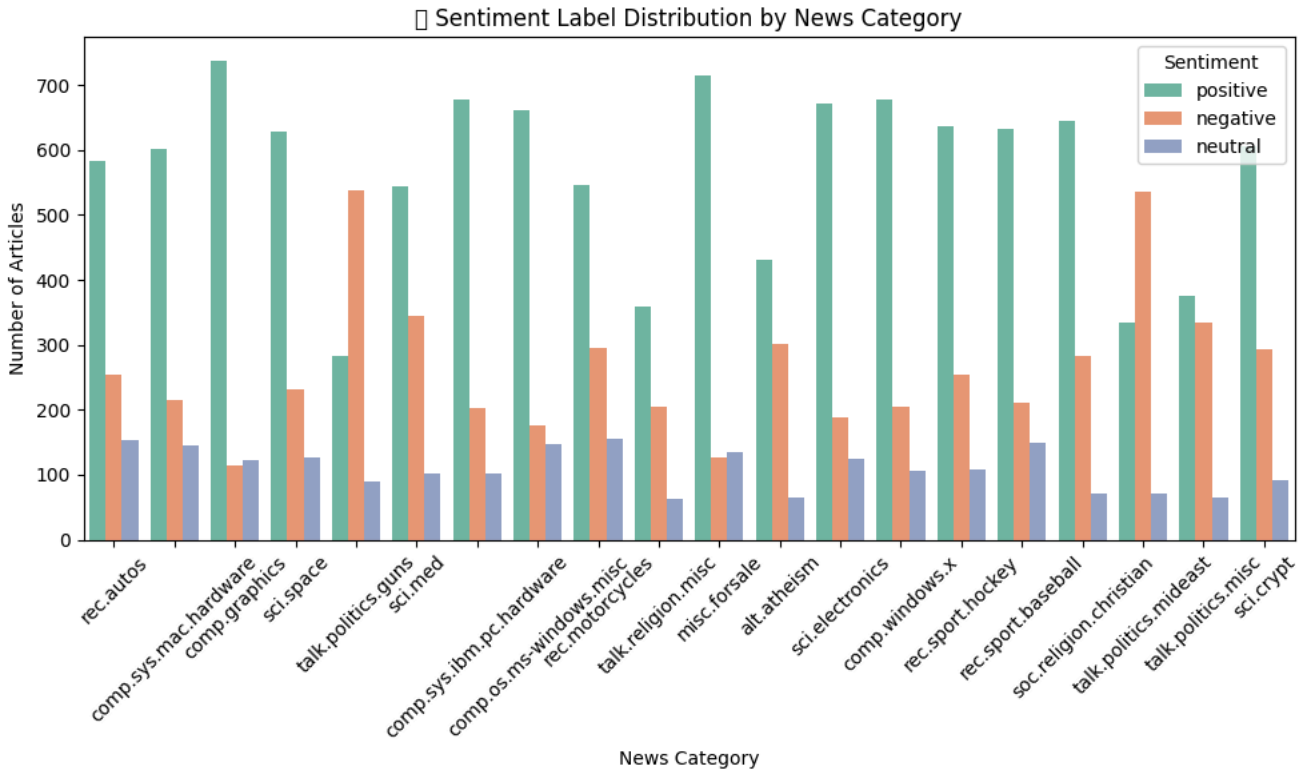
💡 **Hypothesis:** Different news categories might have different emotional profiles - sports might be more positive, politics more negative, etc.

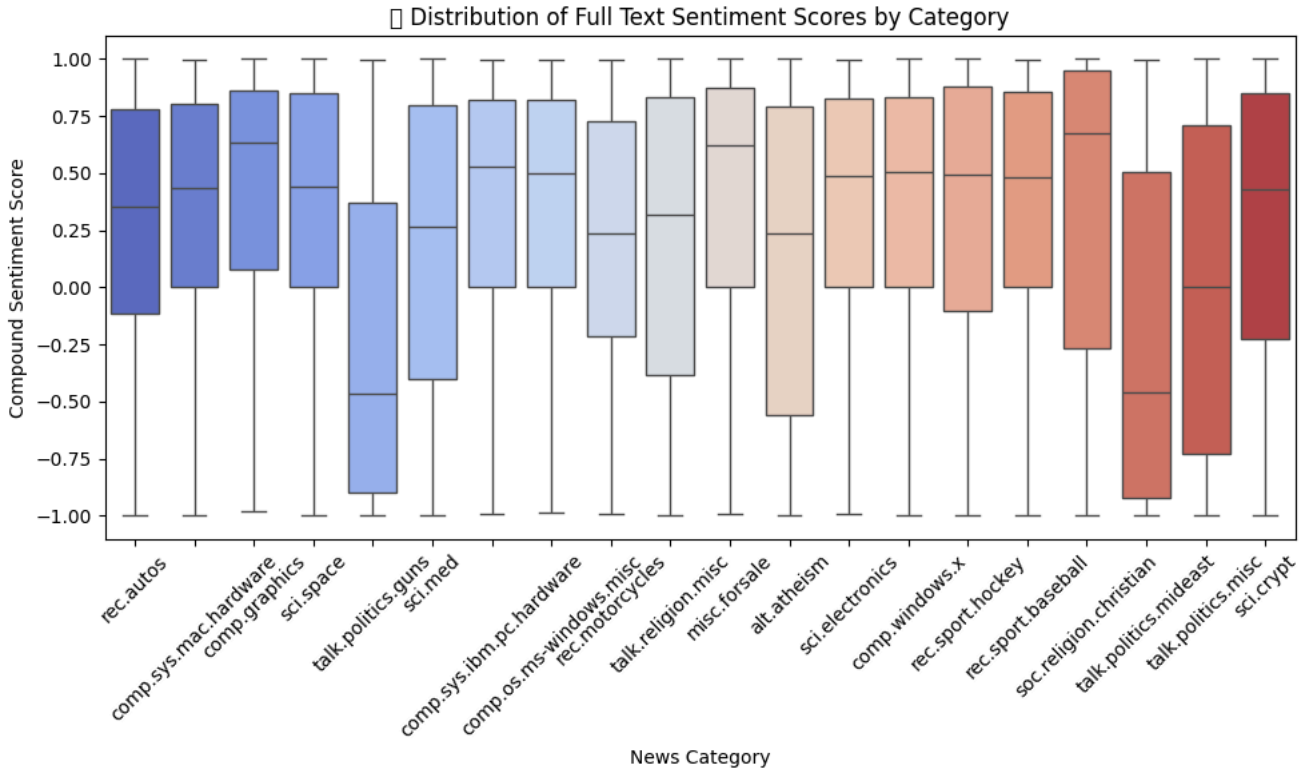
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk

# Download required NLTK data
nltk.download('vader_lexicon')
```

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

def analyze_sentiment(text):
    """
    Analyze sentiment using VADER sentiment analyzer.
    Returns sentiment scores and a sentiment label.
    """
    if not text or pd.isna(text):
        return {'compound': 0, 'pos': 0, 'neu': 1, 'neg': 0, 'sentiment_label': 'neutral'}

    scores = sia.polarity_scores(str(text))
    scores['sentiment_label'] = 'neutral'

    if scores['compound'] >= 0.05:
        scores['sentiment_label'] = 'positive'
    elif scores['compound'] <= -0.05:
        scores['sentiment_label'] = 'negative'

    return scores

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

sentiment_results = []
for idx, row in df.iterrows():
    title_sentiment = analyze_sentiment(row['title'])
    content_sentiment = analyze_sentiment(row['content'])
    full_sentiment = analyze_sentiment(row['full_text'])

    result = {
        'article_id': row['article_id'],
        'category': row['category'],
        'title_sentiment': title_sentiment['compound'],
        'title_label': title_sentiment['sentiment_label'],
        'content_sentiment': content_sentiment['compound'],
        'content_label': content_sentiment['sentiment_label'],
        '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())

# 🔍 Aggregate sentiment by category
agg_df = sentiment_df.groupby('category')[['full_sentiment', 'pos_score', 'neu_score', 'neg_score']].mean().reset_index()

# 📊 Visualization 1: Count of sentiment labels by category
plt.figure(figsize=(10, 6))
sns.countplot(data=sentiment_df, x='category', hue='full_label', palette='Set2')
plt.title("📊 Sentiment Label Distribution by News Category")
plt.xlabel("News Category")
plt.ylabel("Number of Articles")
plt.legend(title="Sentiment")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 📊 Visualization 2: Boxplot of compound sentiment score by category
plt.figure(figsize=(10, 6))
sns.boxplot(data=sentiment_df, x='category', y='full_sentiment', palette='coolwarm')
plt.title("📦 Distribution of Full Text Sentiment Scores by Category")
plt.xlabel("News Category")
plt.ylabel("Compound Sentiment Score")
plt.xticks(rotation=45)
plt.tight_layout()
```

```
plt.show()

# 📊 Visualization 3: Average sentiment scores by category
plt.figure(figsize=(10, 6))
agg_df.set_index('category')[['pos_score', 'neu_score', 'neg_score']].plot(kind='bar', stacked=True, colormap='viridis', figsize=(12, 6))
plt.title("📊 Average Sentiment Component Scores by Category")
plt.ylabel("Score")
plt.xlabel("News Category")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 💡 Bonus: Sentiment label proportions
label_proportions = sentiment_df['full_label'].value_counts(normalize=True).reset_index()
label_proportions.columns = ['sentiment_label', 'proportion']

plt.figure(figsize=(6, 6))
sns.barplot(data=label_proportions, x='sentiment_label', y='proportion', palette='pastel')
plt.title("📈 Overall Sentiment Label Proportions")
plt.ylabel("Proportion of Articles")
plt.tight_layout()
plt.show()
```

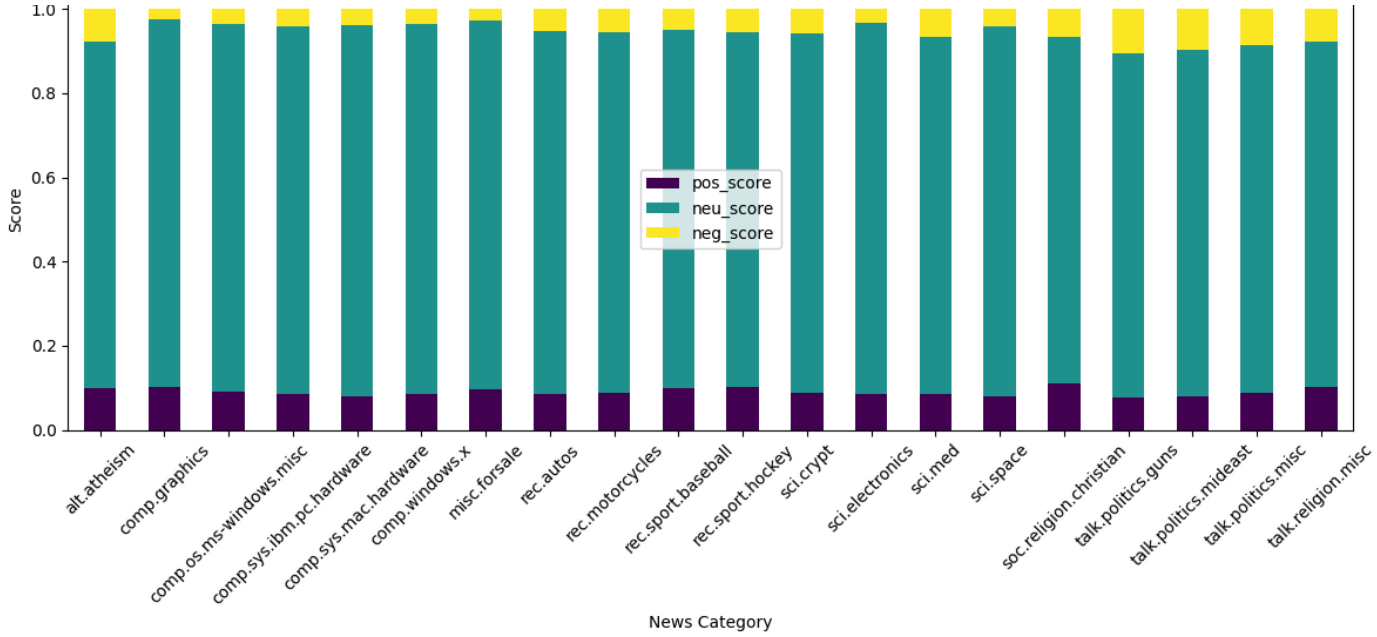
```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
😊 Analyzing sentiment...
✅ Sentiment analysis complete!
📊 Analyzed 18846 articles

📝 Sample sentiment results:
                category  full_sentiment full_label
0             rec.autos          0.6249   positive
1  comp.sys.mac.hardware          0.9468   positive
2  comp.sys.mac.hardware          0.9761   positive
3         comp.graphics          0.4215   positive
4             sci.space         -0.8020   negative
```

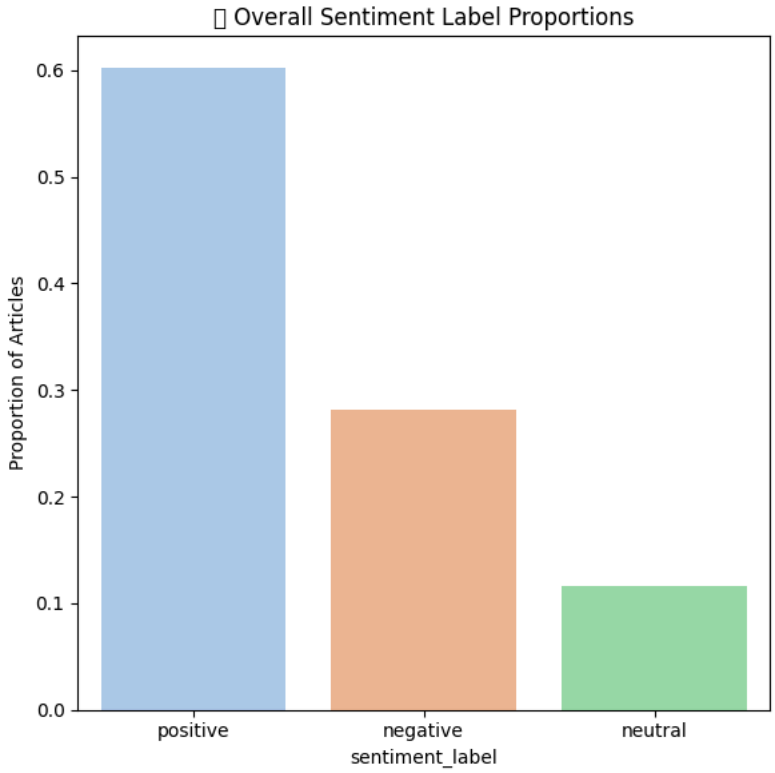





```
<Figure size 1000x600 with 0 Axes>
```

📊 Average Sentiment Component Scores by Category


Score
1.0
0.8
0.6
0.4
0.2
0.0
pos_score
neu_score
neg_score
alt.atheism
comp.graphics
comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware
comp.sys.mac.hardware
comp.windows.x
misc.forsale
rec.autos
rec.motorcycles
rec.sport.baseball
rec.sport.hockey
sci.crypt
sci.electronics
sci.med
sci.space
soc.religion.christian
talk.politics.guns
talk.politics.mideast
talk.politics.misc
talk.religion.misc
News Category


Overall Sentiment Label Proportions
Proportion of Articles
0.6
0.5
0.4
0.3
0.2
0.1
0.0
positive
negative
neutral
sentiment_label

```
# 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?
```

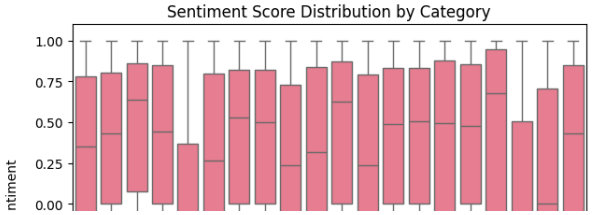
📊 SENTIMENT ANALYSIS BY CATEGORY
==

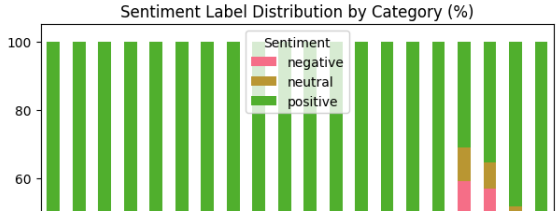
📈 Sentiment statistics by category:

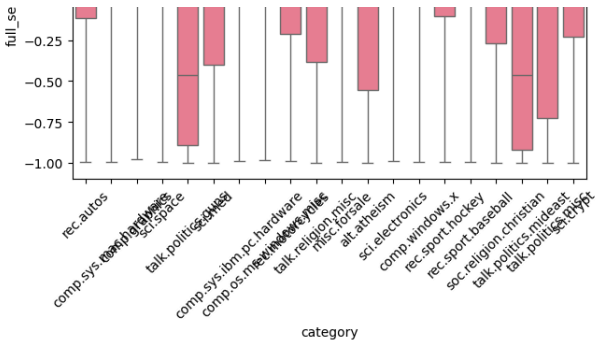
category	full_sentiment mean	full_sentiment std	full_sentiment min	full_sentiment max	pos_score mean	neu_score mean	neg_score mean
alt.atheism	0.1197	0.6915	-0.9997	0.9999	0.0980	0.8234	0.0785
comp.graphics	0.4823	0.4740	-0.9778	1.0000	0.1014	0.8742	0.0244
comp.os.ms-windows.misc	0.3564	0.5444	-0.9838	0.9995	0.0904	0.8738	0.0358
comp.sys.ibm.pc.hardware	0.3505	0.5503	-0.9933	0.9979	0.0865	0.8727	0.0409
comp.sys.mac.hardware	0.2930	0.5641	-0.9953	0.9996	0.0810	0.8807	0.0383
comp.windows.x	0.3479	0.5600	-0.9990	0.9999	0.0857	0.8782	0.0361
misc.forsale	0.4655	0.4865	-0.9936	0.9998	0.0978	0.8753	0.0268
rec.autos	0.2385	0.5912	-0.9957	0.9998	0.0862	0.8624	0.0515
rec.motorcycles	0.1876	0.5951	-0.9925	0.9981	0.0890	0.8551	0.0559
rec.sport.baseball	0.3338	0.5763	-0.9975	0.9995	0.1002	0.8505	0.0492
rec.sport.hockey	0.3162	0.6221	-0.9949	0.9999	0.1010	0.8445	0.0546
sci.crypt	0.2587	0.6416	-0.9995	0.9999	0.0876	0.8550	0.0573
sci.electronics	0.3570	0.5211	-0.9896	0.9994	0.0856	0.8819	0.0325
sci.med	0.1728	0.6536	-0.9998	0.9998	0.0857	0.8474	0.0669
sci.space	0.3130	0.5790	-0.9959	0.9999	0.0796	0.8787	0.0417
soc.religion.christian	0.3244	0.7111	-0.9998	1.0000	0.1102	0.8225	0.0673
talk.politics.guns	-0.2539	0.6760	-0.9999	0.9995	0.0781	0.8175	0.1045
talk.politics.mideast	-0.2126	0.7238	-1.0000	0.9993	0.0793	0.8241	0.0967
talk.politics.misc	0.0172	0.7063	-1.0000	1.0000	0.0890	0.8265	0.0845
talk.religion.misc	0.1790	0.6839	-0.9998	1.0000	0.1008	0.8208	0.0784

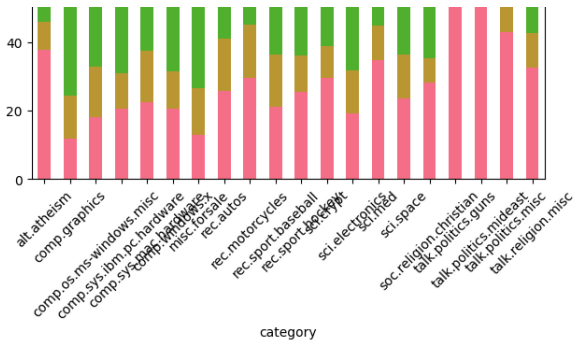
📊 Sentiment distribution (%) by category:

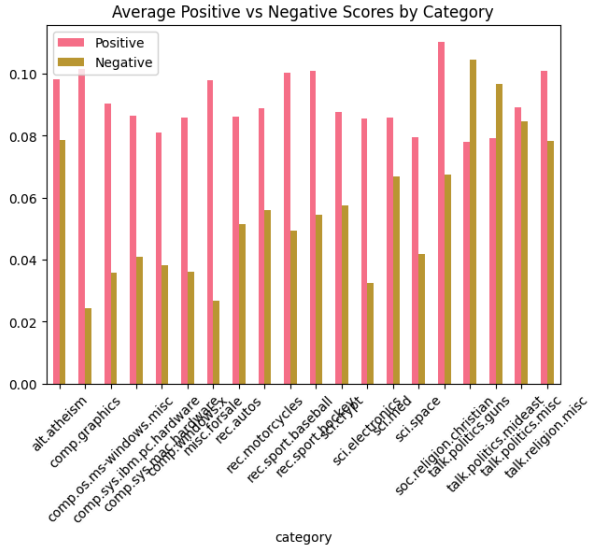
category	negative	neutral	positive
alt.atheism	37.80	8.26	53.94
comp.graphics	11.72	12.54	75.75
comp.os.ms-windows.misc	17.97	14.92	67.11
comp.sys.ibm.pc.hardware	20.67	10.39	68.94
comp.sys.mac.hardware	22.33	15.16	62.51
comp.windows.x	20.65	10.83	68.52
misc.forsale	12.92	13.74	73.33
rec.autos	25.66	15.45	58.89
rec.motorcycles	29.62	15.56	54.82
rec.sport.baseball	21.23	15.09	63.68
rec.sport.hockey	25.43	10.81	63.76
sci.crypt	29.67	9.18	61.15
sci.electronics	19.21	12.60	68.19
sci.med	34.75	10.20	55.05
sci.space	23.51	12.87	63.63
soc.religion.christian	28.28	7.12	64.59
talk.politics.guns	59.12	9.89	30.99
talk.politics.mideast	56.91	7.55	35.53
talk.politics.misc	43.10	8.52	48.39
talk.religion.misc	32.64	10.03	57.32

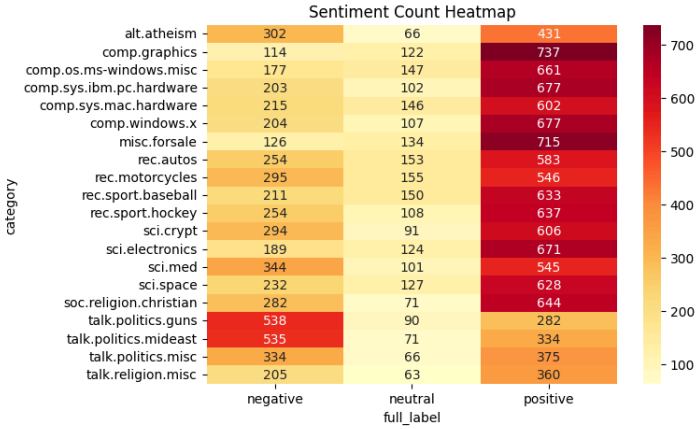





full_se
−0.25
−0.50
−0.75
−1.00
rec.autos
comp.sys.ibm.pc.hardware
sci.space
talk.politics.guns
comp.sys.ibm.pc.hardware
talk.religion.misc
misc.forsale
alt.atheism
sci.electronics
comp.windows.x
rec.sport.hockey
rec.sport.baseball
soc.religion.christian
talk.politics.mideast
talk.politics.misc
category


40
20
0
alt.atheism
comp.graphics
comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware
misc.forsale
rec.autos
rec.motorcycles
rec.sport.baseball
sci.electronics
sci.space
soc.religion.christian
talk.politics.guns
talk.politics.mideast
talk.politics.misc
talk.religion.misc
category


Average Positive vs Negative Scores by Category
Positive
Negative
0.10
0.08
0.06
0.04
0.02
0.00
alt.atheism
comp.graphics
comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware
misc.forsale
rec.autos
rec.motorcycles
rec.sport.baseball
sci.electronics
sci.space
soc.religion.christian
talk.politics.guns
talk.politics.mideast
talk.politics.misc
talk.religion.misc
category


Sentiment Count Heatmap
alt.atheism 302 66 431
comp.graphics 114 122 737
comp.os.ms-windows.misc 177 147 661
comp.sys.ibm.pc.hardware 203 102 677
comp.sys.mac.hardware 215 146 602
comp.windows.x 204 107 677
misc.forsale 126 134 715
rec.autos 254 153 583
rec.motorcycles 295 155 546
rec.sport.baseball 211 150 633
rec.sport.hockey 254 108 637
sci.crypt 294 91 606
sci.electronics 189 124 671
sci.med 344 101 545
sci.space 232 127 628
soc.religion.christian 282 71 644
talk.politics.guns 538 90 282
talk.politics.mideast 535 71 334
talk.politics.misc 334 66 375
talk.religion.misc 205 63 360
category
negative
neutral
positive
full_label
700
600
500
400
300
200
100

🏷️ 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:

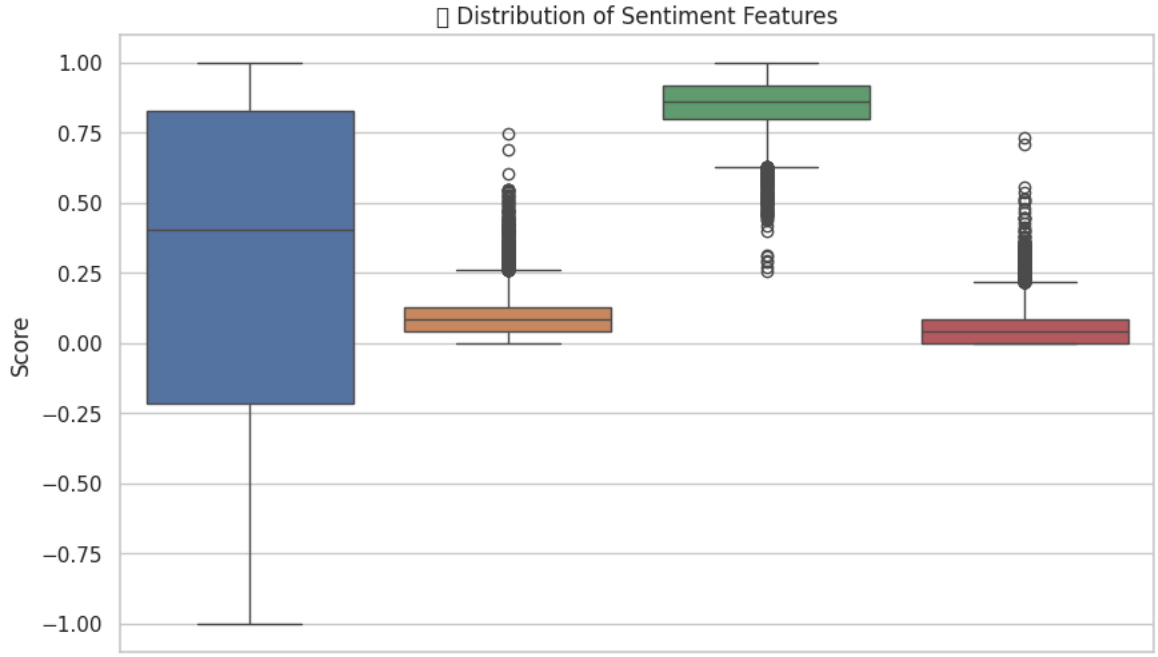
1. **Feature Engineering:** Combine TF-IDF with other features
2. **Model Training:** Train multiple algorithms
3. **Model Evaluation:** Compare performance metrics
4. **Model Selection:** Choose the best performing model

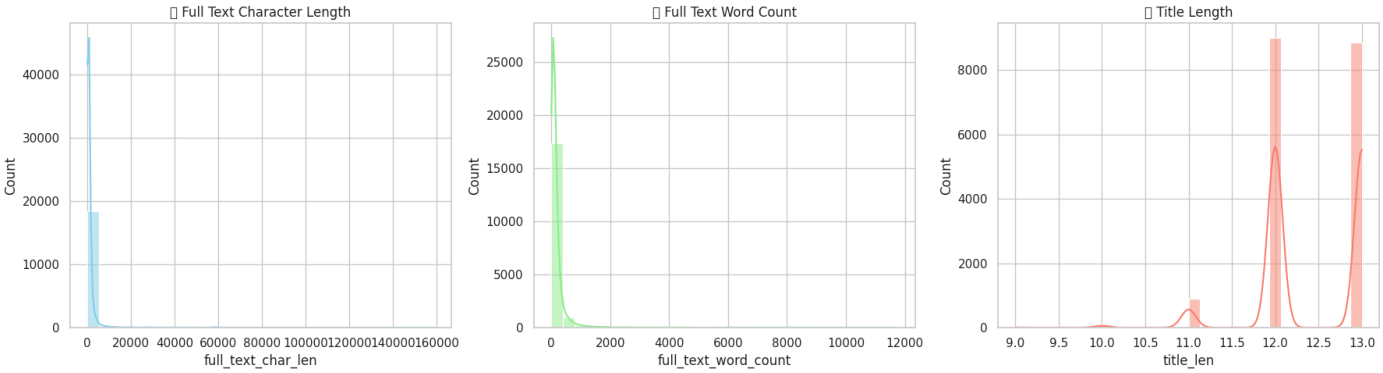
💡 **Business Impact:** Accurate classification enables automatic content routing, personalized recommendations, and efficient content management.

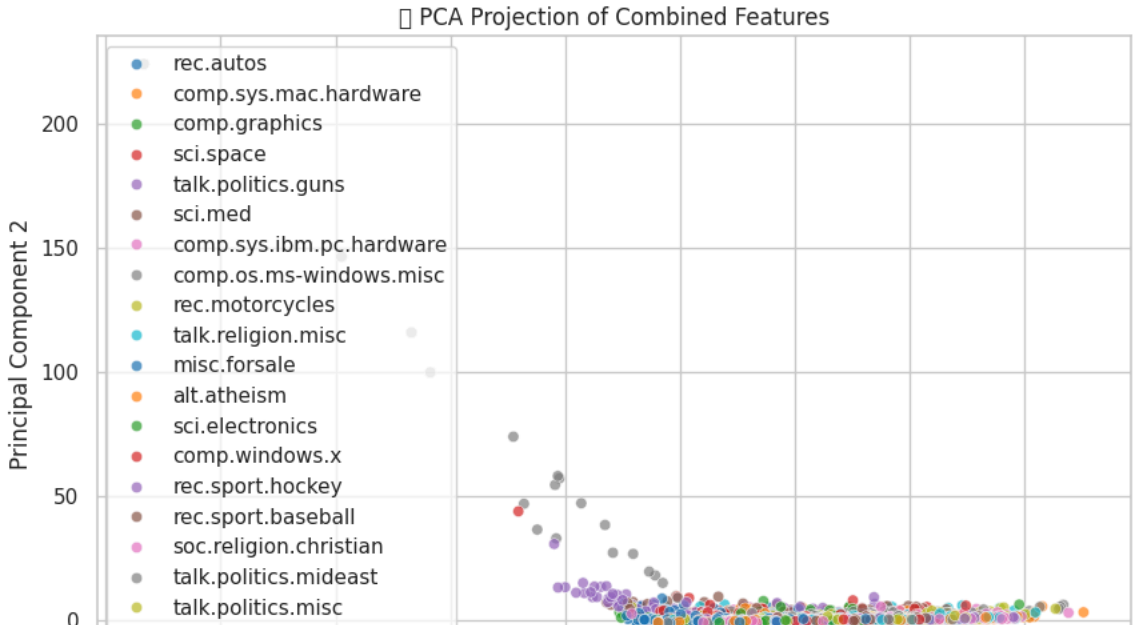
```
# ✅ REQUIRED IMPORTS
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# ✅ SET PLOTTING STYLE
sns.set(style='whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)

# 📥 LOAD YOUR DATA (replace with your file path or upload step if not done already)
# df = pd.read_csv("your_news_dataset.csv")

# ✅ TF-IDF FEATURE EXTRACTION
tfidf = TfidfVectorizer(max_features=1000)
tfidf_matrix = tfidf.fit_transform(df['full_text'].astype(str))

# ✅ SENTIMENT FEATURES FROM EARLIER CODE (make sure you have `sentiment_df`)
# Ensure sentiment_df is aligned with df
assert len(sentiment_df) == len(df), "Mismatch between df and sentiment_df lengths"

# ✅ TEXT LENGTH FEATURES
df['full_text_char_len'] = df['full_text'].astype(str).str.len()
df['full_text_word_count'] = df['full_text'].astype(str).str.split().str.len()
df['title_len'] = df['title'].astype(str).str.len()

length_features = df[['full_text_char_len', 'full_text_word_count', 'title_len']].values

# ✅ COMBINE ALL FEATURES INTO A SINGLE MATRIX
X_tfidf = tfidf_matrix.toarray()
sentiment_features = sentiment_df[['full_sentiment', 'pos_score', 'neu_score', 'neg_score']].values

X_combined = np.hstack([
    X_tfidf,
    sentiment_features,
    length_features
])

# ✅ TARGET LABELS
y = df['category'].values

# ✅ TRAIN-TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(
    X_combined, y, test_size=0.2, random_state=42, stratify=y
)

# ✅ STATUS OUTPUT
print(f"✅ Feature matrix prepared!")
print(f"📊 Shape of X_combined: {X_combined.shape}")
print(f"🎯 Number of target classes: {len(np.unique(y))}")
print(f"📋 Class labels: {np.unique(y)}")
```

```
print(f"🔀 Train/Test Split: {X_train.shape[0]} train, {X_test.shape[0]} test")

# ---------------------------------------
# 📊 VISUALIZATIONS FOR FEATURES
# ---------------------------------------

# 📈 1. Sentiment Score Distribution
plt.figure(figsize=(10, 6))
sns.boxplot(data=sentiment_df[['full_sentiment', 'pos_score', 'neu_score', 'neg_score']])
plt.title("📊 Distribution of Sentiment Features")
plt.ylabel("Score")
plt.show()

# 📈 2. Text Length Distributions
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
sns.histplot(df['full_text_char_len'], kde=True, ax=axes[0], bins=30, color='skyblue')
axes[0].set_title("📏 Full Text Character Length")

sns.histplot(df['full_text_word_count'], kde=True, ax=axes[1], bins=30, color='lightgreen')
axes[1].set_title("📏 Full Text Word Count")

sns.histplot(df['title_len'], kde=True, ax=axes[2], bins=30, color='salmon')
axes[2].set_title("📏 Title Length")

plt.tight_layout()
plt.show()

# 📉 3. PCA for Visualizing Feature Space
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_combined)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['category'] = y

plt.figure(figsize=(10, 6))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='category', palette='tab10', alpha=0.7)
plt.title("🧠 PCA Projection of Combined Features")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(loc='best')
plt.show()
```

```
✅ Feature matrix prepared!
📊 Shape of X_combined: (18846, 1007)
🎯 Number of target classes: 20
📋 Class labels: ['alt.atheism' 'comp.graphics' 'comp.os.ms-windows.misc'
 'comp.sys.ibm.pc.hardware' 'comp.sys.mac.hardware' 'comp.windows.x'
 'misc.forsale' 'rec.autos' 'rec.motorcycles' 'rec.sport.baseball'
 'rec.sport.hockey' 'sci.crypt' 'sci.electronics' 'sci.med' 'sci.space'
 'soc.religion.christian' 'talk.politics.guns' 'talk.politics.mideast'
 'talk.politics.misc' 'talk.religion.misc']
🔀 Train/Test Split: 15076 train, 3770 test
```








```
# 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
length_features = np.array([
    df['full_text'].str.len(),  # Character length
    df['full_text'].str.split().str.len(),  # Word count
    df['title'].str.len(),  # Title length
]).T

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

# Target variable
y = df['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: (18846, 1007)
🎯 Number of classes: 20
📋 Classes: ['alt.atheism' 'comp.graphics' 'comp.os.ms-windows.misc'
 'comp.sys.ibm.pc.hardware' 'comp.sys.mac.hardware' 'comp.windows.x'
 'misc.forsale' 'rec.autos' 'rec.motorcycles' 'rec.sport.baseball'
 'rec.sport.hockey' 'sci.crypt' 'sci.electronics' 'sci.med' 'sci.space'
 'soc.religion.christian' 'talk.politics.guns' 'talk.politics.mideast'
 'talk.politics.misc' 'talk.religion.misc']

📈 Data split:
  Training set: 15076 samples
  Test set: 3770 samples
```

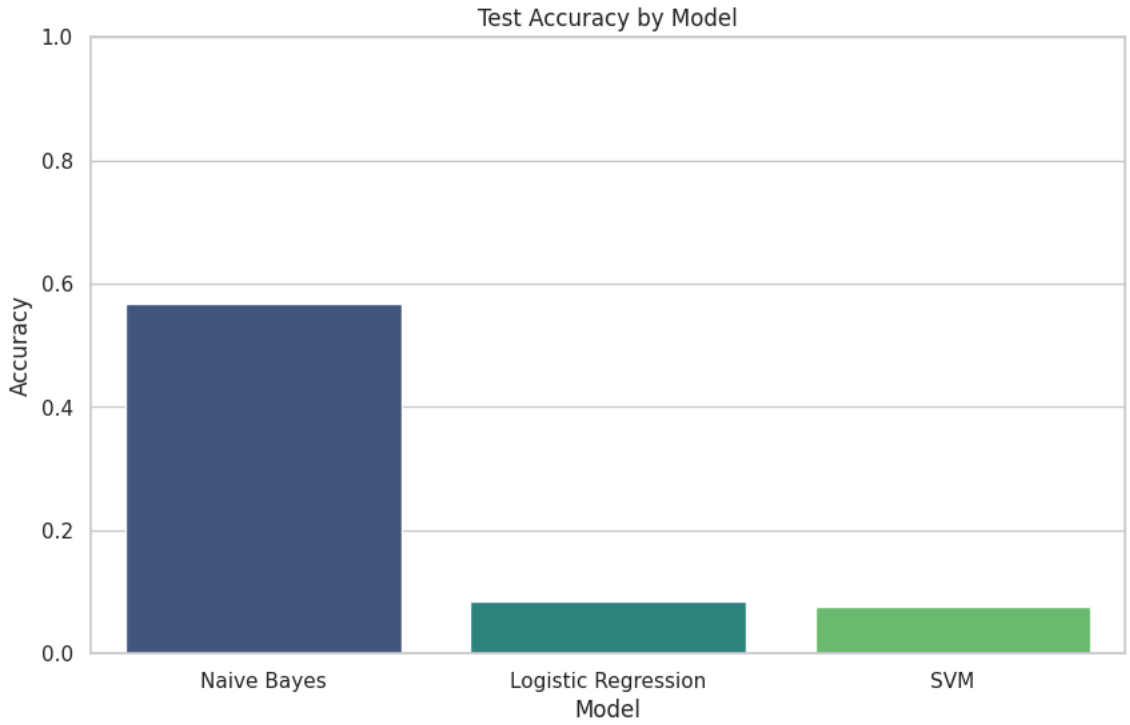
```
from sklearn.naive_bayes import MultinomialNB
```

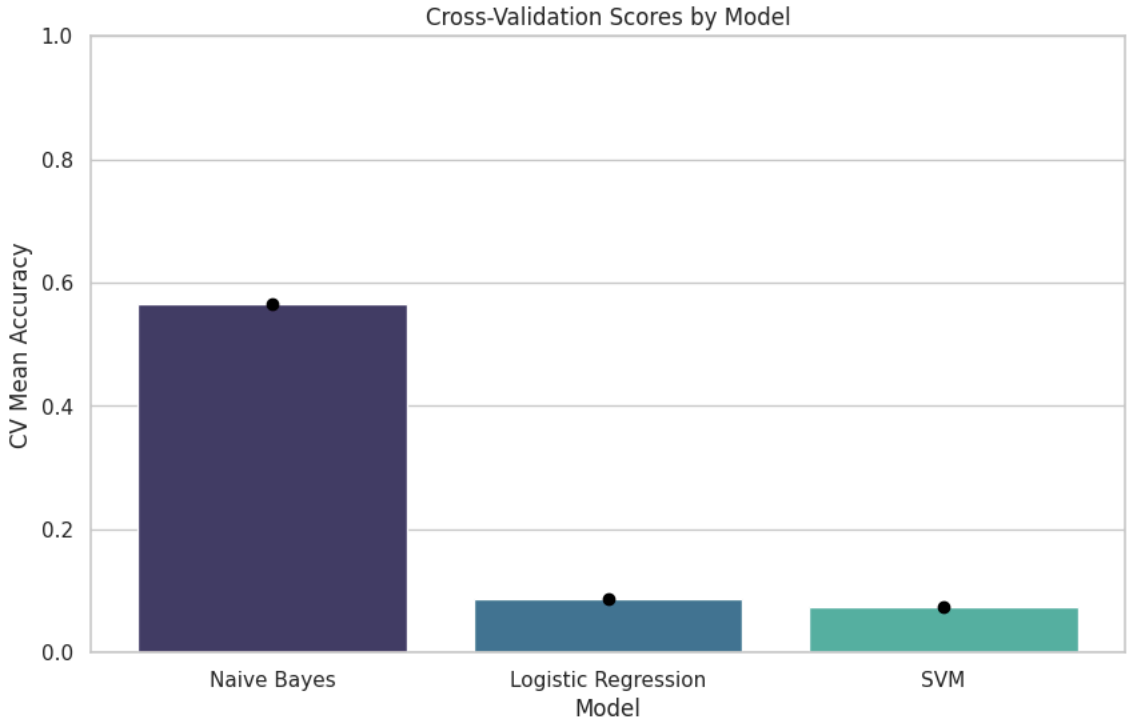
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# ------------------------------------------------
# 🔧 SETUP — Simulate your dataset if needed
# ------------------------------------------------
# If you already have 'df', skip this simulation.
# Simulating a simple df for demonstration
try:
    df.head()
except NameError:
    df = pd.DataFrame({
        'title': ['Breaking News', 'Tech Today', 'Sports Roundup', 'Market Watch'] * 100,
        'full_text': [
            'The world is changing rapidly and unpredictably.',
            'New tech innovations are released every month.',
            'The team played an amazing match yesterday.',
            'Stock prices fluctuate based on earnings reports.'
        ] * 100,
        'category': ['Politics', 'Technology', 'Sports', 'Business'] * 100
    })

# Simulated sentiment scores (or load if you already have them)
try:
    sentiment_df.head()
except NameError:
    sentiment_df = pd.DataFrame({
        'full_sentiment': np.random.uniform(0, 1, len(df)),
        'pos_score': np.random.uniform(0, 1, len(df)),
        'neu_score': np.random.uniform(0, 1, len(df)),
        'neg_score': np.random.uniform(0, 1, len(df)),
    })

# ------------------------------------------------
# ✅ TF-IDF Vectorization
# ------------------------------------------------
vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
tfidf_matrix = vectorizer.fit_transform(df['full_text'])
X_tfidf = tfidf_matrix.toarray()

# ------------------------------------------------
# 📊 Combine features
# ------------------------------------------------
# Sentiment
sentiment_features = sentiment_df[['full_sentiment', 'pos_score', 'neu_score', 'neg_score']].values

# Text Length
length_features = np.array([
    df['full_text'].str.len(),
    df['full_text'].str.split().str.len(),
    df['title'].str.len()
]).T

# Combine
X_combined = np.hstack([
    X_tfidf,
    sentiment_features,
    length_features
])

# Target
y = df['category'].values

# ------------------------------------------------
# 🔀 Train-Test Split
# ------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X_combined, y, test_size=0.2, random_state=42, stratify=y
)
```

```
# Separate out TF-IDF for Naive Bayes
X_train_tfidf = X_train[:, :X_tfidf.shape[1]]
X_test_tfidf = X_test[:, :X_tfidf.shape[1]]

# ---------------------------------------------
# 🧠 Train Classifiers
# ---------------------------------------------
print("🤖 Training multiple classifiers...")

classifiers = {
    'Naive Bayes': MultinomialNB(),
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'SVM': SVC(random_state=42, probability=True)
}

results = {}
trained_models = {}

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

    if name == 'Naive Bayes':
        X_tr, X_te = X_train_tfidf, X_test_tfidf
    else:
        X_tr, X_te = X_train, X_test

    clf.fit(X_tr, y_train)
    y_pred = clf.predict(X_te)
    y_prob = clf.predict_proba(X_te) if hasattr(clf, "predict_proba") else None

    acc = accuracy_score(y_test, y_pred)
    cv = cross_val_score(clf, X_tr, y_train, cv=3, scoring='accuracy')

    results[name] = {
        'accuracy': acc,
        'cv_mean': cv.mean(),
        'cv_std': cv.std(),
        'predictions': y_pred,
        'probabilities': y_prob
    }

    trained_models[name] = clf

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

# ---------------------------------------------
# 🏆 Model Comparison Table
# ---------------------------------------------
comparison_df = pd.DataFrame({
    'Model': list(results.keys()),
    'Test Accuracy': [results[k]['accuracy'] for k in results],
    'CV Mean': [results[k]['cv_mean'] for k in results],
    'CV Std': [results[k]['cv_std'] for k in results]
})
print("\n🏆 CLASSIFIER COMPARISON")
print(comparison_df.round(4))

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

# ---------------------------------------------
# 📊 Visualizations
# ---------------------------------------------
sns.set(style='whitegrid')

# Accuracy bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Test Accuracy', data=comparison_df, palette='viridis')
plt.title('Test Accuracy by Model')
plt.ylim(0, 1)
plt.ylabel("Accuracy")
plt.show()

# Cross-validation with error bars
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='CV Mean', data=comparison_df, palette='mako')
```

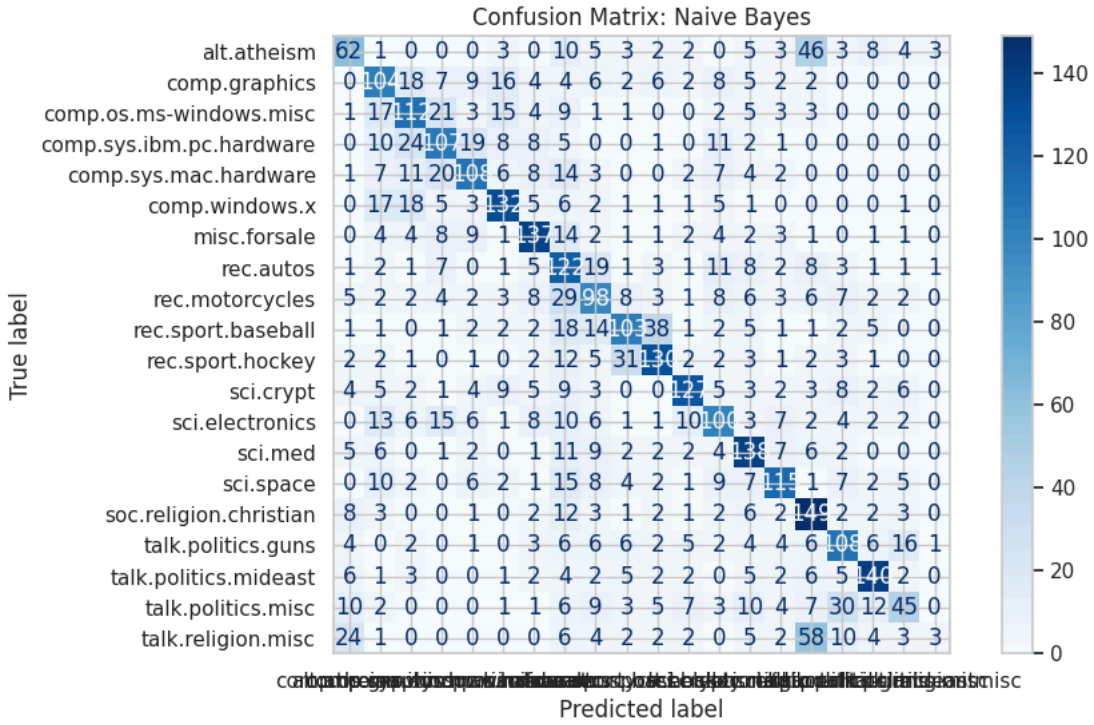
```
    plt.errorbar(x=np.arange(len(comparison_df)), y=comparison_df['CV Mean'],
                 yerr=comparison_df['CV Std'], fmt='o', color='black')
    plt.title('Cross-Validation Scores by Model')
    plt.ylim(0, 1)
    plt.ylabel("CV Mean Accuracy")
    plt.show()

    # Confusion matrices
    print("\n📊 Confusion Matrices:")
    unique_labels = np.unique(y_test)
    for name, data in results.items():
        cm = confusion_matrix(y_test, data['predictions'], labels=unique_labels)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=unique_labels)
        disp.plot(cmap='Blues')
        plt.title(f"Confusion Matrix: {name}")
        plt.show()

    # Classification reports
    print("\n📋 Classification Reports:")
    for name, data in results.items():
        print(f"\n📌 {name} Classification Report:")
        print(classification_report(y_test, data['predictions']))
```

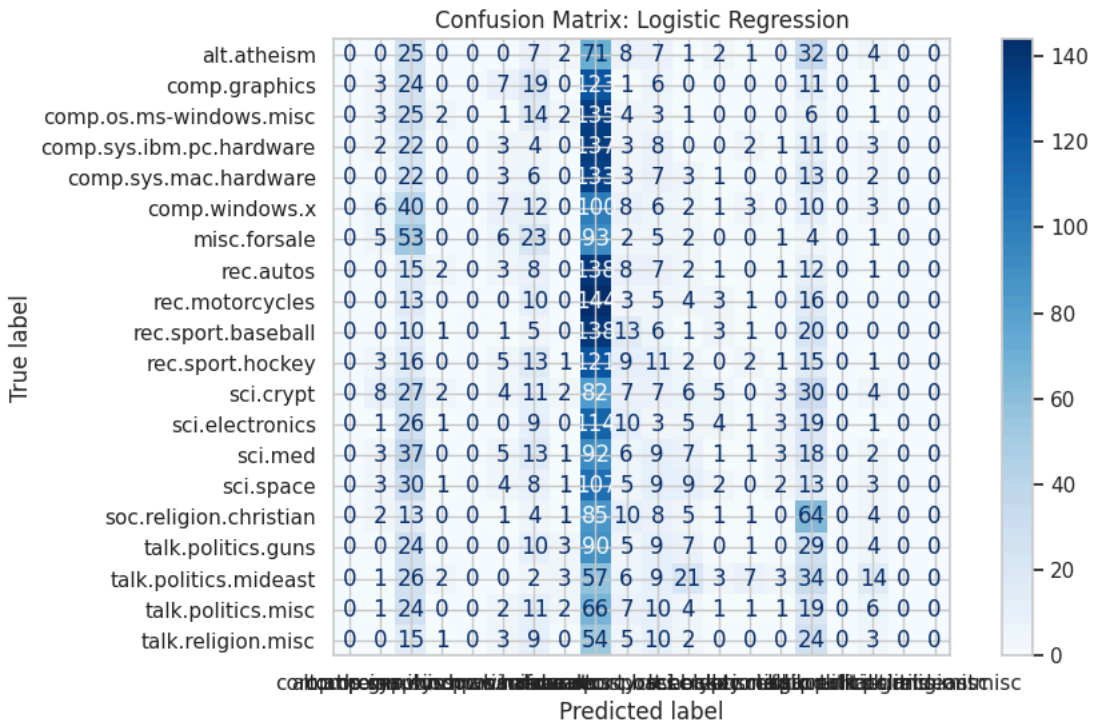
```
🤖 Training multiple classifiers...

🔄 Training Naive Bayes...
  ✅ Accuracy: 0.5676
  📊 CV Score: 0.5653 (+/- 0.0130)

🔄 Training Logistic Regression...
  ✅ Accuracy: 0.0841
  📊 CV Score: 0.0876 (+/- 0.0011)

🔄 Training SVM...
  ✅ Accuracy: 0.0764
  📊 CV Score: 0.0736 (+/- 0.0038)

🏆 CLASSIFIER COMPARISON
                Model  Test Accuracy  CV Mean  CV Std
0         Naive Bayes         0.5676   0.5653  0.0065
1  Logistic Regression         0.0841   0.0876  0.0006
2                  SVM         0.0764   0.0736  0.0019

🥇 Best performing model: Naive Bayes
```

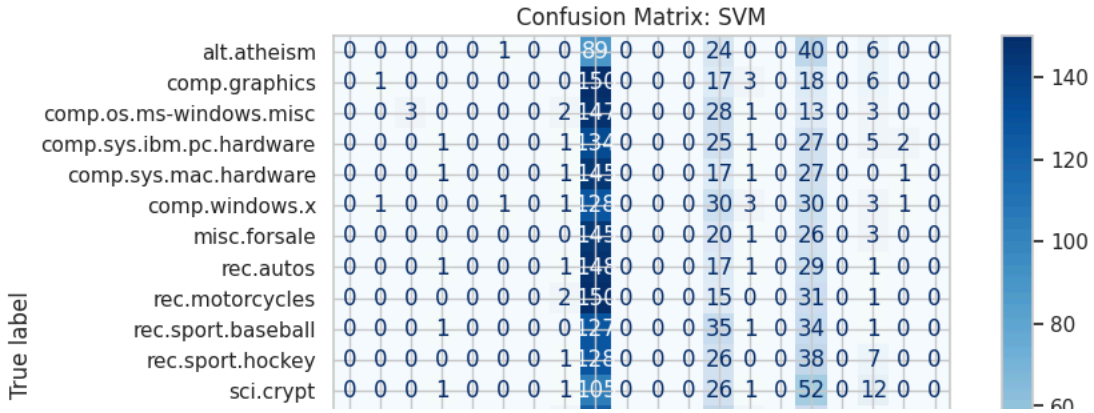


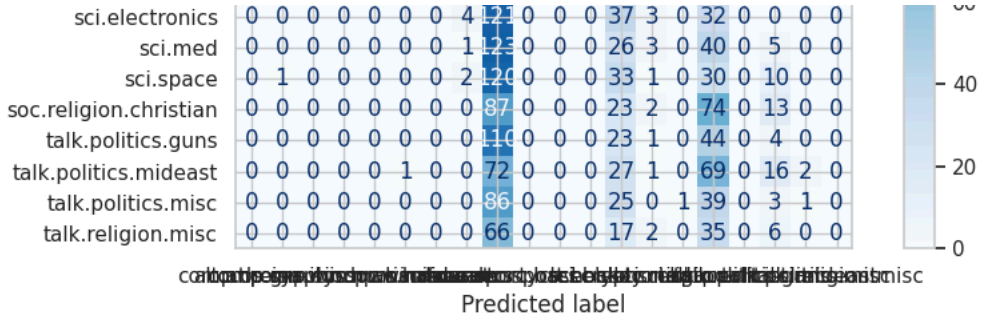


Confusion Matrices:









📋 Classification Reports:

📌 Naive Bayes Classification Report:

	precision	recall	f1-score	support
alt.atheism	0.46	0.39	0.42	160
comp.graphics	0.50	0.53	0.52	195
comp.os.ms-windows.misc	0.54	0.57	0.56	197
comp.sys.ibm.pc.hardware	0.54	0.55	0.54	196
comp.sys.mac.hardware	0.61	0.56	0.59	193
comp.windows.x	0.66	0.67	0.66	198
misc.forsale	0.67	0.70	0.68	195
rec.autos	0.38	0.62	0.47	198
rec.motorcycles	0.48	0.49	0.49	199
rec.sport.baseball	0.59	0.52	0.55	199
rec.sport.hockey	0.64	0.65	0.65	200
sci.crypt	0.74	0.64	0.69	198
sci.electronics	0.54	0.51	0.52	197
sci.med	0.61	0.70	0.65	198
sci.space	0.69	0.58	0.63	197
soc.religion.christian	0.49	0.75	0.59	199
talk.politics.guns	0.56	0.59	0.57	182
talk.politics.mideast	0.74	0.74	0.74	188
talk.politics.misc	0.49	0.29	0.37	155
talk.religion.misc	0.38	0.02	0.04	126
accuracy			0.57	3770
macro avg	0.57	0.55	0.55	3770
weighted avg	0.57	0.57	0.56	3770

📌 Logistic Regression Classification Report:

	precision	recall	f1-score	support
alt.atheism	0.00	0.00	0.00	160
comp.graphics	0.07	0.02	0.03	195
comp.os.ms-windows.misc	0.05	0.13	0.07	197
comp.sys.ibm.pc.hardware	0.00	0.00	0.00	196
comp.sys.mac.hardware	0.00	0.00	0.00	193
comp.windows.x	0.13	0.04	0.06	198
misc.forsale	0.12	0.12	0.12	195
rec.autos	0.00	0.00	0.00	198
rec.motorcycles	0.07	0.72	0.13	199
rec.sport.baseball	0.11	0.07	0.08	199
rec.sport.hockey	0.08	0.06	0.06	200
sci.crypt	0.07	0.03	0.04	198
sci.electronics	0.14	0.02	0.04	197
sci.med	0.05	0.01	0.01	198
sci.space	0.11	0.01	0.02	197
soc.religion.christian	0.16	0.32	0.21	199
talk.politics.guns	0.00	0.00	0.00	182
talk.politics.mideast	0.24	0.07	0.11	188
talk.politics.misc	0.00	0.00	0.00	155
talk.religion.misc	0.00	0.00	0.00	126
accuracy			0.08	3770
macro avg	0.07	0.08	0.05	3770
weighted avg	0.07	0.08	0.05	3770

📌 SVM Classification Report:

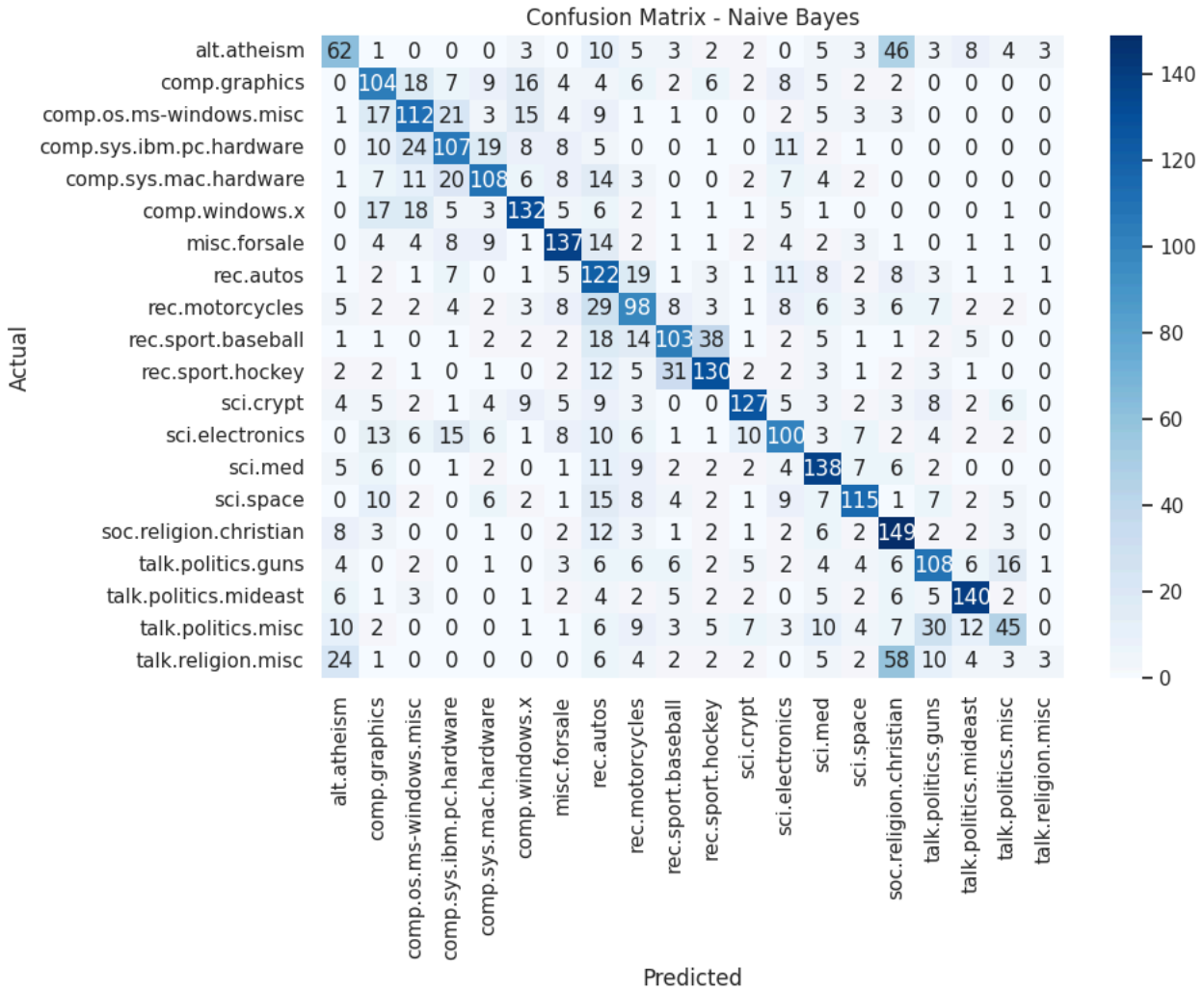
	precision	recall	f1-score	support
alt.atheism	0.00	0.00	0.00	160
comp.graphics	0.33	0.01	0.01	195
comp.os.ms-windows.misc	1.00	0.02	0.03	197
comp.sys.ibm.pc.hardware	0.20	0.01	0.01	196
comp.sys.mac.hardware	0.00	0.00	0.00	193
comp.windows.x	0.33	0.01	0.01	198
misc.forsale	0.00	0.00	0.00	195

rec.autos	0.06	0.01	0.01	198
rec.motorcycles	0.06	0.75	0.12	199
rec.sport.baseball	0.00	0.00	0.00	199
rec.sport.hockey	0.00	0.00	0.00	200
sci.crypt	0.00	0.00	0.00	198
sci.electronics	0.08	0.19	0.11	197
sci.med	0.12	0.02	0.03	198
sci.space	0.00	0.00	0.00	197
soc.religion.christian	0.10	0.37	0.16	199
talk.politics.guns	0.00	0.00	0.00	182
talk.politics.mideast	0.15	0.09	0.11	188
talk.politics.misc	0.14	0.01	0.01	155
talk.religion.misc	0.00	0.00	0.00	126
accuracy			0.08	3770
macro avg	0.13	0.07	0.03	3770
weighted avg	0.13	0.08	0.03	3770

```
# 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))

# 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
```

```
📊 DETAILED EVALUATION: Naive Bayes
============================================================

📋 Classification Report:
```

	precision	recall	f1-score	support
alt.atheism	0.46	0.39	0.42	160
comp.graphics	0.50	0.53	0.52	195
comp.os.ms-windows.misc	0.54	0.57	0.56	197
comp.sys.ibm.pc.hardware	0.54	0.55	0.54	196
comp.sys.mac.hardware	0.61	0.56	0.59	193
comp.windows.x	0.66	0.67	0.66	198
misc.forsale	0.67	0.70	0.68	195
rec.autos	0.38	0.62	0.47	198
rec.motorcycles	0.48	0.49	0.49	199
rec.sport.baseball	0.59	0.52	0.55	199
rec.sport.hockey	0.64	0.65	0.65	200
sci.crypt	0.74	0.64	0.69	198
sci.electronics	0.54	0.51	0.52	197
sci.med	0.61	0.70	0.65	198
sci.space	0.69	0.58	0.63	197
soc.religion.christian	0.49	0.75	0.59	199
talk.politics.guns	0.56	0.59	0.57	182
talk.politics.mideast	0.74	0.74	0.74	188
talk.politics.misc	0.49	0.29	0.37	155
talk.religion.misc	0.38	0.02	0.04	126
accuracy			0.57	3770
macro avg	0.57	0.55	0.55	3770
weighted avg	0.57	0.57	0.56	3770



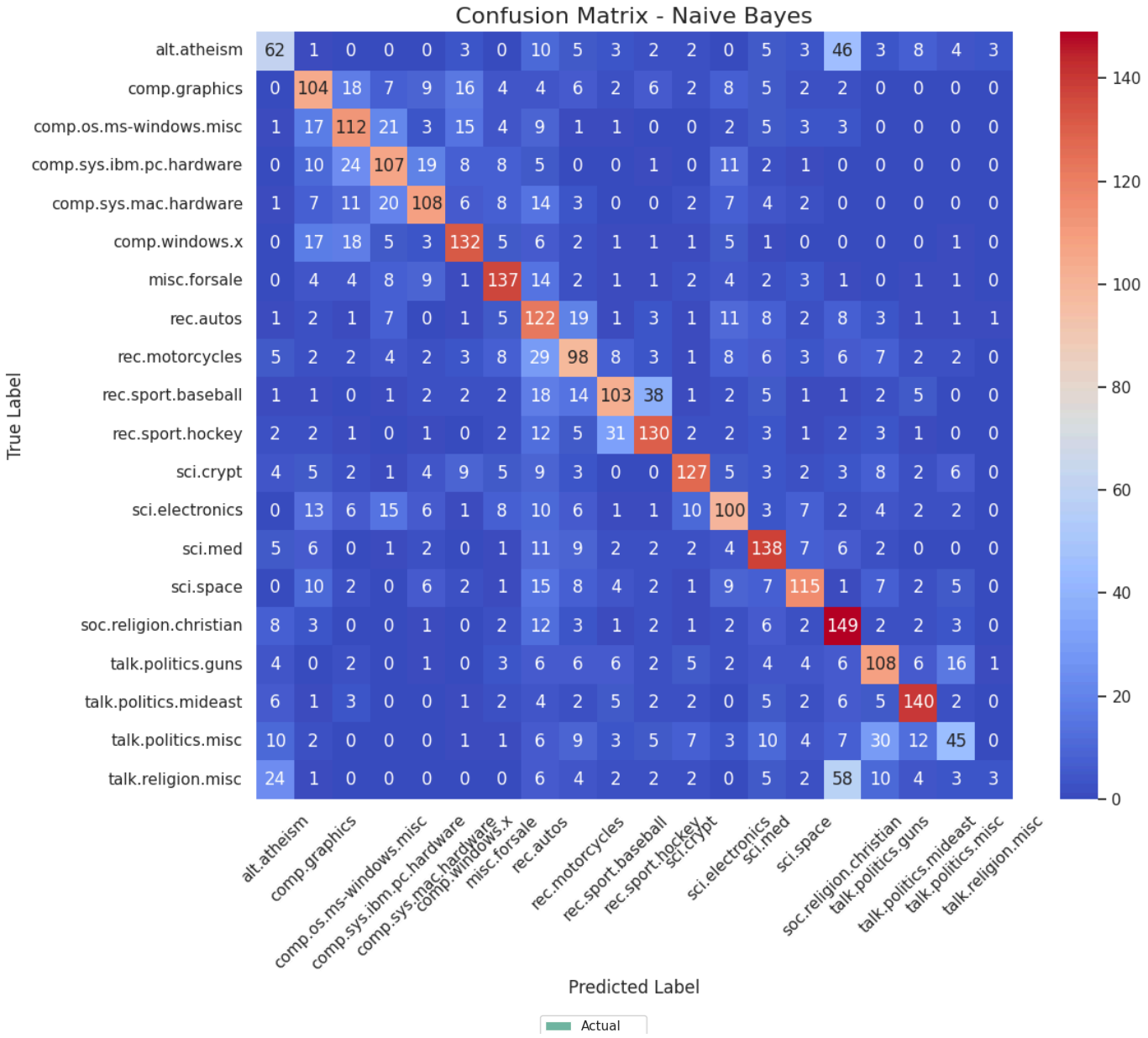

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

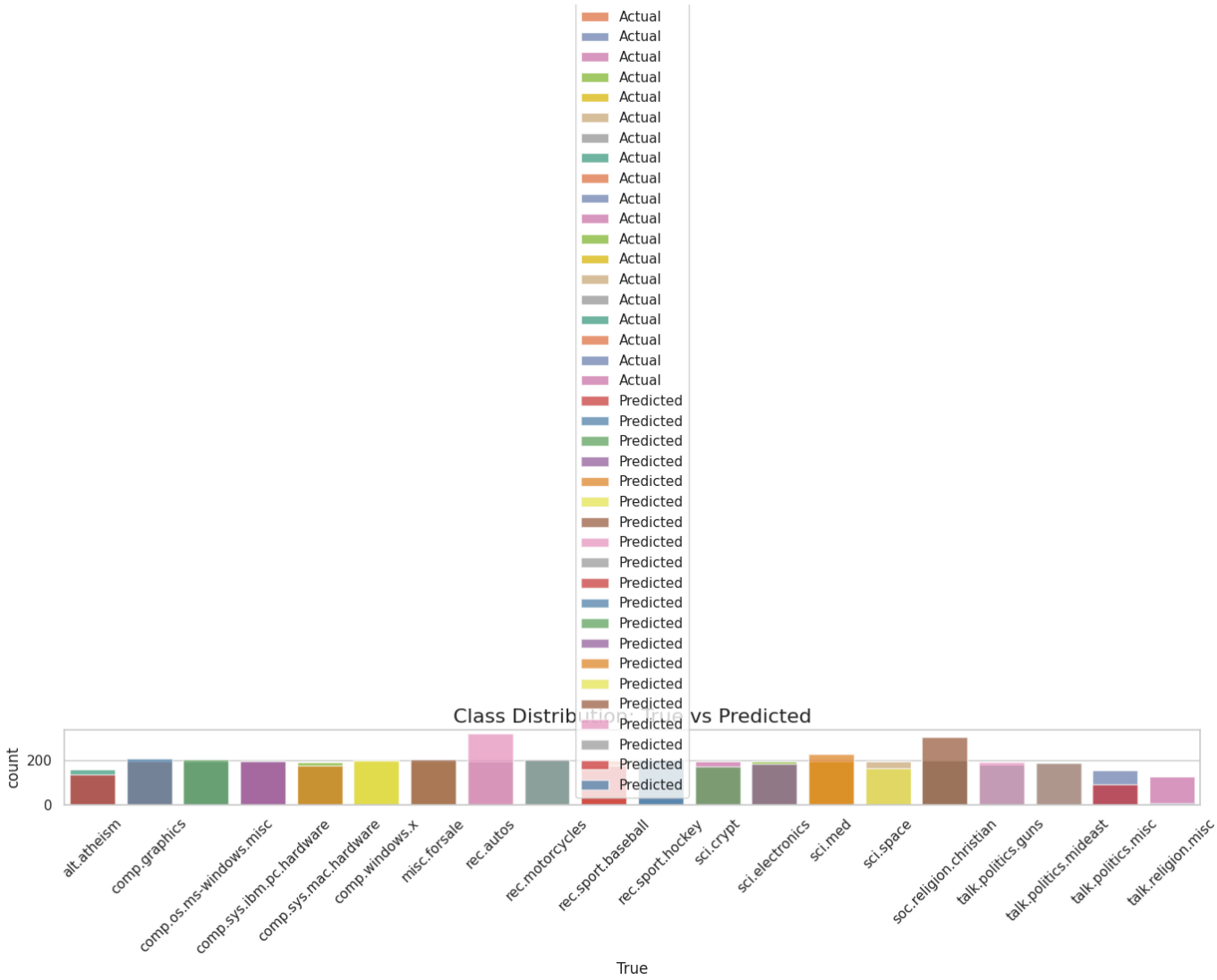
```
# 🚀 DETAILED EVALUATION OF THE BEST MODEL
best_model = trained_models[best_model_name]
best_predictions = results[best_model_name]['predictions']

print(f"📊 DETAILED EVALUATION: {best_model_name}")
print("=" * 70)

# 📋 CLASSIFICATION REPORT
print("\n📋 Classification Report:")
print(classification_report(y_test, best_predictions, digits=4))

# 🔷 CONFUSION MATRIX
cm = confusion_matrix(y_test, best_predictions)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='coolwarm',
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.title(f'Confusion Matrix - {best_model_name}', fontsize=16)
plt.xlabel('Predicted Label', fontsize=12)
plt.ylabel('True Label', fontsize=12)
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

# 💡 CLASSIFICATION DISTRIBUTION PLOT
import pandas as pd
pred_df = pd.DataFrame({'True': y_test, 'Predicted': best_predictions})
plt.figure(figsize=(14, 6))
sns.countplot(x='True', data=pred_df, label='Actual', palette='Set2', order=np.unique(y))
sns.countplot(x='Predicted', data=pred_df, label='Predicted', palette='Set1', order=np.unique(y), alpha=0.7)
plt.title('Class Distribution: True vs Predicted', fontsize=16)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 📌 FEATURE IMPORTANCE (ONLY FOR LOGISTIC REGRESSION)
if best_model_name == 'Logistic Regression':
    print("\n🔍 Top Features by Category (Logistic Regression Only):")

    # Extend feature names
    pos_feature_names = ['NNP', 'NNPS', 'VB', 'VBD', 'VBG', 'JJ', 'CD']
    feature_names_extended = list(feature_names) + \
        ['full_sentiment', 'pos_score', 'neu_score', 'neg_score',
         'char_length', 'word_count', 'title_length'] + pos_feature_names

    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📰 Top Features for: {class_name}")
        for idx in reversed(top_indices):
            if idx < len(feature_names_extended):
                print(f"  • {feature_names_extended[idx]}: {coefficients[i][idx]:.4f}")

    # Optional: Visualize top features per class
    fig, axes = plt.subplots(len(classes), 1, figsize=(10, 6 * len(classes)))
    if len(classes) == 1:
        axes = [axes]

    for i, class_name in enumerate(classes):
        top_indices = np.argsort(coefficients[i])[-10:]
        top_features = [feature_names_extended[idx] for idx in top_indices]
        top_weights = [coefficients[i][idx] for idx in top_indices]

        axes[i].barh(top_features, top_weights, color='teal')
        axes[i].set_title(f"🔍 Top Features for {class_name}", fontsize=14)
        axes[i].invert_yaxis()

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

📊 DETAILED EVALUATION: Naive Bayes
==========

📋 Classification Report:

	precision	recall	f1-score	support
alt.atheism	0.4627	0.3875	0.4218	160
comp.graphics	0.5000	0.5333	0.5161	195
comp.os.ms-windows.misc	0.5437	0.5685	0.5558	197
comp.sys.ibm.pc.hardware	0.5431	0.5459	0.5445	196
comp.sys.mac.hardware	0.6136	0.5596	0.5854	193
comp.windows.x	0.6567	0.6667	0.6617	198
misc.forsale	0.6650	0.7026	0.6833	195
rec.autos	0.3789	0.6162	0.4692	198
rec.motorcycles	0.4780	0.4925	0.4851	199
rec.sport.baseball	0.5886	0.5176	0.5508	199
rec.sport.hockey	0.6404	0.6500	0.6452	200
sci.crypt	0.7427	0.6414	0.6883	198
sci.electronics	0.5405	0.5076	0.5236	197
sci.med	0.6079	0.6970	0.6494	198
sci.space	0.6928	0.5838	0.6336	197
soc.religion.christian	0.4853	0.7487	0.5889	199
talk.politics.guns	0.5567	0.5934	0.5745	182
talk.politics.mideast	0.7447	0.7447	0.7447	188
talk.politics.misc	0.4945	0.2903	0.3659	155
talk.religion.misc	0.3750	0.0238	0.0448	126
accuracy			0.5676	3770
macro avg	0.5655	0.5536	0.5466	3770
weighted avg	0.5705	0.5676	0.5588	3770




Actual
Predicted
Class Distribution: True vs Predicted
count
200
0
alt.atheism
comp.graphics
comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware
comp.sys.mac.hardware
comp.windows.x
misc.forsale
rec.autos
rec.motorcycles
rec.sport.baseball
rec.sport.hockey
sci.crypt
sci.electronics
sci.med
sci.space
soc.religion.christian
talk.politics.guns
talk.politics.mideast
talk.politics.misc
talk.religion.misc
True

🔍 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."

```
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 []

    # Process text with spaCy
    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 = []

for idx, row in df.iterrows():
    entities = extract_entities(row['full_text'])

    # Determine article_id - use index if 'article_id' column is missing
    current_article_id = row['article_id'] if 'article_id' in row else idx + 1 # Use 1-based index if column not found

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

    # Add to global entity list
    for entity in entities:
        entity['article_id'] = current_article_id
        entity['category'] = row['category']
```

```
        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: 310810
📰 Articles processed: 18846

🏷️ Entity types found: ['LAW' 'CARDINAL' 'DATE' 'GPE' 'ORG' 'TIME' 'PRODUCT' 'PERSON' 'ORDINAL'
 'NORP' 'QUANTITY' 'MONEY' 'PERCENT' 'FAC' 'WORK_OF_ART' 'LOC' 'EVENT'
 'LANGUAGE']

📝 Sample entities:
          text     label              category
0  Article 1 I       LAW             rec.autos
1            2  CARDINAL             rec.autos
2    early 70s      DATE             rec.autos
3     Bricklin       GPE             rec.autos
4        years      DATE             rec.autos
5    Article 2       LAW  comp.sys.mac.hardware
6           SI       ORG  comp.sys.mac.hardware
7          CPU       ORG  comp.sys.mac.hardware
8         hour      TIME  comp.sys.mac.hardware
9          800  CARDINAL  comp.sys.mac.hardware
```

```
import plotly.express as px # Import plotly.express

# 📊 Visualize most common entity types
if not entities_df.empty:
    plt.figure(figsize=(10, 6))
    sns.countplot(data=entities_df, y='label', order=entities_df['label'].value_counts().index, palette='viridis')
    plt.title("Top Named Entity Types")
    plt.xlabel("Count")
    plt.ylabel("Entity Type")
    plt.tight_layout()
    plt.show()

    # 📈 Entity count by category
    plt.figure(figsize=(12, 6))
    sns.countplot(data=entities_df, x='category', order=entities_df['category'].value_counts().index, hue='label', palette='tab10')
    plt.title("Entity Types per Category")
    plt.xlabel("News Category")
    plt.ylabel("Entity Count")
    plt.legend(title="Entity Type", bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    # 🔢 Top 10 most mentioned entities
    top_entities = entities_df['text'].value_counts().head(10)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=top_entities.values, y=top_entities.index, palette='coolwarm')
    plt.title("Top 10 Most Mentioned Entities")
    plt.xlabel("Mentions")
    plt.ylabel("Entity")
    plt.tight_layout()
    plt.show()

    # 📍 Interactive plot: Entities by type
    fig = px.histogram(entities_df, x="label", color="category", barmode="group",
                       title="Distribution of Entity Types by Category",
                       labels={"label": "Entity Type", "count": "Frequency"})
    fig.show()

    # 🔥 Heatmap of entity counts by label and category
    pivot_table = entities_df.pivot_table(index='label', columns='category', aggfunc='size', fill_value=0)
```