```
# Section 0: Setup
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import string
import re
import spacy
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag, RegexpParser
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

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

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

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

```
# Section 1: Load Dataset
# Load fake and real news datasets
fake_df = pd.read_csv('Fake.csv')
true_df = pd.read_csv('True.csv')

# Add labels
fake_df['label'] = 'FAKE'
true_df['label'] = 'REAL'

# Combine datasets
df = pd.concat([fake_df, true_df], ignore_index=True)

# Combine title and text
df['content'] = df['title'] + ' ' + df['text']

# Shuffle the dataset
df = df.sample(frac=1, random_state=42).reset_index(drop=True)

# Display dataset information
print("Dataset shape:", df.shape)
print(df['label'].value_counts())
```

```
Dataset shape: (44898, 6)
label
FAKE    23481
REAL    21417
Name: count, dtype: int64
```

```
import nltk
import string
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

# Make sure required NLTK resources are downloaded
```

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

# Define stopwords and punctuation
stop_words = set(stopwords.words('english'))
punctuations = string.punctuation

# Function for text normalization
def normalize_text(text):
    if not isinstance(text, str):
        return []

    # Sentence tokenization
    sentences = sent_tokenize(text)

    # Word tokenization and normalization
    normalized_sentences = []
    for sentence in sentences:
        words = word_tokenize(sentence)
        words = [word.lower() for word in words if word.isalpha()]  # Remove punctuation/numbers
        words = [word for word in words if word not in stop_words]
        normalized_sentences.append(words)
    return normalized_sentences

# Ensure the 'content' column exists and contains valid text
if 'content' in df.columns and not df['content'].isna().all():
    sample_text = df['content'].dropna().iloc[0]
    normalized_sample = normalize_text(sample_text)
    print("Normalized Sample:", normalized_sample)
else:
    print("[ERROR] 'content' column is missing or contains no valid text.")
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
Normalized Sample: [['ben', 'stein', 'calls', 'circuit', 'court', 'committed', 'coup', 'état', 'constitution', 'century', 'wire',
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# Section 3: Stemming and Lemmatization
# Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Function to compare stemming and lemmatization
def compare_stem_lemma(words):
    stemmed = [stemmer.stem(word) for word in words]
    lemmatized = [lemmatizer.lemmatize(word) for word in words]
    return stemmed, lemmatized

# Select a sample of words
sample_words = normalized_sample[0][:10]
stemmed_words, lemmatized_words = compare_stem_lemma(sample_words)

# Display comparison
print("Original Words:", sample_words)
print("Stemmed Words:", stemmed_words)
print("Lemmatized Words:", lemmatized_words)
```

```
Original Words: ['ben', 'stein', 'calls', 'circuit', 'court', 'committed', 'coup', 'état', 'constitution', 'century']
Stemmed Words: ['ben', 'stein', 'call', 'circuit', 'court', 'commit', 'coup', 'état', 'constitut', 'centuri']
Lemmatized Words: ['ben', 'stein', 'call', 'circuit', 'court', 'committed', 'coup', 'état', 'constitution', 'century']
```

```
# Section 4: POS Tagging and Chunking
# POS tagging
import nltk
nltk.download('averaged_perceptron_tagger_eng')
pos_tags = pos_tag(sample_words)
print("POS Tags:", pos_tags)

# Define a chunk grammar and parser
grammar = "NP: {<DT>?<JJ>*<NN>}"
```

```
chunk_parser = RegexpParser(grammar)
tree = chunk_parser.parse(pos_tags)
print("Chunking Result:")
print(tree)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
POS Tags: [('ben', 'NN'), ('stein', 'NN'), ('calls', 'VBZ'), ('circuit', 'NN'), ('court', 'NN'), ('committed', 'VBD'), ('coup', '
Chunking Result:
(S
  (NP ben/NN)
  (NP stein/NN)
  calls/VBZ
  (NP circuit/NN)
  (NP court/NN)
  committed/VBD
  (NP coup/NN)
  état/NNP
  (NP constitution/NN)
  (NP century/NN))
```

```
# Section 5: Named Entity Recognition (NER)
# Apply spaCy NER
doc = nlp(sample_text)
print("Named Entities:")
for ent in doc.ents:
    print(f"{ent.text} -> {ent.label_}")
```

```
Named Entities:
Ben Stein Calls -> PERSON
9th Circuit Court -> ORG
the Constitution 21st Century Wire -> LAW
Ben Stein -> PERSON
Pepperdine University -> ORG
Hollywood -> GPE
Ferris Bueller -> PERSON
Jeanine Pirro -> PERSON
Trump s Executive -> PERSON
Stein -> PERSON
the 9th Circuit Court -> ORG
Washington -> GPE
Stein -> PERSON
Seattle -> GPE
the Executive Order -> ORG
21st Century -> DATE
```

```
# Section 6: WordNet
# Explore synonyms and definitions using WordNet
synonyms = []
for syn in wordnet.synsets("news"):
    for lemma in syn.lemmas():
        synonyms.append(lemma.name())
    print(f"Definition: {syn.definition()}")
    print(f"Examples: {syn.examples()}")
    print(f"Synonyms: {synonyms}")
    break  # Display only the first synset for brevity
```

```
Definition: information about recent and important events
Examples: ['they awaited news of the outcome']
Synonyms: ['news', 'intelligence', 'tidings', 'word']
```

```
# Section 7: Feature Extraction
# Apply TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
tfidf_matrix = tfidf_vectorizer.fit_transform(df['content'])

# Display top features
feature_names = tfidf_vectorizer.get_feature_names_out()
print("Top TF-IDF Features:", feature_names[:20])
```

```
Top TF-IDF Features: ['000' '10' '100' '11' '12' '13' '14' '15' '16' '18' '20' '2012' '2013'
 '2014' '2015' '2016' '2017' '24' '25' '30']
```

```
# Section 8: Document Similarity
# Compute cosine similarity between first two articles
cos_sim = cosine_similarity(tfidf_matrix[0], tfidf_matrix[1])
print(f"Cosine Similarity between first two articles: {cos_sim[0][0]:.4f}")
```

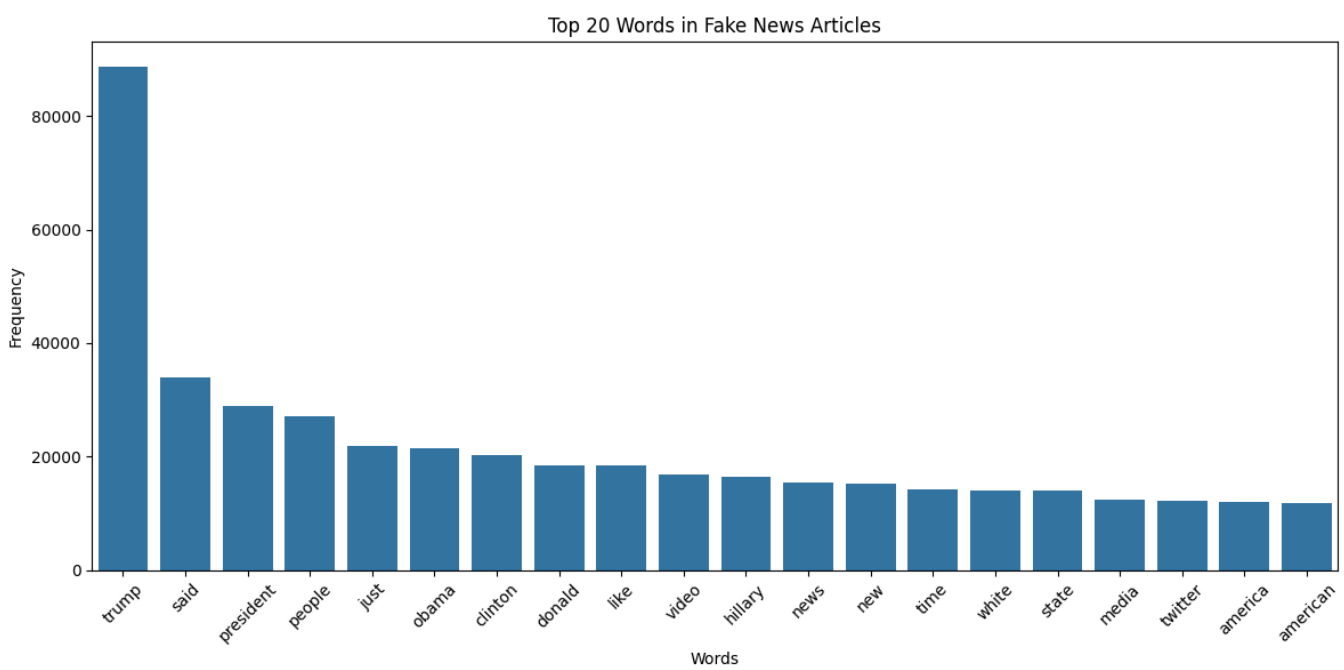
```
Cosine Similarity between first two articles: 0.3846
```

```
#  Section 9: Word Frequency Plot
# Filter fake news articles
fake_articles = df[df['label'] == 'FAKE']

# Apply CountVectorizer
count_vectorizer = CountVectorizer(stop_words='english', max_features=20)
word_counts = count_vectorizer.fit_transform(fake_articles['content'])

# Sum word frequencies
sum_words = word_counts.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in count_vectorizer.vocabulary_.items()]
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)

# Plot top 20 words
words, counts = zip(*words_freq)
plt.figure(figsize=(12, 6))
sns.barplot(x=list(words), y=list(counts))
plt.title("Top 20 Words in Fake News Articles")
plt.xlabel("Words")
plt.ylabel("Frequency")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




0
/
200