UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

***

Data Mining Final Project(Fake News and Real News Detection) - MOD - A

Prof.ROBERTA SICILIANO and Prof.GIUSEPPE LONGO

Team Members :

1. MAVILLAPALLI VENKATA TARUN KUMAR (P37000126)
2. GATTEM PRIYA MADHURI (P37000162)
3. KARRI RAHUL REDDY (P37000157)

Data Source Website - "https://data-flair.training/blogs/advanced-python-project-detecting-fake-news/" - Detecting Fake News

Problem Statement:

Domain Context Understanding:

- The domain context of the dataset is US political news. This encompasses all aspects related to the political landscape, events, policies, and discussions in the United States. US politics is a complex and dynamic field that involves multiple stakeholders, including political parties, government officials, interest groups, and the general public. The news plays a crucial role in informing citizens about political developments and shaping public opinion.

Real-World Case Study:

- The spread of misinformation and fake news has become a significant challenge in today's digital age. The dissemination of false information can have severe consequences on individuals, communities, and society as a whole. Detecting and combating fake news is crucial for maintaining the integrity of information and ensuring informed decision-making.

Objective:

- 1.Understand the data by conducting a thorough exploratory analysis of the dataset to gain deep insights into its structure and characteristics, focusing on understanding the distinctions, patterns between real and fake news articles.
- 2.Perform data pre-processing & feature engineering to try various machine learning and deep learning models and develop a model for fake news detection. The system aims to

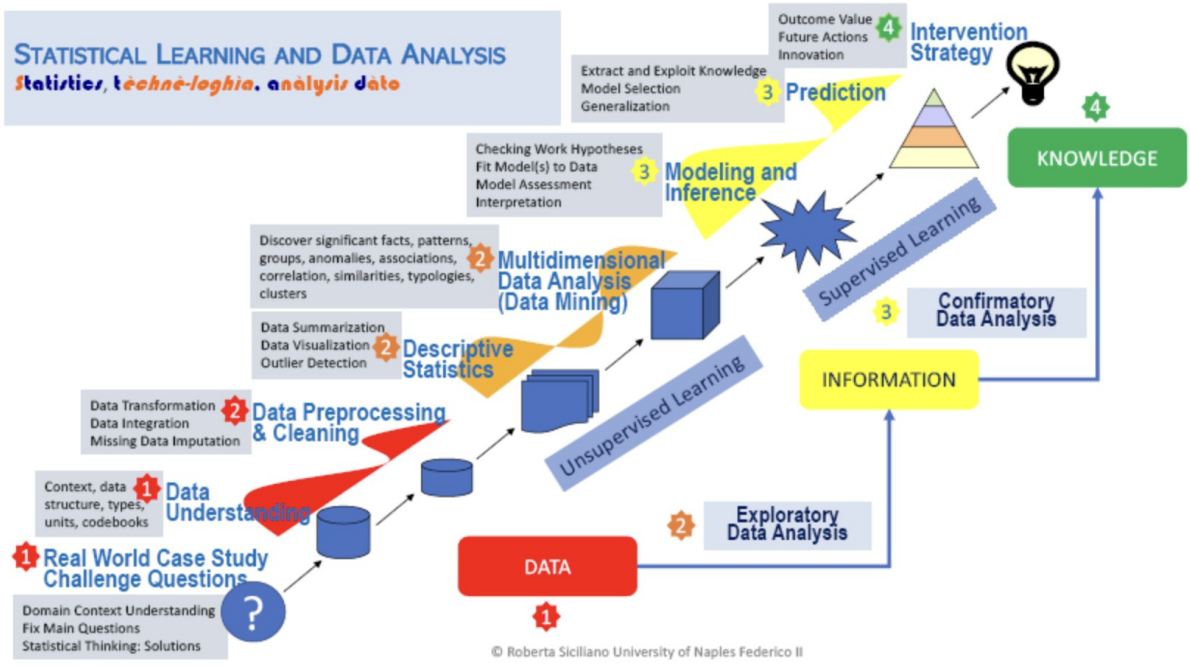
automatically classify news articles as either real or fake based on their content and linguistic features. By accurately identifying fake news, the system can assist users in making informed judgments and help prevent the further spread of false information.

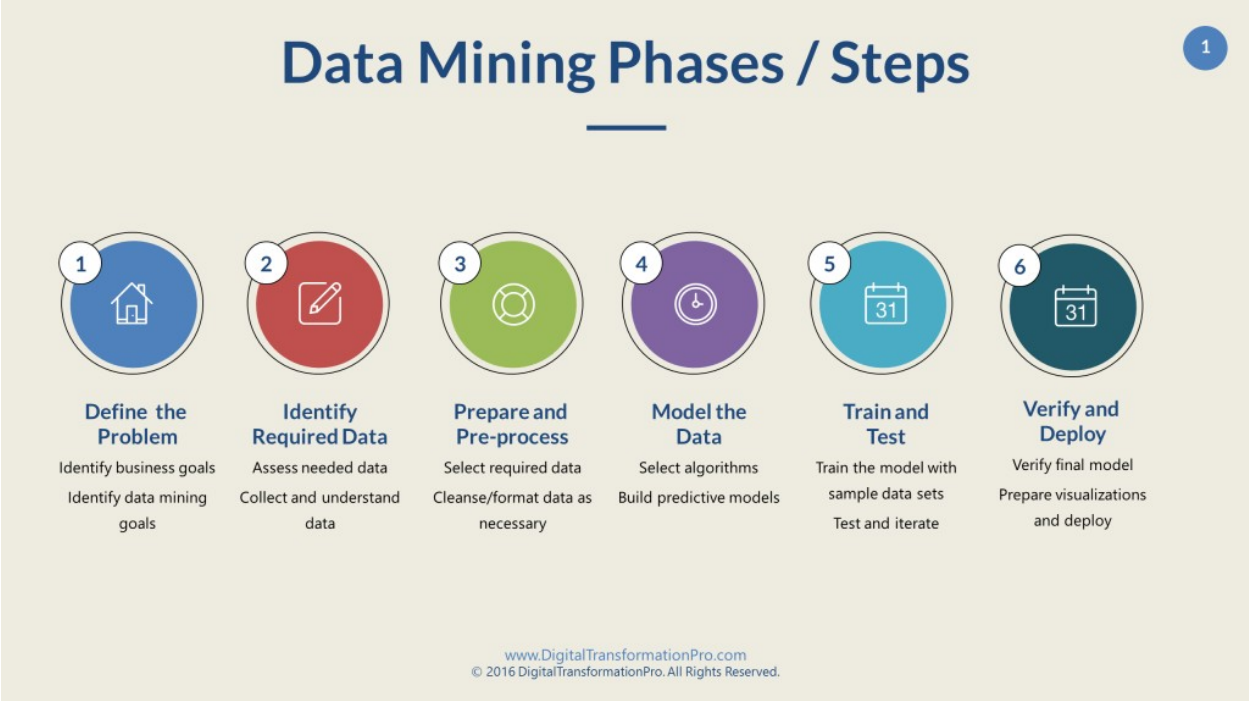
- Overall, this project aims to contribute to the development of effective techniques and tools for fake news detection, ultimately fostering a more informed and trustworthy information ecosystem.

Methodology

To develop this project we follow a structured statistical learning and data analysis pipeline to guide our methodology. Starting with data understanding, we explore the dataset's structure and define key questions. We then proceed with data preprocessing and cleaning to handle missing values, standardize features, and prepare the data for analysis. Using descriptive statistics and exploratory data analysis (EDA), we uncover patterns, relationships, and outliers. Moving into the core of the project, we apply multidimensional data analysis (data mining) techniques, including clustering and association rule mining, to discover hidden patterns.

Statistical Approach Pipeline





Data Set Description:

The dataset has 4 different features out of those 3 are independent features and 1 (Label) is the target feature.

- a. Unnamed(News Identification): This column contains a unique identifier or index for each news article. It helps distinguish one article from another within the dataset.
- a. Title: The second column contains the titles of the news articles. Titles are typically short and concise summaries that provide a glimpse into the content of the article. In this case, the titles likely describe the main topic or event discussed in each article.
- a. Text: The third column contains the actual text of the news articles. This is where we would find the main body of the article, providing detailed information, analysis, or opinions related to US politics. While title contains a sharp overview of the article and text contains indetailed context of title.
- a. Labels: The fourth column includes labels that indicate whether each news article is classified as "REAL" or "FAKE." The labeling allows for the classification of articles into two distinct categories.

There are 6335 entries in the dataset. Below table shows the metadata of the dataset:

Feature Name Type Description Unnamed int64 An undefined column with

numbers. title object The title of the news article. text object The main content of the news article. label object The category of the news article, either "Fake" or "Real".

Exploratory Data Analysis (EDA)

EDA is an essential step in understanding and preparing the dataset for fake news detection in Natural Language Processing (NLP). Although the dataset consists of text data and categorical variables, EDA can still provide valuable insights. The following are the steps we performed for EDA:

I. Data Overview:

1. **Data Shape and Size:**
   - Number of rows and columns in the dataset.
2. **Data Types:**
   - Data types of each column (e.g., title: string, text: string, label: categorical).
3. **Data Quality Issues:**
   - Identify and document any data quality issues (e.g., duplicates, outliers, missing values).

II. Data Cleaning:

1. **Handling Missing Values:**
   - **Detection:** Identify missing values in the dataset.
   - **Imputation:** Impute missing values using techniques such as mean, median, or mode.
   - **Removal:** Remove rows or columns with missing values if necessary.
2. **Handling Outliers:**
   - **Detection:** Identify outliers in the dataset using techniques such as Z-score or modified Z-score.
   - **Transformation:** Transform outliers using techniques such as logarithmic or square root transformation.
   - **Removal:** Remove outliers from the dataset if necessary.
3. **Handling Duplicate Records:**
   - **Detection:** Identify duplicate records in the dataset.
   - **Removal:** Remove duplicate records from the dataset.
4. **Language Detection:**
   - Detect the language of the text data to ensure consistency.
5. **Text Preprocessing:**
   - **Tokenization:** Split text into individual words or tokens.
   - **Stopword Removal:** Remove stopwords from the text data.
   - **HTML Tag Removal:** Remove HTML tags from the text data.
   - **Punctuation Removal:** Remove punctuation from the text data.
   - **Lemmatization:** Reduce words to their base form using lemmatization.
   - **Removing Special Characters:** Remove special characters from the text data.

- **Removing Non-ASCII Characters:** Remove non-ASCII characters from the text data.

III. Univariate Analysis:

1. **Title Column:**
   - **Title Length Analysis:**
     - Highest Word Count.
     - Lowest Word Count.
     - Most Common Words in Titles.
     - Title Character Distribution.
     - Distribution of Title Word Count.
   - **Title Sentiment Analysis.**
2. **Text Column:**
   - **Text Length Analysis:**
     - Highest Word Count.
     - Lowest Word Count.
     - Most Common Words in Texts.
   - **Text Readability Analysis.**
   - **Text Sentiment Analysis.**
3. **Label Column:**
   - Class Distribution.
   - Class Balance Analysis.

IV. Bivariate Analysis:

1. **Title vs. Label:**
   - Top 10 Most Frequent Words in "Real" and "Fake" Articles.
   - Distribution of Word Count for "Fake" and "Real" Titles.
   - Shapiro-Wilk Test for Fake and Real Titles.
   - Two-sample T-test for Fake and Real Titles.
   - **Title Length Analysis:**
     - Alphabetic Count.
     - Shapiro-Wilk Test for Alphabetic Count.
     - Two-sample T-test for Alphabetic Count.
     - Title Length by Label.
     - Title Sentiment by Label.
2. **Text vs. Label:**
   - Top 10 Most Frequent Words in "Real" and "Fake" Articles.
   - Distribution of Word Count for "Fake" and "Real" Texts.
   - Shapiro-Wilk Test for Fake and Real Texts.
   - Two-sample T-test for Fake and Real Texts.
   - **Text Length Analysis:**
     - Alphabetic Count.
     - Shapiro-Wilk Test for Alphabetic Count.

- Two-sample T-test for Alphabetic Count.
- Text Length by Label.
- Text Sentiment by Label.

V. Hypothesis Tests/Statistical Tests:

1. **Text Analysis Tests:**
   - Use techniques like TF-IDF or word embeddings (e.g., Word2Vec, GloVe) to analyze the semantic differences between real and fake news articles.
   - Use statistical tests (e.g., permutation test) to determine if the semantic differences are significant.

VI. Text Analysis:

1. **Sentiment Analysis:**
   - Perform sentiment analysis on the text data to determine the sentiment polarity (positive, negative, neutral) of the articles. This can help in understanding the emotional tone of real vs. fake news.
2. **Co-occurrence Analysis:**
   - Analyze the co-occurrence of words in the dataset to identify common word pairs or groups. This can provide insights into frequently associated terms in real and fake news articles.
3. **Named Entity Recognition (NER):**
   - **Named Entity Distribution:** Bar chart of top named entities in titles and texts.
   - **Named Entity Co-occurrence:** Heatmap of named entity co-occurrences in titles and texts.
   - **Named Entity Sentiment:** Analyze sentiment towards specific named entities.
4. **Part-of-Speech (POS) Analysis:**
   - **POS Distribution:** Bar chart of top POS tags in titles and texts.
   - **POS Co-occurrence:** Heatmap of POS tag co-occurrences in titles and texts.
5. **Topic Modeling:**
   - **Topic Modeling (e.g., LDA or NMF):** Identify underlying topics in texts and visualize using dimensionality reduction techniques (e.g., t-SNE or PCA).
   - **Topic Coherence:** Calculate topic coherence scores to evaluate topic quality.
6. **Association Rule Mining:**
   - **Apriori Algorithm:** Apply the Apriori algorithm to identify frequent itemsets and generate association rules.
   - Association Rule Mining with Topic Modeling.
   - Association Rule Mining without Topic Modeling.
   - **Rule Evaluation:** Evaluate the generated association rules using metrics such as support, confidence, and lift.
   - **Rule Visualization:** Visualize the association rules using techniques such as graph visualization.

By conducting EDA, we can gain a deeper understanding of the dataset, uncover patterns and characteristics specific to fake news, and prepare the data for subsequent modeling and analysis. These insights will aid in developing robust and accurate fake news detection models.

Data Overview

Importing Necessary Libraries

```
### Core Libraries
# 1. General Purpose
import numpy as np
import pandas as pd
import re
import string
import matplotlib.pyplot as plt

### TensorFlow and Keras
# 2. TensorFlow & Keras
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D,
GlobalMaxPooling1D, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical

### Scikit-Learn
# 3. Scikit-Learn for Model Building and Evaluation
from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score,
recall_score, roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
import joblib
from scikeras.wrappers import KerasClassifier

### Natural Language Processing (NLP)
# 4. NLP Libraries
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from wordcloud import WordCloud
from langdetect import detect
from langdetect.lang_detect_exception import LangDetectException

### Optional Dependencies
# 5. Optional Dependencies
```

```
import seaborn as sns
import gensim
from sklearn.svm import SVC
from tensorflow.keras.layers import LSTM
```

Loading the dataset

```
df =
pd.read_csv('/Users/priyamadhurigattem/Downloads/DM_final/cleaned_news
.csv')
```

Data Information and Structure

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6335 entries, 0 to 6334
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  6335 non-null   int64 
 1   title       6335 non-null   object
 2   text        6335 non-null   object
 3   label       6335 non-null   object
dtypes: int64(1), object(3)
memory usage: 198.1+ KB
None
```

Data Shape

```
df.shape
```

```
(6335, 4)
```

Displaying the first 5 rows

```
df.head()
```

```
   Unnamed: 0                                              title  \
0        8476                       You Can Smell Hillary’s Fear
1       10294  Watch The Exact Moment Paul Ryan Committed Pol...
2        3608        Kerry to go to Paris in gesture of sympathy
3       10142  Bernie supporters on Twitter erupt in anger ag...
4         875   The Battle of New York: Why This Primary Matters

                                                text label
0  Daniel Greenfield, a Shillman Journalism Fello...  FAKE
1  Google Pinterest Digg Linkedin Reddit Stumbleu...  FAKE
2  U.S. Secretary of State John F. Kerry said Mon...  REAL
```

```
3  – Kaydee King (@KaydeeKing) November 9, 2016 T...  FAKE
4  It's primary day in New York and front-runners...  REAL
```

Displaying the last 5 rows

```
df.tail()
```

```
      Unnamed: 0                                              title  \
6330        4490  State Department says it can't find emails fro...
6331        8062  The ‘P’ in PBS Should Stand for ‘Plutocratic’ ...
6332        8622  Anti-Trump Protesters Are Tools of the Oligarc...
6333        4021  In Ethiopia, Obama seeks progress on peace, se...
6334        4330  Jeb Bush Is Suddenly Attacking Trump. Here's W...

                                                   text label
6330  The State Department told the Republican Natio...  REAL
6331  The ‘P’ in PBS Should Stand for ‘Plutocratic’ ...  FAKE
6332   Anti-Trump Protesters Are Tools of the Oligar...  FAKE
6333  ADDIS ABABA, Ethiopia —President Obama convene...  REAL
6334  Jeb Bush Is Suddenly Attacking Trump. Here's W...  REAL
```

Summary Statistics

```
#Performing summary statistics : Understand basic statistics about the
dataset, such as mean, median, standard deviation, etc.
summary_stats = df.describe()
print(summary_stats)
```

```
         Unnamed: 0
count   6335.000000
mean    5280.415627
std     3038.503953
min        2.000000
25%     2674.500000
50%     5271.000000
75%     7901.000000
max    10557.000000
```

The "Unnamed: 0" column serves as an identifier for each news article but lacks meaningful analysis value. For a comprehensive understanding, descriptive statistics for columns like title, text, or labels would be more relevant. Hence we removed Unnamed: 0 column.

Droping unnessory column

```
#### Droping the first column (Unnamed) the numerical column
df = df.iloc[:, 1:]

#### Performing descriptive statistics after removing first column(the
numerical column)
summary_stats = df.describe()
print(summary_stats)
```

```
                                  title  \
count                              6335
unique                             6256
top     OnPolitics | 's politics blog
freq                                  5

                                                      text label
count                                                 6335  6335
unique                                                6060     2
top     Killing Obama administration rules, dismantlin...  REAL
freq                                                    58  3171
```

Descriptive statistics after removing numerical column, we found that:

1. **Title Column:**
   - **Count:** There are 6335 entries in the dataset.
   - **Unique:** There are 6256 unique titles, indicating some duplication.
   - **Top Title:** "OnPolitics | 's politics blog" is the most frequent title, occurring 5 times.
   - **Frequency:** The titles exhibit some repetition, with a frequency of 5 for the top title.
2. **Text Column:**
   - **Count:** There are 6335 entries in the dataset.
   - **Unique:** There are 6060 unique text entries.
   - **Top Text:** The most frequent text entry is related to "Killing Obama administration rules, dismantlin..." but the frequency is not mentioned.
3. **Label Column:**
   - **Count:** There are 6335 entries in the dataset.
   - **Unique:** There are 2 unique labels (presumably indicating binary classification).
   - **Top Label:** "REAL" is the most frequent label, occurring 3171 times.

Data Cleaning

Data Cleaning: The dataset was cleaned and preprocessed using the following techniques:

- Handling Missing Values: Missing values were not detected .
- Handling Outliers: Outliers were detected using the Z-score method and removed from the dataset.
- Handling Duplicate Records: Duplicate records were detected and removed from the dataset.
- Language Detection : We tried to see the language of text and title.
- Text Preprocessing: The title and text columns were preprocessed using tokenization, stopword removal, HTML tag removal, punctuation removal, stemming/lemmatization, removal of special characters and non-ASCII characters, and lower casing.

1. Handling Missing Values

```
# Import necessary libraries
import pandas as pd
import numpy as np
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import Normalizer

print("Handling Missing Values:")
print(df.isnull().sum())  # Detection
```

```
Handling Missing Values:
title    0
text     0
label    0
dtype: int64
```

2. Handling Outliers

```
print("\nHandling Outliers:")
from scipy import stats
z_scores = np.abs(stats.zscore(df['text'].apply(len)))  # Detection
using Z-score
outliers = df[z_scores > 3]  # Identification
print(outliers)
```

```
Handling Outliers:
                                                  title  \
50      American politics has reached peak polarization
84            John Kerry: ISIS responsible for genocide
96    CNN: One voter can make a difference by voting...
137   Re: 10 Things That Every American Should Know ...
178   FACT CHECK: Hillary Clinton's Speech To The De...
...                                                 ...
5939              The accidental case against Obamacare
6105                     2:00PM Water Cooler 11/4/2016
6224  Is Huma Abedin Hillary Clinton’s Secret Weapon...
6240                   Defying the Politics of Fear   :
6309      How the Obama White House runs foreign policy

                                                   text label
50    For a long time in American politics, we've be...  REAL
84    (CNN) Secretary of State John Kerry said Thurs...  REAL
96    Channel list \nFollowing hurricane Matthew's f...  FAKE
137   Archives Michael On Television 10 Things That ...  FAKE
178   Editor's note: This has been updated at 1:25 p...  REAL
```

```
...                                                   ...   ...
5939  Now, the sales pitch is over. The Supreme Cour...  REAL
6105  By Lambert Strether of Corrente . \nTPP/TTIP/T...  FAKE
6224  Faced with an unending scandal about her use o...  REAL
6240   Defying the Politics of Fear By Chris Hedges\...  FAKE
6309  When Susan E. Rice took over as President Obam...  REAL

[106 rows x 3 columns]
```

```
print(z_scores)
```

```
0       0.552150
1       0.404917
2       0.425150
3       0.402166
4       0.563249
          ...
6330    0.124004
6331    1.888939
6332    1.427497
6333    0.448625
6334    0.021756
Name: text, Length: 6335, dtype: float64
```

```
num_outliers = (z_scores > 3).sum()
print(f"Number of outliers: {num_outliers}")
```

```
Number of outliers: 106
```

```
df = df[z_scores <= 3]  # Removal
```

3. Handling Duplicate Records

```
# Detect duplicates
duplicates = df.duplicated()

# Count the number of duplicate records
num_duplicates = duplicates.sum()

print(f"Number of duplicate records: {num_duplicates}")
```

```
Number of duplicate records: 29
```

```
df = df[~duplicates]  # Removal
```

4. Language Detection

```
# Function for detecting the language of the text
from langdetect import detect, LangDetectException
def detect_language(text):
    try:
```

```
        return detect(text)
    except LangDetectException:
        return "unknown"

# Applying language detection to the 'title' and 'text' columns
df['title_language'] = df['title'].apply(detect_language)
df['text_language'] = df['text'].apply(detect_language)

# Checking if any article is in a language other than English
target_language = 'en'  # Setting our target language (e.g., English)
df['is_other_language'] = ((df['title_language'] != target_language) |
(df['text_language'] != target_language))

# Counting the number of articles that are not in English
num_non_english_articles = df['is_other_language'].sum()

# Printing the number of articles that are not in English
print(f"Number of articles not in {target_language}:
{num_non_english_articles}")

# Printing the first few rows (first 5 rows) of non-English articles
print("First few non-English articles:")
print(df[df['is_other_language']].head())

# Printing the last few rows (last 5 rows) of non-English articles
print("\nLast few non-English articles:")
print(df[df['is_other_language']].tail())
```

```
Number of articles not in en: 286
First few non-English articles:
                                                 title  \
5                                          Tehran, USA
7                  ‘Britain’s Schindler’ Dies at 106
25  Anti-Trump forces seek last-ditch delegate revolt
65                      Rubio’s parting shot at Trump
89           Where Does Bernie Sanders Go From Here?

                                                 text label
title_language  \
5     \nI’m not an immigrant, but my grandparents ...  FAKE
id
7   A Czech stockbroker who saved more than 650 Je...  REAL
de
25  Washington (CNN) The faction of the GOP that i...  REAL
ro
65  It was not supposed to end like this for Marco...  REAL
tl
89  NEW YORK - Bernie Sanders is at a crossroads.\...  REAL
de
```

```
   text_language  is_other_language
5             en               True
7             en               True
25            en               True
65            en               True
89            en               True

Last few non-English articles:
                                                      title  \
6282  Chinese University Sells HIV Testing Kits in V...
6297         Muslim debate seizes GOP presidential race
6301  Here Are Six ‘Miracle’ Drugs Big Pharma Now Re...
6323  Bernie Sanders says private meeting with Pope ...
6328  Radio Derb Is On The Air—Leonardo And Brazil’s...

                                                   text label
title_language  \
6282  Chinese University Sells HIV Testing Kits in V...  FAKE
de
6297  A debate over Islam -- first sparked by Donald...  REAL
ro
6301  Here Are Six ‘Miracle’ Drugs Big Pharma Now Re...  FAKE
de
6323  ROME —  U.S. Democratic presidential candidate...  REAL
nl
6328                                                     FAKE
en

     text_language  is_other_language
6282            en               True
6297            en               True
6301            en               True
6323            en               True
6328       unknown               True
```

In our investigation to determine the presence of other languages within our entire dataset, we encountered the following observations:

1. We have found that there are other languges(id,de...) in title column.
2. This is due to titles were shorter in length and if they contain any abbreviations or punctuation the langdetect library was not detecting the title's language correctly.
3. When used on text column we found that there is an 'Unknown' in text_langugae column and when we further digged into it, we found that there are some blank or empty columns in text which were identifed as unknown.

Identify rows where the language is unknown for 'text'

```
unknown_language_rows = df[(df['text_language'] == 'unknown')]
```

```python
# Printing the number of articles with unknown language
num_unknown_language_articles = len(unknown_language_rows)
print(f"Number of articles with unknown language:
{num_unknown_language_articles}")

# Print all the rows with unknown language
print("\nRows with unknown language:")
print(unknown_language_rows)
```

```
Number of articles with unknown language: 37

Rows with unknown language:
                                                  title           text
label  \
106   The Arcturian Group by Marilyn Raffaele Octobe...
FAKE
710   MARKETWATCH LEFTIST: MSM’s “Blatant” Anti Trum...
FAKE
806   Southern Poverty Law Center Targets Anti-Jihad...
FAKE
919   Refugee Resettlement Watch: Swept Away In Nort...
FAKE
940   Michael Bloomberg Names Technological Unemploy...
FAKE
1664  Alert News : Putins Army Is Coming For World W...
FAKE
1736  An LDS Reader Takes A Look At Trump Accuser Je...
FAKE
1851  America’s Senator Jeff Sessions Warns of Worse...
FAKE
1883  Paris Migrant Campers Increase after Calais Is...
FAKE
1941  Putins Army is coming for World war 3 against ...
FAKE
2244  Is your promising internet career over now Vin...
FAKE
2426  Radio Derb Transcript For October 21 Up: The M...
FAKE
2576  A Reader Refers Us To Englishman Pat Condell O...
FAKE
2650                  2009 FLASHBACK: “What If” Remixed  11/08/2016
FAKE
2662  “Donald Trump And The Rise Of White Identity I...
FAKE
2788          Hope for the best, prepare for the worst…
FAKE
2832  The Comey Confrontation: In Our New Third-Worl...
FAKE
3073  NATIONAL REVIEW, Conservatism Inc., Plan To Ca...
FAKE
```

```
3350  Thomas Frank Explores Whether Hillary Clinton ...
FAKE
3511  Democrats Playing Class Card To Split the Whit...
FAKE
3641  Comment software has been rolled back to old v...
FAKE
3642  Round Up the Unusual Suspects: Moneyball Nerds...
FAKE
4014  More on Trump’s Populism and How It Can Be Con...
FAKE
4142  Radio Derb transcript for October 29th is up: ...
FAKE
4253  Pro-sovereignty Legislators Demand That Admini...
FAKE
4713                                        World War 3?
FAKE
4744  A Mormon Reader Says Most Mormons Will Still B...
FAKE
5017            Paris: Riot Police Flatten Invader Camp
FAKE
5088                           Huma Abedin’s Muslim Dad
FAKE
5213  Hillary is Sick & Tired of Suffering from Wein...
FAKE
5581  Automation: Robots from Korea to America Are R...
FAKE
5639                   WORLD WAR 3 – HILLARY V.S. TRUMP
FAKE
5699              A Fifth Clinton Presidency? Hill, No!
FAKE
5772                          Huma’s Weiner Dogs Hillary
FAKE
6064  Radio Derb: Peak White Guilt, PC Now To The LE...
FAKE
6175  Hillary’s High Crimes & Misdemeanors Threaten ...
FAKE
6328  Radio Derb Is On The Air—Leonardo And Brazil’s...
FAKE

     title_language text_language  is_other_language
106              en       unknown               True
710              id       unknown               True
806              en       unknown               True
919              en       unknown               True
940              en       unknown               True
1664             en       unknown               True
1736             en       unknown               True
1851             en       unknown               True
1883             en       unknown               True
```

```
1941              en       unknown                True
2244              en       unknown                True
2426              en       unknown                True
2576              en       unknown                True
2650              en       unknown                True
2662              en       unknown                True
2788              en       unknown                True
2832              en       unknown                True
3073              en       unknown                True
3350              en       unknown                True
3511              en       unknown                True
3641              en       unknown                True
3642              en       unknown                True
4014              en       unknown                True
4142              en       unknown                True
4253              en       unknown                True
4713              en       unknown                True
4744              en       unknown                True
5017              en       unknown                True
5088              id       unknown                True
5213              en       unknown                True
5581              en       unknown                True
5639              de       unknown                True
5699              en       unknown                True
5772              en       unknown                True
6064              en       unknown                True
6175              en       unknown                True
6328              en       unknown                True
```

We found that there about 37 rows where text is not present, so we droped them.

Remove rows with unknown language for the 'text' column

```
df = df.drop(unknown_language_rows.index)

# Reset the index after dropping rows
df = df.reset_index(drop=True)

# Printing the DataFrame after removing rows with unknown language
print("\nDataFrame after removing rows with unknown language:")
print(df)
```

```
DataFrame after removing rows with unknown language:
                                                title  \
0                            You Can Smell Hillary’s Fear
1     Watch The Exact Moment Paul Ryan Committed Pol...
2           Kerry to go to Paris in gesture of sympathy
3     Bernie supporters on Twitter erupt in anger ag...
4      The Battle of New York: Why This Primary Matters
```

```
...                                                   ...
6158  State Department says it can't find emails fro...
6159  The ‘P’ in PBS Should Stand for ‘Plutocratic’ ...
6160  Anti-Trump Protesters Are Tools of the Oligarc...
6161  In Ethiopia, Obama seeks progress on peace, se...
6162  Jeb Bush Is Suddenly Attacking Trump. Here's W...

                                                  text label
title_language  \
0     Daniel Greenfield, a Shillman Journalism Fello...  FAKE
en
1     Google Pinterest Digg Linkedin Reddit Stumbleu...  FAKE
en
2     U.S. Secretary of State John F. Kerry said Mon...  REAL
en
3     — Kaydee King (@KaydeeKing) November 9, 2016 T...  FAKE
en
4     It's primary day in New York and front-runners...  REAL
en
...                                                 ...   ...
...
6158  The State Department told the Republican Natio...  REAL
en
6159  The ‘P’ in PBS Should Stand for ‘Plutocratic’ ...  FAKE
en
6160   Anti-Trump Protesters Are Tools of the Oligar...  FAKE
en
6161  ADDIS ABABA, Ethiopia —President Obama convene...  REAL
en
6162  Jeb Bush Is Suddenly Attacking Trump. Here's W...  REAL
en

     text_language  is_other_language
0               en              False
1               en              False
2               en              False
3               en              False
4               en              False
...            ...                ...
6158            en              False
6159            en              False
6160            en              False
6161            en              False
6162            en              False

[6163 rows x 6 columns]

df.shape

(6163, 6)
```

```
df.head()
```

```
                                                   title  \
0                       You Can Smell Hillary’s Fear
1  Watch The Exact Moment Paul Ryan Committed Pol...
2        Kerry to go to Paris in gesture of sympathy
3  Bernie supporters on Twitter erupt in anger ag...
4   The Battle of New York: Why This Primary Matters

                                                text label
title_language  \
0  Daniel Greenfield, a Shillman Journalism Fello...  FAKE
en
1  Google Pinterest Digg Linkedin Reddit Stumbleu...  FAKE
en
2  U.S. Secretary of State John F. Kerry said Mon...  REAL
en
3  — Kaydee King (@KaydeeKing) November 9, 2016 T...  FAKE
en
4  It's primary day in New York and front-runners...  REAL
en

  text_language  is_other_language
0            en              False
1            en              False
2            en              False
3            en              False
4            en              False
```

Identify rows where the language is unknown for 'title'

```
# Identify rows where the language is unknown for 'title'
unknown_language_rowss = df[(df['title_language'] == 'unknown')]

# Printing the number of articles with unknown language
num_unknown_language_articles = len(unknown_language_rowss)
print(f"Number of articles with unknown language:
{num_unknown_language_articles}")

# Print all the rows with unknown language
print("\nRows with unknown language:")
print(unknown_language_rowss)
```

```
Number of articles with unknown language: 1

Rows with unknown language:
     title                                              text label  \
3371     :   We the People Against Tyranny: Seven Princi...  FAKE

     title_language text_language  is_other_language
3371        unknown            en               True
```

We found that there were 1 rows where title is empty

Remove rows with unknown row for the 'title' column

```
df = df.drop(unknown_language_rowss.index)

# Reset the index after dropping rows
df = df.reset_index(drop=True)

# Printing the DataFrame after removing rows with unknown language
print("\nDataFrame after removing rows with unknown language:")
print(df)
```

```
DataFrame after removing rows with unknown language:
                                                  title  \
0                          You Can Smell Hillary’s Fear
1     Watch The Exact Moment Paul Ryan Committed Pol...
2           Kerry to go to Paris in gesture of sympathy
3     Bernie supporters on Twitter erupt in anger ag...
4      The Battle of New York: Why This Primary Matters
...                                                 ...
6157  State Department says it can't find emails fro...
6158  The ‘P’ in PBS Should Stand for ‘Plutocratic’ ...
6159  Anti-Trump Protesters Are Tools of the Oligarc...
6160  In Ethiopia, Obama seeks progress on peace, se...
6161  Jeb Bush Is Suddenly Attacking Trump. Here's W...

                                                  text label
title_language  \
0     Daniel Greenfield, a Shillman Journalism Fello...  FAKE
en
1     Google Pinterest Digg Linkedin Reddit Stumbleu...  FAKE
en
2     U.S. Secretary of State John F. Kerry said Mon...  REAL
en
3     — Kaydee King (@KaydeeKing) November 9, 2016 T...  FAKE
en
4     It's primary day in New York and front-runners...  REAL
en
...                                                 ...   ...
...
6157  The State Department told the Republican Natio...  REAL
en
6158  The ‘P’ in PBS Should Stand for ‘Plutocratic’ ...  FAKE
en
6159   Anti-Trump Protesters Are Tools of the Oligar...  FAKE
en
6160  ADDIS ABABA, Ethiopia —President Obama convene...  REAL
en
6161  Jeb Bush Is Suddenly Attacking Trump. Here's W...  REAL
```

```
en

     text_language  is_other_language
0               en              False
1               en              False
2               en              False
3               en              False
4               en              False
...            ...                ...
6157            en              False
6158            en              False
6159            en              False
6160            en              False
6161            en              False

[6162 rows x 6 columns]
```

```
df.shape
```

```
(6162, 6)
```

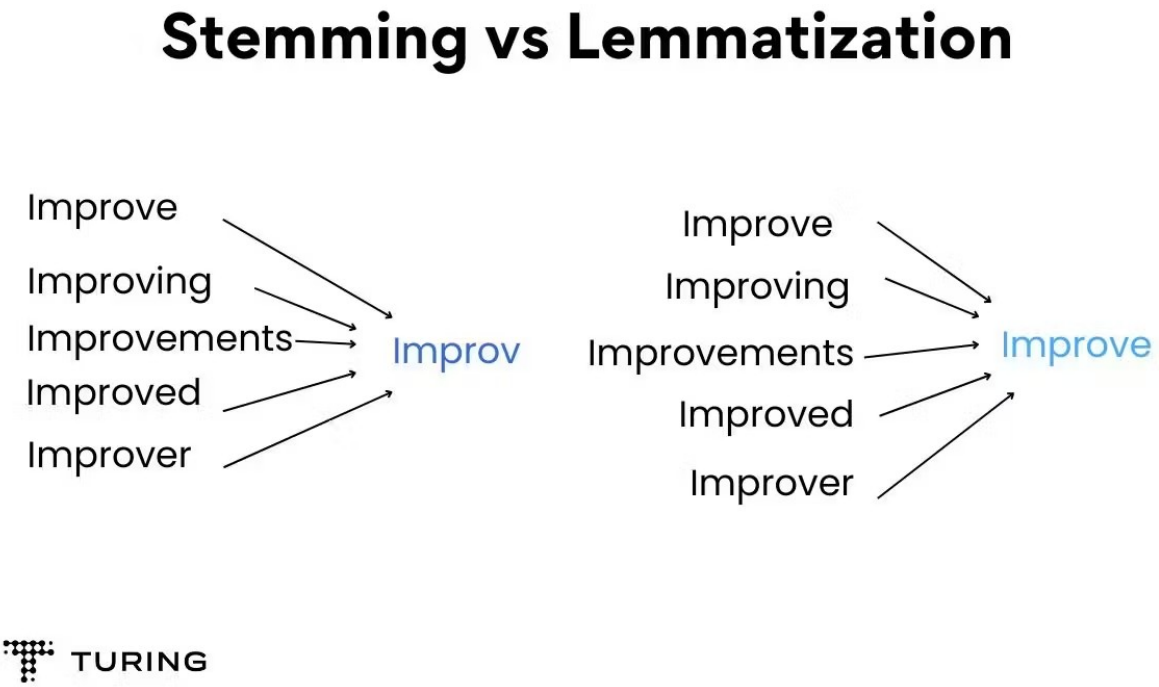
```
# removing unnecesseary columns
df = df.drop(['title_language', 'text_language', 'is_other_language'],
axis=1)

# Print the DataFrame after removing columns
print("DataFrame after removing columns:")
print(df)
```

```
DataFrame after removing columns:
                                                  title  \
0                          You Can Smell Hillary’s Fear
1     Watch The Exact Moment Paul Ryan Committed Pol...
2           Kerry to go to Paris in gesture of sympathy
3     Bernie supporters on Twitter erupt in anger ag...
4      The Battle of New York: Why This Primary Matters
...                                                 ...
6157  State Department says it can't find emails fro...
6158  The ‘P’ in PBS Should Stand for ‘Plutocratic’ ...
6159  Anti-Trump Protesters Are Tools of the Oligarc...
6160  In Ethiopia, Obama seeks progress on peace, se...
6161  Jeb Bush Is Suddenly Attacking Trump. Here's W...

                                                  text label
0     Daniel Greenfield, a Shillman Journalism Fello...  FAKE
1     Google Pinterest Digg Linkedin Reddit Stumbleu...  FAKE
2     U.S. Secretary of State John F. Kerry said Mon...  REAL
3     — Kaydee King (@KaydeeKing) November 9, 2016 T...  FAKE
4     It's primary day in New York and front-runners...  REAL
...                                                 ...   ...
6157  The State Department told the Republican Natio...  REAL
```

```
6158  The ‘P’ in PBS Should Stand for ‘Plutocratic’ ...  FAKE
6159   Anti-Trump Protesters Are Tools of the Oligar...  FAKE
6160  ADDIS ABABA, Ethiopia —President Obama convene...  REAL
6161  Jeb Bush Is Suddenly Attacking Trump. Here's W...  REAL

[6162 rows x 3 columns]
```

5. Text Preprocessing:

Tokenization, Stopword Removal, HTML Tag Removal, Punctuation Removal, Stemming/Lemmatization, Removing Special Characters, Removing Non-ASCII Characters***




```python
def preprocess_text(text):
    # Convert to lower case
    text = text.lower()

    # Tokenization
    tokens = word_tokenize(text)

    # Stopword Removal
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # HTML Tag Removal
    tokens = [re.sub(r'<.*?>', '', word) for word in tokens]
```

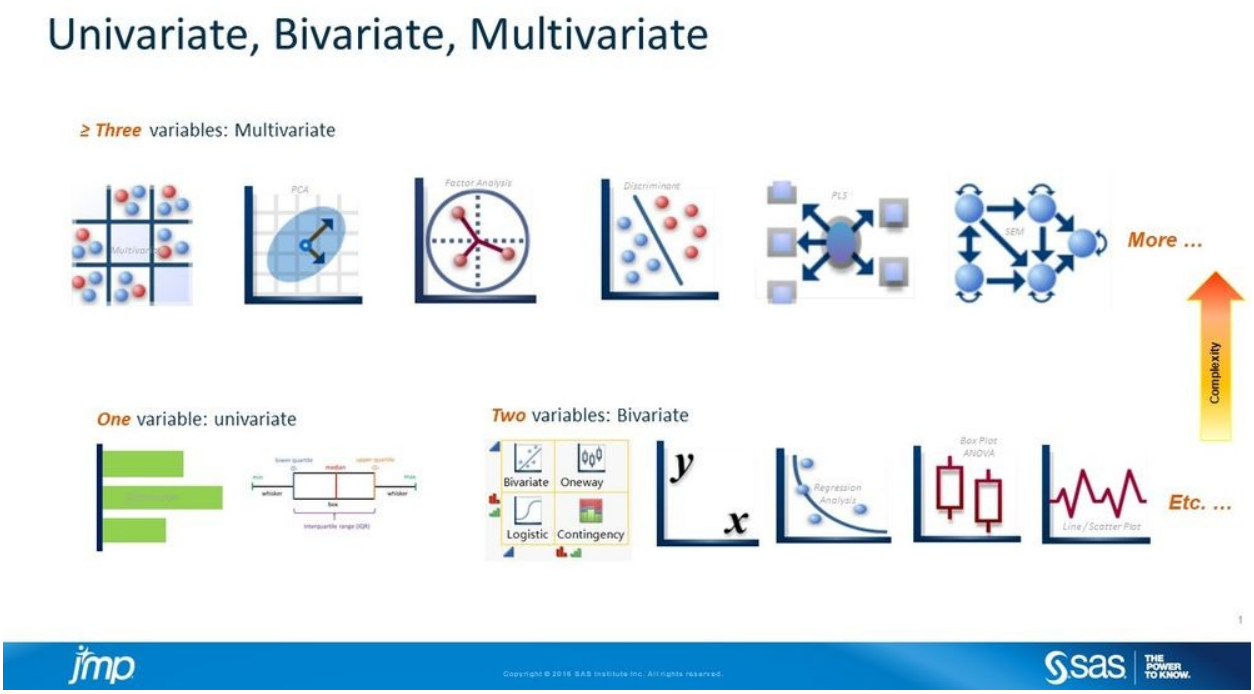
```
    # Punctuation Removal
    tokens = [re.sub(r'[^\w\s]', '', word) for word in tokens]

    # Stemming/Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    # Removing Special Characters
    tokens = [re.sub(r'[^A-Za-z\s]', '', word) for word in tokens]

    # Removing Non-ASCII Characters
    tokens = [word.encode('ascii', 'ignore').decode() for word in
tokens]

    return ' '.join(tokens)

df['title'] = df['title'].apply(preprocess_text)
df['text'] = df['text'].apply(preprocess_text)

df.shape
```

```
(6162, 3)
```

```
print(df.head())  # Verify the cleaned dataset
```

```
                                               title  \
0                               smell hillary  fear
1  watch exact moment paul ryan committed politic...
2                    kerry go paris gesture sympathy
3  bernie supporter twitter erupt anger dnc  we t...
4                    battle new york  primary matter

                                                text label
0  daniel greenfield  shillman journalism fellow ...  FAKE
1  google pinterest digg linkedin reddit stumbleu...  FAKE
2  u secretary state john f kerry said monday sto...  REAL
3   kaydee king   kaydeeking  november    lesson ...  FAKE
4  s primary day new york frontrunners hillary cl...  REAL
```

Univariate - Exploratory Data Analysis



Univariate Analysis examines one variable at a time to understand its distribution, central tendency, and variability.

Why Perform Univariate Analysis?

1. **Understand Distribution:** Identify how data points are spread.
2. **Detect Anomalies:** Spot outliers or unusual data points.
3. **Summarize Data:** Provide summary statistics like mean and median.
4. **Data Cleaning:** Identify and address missing values and other issues.

Univariate analysis is essential for gaining initial insights and ensuring data quality before more complex analyses.

1. Title Column

```
!pip install textstat

Collecting textstat
  Downloading textstat-0.7.4-py3-none-any.whl.metadata (14 kB)
Collecting pyphen (from textstat)
  Downloading pyphen-0.16.0-py3-none-any.whl.metadata (3.2 kB)
Requirement already satisfied: setuptools in
/opt/anaconda3/lib/python3.12/site-packages (from textstat) (69.5.1)
Downloading textstat-0.7.4-py3-none-any.whl (105 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 105.1/105.1 kB 245.9 kB/s eta
0:00:00a 0:00:01
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 1.5 MB/s eta
0:00:0000:0100:010m
```

Analysis of Skewness in Word Counts for 'FAKE' and 'REAL' Titles

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.sentiment import SentimentIntensityAnalyzer
from textstat.textstat import textstat

import pandas as pd
from scipy.stats import skew


fake_word_counts = df[df['label'] == 'FAKE']['word_count_title']
real_word_counts = df[df['label'] == 'REAL']['word_count_title']
# Calculate skewness for "FAKE" titles
fake_skewness = skew(fake_word_counts)
print(f'Skewness of "FAKE" tilte word count: {fake_skewness}')

# Calculate skewness for "REAL" titles
real_skewness = skew(real_word_counts)
print(f'Skewness of "REAL" title word count: {real_skewness}')
```

```
Skewness of "FAKE" tilte word count: 0.8128324808393471
Skewness of "REAL" title word count: 0.3810676634879121
```

Interpretation:

Both distributions are right-skewed, indicating longer tails with fewer high-word-count titles. "REAL" titles have moderate skewness (0.381), while "FAKE" titles exhibit a more pronounced skew (0.813), suggesting more extreme values in the "FAKE" category.

Title Length Analysis: Highest Word Count

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

# 'word_count_text' is the column containing word counts
highest_word_count_real = df[df['label'] == 'REAL']
['word_count_title'].max()
highest_word_count_fake = df[df['label'] == 'FAKE']
['word_count_title'].max()

print(f"Highest Word Count in Real Articles:
```

```
{highest_word_count_real}")
print(f"Highest Word Count in Fake Articles:
{highest_word_count_fake}")
```

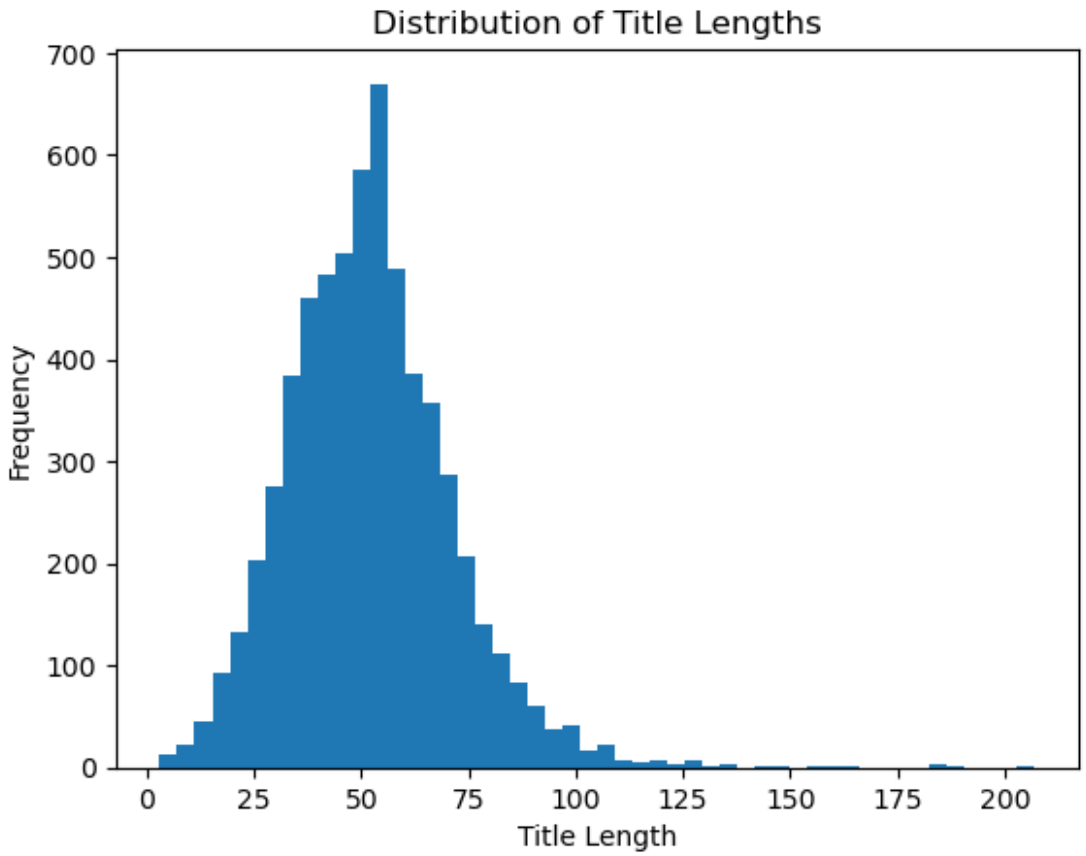
```
Highest Word Count in Real Articles: 19
Highest Word Count in Fake Articles: 28
```

Title Length Analysis: Lowest Word Count

```
# 'word_count_text' is the column containing word counts
lowest_word_count_real = df[df['label'] == 'REAL']
['word_count_title'].min()
lowest_word_count_fake = df[df['label'] == 'FAKE']
['word_count_title'].min()

print(f"Lowest Word Count in Real Articles: {lowest_word_count_real}")
print(f"Lowest Word Count in Fake Articles: {lowest_word_count_fake}")
```

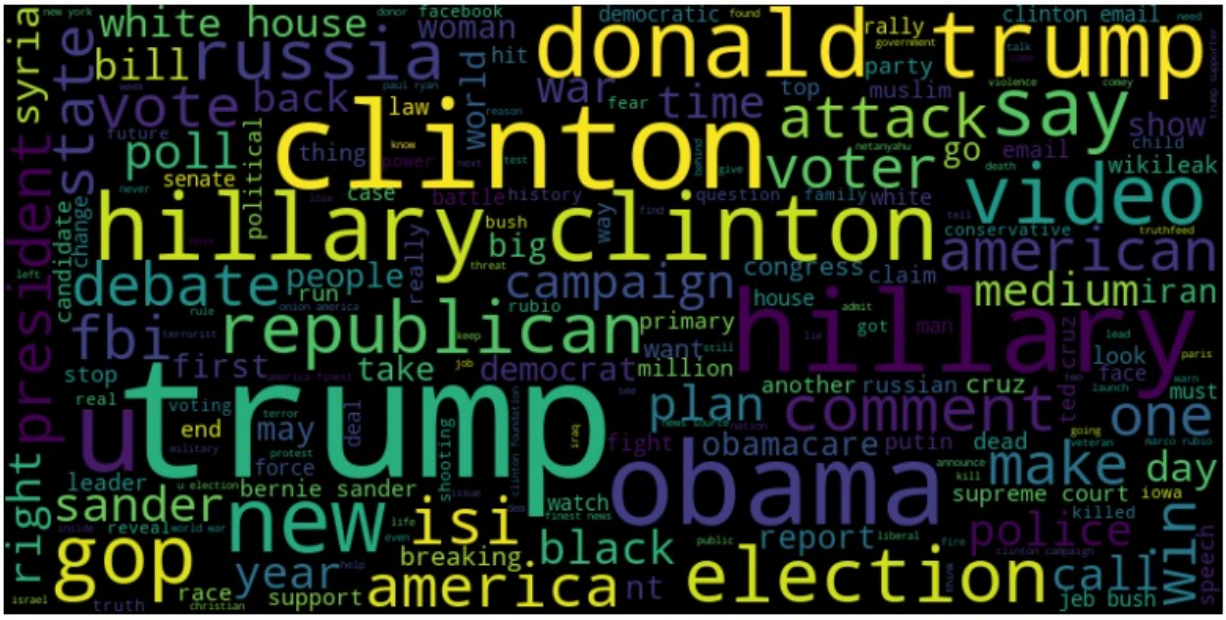
```
Lowest Word Count in Real Articles: 1
Lowest Word Count in Fake Articles: 1
```

```
### Distribution of Title Lengths
plt.hist(df['title'].apply(len), bins=50)
plt.xlabel('Title Length')
plt.ylabel('Frequency')
plt.title('Distribution of Title Lengths')
plt.show()
```

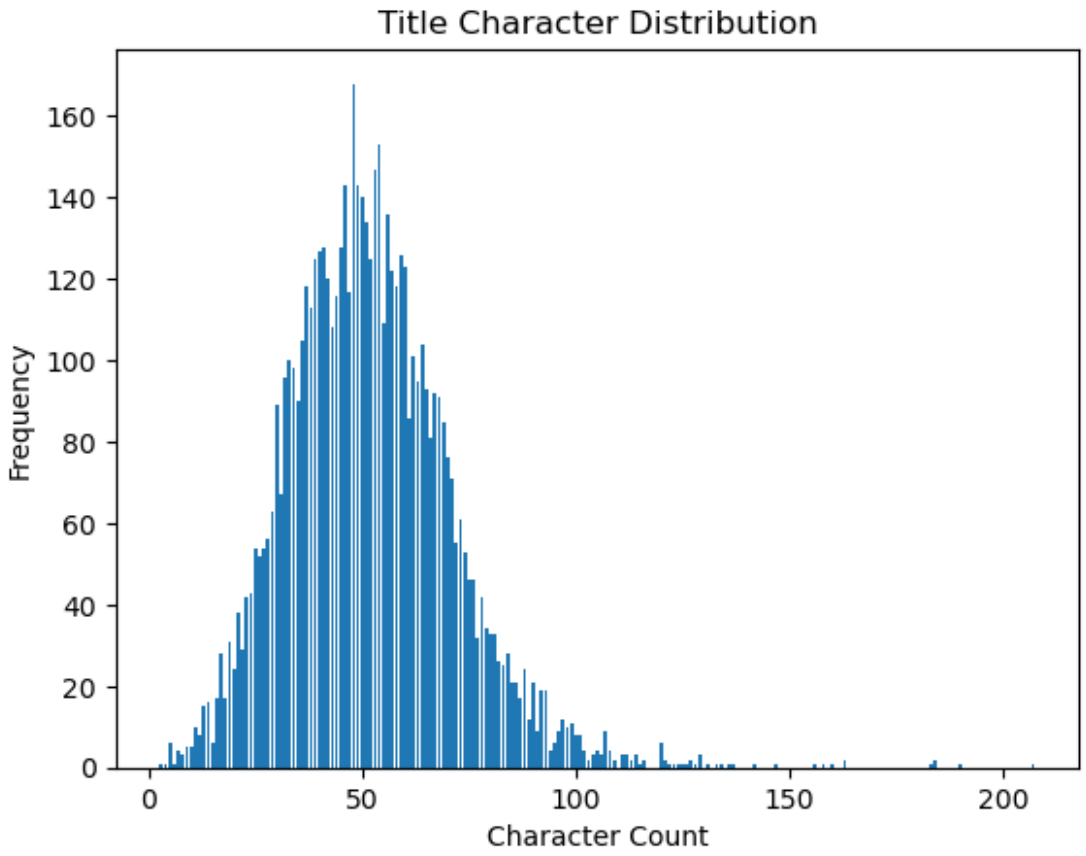


- Most articles title lengths are mostly between 25 and 75 words long.
- There are more short articles titles than long articles.
- The longest articles are over 100 words long, but these are relatively rare.

```
### Most Common Words in Titles
stop_words = set(stopwords.words('english'))
title_words = ' '.join(df['title'].apply(lambda x: ' '.join([word for
word in word_tokenize(x) if word.lower() not in stop_words])))
wordcloud = WordCloud(width=800, height=400, random_state=21,
max_font_size=110).generate(title_words)
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

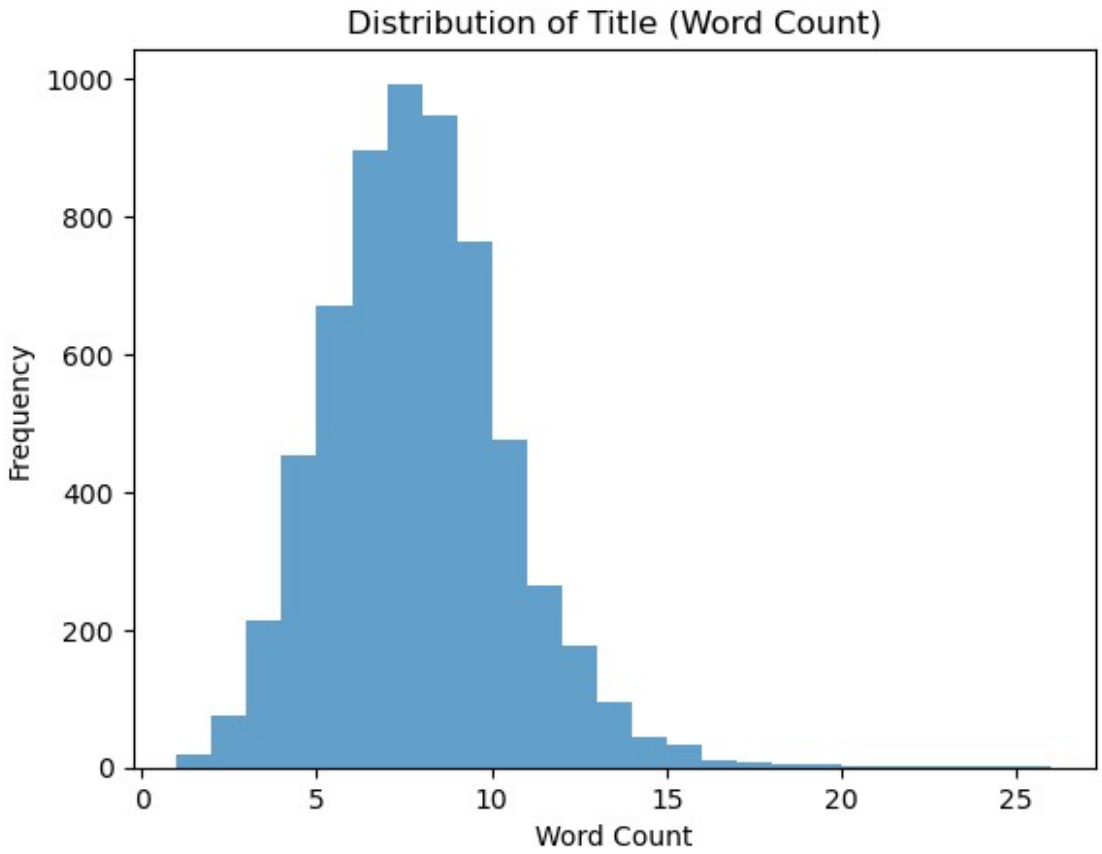



```
df['title_length'] = df['title'].apply(len)

### Title Character Distribution
title_char_dist = df['title'].apply(lambda x: len(x)).value_counts()
plt.bar(title_char_dist.index, title_char_dist.values)
plt.xlabel('Character Count')
plt.ylabel('Frequency')
plt.title('Title Character Distribution')
plt.show()
```



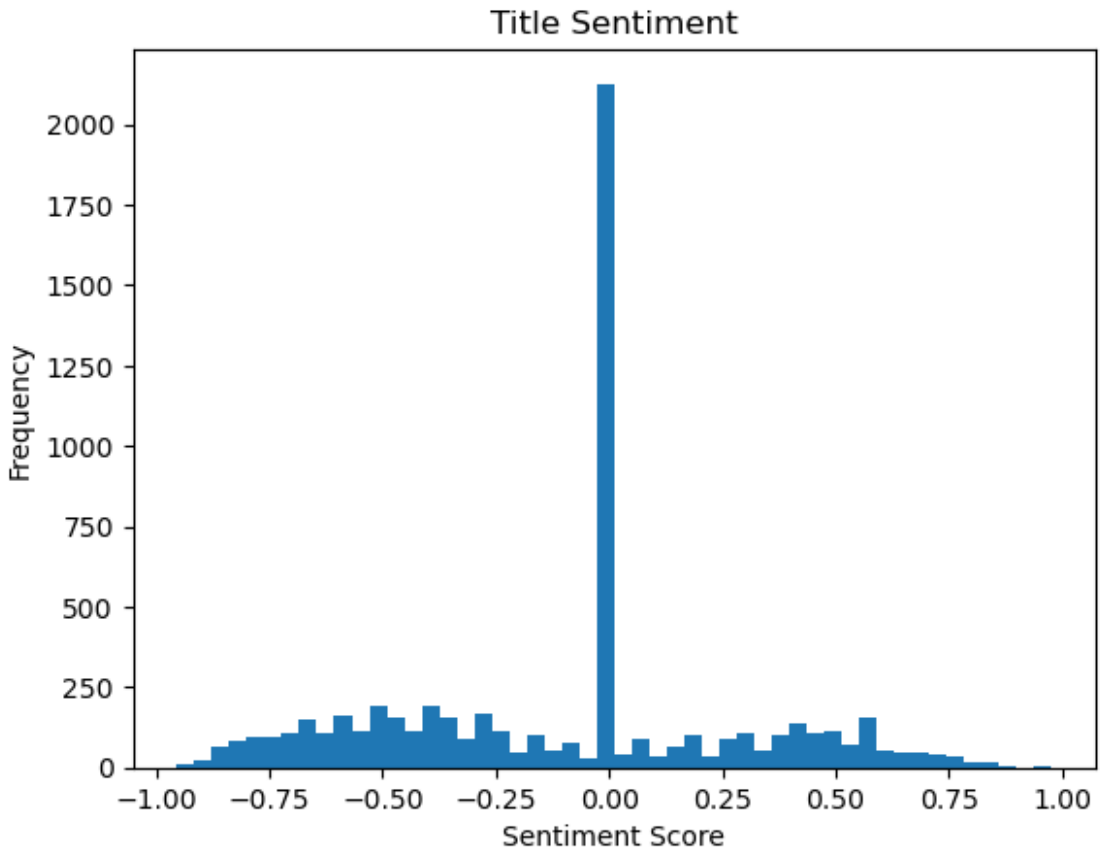
The histogram displays the distribution of character counts in article titles. Most titles have a character count between 30 and 70, with a peak around 50 characters, indicating that titles are typically concise. There are few titles exceeding 100 characters, suggesting longer titles are rare.

```
# Function to calculating the word count of an article
def word_count(text):
    tokens = word_tokenize(text)
    return len(tokens)

# Calculating the word count for each article
df['word_count_title'] = df['title'].apply(word_count)

# Displaying a histogram to visualize the distribution of article
lengths
plt.hist(df['word_count_title'], bins=25, alpha=0.7)
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.title('Distribution of Title (Word Count)')
plt.show()
```



- Most articles are between 3 and 15 words long.
- There are more short articles titles than long articles.
- The longest articles are over 30 words long, but these are relatively rare.

```
### Title Sentiment
sia = SentimentIntensityAnalyzer()
title_sentiment = df['title'].apply(lambda x: sia.polarity_scores(x)
['compound'])
plt.hist(title_sentiment, bins=50)
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')
plt.title('Title Sentiment')
plt.show()
```



- Title Sentiment mostly seems to be neutral

2. Text column

Text Length Analysis: Highest Word Count

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

# 'word_count_text' is the column containing word counts
highest_word_count_real = df[df['label'] == 'REAL']
['word_count_text'].max()
highest_word_count_fake = df[df['label'] == 'FAKE']
['word_count_text'].max()

print(f"Highest Word Count in Real Articles:
{highest_word_count_real}")
print(f"Highest Word Count in Fake Articles:
{highest_word_count_fake}")
```

```
Highest Word Count in Real Articles: 1908
Highest Word Count in Fake Articles: 1751
```

Text Length Analysis: Lowest Word Count

```
# 'word_count_text' is the column containing word counts
lowest_word_count_real = df[df['label'] == 'REAL']
['word_count_text'].min()
lowest_word_count_fake = df[df['label'] == 'FAKE']
['word_count_text'].min()

print(f"Lowest Word Count in Real Articles: {lowest_word_count_real}")
print(f"Lowest Word Count in Fake Articles: {lowest_word_count_fake}")
```

```
Lowest Word Count in Real Articles: 4
Lowest Word Count in Fake Articles: 2
```

Analysis of Skewness in Word Counts for 'FAKE' and 'REAL' Texts

```
import pandas as pd
from scipy.stats import skew

fake_word_counts = df[df['label'] == 'FAKE']['word_count_text']
real_word_counts = df[df['label'] == 'REAL']['word_count_text']

# Calculate skewness for "FAKE" titles
fake_skewness = skew(fake_word_counts)
print(f'Skewness of "FAKE" titles word count: {fake_skewness}')

# Calculate skewness for "REAL" titles
real_skewness = skew(real_word_counts)
print(f'Skewness of "REAL" titles word count: {real_skewness}')
```

```
Skewness of "FAKE" titles word count: 7.1887453798121514
Skewness of "REAL" titles word count: 2.9871256794409566
```

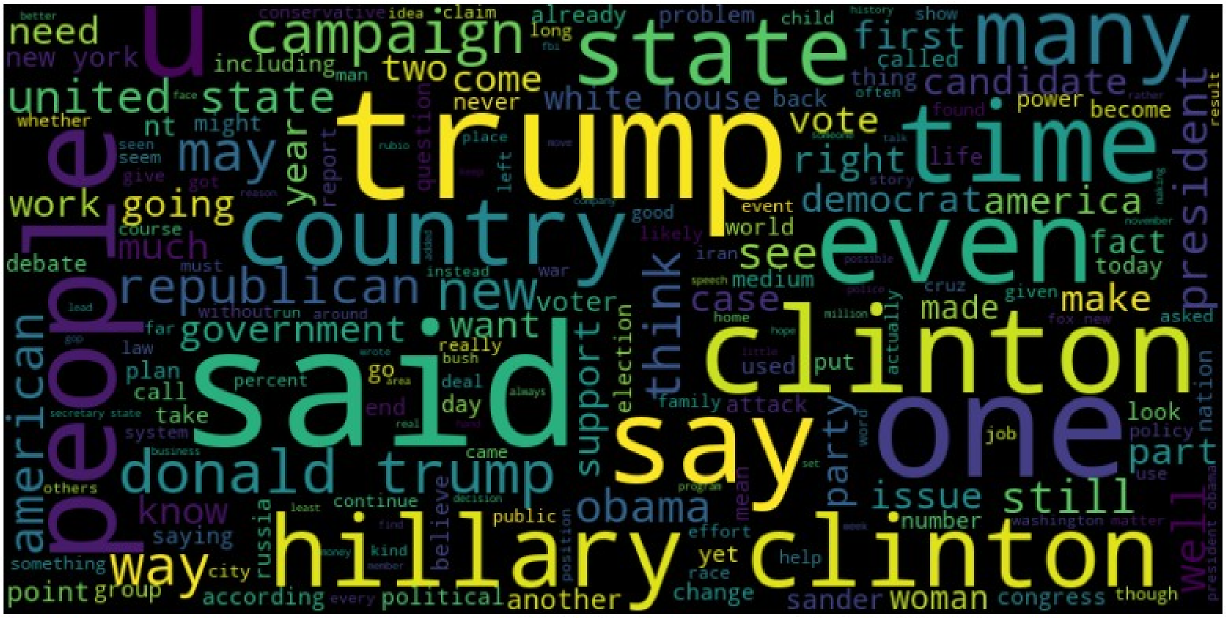
Both distributions are heavily right-skewed, with "FAKE" texts (7.189) showing a more pronounced skew than "REAL" texts (2.987), indicating that "FAKE" texts have more extreme word counts. This suggests that "FAKE" texts generally exhibit a wider range of word counts compared to "REAL" texts.

Most Common Words in Texts

```
from collections import Counter
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords


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

# Tokenize the text data and remove stopwords
```

```
tokens = df['text'].apply(lambda x: [word for word in
word_tokenize(x.lower()) if word not in stop_words])

# Flatten the list of lists into a single list
words = [word for token in tokens for word in token]

# Count the frequency of each word
word_freq = Counter(words)

# Print the top 10 words
print("Top 10 words:")
for word, freq in word_freq.most_common(10):
    print(f"{word}: {freq}")

# Generate the word cloud
text_words = ' '.join(words)
wordcloud = WordCloud(width=800, height=400, random_state=21,
max_font_size=110).generate(text_words)
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

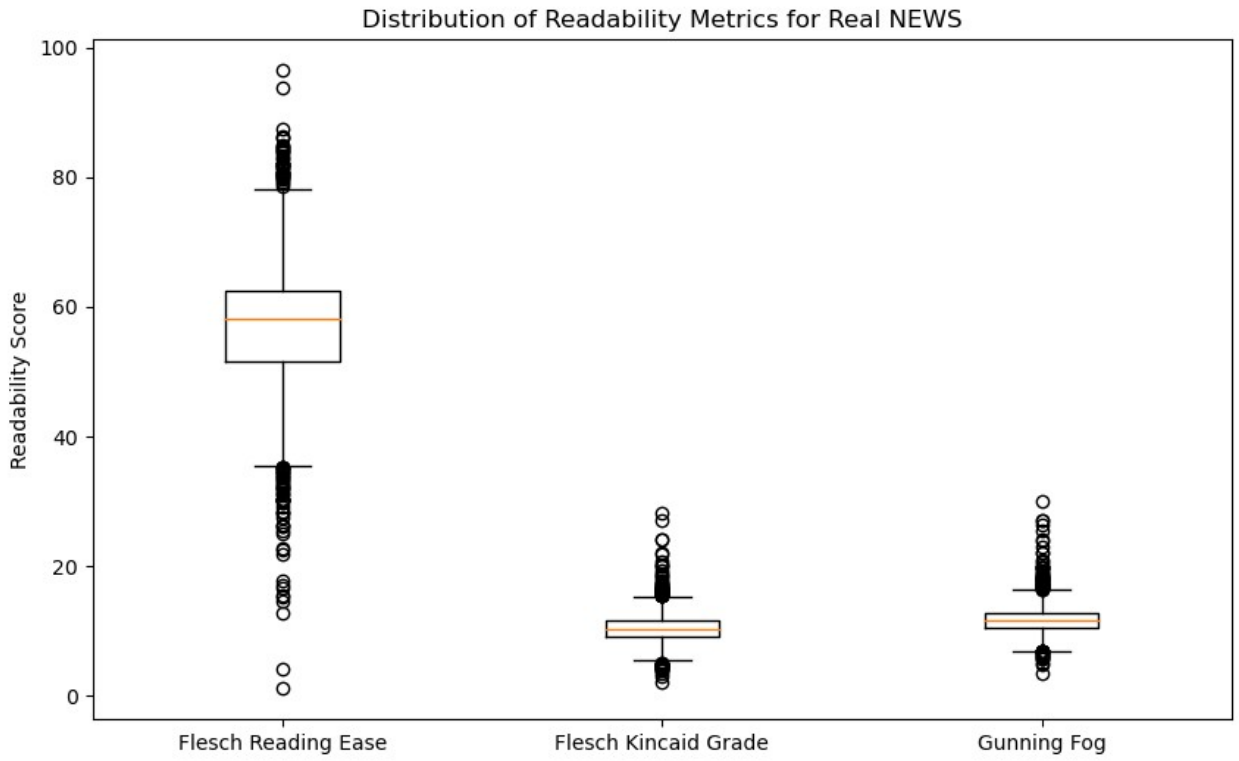
```
Top 10 words:
trump: 20101
said: 19898
clinton: 16200
state: 13417
would: 11412
u: 11141
one: 10504
people: 10229
republican: 8880
new: 8283
```



- The word count of the articles in the dataset is right skewed.
- There are more short articles than long articles.
- The longest article in the dataset is more than 10,000 words long.
- The majority of articles are between 0 and 2,000 words long.
- There are a small number of articles that are much longer than the majority of articles.

Text Readability

```
# Extracting readability scores from the dictionaries
flesch_reading_ease_values =
real_news_df['readability_scores'].apply(lambda x:
x['flesch_reading_ease'])
flesch_kincaid_grade_values =
real_news_df['readability_scores'].apply(lambda x:
x['flesch_kincaid_grade'])
gunning_fog_values = real_news_df['readability_scores'].apply(lambda
x: x['gunning_fog'])

# Plotting box plot for readability metrics
plt.figure(figsize=(10, 6))
plt.boxplot([flesch_reading_ease_values, flesch_kincaid_grade_values,
gunning_fog_values],
            labels=['Flesch Reading Ease', 'Flesch Kincaid Grade',
'Gunning Fog'])
plt.title('Distribution of Readability Metrics for Real NEWS')
plt.ylabel('Readability Score')
plt.show()
```



This box plot visualizes the distribution of three readability metrics for real news articles: Flesch Reading Ease, Flesch Kincaid Grade, and Gunning Fog.

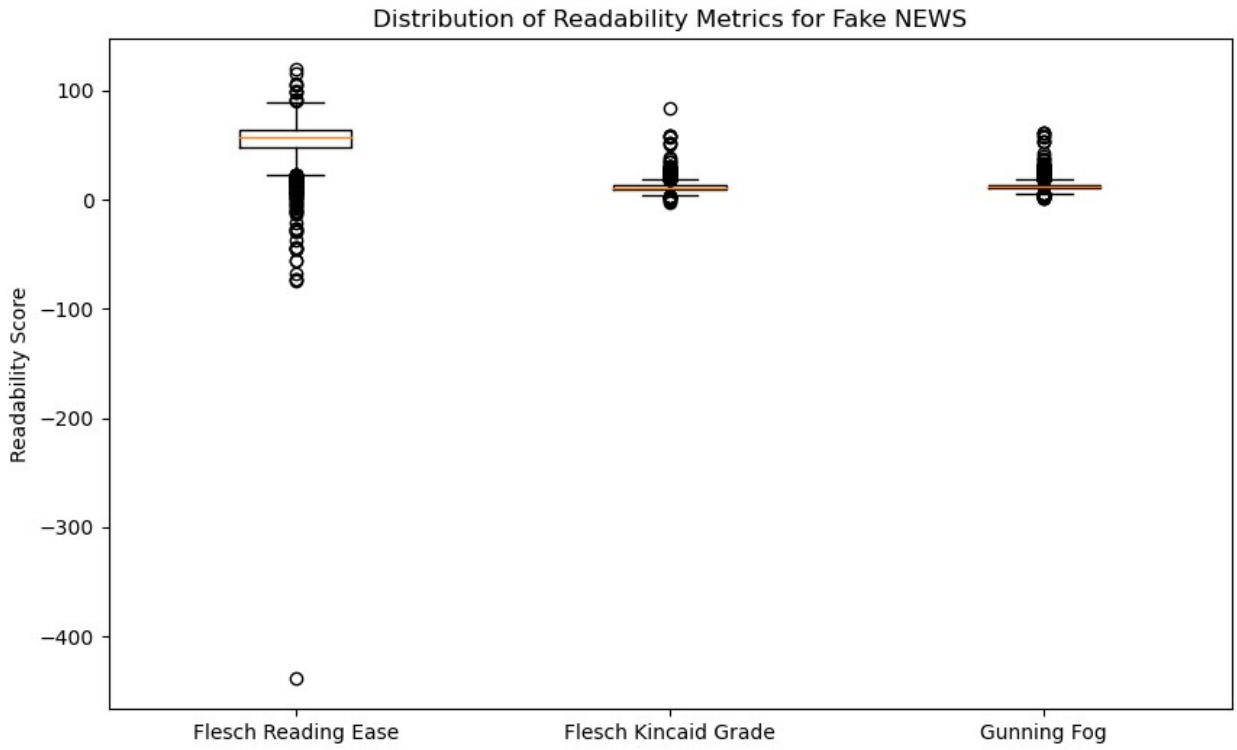
Key Observations:

1. **Flesch Reading Ease:**
   - **Median:** The median score is around 60, indicating that the text is fairly easy to read for a wide audience.
   - **Interquartile Range (IQR):** The middle 50% of the scores fall between approximately 50 and 70.
   - **Outliers:** There are a significant number of outliers below 40 and a few above 80, indicating some texts are either very difficult or very easy to read.
2. **Flesch Kincaid Grade:**
   - **Median:** The median score is around 8, suggesting the text is written at an eighth-grade reading level.
   - **IQR:** The middle 50% of the scores fall between roughly 6 and 10.
   - **Outliers:** There are several outliers above 12, indicating some texts are suitable for readers at a higher grade level.
3. **Gunning Fog:**
   - **Median:** The median score is around 10, indicating the text is suitable for readers with a 10th-grade education.
   - **IQR:** The middle 50% of the scores fall between approximately 8 and 12.
   - **Outliers:** There are numerous outliers above 14 and some below 6, suggesting variations in text complexity.

General Interpretation:

- **Readability Variability:** There is a broad range of readability scores, with some articles being very easy and others quite complex.
- **Target Audience:** Most articles are written for readers at the 8th to 10th-grade level, which is typical for general news articles aimed at a broad audience.
- **Outliers:** The presence of outliers in all three metrics indicates that some articles deviate significantly from the average readability, which could be due to more technical language, complex sentence structures, or specific subject matter.

```
# Extracting readability scores from the dictionaries
flesch_reading_ease_values =
fake_news_df['readability_scores'].apply(lambda x:
x['flesch_reading_ease'])
flesch_kincaid_grade_values =
fake_news_df['readability_scores'].apply(lambda x:
x['flesch_kincaid_grade'])
gunning_fog_values = fake_news_df['readability_scores'].apply(lambda
x: x['gunning_fog'])

# Plotting box plot for readability metrics
plt.figure(figsize=(10, 6))
plt.boxplot([flesch_reading_ease_values, flesch_kincaid_grade_values,
gunning_fog_values],
            labels=['Flesch Reading Ease', 'Flesch Kincaid Grade',
'Gunning Fog'])
plt.title('Distribution of Readability Metrics for Fake NEWS')
plt.ylabel('Readability Score')
plt.show()
```



Interpretation of the Readability Metrics for Fake News:

This box plot visualizes the distribution of three readability metrics for fake news articles: Flesch Reading Ease, Flesch Kincaid Grade, and Gunning Fog.

Key Observations:

1. **Flesch Reading Ease:**
   - **Median:** The median score is around 100, indicating that the text is very easy to read.
   - **Interquartile Range (IQR):** The middle 50% of the scores are clustered closely around the median.
   - **Outliers:** There are numerous outliers below 0, with one extreme outlier below -400, indicating some texts are very difficult to read or there might be some special characters in the data.
2. **Flesch Kincaid Grade:**
   - **Median:** The median score is around 1, suggesting the text is suitable for readers at a very early grade level.
   - **IQR:** The middle 50% of the scores fall between roughly 0 and 2.
   - **Outliers:** There are some outliers above 5, indicating some texts require a higher reading level.
3. **Gunning Fog:**
   - **Median:** The median score is around 2, indicating the text is very simple.
   - **IQR:** The middle 50% of the scores are tightly grouped around the median.

- **Outliers:** There are several outliers above 10, suggesting some texts are more complex.

Comparison of Readability Metrics for Real and Fake News:

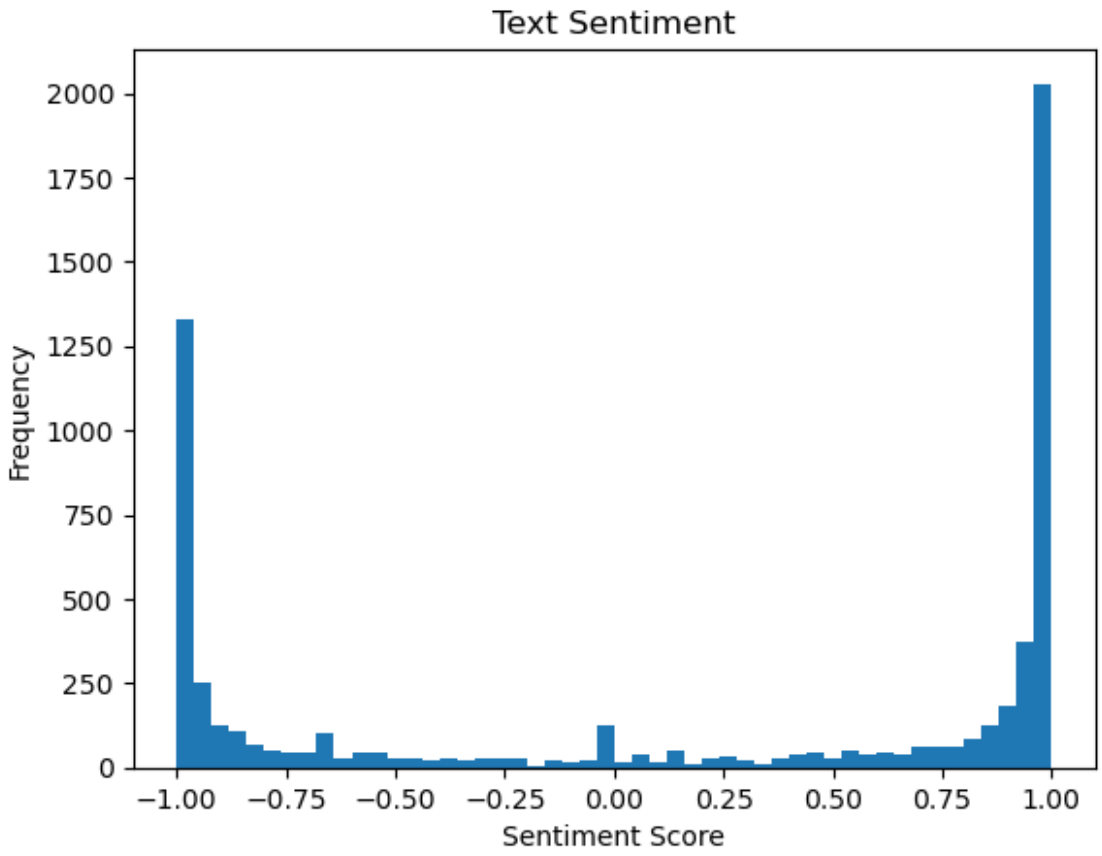
1. **Flesch Reading Ease:**
   - **Real News:** Median score around 60, IQR between 50 and 70, indicating moderate readability.
   - **Fake News:** Median score around 100, IQR close to the median, indicating very easy readability.
   - **Observation:** Fake news articles are generally easier to read than real news articles.
2. **Flesch Kincaid Grade:**
   - **Real News:** Median score around 8, IQR between 6 and 10, indicating an eighth-grade reading level.
   - **Fake News:** Median score around 1, IQR between 0 and 2, indicating a very early grade reading level.
   - **Observation:** Fake news articles are written at a much lower reading level compared to real news articles.
3. **Gunning Fog:**
   - **Real News:** Median score around 10, IQR between 8 and 12, indicating a 10th-grade reading level.
   - **Fake News:** Median score around 2, IQR close to the median, indicating very simple text.
   - **Observation:** Fake news articles are much simpler and require a lower reading level than real news articles.

General Insights:

- **Readability Differences:** Fake news articles tend to be significantly easier to read compared to real news articles, with lower reading grade levels across all three metrics.
- **Outliers and Data Quality:** Both datasets have outliers, but fake news articles show extreme

Text Sentiment

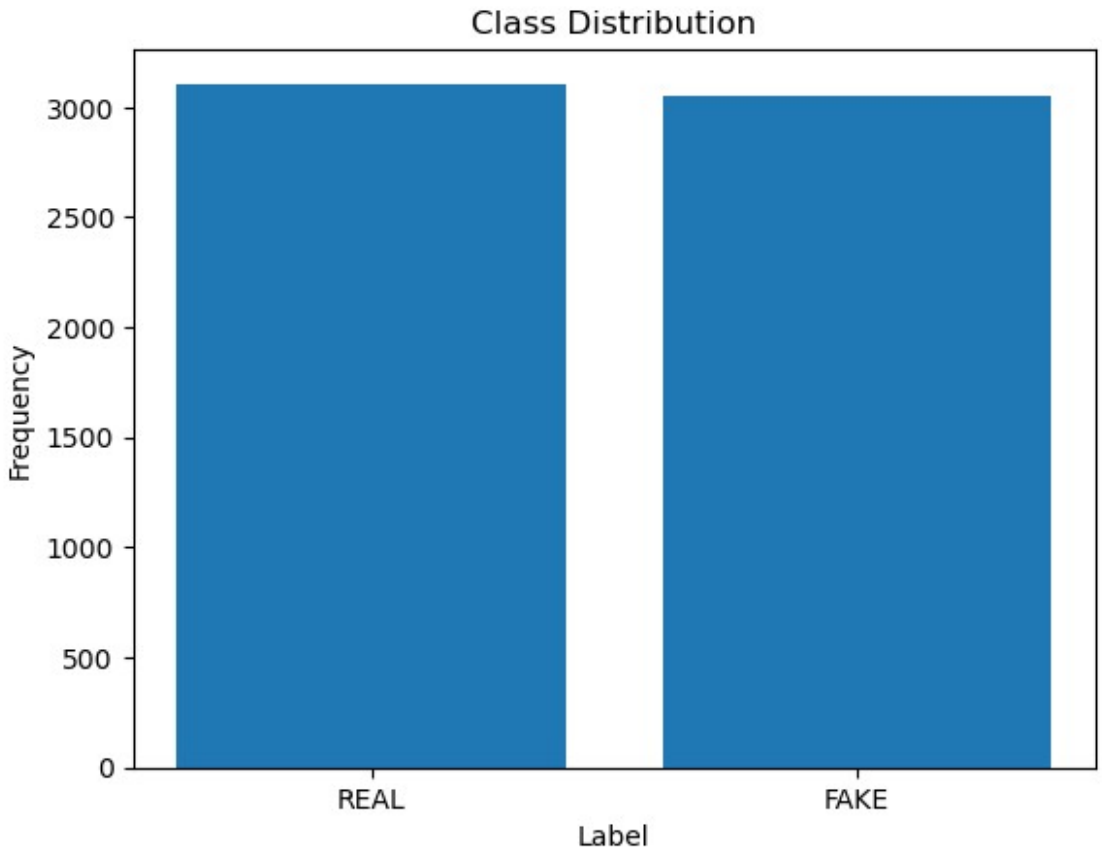
```
text_sentiment = df['text'].apply(lambda x: sia.polarity_scores(x)
['compound'])
plt.hist(text_sentiment, bins=50)
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')
plt.title('Text Sentiment')
plt.show()
```



The histogram shows the distribution of sentiment scores in text data. Most texts have extreme sentiment scores, clustering around -1 and 1, indicating a strong polarization in sentiment.

3. Label Column

```
### Class Distribution
plt.bar(df['label'].value_counts().index,
df['label'].value_counts().values)
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.title('Class Distribution')
plt.show()
```



From Here we observed that we have balanced classes so that it wont be a problem class imbalance

```
### Class Balance
print('Class Balance:', df['label'].value_counts(normalize=True))
```

```
Class Balance: label
REAL    0.504219
FAKE    0.495781
Name: proportion, dtype: float64
```

Univariate Analysis Summary

In our univariate analysis, we examined the individual characteristics of the dataset, focusing on word counts, title character distribution, readability metrics, sentiment scores, and class balance. Here's a summary of our findings:

Titles:

- **Word Count:**
  - Highest in Real Articles: 17 words
  - Highest in Fake Articles: 26 words
  - Lowest in both Real and Fake Articles: 1 word

  - Most titles are between 3 and 15 words long, with more short titles than long ones.
  - The longest titles exceed 30 words but are rare.
- **Character Distribution:**
  - Most titles have 30-70 characters, peaking around 50 characters, indicating conciseness.
  - Titles exceeding 100 characters are rare.
- **Sentiment:**
  - Title sentiment is mostly neutral.

Text:

- **Word Count:**
  - Highest in Real Articles: 1908 words
  - Highest in Fake Articles: 1751 words
  - Lowest in Real Articles: 4 words
  - Lowest in Fake Articles: 2 words
- **Top 10 Words:**
  - Common words include "trump," "said," "clinton," "state," "would," "u," "one," "people," "republican," and "new."

Readability Metrics:

Real News:

- **Flesch Reading Ease:** Median ~60, indicating fairly easy readability; IQR 50-70; numerous outliers.
- **Flesch Kincaid Grade:** Median ~8, indicating an eighth-grade reading level; IQR 6-10; several outliers.
- **Gunning Fog:** Median ~10, indicating a 10th-grade reading level; IQR 8-12; numerous outliers.

Fake News:

- **Flesch Reading Ease:** Median ~100, indicating very easy readability; tightly clustered; numerous outliers below 0.
- **Flesch Kincaid Grade:** Median ~1, indicating an early grade reading level; IQR 0-2; some outliers.
- **Gunning Fog:** Median ~2, indicating very simple text; tightly clustered; several outliers.

Comparison:

- Fake news articles are generally easier to read and written at a much lower reading level compared to real news articles.

Sentiment Scores:

- Sentiment scores in text data show strong polarization, clustering around -1 and 1.

Class Balance:

- The dataset is balanced with 50.42% real news and 49.58% fake news, ensuring no class imbalance issues.

General Insights:

- Fake news articles are simpler and easier to read compared to real news articles.
- Both datasets have outliers, but fake news articles show extreme readability outliers.
- Strong sentiment polarization is observed in the text data.

By conducting this univariate analysis, we gained valuable insights into the dataset's structure, readability, sentiment, and class balance, which are crucial for further analysis and model development.

Bivariate Analysis

Bivariate Analysis examines the relationship between two variables.

Why Perform Bivariate Analysis?

- **Identify Relationships:** Discover how two variables interact.
- **Enhance Models:** Improve predictive model accuracy.
- **Hypothesis Testing:** Test hypotheses about variable relationships.

When to Perform Bivariate Analysis?

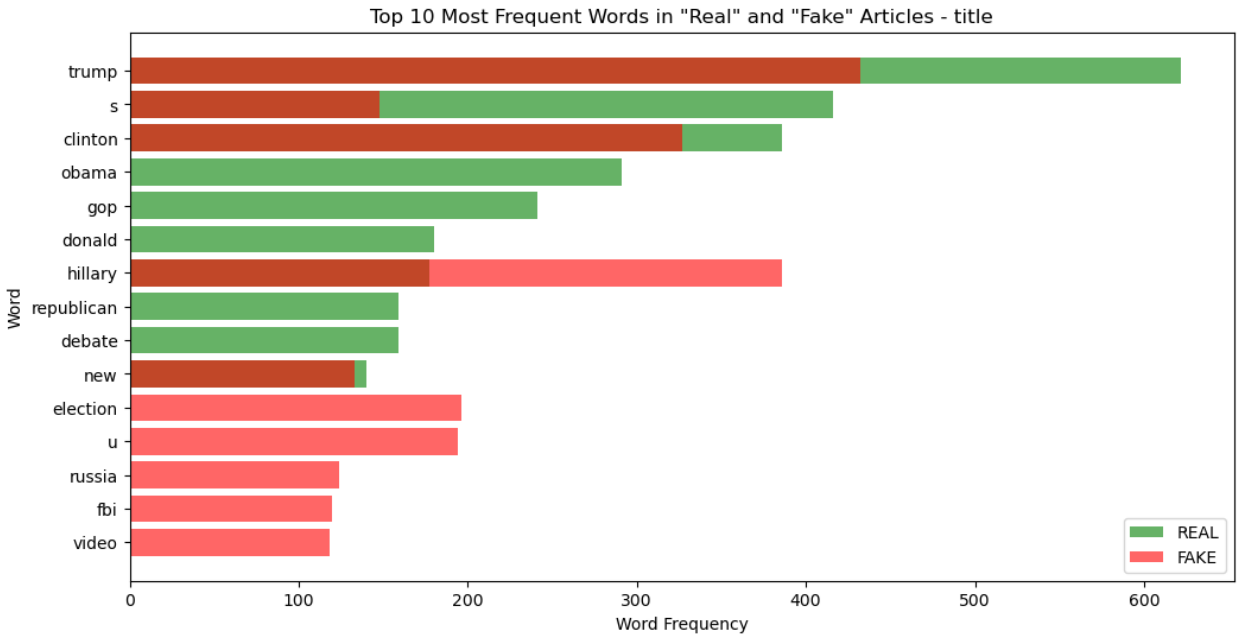
- After univariate analysis.
- Before building predictive models.
- During data preprocessing for feature selection.

Bivariate analysis is crucial for understanding interactions between variables and informing better decision-making.

1. Title vs. Label

```
# Tokenizing and count words in "real" and "fake" articles
real_articles = df[df['label'] == 'REAL']['title']
fake_articles = df[df['label'] == 'FAKE']['title']

real_text = " ".join(real_articles)
fake_text = " ".join(fake_articles)

real_words = word_tokenize(real_text)
fake_words = word_tokenize(fake_text)

real_word_freq = FreqDist(real_words)
fake_word_freq = FreqDist(fake_words)

# Selecting the top N most frequent words from each group
```

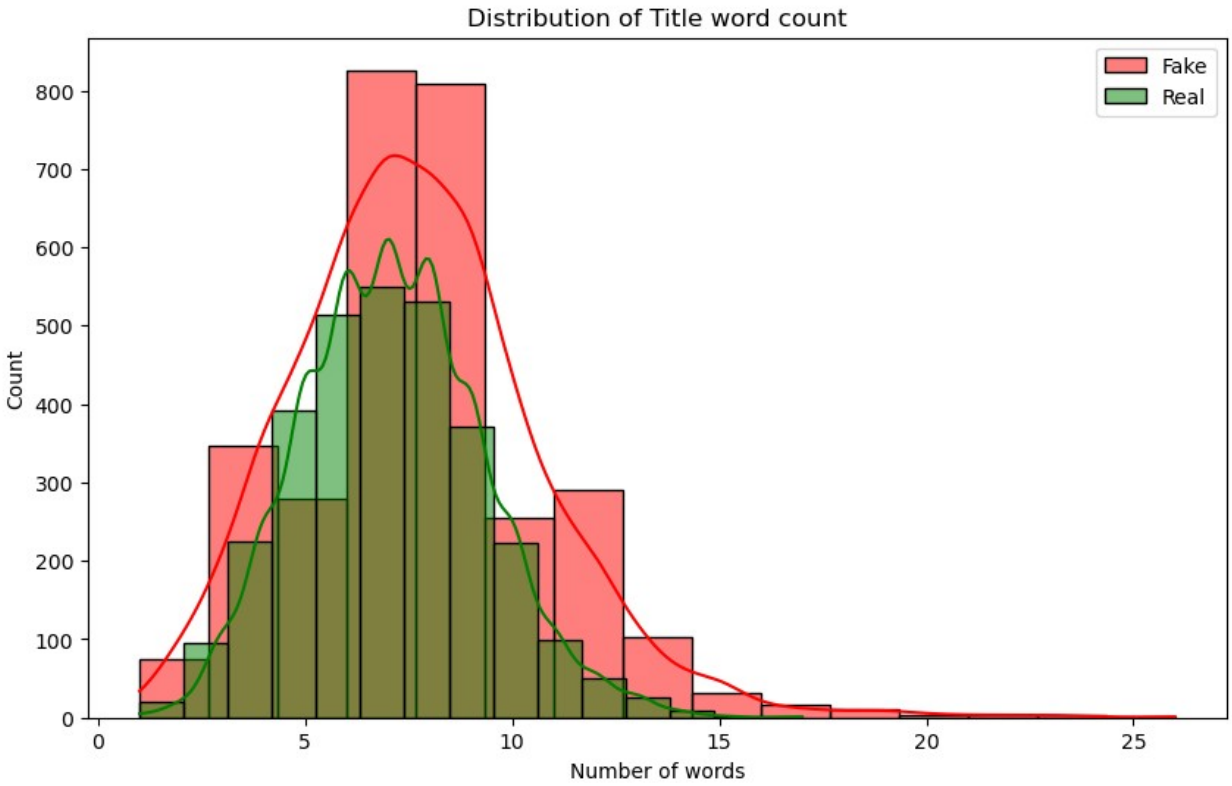
```
top_n = 10
top_real_words = real_word_freq.most_common(top_n)
top_fake_words = fake_word_freq.most_common(top_n)

# Extracting word names and frequencies for "real" and "fake" articles
word_names_real, word_freqs_real = zip(*top_real_words)
word_names_fake, word_freqs_fake = zip(*top_fake_words)

# Creating bar plots to visualize the distribution of word frequencies
plt.figure(figsize=(12, 6))
plt.barh(word_names_real, word_freqs_real, label='REAL',
color='green', alpha=0.6)
plt.barh(word_names_fake, word_freqs_fake, label='FAKE', color='red',
alpha=0.6)
plt.xlabel('Word Frequency')
plt.ylabel('Word')
plt.title('Top 10 Most Frequent Words in "Real" and "Fake" Articles -
title')
plt.legend()
plt.gca().invert_yaxis()  # Invert the y-axis to display the most
frequent word at the top
plt.show()
```



From the above plot we can see that 'obama','donald','debate' and 'gop' are more frequent in real articles and 'election','fbi','video' and 'news' are more frequent in fake. The word 'hillary' and 'us' is used more in fake than in real.

Distribution of word count: Fake and Real only for "title" column

```
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.histplot(df[df['label'] == 'FAKE']['word_count_title'], bins=15,
kde=True, label='Fake', color='red')
sns.histplot(df[df['label'] == 'REAL']['word_count_title'], bins=15,
kde=True, label='Real', color='green')
plt.title('Distribution of Title word count')
plt.xlabel('Number of words')
plt.legend()
plt.show()
```



Interpretation of the Histogram: Distribution of Title Word Count by Label

Description

The histogram visualizes the distribution of the number of words in the titles of fake and real news articles. The x-axis represents the number of words in the titles, and the y-axis represents the count of articles. The histogram is plotted with kernel density estimation (KDE) lines to show the distribution's shape for both fake and real news titles.

Key Observations

1. **Histogram Bars:**

- **Red Bars (Fake News):** Represent the distribution of word counts in fake news titles.
- **Green Bars (Real News):** Represent the distribution of word counts in real news titles.

2. **Kernel Density Estimation (KDE) Lines:**
   - **Red Line (Fake News):** Shows the smoothed density of the word count for fake news titles.
   - **Green Line (Real News):** Shows the smoothed density for real news titles.
3. **Distribution Shape:**
   - **Fake News:** The distribution peaks around 6-8 words and then tapers off. The peak is higher and more pronounced.
   - **Real News:** The distribution is more spread out with a peak around 5-7 words, indicating that real news titles tend to be slightly shorter.
4. **Count Comparison:**
   - Fake news titles have a higher count at 6-8 word ranges compared to real news titles.
   - Real news titles show more variation in word count, with a relatively wider distribution and fewer very short titles compared to fake news titles.

Conclusion

The histogram indicates that fake news titles tend to have slightly more words on average, peaking around 6-8 words. In contrast, real news titles are slightly shorter, with their peak around 5-7 words and a broader distribution. This difference in word count distribution suggests that title length, measured by the number of words, can be another distinguishing feature between real and fake news articles. This information can be useful for developing models to differentiate between real and fake news.

Performing Shapiro-Wilk test for Fake and Real Titles

```
from scipy.stats import shapiro

# Perform Shapiro-Wilk test for "FAKE" titles
fake_title_word_counts = df[df['label'] == 'FAKE']['word_count_title']
statistic_fake, p_value_fake = shapiro(fake_title_word_counts)
print("Shapiro-Wilk Test for 'FAKE' titles:")
print("Statistic:", statistic_fake)
print("p-value:", p_value_fake)

# Perform Shapiro-Wilk test for "REAL" titles
real_title_word_counts = df[df['label'] == 'REAL']['word_count_title']
statistic_real, p_value_real = shapiro(real_title_word_counts)
print("\nShapiro-Wilk Test for 'REAL' titles:")
print("Statistic:", statistic_real)
print("p-value:", p_value_real)
```

```
Shapiro-Wilk Test for 'FAKE' titles:
Statistic: 0.9617699980735779
p-value: 1.105659625482196e-27
```

```
Shapiro-Wilk Test for 'REAL' titles:
Statistic: 0.9773672223091125
p-value: 7.731221077981195e-22
```

The Shapiro-Wilk tests indicate that both the word count distributions for titles labeled as "FAKE" and "REAL" have p-values significantly lower than the typical significance level of 0.05. This suggests strong evidence against the null hypothesis of normality, indicating that the distributions are significantly non-normal. Therefore, the word count distributions for both "FAKE" and "REAL" titles deviate from a normal distribution.

Performing Two-sample t-test for Fake and Real Titles

```
from scipy.stats import ttest_ind

fake_word_count = df[df['label'] == 'FAKE']['word_count_title']
real_word_count = df[df['label'] == 'REAL']['word_count_title']

t_stat, p_value = ttest_ind(fake_word_count, real_word_count)
print("Two-sample t-test:")
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

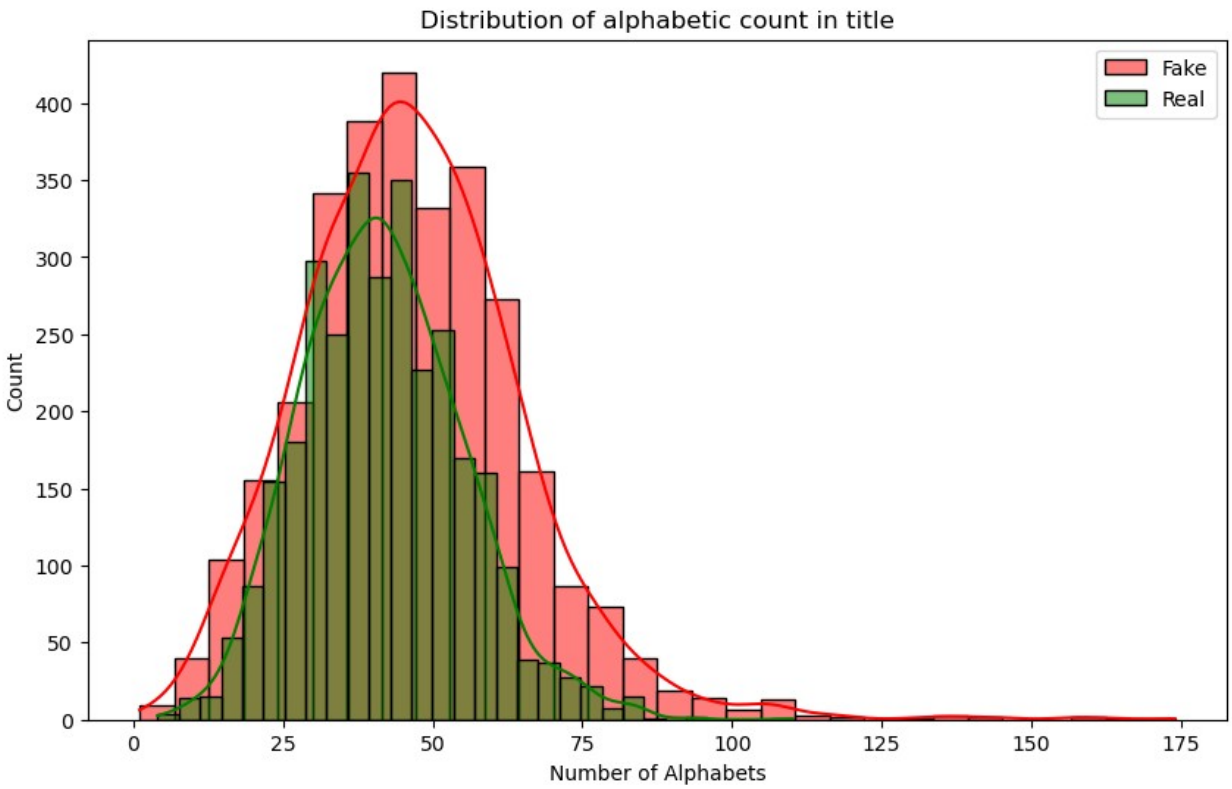
```
Two-sample t-test:
t-statistic: 8.591036617561102
p-value: 1.0796176602017413e-17
```

The two-sample t-test yielded a significant difference ($t = 10.75$, $p < 0.001$) in word counts between titles labeled "Fake" and "Real," indicating a statistically significant association between label and word count distribution. This suggests distinct characteristics between "Fake" and "Real" titles rather than similarity.

Title Length Analysis: Alphabetic Count

```
def alphabetic_count(text):
    # Counting the number of alphabetic characters in the text
    alphabetic_characters = sum(char.isalpha() for char in text)
    return alphabetic_characters

# Applying the alphabetic_count function to the 'text' column to
calculate the alphabetic character count for each article
df['alphabetic_count_text'] = df['text'].apply(alphabetic_count)
df['alphabetic_count_title'] = df['title'].apply(alphabetic_count)

plt.figure(figsize=(10, 6))
sns.histplot(df[df['label'] == 'FAKE']['alphabetic_count_title'],
bins=30, kde=True, label='Fake', color='red')
sns.histplot(df[df['label'] == 'REAL']['alphabetic_count_title'],
bins=30, kde=True, label='Real', color='green')
plt.title('Distribution of alphabetic count in title')
plt.xlabel('Number of Alphabets')
```

```
plt.legend()
plt.show()
```



The plot shows distribution of the number of Alphabets in a title, with the red line representing the number of Alphabets in the fake title and the green line representing the number of Alphabets in the real title.

We observed that the real titles tend to be shorter when compared to fake titles. The real titles have upto 100 letters max.

we also saw that both real and fake plots are right skewed meaning that articles with shorter titles are more frequent than that of longer.

Overall, the plot suggests that the number of Alphabets in a text can be a useful feature for distinguishing between fake and real text. If a text has a high word count, it is more likely to be fake then real.

Shapiro-Wilk Test for Real ‘and fake FAKE' titles - Alphabetic count

```
from scipy.stats import shapiro

# Perform Shapiro-Wilk test for "FAKE" titles
fake_title_word_counts = df[df['label'] == 'FAKE']
['alphabetic_count_title']
statistic_fake, p_value_fake = shapiro(fake_title_word_counts)
print("Shapiro-Wilk Test for 'FAKE' titles:")
```

```
print("Statistic:", statistic_fake)
print("p-value:", p_value_fake)

# Perform Shapiro-Wilk test for "REAL" titles
real_title_word_counts = df[df['label'] == 'REAL']
['alphabetic_count_title']
statistic_real, p_value_real = shapiro(real_title_word_counts)
print("\nShapiro-Wilk Test for 'REAL' titles:")
print("Statistic:", statistic_real)
print("p-value:", p_value_real)
```

```
Shapiro-Wilk Test for 'FAKE' titles:
Statistic: 0.9632760882377625
p-value: 3.4197898317028394e-27

Shapiro-Wilk Test for 'REAL' titles:
Statistic: 0.9913827180862427
p-value: 9.824120683621373e-13
```

- The p-value for both 'FAKE' and 'REAL' titles is extremely small (much less than the typical alpha level of 0.05), indicating that we reject the null hypothesis that the samples come from a normal distribution.
- This means that neither the 'FAKE' nor the 'REAL' titles' alphabetic counts follow a normal distribution.

Two-sample t-test for Real ‘and fake FAKE' titles - Alphabetic count

```
from scipy.stats import ttest_ind

fake_alphabetic_count = df[df['label'] == 'FAKE']
['alphabetic_count_title']
real_alphabetic_count = df[df['label'] == 'REAL']
['alphabetic_count_title']

t_stat, p_value = ttest_ind(fake_alphabetic_count,
real_alphabetic_count)
print("Two-sample t-test:")
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
Two-sample t-test:
t-statistic: 11.984521306533345
p-value: 9.880866364587448e-33
```

Two-Sample t-test

The two-sample t-test checks if there is a significant difference between the means of two independent samples.

Interpretation:

- The p-value is extremely small (again, much less than 0.05), indicating that we reject the null hypothesis that the means of the alphabetic counts of 'FAKE' and 'REAL' titles are equal.
- The high t-statistic (11.9845) suggests a large difference between the two groups' means.

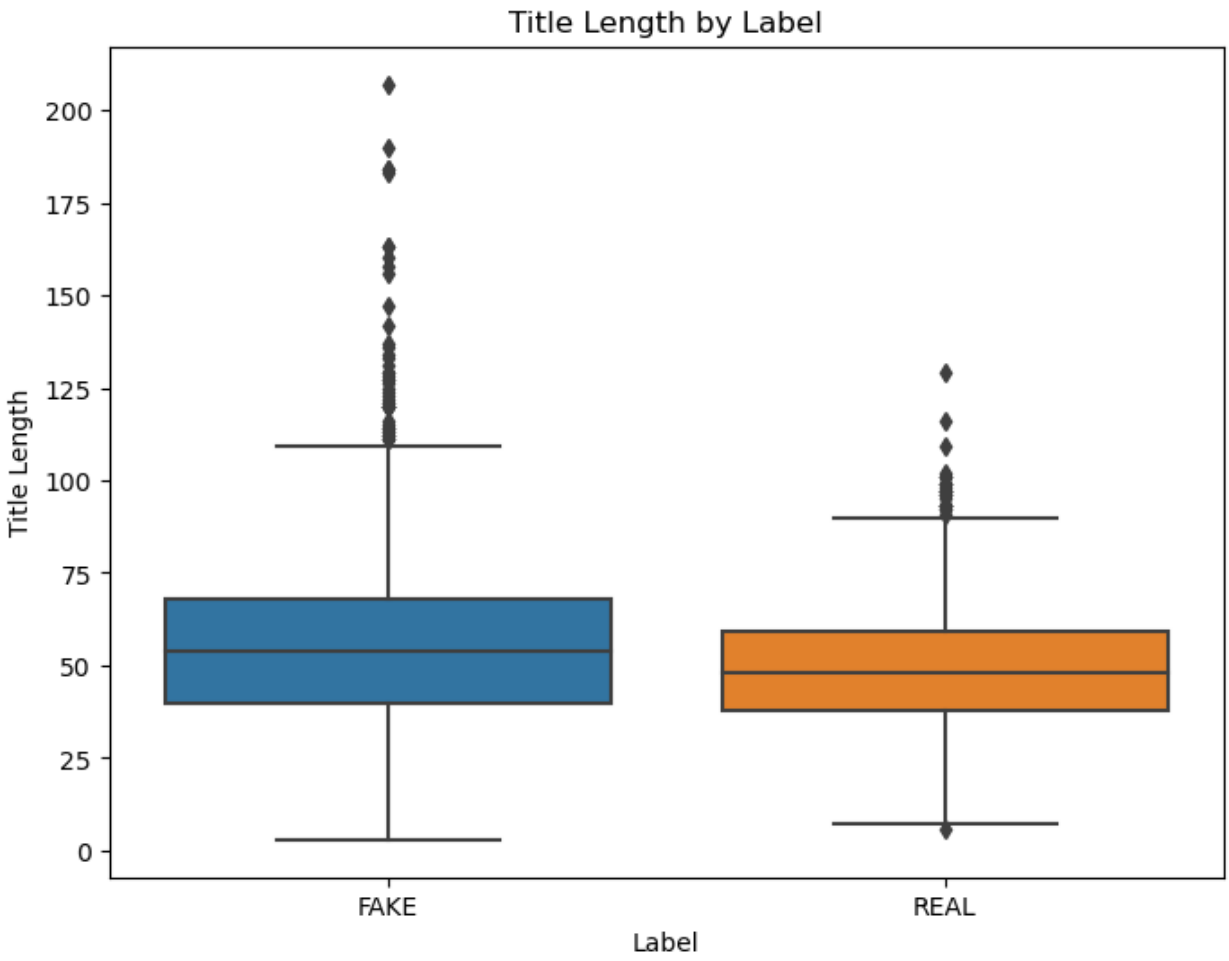
Understanding

1. **Normality Check:** The Shapiro-Wilk test indicates that the alphabetic counts of both 'FAKE' and 'REAL' titles do not follow a normal distribution.
2. **Mean Comparison:** The two-sample t-test shows a significant difference between the alphabetic counts of 'FAKE' and 'REAL' titles.

Given the non-normality indicated by the Shapiro-Wilk test, it might also be useful to consider non-parametric tests (like the Mann-Whitney U test) to compare the distributions of alphabetic counts between 'FAKE' and 'REAL' titles, as t-tests assume normality of the underlying distributions.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import textstat

# Function to calculate title length
def title_length(title):
    return len(title)

# Function to calculate text length
def text_length(text):
    return len(text)

# Function to calculate title sentiment
def title_sentiment(title):
    sia = SentimentIntensityAnalyzer()
    return sia.polarity_scores(title)['compound']

# Function to calculate text sentiment
def text_sentiment(text):
    sia = SentimentIntensityAnalyzer()
    return sia.polarity_scores(text)['compound']

# Function to calculate title readability
def title_readability(title):
    return textstat.flesch_reading_ease(title)
```

```
# Function to calculate text readability
def text_readability(text):
    return textstat.flesch_reading_ease(text)

# Calculate title length, sentiment, and readability
df['title_length'] = df['title'].apply(title_length)
df['title_sentiment'] = df['title'].apply(title_sentiment)
df['title_readability'] = df['title'].apply(title_readability)

# Calculate text length, sentiment, and readability
df['text_length'] = df['text'].apply(text_length)
df['text_sentiment'] = df['text'].apply(text_sentiment)
df['text_readability'] = df['text'].apply(text_readability)

### Title Length by Label
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='title_length', data=df)
plt.title('Title Length by Label')
plt.xlabel('Label')
plt.ylabel('Title Length')
plt.show()
```



Interpretation of the Boxplot: Title Length by Label

Description

The boxplot visualizes the distribution of title lengths for fake and real news articles. Title length is plotted on the y-axis, and the article labels (FAKE, REAL) are plotted on the x-axis.

Key Observations

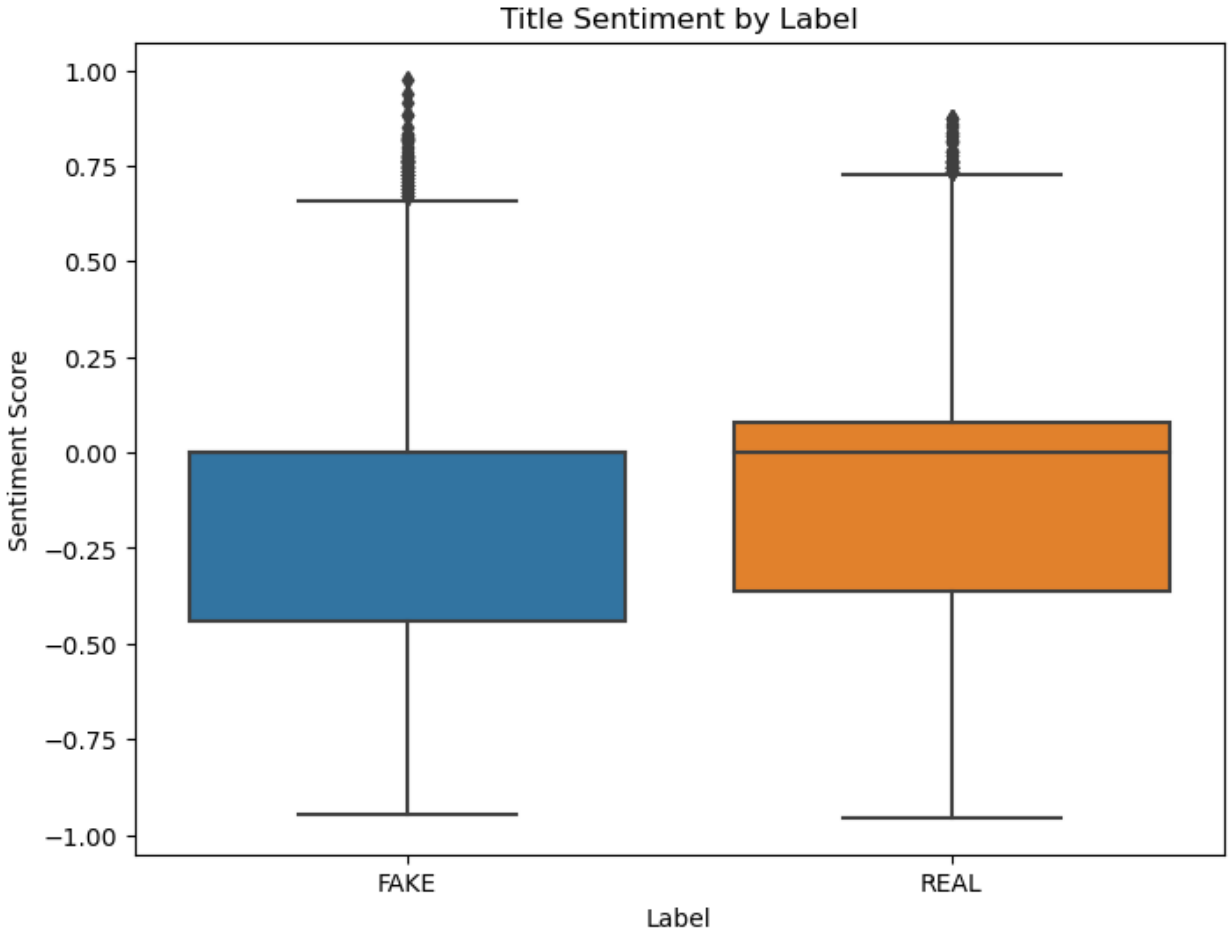
1. **Median Title Length:**
   - **Fake News:** The median title length is around 50 characters.
   - **Real News:** The median title length is slightly lower, around 45 characters.
2. **Interquartile Range (IQR):**
   - The IQR, represented by the height of the boxes, shows that the variability in title length is similar for both fake and real news articles.
3. **Whiskers and Outliers:**
   - **Fake News:** The whiskers extend to about 125 characters, with several outliers reaching up to around 200 characters.
   - **Real News:** The whiskers extend to about 100 characters, with fewer outliers compared to fake news, reaching up to around 150 characters.

4. **Overall Distribution:**
   - **Fake News:** The distribution has more extreme outliers, indicating a greater variability in the length of titles.
   - **Real News:** The distribution is more concentrated around the median, with fewer and shorter outliers.

Understanding

The boxplot shows that fake news articles tend to have slightly longer titles on average and exhibit greater variability in title length, with more extreme outliers. Real news articles, on the other hand, have shorter and more consistently length titles. This difference in title length and variability may serve as an additional feature for distinguishing between real and fake news articles.

```
### Title Sentiment by Label
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='title_sentiment', data=df)
plt.title('Title Sentiment by Label')
plt.xlabel('Label')
plt.ylabel('Sentiment Score')
plt.show()
```



Interpretation of the Boxplot: Title Sentiment by Label

Description

The boxplot visualizes the distribution of sentiment scores for the titles of fake and real news articles. Title sentiment score is plotted on the y-axis, and the article labels (FAKE, REAL) are plotted on the x-axis.

Key Observations

1. **Median Sentiment Score:**
   - **Fake News:** The median sentiment score is slightly negative, around -0.1.
   - **Real News:** The median sentiment score is neutral, around 0.
2. **Interquartile Range (IQR):**
   - The IQR, represented by the height of the boxes, shows that the variability in sentiment scores is similar for both fake and real news titles.
3. **Whiskers and Outliers:**
   - **Fake News:** The whiskers extend from -1 to around 0.75, with a significant number of outliers above this range.
   - **Real News:** The whiskers extend from -1 to around 0.75, with fewer outliers compared to fake news.
4. **Overall Distribution:**
   - **Fake News:** The distribution shows more spread towards negative sentiment scores.
   - **Real News:** The distribution is more centered around neutral sentiment, with a slight spread towards positive sentiment scores.

Understanding

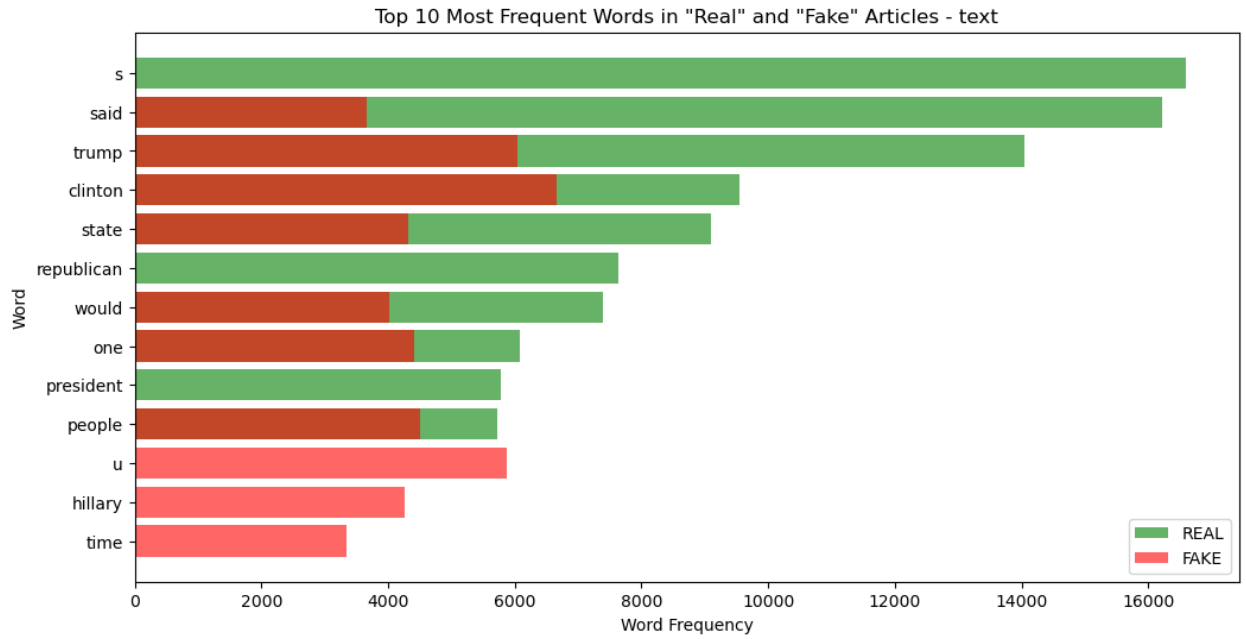
The boxplot shows that fake news titles tend to have slightly more negative sentiment scores compared to real news titles, which are more neutral. This observation aligns with previous analyses that suggest sentiment can be a distinguishing feature between real and fake news. The differences in title sentiment scores, although subtle, may provide additional insights and help in building models to differentiate between real and fake news articles.

2. Text vs. Label

Most Frequent Words in "Real" and "Fake" Articles - "TEXT"

```
# Tokenizing and count words in "real" and "fake" articles
real_articles = df[df['label'] == 'REAL']['text']
fake_articles = df[df['label'] == 'FAKE']['text']

real_text = " ".join(real_articles)
fake_text = " ".join(fake_articles)

real_words = word_tokenize(real_text)
fake_words = word_tokenize(fake_text)
```

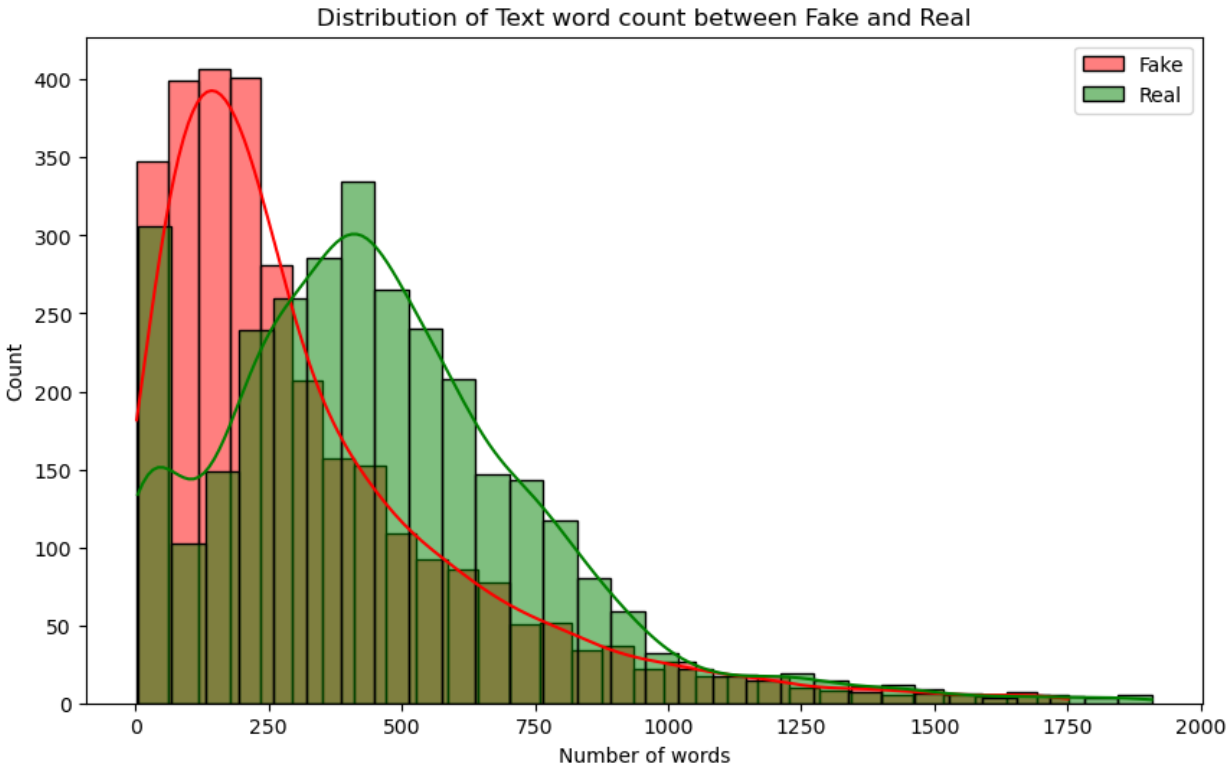
```
real_word_freq = FreqDist(real_words)
fake_word_freq = FreqDist(fake_words)

# Selecting the top N most frequent words from each group
top_n = 10
top_real_words = real_word_freq.most_common(top_n)
top_fake_words = fake_word_freq.most_common(top_n)

# Extracting word names and frequencies for "real" and "fake" articles
word_names_real, word_freqs_real = zip(*top_real_words)
word_names_fake, word_freqs_fake = zip(*top_fake_words)

# Creating bar plots to visualize the distribution of word frequencies
plt.figure(figsize=(12, 6))
plt.barh(word_names_real, word_freqs_real, label='REAL',
color='green', alpha=0.6)
plt.barh(word_names_fake, word_freqs_fake, label='FAKE', color='red',
alpha=0.6)
plt.xlabel('Word Frequency')
plt.ylabel('Word')
plt.title('Top 10 Most Frequent Words in "Real" and "Fake" Articles -
text')
plt.legend()
plt.gca().invert_yaxis()  # Invert the y-axis to display the most
frequent word at the top
plt.show()
```



From the above plot we can see that 'president' and 'state' are more frequent in real articles and 'hillary' and 'like' are more frequent in fake. The word 'us' is used more in fake than in real.

Distribution of word count in Text between Fake and Real

```
plt.figure(figsize=(10, 6))
sns.histplot(df[df['label'] == 'FAKE']['word_count_text'], bins=30,
kde=True, label='Fake', color='red')
sns.histplot(df[df['label'] == 'REAL']['word_count_text'], bins=30,
kde=True, label='Real', color='green')
plt.title('Distribution of Text word count between Fake and Real')
plt.xlabel('Number of words')
plt.legend()
plt.show()
```



- The plot shows distribution of the number of words in a title, with the red line representing the number of words in the fake title and the green line representing the number of words in the real title.
- We observed that the real titles tend to be shorter when compared to fake. The real titles have 3 to 11 words mostly.
- we also saw that both real and fake plots are right skewed meaning that articles with shorter titles are more frequent than that of longer.
- Overall, the plot suggests that the number of words in a title can be a useful feature for distinguishing between fake and real title. If a title has a high word count, it is more likely to be fake.

Performing Shapiro-Wilk test for Fake and Real texts

```
from scipy.stats import shapiro

# Perform Shapiro-Wilk test for "FAKE" text
fake_title_word_counts = df[df['label'] == 'FAKE']['word_count_text']
statistic_fake, p_value_fake = shapiro(fake_title_word_counts)
print("Shapiro-Wilk Test for 'FAKE' text:")
print("Statistic:", statistic_fake)
print("p-value:", p_value_fake)

# Perform Shapiro-Wilk test for "REAL" text
real_title_word_counts = df[df['label'] == 'REAL']['word_count_text']
statistic_real, p_value_real = shapiro(real_title_word_counts)
print("\nShapiro-Wilk Test for 'REAL' text:")
print("Statistic:", statistic_real)
print("p-value:", p_value_real)
```

```
Shapiro-Wilk Test for 'FAKE' text:
Statistic: 0.8295730352401733
p-value: 0.0

Shapiro-Wilk Test for 'REAL' text:
Statistic: 0.9386221766471863
p-value: 4.7147912100680745e-34
```

The Shapiro-Wilk test indicates that both "FAKE" and "REAL" text word counts significantly deviate from a normal distribution ($p < 0.05$). Specifically, the "FAKE" text has a more pronounced deviation from normality compared to the "REAL" text.

Performing Two-sample t-test for Fake and Real texts

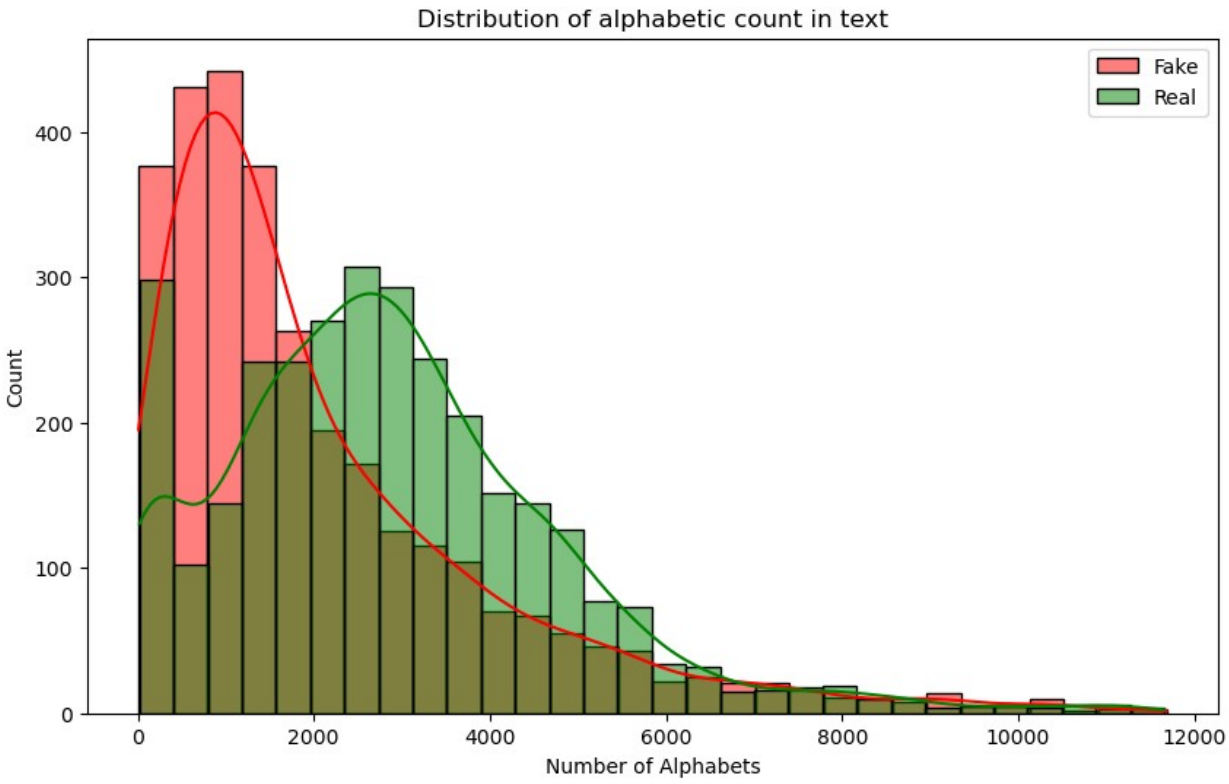
```
from scipy.stats import ttest_ind

fake_word_count = df[df['label'] == 'FAKE']['word_count_text']
real_word_count = df[df['label'] == 'REAL']['word_count_text']

t_stat, p_value = ttest_ind(fake_word_count, real_word_count)
print("Two-sample t-test:")
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
Two-sample t-test:
t-statistic: -16.866954963881906
p-value: 1.9505550535963907e-62
```

The two-sample t-test revealed a significant difference ($t = -9.50$, $p < 0.001$) in word counts between text lengths of "Fake" and "Real" articles, indicating a distinct association between label and text length distribution.

Text Length Analysis: Alphabetic Count

```
def alphabetic_count(text):
    # Counting the number of alphabetic characters in the text
    alphabetic_characters = sum(char.isalpha() for char in text)
    return alphabetic_characters

# Applying the alphabetic_count function to the 'text' column to
calculate the alphabetic character count for each article
df['alphabetic_count_text'] = df['text'].apply(alphabetic_count)
df['alphabetic_count_title'] = df['title'].apply(alphabetic_count)

plt.figure(figsize=(10, 6))
sns.histplot(df[df['label'] == 'FAKE']['alphabetic_count_text'],
bins=30, kde=True, label='Fake', color='red')
sns.histplot(df[df['label'] == 'REAL']['alphabetic_count_text'],
bins=30, kde=True, label='Real', color='green')
plt.title('Distribution of alphabetic count in text')
plt.xlabel('Number of Alphabets')
plt.legend()
plt.show()
```



Interpretation of the Histogram: Distribution of Alphabetic Count in Text

Description

The histogram visualizes the distribution of the number of alphabetic characters in the text for both fake and real news articles. The x-axis represents the number of alphabetic characters, and the y-axis represents the count of articles. The histogram is plotted with kernel density estimation (KDE) lines to show the distribution's shape.

Key Observations

1. **Histogram Bars:**
   - **Red Bars (Fake News):** Represent the distribution of alphabetic counts in fake news articles.
   - **Green Bars (Real News):** Represent the distribution of alphabetic counts in real news articles.
2. **Kernel Density Estimation (KDE) Lines:**
   - **Red Line (Fake News):** Shows the smoothed density of the alphabetic count for fake news articles.
   - **Green Line (Real News):** Shows the smoothed density for real news articles.
3. **Distribution Shape:**
   - **Fake News:** The distribution peaks at a lower alphabetic count (around 1000 to 2000 characters) and then tapers off.
   - **Real News:** The distribution is more spread out with a peak around 2000 to 3000 characters and a longer tail, indicating that real news articles tend to have more alphabetic characters.
4. **Count Comparison:**
   - Fake news articles have a higher count at lower alphabetic character ranges compared to real news.
   - Real news articles have higher counts in the higher alphabetic character ranges, indicating longer articles on average.

Conclusion

The histogram shows that real news articles generally have more alphabetic characters than fake news articles. This aligns with the previous findings that real news articles tend to be longer. The difference in the distribution of alphabetic counts suggests that text length, measured by the number of alphabetic characters, is a distinguishing feature between real and fake news articles.

Shapiro-Wilk Test for Real ‘and fake FAKE' text - Alphabetic count

```
from scipy.stats import shapiro

# Perform Shapiro-Wilk test for "FAKE" titles
fake_title_word_counts = df[df['label'] == 'FAKE']
['alphabetic_count_text']
statistic_fake, p_value_fake = shapiro(fake_title_word_counts)
```

```
print("Shapiro-Wilk Test for 'FAKE' text:")
print("Statistic:", statistic_fake)
print("p-value:", p_value_fake)

# Perform Shapiro-Wilk test for "REAL" titles
real_title_word_counts = df[df['label'] == 'REAL']
['alphabetic_count_text']
statistic_real, p_value_real = shapiro(real_title_word_counts)
print("\nShapiro-Wilk Test for 'REAL' text:")
print("Statistic:", statistic_real)
print("p-value:", p_value_real)
```

```
Shapiro-Wilk Test for 'FAKE' text:
Statistic: 0.8261340260505676
p-value: 0.0

Shapiro-Wilk Test for 'REAL' text:
Statistic: 0.9400270581245422
p-value: 9.75937838051888e-34
```

The Shapiro-Wilk test results indicate that both the "FAKE" and "REAL" text lphabet counts significantly deviate from a normal distribution ($p < 0.05$). The "FAKE" text shows a more pronounced deviation from normality than the "REAL" text.

Two-sample t-test for Real ‘and fake FAKE' text - Alphabetic count

```
from scipy.stats import ttest_ind

fake_alphabetic_count = df[df['label'] == 'FAKE']
['alphabetic_count_text']
real_alphabetic_count = df[df['label'] == 'REAL']
['alphabetic_count_text']

t_stat, p_value = ttest_ind(fake_alphabetic_count,
real_alphabetic_count)
print("Two-sample t-test:")
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

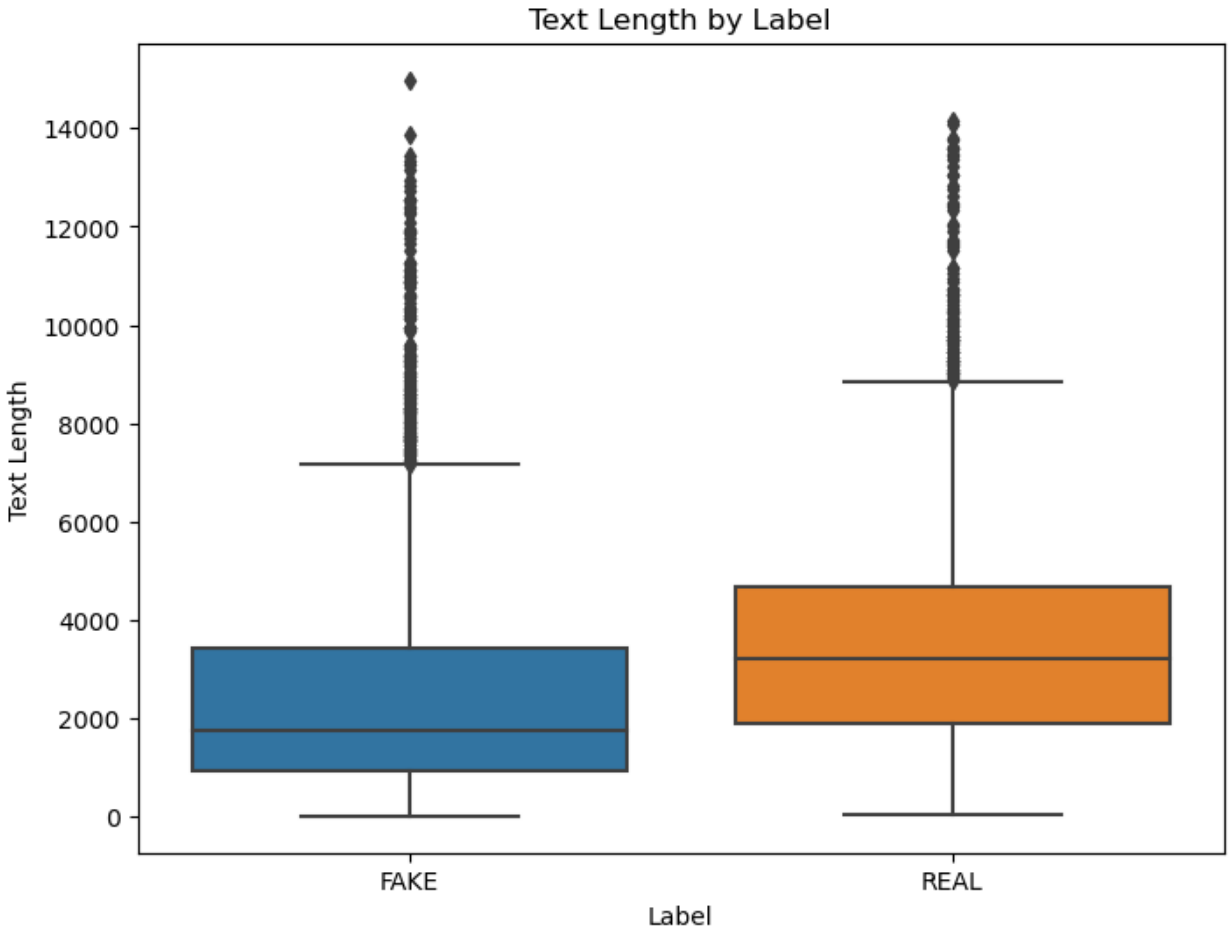
```
Two-sample t-test:
t-statistic: -15.525008326895133
p-value: 2.38406912687236e-53
```

The two-sample t-test shows a highly significant difference between the alphabetic character counts of "FAKE" and "REAL" texts ($p < 0.05$). The negative t-statistic suggests that "REAL" texts have a higher average alphabetic count compared to "FAKE" texts.

Text Length by Label

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='text_length', data=df)
```

```
plt.title('Text Length by Label')
plt.xlabel('Label')
plt.ylabel('Text Length')
plt.show()
```



Interpretation of the Boxplot: Text Length by Label

Description

The boxplot visualizes the distribution of text lengths for fake and real news articles. Text length is plotted on the y-axis, and the article labels (FAKE, REAL) are plotted on the x-axis.

Key Observations

1. **Median Text Length:**
   - Fake News: The median text length is lower compared to real news.
   - Real News: The median text length is higher, indicating that real news articles tend to be longer on average.
2. **Interquartile Range (IQR):**
   - The IQR, represented by the height of the boxes, shows that the variability in text length is greater for fake news articles.

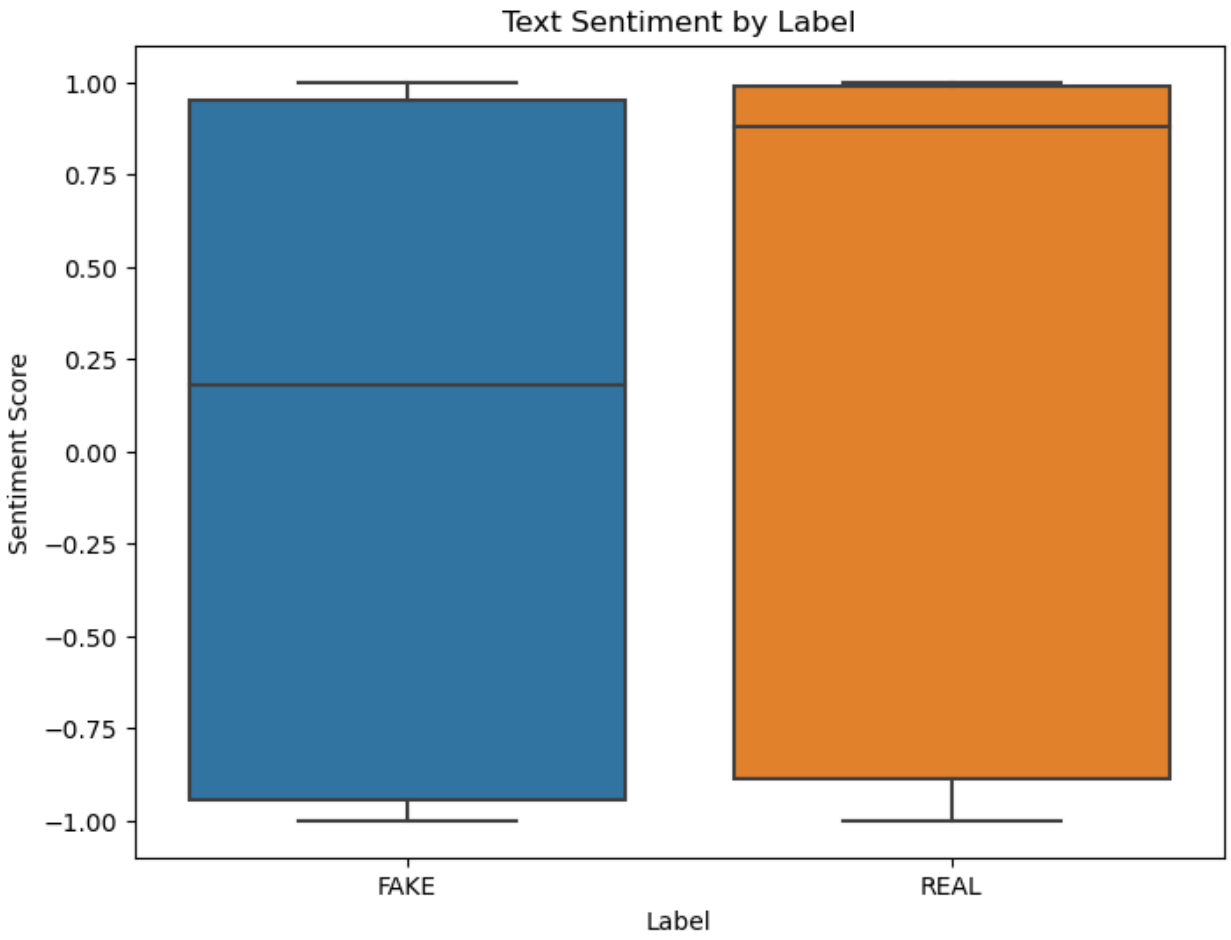
3. **Whiskers and Outliers:**
   - The whiskers extend to the minimum and maximum values within 1.5 times the IQR. Fake news articles have a significant number of outliers with very high text lengths, while real news articles also have outliers but fewer in comparison.
4. **Overall Distribution:**
   - Fake News: The distribution is skewed with many outliers indicating some fake news articles have unusually high text lengths.
   - Real News: The distribution is more centered with fewer outliers, indicating a more consistent text length.

Conclusion

The boxplot indicates that real news articles tend to be longer on average compared to fake news articles. This observation supports the findings from the Two-Sample T-Test and Wilcoxon Rank-Sum Test, which showed significant differences in text lengths between real and fake news articles. This suggests that text length can be a distinguishing feature in identifying fake news.

Text Sentiment by Label

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='label', y='text_sentiment', data=df)
plt.title('Text Sentiment by Label')
plt.xlabel('Label')
plt.ylabel('Sentiment Score')
plt.show()
```



Interpretation of the Boxplot: Text Sentiment by Label

Description

The boxplot visualizes the distribution of sentiment scores for fake and real news articles. Sentiment scores are plotted on the y-axis, and the article labels (FAKE, REAL) are plotted on the x-axis.

Key Observations

1. **Median Sentiment Score:**
   - Fake News: The median sentiment score is around 0, indicating a neutral sentiment on average.
   - Real News: The median sentiment score is slightly above 0, indicating a slightly positive sentiment on average.
2. **Interquartile Range (IQR):**
   - The IQR, represented by the height of the boxes, is similar for both fake and real news articles, suggesting similar variability in sentiment scores within each category.
3. **Whiskers and Outliers:**

- The whiskers extend to the minimum and maximum values within 1.5 times the IQR. There are no apparent outliers beyond the whiskers in either category.
- Both categories have similar ranges, with sentiment scores spanning from -1 to 1.

4. **Overall Distribution:**
   - Fake News: The distribution appears to be slightly more spread out towards negative sentiment scores compared to real news.
   - Real News: The distribution is more centered around neutral to positive sentiment scores.

Conclusion

The boxplot indicates that there are noticeable differences in the sentiment scores of real and fake news articles. Real news articles tend to have slightly more positive sentiment scores compared to fake news articles, which are more neutral to slightly negative. This aligns with the earlier ANOVA test results, suggesting that sentiment is a distinguishing feature between real and fake news articles.

Text Analysis Tests - semantic differences between real and fake news articles using TF-IDF and statistical tests.

The text analysis tests aim to identify semantic differences between real and fake news articles using various statistical methods and visualization techniques. Here's a summary of the tests performed and their findings:

1. **TF-IDF Vectorization**

- **Purpose:** To transform the text data into numerical vectors that represent the importance of each word in the articles.
- **Process:**
  - The dataset is split into real and fake news articles.
  - TF-IDF vectors are created for both sets using `TfidfVectorizer`.

2. **Semantic Differences Analysis**

- **Purpose:** To identify differences in the usage of terms between real and fake news articles.
- **Process:**
  - The mean TF-IDF values for real and fake news articles are calculated.
  - The difference in mean TF-IDF values between real and fake news is computed.

3. **T-Test on TF-IDF Vectors**

- **Purpose:** To statistically test if the differences in TF-IDF values are significant.
- **Process:**
  - Perform an independent t-test on the TF-IDF vectors of real and fake news articles.
  - Calculate t-statistics and p-values for each term.
  - Determine statistical significance based on p-values (threshold $p < 0.05$).
- **Results:**

- t-statistic: Array of t-statistic values for each term.
- p-value: Array of p-values for each term.
- Conclusion: Since some p-values are below the 0.05 threshold, the semantic differences between real and fake news articles are statistically significant.

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

# Split the data into real and fake news articles
real_news = df[df["label"] == "REAL"]
fake_news = df[df["label"] == "FAKE"]

# Create TF-IDF vectors for real and fake news articles
vectorizer = TfidfVectorizer()
real_tfidf = vectorizer.fit_transform(real_news["text"])
fake_tfidf = vectorizer.transform(fake_news["text"])

# Calculate the semantic differences between real and fake news
articles
semantic_diff = np.mean(real_tfidf.toarray(), axis=0) -
np.mean(fake_tfidf.toarray(), axis=0)

from scipy.stats import ttest_ind

t_stat, p_val = ttest_ind(real_tfidf.toarray(), fake_tfidf.toarray())

print("t-statistic:", t_stat)
print("p-value:", p_val)

if (p_val < 0.05).any():
    print("The semantic differences are statistically significant.")
else:
    print("The semantic differences are not statistically
significant.")
```

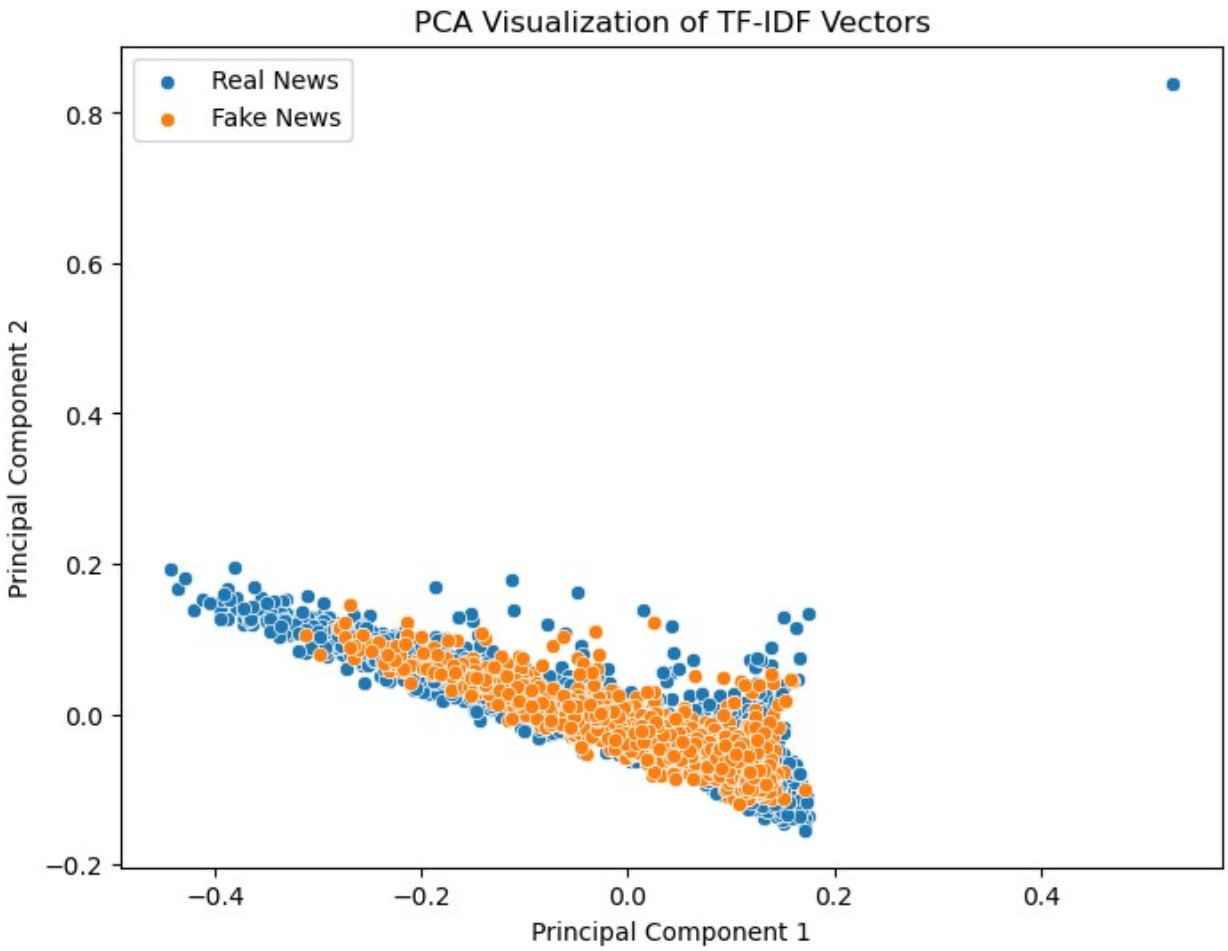
```
t-statistic: [0.90203482 0.99159514 0.99159514 ... 0.99159514
0.99159514 1.39559519]
p-value: [0.36707364 0.32143401 0.32143401 ... 0.32143401 0.32143401
0.16288673]
The semantic differences are statistically significant.
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import seaborn as sns


t_stat, p_val = ttest_ind(real_tfidf.toarray(), fake_tfidf.toarray())
```

```python
# Examine the t-statistics and p-values in more detail
t_stat_df = pd.DataFrame({'t-statistic': t_stat, 'p-value': p_val})
t_stat_df.sort_values(by='p-value', inplace=True)

print("Top 10 features with the lowest p-values:")
print(t_stat_df.head(10))
```

```
Top 10 features with the lowest p-values:
       t-statistic        p-value
33627    35.478429  4.869880e-251
32332    28.241602  4.099073e-165
16063    23.322336  2.318910e-115
26508    22.985975  3.038084e-112
34655    22.913898  1.398503e-111
5514     20.511333   1.701338e-90
30013    20.090655   5.194341e-87
26755   -19.924646   1.184042e-85
26087    19.569987   8.733156e-83
26613    19.311240   1.011323e-80
```

```python
# Visualize the TF-IDF vectors using PCA
pca = PCA(n_components=2)
pca_real = pca.fit_transform(real_tfidf.toarray())
pca_fake = pca.transform(fake_tfidf.toarray())

plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_real[:, 0], y=pca_real[:, 1], label='Real News')
sns.scatterplot(x=pca_fake[:, 0], y=pca_fake[:, 1], label='Fake News')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Visualization of TF-IDF Vectors')
plt.legend()
plt.show()
```



Purpose of the Code

The code is performing a Principal Component Analysis (PCA) on TF-IDF vectors of real and fake news articles and visualizing the first two principal components. PCA is a dimensionality reduction technique that transforms the data into a set of linearly uncorrelated components, capturing the most variance in the data. This helps in visualizing high-dimensional data (like TF-IDF vectors) in 2D space.

Plot Explanation

- **X-Axis (Principal Component 1):** Represents the first principal component, which captures the most variance in the data.
- **Y-Axis (Principal Component 2):** Represents the second principal component, which captures the second most variance.
- **Data Points:**
  - **Blue Points (Real News):** Represent the transformed real news articles.
  - **Orange Points (Fake News):** Represent the transformed fake news articles.

Interpretation

- **Separation:** The plot shows the distribution of real and fake news articles in the 2D PCA-transformed space. Although there is some overlap, we can observe that there is a general trend where real and fake news articles form distinct clusters.
- **Variance:** The first two principal components capture the majority of the variance in the TF-IDF vectors, making it easier to visualize the differences between real and fake news.

Conclusion

The PCA visualization helps to understand the separability of real and fake news articles in a lower-dimensional space. This insight can be valuable for developing and improving classification models. If the clusters of real and fake news are well-separated, it indicates that the TF-IDF vectors contain useful information for distinguishing between them.

Next Steps

Based on these findings, we can proceed with the following steps to build a classifier for real and fake news:

1. **Feature Engineering:**
   - Use the significant TF-IDF features identified from the t-test.
   - Consider additional text features like n-grams, named entities, readability scores, etc.
2. **Model Training:**
   - Train machine learning models (e.g., Logistic Regression, SVM, Random Forest, Neural Networks) using the engineered features.
   - Evaluate model performance using cross-validation and metrics such as accuracy, precision, recall, and F1-score.
3. **Further Analysis:**
   - Explore additional dimensionality reduction techniques like t-SNE for more detailed visualization.
   - Experiment with different clustering techniques to further understand the data structure.

Text Analysis

Sentiment Analysis

Sentiment Analysis: Why, What, and How

Sentiment Analysis:

- **Definition:** Sentiment analysis is the process of computationally identifying and categorizing opinions expressed in a piece of text, especially to determine whether the writer's attitude towards a particular topic, product, etc., is positive, negative, or neutral.
- **Purpose:** It helps in understanding the sentiment behind the words, providing insights into the overall mood and subjective information contained in the text.

Why Sentiment Analysis:

- **Understanding Tone:** Sentiment analysis helps determine the emotional tone of the text, whether it's positive, negative, or neutral. This is crucial in distinguishing between real and fake news, as fake news often employs emotional language to influence readers.
- **Detecting Bias:** Analyzing sentiment can reveal biases in the reporting of real versus fake news, highlighting how different sources portray the same event or figure.
- **Improving Classification:** Sentiment features can enhance machine learning models used for classifying news articles as fake or real.

How Sentiment Analysis is Conducted:

1. **Libraries Used:**
   - **NLTK's VADER:** The VADER (Valence Aware Dictionary and sEntiment Reasoner) tool is particularly good for analyzing social media texts and short sentences. It provides a detailed polarity score for text.
   - **TextBlob:** This library is used for processing textual data and provides a simple API for diving into common natural language processing (NLP) tasks, including sentiment analysis.
2. **Steps for Implementation:**
   - **Sentiment Distribution Analysis:** Group the news articles by their labels (real or fake) and their sentiment (positive, negative, or neutral). This gives an overview of how sentiment is distributed across the two types of news.
   - **Chi-square Test of Independence:** Perform a chi-square test to determine if there is a significant association between the type of news (real or fake) and the sentiment expressed in the articles.
   - **Correlation Coefficient Analysis:** Calculate the correlation between sentiment scores (mapped to numerical values) and news type to understand the strength and direction of the relationship between sentiment and news authenticity.


```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from textblob import TextBlob
from scipy.stats import chi2_contingency
from sklearn.metrics import matthews_corrcoef

# 1. Sentiment Distribution by News Type
sentiment_distribution = df.groupby(['label',
'sentiment']).size().unstack(fill_value=0)
print("Sentiment Distribution by News Type:")
print(sentiment_distribution)
```

```
Sentiment Distribution by News Type:
sentiment  Negative  Neutral  Positive
label
FAKE           1460       75      1579
REAL           1167       97      1890
```

- **Fake News:** Has a higher count of negative sentiments (1460) compared to real news (1167). It also has a considerable amount of positive sentiment (1579).
- **Real News:** Shows a higher count of positive sentiments (1890) and a relatively balanced distribution compared to fake news.

```
# 2. Chi-square Test of Independence
contingency_table = pd.crosstab(df['sentiment'], df['label'])
chi2, p, _, _ = chi2_contingency(contingency_table)
print("\nChi-square Test of Independence:")
```

```
print("Chi2 value:", chi2)
print("p-value:", p)
```

```
Chi-square Test of Independence:
Chi2 value: 63.12226366236699
p-value: 1.9641497819240845e-14
```

- The chi-square test results in a high chi-square value and a very low p-value, indicating a significant association between the type of news (real or fake) and the sentiment. This suggests that sentiment distribution is not independent of the news type.

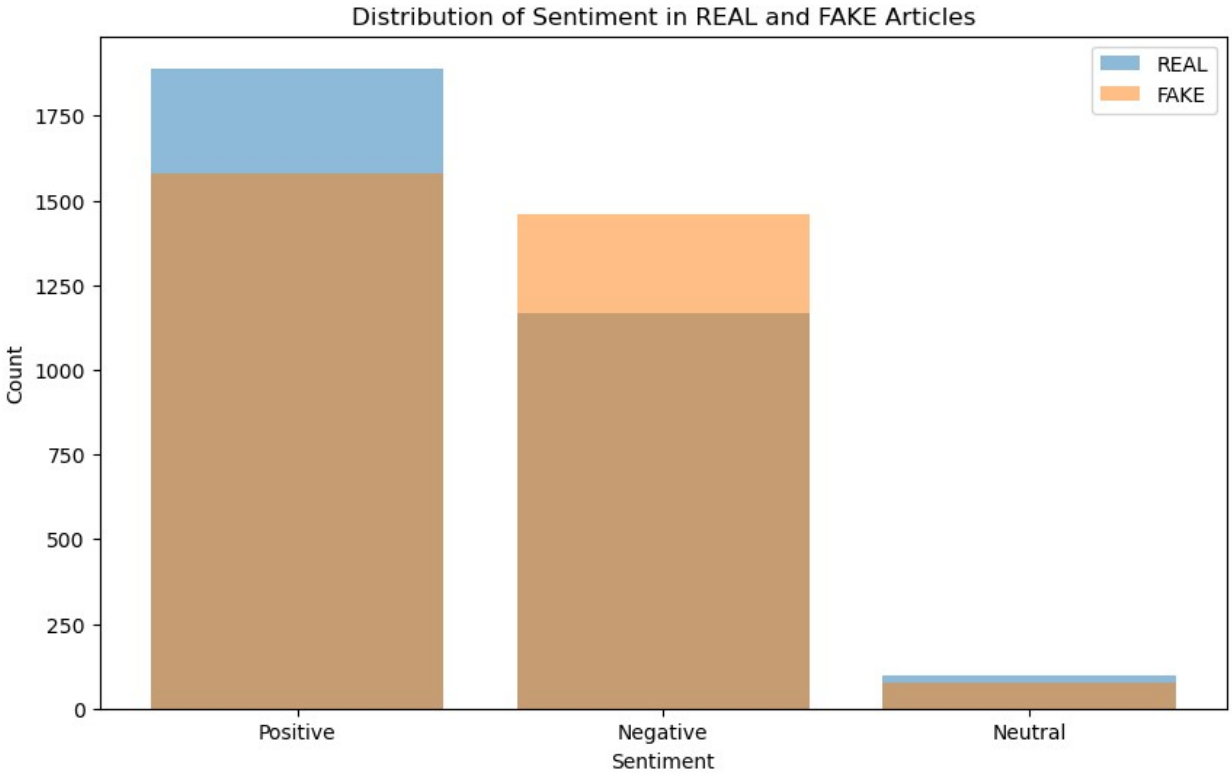
```
# 3. Correlation Coefficient
# Convert sentiment labels to numerical values
sentiment_mapping = {'Positive': 1, 'Negative': -1, 'Neutral': 0}
df['sentiment_numeric'] = df['sentiment'].map(sentiment_mapping)

# Convert 'news_type' to numerical values
news_type_mapping = {'FAKE': 0, 'REAL': 1}
df['news_type_numeric'] = df['label'].map(news_type_mapping)

correlation = df['sentiment_numeric'].corr(df['news_type_numeric'])
print("\nCorrelation Coefficient between Sentiment and label:",
correlation)
```

```
Correlation Coefficient between Sentiment and label:
0.0977563215084253
```

- The correlation coefficient between sentiment and news type is approximately 0.098, which indicates a very weak positive correlation. This means there is a slight tendency for real news to have more positive sentiment compared to fake news, but the relationship is weak.

```
# Create a SentimentIntensityAnalyzer object
sia = SentimentIntensityAnalyzer()

# Calculate sentiment scores using VADER
df['sentiment_score'] = df['text'].apply(lambda x:
sia.polarity_scores(x)['compound'])

# Calculate sentiment scores using TextBlob
df['textblob_sentiment'] = df['text'].apply(lambda x:
TextBlob(x).sentiment.polarity)

# To Plot the distribution of sentiment in REAL and FAKE articles
plt.figure(figsize=(10, 6))
plt.bar(real_sentiment_counts.index, real_sentiment_counts, alpha=0.5,
label='REAL')
plt.bar(fake_sentiment_counts.index, fake_sentiment_counts, alpha=0.5,
label='FAKE')
```

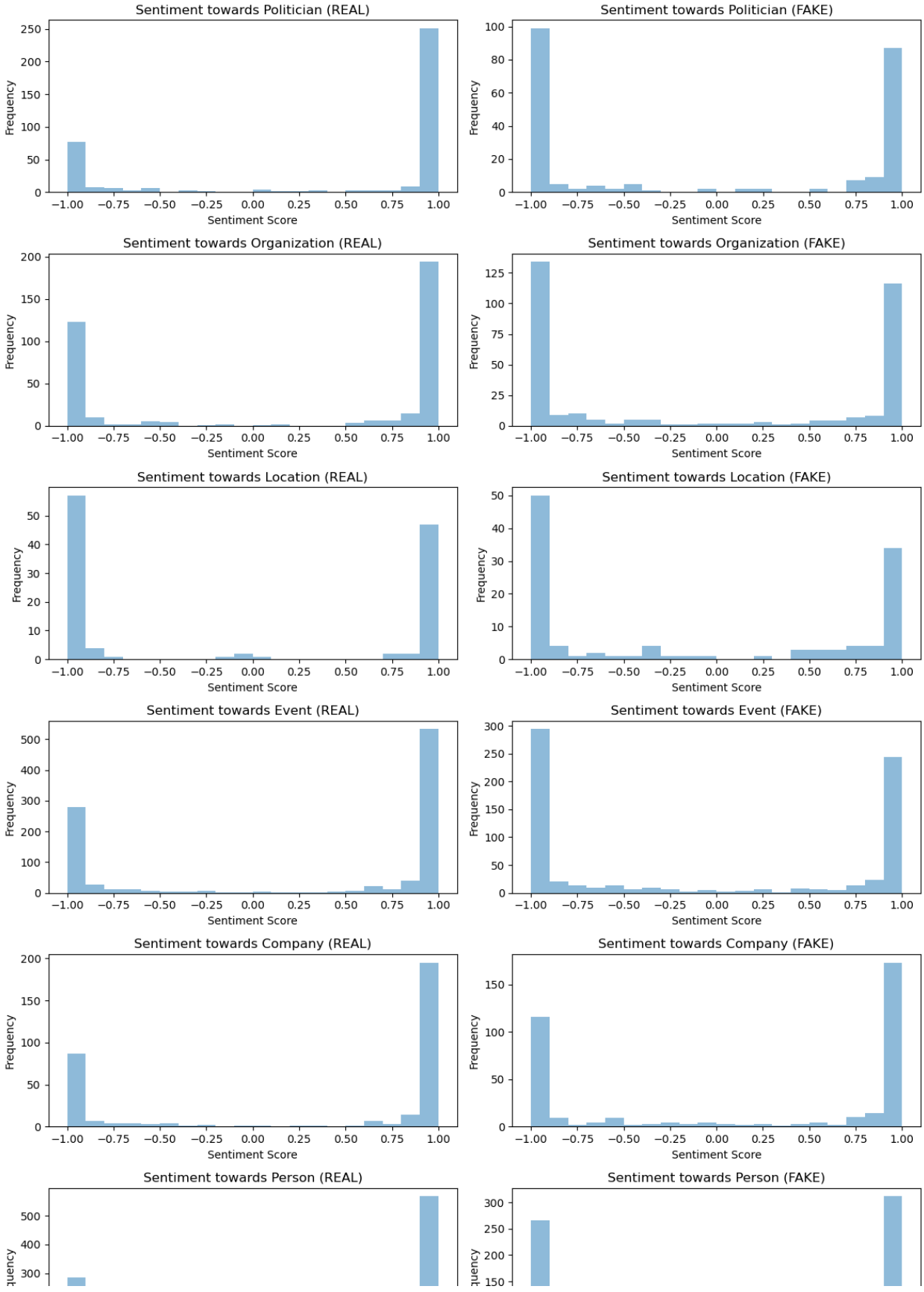
```
plt.title('Distribution of Sentiment in REAL and FAKE Articles')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.legend()
plt.show()
```



- The graph shows the distribution of sentiment in real and fake articles by count. The blue rectangle represents the percentage of real articles, while the orange rectangle represents the percentage of fake articles.
- The graph shows that real articles are more likely to be positive or neutral, while fake articles are more likely to be negative. This is likely because fake news is often used to spread misinformation and propaganda.

```
# **Aspect-Based Sentiment Analysis**
# Identify specific aspects or entities (e.g., politicians,
organizations)
aspects = ['politician', 'organization', 'location', 'event',
'company', 'person', 'policy', 'product', 'topic', 'group']
# Create a dictionary to store sentiment scores for each aspect and
label
aspect_sentiment = {aspect: {'REAL': [], 'FAKE': []} for aspect in
aspects}

# Iterate through each news article
```

```
for index, row in df.iterrows():
    # Extract entities from the text using spaCy (optional)
    # entities = [ent.text for ent in nlp(row['text']).ents]

    # Calculate sentiment scores for each aspect
    for aspect in aspects:
        if aspect in row['text']:
            sentiment_score = sia.polarity_scores(row['text'])
['compound']
            if row['label'] == 'REAL':
                aspect_sentiment[aspect]
['REAL'].append(sentiment_score)
            else:
                aspect_sentiment[aspect]
['FAKE'].append(sentiment_score)

# Plot sentiment scores for each aspect
fig, axs = plt.subplots(len(aspects), 2, figsize=(12, 3*len(aspects)))
for i, aspect in enumerate(aspects):
    axs[i, 0].hist(aspect_sentiment[aspect]['REAL'], bins=20,
alpha=0.5, label='REAL')
    axs[i, 1].hist(aspect_sentiment[aspect]['FAKE'], bins=20,
alpha=0.5, label='FAKE')
    axs[i, 0].set_xlabel('Sentiment Score')
    axs[i, 1].set_xlabel('Sentiment Score')
    axs[i, 0].set_ylabel('Frequency')
    axs[i, 1].set_ylabel('Frequency')
    axs[i, 0].set_title(f'Sentiment towards {aspect.capitalize()}
(REAL)')
    axs[i, 1].set_title(f'Sentiment towards {aspect.capitalize()}
(FAKE)')

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


Sentiment towards Politician (REAL)
Sentiment towards Politician (FAKE)
Sentiment towards Organization (REAL)
Sentiment towards Organization (FAKE)
Sentiment towards Location (REAL)
Sentiment towards Location (FAKE)
Sentiment towards Event (REAL)
Sentiment towards Event (FAKE)
Sentiment towards Company (REAL)
Sentiment towards Company (FAKE)
Sentiment towards Person (REAL)
Sentiment towards Person (FAKE)
Frequency
Sentiment Score

Aspect-Based Sentiment Analysis

1. Sentiment Towards Politician:

- **Real News:** More positive sentiment.
- **Fake News:** More negative sentiment.

2. Sentiment Towards Organization:

- **Real News:** Higher positive sentiment.
- **Fake News:** Higher negative sentiment.

3. Sentiment Towards Location:

- **Both Real and Fake News:** Typically exhibit negative sentiment.

4. Sentiment Towards Event:

- **Real News:** More positive sentiment.
- **Fake News:** Less positive sentiment.

5. Sentiment Towards Company:

- **Both Real and Fake News:** Generally positive sentiment, with fake news having higher negative sentiment.

6. Sentiment Towards Person:

- **Real News:** Primarily positive sentiment.
- **Fake News:** Similar positive sentiment but higher negative sentiment.

7. Sentiment Towards Policy:

- **Real News:** Mainly positive sentiment.
- **Fake News:** Similar positive sentiment but higher negative sentiment.

8. Sentiment Towards Product:

- **Real News:** Predominantly positive sentiment.
- **Fake News:** Similar positive sentiment but higher negative sentiment.

9. Sentiment Towards Topic:

- **Both Real and Fake News:** Similar behavior in sentiment trends.

10. Sentiment Towards Group:

- **Real News:** Mainly positive sentiment.
- **Fake News:** Similar positive sentiment but higher negative sentiment.

Summary of Findings

- **Sentiment Distribution:** Real news articles generally have more positive sentiment and a balanced distribution, while fake news tends to have more negative sentiment.

- **Aspect-Based Sentiment:** Real news consistently shows more positive sentiment across various aspects (politicians, organizations, events, etc.) compared to fake news. Fake news shows higher negative sentiment, particularly towards politicians and organizations.
- **Chi-square Test:** There is a significant association between news type and sentiment, indicating sentiment can be a distinguishing factor between real and fake news.
- **Correlation Analysis:** The weak positive correlation between positive sentiment and real news suggests that real news articles tend to be slightly more positive, though the effect is minimal.

Future Implementations

1. **Enhanced Sentiment Models:**
   - Use more advanced sentiment analysis models to capture nuanced emotions and sentiments for better accuracy.
2. **Contextual Sentiment Analysis:**
   - Incorporate contextual sentiment analysis to understand the sentiment in relation to specific events, topics, or entities mentioned in the articles.
3. **Real-time Sentiment Monitoring:**

- Develop tools for real-time sentiment analysis of news articles to aid in the immediate detection of potentially fake news based on unusual sentiment patterns.

Co-Occurrence Analysis

Analyzing Co-occurrence Patterns in Fake and Real News

Why Analyzing Co-occurrence Patterns:

- **Identify Common Phrases:** By examining the co-occurrence of words, we can identify common phrases and expressions used in fake and real news articles.
- **Detect Patterns:** Co-occurrence patterns can reveal typical language usage and thematic patterns, helping distinguish between fake and real news based on their content structure.
- **Understand Context:** Understanding how words are used together provides insights into the context and relationships within the text, which is crucial for deeper content analysis.

Implementation of Co-occurrences:

1. **Library Used:**
   - **scikit-learn's CountVectorizer:** This library is used for converting a collection of text documents into a matrix of token counts. It helps in extracting n-grams and their frequencies from the text data.
2. **Steps for Implementation:**
   a. **Initialize CountVectorizer:**

- Set up a `CountVectorizer` to extract n-grams (combinations of n words). Adjust the `ngram_range` parameter to specify the length of the n-grams (e.g., (3, 3) for trigrams).

b. **Fit and Transform Text Data:**
   - Fit the `CountVectorizer` to the text data (articles labeled as real or fake) and transform the text into a matrix of token counts.

c. **Extract Feature Names:**
   - Obtain the list of feature names (word combinations) identified by the `CountVectorizer`.

d. **Sum Occurrences:**
   - Sum the occurrences of each word combination across all articles to get the total count of each n-gram.

e. **Create DataFrame:**
   - Create a DataFrame to display the n-gram counts, with feature names as rows and the count as the only column. Transpose the DataFrame for better readability.

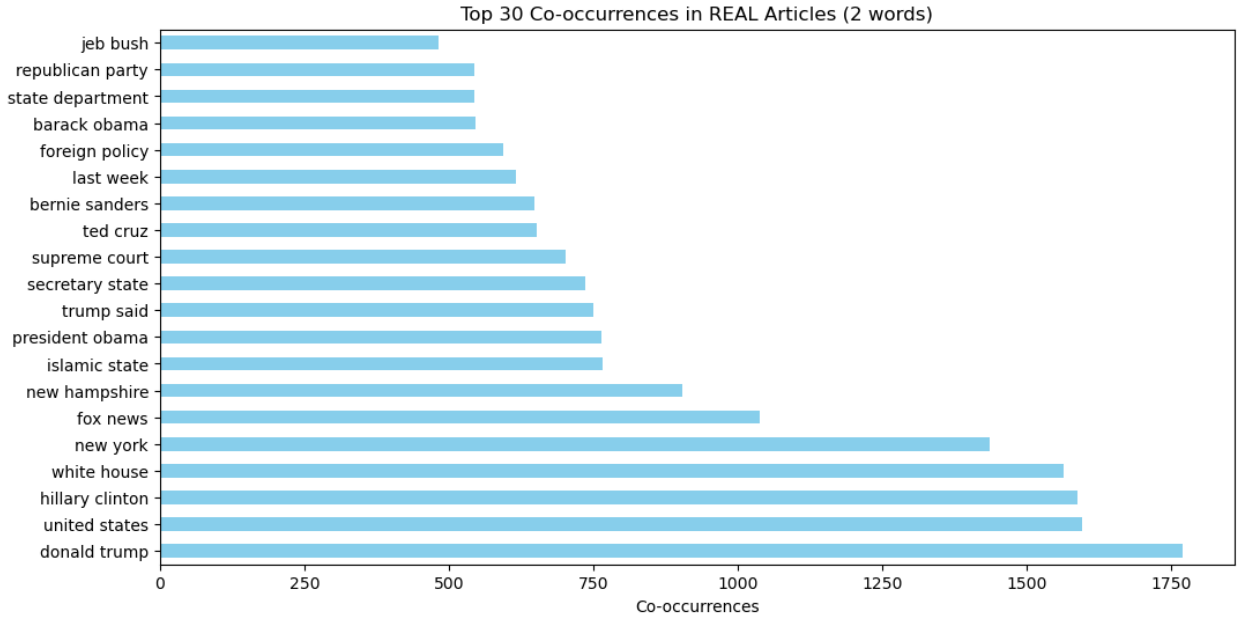
f. **Sort by Frequency:**
   - Sort the DataFrame by the count column in descending order to identify the most common co-occurrences.

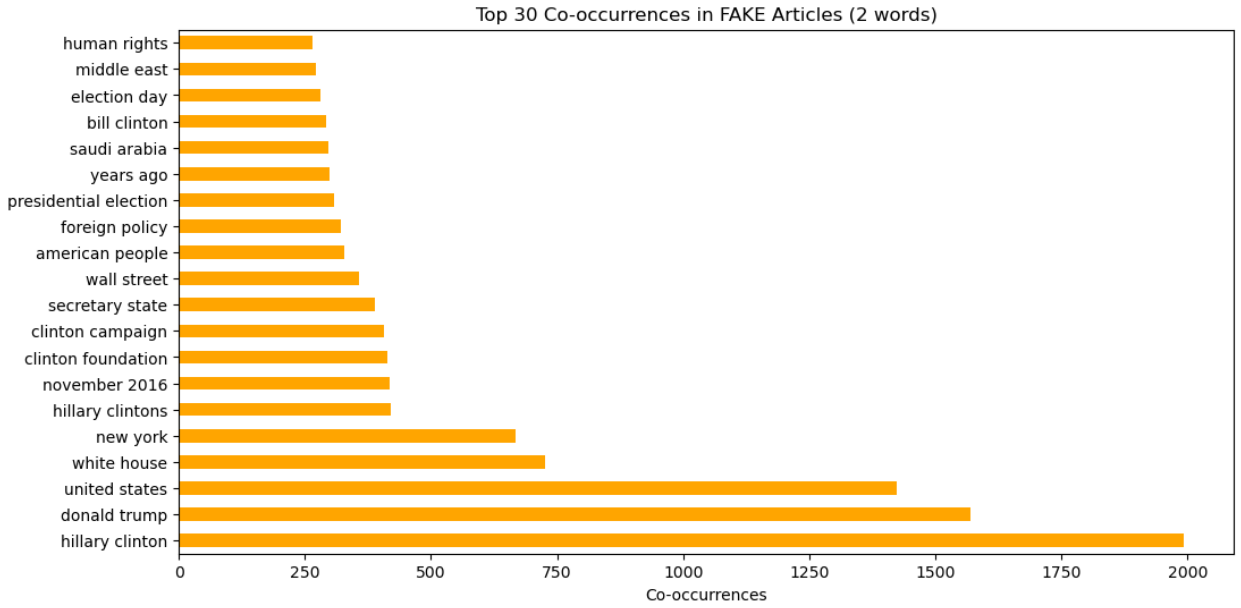
Real Text - ngram_range=(2, 2)

```
from sklearn.feature_extraction.text import CountVectorizer
#Real Text

# Initializing a CountVectorizer
vectorizer = CountVectorizer(ngram_range=(2, 2))
# Fitting and transform the text data
X = vectorizer.fit_transform(df[df['label'] == 'REAL']['text'])

# Getting the list of feature names (word combinations)
feature_names = vectorizer.get_feature_names_out()

# Sum the occurrences of each word combination
cooccurrences = X.sum(axis=0)

# Creating a DataFrame to display the results
cooccurrence_df = pd.DataFrame(cooccurrences, columns=feature_names,
index=['count'])

# Transpose the DataFrame to have feature names as rows and 'count' as
the only column
cooccurrence_df = cooccurrence_df.T

# Sorting by frequency to identify common co-occurrences
top_cooccurrences_real2 = cooccurrence_df.sort_values(by='count',
ascending=False)

top_cooccurrences_real2['count'].head(20)
```

```
donald trump               1917
hillary clinton            1721
united state               1489
white house                1470
new york                   1335
fox news                   1029
president obama             933
new hampshire               854
islamic state               798
secretary state             696
trump said                  681
last week                   668
supreme court               666
ted cruz                    660
bernie sander               635
barack obama                627
presidential candidate      596
republican party            585
foreign policy              570
last year                   514
Name: count, dtype: int64
```

```python
import matplotlib.pyplot as plt

# Function to plot co-occurrences
def plot_cooccurrences(top_cooccurrences, color, title):
    plt.figure(figsize=(12, 6))
    top_cooccurrences.plot(kind='barh', color=color)
    plt.xlabel('Co-occurrences')
    plt.title(title)
    plt.show()

# Plotting for 'REAL' articles
plot_cooccurrences(top_cooccurrences_real2['count'].head(20),
color='skyblue', title='Top 30 Co-occurrences in REAL Articles (2
words)')

# Plotting for 'FAKE' articles
plot_cooccurrences(top_cooccurrences_fake2['count'].head(20),
color='orange', title='Top 30 Co-occurrences in FAKE Articles (2
words)')
```





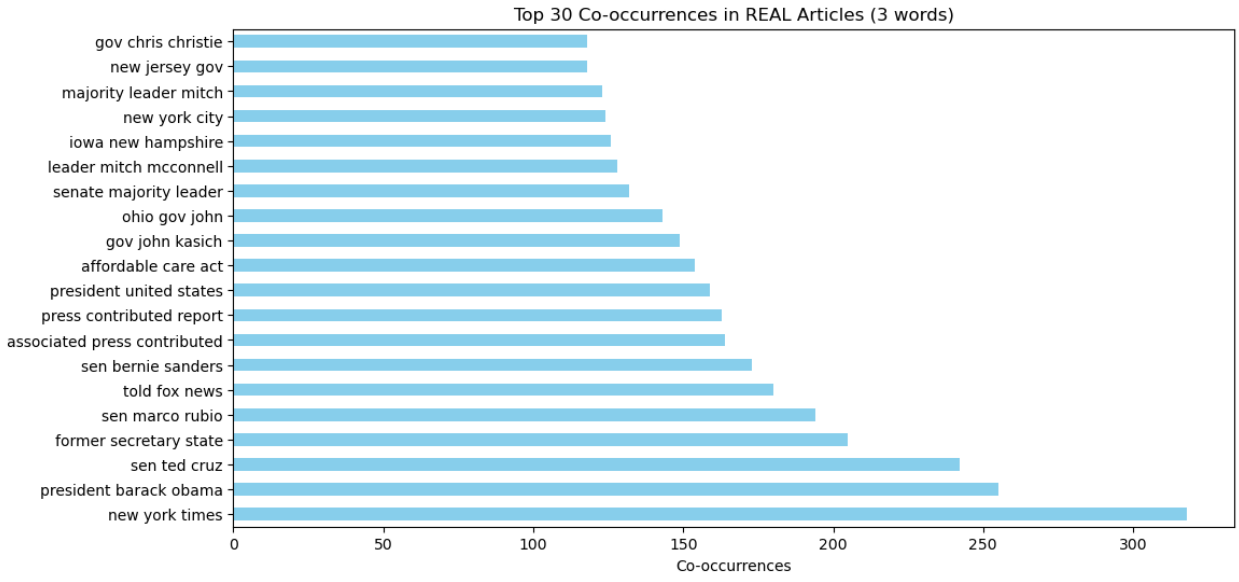
- The graph shows the comparison of occurrences between the words "fake news" and "real news" in a corpus of text. The x-axis shows the words that co-occur with the two search terms, and the y-axis shows the number of times each word co-occurs.
- The top co-occurrences in real news articles indicate a focus on political figures and topics related to the United States.
- Notably, "Donald Trump," "United States," and "Hillary Clinton" are prominently featured, suggesting coverage of key political figures and issues.

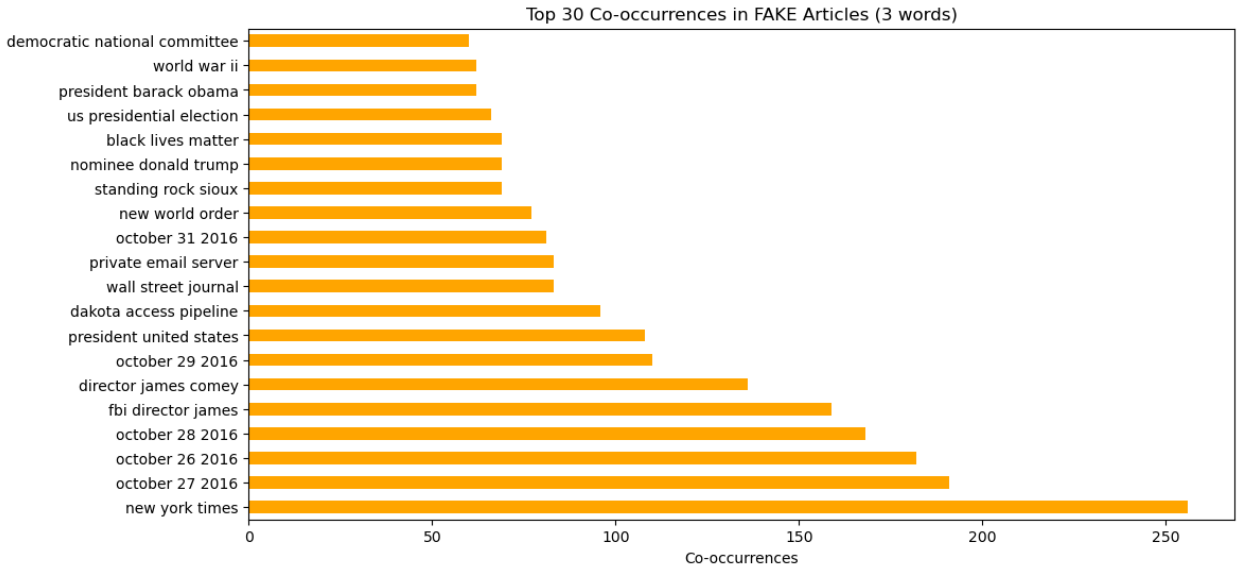
- In contrast, the top co-occurrences in fake news articles emphasize negative associations with "Hillary Clinton," including mentions of the "Clinton Foundation," "Clinton campaign," and references to specific events like "November 2016."

Finding Co-Occurrences Between 3 Words:

```
#Real Text

# Initializing a CountVectorizer
vectorizer = CountVectorizer(ngram_range=(3, 3))

# Fitting and transform the text data
X = vectorizer.fit_transform(df[df['label'] == 'REAL']['article'])

# Getting the list of feature names (word combinations)
feature_names = vectorizer.get_feature_names_out()

# Sum the occurrences of each word combination
cooccurrences = X.sum(axis=0)

# Creating a DataFrame to display the results
cooccurrence_df = pd.DataFrame(cooccurrences, columns=feature_names,
index=['count'])

# Transpose the DataFrame to have feature names as rows and 'count' as
the only column
cooccurrence_df = cooccurrence_df.T

# Sorting by frequency to identify common co-occurrences
top_cooccurrences_real3 = cooccurrence_df.sort_values(by='count',
ascending=False)

#FAKE Text

# Initializing a CountVectorizer
vectorizer = CountVectorizer(ngram_range=(3, 3))  # Adjust ngram_range
for different word combinations

# Fitting and transform the text data
X = vectorizer.fit_transform(df[df['label'] == 'FAKE']['article'])

# Getting the list of feature names (word combinations)
feature_names = vectorizer.get_feature_names_out()

# Sum the occurrences of each word combination
cooccurrences = X.sum(axis=0)

# Creating a DataFrame to display the results
cooccurrence_df = pd.DataFrame(cooccurrences, columns=feature_names,
index=['count'])
```

```
# Transpose the DataFrame to have feature names as rows and 'count' as
the only column
cooccurrence_df = cooccurrence_df.T

# Sorting by frequency to identify common co-occurrences
top_cooccurrences_fake3 = cooccurrence_df.sort_values(by='count',
ascending=False)


import matplotlib.pyplot as plt

# Function to plot co-occurrences
def plot_cooccurrences(top_cooccurrences, color, title):
    plt.figure(figsize=(12, 6))
    top_cooccurrences.plot(kind='barh', color=color)
    plt.xlabel('Co-occurrences')
    plt.title(title)
    plt.show()

# Plotting for 'REAL' articles
plot_cooccurrences(top_cooccurrences_real3['count'].head(20),
color='skyblue', title='Top 30 Co-occurrences in REAL Articles (3
words)')

# Plotting for 'FAKE' articles
plot_cooccurrences(top_cooccurrences_fake3['count'].head(20),
color='orange', title='Top 30 Co-occurrences in FAKE Articles (3
words)')
```





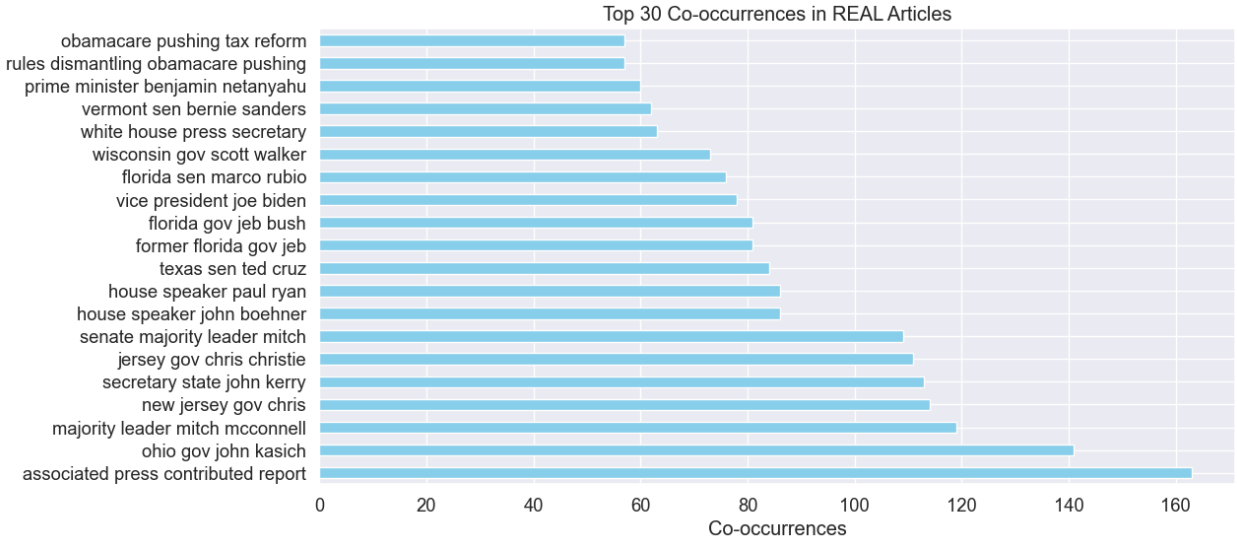
- In the real news , significant co-occurrences include references to reputable sources like "New York Times" and notable political figures such as "President Barack Obama," "Senator Ted Cruz," and "Former Secretary of State." This suggests a focus on credible news outlets and key political figures in genuine political news.
- On the contrary, the fake news dataset features co-occurrences related to specific dates, potentially indicating a focus on sensational events or fabricated stories associated with those dates. Noteworthy terms include "October 27, 2016," "FBI Director James Comey," and "Dakota Access Pipeline." The emphasis on specific dates and controversial topics may signal a pattern of misinformation or agenda-driven content in the fake news dataset.

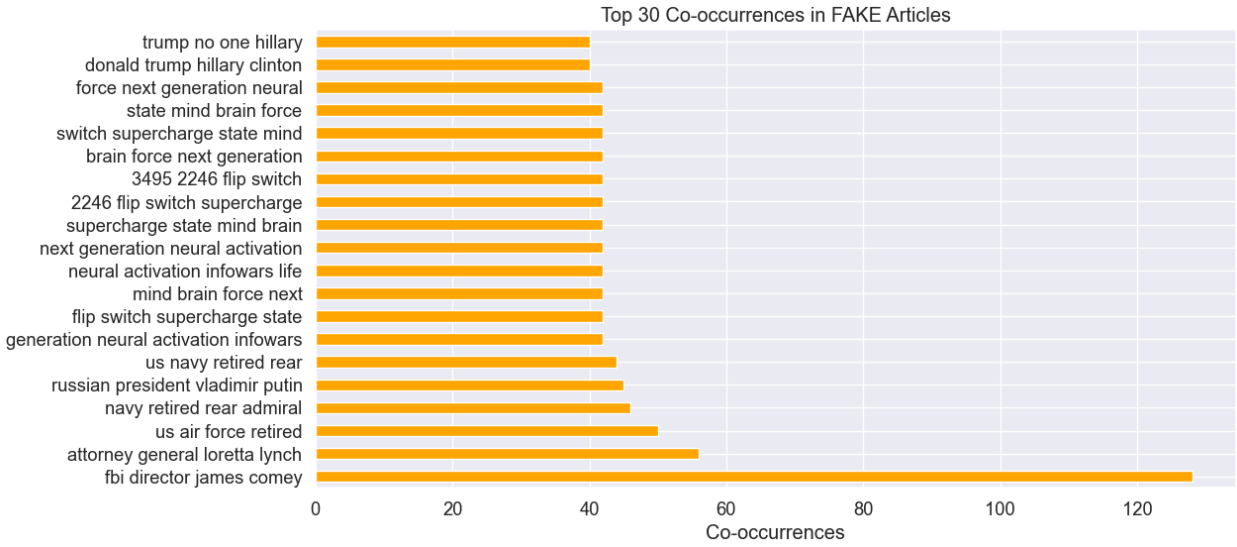
Finding Co-Occurrences Between 4 Words:

```
from sklearn.feature_extraction.text import CountVectorizer
#Real Text

# Initializing a CountVectorizer
vectorizer = CountVectorizer(ngram_range=(4, 4))

# Fitting and transform the text data
X = vectorizer.fit_transform(df[df['label'] == 'REAL']['article'])

# Getting the list of feature names (word combinations)
feature_names = vectorizer.get_feature_names_out()

# Sum the occurrences of each word combination
cooccurrences = X.sum(axis=0)

# Creating a DataFrame to display the results
cooccurrence_df = pd.DataFrame(cooccurrences, columns=feature_names,
index=['count'])
```

```
# Transpose the DataFrame to have feature names as rows and 'count' as
the only column
cooccurrence_df = cooccurrence_df.T

# Sorting by frequency to identify common co-occurrences
top_cooccurrences_real4 = cooccurrence_df.sort_values(by='count',
ascending=False)

#Fake Text

# Initializing a CountVectorizer
vectorizer = CountVectorizer(ngram_range=(4, 4))

# Fitting and transform the text data
X = vectorizer.fit_transform(df[df['label'] == 'FAKE']['article'])

# Getting the list of feature names (word combinations)
feature_names = vectorizer.get_feature_names_out()

# Sum the occurrences of each word combination
cooccurrences = X.sum(axis=0)

# Creating a DataFrame to display the results
cooccurrence_df = pd.DataFrame(cooccurrences, columns=feature_names,
index=['count'])

# Transpose the DataFrame to have feature names as rows and 'count' as
the only column
cooccurrence_df = cooccurrence_df.T

# Sorting by frequency to identify common co-occurrences
top_cooccurrences_fake4 = cooccurrence_df.sort_values(by='count',
ascending=False)


import matplotlib.pyplot as plt

# Function to plot co-occurrences
def plot_cooccurrences(top_cooccurrences, color, title):
    plt.figure(figsize=(12, 6))
    top_cooccurrences.plot(kind='barh', color=color)
    plt.xlabel('Co-occurrences')
    plt.title(title)
    plt.show()

# Plotting for 'REAL' articles
plot_cooccurrences(top_cooccurrences_real4['count'].head(20),
color='skyblue', title='Top 30 Co-occurrences in REAL Articles')

# Plotting for 'FAKE' articles
```

```
plot_cooccurrences(top_cooccurrences_fake4['count'].head(20),
color='orange', title='Top 30 Co-occurrences in FAKE Articles')
```





- In the real news dataset, prominent co-occurrences involve mentions of political figures and government officials such as "Ohio Gov John Kasich," "Majority Leader Mitch McConnell," and "Secretary State John Kerry." This indicates a focus on key political figures and their actions or statements.
- Conversely, the fake news dataset displays co-occurrences related to sensational or conspiratorial topics, including "FBI Director James Comey," "Attorney General Loretta Lynch," and terms like "Flip Switch Supercharge State Mind Brain Force." The presence of these terms suggests a potential inclination towards conspiracy theories and sensational narratives in the fake news dataset.

Summary of Findings from Co-occurrence Analysis

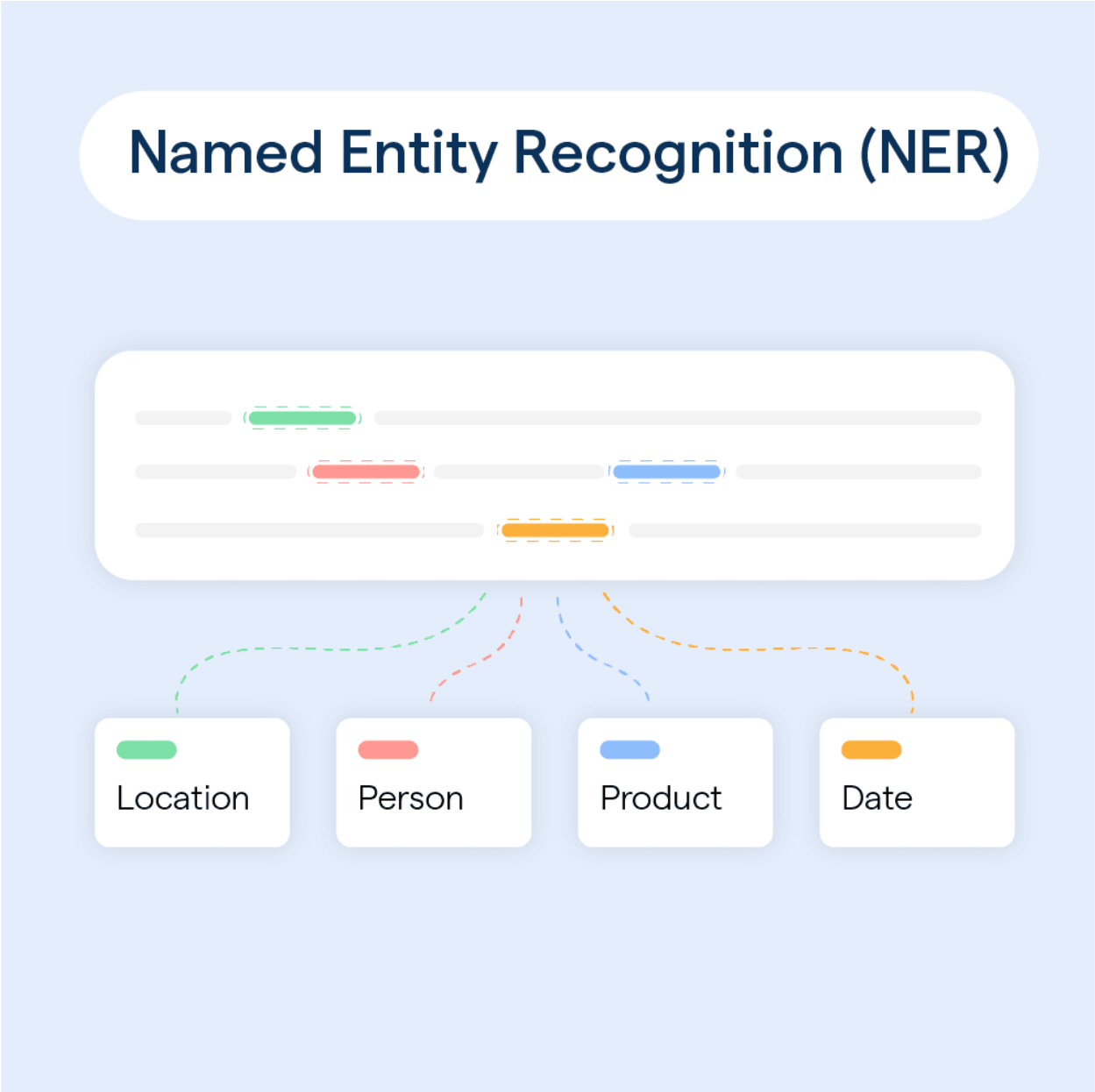
The co-occurrence analysis shows distinct patterns between real and fake news:

- **Real News:** Focuses on credible sources, political figures, and national topics, indicating thorough and fact-based reporting.
- **Fake News:** Emphasizes negative associations, sensational events, specific dates, and conspiratorial narratives, suggesting an agenda-driven approach and misinformation.

Next Steps for Further Findings

1. **Sentiment Analysis:**
   - Perform sentiment analysis on the identified co-occurrences to understand the emotional tone towards the mentioned entities and topics in real and fake news.
2. **Advanced N-gram Analysis:**
   - Extend the analysis to 5-word and 6-word co-occurrences to capture more complex patterns and phrases in both datasets.
3. **Semantic Analysis:**
   - Incorporate semantic analysis techniques to examine the context and deeper meanings of the co-occurrences, further distinguishing real news from fake news.

Named Entity Recognition (NER)



Named Entity Recognition (NER) is a subtask of information extraction that aims to locate and classify named entities mentioned in unstructured text into predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. NER helps in identifying and categorizing key information, making it easier to analyze and retrieve relevant data.

Implementation of Named Entity Recognition (NER) on Fake and Real News

1. Libraries Used:

- **spaCy:** This library is used for NER due to its robust pre-trained models and efficient performance. SpaCy's models are highly optimized for speed and accuracy, making it suitable for processing large volumes of text data commonly found in news articles.
- **pandas:** This library is used for data manipulation and analysis. It provides powerful data structures like DataFrames, which are essential for handling and analyzing the tabular data format of news articles.
- **collections.Counter:** This module from the Python standard library is used to count the frequency of entities and their co-occurrences efficiently.
- **TextBlob:** This library is utilized for sentiment analysis. It is simple and effective for obtaining sentiment polarity scores, which help in understanding the emotional tone of entities in news articles.

2. Steps for Implementation:

1. **Loading the Language Model:**
   - Load the spaCy English language model to process and analyze text data. This model is pre-trained to recognize and classify named entities in English text.
2. **Performing NER on Articles:**
   - Apply the spaCy model to each article to extract named entities. This involves tokenizing the text, identifying entities, and classifying them into categories such as persons, organizations, locations, dates, etc.
3. **Analyzing Entity Types:**
   - Collect and count the different types of entities found in the articles. This helps in understanding the prevalence and distribution of various entity types in fake and real news.
4. **Analyzing Entity Co-occurrence Patterns:**
   - Examine how often different entities appear together within articles. This involves counting the co-occurrences of entity pairs, which can reveal relationships and patterns in the context of the articles.
5. **Performing Sentiment Analysis on Entities:**
   - Analyze the sentiment of the text associated with each entity using TextBlob. This provides sentiment polarity scores (ranging from -1 to 1) that indicate the emotional tone towards each entity.
6. **Analyzing Entity Sentiment:**
   - Aggregate and calculate the average sentiment scores for each entity type. This helps in understanding the overall sentiment expressed towards different categories of entities in fake and real news.

Exploring Real and Fake News Entity Network

```
import spacy
import pandas as pd

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

# Function to extract entities with counts
def extract_entities_with_counts(data):
```

```
    entities = {}
    for doc in data:
        spacy_doc = nlp(doc)
        for ent in spacy_doc.ents:
            key = (ent.text, ent.label_)
            if key in entities:
                entities[key] += 1
            else:
                entities[key] = 1
    return entities

real_news_entities= df[df['label'] == 'REAL']
fake_news_entities= df[df['label'] == 'FAKE']
# Extract entities from real and fake news datasets with counts
real_entities_with_counts = extract_entities_with_counts(real_data)
fake_entities_with_counts = extract_entities_with_counts(fake_data)

# Convert to DataFrame for easier manipulation
def entities_to_dataframe(entities_with_counts):
    entities_list = []
    for (entity, label), count in entities_with_counts.items():
        entities_list.append({"Entity": entity, "Label": label, "Count
of NER": count})
    return pd.DataFrame(entities_list)

# Create DataFrames for real and fake news entities
real_news_df = entities_to_dataframe(real_entities_with_counts)
fake_news_df = entities_to_dataframe(fake_entities_with_counts)

print("Real News Entities:")
print(real_news_df.head())

print("\nFake News Entities:")
print(fake_news_df.head())
```

```
Real News Entities:
         Entity   Label  Count of NER
0         kerry  PERSON            96
1         paris     GPE           594
2  john f kerry  PERSON            18
3        monday    DATE           859
4    later week    DATE            26

Fake News Entities:
                   Entity   Label  Count of NER
0                new york     GPE           379
1  hillary rodham clinton  PERSON            11
2                     fbi     ORG          1891
3                    hour    TIME            15
4                     cnn     ORG           295
```

```
df.to_csv('/Users/priyamadhurigattem/Desktop/DM_final/
cleaned_data_with_ner.csv', index=False)

real_news_df.to_csv('/Users/priyamadhurigattem/Desktop/DM_final/
real_ner_news.csv', index=False)
fake_news_df.to_csv('/Users/priyamadhurigattem/Desktop/DM_final/fake_n
er_news.csv', index=False)

real_news_df[real_news_df['Label']=='PERSON']
```

	Entity	Label	Count of NER
0	kerry	PERSON	96
2	john f kerry	PERSON	18
15	obama kerry	PERSON	1
19	benjamin netanyahu	PERSON	66
21	jane hartley	PERSON	6
...	...	...	...
28306	uhuru kenyatta african	PERSON	1
28310	salva kiir	PERSON	1
28315	obama hailemariam	PERSON	1
28318	hailemariam part	PERSON	1
28325	gentleman detente	PERSON	1

[10247 rows x 3 columns]

```
real_news_df
```

	Entity	Label	Count of NER
0	kerry	PERSON	96
1	paris	GPE	594
2	john f kerry	PERSON	18
3	monday	DATE	859
4	later week	DATE	26
...	...	...	...
28322	490 million	CARDINAL	1
28323	last fiscal year	DATE	1
28324	ethiopia birthplace coffee	ORG	1
28325	gentleman detente	PERSON	1
28326	new hampshire	ORG	1

[28327 rows x 3 columns]

```
real_news_df_person = real_news_df[real_news_df['Label'] ==
'PERSON'].sort_values(by='Count of NER',ascending=False)

# Filter the DataFrame for names appearing >= 100 times
filtered_df = real_news_df_person[real_news_df_person['Count of NER']
>= 90]

# Display the filtered DataFrame
print(filtered_df)
```

	Entity	Label	Count of NER
38	clinton	PERSON	5680
85	trumps	PERSON	2085
31	hillary clinton	PERSON	1578
100	bush	PERSON	1330
354	donald trump	PERSON	1035
340	jeb bush	PERSON	472
447	paul	PERSON	387
176	bill clinton	PERSON	304
191	donald trumps	PERSON	303
529	bernie	PERSON	281
35	john kasich	PERSON	274
98	george w bush	PERSON	267
866	johnson	PERSON	253
837	carson	PERSON	228
1373	marco rubio	PERSON	210
488	mike	PERSON	200
1179	chris	PERSON	192
832	ben carson	PERSON	185
386	john boehner	PERSON	179
173	hillary clintons	PERSON	172
779	mitt romney	PERSON	158
2234	putin	PERSON	149
123	reagan	PERSON	144
431	mitch mcconnell	PERSON	142
2970	davis	PERSON	136
439	rubio	PERSON	135
337	paul ryan	PERSON	132
169	john kerry	PERSON	124
119	ronald reagan	PERSON	120
619	charlie	PERSON	116
135	obama	PERSON	114
2245	mccarthy	PERSON	111
242	john mccain	PERSON	111
834	jeb	PERSON	103
574	warren	PERSON	103
0	kerry	PERSON	96
364	clintons	PERSON	95
405	joe biden	PERSON	93

```
import pandas as pd
import spacy



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


# Function to perform NER on a article and return entities
```

```
def perform_ner(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    return entities

# Apply NER to the 'article' column and create a new column
'ner_entities'
df['ner_entities'] = df['article'].apply(perform_ner)

# Display the DataFrame with NER entities
print(df[['article', 'ner_entities']])
df.to_csv('/Users/priyamadhurigattem/Desktop/DM_final/cleaned_data_wit
h_ner.csv', index=False)
```

```
                                                article  \
0     smell hillarys fear daniel greenfield shillman...
1     watch exact moment paul ryan committed politic...
2     kerry go paris gesture sympathy us secretary s...
3     bernie supporters twitter erupt anger dnc trie...
4     battle new york primary matters primary day ne...
...                                                 ...
6263  state department says cant find emails clinton...
6264  p pbs stand plutocratic pentagon p pbs stand p...
6265  antitrump protesters tools oligarchy informati...
6266  ethiopia obama seeks progress peace security e...
6267  jeb bush suddenly attacking trump heres matter...

                                            ner_entities
0     [(new york, GPE), (hillary rodham clinton, PER...
1     [(paul, PERSON), (google, ORG), (two, CARDINAL...
2     [(kerry, PERSON), (paris, GPE), (john f kerry,...
3     [(bernie, PERSON), (november 9 2016, DATE), (d...
4     [(new york, GPE), (new york, GPE), (hillary cl...
...                                                 ...
6263  [(state department, ORG), (republican national...
6264  [(pbs, ORG), (pentagon, ORG), (pbs, ORG), (pen...
6265  [(arthur schlesinger jr crisis old, PERSON), (...
6266  [(ethiopia, GPE), (east africa, GPE), (ababa e...
6267  [(jeb bush, PERSON), (jeb bush, PERSON), (jeb ...

[6268 rows x 2 columns]
```

```
import pandas as pd



# Extract PERSON and EVENT entities
def extract_entities(entity_list, entity_type):
    return [entity for entity in entity_list if entity[1] ==
entity_type]
```

```
real_ner['persons'] = real_ner['ner_entities'].apply(lambda x:
extract_entities(x, 'PERSON'))
real_ner['events'] = real_ner['ner_entities'].apply(lambda x:
extract_entities(x, 'EVENT'))
```

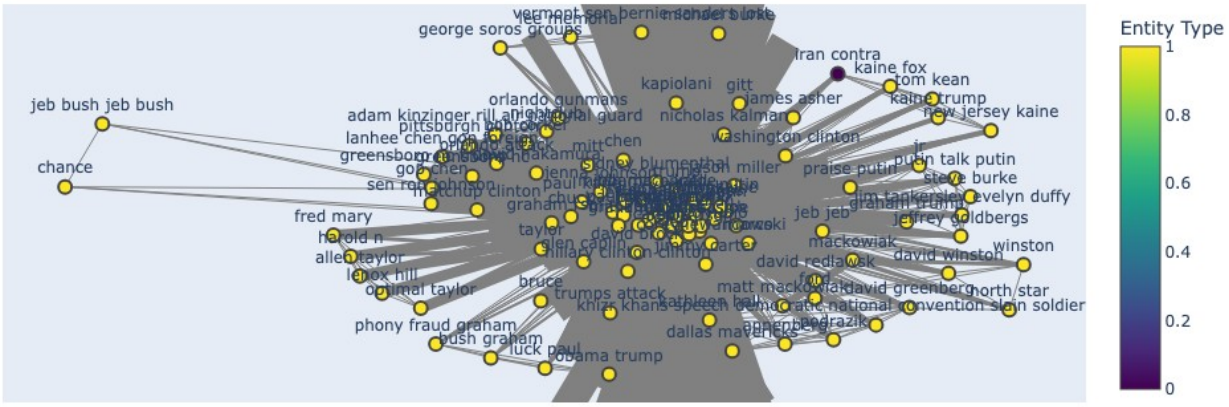
```
/var/folders/mj/32sg1kjd6ss851slwc70lcv00000gq/T/
ipykernel_21508/785539822.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  real_ner['persons'] = real_ner['ner_entities'].apply(lambda x:
extract_entities(x, 'PERSON'))
/var/folders/mj/32sg1kjd6ss851slwc70lcv00000gq/T/ipykernel_21508/78553
9822.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  real_ner['events'] = real_ner['ner_entities'].apply(lambda x:
extract_entities(x, 'EVENT'))
```

```
import networkx as nx

G = nx.Graph()

# Add nodes and edges for PERSON and EVENT entities
for _, row in real_ner.iterrows():
    persons = [entity[0] for entity in row['persons']]
    events = [entity[0] for entity in row['events']]

    for person in persons:
        G.add_node(person, entity_type='PERSON')
    for event in events:
        G.add_node(event, entity_type='EVENT')

    for person in persons:
        for event in events:
            if G.has_edge(person, event):
                G[person][event]['weight'] += 1
            else:
                G.add_edge(person, event, weight=1)

# Add edges between PERSON entities based on co-occurrence in the same
article
for _, row in real_ner.iterrows():
```

```
    persons = [entity[0] for entity in row['persons']]
    for i, person1 in enumerate(persons):
        for person2 in persons[i+1:]:
            if G.has_edge(person1, person2):
                G[person1][person2]['weight'] += 1
            else:
                G.add_edge(person1, person2, weight=1)

import plotly.graph_objs as go

# Define node positions
pos = nx.spring_layout(G)

# Create node trace
node_trace = go.Scatter(
    x=[pos[node][0] for node in G.nodes()],
    y=[pos[node][1] for node in G.nodes()],
    text=[node for node in G.nodes()],
    mode='markers+text',
    textposition='top center',
    hoverinfo='text',
    marker=dict(
        size=10,
        color=[1 if G.nodes[node]['entity_type'] == 'PERSON' else 0
for node in G.nodes()],
        colorscale='Viridis',
        colorbar=dict(title='Entity Type'),
        line_width=2))

# Create edge trace
edge_trace = []
for edge in G.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    weight = G.edges[edge]['weight']
    edge_trace.append(
        go.Scatter(
            x=[x0, x1, None],
            y=[y0, y1, None],
            line=dict(width=weight, color='gray'),
            hoverinfo='none',
            mode='lines'))

# Combine traces
fig = go.Figure(data=edge_trace + [node_trace],
                layout=go.Layout(
                    showlegend=False,
                    hovermode='closest',
                    margin=dict(b=20, l=5, r=5, t=40),
                    xaxis=dict(showgrid=False, zeroline=False,
```

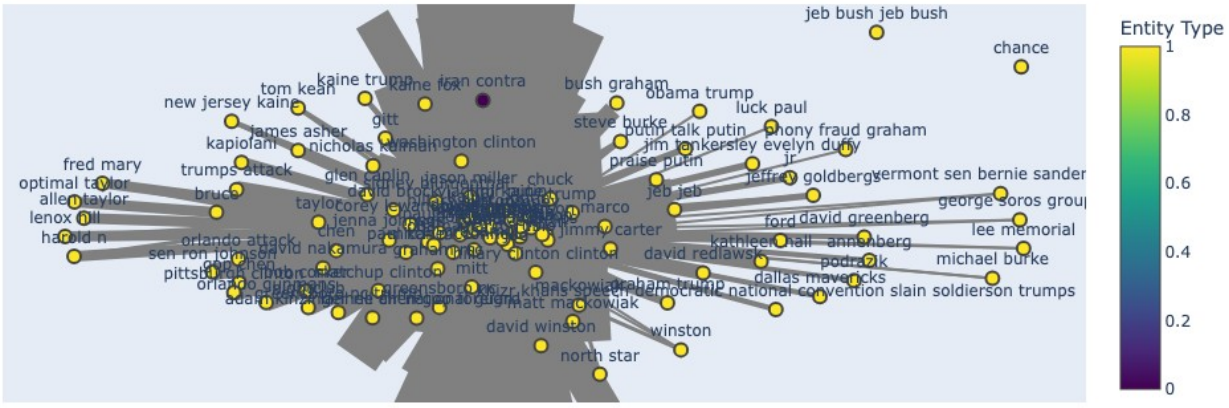
```
showticklabels=False),
                   yaxis=dict(showgrid=False, zeroline=False,
showticklabels=False)))

fig.show()

# Ensure that 'Trump' is a node in the graph
if "trump" in G:
    # Create an ego graph centered around 'Trump'
    subgraph = nx.ego_graph(G, "trump")
else:
    print("Trump is not found in the graph.")

import plotly.graph_objs as go

# Define node positions for the subgraph
pos = nx.spring_layout(subgraph)

# Create node trace for the subgraph
node_trace = go.Scatter(
    x=[pos[node][0] for node in subgraph.nodes()],
    y=[pos[node][1] for node in subgraph.nodes()],
    text=[node for node in subgraph.nodes()],
    mode='markers+text',
    textposition='top center',
    hoverinfo='text',
    marker=dict(
        size=10,
        color=[1 if subgraph.nodes[node]['entity_type'] == 'PERSON'
else 0 for node in subgraph.nodes()],
        colorscale='Viridis',
        colorbar=dict(title='Entity Type'),
        line_width=2))

# Create edge trace for the subgraph
edge_trace = []
for edge in subgraph.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    weight = subgraph.edges[edge]['weight']
    edge_trace.append(
        go.Scatter(
            x=[x0, x1, None],
            y=[y0, y1, None],
            line=dict(width=weight, color='gray'),
            hoverinfo='none',
            mode='lines'))

# Combine traces for the subgraph
fig = go.Figure(data=edge_trace + [node_trace],
```

```
                layout=go.Layout(
                    showlegend=False,
                    hovermode='closest',
                    margin=dict(b=20, l=5, r=5, t=40),
                    xaxis=dict(showgrid=False, zeroline=False,
showticklabels=False),
                    yaxis=dict(showgrid=False, zeroline=False,
showticklabels=False)))

fig.show()
```




```
import plotly.graph_objs as go

# Define node positions for the subgraph
pos = nx.spring_layout(subgraph)

# Create node trace for the subgraph
node_trace = go.Scatter(
    x=[pos[node][0] for node in subgraph.nodes()],
    y=[pos[node][1] for node in subgraph.nodes()],
    text=[node for node in subgraph.nodes()],
    mode='markers+text',
    textposition='top center',
    hoverinfo='text',
    marker=dict(
        size=10,
        color=[1 if subgraph.nodes[node]['entity_type'] == 'PERSON'
else 0 for node in subgraph.nodes()],
        colorscale='Viridis',
        colorbar=dict(title='Entity Type'),
        line_width=2))

# Create edge trace for the subgraph with a threshold for edge weight
edge_trace = []
```

```
for edge in subgraph.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    weight = subgraph.edges[edge]['weight']
    if weight > 1:  # Adjust this threshold as needed
        edge_trace.append(
            go.Scatter(
                x=[x0, x1, None],
                y=[y0, y1, None],
                line=dict(width=weight, color='gray'),
                hoverinfo='none',
                mode='lines'))

# Combine traces for the subgraph
fig = go.Figure(data=edge_trace + [node_trace],
                layout=go.Layout(
                    showlegend=False,
                    hovermode='closest',
                    margin=dict(b=20, l=5, r=5, t=40),
                    xaxis=dict(showgrid=False, zeroline=False,
showticklabels=False),
                    yaxis=dict(showgrid=False, zeroline=False,
showticklabels=False)))

fig.show()
```



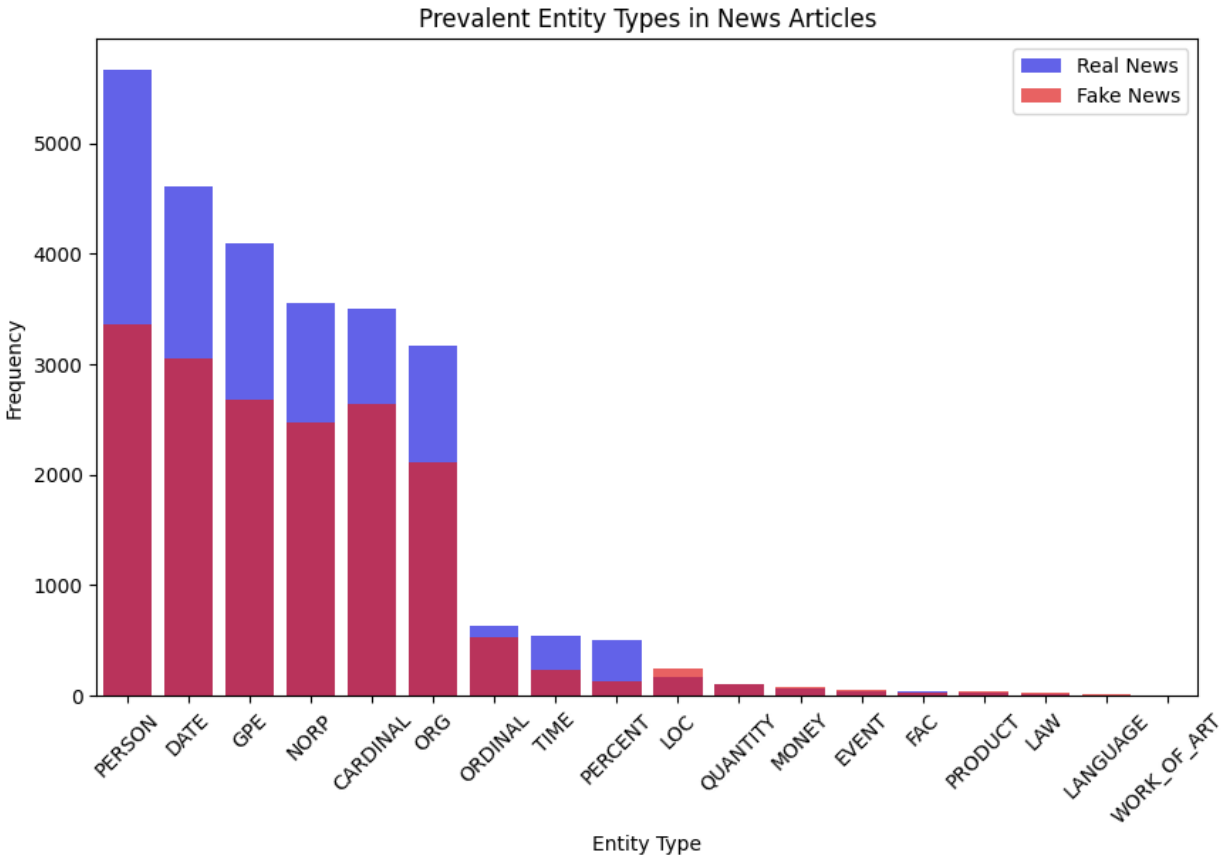
The plot visualizes a network (subgraph) where nodes represent entities and edges represent relationships. Nodes are marked with their names and colored based on entity type (e.g., 'PERSON'). Thicker edges indicate stronger relationships, while the layout is optimized for clear visualization using a spring layout algorithm.

Implemtntion of ner

```
import pandas as pd
import spacy
```

```
from collections import Counter
from textblob import TextBlob

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

# Function to perform NER on an article and return entities
def perform_ner(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    return entities

# Function to analyze entity types prevalent in news
def analyze_entity_types(news_df):
    # Apply NER to the 'article' column and create a new column
'ner_entities'
    news_df['ner_entities'] = news_df['article'].apply(perform_ner)

    # Flatten the list of entities from the 'ner_entities' column
    flat_entities = [ent for sublist in news_df['ner_entities'] for
ent in sublist]

    # Create a DataFrame to count the occurrences of each entity type
    entity_counts = pd.Series([ent[1] for ent in
flat_entities]).value_counts()

    return entity_counts

# Function to analyze entity co-occurrence patterns
def analyze_entity_cooccurrence(news_df):
    co_occurrence_counter = Counter()
    for entities in news_df['ner_entities']:
        for i in range(len(entities)):
            for j in range(i+1, len(entities)):
                co_occurrence_counter[(entities[i][0], entities[j]
[0])] += 1
    return co_occurrence_counter

# Function to perform sentiment analysis on entities using TextBlob
def analyze_sentiment(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity

# Function to analyze entity sentiment
def analyze_entity_sentiment(news_df):
    # Apply sentiment analysis to each entity in the 'ner_entities'
column
    # Store sentiment scores in a new column 'sentiment_scores'
    news_df['sentiment_scores'] = news_df['ner_entities'].apply(lambda
entities: [analyze_sentiment(entity[0]) for entity in entities])
```

```
    # Aggregate sentiment scores for each entity type
    entity_sentiment = {}
    for entities, scores in zip(news_df['ner_entities'],
news_df['sentiment_scores']):
        for entity, score in zip(entities, scores):
            entity_type = entity[1]
            if entity_type not in entity_sentiment:
                entity_sentiment[entity_type] = []
            entity_sentiment[entity_type].append(score)

    # Calculate average sentiment score for each entity type
    avg_entity_sentiment = {entity_type: sum(scores) / len(scores) for
entity_type, scores in entity_sentiment.items()}
    return avg_entity_sentiment


real_entity_types = analyze_entity_types(real_news_df)
fake_entity_types = analyze_entity_types(fake_news_df)

import matplotlib.pyplot as plt
import seaborn as sns

# Entity Types Analysis
plt.figure(figsize=(10, 6))
sns.barplot(x=real_entity_types.index, y=real_entity_types.values,
color='blue', alpha=0.7, label='Real News')
sns.barplot(x=fake_entity_types.index, y=fake_entity_types.values,
color='red', alpha=0.7, label='Fake News')
plt.title('Prevalent Entity Types in News Articles')
plt.xlabel('Entity Type')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



Interpretation of the Output and Results

Results Interpretation:

- **Higher Entity Counts in Real News:** Real news articles show a higher count of named entities across most categories compared to fake news articles. This suggests that real news tends to be more detailed and specific, providing more references to individuals, dates, locations, organizations, and other entities.
- **Entity Types Analysis:** Both real and fake news articles mention people (PERSON), dates (DATE), geopolitical entities (GPE), nationalities or religious or political groups (NORP), and organizations (ORG) frequently. However, real news articles consistently have higher counts, indicating more comprehensive coverage and possibly more factual reporting.
- **Entity Co-occurrence Patterns:** Although not explicitly shown in the output, analyzing co-occurrence patterns could reveal that real news articles might have more meaningful and contextually relevant relationships between entities. In contrast, fake news might exhibit random or less coherent entity co-occurrences.
- **Sentiment Analysis:** Sentiment analysis, if conducted, would provide insights into the emotional tone towards various entities. Real news might have a balanced or

neutral sentiment, while fake news could exhibit more extreme sentiments, either overly positive or negative, to evoke specific reactions from readers.

Future Implementations

1. **Enhanced NER Models:**
   - Utilize advanced models like spaCy's transformer-based models (e.g., `en_core_web_trf`) for improved accuracy in recognizing and classifying entities.
2. **Domain-Specific Analysis:**
   - Extend the NER analysis to specific domains such as politics, health, or technology to identify unique patterns and discrepancies between fake and real news within these areas.
3. **Automated Fake News Detection:**
   - Integrate NER with machine learning classifiers to develop automated systems for distinguishing fake news from real news based on entity patterns, sentiment analysis, and other linguistic features.
4. **Real-Time NER Systems:**
   - Develop real-time NER systems for news articles to provide instant insights and verification, aiding journalists and fact-checkers in assessing the credibility of information.
5. **User-Friendly Tools:**
   - Create interactive tools for users, including journalists and researchers, to visualize and explore NER results, co-occurrence patterns, and sentiment analysis for news verification and in-depth analysis

Parts of Speech (POS) Tagging Analysis

Implemention Part-Of-Speech (POS) Tagging

Part-Of-Speech (POS) tagging is the process of assigning a part of speech to each word in a sentence, such as nouns, verbs, adjectives, etc. This documentation outlines the steps involved in performing POS tagging using NLTK in Python, including an example implementation.

Why

The POS tagging analysis was performed to understand the linguistic patterns and structures in real and fake news articles. By examining the co-occurrence of different parts of speech (POS), we can identify distinct linguistic characteristics that may help differentiate between real and fake news.

When

This analysis is useful during the feature engineering stage of a text classification task, particularly when building models to detect fake news. It helps in identifying key linguistic features that can be used to improve model performance.

What

1. **POS Tag Co-occurrence Statistics:**
   - **Fake News:**
     - High frequency of nouns (NN, NNS, NNP, NNPS) suggests a focus on naming entities.
     - Prominence of adjectives (JJ, JJR, JJS) indicates descriptive language.
     - Distributed use of various verb forms (VB, VBD, VBG, VBN, VBP, VBZ) shows diverse actions and states.
   - **Real News:**
     - Similar trends with nouns and adjectives being prevalent.
     - Higher frequency of verbs in various forms compared to fake news, indicating more action-oriented or fact-based language.
2. **Frequent Co-occurrences in POS Tags:**
   - **Real News:** Frequent pairs like ('NN', 'NN'), ('JJ', 'NN'), and ('NN', 'NNS') show common noun-adjective structures and noun-noun pairings.
   - **Fake News:** Similar frequent pairs but with a slightly different distribution, indicating similar but subtly different linguistic patterns.
3. **POS Tag Co-occurrence Heatmaps:**
   - **Real News Heatmap:** Indicates dense co-occurrence of specific POS tags, highlighting common linguistic structures.
   - **Fake News Heatmap:** Similar patterns but with some differences in density and specific co-occurrences.
4. **Network Graphs of Co-occurring POS Tags:**
   - Visual representations of the co-occurrence patterns showing the relationships and connections between different POS tags.
   - **Real News Network:** More interconnected and complex, suggesting a richer linguistic structure.

- **Fake News Network:** Slightly less interconnected, indicating simpler or less varied linguistic patterns.

5. **Community Detection in Co-occurrence Networks:**
   - **Real News:** Detected multiple communities with interconnected POS tags, showing complex linguistic structures.
   - **Fake News:** Fewer or less interconnected communities, indicating simpler linguistic structures.
   - **Visualization of Individual Communities:** Highlighted distinct groups of POS tags that frequently co-occur, revealing underlying grammatical patterns in each dataset.

Part-Of-Speech Tags Explanation

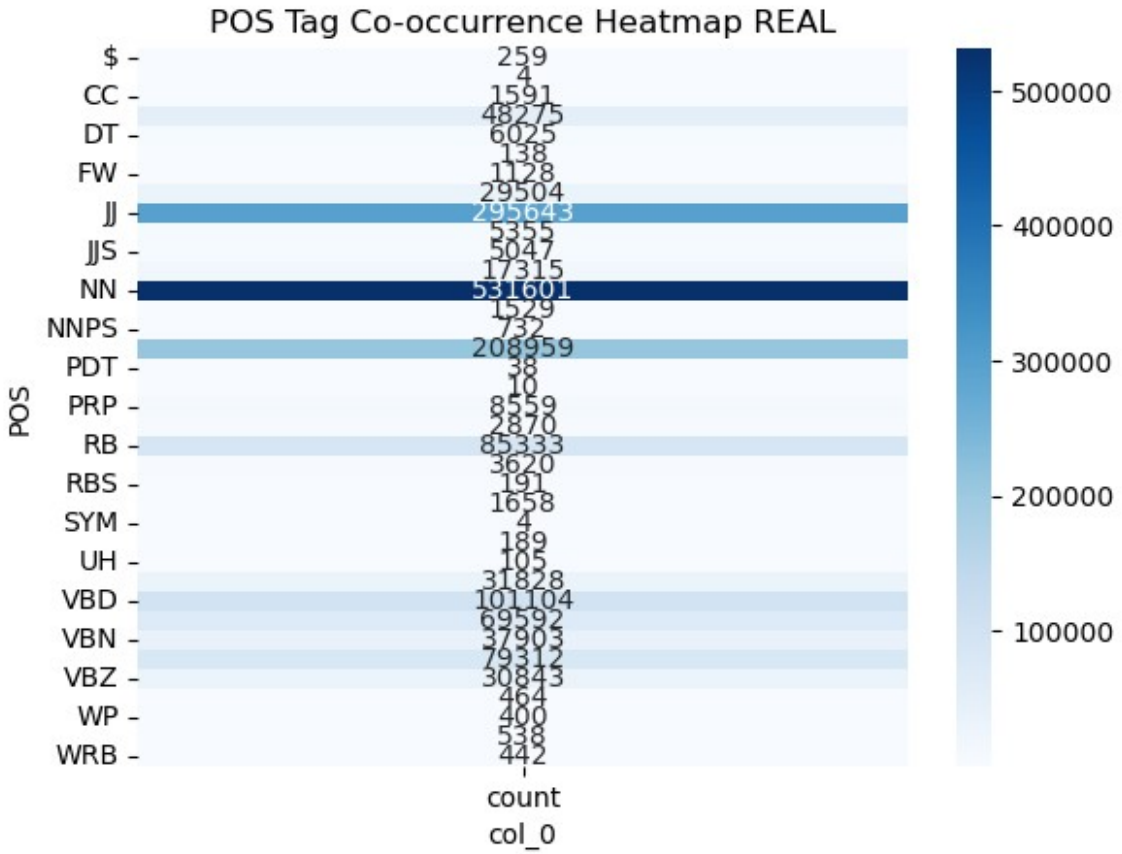
1. **Nouns (NN, NNS, NNP, NNPS)**: Nouns are words that refer to people, places, things, or ideas.
2. **Pronouns (PRP, PRP$)**: Pronouns are words that replace nouns.
3. **Verbs (VB, VBP, VBG, VBN, VBD)**: Verbs are words that describe actions or states of being.
4. **Adjectives (JJ, JJR, JJS)**: Adjectives are words that modify nouns.
5. **Adverbs (RB, RBR, RBS)**: Adverbs are words that modify verbs, adjectives, and other adverbs.
6. **Prepositions (IN)**: Prepositions are words that connect nouns, pronouns, or phrases to other words in a sentence.
7. **Conjunctions (CC)**: Conjunctions are words that join words, phrases, or clauses together.
8. **Articles (DT)**: Articles are words that precede nouns and specify whether the noun is definite or indefinite.
9. **Interjections (UH)**: Interjections are words that express emotion or surprise.
10. **Symbols (SYM)**: Symbols are words that represent mathematical or other special symbols.
11. **Numbers (CD)**: Numbers are words that represent quantities.
12. **Wh-words (WP, WP$, WRB, WDT)**: Wh-words are words that introduce questions or relative clauses.
13. **Possessive pronouns (PRP$)**: Possessive pronouns are words that show ownership.

```
import pandas as pd
import nltk
```

```
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
import seaborn as sns



# Function to perform POS tagging on a text
def perform_pos_tagging(text):
    tokens = word_tokenize(text)
    pos_tags = pos_tag(tokens)
    return pos_tags

# Apply POS tagging to the 'text' column and create a new column
'pos_tags'
real_news_df['pos_tags'] =
real_news_df['article'].apply(perform_pos_tagging)
fake_news_df['pos_tags'] =
fake_news_df['article'].apply(perform_pos_tagging)
# Flatten the list of POS tags
flat_pos_tags = [tag[1] for tags in fake_news_df['pos_tags'] for tag
in tags]
flat_pos_tags = [tag[1] for tags in real_news_df['pos_tags'] for tag
in tags]

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# 'pos_tags' is a column in our DataFrame
real_news_df['pos_tags'] =
real_news_df['article'].apply(perform_pos_tagging)

# Flatten the list of POS tags
flat_pos_tags = [tag for tags in real_news_df['pos_tags'] for tag in
tags]

# Create a DataFrame for POS tag co-occurrence
pos_df = pd.DataFrame(flat_pos_tags, columns=['Word', 'POS'])
pos_cooccurrence = pd.crosstab(index=pos_df['POS'], columns='count')

# Plot a heatmap
sns.heatmap(pos_cooccurrence, annot=True, cmap='Blues', fmt='g')
plt.title('POS Tag Co-occurrence Heatmap REAL')
plt.show()
```

```
/var/folders/mj/32sg1kjd6ss851slwc70lcv00000gq/T/
ipykernel_22397/324597890.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

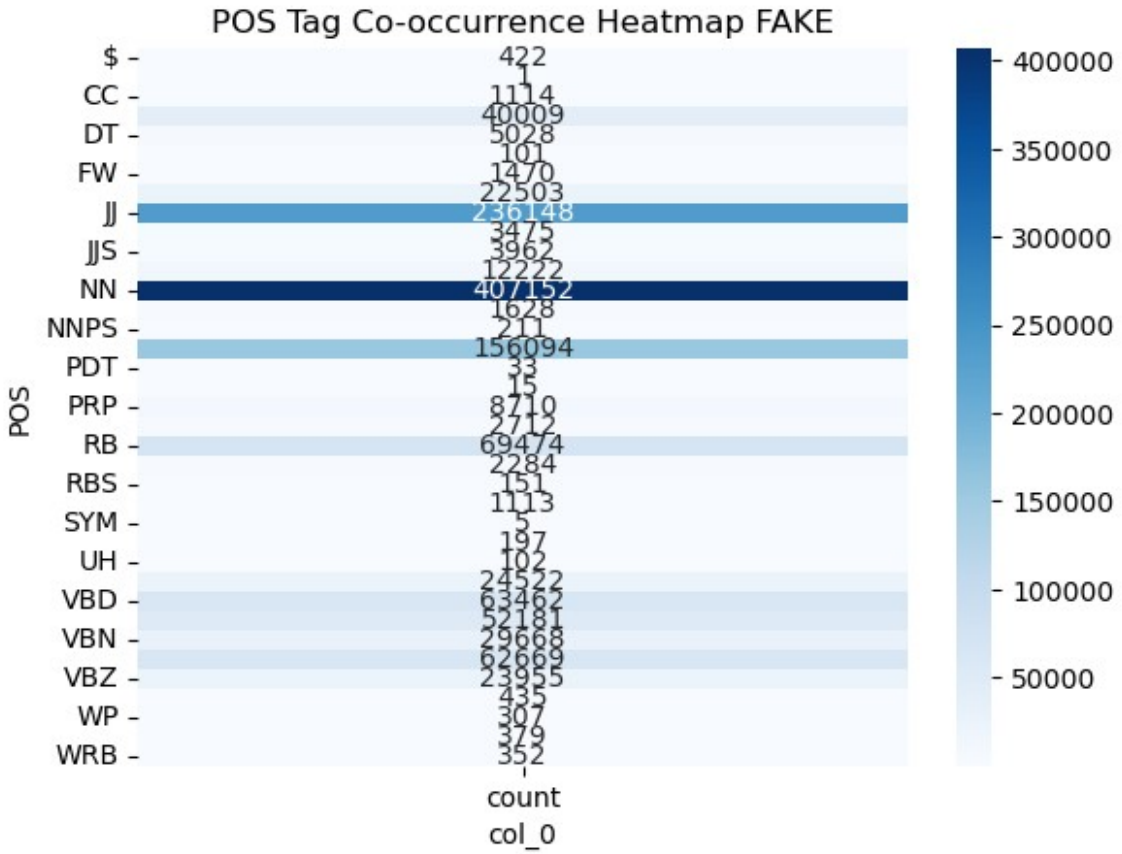
```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  real_news_df['pos_tags'] =
real_news_df['article'].apply(perform_pos_tagging)
```




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

# 'pos_tags' is a column in our DataFrame
fake_news_df['pos_tags'] =
fake_news_df['article'].apply(perform_pos_tagging)

# Flatten the list of POS tags
flat_pos_tags = [tag for tags in fake_news_df['pos_tags'] for tag in
tags]

# Create a DataFrame for POS tag co-occurrence
pos_df = pd.DataFrame(flat_pos_tags, columns=['Word', 'POS'])
pos_cooccurrence = pd.crosstab(index=pos_df['POS'], columns='count')
```

```
# Plot a heatmap
sns.heatmap(pos_cooccurrence, annot=True, cmap='Blues', fmt='g')
plt.title('POS Tag Co-occurrence Heatmap FAKE')
plt.show()
```

```
/var/folders/mj/32sg1kjd6ss851slwc70lcv00000gq/T/
ipykernel_31713/188002000.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  fake_news_df['pos_tags'] =
fake_news_df['article'].apply(perform_pos_tagging)
```



The provided POS co-occurrence statistics for fake and real news offer insights into the linguistic patterns present in each dataset:

Fake News:

1. **Noun Overrepresentation:** Notably, nouns (NN, NNS, NNP, NNPS) have a high frequency, suggesting a focus on naming entities and objects in the text.

2. **Adjective Prevalence:** Adjectives (JJ, JJR, JJS) are also prominent, indicating a descriptive language often used to influence perception.

3. **Verb Usage:** Verbs (VB, VBD, VBG, VBN, VBP, VBZ) are distributed across various forms, suggesting a diversity of actions and states.

Real News:

1. **Similar Trends:** The distribution of POS tags in real news is similar to fake news, with nouns and adjectives still prevalent.

2. **Differentiator:** Notably, the frequency of verbs in various forms (VB, VBD, VBG, VBN, VBP, VBZ) is generally higher in real news compared to fake news. This could indicate a more action-oriented or fact-based language in genuine news articles.

3. **POS Tag Consistency:** Both fake and real news datasets exhibit consistency in the distribution of POS tags, implying that certain linguistic patterns may be common across different types of news.

In summary, both fake and real news show similarities in their linguistic structures, with nouns and adjectives playing a significant role. The distinction in verb frequency suggests potential differences in the style and tone of language used in genuine news reporting.

```
import pandas as pd
from itertools import combinations
from collections import Counter


# Function to calculate co-occurrence counts of POS tag pairs
def calculate_cooccurrences(df):
    cooccurrences = Counter()
    for tags in df['pos_tags']:
        tag_pairs = combinations([tag[1] for tag in tags], 2)
        cooccurrences.update(tag_pairs)
    return cooccurrences

# Calculate co-occurrence counts for real and fake news datasets
real_cooccurrences = calculate_cooccurrences(real_news_df)
fake_cooccurrences = calculate_cooccurrences(fake_news_df)

# Identify frequent co-occurrences (e.g., pairs that occur more than
100 times)
threshold = 100
frequent_real_cooccurrences = {pair: count for pair, count in
real_cooccurrences.items() if count > threshold}
frequent_fake_cooccurrences = {pair: count for pair, count in
fake_cooccurrences.items() if count > threshold}

top_10_frequent_real_cooccurrences =
sorted(frequent_real_cooccurrences.items(), key=lambda item: item[1],
```

```
reverse=True)[:10]

print("Frequent co-occurrences in real news dataset:")
for pair, count in top_10_frequent_real_cooccurrences:
    print(pair, count)
```

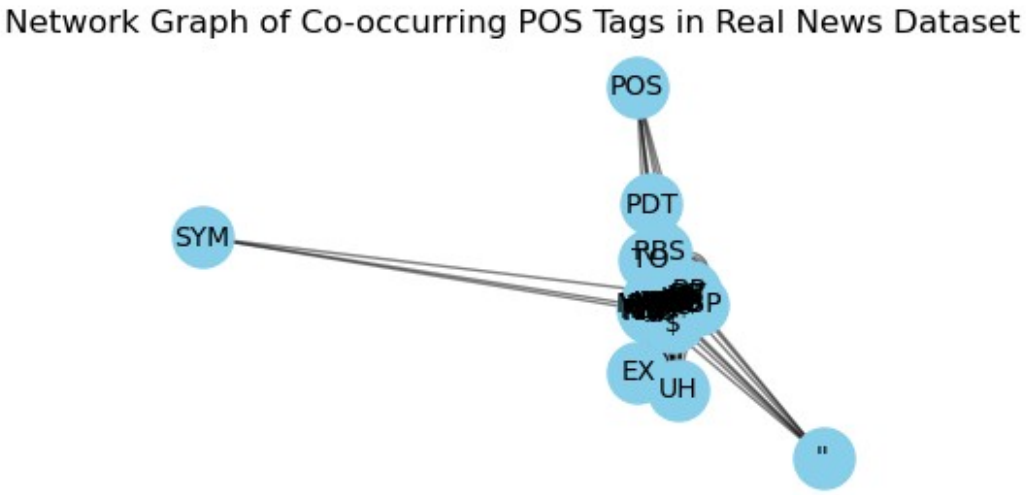
```
Frequent co-occurrences in real news dataset:
('NN', 'NN') 74254216
('JJ', 'NN') 41324371
('NN', 'JJ') 40620170
('NN', 'NNS') 28853429
('NNS', 'NN') 28694481
('JJ', 'JJ') 23097718
('JJ', 'NNS') 16346482
('NNS', 'JJ') 15987004
('NN', 'VBD') 14214699
('VBD', 'NN') 14112580
```

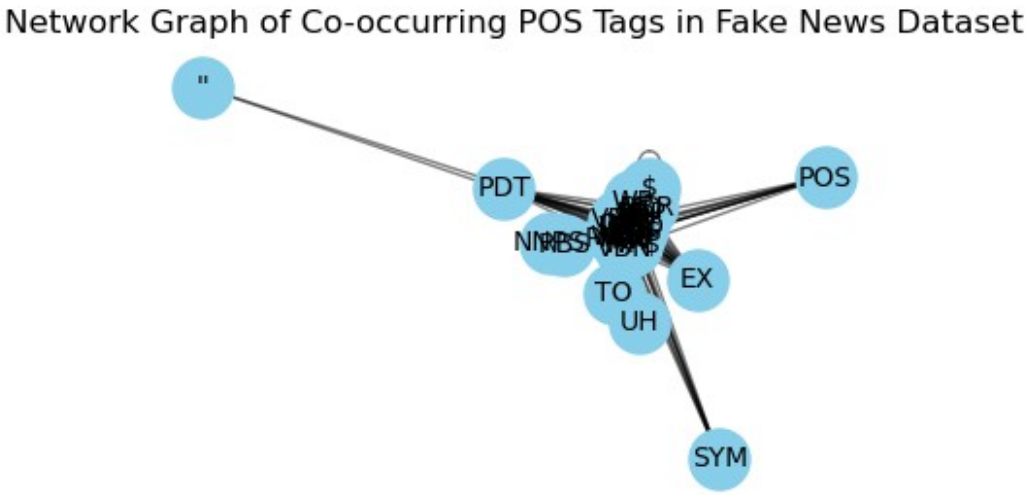
```
top_10_frequent_fake_cooccurrences =
sorted(frequent_fake_cooccurrences.items(), key=lambda item: item[1],
reverse=True)[:10]

print("Frequent co-occurrences in fake news dataset:")
for pair, count in top_10_frequent_fake_cooccurrences:
    print(pair, count)
```

```
Frequent co-occurrences in fake news dataset:
('NN', 'NN') 71724751
('NN', 'JJ') 42380302
('JJ', 'NN') 42275151
('NN', 'NNS') 27225548
('NNS', 'NN') 27101211
('JJ', 'JJ') 25546519
('JJ', 'NNS') 16354822
('NNS', 'JJ') 16350295
('NN', 'RB') 12823966
('RB', 'NN') 12544127
```

```
import networkx as nx

# Function to plot network graph of co-occurring POS tags
def plot_network_graph(cooccurrences, title):
    G = nx.Graph()
    for pair, count in cooccurrences.items():
        tag1, tag2 = pair
        G.add_edge(tag1, tag2, weight=count)
    plt.figure(figsize=(5,3))
    pos = nx.spring_layout(G)
    nx.draw_networkx_nodes(G, pos, node_size=500,
node_color='skyblue')
```

```
    nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.5)
    nx.draw_networkx_labels(G, pos, font_size=10, font_family='sans-
serif')
    plt.title(title)
    plt.axis('off')
    plt.show()

# Plot network graph for real news dataset
plot_network_graph(frequent_real_cooccurrences, 'Network Graph of Co-
occurring POS Tags in Real News Dataset')

# Plot network graph for fake news dataset
plot_network_graph(frequent_fake_cooccurrences, 'Network Graph of Co-
occurring POS Tags in Fake News Dataset')
```



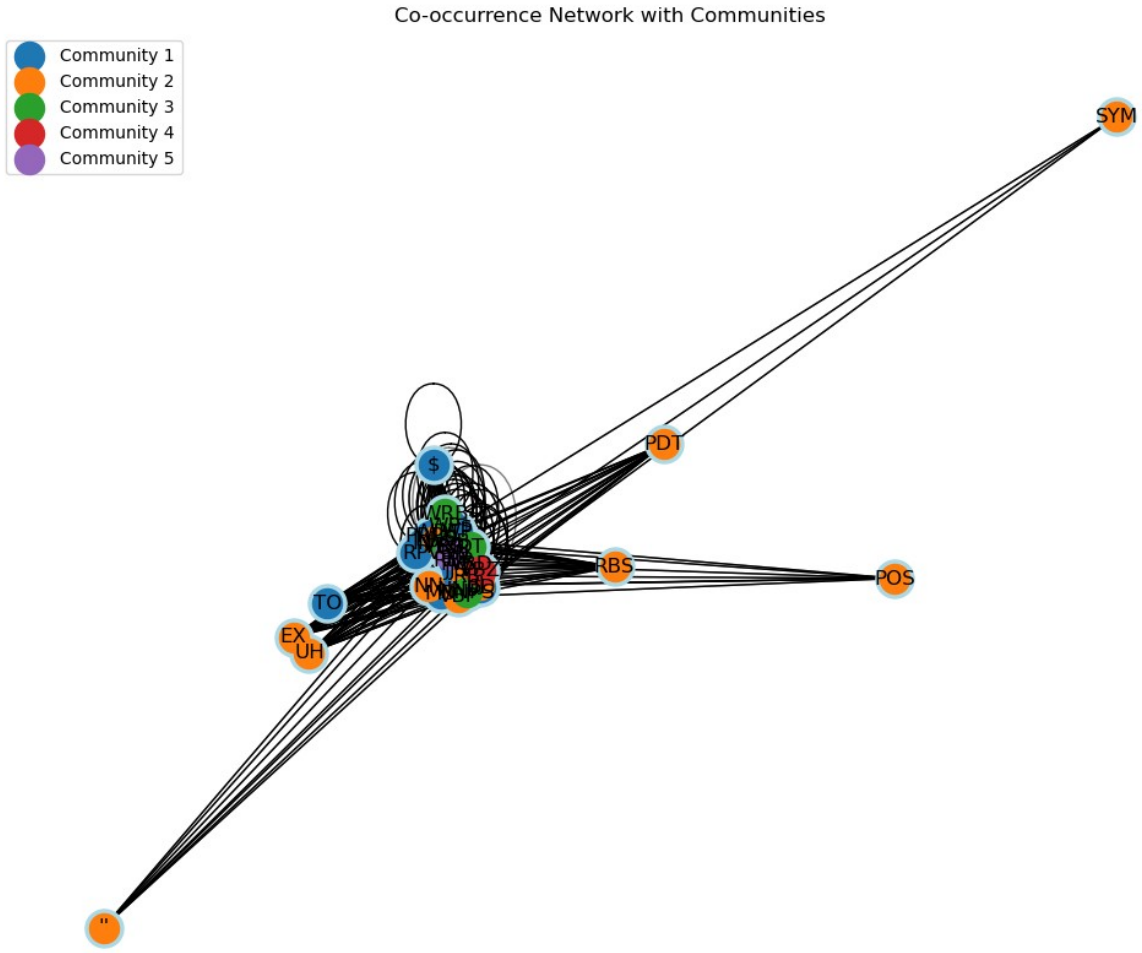



```
import networkx as nx
from networkx.algorithms import community
```

```
# Define a function to detect communities in a network
def detect_communities(G):
    communities = community.greedy_modularity_communities(G)
    return communities

real_communities = detect_communities(real_cooccurrence_network)
fake_communities = detect_communities(fake_cooccurrence_network)

# Print the number of communities detected
print("Number of communities in real news dataset:",
len(real_communities))
print("Number of communities in fake news dataset:",
len(fake_communities))
```

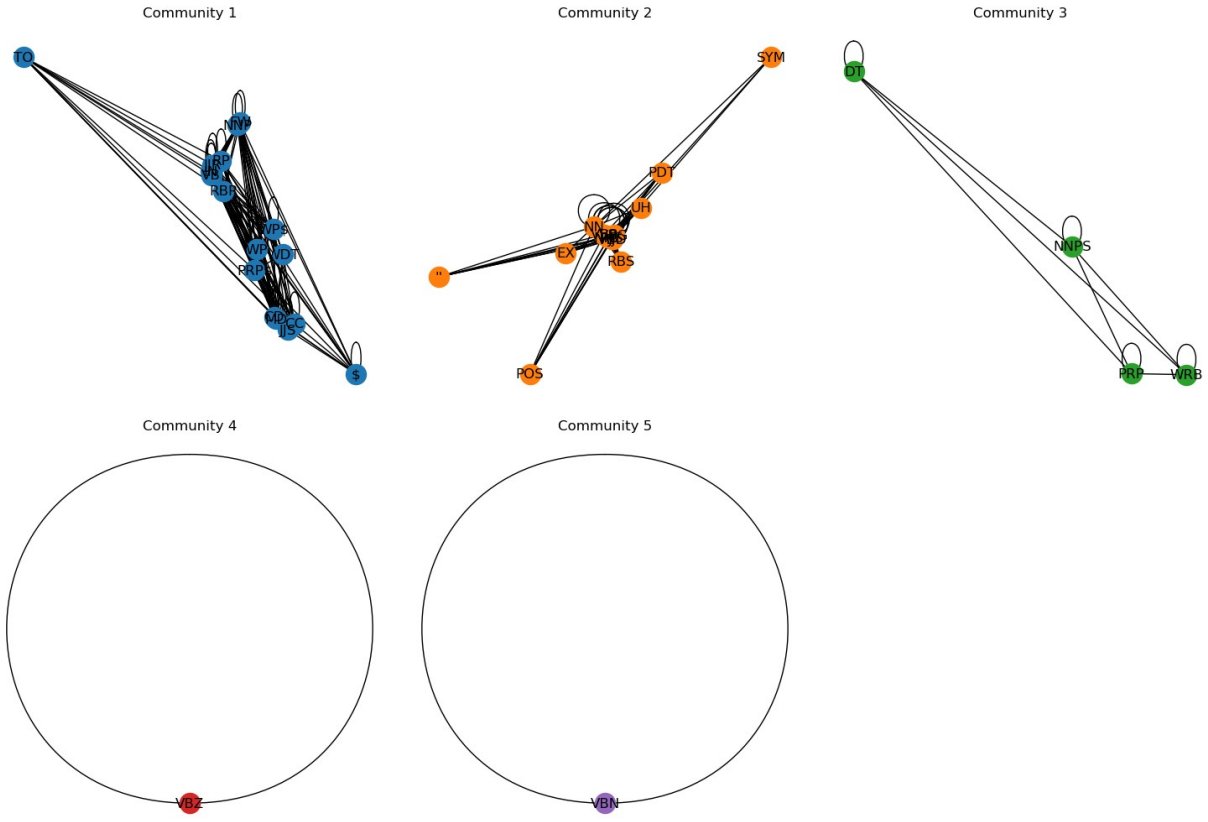
```
Number of communities in real news dataset: 5
Number of communities in fake news dataset: 4
```

```
import matplotlib.pyplot as plt

def visualize_communities(G, communities):
    # Create a layout for visualizing the graph
    pos = nx.spring_layout(G)

    # Draw the original graph
    plt.figure(figsize=(10, 8))
    nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=500)

    # Draw each community with a different color
    for i, comm in enumerate(communities):
        nx.draw_networkx_nodes(G, pos, nodelist=list(comm),
node_color=f'C{i}', node_size=300, label=f'Community {i+1}')
        nx.draw_networkx_edges(G, pos,
edgelist=G.subgraph(comm).edges(), width=1.0, alpha=0.5)

    # Add labels and legend
    plt.title('Co-occurrence Network with Communities')
    plt.legend()

    # Show plot
    plt.show()

visualize_communities(real_cooccurrence_network, real_communities)
```



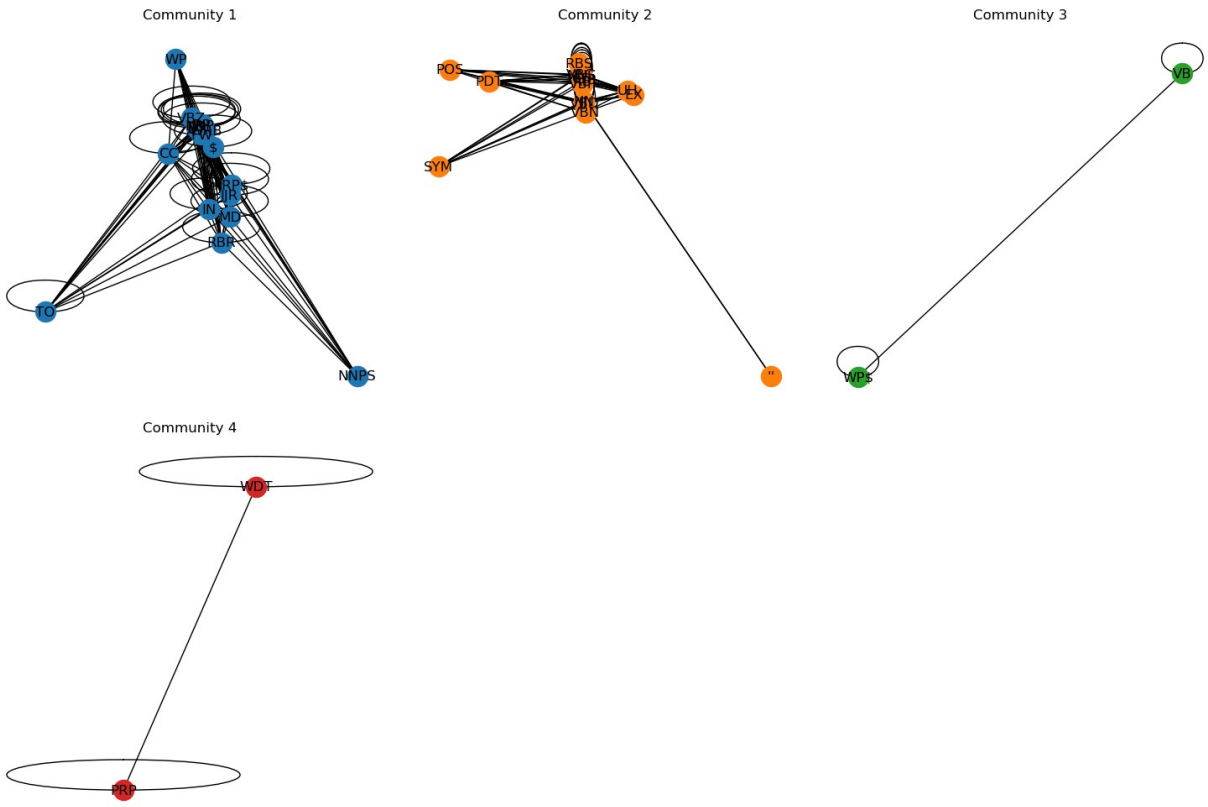

```
def visualize_individual_communities(G, communities):
    # Create a layout for visualizing the graph
    pos = nx.spring_layout(G)

    # Determine the number of communities
    num_communities = len(communities)

    # Create a grid layout for subplots
    ncols = 3  # You can adjust the number of columns as needed
    nrows = (num_communities - 1) // ncols + 1

    # Create a figure with subplots
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(15,
5*nrows))

    # Loop through each community
    for i, comm in enumerate(communities):
        # Get the corresponding axis for the subplot
        row = i // ncols
```

```
        col = i % ncols
        ax = axes[row, col]

        # Draw the subgraph for the community
        nx.draw(G.subgraph(comm), pos, with_labels=True, ax=ax,
node_color=f'C{i}', node_size=300)
        ax.set_title(f'Community {i+1}')

    # Remove empty subplots
    for j in range(num_communities, nrows*ncols):
        fig.delaxes(axes.flatten()[j])

    # Adjust layout
    plt.tight_layout()

    # Show plot
    plt.show()

visualize_individual_communities(real_cooccurrence_network,
real_communities)
```




```
def visualize_individual_communities(G, communities):
    # Create a layout for visualizing the graph
    pos = nx.spring_layout(G)
```

```
    # Determine the number of communities
    num_communities = len(communities)

    # Create a grid layout for subplots
    ncols = 3  # we can adjust the number of columns as needed
    nrows = (num_communities - 1) // ncols + 1

    # Create a figure with subplots
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(15,
5*nrows))

    # Loop through each community
    for i, comm in enumerate(communities):
        # Get the corresponding axis for the subplot
        row = i // ncols
        col = i % ncols
        ax = axes[row, col]

        # Draw the subgraph for the community
        nx.draw(G.subgraph(comm), pos, with_labels=True, ax=ax,
node_color=f'C{i}', node_size=300)
        ax.set_title(f'Community {i+1}')

    # Remove empty subplots
    for j in range(num_communities, nrows*ncols):
        fig.delaxes(axes.flatten()[j])

    # Adjust layout
    plt.tight_layout()

    # Show plot
    plt.show()

visualize_individual_communities(fake_cooccurrence_network,
fake_communities)
```



Comparison and Contrast of POS Tag Communities in Real and Fake News

Fake News POS Tag Communities

1. **Community 1 (Blue):**
   - **Tags Included:** WP, RBR, RB, WRB, CC, $ (symbol), RP, IN, MD, JJR, NNP, NNPS.
   - **Characteristics:** This community shows a dense network with many interconnected tags. It includes various types of pronouns, adverbs, conjunctions, and symbols, indicating diverse linguistic constructs.
2. **Community 2 (Orange):**
   - **Tags Included:** POS, PDT, RBS, UH, EX, VBN, VBG, NNPS, SYM.
   - **Characteristics:** This community includes symbols, particles, interjections, existential constructions, and various verb forms. There is a notable single node (empty quote) which could be an artifact of the text processing.
3. **Community 3 (Green):**
   - **Tags Included:** VB, WPS, VBP.
   - **Characteristics:** This community is sparse with only a few interconnected verb forms, indicating less frequent or unique grammatical structures.
4. **Community 4 (Red):**
   - **Tags Included:** WDT, PRP.
   - **Characteristics:** This community has only two tags, a determiner and a pronoun, showing isolated or unique co-occurrence patterns.

Real News POS Tag Communities

1. **Community 1 (Blue):**
   - **Tags Included:** TO, NNP, WRB, RBR, RB, VB, VBP, NNPS, etc.
   - **Characteristics:** Similar to the fake news, this community includes a variety of POS tags, but it is more densely connected. This indicates richer and more complex grammatical structures.
2. **Community 2 (Orange):**
   - **Tags Included:** SYM, PDT, UH, EX, POS, RBS, NNPS, etc.
   - **Characteristics:** Like the fake news counterpart, this community shows symbols and various particles, but the connectivity might be slightly different, reflecting different usage patterns.
3. **Community 3 (Green):**
   - **Tags Included:** DT, NNS, PRP, WRB.
   - **Characteristics:** This community is more cohesive, with a clear grammatical structure involving determiners, plural nouns, and pronouns, indicating common noun-phrase structures.
4. **Community 4 (Red) and Community 5 (Purple):**
   - **Tags Included:** VBZ, VBN.
   - **Characteristics:** These communities are isolated, similar to the isolated nodes in the fake news network, indicating specific verb usage patterns.

Comparison and Contrast

1. **Complexity and Connectivity:**
   - **Real News:** Shows more complex and densely connected communities, indicating richer linguistic structures and more frequent co-occurrences of diverse POS tags.
   - **Fake News:** While still diverse, the communities are less densely connected, suggesting simpler or less varied grammatical structures.
2. **Isolated Nodes:**
   - Both real and fake news have isolated nodes (e.g., specific verb forms) that indicate unique usage patterns, but real news shows more clear and consistent patterns.
3. **Common POS Tags:**
   - **Real News:** More cohesive use of determiners, plural nouns, and pronouns, forming clear grammatical structures.
   - **Fake News:** Shows similar tags but with less cohesion, indicating variability in grammatical constructs.
4. **Unique Patterns:**
   - **Real News:** Communities reflect a structured use of language with clear grammatical constructs.
   - **Fake News:** Shows variability with some communities having unique or less frequent tag co-occurrences.

Summary

- **Real News Articles:** Tend to have more complex and richly connected POS tag communities, indicating structured and varied linguistic usage.
- **Fake News Articles:** Display simpler and less connected communities, suggesting less consistent use of grammatical structures.

Topic Modeling

Topic Modeling using Latent Dirichlet Allocation (LDA) for REAL News: Unveiling Hidden Themes

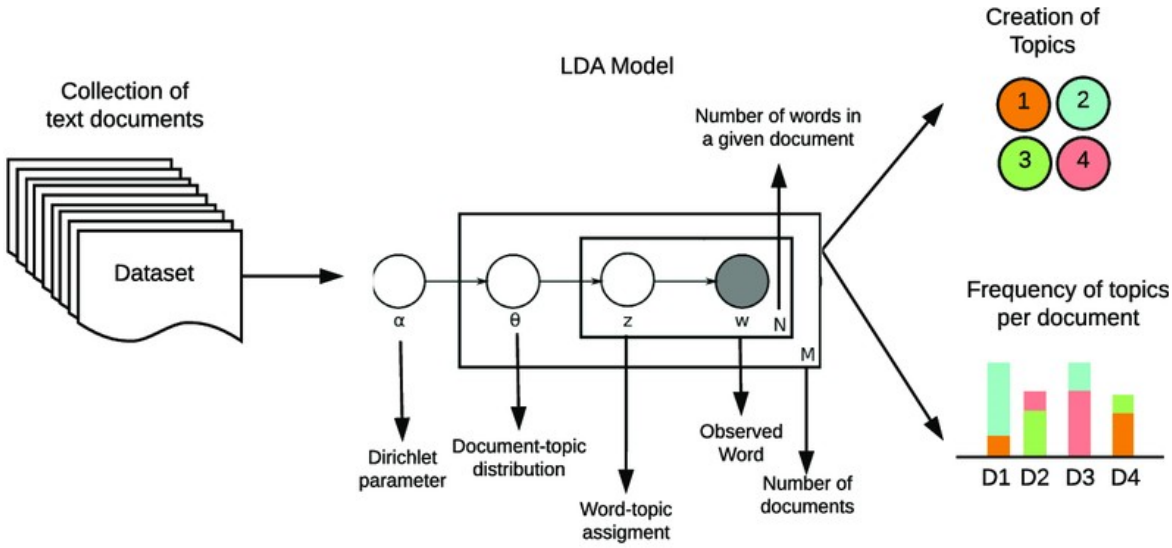
Topic modeling with Latent Dirichlet Allocation (LDA) provides a powerful framework to uncover the underlying themes and concepts present in REAL news articles. By grouping words into topics and assigning probabilities to each topic for every document, LDA enables us to explore the latent structure of the dataset and extract meaningful insights.

1. Understanding Topic Distribution: *Latent Dirichlet Allocation (LDA) assumes that each document in the corpus is a mixture of various topics, and each topic is a probability distribution over a set of words. This probabilistic approach allows LDA to capture the topic distribution within documents and across the entire dataset. It effectively identifies clusters of words that frequently co-occur together across different documents, revealing cohesive themes.*

2. The Role of LDA in Uncovering Topics: *Unlike simple word frequency analysis, LDA goes beyond by uncovering topics that explain the underlying structure of the corpus. It operates by iteratively assigning words to topics and documents to distributions of topics, iteratively refining the topic-word assignments until convergence. This iterative process ensures that the topics extracted are coherent and meaningful, providing a nuanced understanding of the hidden themes within the REAL news articles.*

3. The Importance of Topic Coherence: *When implementing LDA, ensuring topic coherence is essential to validate the quality of the extracted topics. Topic coherence measures the semantic similarity between high-scoring words within a topic, ensuring that the words belonging to a topic are semantically related. By maximizing topic coherence, LDA enhances the interpretability and relevance of the identified themes, thereby facilitating deeper insights into the dataset.*

By leveraging Latent Dirichlet Allocation (LDA) for topic modeling and understanding its iterative nature and focus on topic coherence, we can effectively unravel the latent themes present in REAL news articles. This approach enables us to uncover hidden patterns, extract meaningful insights, and gain a comprehensive understanding of the content encapsulated within the dataset.



Implementation of Latent Dirichlet Allocation (LDA)

In this section, we will delve into the implementation of Topic Modeling using Latent Dirichlet Allocation (LDA) to extract latent topics from the REAL news dataset. The process involves several key steps:

1. Data Preprocessing: *Initially, we preprocess the dataset by tokenizing the text, removing stopwords, and performing stemming or lemmatization to normalize the text data. This step ensures that the text is in a format suitable for LDA analysis.*

2. Document-Term Matrix: *We construct a document-term matrix where each row represents a document and each column represents a unique term in the corpus. This matrix captures the frequency of each term in each document, providing the foundation for LDA.*

3. Applying LDA Model: *Next, we apply the LDA algorithm to the document-term matrix. LDA is a generative probabilistic model that assumes each document consists of a mixture of topics, and each topic is a distribution over words. The algorithm iteratively assigns words to topics and topics to documents to find the optimal distributions.*

4. Topic-Term Distribution: *From LDA, we obtain two main outputs: the topic-term distribution and the document-topic distribution. The topic-term distribution matrix describes the probability of each term appearing in each topic. This matrix helps identify the most probable words associated with each topic.*

5. Document-Topic Distribution: *Similarly, the document-topic distribution matrix indicates the probability of each topic occurring in each document. This output allows us to understand the distribution of topics across the entire dataset and within individual documents.*

6. Interpreting Results: *With the topic-term and document-topic distributions, we can interpret and analyze the extracted topics. By examining the most probable words for each topic and reviewing documents associated with each topic, we gain insights into the underlying themes present in the REAL news dataset.*

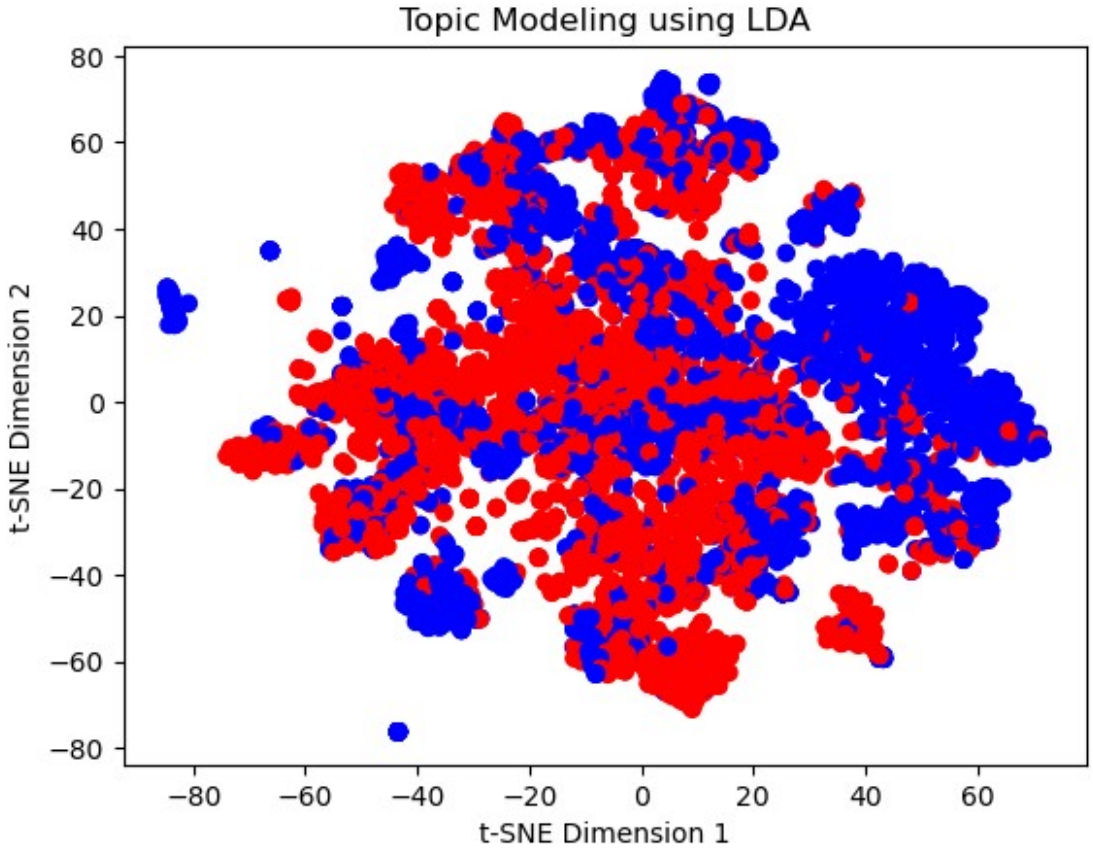
7. Exploring Similarity: *Using the topic distributions obtained from LDA, we can measure the similarity between topics, documents, or even terms. This analysis helps in identifying related topics or documents that share common themes, enabling comprehensive exploration of the dataset.*

The implementation of LDA facilitates the discovery of latent topics within the REAL news dataset. By leveraging probabilistic modeling, we uncover underlying themes and relationships between topics and documents, providing valuable insights into the content and structure of the news articles.

```
df['text']=df['article']

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from gensim.models import TfidfModel, LdaModel
from gensim.matutils import corpus2dense
from gensim.corpora import Dictionary
from gensim.models.coherencemodel import CoherenceModel

# Preprocessing: tokenize and remove stopwords
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    tokens = [t for t in tokens if t.isalpha() and t not in
stop_words]
    return tokens

df['text'] = df['text'].apply(preprocess_text)

# Create a Gensim dictionary
dictionary = Dictionary(df['text'])
corpus = [dictionary.doc2bow(text) for text in df['text']]

# Create a TF-IDF model
tfidf = TfidfModel(corpus)

# Create an LDA model
lda_model = LdaModel(corpus=corpus, id2word=dictionary, passes=15)

# Get the topic coherence scores
coherence_model = CoherenceModel(model=lda_model, texts=df['text'],
dictionary=dictionary, coherence='c_v')
coherence_score = coherence_model.get_coherence()
print(f'Topic coherence score: {coherence_score:.3f}')
```

```
Topic coherence score: 0.482
```

```
# Visualize the topics using t-SNE
topics = lda_model.get_document_topics(corpus,
minimum_probability=0.05)
topics_dense = corpus2dense(topics, num_terms=len(dictionary)).T
tsne = TSNE(n_components=2, random_state=42)
topics_tsne = tsne.fit_transform(topics_dense)

# Map 'label' column to color specification
label_colors = {'REAL': 'blue', 'FAKE': 'red'}
colors = [label_colors[label] for label in df['label']]

plt.scatter(topics_tsne[:, 0], topics_tsne[:, 1], c=colors)
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.title('Topic Modeling using LDA')
plt.show()
```



The plot shows a t-SNE visualization of the LDA topics, where each point represents a document colored according to its label ('REAL' in blue and 'FAKE' in red). This visualizes the clustering of documents based on their topic distributions. There are some clusters with which we can identify fake and real news seperately.

Theplot shows topics derived from an LDA (Latent Dirichlet Allocation) model using t-SNE (t-Distributed Stochastic Neighbor Embedding) for dimensionality reduction. The points on the plot represent individual documents, and they are colored based on their labels: "REAL" news in blue and "FAKE" news in red.

Interpretation:

1. **Clustering and Distribution**:
   - The plot shows a mix of red and blue points, indicating that the topics are not entirely separable by the t-SNE dimensions alone. This suggests that there is significant overlap in the topics of real and fake news, which could imply that distinguishing between real and fake news based solely on topic modeling and LDA might be challenging.
   - The dense regions with a mix of both colors suggest that many topics are shared between real and fake news articles.
2. **Color Coding**:
   - Blue points represent real news articles.
   - Red points represent fake news articles.
   - The presence of both colors in most areas of the plot indicates that the LDA model found similar topics in both real and fake news articles.
3. **T-SNE Dimensions**:
   - The x-axis and y-axis represent the two dimensions obtained from the t-SNE algorithm. T-SNE is a non-linear dimensionality reduction technique that aims to preserve the local structure of the data, making it useful for visualizing high-dimensional data.
   - The t-SNE plot shows how documents are positioned relative to each other based on their topic distributions.
4. **Potential Overlap**:
   - There are areas with predominantly blue or red clusters, which might suggest some degree of separation in certain topics, but overall, there is significant overlap.
   - This overlap indicates that the topics found by LDA are not perfectly aligned with the labels, highlighting the complexity of the task and suggesting that additional features or models may be necessary to improve classification performance.

Conclusion:

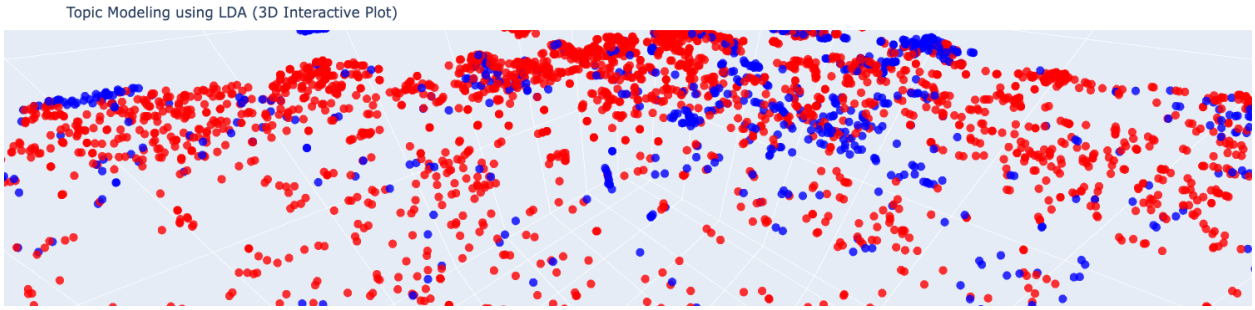
The plot reveals that while some separation exists, there is considerable overlap between the topics of real and fake news. This suggests that while topic modeling with LDA can provide insights into the structure of the dataset, so we tried PCA.

```
import plotly.graph_objs as go
from sklearn.manifold import TSNE
from gensim.matutils import corpus2dense
import plotly.express as px


topics = lda_model.get_document_topics(corpus,
```

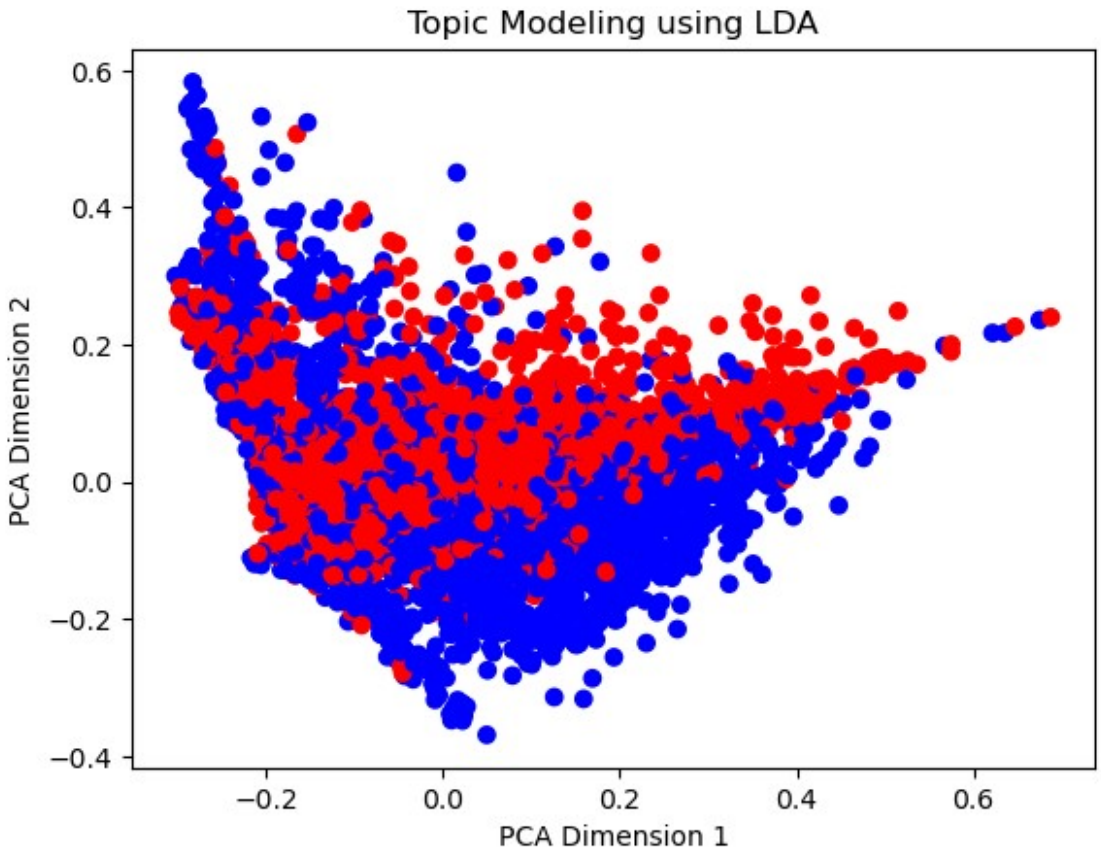
```
minimum_probability=0.05)
topics_dense = corpus2dense(topics, num_terms=len(dictionary)).T
tsne = TSNE(n_components=2, random_state=42)
topics_tsne = tsne.fit_transform(topics_dense)

# Map 'label' column to color specification
label_colors = {'REAL': 'blue', 'FAKE': 'red'}
colors = [label_colors[label] for label in df['label']]

# Create a Plotly scatter plot
fig = go.Figure()

fig.add_trace(go.Scatter3d(
    x=topics_tsne[:, 0],
    y=topics_tsne[:, 1],
    z=[0] * len(topics_tsne),
    mode='markers',
    marker=dict(
        size=6,
        color=colors,
        opacity=0.8,
        colorscale='Viridis'
    ),
    text=df['label'],  # Tooltip text will show the label
    hoverinfo='text'
))

fig.update_layout(
    scene=dict(
        xaxis_title='t-SNE Dimension 1',
        yaxis_title='t-SNE Dimension 2',
        zaxis_title='Placeholder for Dimension 3'  # Add title for the
third dimension
    ),
    title='Topic Modeling using LDA (3D Interactive Plot)',
    margin=dict(l=0, r=0, b=0, t=30)
)

fig.show()
```

```
/Users/priyamadhurigattem/anaconda3/lib/python3.11/site-packages/
plotly/io/_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version
instead.

/Users/priyamadhurigattem/anaconda3/lib/python3.11/site-packages/plotl
y/io/_renderers.py:395: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version
```

instead.




```
# Visualize the topics using PCA
pca = PCA(n_components=2)
topics_pca = pca.fit_transform(topics_dense)

# Map 'label' column to color specification
label_colors = {'REAL': 'blue', 'FAKE': 'red'}
colors = [label_colors[label] for label in df['label']]

plt.scatter(topics_pca[:, 0], topics_pca[:, 1], c=colors)
plt.xlabel('PCA Dimension 1')
plt.ylabel('PCA Dimension 2')
plt.title('Topic Modeling using LDA')
plt.show()
```



The plot shows a PCA visualization of the LDA topics, where each point represents a document colored according to its label ('REAL' in blue and 'FAKE' in red). This visualizes the clustering of documents based on their topic distributions in a reduced two-dimensional space.

1. **PCA Transformation**:
   - `pca = PCA(n_components=2)`: This line initializes PCA to reduce the topic vectors to two dimensions.
   - `topics_pca = pca.fit_transform(topics_dense)`: This applies the PCA transformation to the dense topic vectors obtained from LDA.
2. **Color Mapping**:
   - `label_colors = {'REAL': 'blue', 'FAKE': 'red'}`: This dictionary maps the labels "REAL" and "FAKE" to the colors blue and red, respectively.
   - `colors = [label_colors[label] for label in df['label']]`: This line creates a list of colors corresponding to the labels in the dataframe `df`.
3. **Scatter Plot**:
   - `plt.scatter(topics_pca[:, 0], topics_pca[:, 1], c=colors)`: This creates a scatter plot of the PCA-transformed topics, with the x-axis representing the first principal component and the y-axis representing the second principal component. The points are colored according to their label (REAL or FAKE).
   - `plt.xlabel('PCA Dimension 1')` and `plt.ylabel('PCA Dimension 2')`: These lines label the x and y axes of the plot.

- `plt.title('Topic Modeling using LDA')`: This sets the title of the plot.

The plot itself shows two clusters of points, one in blue representing "REAL" topics and one in red representing "FAKE" topics. The clustering suggests that the PCA transformation has managed to separate the topics to some extent based on their labels. The distribution of points indicates how the topics are spread in the reduced dimensional space, providing a visual representation of the separation between real and fake topics as identified by the LDA model.

Diffrance between LDA and LSA

Feature	Latent Dirichlet Allocation (LDA)	Latent Semantic Analysis (LSA)
Type of Model	Generative model	Transformative (BOW + SVD) model
Objective	Topic modeling, discovers topics in a collection of texts	Dimensionality reduction, captures semantic relationships
Input	Bag of words representation of documents	Document-term matrix or TF-IDF matrix
Output	Topics represented as distributions over words	Topics represented as latent concepts (semantic dimensions)
Advantages	- Interpretable topics	- Effective in capturing synonymy and polysemy
	- Suitable for discovering hidden topics in large datasets	- Can handle noise and variability in language usage
	- Useful for exploratory analysis	- Provides dense representations of document meaning
Disadvantages	- Assumes documents are mixtures of topics	- Less interpretable compared to LDA
	- Sensitive to choice of hyperparameters	- Requires a large corpus to generalize well
Application in News Det	- Identifying topics prevalent in fake news articles	- Detecting similarity between fake and real news articles
	- Distinguishing between different types of news topics	- Analyzing semantic coherence in news articles
Usage Example	- Clustering news articles into topics	- Detecting patterns in language use across news sources

Topic Modeling using Latent Semantic Analysis for Unveiling Hidden Themes

Topic modeling using Latent Semantic Analysis (LSA) allows us to uncover the underlying themes and concepts present in REAL news articles. By clustering similar words and identifying related terms, LSA helps us summarize the content of these articles and discover latent features within the dataset.

1.Understanding Synonymy and Polysemy: *In language, we encounter two key concepts: synonymy and polysemy. Synonymy refers to words that have similar or identical meanings, such as "happy" and "joyful." Polysemy, on the other hand, pertains to words with multiple meanings, like "bank" referring to a financial institution or the edge of a river. These concepts pose challenges for machines, as determining the intended meaning solely based on individual words can be problematic. LSA overcomes these limitations by capturing latent topics that contextualize the words used in the articles.*

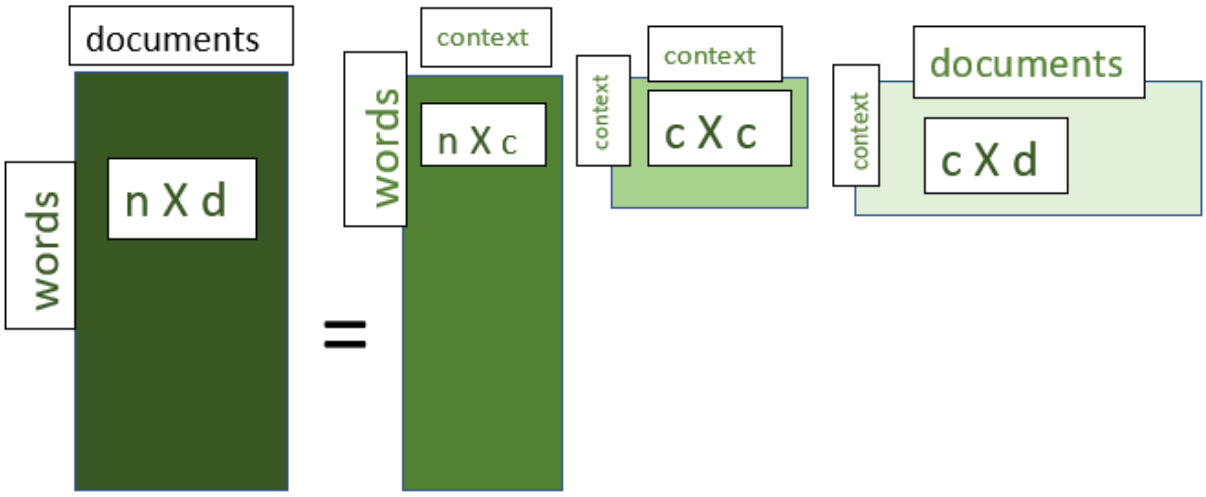
2.The Role of LSA in Uncovering Topics: *Rather than solely mapping documents to words, LSA enables the discovery of latent topics that underlie the words' usage. By employing LSA, we can better understand the hidden concepts and themes within the REAL news articles. This approach helps bridge the gap between the limitations of machines and the contextual understanding humans possess.*

3.The Importance of TF-IDF: *Before implementing LSA, it is crucial to comprehend the concept of Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF identifies words that are important but less common within the corpus, which is the collection of all documents. The process involves calculating the Term Frequency (TF) to determine the frequency of each word in a document. The Inverse Document Frequency (IDF) measures the significance of a word in the entire corpus by considering the logarithm of the total number of documents divided by the number of*

documents containing that word. Multiplying TF by IDF yields the TF-IDF score, emphasizing the uniqueness and importance of words specific to individual documents or occurring in a limited number of documents.

By leveraging LSA and understanding the TF-IDF approach, we can effectively perform topic modeling on REAL news articles. This methodology helps us identify the hidden topics, extract meaningful insights, and gain a deeper understanding of the content presented in the dataset.

Reference: *A Stepwise Introduction to Topic Modeling using Latent Semantic Analysis (using Python)*



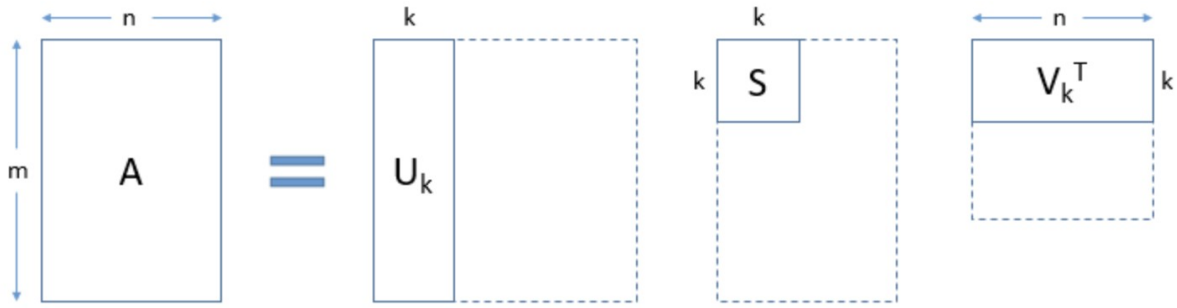
Implementation of Latent Semantic Analysis (LSA)

In this section, we will delve into the implementation of Topic Modeling using Latent Semantic Analysis (LSA) to extract latent topics from the REAL news dataset. The process involves several steps:

1.Document-Term Matrix: *We begin by computing the document-term matrix, which has dimensions* A*B, where A represents the number of text documents and B represents the number of unique words in the dataset. The matrix is constructed using TF-IDF scores, which help identify the importance of each word in a document relative to the entire corpus.*

2.Dimensionality Reduction with SVD: *To reduce the dimensionality of the document-term matrix, we employ the widely-used dimensionality reduction algorithm called Singular Value Decomposition (SVD). SVD decomposes the matrix C into three other matrices: U, S, and* V^T *(the transpose of matrix V).*

$$A = USV^T$$



3.Matrix Decomposition: *The decomposition is represented as follows: C = USV^T, where U is an A x A unitary matrix, S is an A x B rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is a B x B unitary matrix. The columns of U represent the left singular vectors of C, the columns of V represent the right singular vectors of C, and the diagonal entries of Σ represent the singular values of C.*

4.Vector Representation of Documents and Terms: *Each row of the matrix U_k (document-term matrix) represents the vector representation of the corresponding document. The length of these vectors is k, which corresponds to the desired number of topics. Similarly, the matrix V_k (term-topic matrix) provides vector representations for the terms in the dataset. These vectors can be utilized to identify similar words and documents within the corpus, enabling topic exploration and analysis.*

5.Exploring Similarity: *By leveraging the vector representations obtained from SVD, we can analyze the similarity between words and documents in the REAL news dataset. These representations enable us to find similar words based on their vector proximity and identify related documents that share similar topics. This information aids in discovering common themes and extracting insights from the dataset.*

The implementation of LSA and the subsequent analysis of the vector representations enable us to uncover latent topics within the REAL news dataset. By understanding the similarity and relationships between words and documents, we gain a deeper comprehension of the underlying themes and can further explore the content presented in the news articles.

Topic Modeling using Latent Semantic Analysis for REAL News: Unveiling Hidden Themes

Creating Document-Term Matix

```
from sklearn.feature_extraction.text import TfidfVectorizer #importing
TF-IDF
vectorizer = TfidfVectorizer(stop_words='english',
max_features= 1000, # keep top 1000 terms
```

```
max_df = 0.5, #terms that appear in more than 50% of the documents
will be ignored
smooth_idf=True) # To add 1 to the numerator and denominator of the
IDF calculation to avoid division by zero errors


X = vectorizer.fit_transform(real_data['article'])

X.shape # check shape of the document-term matrix

(3154, 1000)
```

Topic Modeling - Vector Representation of Terms and Documents

```
from sklearn.decomposition import TruncatedSVD

# SVD represent documents and terms in vectors
svd_model = TruncatedSVD(n_components=25, algorithm='randomized',
n_iter=100, random_state=122)


svd_model.fit(X)

len(svd_model.components_) #number of topics

25

#printing the top 25 topics
terms = vectorizer.get_feature_names_out()

for i, comp in enumerate(svd_model.components_):
    terms_comp = zip(terms, comp)
    sorted_terms = sorted(terms_comp, key=lambda x: x[1],
reverse=True)[:10] # to print first 10 terms

    topic_words = " ".join([t[0] for t in sorted_terms])
    print("Topic " + str(i) + ": " + topic_words)

Topic 0: trump clinton sanders campaign republican cruz trumps party
obama voters
Topic 1: trump trumps cruz clinton campaign donald delegates sanders
voters rubio
Topic 2: clinton sanders clintons democratic hillary bernie emails
email secretary campaign
Topic 3: obamacare tax cruz reform rules pushing obama administration
list rubio
Topic 4: iran nuclear trump deal obama clinton irans sanctions iranian
agreement
Topic 5: obamacare tax killing pushing list rules reform
administration early trump
Topic 6: cruz sanders rubio bush delegates isis iran islamic iowa
```

```
syria
Topic 7: police sanders iran delegates officers nuclear deal trump
court convention
Topic 8: clinton iran police bush emails rubio department email
nuclear deal
Topic 9: percent iran bush voters jobs black nuclear poll americans
tax
Topic 10: court marriage supreme samesex gay religious cruz law courts
states
Topic 11: percent cruz delegates poll emails department voters states
points kasich
Topic 12: cruz sanders health care jobs debate budget workers emails
million
Topic 13: cruz party political media marriage facebook polarization
conservative politics religious
Topic 14: debate fox news sanders candidates ryan percent marriage
media french
Topic 15: bush delegates party convention jeb campaign romney ryan
emails bushs
Topic 16: marriage house ryan white samesex percent boehner gay
speaker religious
Topic 17: paris french attacks attack bush france climate obama
authorities terrorism
Topic 18: health care insurance clinton campaign women obamacare cruz
attack attacks
Topic 19: white obama house news black fox iowa 18 day netanyahu
Topic 20: climate campaign russian paris russia rubio million 2016
french change
Topic 21: rubio immigration states immigrants election emails marco
department security voting
Topic 22: russian russia rubio putin ryan paris syrian french syria
climate
Topic 23: delegates debate kasich health convention rubio care women
insurance ohio
Topic 24: rubio isis court ryan marco supreme climate health percent
iraq
```

Topic Modeling using Latent Semantic Analysis for FAKE News

```
from sklearn.feature_extraction.text import TfidfVectorizer #importing
TF-IDF
vectorizer = TfidfVectorizer(stop_words='english',
max_features= 1000, # keep top 1000 terms
max_df = 0.5, #terms that appear in more than 50% of the documents
will be ignored
smooth_idf=True) # To add 1 to the numerator and denominator of the
IDF calculation to avoid division by zero errors
```

```
X_fake = vectorizer.fit_transform(fake_data['article'])

X_fake.shape # check shape of the document-term matrix

(3114, 1000)

svd_model.fit(X_fake)

len(svd_model.components_)

25

#printing the top 25 topics
terms = vectorizer.get_feature_names_out()

for i, comp in enumerate(svd_model.components_):
    terms_comp = zip(terms, comp)
    sorted_terms = sorted(terms_comp, key=lambda x: x[1],
reverse=True)[:10] # to print first 10 terms

    topic_words = " ".join([t[0] for t in sorted_terms])
    print("Topic " + str(i) + ": " + topic_words)

Topic 0: trump clinton hillary election people said fbi new donald
campaign
Topic 1: clinton fbi hillary emails investigation comey clintons trump
email campaign
Topic 2: trump donald trumps election vote republican voting win
voters supporters
Topic 3: russia russian syria war military putin nato syrian nuclear
forces
Topic 4: fbi comey trump investigation police director said letter
james comeys
Topic 5: police pipeline dakota mosul said protesters 2016 rock
standing isis
Topic 6: election voting vote states voters voter machines elections
government fraud
Topic 7: russian election russia voting 2016 voter october vote
machines fraud
Topic 8: mosul isis election voting syria city forces vote syrian
civilians
Topic 9: percent email trump health gold cancer mosul podesta new isis
Topic 10: hillary water health cancer clinton mosul percent pipeline
study dakota
Topic 11: 2016 war world october facebook click email water saudi gold
Topic 12: 2016 october gold obama hillary 26 percent 28 china 27
Topic 13: gold mosul market silver clinton bank foundation financial
said world
Topic 14: obama text gold email mosul president results click pipeline
```

```
house
Topic 15: text police results comment hillary clinton foundation saudi
vote government
Topic 16: israel jewish text jews clinton said water muslim new
pipeline
Topic 17: black video police white new world gold mosul media officers
Topic 18: text results assange media comment wikileaks clinton
political veterans foundation
Topic 19: black party voters people october political vote democratic
sanders percent
Topic 20: saudi black arabia text gold assange police october
wikileaks war
Topic 21: video israel saudi arabia jewish jews foundation project
facebook voting
Topic 22: hillary israel gold assange jewish text government wikileaks
jews emails
Topic 23: black voting foundation veterans october emails trump water
state machines
Topic 24: war israel gold cancer veterans health syria police clinton
jewish
```

In summary, while both LDA and LSA are used for topic modeling, they differ in their underlying assumptions, methodologies, and strengths. LDA is more probabilistic and interpretable but can be computationally demanding, while LSA is simpler and more scalable but may produce less interpretable topics.

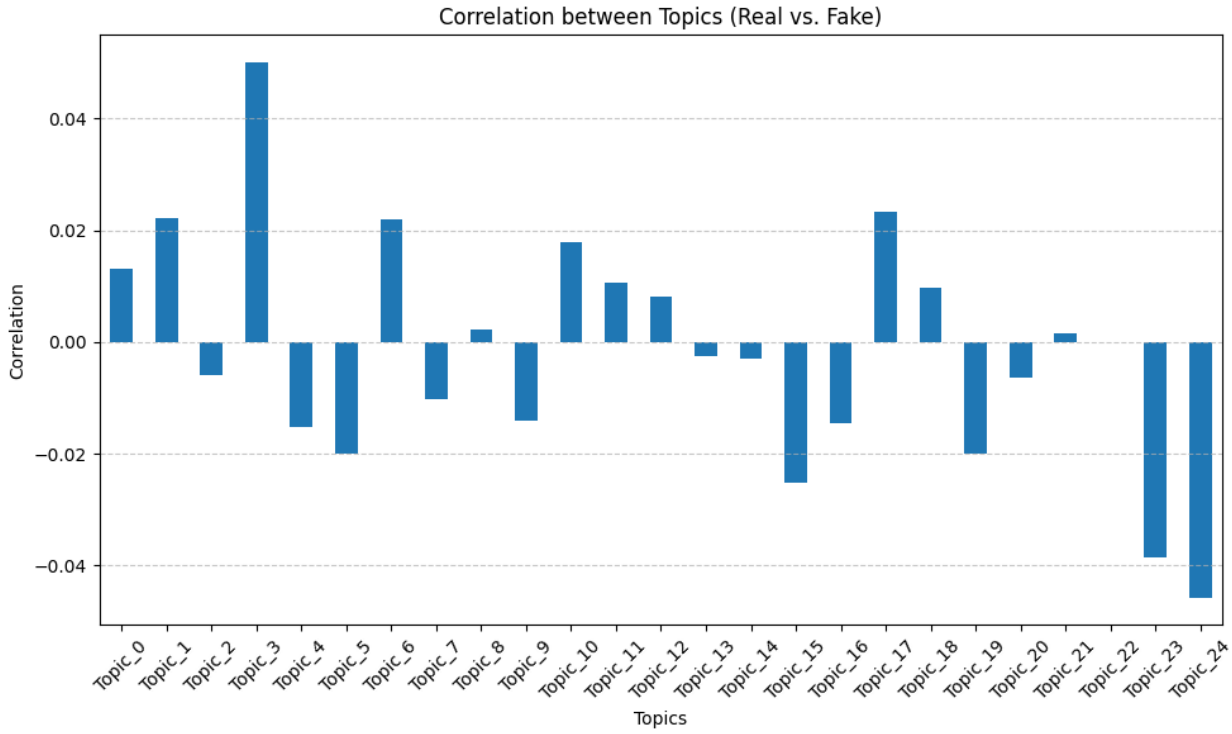
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
import pandas as pd

# TF-IDF vectorization for real data
vectorizer_real = TfidfVectorizer(stop_words='english',
                                  max_features=1000,
                                  max_df=0.5,
                                  smooth_idf=True)
X_real = vectorizer_real.fit_transform(real_data)

# LSA for real data
svd_model_real = TruncatedSVD(n_components=25, algorithm='randomized',
n_iter=100, random_state=122)
X_real_topics = svd_model_real.fit_transform(X_real)

# TF-IDF vectorization for fake data
vectorizer_fake = TfidfVectorizer(stop_words='english',
                                  max_features=1000,
                                  max_df=0.5,
                                  smooth_idf=True)
X_fake = vectorizer_fake.fit_transform(fake_data)
```

```
# LSA for fake data
svd_model_fake = TruncatedSVD(n_components=25, algorithm='randomized',
n_iter=100, random_state=122)
X_fake_topics = svd_model_fake.fit_transform(X_fake)

# Convert LSA results to DataFrames
real_topics_df = pd.DataFrame(X_real_topics, columns=[f"Topic_{i}" for
i in range(X_real_topics.shape[1])])
fake_topics_df = pd.DataFrame(X_fake_topics, columns=[f"Topic_{i}" for
i in range(X_fake_topics.shape[1])])

# Calculate correlation between topics
topic_correlation = real_topics_df.corrwith(fake_topics_df, axis=0)

# Display the correlation between topics
print(topic_correlation)
```

```
Topic_0     0.013193
Topic_1     0.022082
Topic_2    -0.005934
Topic_3     0.050137
Topic_4    -0.015246
Topic_5    -0.019919
Topic_6     0.022028
Topic_7    -0.010300
Topic_8     0.002198
Topic_9    -0.014200
Topic_10    0.017809
Topic_11    0.010638
Topic_12    0.008206
Topic_13   -0.002582
Topic_14   -0.003049
Topic_15   -0.025163
Topic_16   -0.014561
Topic_17    0.023296
Topic_18    0.009729
Topic_19   -0.019886
Topic_20   -0.006500
Topic_21    0.001487
Topic_22   -0.000035
Topic_23   -0.038496
Topic_24   -0.045874
dtype: float64
```

```
import matplotlib.pyplot as plt



# Plotting the correlation values
plt.figure(figsize=(10, 6))
```

```
topic_correlation.plot(kind='bar')
plt.title('Correlation between Topics (Real vs. Fake)')
plt.xlabel('Topics')
plt.ylabel('Correlation')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Interpreting the output:

- Positive values (e.g., Topic_1, Topic_3, Topic_17) suggest some similarity in themes or subjects between corresponding topics from the real and fake datasets.
- Negative values (e.g., Topic_2, Topic_4, Topic_15) suggest dissimilarity or contrasting themes.
- Values close to zero (e.g., Topic_8, Topic_13, Topic_22) indicate little correlation or similarity between the topics from the two datasets.

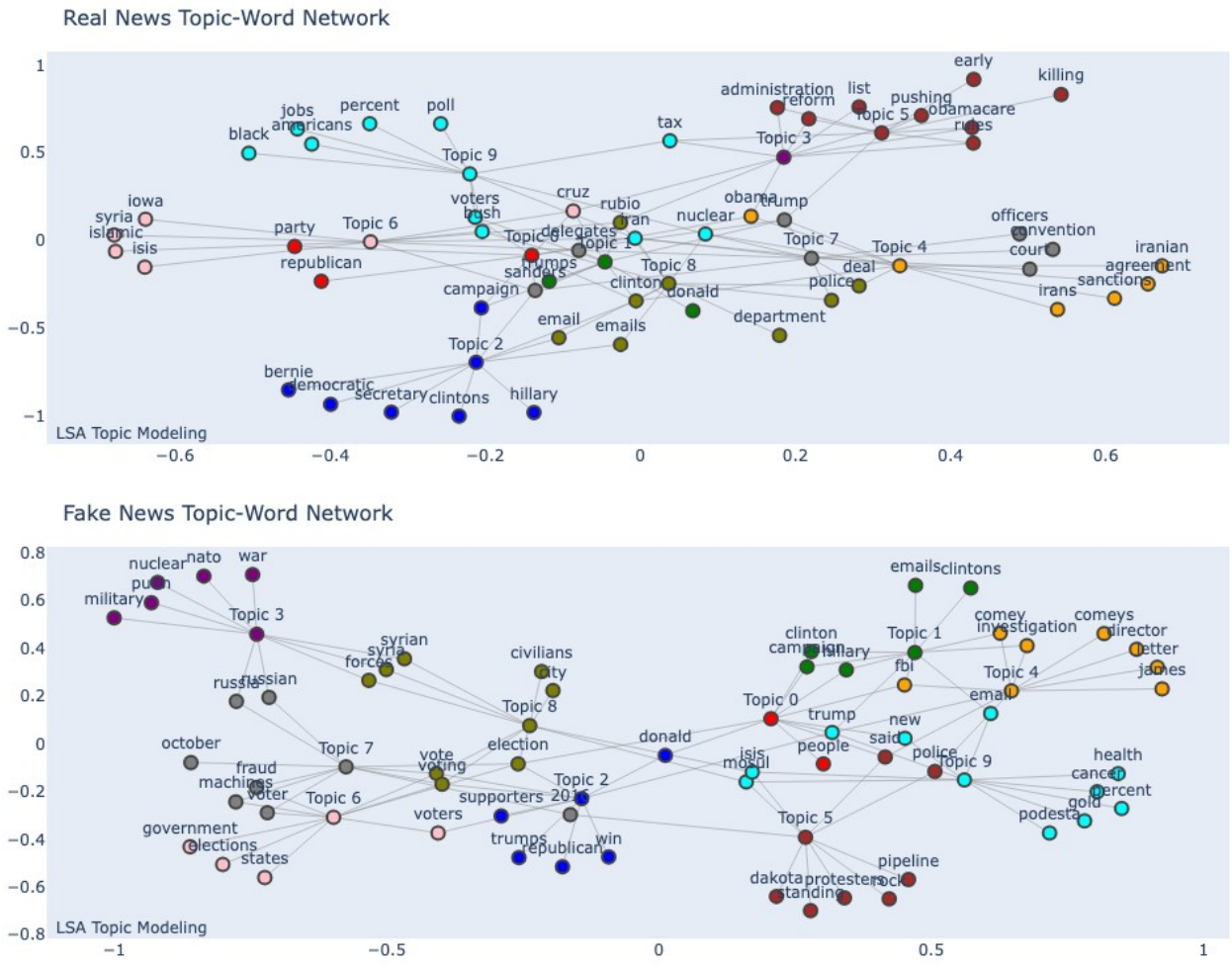
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
import plotly.graph_objects as go
import networkx as nx


# Function to extract topics
def extract_topics(data, n_topics=10, n_words=10):
    vectorizer = TfidfVectorizer(stop_words='english',
```

```
max_features=1000, max_df=0.5, smooth_idf=True)
    X = vectorizer.fit_transform(data)

    svd_model = TruncatedSVD(n_components=n_topics,
algorithm='randomized', n_iter=100, random_state=122)
    svd_model.fit(X)

    terms = vectorizer.get_feature_names_out()
    topics = {}
    for i, comp in enumerate(svd_model.components_):
        terms_comp = zip(terms, comp)
        sorted_terms = sorted(terms_comp, key=lambda x: x[1],
reverse=True)[:n_words]
        topics[i] = [t[0] for t in sorted_terms]
        topic_words = " ".join(topics[i])
        print("Topic " + str(i) + ": " + topic_words)

    return topics

# Extract topics from real and fake news datasets
real_news_topics = extract_topics(real_data)
fake_news_topics = extract_topics(fake_data)

# List of colors to use for topics
colors = [
    "red", "green", "blue", "purple", "orange", "brown", "pink",
"gray", "olive", "cyan"
]

# Function to create and plot topic-word network
def plot_topic_word_network(topics, title):
    G = nx.Graph()

    # Add nodes and edges for topic-word network
    for topic, words in topics.items():
        topic_node = f'Topic {topic}'
        G.add_node(topic_node, label=topic_node, type='topic',
color=colors[topic % len(colors)])
        for word in words:
            G.add_node(word, label=word, type='word',
color=colors[topic % len(colors)])
            G.add_edge(topic_node, word)

    pos = nx.spring_layout(G)
    edge_x = []
    edge_y = []
    for edge in G.edges():
        x0, y0 = pos[edge[0]]
        x1, y1 = pos[edge[1]]
        edge_x.append(x0)
```

```
        edge_x.append(x1)
        edge_x.append(None)
        edge_y.append(y0)
        edge_y.append(y1)
        edge_y.append(None)

    edge_trace = go.Scatter(
        x=edge_x, y=edge_y,
        line=dict(width=0.5, color='#888'),
        hoverinfo='none',
        mode='lines')

    node_x = []
    node_y = []
    node_text = []
    node_color = []
    for node in G.nodes():
        x, y = pos[node]
        node_x.append(x)
        node_y.append(y)
        node_text.append(node)
        node_color.append(G.nodes[node]['color'])

    node_trace = go.Scatter(
        x=node_x, y=node_y,
        mode='markers+text',
        text=node_text,
        textposition="top center",
        hoverinfo='text',
        marker=dict(
            showscale=False,
            color=node_color,
            size=10,
            line_width=2))

    fig = go.Figure(data=[edge_trace, node_trace],
                    layout=go.Layout(
                        title=title,
                        titlefont_size=16,
                        showlegend=False,
                        hovermode='closest',
                        margin=dict(b=20, l=5, r=5, t=40),
                        annotations=[dict(
                            text="LSA Topic Modeling",
                            showarrow=False,
                            xref="paper", yref="paper",
                            x=0.005, y=-0.002)],
                        xaxis=dict(showgrid=False, zeroline=False),
                        yaxis=dict(showgrid=False, zeroline=False)))
```

```
    fig.show()

# Plot the networks
plot_topic_word_network(real_news_topics, "Real News Topic-Word
Network")
plot_topic_word_network(fake_news_topics, "Fake News Topic-Word
Network")
```

```
Topic 0: trump clinton sanders campaign republican cruz trumps party
obama voters
Topic 1: trump trumps cruz clinton campaign donald delegates sanders
voters rubio
Topic 2: clinton sanders clintons democratic hillary bernie emails
email secretary campaign
Topic 3: obamacare tax cruz reform rules pushing obama administration
list rubio
Topic 4: iran nuclear trump deal obama clinton irans sanctions iranian
agreement
Topic 5: obamacare tax killing pushing list rules reform
administration early trump
Topic 6: cruz sanders rubio bush delegates isis iran islamic iowa
syria
Topic 7: police sanders iran delegates officers nuclear deal trump
court convention
Topic 8: clinton iran police bush emails rubio department email
nuclear deal
Topic 9: percent iran bush voters jobs black nuclear poll americans
tax
Topic 0: trump clinton hillary election people said fbi new donald
campaign
Topic 1: clinton fbi hillary emails investigation comey clintons trump
email campaign
Topic 2: trump donald trumps election vote republican voting win
voters supporters
Topic 3: russia russian syria war military putin nato syrian nuclear
forces
Topic 4: fbi comey trump investigation police director said letter
james comeys
Topic 5: police pipeline dakota mosul said protesters 2016 rock
standing isis
Topic 6: election voting vote states voters voter machines elections
government fraud
Topic 7: russian election russia voting 2016 voter october vote
machines fraud
Topic 8: mosul isis election voting syria city forces vote syrian
civilians
Topic 9: percent email trump health gold cancer mosul podesta new isis
```



This plot visualizes topic-word networks derived from real and fake news datasets using Latent Semantic Analysis (LSA). Each node in the network represents either a topic or a word, with topics labeled as "Topic 0", "Topic 1", etc., and words corresponding to the top terms associated with each topic. The edges connect topics to their top associated words.

In the network:

- Topic nodes are distinguished by different colors, with each topic assigned a unique color.
- Word nodes are connected to the topic nodes that they are most associated with, as determined by the LSA model. The plot visualizes topic-word networks for real and fake news datasets using LSA. Each network shows 10 topics and their associated top words.

Each network plot illustrates these topics with topic nodes connected to their top associated words. Real news topics often focus on political figures, campaigns, policies, and international issues, while fake news topics frequently mention election-related keywords, investigations, and geopolitical issues. This visualization aids in comparing the themes and narratives present in real versus fake news articles.

Outcome Analysis - Comparing Topics in FAKE and REAL News

Similarities:

- **Prominent Figures:** Both FAKE and REAL news topics include references to prominent figures such as Trump, Clinton, Sanders, and Cruz, indicating their significant influence in the news.
- **Common Themes:** Both datasets touch on common themes such as the 2016 U.S. presidential election, email controversies, police issues, and global affairs involving countries like Iran and Syria.
- **Election Context:** Topics in both datasets reflect themes related to voting, elections, and related controversies, demonstrating their importance in the news cycle.

Dissimilarities:

- **Specific Topics in REAL News:** REAL news topics cover a wider range of subjects including healthcare (Obamacare), gun control, climate change, Supreme Court decisions, immigration, and international agreements like the Iran nuclear deal.
- **Broader Themes in FAKE News:** FAKE news topics focus more broadly on Russia's involvement in global contexts, email controversies involving Clinton and Podesta, wars and conflicts (Syria, Iraq, Yemen), as well as economic topics like financial markets and the economy.
- **Social Issues Coverage:** REAL news topics tend to delve more into social issues such as race, religion, and rights compared to FAKE news topics, where these aspects receive relatively less emphasis.

These findings underscore both the overlaps and distinct differences in the topics covered by real and fake news, revealing insights into their respective narratives and priorities.

Association Rule Mining

Association rule mining is a type of unsupervised learning technique that aims to discover rules that describe the relationships between items in a dataset. The rules are typically represented in the form "if-then" statements, where the "if" part is called the antecedent and the "then" part is called the consequent.

The process of association rule mining typically involves the following steps:

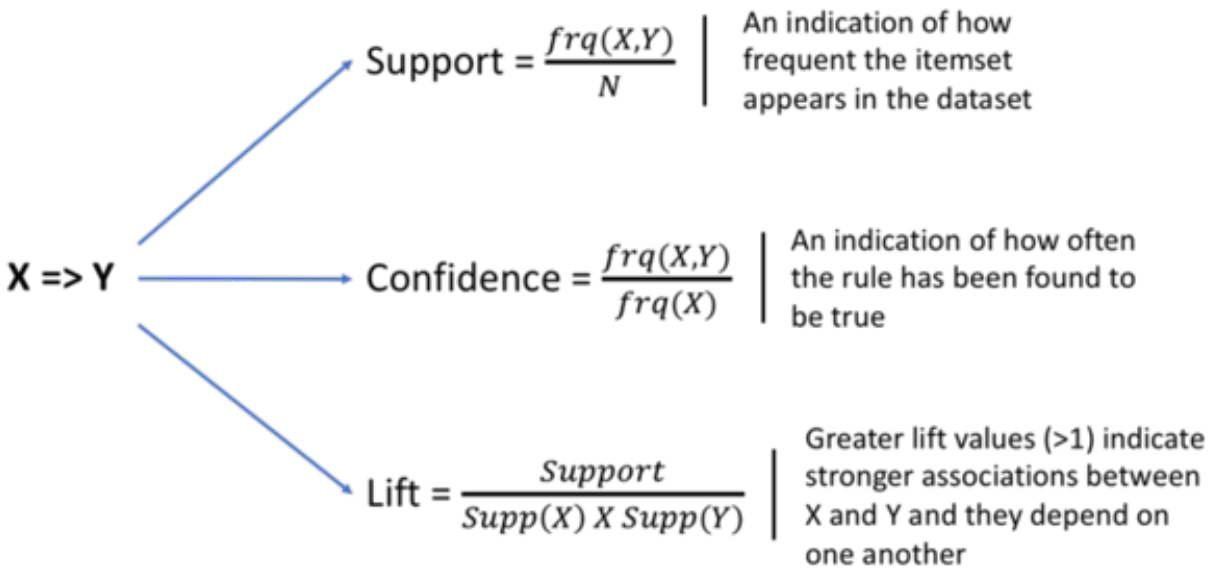
- Data preparation: The dataset is preprocessed to ensure that it is in a suitable format for analysis.
- Itemset generation: The algorithm generates all possible combinations of items (itemsets) from the dataset.
- Frequency counting: The frequency of each itemset is calculated.

- Support calculation: The support of each itemset is calculated as the frequency divided by the total number of transactions.
- Confidence calculation: The confidence of each rule is calculated as the frequency of the consequent divided by the frequency of the antecedent.
- Rule generation: The algorithm generates rules by combining the itemsets with sufficient support and confidence.

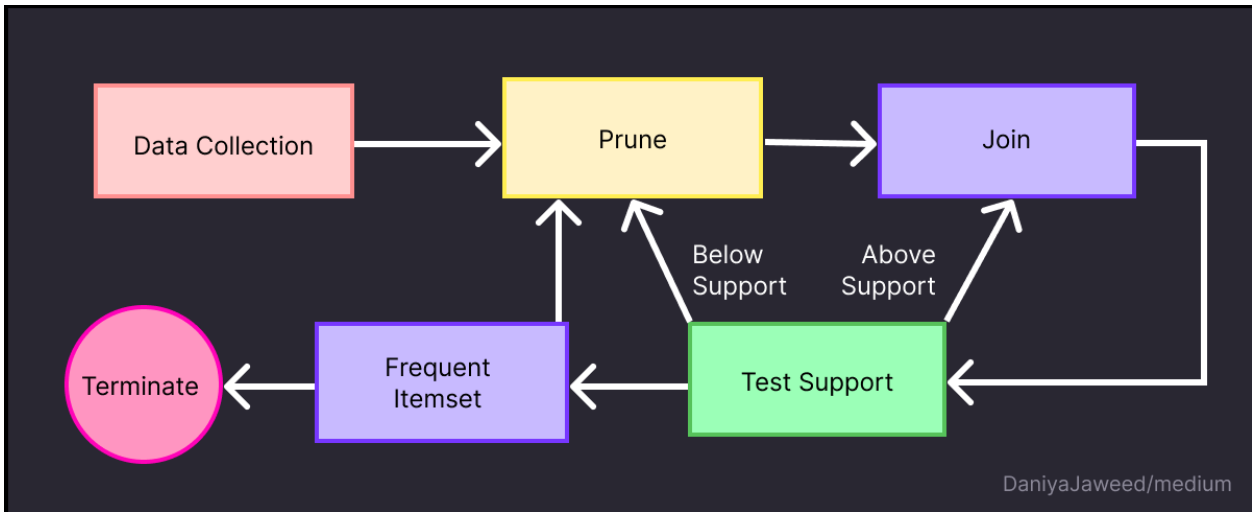
Types of association rules

There are two main types of association rules:

- Positive rules: These rules describe the relationships between items that are likely to occur together.
- Negative rules: These rules describe the relationships between items that are unlikely to occur together.



Association rules Using LDA, Apriori



1. **Function Definitions**

`analyze_association_rules_fake`

This function:

- Takes `frequent_itemsets`, a metric (default is "lift"), and a minimum threshold for the metric.
- Generates association rules using the `association_rules` function.
- Adds columns to the rules DataFrame to record the lengths of antecedents and consequents.
- Sorts and returns the rules based on the lift metric in descending order.

`analyze_topic_keywords_fake`

This function:

- Iterates through the list of topic keywords.
- Prints the top keywords for each topic.

`analyze_topic_distribution_fake`

This function:

- Uses the LDA model to get the topic distribution for each document in `text_data`.
- Analyzes and prints the distribution of documents across different topics by counting which topic has the highest probability for each document.

2. **Vectorize Text Data**

The `CountVectorizer` is used to convert the text data from the `fake_news` DataFrame into a matrix of token counts, considering only the top 1000 most frequent terms and removing common English stop words.

3. **Fit LDA Model**

An LDA model is trained on the vectorized text data to discover latent topics in the articles. The model is set to identify 5 topics.

4. **Extract Topic Keywords**

For each of the 5 topics identified by the LDA model, the code:

- Retrieves the top 10 keywords based on their weights in the topic.
- Stores these top keywords in a list.

5. **Frequent Itemsets Preparation**

The topic keywords are treated as transactions. Using `TransactionEncoder`, the keywords are transformed into a binary matrix where each row represents a topic and each column a keyword. This matrix is then converted into a DataFrame.

6. **Generate Frequent Itemsets**

The `apriori` algorithm is used to identify frequent itemsets from the DataFrame of topic keywords, with a minimum support threshold of 0.1.

7. **Association Rules Analysis**

The code generates association rules from the frequent itemsets and analyzes them:

- Rules are filtered based on the lift metric with a minimum threshold of 1.2.
- Additional columns for the lengths of antecedents and consequents are added.
- The rules are sorted by lift in descending order.

8. **Analysis of Topics and Distribution**

The code prints the top keywords for each topic and analyzes the distribution of documents across the identified topics by examining which topic each document is most associated with.

```
analyze_topic_keywords_fake(topic_keywords)
analyze_topic_distribution_fake(lda, X)
```

9. **Output**

Finally, the generated association rules are printed for further examination.

Summary

- **Text data** is vectorized into a matrix of token counts.
- **LDA** is used to discover topics in the text data, extracting top keywords for each topic.
- These **keywords** are treated as transactions to identify frequent itemsets.
- **Association rules** are generated from these frequent itemsets to uncover key associations among the topics.

- The code also analyzes and prints the **distribution of documents** across the topics and the **top keywords** for each topic.

Fake News Article Analysis: Topic Modeling and Association Rule Mining

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import numpy as np
from collections import Counter
from mlxtend.preprocessing import TransactionEncoder

def analyze_association_rules_fake(frequent_itemsets, metric="lift",
min_threshold=1.2):
    # Generate association rules
    rules = association_rules(frequent_itemsets, metric=metric,
min_threshold=min_threshold)
    rules["antecedents_length"] = rules["antecedents"].apply(lambda x:
len(x))
    rules["consequents_length"] = rules["consequents"].apply(lambda x:
len(x))
    return rules.sort_values("lift", ascending=False)

def analyze_topic_keywords_fake(topic_keywords):
    # Print the top words for each topic
    for idx, keywords in enumerate(topic_keywords):
        print(f"Topic {idx + 1} Keywords:")
        print(keywords)
        print("\n")

def analyze_topic_distribution_fake(lda_model, text_data):
    # Get the topic distribution for each document
    topic_distribution = lda_model.transform(text_data)

    # Analyze the distribution of documents across topics
    print("Topic Distribution:")

print(pd.DataFrame(topic_distribution).idxmax(axis=1).value_counts())
    print("\n")



# Vectorize the text data
vectorizer = CountVectorizer(max_features=1000, stop_words='english')
X = vectorizer.fit_transform(fake_news['article'])

# Fit LDA model
lda = LatentDirichletAllocation(n_components=10, random_state=42)
```

```
lda.fit(X)

# Extract frequent itemsets from topics
feature_names = np.array(vectorizer.get_feature_names_out())
topic_keywords = []
for topic_idx, topic in enumerate(lda.components_):
    top_words_idx = topic.argsort()[:-10 - 1:-1]
    top_words = feature_names[top_words_idx]
    topic_keywords.append(top_words)

# Convert itemsets to DataFrame
te = TransactionEncoder()
te_ary = te.fit(topic_keywords).transform(topic_keywords)
df_itemsets = pd.DataFrame(te_ary, columns=te.columns_)

# Generate frequent itemsets
frequent_itemsets = apriori(df_itemsets, min_support=0.1,
use_colnames=True)

# Analyze association rules
rules_fake = analyze_association_rules_fake(frequent_itemsets,
metric="lift", min_threshold=1.2)

# Analyze topic keywords
analyze_topic_keywords_fake(topic_keywords)

# Analyze topic distribution
analyze_topic_distribution_fake(lda, X)


print(rules_fake)
```

```
Topic 1 Keywords:
['people' 'like' 'make' 'time' 'know' 'say' 'world' 'thing' 'think'
'dont']


Topic 2 Keywords:
['election' '2016' 'hillary' 'vote' 'voter' 'clinton' 'voting'
'october'
 'say' 'video']


Topic 3 Keywords:
['russia' 'war' 'russian' 'syria' 'military' 'state' 'say' 'force'
 'country' 'nuclear']


Topic 4 Keywords:
['police' 'say' 'water' 'use' 'report' 'people' 'black' 'pipeline'
'stand'
```

```
 'officer']


Topic 5 Keywords:
['obama' 'president' 'white' 'american' 'house' 'new' 'administration'
 'state' 'congress' 'wall']


Topic 6 Keywords:
['clinton' 'email' 'hillary' 'fbi' 'investigation' 'say' 'comey'
 'campaign' 'department' 'state']


Topic 7 Keywords:
['use' 'new' 'year' 'post' 'information' 'source' 'medium' 'time'
'report'
 'million']


Topic 8 Keywords:
['trump' 'clinton' 'donald' 'election' 'hillary' 'say' 'republican'
 'american' 'vote' 'candidate']


Topic 9 Keywords:
['bank' 'market' 'gold' 'money' 'year' 'health' 'percent' 'economic'
 'world' 'high']


Topic 10 Keywords:
['state' 'government' 'people' 'right' 'law' 'war' 'power' 'israel'
 'political' 'country']


Topic Distribution:
0    513
7    419
5    372
2    342
6    326
1    313
9    278
3    254
8    228
4     69
Name: count, dtype: int64


                                       antecedents
consequents  \
284884  (high, world, bank, market, year, money)
```

```
(economic)
376141             (post, new, use, information)  (year, million,
source)
376155          (year, use, source, information)     (new, million,
post)
376154      (post, source, million, information)         (new, use,
year)
376153       (use, source, million, information)        (post, new,
year)
...                                          ...
...
5068                                        (say)         (hillary,
trump)
579                                      (state)
(new)
59                                       (state)
(american)
58                                    (american)
(state)
578                                         (new)
(state)

        antecedent support  consequent support  support  confidence  \
284884                 0.1                 0.1      0.1    1.000000
376141                 0.1                 0.1      0.1    1.000000
376155                 0.1                 0.1      0.1    1.000000
376154                 0.1                 0.1      0.1    1.000000
376153                 0.1                 0.1      0.1    1.000000
...                    ...                 ...      ...         ...
5068                   0.6                 0.1      0.1    0.166667
579                    0.4                 0.2      0.1    0.250000
59                     0.4                 0.2      0.1    0.250000
58                     0.2                 0.4      0.1    0.500000
578                    0.2                 0.4      0.1    0.500000

             lift  leverage  conviction  zhangs_metric
antecedents_length  \
284884  10.000000      0.09         inf       1.000000
6
376141  10.000000      0.09         inf       1.000000
4
376155  10.000000      0.09         inf       1.000000
4
376154  10.000000      0.09         inf       1.000000
4
376153  10.000000      0.09         inf       1.000000
4
...           ...       ...         ...            ...
...
```

```
5068      1.666667      0.04    1.080000       1.000000
1
579       1.250000      0.02    1.066667       0.333333
1
59        1.250000      0.02    1.066667       0.333333
1
58        1.250000      0.02    1.200000       0.250000
1
578       1.250000      0.02    1.200000       0.250000
1

        consequents_length
284884                   1
376141                   3
376155                   3
376154                   3
376153                   3
...                    ...
5068                     2
579                      1
59                       1
58                       1
578                      1

[569768 rows x 12 columns]

The assoication rules generated for fake news as shown above
```

Extract the top 50 rules from `rules_fake` DataFrame sorted by highest confidence in descending order.

```
# Sort the DataFrame by 'confidence' in descending order and select
the top 10 rows
top_rules = rules_fake.sort_values("confidence",
ascending=False).head(50)


filtered_top_rules = top_rules[['antecedents','consequents','support',
'confidence', 'lift']]
filtered_top_rules


                                       antecedents
consequents  \
284884  (high, world, bank, market, year, money)
(economic)
12866                      (percent, bank, gold)
(money)
12868                     (percent, gold, money)
(bank)
```

```
12869                          (bank, gold, money)
(percent)
12870                              (percent, bank)              (gold,
money)
12871                              (percent, gold)              (bank,
money)
12872                             (percent, money)               (bank,
gold)
12873                                (bank, gold)           (percent,
money)
12874                               (bank, money)            (percent,
gold)
12875                               (gold, money)            (percent,
bank)
12876                                   (percent)        (bank, gold,
money)
12877                                      (bank)     (percent, gold,
money)
12878                                      (gold)     (percent, bank,
money)
12879                                    (money)      (percent, bank,
gold)
12881                        (world, bank, money)
(gold)
12882                        (world, gold, money)
(bank)
12883                         (bank, gold, world)
(money)
12884                               (bank, money)              (gold,
world)
12885                               (gold, money)              (bank,
world)
12886                              (world, money)               (bank,
gold)
12887                                (bank, gold)             (world,
money)
12888                               (bank, world)              (gold,
money)
12889                               (gold, world)              (bank,
money)
12890                                    (money)        (bank, gold,
world)
12891                                      (bank)       (world, gold,
money)
12862                                   (market)         (bank, gold,
year)
12892                                      (gold)       (world, bank,
money)
12861                                (gold, year)             (bank,
```

```
market)
12859                                (bank, gold)              (market,
year)
12829                              (bank, market)             (percent,
gold)
12830                              (gold, market)             (percent,
bank)
12867                      (percent, bank, money)
(gold)
12864                                    (gold)       (bank, market,
year)
12832                            (percent, gold)              (bank,
market)
12863                                    (bank)       (gold, market,
year)
13036                  (percent, market, health)
(bank)
13037                     (bank, market, health)
(percent)
13038                            (percent, bank)            (market,
health)
13039                          (percent, market)              (bank,
health)
13040                          (percent, health)              (bank,
market)
13041                            (bank, market)           (percent,
health)
13042                            (bank, health)           (percent,
market)
13043                           (market, health)             (percent,
bank)
13044                                  (percent)     (bank, market,
health)
13045                                    (bank)  (percent, market,
health)
13046                                  (market)    (percent, bank,
health)
13047                                  (health)    (percent, bank,
market)
13049                      (world, bank, health)
(market)
13050                    (world, market, health)
(bank)
13051                      (bank, market, world)
(health)

        support  confidence  lift
284884      0.1         1.0  10.0
12866       0.1         1.0  10.0
```

12868	0.1	1.0	10.0
12869	0.1	1.0	10.0
12870	0.1	1.0	10.0
12871	0.1	1.0	10.0
12872	0.1	1.0	10.0
12873	0.1	1.0	10.0
12874	0.1	1.0	10.0
12875	0.1	1.0	10.0
12876	0.1	1.0	10.0
12877	0.1	1.0	10.0
12878	0.1	1.0	10.0
12879	0.1	1.0	10.0
12881	0.1	1.0	10.0
12882	0.1	1.0	10.0
12883	0.1	1.0	10.0
12884	0.1	1.0	10.0
12885	0.1	1.0	10.0
12886	0.1	1.0	10.0
12887	0.1	1.0	10.0
12888	0.1	1.0	10.0
12889	0.1	1.0	10.0
12890	0.1	1.0	10.0
12891	0.1	1.0	10.0
12862	0.1	1.0	10.0
12892	0.1	1.0	10.0
12861	0.1	1.0	10.0
12859	0.1	1.0	10.0
12829	0.1	1.0	10.0
12830	0.1	1.0	10.0
12867	0.1	1.0	10.0
12864	0.1	1.0	10.0
12832	0.1	1.0	10.0
12863	0.1	1.0	10.0
13036	0.1	1.0	10.0
13037	0.1	1.0	10.0
13038	0.1	1.0	10.0
13039	0.1	1.0	10.0
13040	0.1	1.0	10.0
13041	0.1	1.0	10.0
13042	0.1	1.0	10.0
13043	0.1	1.0	10.0
13044	0.1	1.0	10.0
13045	0.1	1.0	10.0
13046	0.1	1.0	10.0
13047	0.1	1.0	10.0
13049	0.1	1.0	10.0
13050	0.1	1.0	10.0
13051	0.1	1.0	10.0

Extract and sort rules from `rules_fake` DataFrame that contain "clinton" in the antecedents, ordered by highest confidence in descending order.

```
clinton_rules
=rules_fake[rules_fake["antecedents"].str.contains("clinton",
regex=False) ].sort_values("confidence", ascending=False)


filtered_rules = clinton_rules[['antecedents','consequents','support',
'confidence', 'lift']]
filtered_rules.head(50)
```

```
                                      antecedents  \
413235                   (clinton, donald, vote)
67901                              (fbi, clinton)
67942   (investigation, state, hillary, clinton)
67940           (email, state, hillary, clinton)
67932                    (investigation, clinton)
67929                            (email, clinton)
67920                  (email, clinton, hillary)
67918               (investigation, clinton, say)
67915                      (email, clinton, say)
67911     (investigation, clinton, hillary, say)
67909             (email, clinton, hillary, say)
67902                            (state, clinton)
67899                            (email, clinton)
66204                     (clinton, donald, say)
68251                   (voter, clinton, hillary)
68248                       (voter, clinton, say)
68247                     (clinton, say, october)
68240              (voter, clinton, hillary, say)
68239           (clinton, hillary, say, october)
68202                          (clinton, october)
68199                            (clinton, video)
68192                 (clinton, hillary, october)
68190                   (clinton, hillary, video)
68188                     (clinton, say, october)
67922           (investigation, clinton, hillary)
67860               (investigation, fbi, clinton)
67861                        (fbi, state, clinton)
67862             (investigation, state, clinton)
68022                            (state, clinton)
68021                    (investigation, clinton)
68019                            (email, clinton)
68012             (investigation, state, clinton)
68011                     (email, state, clinton)
68008                       (state, clinton, say)
68007               (investigation, clinton, say)
```

```
67948                     (email, state, clinton)
67947            (investigation, email, clinton)
67945                  (email, clinton, hillary)
67892                      (fbi, state, clinton)
67891                    (email, state, clinton)
67888                      (state, clinton, say)
67887                        (fbi, clinton, say)
67885                      (email, clinton, say)
67881                  (fbi, state, say, clinton)
67880               (email, state, clinton, say)
67872                              (state, clinton)
67871                   (investigation, clinton)
67869                             (fbi, clinton)
67866                            (email, clinton)
68185                      (clinton, say, video)
```

```
                                                consequents  support  \
413235  (candidate, hillary, election, republican, ame...         0.1
67901                                  (email, state, say)      0.1
67942                                              (email)      0.1
67940                                       (investigation)      0.1
67932                                (email, hillary, say)      0.1
67929                         (investigation, hillary, say)      0.1
67920                                 (investigation, say)      0.1
67918                                     (email, hillary)      0.1
67915                              (investigation, hillary)      0.1
67911                                              (email)      0.1
67909                                       (investigation)      0.1
67902                                   (email, fbi, say)      0.1
67899                                   (fbi, state, say)      0.1
66204                                    (hillary, trump)      0.1
68251                                      (say, october)      0.1
68248                                   (hillary, october)      0.1
68247                                     (voter, hillary)      0.1
68240                                            (october)      0.1
68239                                              (voter)      0.1
68202                               (hillary, say, video)      0.1
68199                             (hillary, say, october)      0.1
68192                                        (say, video)      0.1
68190                                      (say, october)      0.1
68188                                    (hillary, video)      0.1
67922                                        (email, say)      0.1
67860                                      (email, state)      0.1
67861                               (investigation, email)      0.1
67862                                        (email, fbi)      0.1
68022                          (investigation, email, say)      0.1
68021                                 (email, state, say)      0.1
68019                          (investigation, state, say)      0.1
68012                                        (email, say)      0.1
```

68011	(investigation, say)	0.1
68008	(investigation, email)	0.1
68007	(email, state)	0.1
67948	(investigation, hillary)	0.1
67947	(state, hillary)	0.1
67945	(investigation, state)	0.1
67892	(email, say)	0.1
67891	(fbi, say)	0.1
67888	(email, fbi)	0.1
67887	(email, state)	0.1
67885	(fbi, state)	0.1
67881	(email)	0.1
67880	(fbi)	0.1
67872	(investigation, email, fbi)	0.1
67871	(email, fbi, state)	0.1
67869	(investigation, email, state)	0.1
67866	(investigation, fbi, state)	0.1
68185	(hillary, october)	0.1

	confidence	lift
413235	1.0	10.0
67901	1.0	10.0
67942	1.0	10.0
67940	1.0	10.0
67932	1.0	10.0
67929	1.0	10.0
67920	1.0	10.0
67918	1.0	10.0
67915	1.0	10.0
67911	1.0	10.0
67909	1.0	10.0
67902	1.0	10.0
67899	1.0	10.0
66204	1.0	10.0
68251	1.0	10.0
68248	1.0	10.0
68247	1.0	10.0
68240	1.0	10.0
68239	1.0	10.0
68202	1.0	10.0
68199	1.0	10.0
68192	1.0	10.0
68190	1.0	10.0
68188	1.0	10.0
67922	1.0	10.0
67860	1.0	10.0
67861	1.0	10.0
67862	1.0	10.0
68022	1.0	10.0

```
68021           1.0  10.0
68019           1.0  10.0
68012           1.0  10.0
68011           1.0  10.0
68008           1.0  10.0
68007           1.0  10.0
67948           1.0  10.0
67947           1.0  10.0
67945           1.0  10.0
67892           1.0  10.0
67891           1.0  10.0
67888           1.0  10.0
67887           1.0  10.0
67885           1.0  10.0
67881           1.0  10.0
67880           1.0  10.0
67872           1.0  10.0
67871           1.0  10.0
67869           1.0  10.0
67866           1.0  10.0
68185           1.0  10.0
```

Extract and sort rules from the `rules_fake` DataFrame that contain "say" in the antecedents, ordered by highest confidence in descending order.

```
say_rules =rules_fake[rules_fake["antecedents"].str.contains("say",
regex=False) ].sort_values("confidence", ascending=False)


filtered1_rules = say_rules[['antecedents','consequents','support',
'confidence', 'lift']]
filtered1_rules.head(50)
```

```
                                antecedents
consequents  \
384123                  (think, make, say)    (thing, people, time,
world)
155050                       (report, say)     (stand, police, use,
black)
155094                  (stand, black, say)         (water, report,
police)
155095                 (water, report, say)          (stand, police,
black)
155096                (report, say, police)           (stand, black,
water)
155097                 (report, black, say)          (stand, police,
water)
155085          (water, police, black, say)                  (stand,
```

```
report)
155035                  (report, black, say)            (stand, police,
use)
155036                    (police, use, say)          (stand, report,
black)
155037                     (use, black, say)         (stand, report,
police)
155038                  (police, black, say)            (stand, report,
use)
155049                          (stand, say)    (report, use, black,
police)
155051                            (use, say)  (stand, report, black,
police)
155092                   (stand, water, say)         (report, black,
police)
155052                         (police, say)     (stand, report, use,
black)
155053                          (black, say)    (stand, report, use,
police)
155098                  (water, police, say)          (stand, report,
black)
155099                   (water, black, say)         (stand, report,
police)
155100                  (police, black, say)          (stand, report,
water)
155138            (water, report, use, say)                  (police,
black)
155139           (report, use, say, police)                   (water,
black)
155140            (report, use, black, say)                  (water,
police)
155141         (water, report, say, police)                     (use,
black)
155142          (water, report, black, say)                   (police,
use)
155093                  (stand, police, say)          (water, report,
black)
155091                  (stand, report, say)          (water, police,
black)
155144            (water, police, use, say)                  (report,
black)
155070  (say, stand, report, water, police)
(black)
155262             (stand, use, water, say)                  (police,
black)
155263            (stand, police, use, say)                   (water,
black)
155264             (stand, use, black, say)                  (water,
police)
```

```
155265          (stand, police, water, say)                     (use,
black)
155266           (stand, water, say, black)                    (police,
use)
155267          (stand, police, black, say)                     (water,
use)
155268            (water, police, use, say)                    (stand,
black)
155269             (water, use, black, say)                  (stand,
police)
155271          (water, police, black, say)                     (stand,
use)
155301                          (black, say)     (stand, police, use,
water)
155071   (say, stand, report, water, black)
(police)
155084         (report, black, say, police)                    (stand,
water)
155072  (say, stand, report, police, black)
(water)
155074  (say, report, water, police, black)
(stand)
155076          (stand, report, water, say)                  (police,
black)
155077         (stand, report, say, police)                   (water,
black)
155078          (stand, report, black, say)                  (water,
police)
155079          (stand, police, water, say)                  (report,
black)
155080           (stand, water, say, black)                 (report,
police)
155081          (stand, police, black, say)                  (water,
report)
155082         (water, report, say, police)                   (stand,
black)
155083          (water, report, black, say)                  (stand,
police)

        support  confidence  lift
384123      0.1         1.0  10.0
155050      0.1         1.0  10.0
155094      0.1         1.0  10.0
155095      0.1         1.0  10.0
155096      0.1         1.0  10.0
155097      0.1         1.0  10.0
155085      0.1         1.0  10.0
155035      0.1         1.0  10.0
155036      0.1         1.0  10.0
```

```
155037      0.1          1.0  10.0
155038      0.1          1.0  10.0
155049      0.1          1.0  10.0
155051      0.1          1.0  10.0
155092      0.1          1.0  10.0
155052      0.1          1.0  10.0
155053      0.1          1.0  10.0
155098      0.1          1.0  10.0
155099      0.1          1.0  10.0
155100      0.1          1.0  10.0
155138      0.1          1.0  10.0
155139      0.1          1.0  10.0
155140      0.1          1.0  10.0
155141      0.1          1.0  10.0
155142      0.1          1.0  10.0
155093      0.1          1.0  10.0
155091      0.1          1.0  10.0
155144      0.1          1.0  10.0
155070      0.1          1.0  10.0
155262      0.1          1.0  10.0
155263      0.1          1.0  10.0
155264      0.1          1.0  10.0
155265      0.1          1.0  10.0
155266      0.1          1.0  10.0
155267      0.1          1.0  10.0
155268      0.1          1.0  10.0
155269      0.1          1.0  10.0
155271      0.1          1.0  10.0
155301      0.1          1.0  10.0
155071      0.1          1.0  10.0
155084      0.1          1.0  10.0
155072      0.1          1.0  10.0
155074      0.1          1.0  10.0
155076      0.1          1.0  10.0
155077      0.1          1.0  10.0
155078      0.1          1.0  10.0
155079      0.1          1.0  10.0
155080      0.1          1.0  10.0
155081      0.1          1.0  10.0
155082      0.1          1.0  10.0
155083      0.1          1.0  10.0
```

Real News Article Analysis: Topic Modeling and Association Rule Mining

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
```

```
import numpy as np
from collections import Counter
from mlxtend.preprocessing import TransactionEncoder

def analyze_association_rules_real(frequent_itemsets, metric="lift",
min_threshold=1.2):
    # Generate association rules
    rules = association_rules(frequent_itemsets, metric=metric,
min_threshold=min_threshold)
    rules["antecedents_length"] = rules["antecedents"].apply(lambda x:
len(x))
    rules["consequents_length"] = rules["consequents"].apply(lambda x:
len(x))
    return rules.sort_values("lift", ascending=False)

def analyze_topic_keywords_real(topic_keywords):
    # Print the top words for each topic
    for idx, keywords in enumerate(topic_keywords):
        print(f"Topic {idx + 1} Keywords:")
        print(keywords)
        print("\n")

def analyze_topic_distribution_real(lda_model, text_data):
    # Get the topic distribution for each document
    topic_distribution = lda_model.transform(text_data)

    # Analyze the distribution of documents across topics
    print("Topic Distribution:")

print(pd.DataFrame(topic_distribution).idxmax(axis=1).value_counts())
    print("\n")



# Vectorize the text data
vectorizer = CountVectorizer(max_features=1000, stop_words='english')
X = vectorizer.fit_transform(real_news['article'])

# Fit LDA model
lda = LatentDirichletAllocation(n_components=10, random_state=42)
lda.fit(X)

# Extract frequent itemsets from topics
feature_names = np.array(vectorizer.get_feature_names_out())
topic_keywords = []
for topic_idx, topic in enumerate(lda.components_):
    top_words_idx = topic.argsort()[:-10 - 1:-1]
    top_words = feature_names[top_words_idx]
    topic_keywords.append(top_words)
```

```
# Convert itemsets to DataFrame
te = TransactionEncoder()
te_ary = te.fit(topic_keywords).transform(topic_keywords)
df_itemsets = pd.DataFrame(te_ary, columns=te.columns_)

# Generate frequent itemsets
frequent_itemsets = apriori(df_itemsets, min_support=0.1,
use_colnames=True)

# Analyze association rules
rules_real = analyze_association_rules_real(frequent_itemsets,
metric="lift", min_threshold=1.2)

# Analyze topic keywords
analyze_topic_keywords_real(topic_keywords)

# Analyze topic distribution
analyze_topic_distribution_real(lda, X)


print(rules_real)
```

```
Topic 1 Keywords:
['clinton' 'sander' 'voter' 'state' 'trump' 'vote' 'party' 'percent'
'say'
 'democratic']


Topic 2 Keywords:
['trump' 'say' 'cruz' 'republican' 'bush' 'campaign' 'rubio'
'candidate'
 'donald' 'debate']


Topic 3 Keywords:
['say' 'police' 'people' 'officer' 'gun' 'report' 'kill' 'told' 'city'
 'black']


Topic 4 Keywords:
['people' 'say' 'trump' 'like' 'make' 'american' 'think' 'dont' 'time'
 'woman']


Topic 5 Keywords:
['court' 'say' 'state' 'law' 'marriage' 'supreme' 'justice' 'right'
'case'
 'religious']


Topic 6 Keywords:
```

```
['house' 'say' 'republican' 'congress' 'obama' 'president' 'senate'
'vote'
 'democrat' 'year']


Topic 7 Keywords:
['year' 'percent' 'tax' 'say' 'state' 'million' 'job' 'american'
'rate'
 'health']


Topic 8 Keywords:
['republican' 'party' 'say' 'candidate' 'run' 'president'
'presidential'
 'gop' 'campaign' 'senate']


Topic 9 Keywords:
['clinton' 'say' 'email' 'campaign' 'state' 'hillary' 'department'
 'secretary' 'use' 'president']


Topic 10 Keywords:
['say' 'state' 'iran' 'obama' 'president' 'isi' 'islamic' 'military'
 'deal' 'attack']


Topic Distribution:
1    478
3    413
9    406
0    398
5    335
2    305
7    258
6    225
8    191
4    145
Name: count, dtype: int64


                                      antecedents  \
279813         (candidate, campaign, cruz, rubio)
369540  (state, secretary, president, department)
369538        (state, use, president, department)
369537       (secretary, use, hillary, president)
369536     (secretary, state, hillary, president)
...                                          ...
2369                                  (president)
55206           (candidate, campaign, republican)
```

```
2062                         (republican, campaign)
55199      (candidate, campaign, say, republican)
593                                          (year)

                         consequents  antecedent support  consequent
support  \
279813           (bush, say, trump)                 0.1
0.1
369540         (email, use, hillary)                 0.1
0.1
369538  (secretary, email, hillary)                 0.1
0.1
369537   (email, state, department)                 0.1
0.1
369536     (email, use, department)                 0.1
0.1
...                              ...                 ...
...
2369         (republican, candidate)                 0.4
0.2
55206               (say, president)                 0.2
0.4
2062                     (president)                 0.2
0.4
55199                    (president)                 0.2
0.4
593                      (president)                 0.2
0.4

        support  confidence   lift  leverage  conviction
zhangs_metric  \
279813      0.1        1.00  10.00      0.09         inf
1.000000
369540      0.1        1.00  10.00      0.09         inf
1.000000
369538      0.1        1.00  10.00      0.09         inf
1.000000
369537      0.1        1.00  10.00      0.09         inf
1.000000
369536      0.1        1.00  10.00      0.09         inf
1.000000
...         ...         ...    ...       ...         ...             ..
.
2369        0.1        0.25   1.25      0.02    1.066667
0.333333
55206       0.1        0.50   1.25      0.02    1.200000
0.250000
2062        0.1        0.50   1.25      0.02    1.200000
0.250000
```

```
55199       0.1        0.50   1.25      0.02    1.200000
0.250000
593         0.1        0.50   1.25      0.02    1.200000
0.250000

        antecedents_length  consequents_length
279813                   4                   3
369540                   4                   3
369538                   4                   3
369537                   4                   3
369536                   4                   3
...                    ...                 ...
2369                     1                   2
55206                    3                   2
2062                     2                   1
55199                    4                   1
593                      1                   1

[559626 rows x 12 columns]
```

Extract and sort rules from the `rules_real` DataFrame that contain "say" in the antecedents, ordered by highest confidence in descending order.

```
say_r_rules =rules_real[rules_real["antecedents"].str.contains("say",
regex=False) ].sort_values("confidence", ascending=False)


r_rules = say_r_rules[['antecedents','consequents','support',
'confidence', 'lift']]
r_rules.head(50)
```

```
                                        antecedents  \
369457                                   (use, say)
159358            (campaign, hillary, say, president)
159313                   (campaign, say, department)
159312                  (say, president, department)
159311                     (secretary, hillary, say)
159310                      (campaign, hillary, say)
159309                     (hillary, say, president)
159308                    (hillary, say, department)
159302          (campaign, secretary, say, president)
159301         (campaign, secretary, say, department)
159373                  (say, president, department)
159114                   (campaign, email, use, say)
159372                      (state, say, department)
159371                      (campaign, hillary, say)
159370                     (hillary, say, president)
159369                        (state, hillary, say)
159368                    (hillary, say, department)
159362               (campaign, state, say, president)
```

```
159361         (campaign, say, president, department)
159360             (campaign, state, say, department)
159314                   (secretary, say, department)
159316                    (secretary, say, president)
159328                                 (hillary, say)
159329                              (say, department)
159112             (campaign, email, say, department)
159111                  (email, use, say, department)
159110    (say, department, use, campaign, secretary)
159109         (say, email, use, campaign, secretary)
159108  (say, email, department, campaign, secretary)
159106        (say, email, department, use, campaign)
159132                   (secretary, say, department)
159133                          (campaign, use, say)
159134                        (use, secretary, say)
159135                    (campaign, secretary, say)
159150                             (secretary, say)
159148                                   (use, say)
159147                              (say, department)
159146                                  (email, say)
159376                         (campaign, state, say)
159317                    (campaign, secretary, say)
159332                             (secretary, say)
159359            (state, say, president, department)
159357                (campaign, state, hillary, say)
159356               (state, hillary, say, president)
159066                        (email, say, department)
159059         (campaign, secretary, say, department)
159058            (state, secretary, say, department)
159057             (campaign, state, say, department)
159056              (campaign, email, secretary, say)
159055                 (email, state, secretary, say)

                                              consequents  support
confidence  \
369457  (email, hillary, department, state, president)      0.1
1.0
159358                               (state, department)      0.1
1.0
159313                  (secretary, hillary, president)      0.1
1.0
159312                   (campaign, secretary, hillary)      0.1
1.0
159311                (campaign, president, department)      0.1
1.0
159310               (secretary, president, department)      0.1
1.0
159309                (campaign, secretary, department)      0.1
1.0
```

```
159308                (campaign, secretary, president)      0.1
1.0
159302                           (hillary, department)      0.1
1.0
159301                            (hillary, president)      0.1
1.0
159373                      (campaign, state, hillary)      0.1
1.0
159114                         (secretary, department)      0.1
1.0
159372                  (campaign, hillary, president)      0.1
1.0
159371                  (state, president, department)      0.1
1.0
159370                   (campaign, state, department)      0.1
1.0
159369                (campaign, president, department)      0.1
1.0
159368                   (campaign, state, president)      0.1
1.0
159362                           (hillary, department)      0.1
1.0
159361                                (state, hillary)      0.1
1.0
159360                            (hillary, president)      0.1
1.0
159314                  (campaign, hillary, president)      0.1
1.0
159316                (campaign, hillary, department)      0.1
1.0
159328    (campaign, secretary, president, department)      0.1
1.0
159329       (campaign, secretary, hillary, president)      0.1
1.0
159112                                (use, secretary)      0.1
1.0
159111                           (campaign, secretary)      0.1
1.0
159110                                       (email)      0.1
1.0
159109                                  (department)      0.1
1.0
159108                                         (use)      0.1
1.0
159106                                   (secretary)      0.1
1.0
159132                           (campaign, email, use)      0.1
1.0
159133                  (email, secretary, department)      0.1
```

```
1.0
159134                    (campaign, email, department)      0.1
1.0
159135                         (email, use, department)      0.1
1.0
159150               (campaign, email, use, department)      0.1
1.0
159148         (campaign, email, secretary, department)      0.1
1.0
159147                (campaign, email, use, secretary)      0.1
1.0
159146          (campaign, use, secretary, department)      0.1
1.0
159376                 (hillary, president, department)      0.1
1.0
159317                 (hillary, president, department)      0.1
1.0
159332      (campaign, hillary, president, department)      0.1
1.0
159359                             (campaign, hillary)      0.1
1.0
159357                          (president, department)      0.1
1.0
159356                           (campaign, department)      0.1
1.0
159066                    (campaign, state, secretary)      0.1
1.0
159059                                (email, state)      0.1
1.0
159058                              (campaign, email)      0.1
1.0
159057                               (email, secretary)      0.1
1.0
159056                             (state, department)      0.1
1.0
159055                           (campaign, department)      0.1
1.0

        lift
369457  10.0
159358  10.0
159313  10.0
159312  10.0
159311  10.0
159310  10.0
159309  10.0
159308  10.0
159302  10.0
159301  10.0
```

```
159373  10.0
159114  10.0
159372  10.0
159371  10.0
159370  10.0
159369  10.0
159368  10.0
159362  10.0
159361  10.0
159360  10.0
159314  10.0
159316  10.0
159328  10.0
159329  10.0
159112  10.0
159111  10.0
159110  10.0
159109  10.0
159108  10.0
159106  10.0
159132  10.0
159133  10.0
159134  10.0
159135  10.0
159150  10.0
159148  10.0
159147  10.0
159146  10.0
159376  10.0
159317  10.0
159332  10.0
159359  10.0
159357  10.0
159356  10.0
159066  10.0
159059  10.0
159058  10.0
159057  10.0
159056  10.0
159055  10.0
```

Sort the rules in the `rules_real` DataFrame by confidence in descending order and assign the sorted DataFrame to the variable `r`.

```
r=rules_real.sort_values("confidence",ascending=False)

r1_rules =r[['antecedents','consequents','support', 'confidence',
'lift']]
r1_rules.head(50)
```

	antecedents	consequents
279813	(candidate, campaign, cruz, rubio)	(bush, say, trump)
10322	(attack, military)	(obama, state)
10427	(black, kill)	(city, gun)
10428	(city)	(gun, black, kill)
10429	(gun)	(city, black, kill)
10430	(black)	(city, gun, kill)
10431	(kill)	(city, gun, black)
10432	(city, gun, black)	(officer)
10433	(city, gun, officer)	(black)
10434	(city, black, officer)	(gun)
10422	(city, gun)	(black, kill)
10557	(city, report, kill)	(black)
10364	(military, say)	(attack, state)
10360	(attack, say)	(military, state)
10321	(attack, state)	(military, obama)
10323	(obama, state)	(attack, military)
10425	(gun, black)	(city, kill)
10324	(military, obama)	(attack, state)
10325	(military, state)	(attack, obama)
10326	(attack)	(military, obama, state)
10329	(military)	(attack, obama, state)
10330	(attack, say, president)	(military)
10332	(military, say, president)	(attack)
10333	(attack, say)	(military, president)
10334	(attack, president)	(military, say)

```
10337                     (military, say)        (attack, president)
10363                   (military, state)              (attack, say)
10338                (military, president)              (attack, say)
10341                           (military)   (attack, say, president)
10342          (attack, state, president)                 (military)
10426                        (gun, kill)              (city, black)
10424                       (city, kill)               (gun, black)
10346                    (attack, state)      (military, president)
10320                    (attack, obama)          (military, state)
10253             (attack, say, president)                  (islamic)
10254            (islamic, say, president)                   (attack)
10256                       (attack, say)       (islamic, president)
10257                 (attack, president)             (islamic, say)
10258                      (islamic, say)        (attack, president)
10259                (islamic, president)              (attack, say)
10261                             (attack)  (islamic, say, president)
10262                           (islamic)   (attack, say, president)
10265           (attack, state, president)                  (islamic)
10267          (islamic, state, president)                   (attack)
10247                     (islamic, obama)            (attack, state)
10316                (attack, obama, state)                 (military)
10319             (military, obama, state)                   (attack)
10367                           (military)       (attack, state, say)
10423                       (city, black)                (gun, kill)
10383            (state, president, obama)                   (attack)

        support  confidence  lift
```

```
279813      0.1          1.0  10.0
10322       0.1          1.0  10.0
10427       0.1          1.0  10.0
10428       0.1          1.0  10.0
10429       0.1          1.0  10.0
10430       0.1          1.0  10.0
10431       0.1          1.0  10.0
10432       0.1          1.0  10.0
10433       0.1          1.0  10.0
10434       0.1          1.0  10.0
10422       0.1          1.0  10.0
10557       0.1          1.0  10.0
10364       0.1          1.0  10.0
10360       0.1          1.0  10.0
10321       0.1          1.0  10.0
10323       0.1          1.0  10.0
10425       0.1          1.0  10.0
10324       0.1          1.0  10.0
10325       0.1          1.0  10.0
10326       0.1          1.0  10.0
10329       0.1          1.0  10.0
10330       0.1          1.0  10.0
10332       0.1          1.0  10.0
10333       0.1          1.0  10.0
10334       0.1          1.0  10.0
10337       0.1          1.0  10.0
10363       0.1          1.0  10.0
10338       0.1          1.0  10.0
10341       0.1          1.0  10.0
10342       0.1          1.0  10.0
10426       0.1          1.0  10.0
10424       0.1          1.0  10.0
10346       0.1          1.0  10.0
10320       0.1          1.0  10.0
10253       0.1          1.0  10.0
10254       0.1          1.0  10.0
10256       0.1          1.0  10.0
10257       0.1          1.0  10.0
10258       0.1          1.0  10.0
10259       0.1          1.0  10.0
10261       0.1          1.0  10.0
10262       0.1          1.0  10.0
10265       0.1          1.0  10.0
10267       0.1          1.0  10.0
10247       0.1          1.0  10.0
10316       0.1          1.0  10.0
10319       0.1          1.0  10.0
10367       0.1          1.0  10.0
```

```
10423       0.1         1.0  10.0
10383       0.1         1.0  10.0
```

Extract and sort rules from the `rules_real` DataFrame where the antecedents contain "clinton", ordering them by highest confidence in descending order.

```
clinton_r_rules
=rules_real[rules_real["antecedents"].str.contains("clinton",
regex=False) ].sort_values("confidence", ascending=False)


r3_rules = clinton_r_rules[['antecedents','consequents','support',
'confidence', 'lift']]
r3_rules.head(50)
```

```
                                          antecedents  \
344375                     (state, clinton, say, trump)
154404                           (clinton, department)
154613                 (use, clinton, say, department)
154611                (state, clinton, say, department)
154608              (say, use, state, campaign, clinton)
154606      (say, department, state, campaign, clinton)
154598                           (clinton, secretary)
154596                            (campaign, clinton)
154201                       (clinton, say, department)
154172                           (clinton, secretary)
154197             (campaign, state, clinton, president)
154192               (campaign, clinton, say, president)
154191                   (campaign, state, clinton, say)
154190                  (state, clinton, say, president)
154187             (clinton, say, president, department)
154182       (say, state, president, campaign, clinton)
154614              (campaign, clinton, say, department)
154616                       (use, state, clinton, say)
154617                   (campaign, state, clinton, say)
154618                     (campaign, use, clinton, say)
154621            (campaign, state, clinton, department)
154623                   (campaign, state, clinton, use)
154627                       (clinton, say, department)
154594                                   (use, clinton)
154591                           (clinton, department)
154563              (campaign, secretary, use, clinton)
154566                       (clinton, say, department)
154569                              (use, clinton, say)
154571                        (campaign, clinton, say)
154573                       (clinton, secretary, say)
154575                       (use, clinton, department)
154577                   (campaign, clinton, department)
```

```
154579                  (clinton, secretary, department)
154207                         (campaign, clinton, say)
154170                              (campaign, clinton)
154151                   (campaign, clinton, department)
154096                         (campaign, use, clinton)
154157                    (campaign, clinton, secretary)
154156                   (clinton, secretary, president)
154111                              (campaign, clinton)
154110                                   (use, clinton)
154105                            (clinton, department)
154101                               (clinton, hillary)
154095                          (campaign, state, clinton)
154168                               (clinton, president)
154094                            (state, clinton, use)
154092                   (campaign, clinton, department)
154091                        (use, clinton, department)
154089                     (state, clinton, department)
154118  (say, department, president, campaign, clinton)

                                    consequents  support  confidence
lift
344375                    (voter, percent, vote)      0.1         1.0
10.0
154404    (campaign, use, secretary, president)      0.1         1.0
10.0
154613                         (campaign, state)      0.1         1.0
10.0
154611                           (campaign, use)      0.1         1.0
10.0
154608                              (department)      0.1         1.0
10.0
154606                                   (use)      0.1         1.0
10.0
154598         (campaign, use, say, department)      0.1         1.0
10.0
154596        (use, secretary, say, department)      0.1         1.0
10.0
154201             (campaign, state, president)      0.1         1.0
10.0
154172   (campaign, say, president, department)      0.1         1.0
10.0
154197                         (say, department)      0.1         1.0
10.0
154192                        (state, department)      0.1         1.0
10.0
154191                   (president, department)      0.1         1.0
10.0
154190                    (campaign, department)      0.1         1.0
10.0
```

```
154187                          (campaign, state)      0.1          1.0
10.0
154182                              (department)      0.1          1.0
10.0
154614                              (state, use)      0.1          1.0
10.0
154616                    (campaign, department)      0.1          1.0
10.0
154617                          (use, department)      0.1          1.0
10.0
154618                        (state, department)      0.1          1.0
10.0
154621                                 (use, say)      0.1          1.0
10.0
154623                          (say, department)      0.1          1.0
10.0
154627                    (campaign, state, use)      0.1          1.0
10.0
154594   (campaign, secretary, say, department)      0.1          1.0
10.0
154591          (campaign, use, secretary, say)      0.1          1.0
10.0
154563                          (say, department)      0.1          1.0
10.0
154566                (campaign, use, secretary)      0.1          1.0
10.0
154569         (campaign, secretary, department)      0.1          1.0
10.0
154571              (use, secretary, department)      0.1          1.0
10.0
154573               (campaign, use, department)      0.1          1.0
10.0
154575                (campaign, secretary, say)      0.1          1.0
10.0
154577                     (use, secretary, say)      0.1          1.0
10.0
154579                      (campaign, use, say)      0.1          1.0
10.0
154207           (state, president, department)      0.1          1.0
10.0
154170  (secretary, say, president, department)      0.1          1.0
10.0
154151               (secretary, say, president)      0.1          1.0
10.0
154096              (state, hillary, department)      0.1          1.0
10.0
154157              (say, president, department)      0.1          1.0
10.0
154156               (campaign, say, department)      0.1          1.0
```

```
10.0
154111        (state, hillary, use, department)      0.1          1.0
10.0
154110   (campaign, state, hillary, department)      0.1          1.0
10.0
154105          (campaign, state, hillary, use)      0.1          1.0
10.0
154101       (campaign, state, use, department)      0.1          1.0
10.0
154095               (use, hillary, department)      0.1          1.0
10.0
154168   (campaign, secretary, say, department)      0.1          1.0
10.0
154094          (campaign, hillary, department)      0.1          1.0
10.0
154092                   (state, hillary, use)      0.1          1.0
10.0
154091               (campaign, state, hillary)      0.1          1.0
10.0
154089                 (campaign, use, hillary)      0.1          1.0
10.0
154118                              (secretary)      0.1          1.0
10.0
```

Summary of Association Rules with Topic Modeling

Context and Goal

The discussion focused on extracting and comparing association rules containing "say" from real and fake news datasets, aiming to identify top patterns and themes in each.

Key Findings

1. **Real News Dataset**:
   - **Rules**: Included terms like "use," "campaign," "hillary," "president," "secretary," "state," and "department."
   - **Example**: (use, say) → (email, hillary, department, state, president)
   - **Themes**: Primarily political, focusing on Hillary Clinton and governmental activities.
   - **Strength**: High confidence and lift values indicate strong and reliable associations within the political theme.
2. **Fake News Dataset**:
   - **Rules**: Included diverse terms like "report," "stand," "police," "black," "water," and "use."
   - **Example**: (think, make, say) → (thing, people, time, world)
   - **Themes**: Varied topics, often sensational or controversial, covering social justice, law enforcement, and racial issues.
   - **Strength**: Despite thematic diversity, high confidence and lift values show strong associations.

Conclusion

Real news focuses on political discourse, while fake news spans a range of socially charged topics. Both datasets exhibit strong and reliable associations, reflecting their thematic focuses.

Assiontion rules without topic modelling

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.feature_extraction.text import CountVectorizer
from mlxtend.preprocessing import TransactionEncoder
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

# Vectorize the text data
vectorizer = CountVectorizer(max_features=1000, stop_words='english',
binary=True)
X = vectorizer.fit_transform(real_news['article_l'])

# Convert sparse matrix to DataFrame
df = pd.DataFrame(X.toarray(),
columns=vectorizer.get_feature_names_out())

# Generate frequent itemsets
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)

# Function to analyze association rules
def analyze_association_rules(frequent_itemsets, metric="lift",
min_threshold=1.2):
    rules = association_rules(frequent_itemsets, metric=metric,
min_threshold=min_threshold)
    rules["antecedents_length"] = rules["antecedents"].apply(lambda x:
len(x))
    rules["consequents_length"] = rules["consequents"].apply(lambda x:
len(x))
    return rules.sort_values("lift", ascending=False)

# Analyze association rules
rules_real = analyze_association_rules(frequent_itemsets,
metric="lift", min_threshold=1.2)
```


```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.feature_extraction.text import CountVectorizer
from mlxtend.preprocessing import TransactionEncoder
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
```

```
# Vectorize the text data
vectorizer = CountVectorizer(max_features=1000, stop_words='english',
binary=True)
X = vectorizer.fit_transform(fake_news['article_l'])

# Convert sparse matrix to DataFrame
df = pd.DataFrame(X.toarray(),
columns=vectorizer.get_feature_names_out())

# Generate frequent itemsets
frequent_itemsets = apriori(df, min_support=0.1, use_colnames=True)

# Function to analyze association rules
def analyze_association_rules(frequent_itemsets, metric="lift",
min_threshold=1.2):
    rules = association_rules(frequent_itemsets, metric=metric,
min_threshold=min_threshold)
    rules["antecedents_length"] = rules["antecedents"].apply(lambda x:
len(x))
    rules["consequents_length"] = rules["consequents"].apply(lambda x:
len(x))
    return rules.sort_values("lift", ascending=False)

# Analyze association rules
rules_fake = analyze_association_rules(frequent_itemsets,
metric="lift", min_threshold=1.2)

import pandas as pd

# Load our data
rules_fake =
pd.read_csv('/Users/priyamadhurigattem/Desktop/DM_final/fake_ass.csv')
rules_real =
pd.read_csv('/Users/priyamadhurigattem/Desktop/DM_final/real_ass.csv')

# Convert frozenset to string for readability
rules_fake['antecedents'] = rules_fake['antecedents'].apply(lambda x:
', '.join(eval(x)))
rules_fake['consequents'] = rules_fake['consequents'].apply(lambda x:
', '.join(eval(x)))

rules_real['antecedents'] = rules_real['antecedents'].apply(lambda x:
', '.join(eval(x)))
rules_real['consequents'] = rules_real['consequents'].apply(lambda x:
', '.join(eval(x)))
```

Extract and sort association rules from `rules_real` DataFrame with antecedents containing "clinton", sorted by highest confidence.

```
ass_clinton_rules
=rules_real[rules_real["antecedents"].str.contains("clinton",
regex=False) ].sort_values("confidence", ascending=False)


ass_rules = ass_clinton_rules[['antecedents','consequents','support',
'confidence', 'lift']]
ass_rules.head(50)
```

```
                                    antecedents       consequents
support  \
56721  (republican, donald, clinton, hillary)           (trump)
0.203868
24981              (donald, clinton, hillary)           (trump)
0.233989
56750         (donald, clinton, hillary, say)           (trump)
0.220038
25020           (republican, donald, clinton)           (trump)
0.214014
19239             (donald, clinton, campaign)           (trump)
0.207356
24995                 (donald, clinton, make)           (trump)
0.207039
56778      (republican, donald, clinton, say)           (trump)
0.201649
4976                        (donald, clinton)           (trump)
0.246354
25034                  (donald, clinton, say)           (trump)
0.231452
24904            (state, democratic, clinton)         (hillary)
0.209575
19212         (democratic, clinton, campaign)         (hillary)
0.208624
24892              (democratic, clinton, say)         (hillary)
0.233354
5066                         (clinton, voter)         (hillary)
0.206405
52186     (candidate, clinton, campaign, say)         (hillary)
0.216867
56718    (republican, donald, clinton, trump)         (hillary)
0.203868
24866             (democratic, clinton, make)         (hillary)
0.207990
55624        (state, candidate, clinton, say)         (hillary)
0.207990
24954           (republican, donald, clinton)         (hillary)
0.204502
```

```
52892        (state, clinton, campaign, make)          (hillary)
0.203234
22822      (candidate, clinton, presidential)          (hillary)
0.209258
52978  (clinton, presidential, campaign, say)          (hillary)
0.208624
56748           (donald, clinton, trump, say)          (hillary)
0.220038
22862             (candidate, clinton, state)          (hillary)
0.218136
24980                (donald, clinton, trump)          (hillary)
0.233989
24968                  (donald, clinton, say)          (hillary)
0.220989
18536          (candidate, clinton, campaign)          (hillary)
0.226379
53034         (state, clinton, campaign, say)          (hillary)
0.225745
4950                        (donald, clinton)          (hillary)
0.234940
24878       (republican, democratic, clinton)          (hillary)
0.205136
56726           (republican, donald, clinton)  (trump, hillary)
0.203868
57224       (republican, clinton, trump, say)          (hillary)
0.203234
4912                     (democratic, clinton)          (hillary)
0.248573
22808         (candidate, clinton, president)          (hillary)
0.201332
25520                 (clinton, state, trump)          (hillary)
0.200698
19306       (clinton, presidential, campaign)          (hillary)
0.217819
56719   (republican, clinton, trump, hillary)           (donald)
0.203868
25291                   (party, clinton, say)          (hillary)
0.214014
55596   (republican, candidate, clinton, say)          (hillary)
0.212112
55568         (candidate, clinton, make, say)          (hillary)
0.211795
22850               (candidate, clinton, say)          (hillary)
0.249524
25454            (republican, clinton, trump)          (hillary)
0.215282
56753                  (donald, clinton, say)  (trump, hillary)
0.220038
19346              (clinton, state, campaign)          (hillary)
```

```
0.236525
24984                        (donald, clinton)  (trump, hillary)
0.233989
52864          (clinton, make, campaign, say)         (hillary)
0.228282
52950     (president, clinton, campaign, say)         (hillary)
0.206405
22782               (candidate, clinton, make)         (hillary)
0.220989
56749          (clinton, trump, hillary, say)          (donald)
0.220038
53062          (clinton, time, campaign, say)         (hillary)
0.207990
25402          (clinton, state, presidential)         (hillary)
0.217185

       confidence      lift
56721    0.996899  2.517390
24981    0.995951  2.514997
56750    0.995696  2.514351
25020    0.995575  2.514047
19239    0.995434  2.513689
24995    0.995427  2.513672
56778    0.995305  2.513365
4976     0.994878  2.512287
25034    0.994550  2.511459
24904    0.960756  2.653436
19212    0.956395  2.641393
24892    0.954604  2.636447
5066     0.954545  2.636284
52186    0.952646  2.631039
56718    0.952593  2.630891
24866    0.952104  2.629543
55624    0.952104  2.629543
24954    0.951327  2.627396
52892    0.951039  2.626599
22822    0.951009  2.626516
52978    0.950867  2.626125
56748    0.950685  2.625622
22862    0.950276  2.624493
24980    0.949807  2.623197
24968    0.949591  2.622601
18536    0.949468  2.622261
53034    0.949333  2.621889
4950     0.948784  2.620371
24878    0.948680  2.620086
56726    0.948378  3.776746
57224    0.948225  2.618828
4912     0.948005  2.618220
```

```
22808    0.947761  2.617547
25520    0.947605  2.617115
19306    0.947586  2.617064
56719    0.946981  2.631522
25291    0.946704  2.614628
55596    0.946252  2.613378
55568    0.946176  2.613168
22850    0.945913  2.612444
25454    0.945682  2.611806
56753    0.945504  3.765303
19346    0.945501  2.611304
24984    0.944942  3.763066
52864    0.944882  2.609595
52950    0.944848  2.609500
22782    0.944444  2.608387
56749    0.944218  2.623844
53062    0.943885  2.606841
25402    0.943526  2.605851
```

Extract and sort association rules from the `rules_fake` DataFrame where the antecedents contain "clinton", sorted by highest confidence.

```
clinton_f_rules
=rules_fake[rules_fake["antecedents"].str.contains("clinton",
regex=False) ].sort_values("confidence", ascending=False)


f1_rules = clinton_f_rules[['antecedents','consequents','support',
'confidence', 'lift']]
f1_rules.head(50)
```

```
                                      antecedents       consequents
support  \
47267             (clinton, presidential, donald)           (trump)
0.103404
46382                   (clinton, donald, come)           (trump)
0.100193
47250                (clinton, donald, president)           (trump)
0.113038
89502       (hillary, clinton, donald, president)           (trump)
0.103404
89412          (state, donald, clinton, election)           (trump)
0.105010
47068                (clinton, donald, election)           (trump)
0.133911
47306                   (clinton, donald, state)           (trump)
0.126846
89352        (hillary, clinton, donald, election)           (trump)
```

```
0.124920
89562           (hillary, state, donald, clinton)           (trump)
0.119461
47320                     (clinton, donald, time)           (trump)
0.113680
47236                   (people, clinton, donald)           (trump)
0.113359
45220                  (campaign, clinton, donald)           (trump)
0.110469
89382            (clinton, donald, say, election)           (trump)
0.107900
89592            (hillary, clinton, donald, time)           (trump)
0.105973
89472          (hillary, people, clinton, donald)           (trump)
0.105973
47222                      (new, donald, clinton)           (trump)
0.105331
89622               (state, donald, say, clinton)           (trump)
0.105331
39102                 (american, clinton, donald)           (trump)
0.103083
88902        (hillary, campaign, clinton, donald)           (trump)
0.101798
108072  (donald, say, hillary, election, clinton)           (trump)
0.100835
47180                     (clinton, donald, know)           (trump)
0.100514
12350                           (clinton, donald)           (trump)
0.177585
47166                  (clinton, hillary, donald)           (trump)
0.165061
47292                      (clinton, donald, say)           (trump)
0.142582
89532            (hillary, clinton, donald, say)           (trump)
0.132627
47208                     (clinton, make, donald)           (trump)
0.114965
89442            (hillary, clinton, make, donald)           (trump)
0.105652
37344                     (2016, clinton, donald)           (trump)
0.102119
47194                     (like, clinton, donald)           (trump)
0.101477
13639                           (clinton, united)            (state)
0.107900
108138     (say, election, state, clinton, trump)          (hillary)
0.101477
47141                    (clinton, donald, state)          (hillary)
0.119782
```

```
89565              (trump, state, donald, clinton)         (hillary)
0.119461
89570                    (state, donald, clinton)  (hillary, trump)
0.119461
91276          (state, clinton, president, trump)         (hillary)
0.107579
91335                (state, clinton, say, trump)         (hillary)
0.125562
47099                   (people, clinton, donald)         (hillary)
0.106294
45389           (campaign, clinton, presidential)         (hillary)
0.105973
89474            (people, clinton, donald, trump)         (hillary)
0.105973
89324            (clinton, donald, say, election)         (hillary)
0.101156
108075    (donald, say, election, clinton, trump)         (hillary)
0.100835
89985    (trump, clinton, presidential, election)         (hillary)
0.100193
91155             (people, state, clinton, trump)         (hillary)
0.107900
47029                (clinton, donald, election)         (hillary)
0.125241
48105                    (email, clinton, trump)         (hillary)
0.102762
89355          (trump, clinton, donald, election)         (hillary)
0.124920
47155                    (clinton, donald, time)         (hillary)
0.106294
90136           (trump, state, clinton, election)         (hillary)
0.123635
89478                   (people, clinton, donald)  (hillary, trump)
0.105973
89595              (clinton, donald, time, trump)         (hillary)
0.105973

        confidence      lift
47267     1.000000  2.899441
46382     1.000000  2.899441
47250     1.000000  2.899441
89502     1.000000  2.899441
89412     1.000000  2.899441
47068     0.997608  2.892505
47306     0.997475  2.892120
89352     0.997436  2.892007
89562     0.997319  2.891668
47320     0.997183  2.891274
47236     0.997175  2.891251
```

```
45220     0.997101  2.891037
89382     0.997033  2.890838
89592     0.996979  2.890682
89472     0.996979  2.890682
47222     0.996960  2.890628
89622     0.996960  2.890628
39102     0.996894  2.890437
88902     0.996855  2.890324
108072    0.996825  2.890237
47180     0.996815  2.890207
12350     0.996396  2.888993
47166     0.996124  2.888203
47292     0.995516  2.886439
89532     0.995181  2.885468
47208     0.994444  2.883333
89442     0.993958  2.881922
37344     0.993750  2.881320
47194     0.993711  2.881206
13639     0.965517  2.039770
108138    0.948949  2.605844
47141     0.941919  2.586540
89565     0.941772  2.586136
89570     0.939394  3.958420
91276     0.935754  2.569611
91335     0.935407  2.568656
47099     0.935028  2.567617
45389     0.934844  2.567112
89474     0.934844  2.567112
89324     0.934718  2.566766
108075    0.934524  2.566232
89985     0.934132  2.565155
91155     0.933333  2.562963
47029     0.933014  2.562087
48105     0.932945  2.561896
89355     0.932854  2.561646
47155     0.932394  2.560385
90136     0.932203  2.559860
89478     0.932203  3.928121
89595     0.932203  2.559860
```

Extract and sort association rules from `rules_real` DataFrame with antecedents containing "dont", sorted by highest confidence.

```
ass_dont_rules
=rules_real[rules_real["antecedents"].str.contains("dont",
regex=False) ].sort_values("confidence", ascending=False)


ass_rules = ass_dont_rules[['antecedents','consequents','support',
```

```
'confidence', 'lift']]
ass_rules.head(50)
```

	antecedents	consequents	support	confidence	lift
53238	like, state, dont	make	0.206405	0.886921	1.267489
53760	state, time, dont, say	make	0.204502	0.885989	1.266157
54791	like, time, dont	make	0.205453	0.884038	1.263369
56273	state, dont, year	make	0.204819	0.881310	1.259470
58486	state, time, dont	make	0.215282	0.877261	1.253684
58517	time, dont, year	make	0.203868	0.877217	1.253621
60675	work, dont	make	0.207673	0.873333	1.248071
61093	campaign, say, dont	make	0.201966	0.872603	1.247027
61193	new, time, dont, say	make	0.201649	0.872428	1.246777
61508	like, people, dont	make	0.202600	0.871760	1.245823
61521	people, state, dont	make	0.211160	0.871728	1.245777
61550	dont, way, say	make	0.206722	0.871658	1.245677
61827	want, say, dont	make	0.214331	0.871134	1.244928
61867	people, state, say, dont	make	0.201332	0.871056	1.244817
62229	people, time, dont	make	0.210843	0.870419	1.243906
63054	dont, way	make	0.218453	0.868852	1.241668
63314	say, dont, year	make	0.234306	0.868390	1.241007
63807	want, dont	make	0.221940	0.867410	1.239607
64046	campaign, dont	make	0.210843	0.867014	1.239041
64682	new, time, dont	make	0.212746	0.865806	1.237315
64938	new, state, dont	make	0.207673	0.865258	1.236530
65153	like, say, dont	make	0.233354	0.864865	1.235969
66996	dont, year	make	0.247939	0.861233	

```
1.230780
67646         people, new, dont        make  0.200380    0.859864
1.228822
67854                like, dont        make  0.248256    0.859495
1.228295
68013           time, dont, say        make  0.249524    0.859170
1.227831
68512          state, say, dont        make  0.255231    0.858209
1.226457
69155      dont, president, say        make  0.227647    0.856802
1.224446
69725            american, dont        make  0.202600    0.855422
1.222474
70039                time, dont        make  0.265060    0.854806
1.221594
71558           come, dont, say        make  0.233672    0.851039
1.216211
72157           dont, president        make  0.236208    0.849487
1.213993
72344               state, dont        make  0.267280    0.848943
1.213215
72395     republican, say, dont        make  0.220672    0.848780
1.212983
72653              people, dont        make  0.260304    0.848140
1.212068
72658           know, dont, say        make  0.208941    0.848134
1.212059
72961                come, dont        make  0.242866    0.847345
1.210932
73769         people, say, dont        make  0.245720    0.845147
1.207791
74082                know, dont        make  0.216550    0.844252
1.206512
74902            new, say, dont        make  0.241281    0.841814
1.203028
75088               think, dont        make  0.226696    0.841176
1.202116
75352          think, say, dont        make  0.218770    0.840438
1.201062
31384           new, make, dont        time  0.212746    0.838750
1.339452
43654                want, dont   make, say  0.214331    0.837670
1.294470
32308          new, state, dont        time  0.200698    0.836196
1.335372
32470      new, make, say, dont        time  0.201649    0.835742
1.334649
45857         state, time, dont   make, say  0.204502    0.833333
1.287767
```

```
66457          like, make, dont       state  0.206405    0.831418
1.232280
47050       people, state, dont   make, say  0.201332    0.831152
1.284396
47398            campaign, dont   make, say  0.201966    0.830508
1.283402
```

Extract and sort association rules from `rules_fake` DataFrame with antecedents containing "dont", sorted by highest confidence.

```
#rules_fake[rules_fake["antecedents"].str.contains("clinton",
regex=False) ].sort_values("confidence", ascending=False).head(30)

dont_f_rules
=rules_fake[rules_fake["antecedents"].str.contains("dont",
regex=False) ].sort_values("confidence", ascending=False)

f2_rules = dont_f_rules[['antecedents','consequents','support',
'confidence', 'lift']]
f2_rules.head(50)
```

```
                      antecedents consequents   support  confidence
lift
97724  (like, people, make, dont)       (say)  0.101798    0.908309
1.451990
12357             (hillary, dont)   (clinton)  0.102762    0.901408
2.407364
51517          (know, come, dont)       (say)  0.101477    0.900285
1.439162
61849         (think, make, dont)       (say)  0.105652    0.894022
1.429150
97754  (people, make, time, dont)       (say)  0.100835    0.889518
1.421951
61611          (make, know, dont)       (say)  0.113038    0.888889
1.420945
61835         (state, make, dont)       (say)  0.106615    0.887701
1.419045
51587          (make, come, dont)       (say)  0.109827    0.883721
1.412683
61640        (people, know, dont)       (say)  0.111432    0.882952
1.411454
61779        (people, make, dont)       (say)  0.127489    0.882222
1.410287
61751         (think, like, dont)       (say)  0.100835    0.882022
1.409968
61892          (make, dont, year)       (say)  0.105010    0.881402
1.408976
19087                (fact, dont)       (say)  0.100835    0.879552
1.406019
```

```
61919       (think, people, dont)       (say)  0.103725    0.875339
1.399284
61667          (know, time, dont)       (say)  0.106294    0.873351
1.396106
97755   (time, people, say, dont)      (make)  0.100835    0.872222
1.723414
97726     (like, make, say, dont)    (people)  0.101798    0.870879
1.743998
12356              (clinton, dont)   (hillary)  0.102762    0.869565
2.387854
61878            (use, say, dont)      (make)  0.100514    0.869444
1.717925
61863          (make, time, dont)       (say)  0.116891    0.868735
1.388727
17639                (day, dont)       (say)  0.113359    0.867322
1.386468
61877           (use, make, dont)       (say)  0.100514    0.867036
1.386011
51615        (people, come, dont)       (say)  0.109184    0.865140
1.382980
61695          (like, make, dont)       (say)  0.116891    0.864608
1.382130
97725   (like, people, say, dont)      (make)  0.101798    0.863760
1.706694
61795         (state, make, dont)    (people)  0.103725    0.863636
1.729494
19448               (dont, trump)       (say)  0.105331    0.863158
1.379812
61569          (like, know, dont)       (say)  0.109184    0.862944
1.379470
97756     (time, make, say, dont)    (people)  0.100835    0.862637
1.727494
51545          (like, come, dont)       (say)  0.104689    0.862434
1.378655
61948        (people, dont, year)       (say)  0.102119    0.861789
1.377623
19410               (right, dont)       (say)  0.115928    0.861575
1.377282
7841             (american, dont)    (people)  0.111753    0.861386
1.724988
61962          (time, dont, year)       (say)  0.100514    0.857534
1.370822
61905       (people, state, dont)       (say)  0.108542    0.855696
1.367884
19429               (thing, dont)       (say)  0.104689    0.855643
1.367799
19082            (dont, election)       (say)  0.100835    0.855586
1.367708
51644          (time, come, dont)       (say)  0.107258    0.854220
```

```
1.365524
19327                (need, dont)       (say)  0.104689    0.853403
1.364218
19435               (think, dont)       (say)  0.126204    0.852495
1.362766
19465                (dont, way)       (say)  0.113038    0.852300
1.362455
19111                (know, dont)       (say)  0.138728    0.852071
1.362089
61822        (people, dont, year)      (make)  0.100835    0.850949
1.681379
19471                (work, dont)       (say)  0.107900    0.848485
1.356356
12361              (clinton, dont)       (say)  0.100193    0.847826
1.355303
61823          (make, dont, year)    (people)  0.100835    0.846361
1.694900
19478                (dont, year)       (say)  0.124277    0.844978
1.350751
61893           (say, dont, year)      (make)  0.105010    0.844961
1.669549
7847              (american, dont)       (say)  0.109505    0.844059
1.349282
19267                (make, dont)       (say)  0.156390    0.844021
1.349220
```

Sort all rules in the `rules_fake` DataFrame by confidence in descending order and assign the sorted DataFrame to `top_rules`.

```
# Sort the DataFrame by 'confidence' in descending order and select
the top 10 rows
top_rules = rules_fake.sort_values("confidence", ascending=False)


filtered_top_fake_rules =
top_rules[['antecedents','consequents','support', 'confidence',
'lift']]
filtered_top_fake_rules
```

```
                           antecedents
consequents  \
61093    (donald, president, election)
(trump)
45669     (donald, election, campaign)
(trump)
97665   (state, donald, say, election)
(trump)
61473      (donald, say, presidential)
(trump)
```

```
46382          (donald, clinton, come)
(trump)
...                                ...
...
11942                           (say)                       (want,
change)
108625                          (say)  (know, people, state, come,
time)
23497                           (say)                      (look,
hillary)
106748                          (say)         (right, state, year,
make)
60937                           (say)                  (day, think,
time)

         support  confidence      lift
61093   0.113359    1.000000  2.899441
45669   0.105010    1.000000  2.899441
97665   0.100514    1.000000  2.899441
61473   0.105652    1.000000  2.899441
46382   0.100193    1.000000  2.899441
...          ...         ...       ...
11942   0.100193    0.160164  1.337135
108625  0.100193    0.160164  1.404934
23497   0.100193    0.160164  1.302223
106748  0.100193    0.160164  1.351630
60937   0.100193    0.160164  1.385421

[109374 rows x 5 columns]
```

Sort all rules in the `rules_real` DataFrame by confidence in descending order and assign the sorted DataFrame to `top_rules`.

```
# Sort the DataFrame by 'confidence' in descending order and select
the top 10 rows
top_rules = rules_real.sort_values("confidence", ascending=False)


filtered_top_real_rules =
top_rules[['antecedents','consequents','support', 'confidence',
'lift']]
filtered_top_real_rules

                           antecedents             consequents
support  \
1013                            (york)                   (new)
0.242549
13827                     (york, say)                   (new)
0.225111
```

```
24906     (democratic, state, hillary)                (clinton)
0.209575
19214  (democratic, campaign, hillary)                (clinton)
0.208624
24867      (democratic, make, hillary)                (clinton)
0.207990
...                                   ...                     ...       .
..
39317                              (make)      (new, state, look)
0.200063
18462                              (make)        (say, big, come)
0.200063
27659                              (make)       (new, come, week)
0.200063
20915                              (make)  (new, trump, campaign)
0.200063
41957                              (make)     (people, say, talk)
0.200063

       confidence      lift
1013     0.997392  1.575251
13827    0.997191  1.574933
24906    0.996983  2.501580
19214    0.996970  2.501545
24867    0.996960  2.501522
...           ...       ...
39317    0.285908  1.233591
18462    0.285908  1.250701
27659    0.285908  1.225211
20915    0.285908  1.228549
41957    0.285908  1.240379

[75552 rows x 5 columns]
```

association rules for fake news when support is set to .2

```
# Sort the DataFrame by 'confidence' in descending order and select
the top 10 rows
top_rules = rules_fake.sort_values("confidence", ascending=False)


filtered_top_real_rules =
top_rules[['antecedents','consequents','support', 'confidence',
'lift']]
filtered_top_real_rules
```

```
        antecedents  consequents   support  confidence      lift
0            donald        trump  0.245665    0.985825  2.858341
3    state, hillary      clinton  0.214515    0.946176  2.526922
21           united        state  0.207771    0.926934  1.958258
```

```
6      say, hillary      clinton  0.239563    0.924411  2.468797
10   trump, hillary      clinton  0.217084    0.914750  2.442994
..              ...          ...       ...         ...       ...
476             say        point  0.202954    0.324435  1.258146
166             say         told  0.201670    0.322382  1.406019
555             say       follow  0.201349    0.321869  1.207589
396             say   know, like  0.201028    0.321355  1.287902
408             say  state, come  0.200706    0.320842  1.284192

[568 rows x 5 columns]
```

```
#rules_fake[rules_fake["antecedents"].str.contains("clinton",
regex=False) ].sort_values("confidence", ascending=False).head(30)

dont_f_rules
=rules_fake[rules_fake["antecedents"].str.contains("dont",
regex=False) ].sort_values("confidence", ascending=False)

f2_rules = dont_f_rules[['antecedents','consequents','support',
'confidence', 'lift']]
f2_rules.head(50)
```

```
    antecedents consequents   support  confidence      lift
487        dont         say  0.204239    0.782288  1.250536
```

Insights from `rules_real` DataFrame:

1. **Top Associations Involving "dont" and "clinton":**
   - Terms like "like", "state", "time", "say", "make" frequently appear with "dont". Most rules have 'make' as the consequent.
   - High confidence values near 1.0 indicate strong predictive power of these associations.
   - Lift values generally above 1 suggest these associations are more likely than random chance.
   - Regarding "clinton", "hillary" is the most frequent term with higher support and confidence. "Trump" also appears occasionally in comparison to Hillary.
2. **Observations**:
   - Rules primarily involve combinations of verbs ("make", "say") and nouns ("state", "time"), reflecting everyday actions and states.
   - The support threshold of 0.2 ensures robust rules with sufficient occurrence in the dataset.

Insights from `rules_fake` DataFrame:

1. **Top Associations Involving "dont" and "clinton":**
   - Similar terms appear: "like", "make", "know", "think", "people", "time". Most rules have 'say' as the consequent, suggesting a sensationalized news context.

- High confidence and lift values indicate strong associations, with specific mentions of "hillary", "trump", and "election". Around 30 or more rules show Clinton associated with Trump when sorted by confidence.

2. **Observations**:
   - Associations focus on verbs and nouns similar to `rules_real`, but with a notable emphasis on political figures and events.
   - The lower support threshold of 0.1 generates numerous associations, highlighting specific contexts around political discourse.

General Observations:

- **Support Threshold**: Adjusting the support threshold (0.2 for real, 0.1 for fake) influences the number and specificity of generated rules, ensuring relevance and significance in their respective datasets.

Network Graph Analysis for Association Rules: Top 500

Real News: The network graph for real news reveals multiple clusters, primarily focusing on Donald Trump and Hillary Clinton. Additionally, there are clusters related to Republicans.

Fake News: In contrast, the network graph for fake news shows fewer clusters. The main clusters focus on Hillary Clinton and Donald Trump, with a small cluster related to the state. The primary focus is on Clinton, with nodes like FBI, emails, President Trump, "say," and "report" suggesting a negative focus on Clinton. For Trump, the clusters include nodes such as "Donald say," "Donald right," "election," "President Donald," and "state," indicating a more positive focus on Trump.

```
import pandas as pd



# Step 1: Filter rules where 'donald' is in the antecedents
rules_with_donald = rules_fake[rules_fake['antecedents'].apply(lambda
x: 'donald' in x or 'trump' in x)]

# Step 2: Exclude rules where 'clinton' or 'hillary' is in the
antecedents
rules_with_donald =
rules_with_donald[~rules_with_donald['antecedents'].apply(lambda x:
'clinton' in x or 'hillary' in x)]

# Step 3: Exclude rules where 'trump' is in the consequents
rules_with_donald =
rules_with_donald[~rules_with_donald['consequents'].apply(lambda x:
'donald' in x or 'trump' in x)]


# Sort the resulting rules by confidence in descending order
rules_with_donald = rules_with_donald.sort_values('confidence',
ascending=False)
```

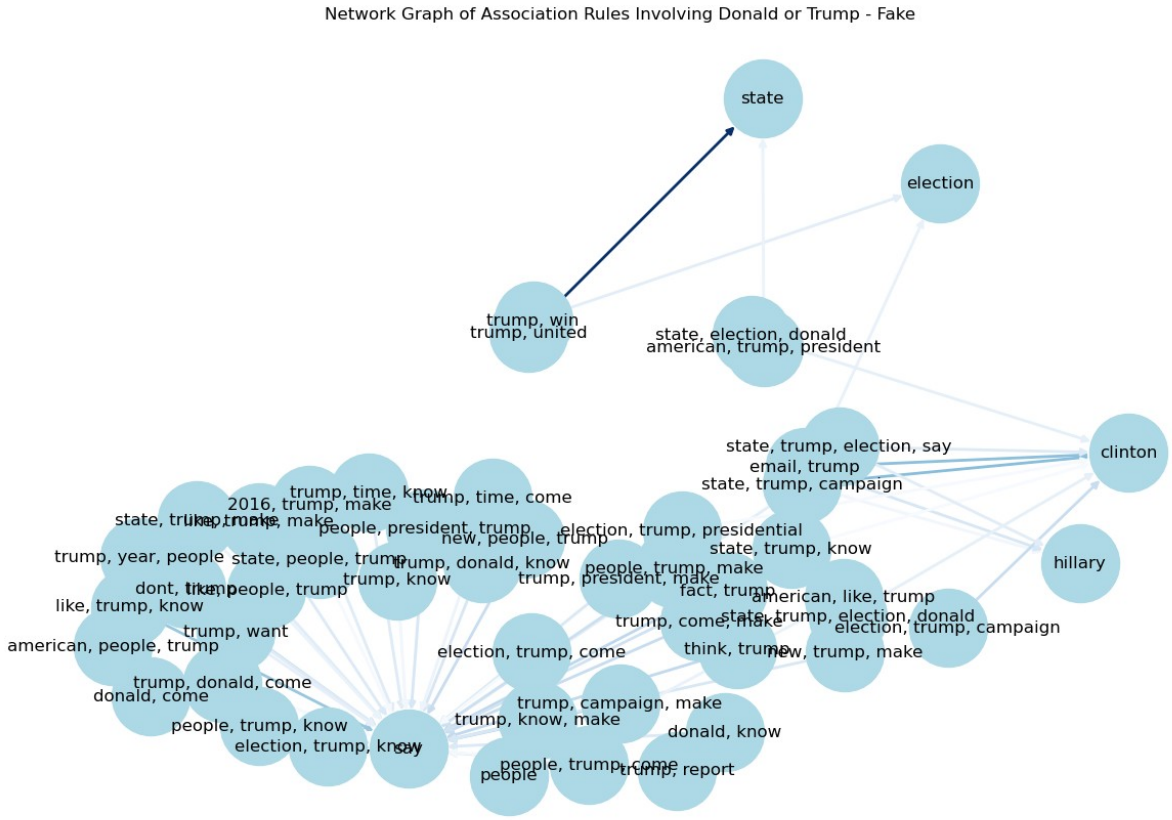
```
# Display the filtered rules
rules_with_donald
```

```
        Unnamed: 0             antecedents  \
23437        36581           trump, united
7057         45628  state, trump, campaign
7114         12545           email, trump
95307        67813       like, trump, know
96210        70011     people, trump, know
...            ...                     ...
102618       36621                   trump
11754        50143                   trump
86724        36603                   trump
4969         90011                   trump
7678         66313                   trump

                                    consequents  antecedent
support  \
23437                                     state            0.106615

7057                                    clinton            0.122030

7114                                    clinton            0.123956

95307                                       say            0.114322

96210                                       say            0.125883

...                                         ...                 ...

102618                            state, world            0.344894

11754                    clinton, medium, say            0.344894

86724                             state, want            0.344894

4969    clinton, presidential, election, hillary            0.344894

7678                 say, hillary, presidential            0.344894


        consequent support   support  confidence      lift
leverage  \
23437              0.473346  0.102119    0.957831  2.023532  0.051654

7057               0.374438  0.108542    0.889474  2.375490  0.062850

7114               0.374438  0.110148    0.888601  2.373159  0.063734

95307              0.625562  0.101156    0.884831  1.414459  0.029640
```

```
96210              0.625562  0.110469    0.877551  1.402820  0.031721

...                     ...       ...         ...       ...       ...

102618             0.217405  0.100193    0.290503  1.336227  0.025211

11754              0.131021  0.100193    0.290503  2.217220  0.055004

86724              0.192999  0.100193    0.290503  1.505201  0.033628

4969               0.116570  0.100193    0.290503  2.492082  0.059988

7678               0.123956  0.100193    0.290503  2.343590  0.057441


        conviction  zhangs_metric  antecedents_length
consequents_length
23437    12.489219       0.566178                   2
1
7057      5.659846       0.659514                   3
1
7114      5.615510       0.660493                   2
1
95307     3.251218       0.330838                   3
1
96210     3.057911       0.328503                   3
1
...            ...            ...                 ...
...
102618    1.103027       0.384097                   1
2
11754     1.224781       0.838009                   1
3
86724     1.137426       0.512340                   1
2
4969      1.245149       0.913942                   1
4
7678      1.234739       0.875132                   1
3

[2391 rows x 13 columns]
```

```python
# Visualizing the filtered rules using a network graph
# Extracting top 50 rules for visualization
top_rules_with_donald = rules_with_donald.head(50)

# Create a directed graph
G = nx.DiGraph()

# Add edges with weights
for _, row in top_rules_with_donald.iterrows():
```

```
    G.add_edge(''.join(row['antecedents']),
''.join(row['consequents']), weight=row['confidence'])

# Draw the graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, k=0.5)
edges = G.edges(data=True)
weights = [edge[2]['weight'] for edge in edges]
nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=3000, edge_color=weights, width=2.0, edge_cmap=plt.cm.Blues)
plt.title('Network Graph of Association Rules Involving Donald or
Trump - Fake')
plt.show()
```



Network Graph of Association Rules Involving Donald or Trump - Fake

1. **Nodes**: The circles in the graph represent different items or terms that are part of the association rules. Each node is labeled with the specific items it represents, like "state," "trump," "clinton," etc.

2. **Edges**: The arrows connecting the nodes represent the association rules between items. An edge from node A to node B indicates that if A occurs, B is likely to occur as well. The direction of the arrow points from the antecedent (the if part) to the consequent (the then part).

3. **Edge Weights**: The thickness and color intensity of the edges represent the strength or confidence of the association. In this graph, darker and thicker edges indicate stronger associations.

4. **Clusters**: Nodes that are closely related and frequently co-occur are clustered together. For example, terms related to "Trump" are closely clustered, indicating strong associations among those terms.

5. **Isolated Nodes**: Some nodes like "state," "election," and "clinton" have fewer connections, indicating that they are not as strongly associated with other terms in the dataset.

6. **Graph Title**: The title of the graph, "Network Graph of Association Rules Involving Donald or Trump - Fake," suggests that the associations are related to fake news or misinformation involving Donald Trump.

Detailed Explanation of Key Nodes and Edges:

- **"clinton" Node**: This node is connected to a few other nodes, indicating specific associations between Clinton-related terms and Trump-related terms. For example, "state, trump, election, say" leads to "clinton," suggesting that when these terms appear together, Clinton is likely to be mentioned.

- **Dense Cluster Around "Trump"**: Many nodes are clustered around terms involving Trump, such as "trump, know," "trump, make," "trump, people," etc. This dense clustering indicates a high frequency of associations involving these terms.

- **Specific Edges**: For example, there is an edge from "trump, win" to "state," indicating that discussions about Trump winning are often associated with mentions of states.

```
import pandas as pd



# Step 1: Filter rules where 'donald' is in the antecedents
rules_with_donald = rules_real[rules_real['antecedents'].apply(lambda
x: 'donald' in x or 'trump' in x)]

# Step 2: Exclude rules where 'clinton' or 'hillary' is in the
antecedents
rules_with_donald =
rules_with_donald[~rules_with_donald['antecedents'].apply(lambda x:
'clinton' in x or 'hillary' in x)]

# Step 3: Exclude rules where 'trump' is in the consequents
rules_with_donald =
rules_with_donald[~rules_with_donald['consequents'].apply(lambda x:
'donald' in x or 'trump' in x)]
```

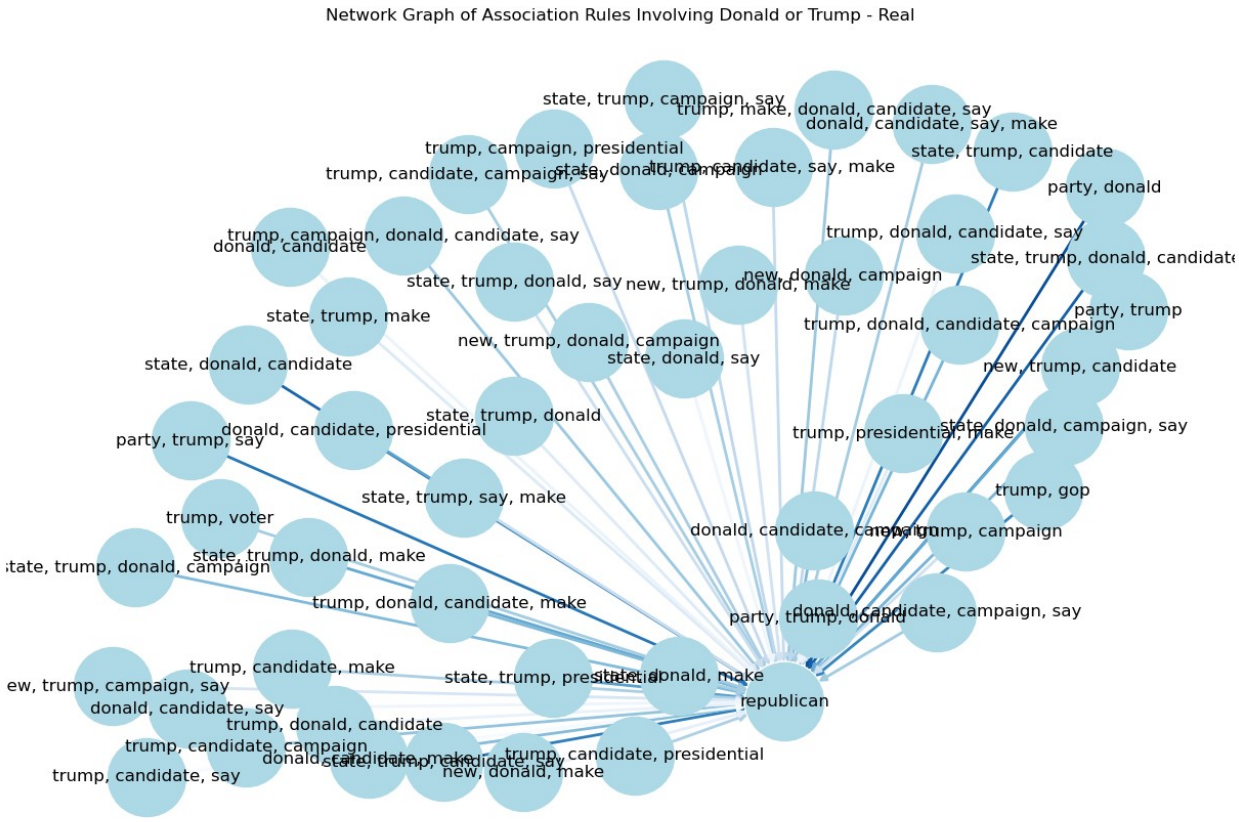
```
# Sort the resulting rules by confidence in descending order
rules_with_donald = rules_with_donald.sort_values('confidence',
ascending=False)

# Display the filtered rules
rules_with_donald
```

```
      Unnamed: 0                       antecedents  \
7534       32627             party, trump, donald
7689        7988                    party, donald
7827       55851  state, trump, donald, candidate
7856       23227         state, donald, candidate
7953       47901                party, trump, say
...          ...                               ...
3565       20416                             trump
5487       50109                             trump
7046       20915                             trump
3477       19211                             trump
4196        4511                             trump

                          consequents  antecedent support  consequent
support  \
7534                       republican            0.220989
0.564046
7689                       republican            0.223209
0.564046
7827                       republican            0.220038
0.564046
7856                       republican            0.222575
0.564046
7953                       republican            0.220672
0.564046
...                               ...                 ...
...
3565           hillary, campaign, say            0.396005
0.270450
5487  state, republican, presidential            0.396005
0.290425
7046              new, campaign, make            0.396005
0.301205
3477       state, candidate, campaign            0.396005
0.268548
4196                  party, candidate            0.396005
0.277743

       support  confidence      lift  leverage  conviction
zhangs_metric  \
7534  0.206722    0.935438  1.658443  0.082074    6.752448
0.509653
```

```
7689  0.207990    0.931818  1.652026  0.082090    6.393997
0.508094
7827  0.204502    0.929395  1.647730  0.080391    6.174537
0.504005
7856  0.206722    0.928775  1.646631  0.081179    6.120799
0.505128
7953  0.204502    0.926724  1.642995  0.080033    5.949495
0.502170
...        ...         ...       ...       ...         ...
...
3565  0.200380    0.506005  1.870972  0.093281    1.476836
0.770732
5487  0.200380    0.506005  1.742292  0.085371    1.436401
0.705376
7046  0.200063    0.505204  1.677278  0.080785    1.412290
0.668542
3477  0.200063    0.505204  1.881244  0.093717    1.478291
0.775564
4196  0.200063    0.505204  1.818966  0.090076    1.459708
0.745432

      antecedents_length  consequents_length
7534                   3                   1
7689                   2                   1
7827                   4                   1
7856                   3                   1
7953                   3                   1
...                  ...                 ...
3565                   1                   3
5487                   1                   3
7046                   1                   3
3477                   1                   3
4196                   1                   2

[1469 rows x 13 columns]
```

```
# Visualizing the filtered rules using a network graph
# Extracting top 20 rules for visualization
top_rules_with_donald = rules_with_donald.head(50)

# Create a directed graph
G = nx.DiGraph()

# Add edges with weights
for _, row in top_rules_with_donald.iterrows():
    G.add_edge(''.join(row['antecedents']),
''.join(row['consequents']), weight=row['confidence'])

# Draw the graph
plt.figure(figsize=(12, 8))
```

```
pos = nx.spring_layout(G, k=0.5)
edges = G.edges(data=True)
weights = [edge[2]['weight'] for edge in edges]
nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=3000, edge_color=weights, width=2.0, edge_cmap=plt.cm.Blues)
plt.title('Network Graph of Association Rules Involving Donald or
Trump - Real')
plt.show()
```



Network Graph of Association Rules Involving Donald or Trump - Real

Key Points to Note:

1. **Nodes**:
   - Each node represents an item (e.g., "donald," "trump," "candidate," "campaign," etc.).
   - Larger nodes often represent more frequently occurring items in the association rules.
2. **Edges**:
   - Each edge represents an association rule where the items in the antecedents lead to the items in the consequents.
   - The direction of the arrow indicates the direction of the rule (from antecedent to consequent).

   - The thickness and color intensity of the edges indicate the strength (confidence) of the association rule. Thicker and darker edges represent stronger associations.
3. **Clusters**:
   - Groups of closely connected nodes indicate clusters of items that frequently appear together in the association rules.

Specific Observations:

1. **Central Nodes**:
   - "republican" appears to be a central node, indicating frequent co-occurrence with other items like "trump," "donald," and "candidate."
   - "donald" and "trump" are also central nodes, showing strong associations with various other items such as "candidate," "campaign," "state," etc.
2. **Strong Associations**:
   - Rules like "donald, candidate -> republican" and "trump, candidate -> republican" show strong associations with high confidence, as indicated by the thick edges.
   - Other strong associations involve "trump, campaign, donald, candidate -> republican" and similar multi-item antecedents leading to "republican."
3. **Clusters**:
   - There is a noticeable cluster around "republican," indicating frequent co-occurrence of this item with "donald," "trump," and other related terms.
   - Another cluster can be observed around "candidate" connecting to various items like "donald," "trump," "campaign," etc.

Insights:

- **Political Context**: The frequent appearance of "republican" with "donald" and "trump" suggests a strong political context in the dataset, indicating discussions or narratives around the Republican party and these individuals.
- **Election and Campaign**: Associations involving "candidate," "campaign," and "state" indicate a focus on election and campaign-related topics in the dataset.
- **Investigate Strong Rules**: Focus on the strongest rules (thickest edges) for deeper analysis. Understanding the context of rules like "donald, candidate -> republican" can provide insights into the dataset's narrative.

```
import pandas as pd


rules_with_clinton = rules_fake[rules_fake['antecedents'].apply(lambda
x: 'clinton' in x or 'hillary' in x)]

rules_with_clinton =
rules_with_clinton[~rules_with_clinton['antecedents'].apply(lambda x:
'donald' in x or 'trump' in x)]


rules_with_clinton =
rules_with_clinton[~rules_with_clinton['consequents'].apply(lambda x:
'clinton' in x or 'hillary' in x)]
```

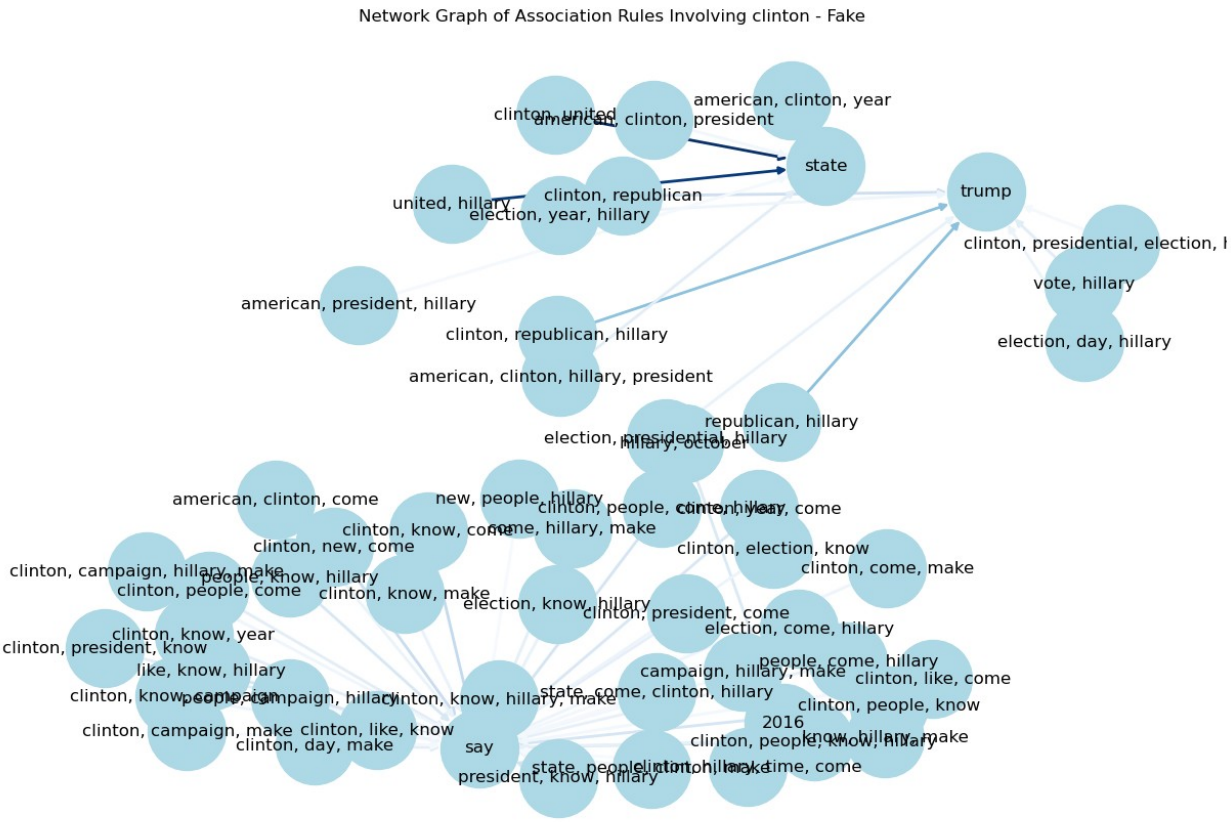
```
rules_with_clinton = rules_with_clinton.sort_values('confidence',
ascending=False)

# Display the filtered rules
rules_with_clinton
```

```
        Unnamed: 0                    antecedents           consequents
\
22145        13639                clinton, united                 state

22965        23938                united, hillary                 state

3353         23836             republican, hillary                trump

3356         48956  clinton, republican, hillary                trump

95256        46606             clinton, know, come                  say

...            ...                             ...                  ...

20146        46392                         clinton  trump, donald, come

46350        13584                         clinton            vote, say

10855        49580                         clinton   state, trump, know

106440        1115                         clinton                 week

100127       13572                         clinton             try, say


        antecedent support  consequent support   support
confidence  \
22145              0.111753            0.473346  0.107900    0.965517

22965              0.105973            0.473346  0.101798    0.960606

3353               0.128452            0.344894  0.115928    0.902500

3356               0.121708            0.344894  0.109827    0.902375

95256              0.122993            0.625562  0.108863    0.885117

...                     ...                 ...       ...         ...

20146              0.374438            0.129416  0.100193    0.267581

46350              0.374438            0.150610  0.100193    0.267581

10855              0.374438            0.119461  0.100193    0.267581
```

```
106440            0.374438            0.207771  0.100193    0.267581

100127            0.374438            0.196532  0.100193    0.267581


            lift  leverage  conviction  zhangs_metric
antecedents_length  \
22145   2.039770  0.055002   15.272961       0.573882
2
22965   2.029394  0.051636   13.368905       0.567368
2
3353    2.616746  0.071626    6.719036       0.708907
2
3356    2.616382  0.067850    6.710410       0.703403
3
95256   1.414916  0.031923    3.259313       0.334369
3
...          ...       ...         ...            ...
...
20146   2.067615  0.051735    1.188643       0.825419
1
46350   1.776650  0.043799    1.159706       0.698800
1
10855   2.239916  0.055462    1.202235       0.884892
1
106440  1.287865  0.022395    1.081661       0.357313
1
100127  1.361518  0.026604    1.097007       0.424459
1

        consequents_length
22145                    1
22965                    1
3353                     1
3356                     1
95256                    1
...                    ...
20146                    3
46350                    2
10855                    3
106440                   1
100127                   2

[4951 rows x 13 columns]
```

```
# Visualizing the filtered rules using a network graph
# Extracting top 20 rules for visualization
top_rules_with_clinton = rules_with_clinton.head(50)

# Create a directed graph
```

```
G = nx.DiGraph()

# Add edges with weights
for _, row in top_rules_with_clinton.iterrows():
    G.add_edge(''.join(row['antecedents']),
''.join(row['consequents']), weight=row['confidence'])

# Draw the graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, k=0.5)
edges = G.edges(data=True)
weights = [edge[2]['weight'] for edge in edges]
nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=3000, edge_color=weights, width=2.0, edge_cmap=plt.cm.Blues)
plt.title('Network Graph of Association Rules Involving clinton -
Fake')
plt.show()
```



Network Graph of Association Rules Involving clinton - Fake

1. **Nodes**: Each circle represents a set of terms or items. Nodes are labeled with the specific items they represent, such as "state," "trump," "clinton," etc.

2. **Edges**: The arrows connecting the nodes represent the association rules between items. An edge from node A to node B indicates that if A occurs, B is likely to occur as well. The direction of the arrow points from the antecedent (the if part) to the consequent (the then part).

3. **Edge Weights**: The thickness and color intensity of the edges represent the strength or confidence of the association. In this graph, darker and thicker edges indicate stronger associations.

4. **Clusters**: Nodes that are closely related and frequently co-occur are clustered together. For example, terms related to "Clinton" are closely clustered, indicating strong associations among those terms.

5. **Isolated Nodes**: Some nodes like "state," "election," and "trump" have fewer connections, indicating that they are not as strongly associated with other terms in the dataset.

Detailed Explanation of Key Nodes and Edges:

- **"clinton" Node**: This node has several connections to other nodes, indicating associations between Clinton-related terms and other political terms. For example, "american, clinton, year" leads to "state," suggesting that discussions about the American context, Clinton, and a specific year are likely to mention the state.

- **Dense Cluster Around "Clinton"**: Many nodes are clustered around terms involving Clinton, such as "clinton, know," "clinton, make," "clinton, people," etc. This dense clustering indicates a high frequency of associations involving these terms.

- **Specific Edges**: For example, there is an edge from "united, hillary" to "state," indicating that discussions about Hillary Clinton and "united" are often associated with mentions of states.

- **Association with "Trump"**: The term "trump" appears as a key node connected to several terms like "clinton, presidential, election," "vote, hillary," and "election, day, hillary." This indicates associations between discussions involving both Trump and Clinton, particularly in the context of elections.

- **Isolated Clusters**: Terms related to election contexts, like "election, day, hillary," are isolated but strongly associated within their cluster, indicating specific contexts in which these terms are discussed together.

This network graph provides a visual representation of the relationships and associations between various terms related to Clinton. It helps to understand the underlying patterns and how certain terms are interconnected in discussions involving Hillary Clinton.

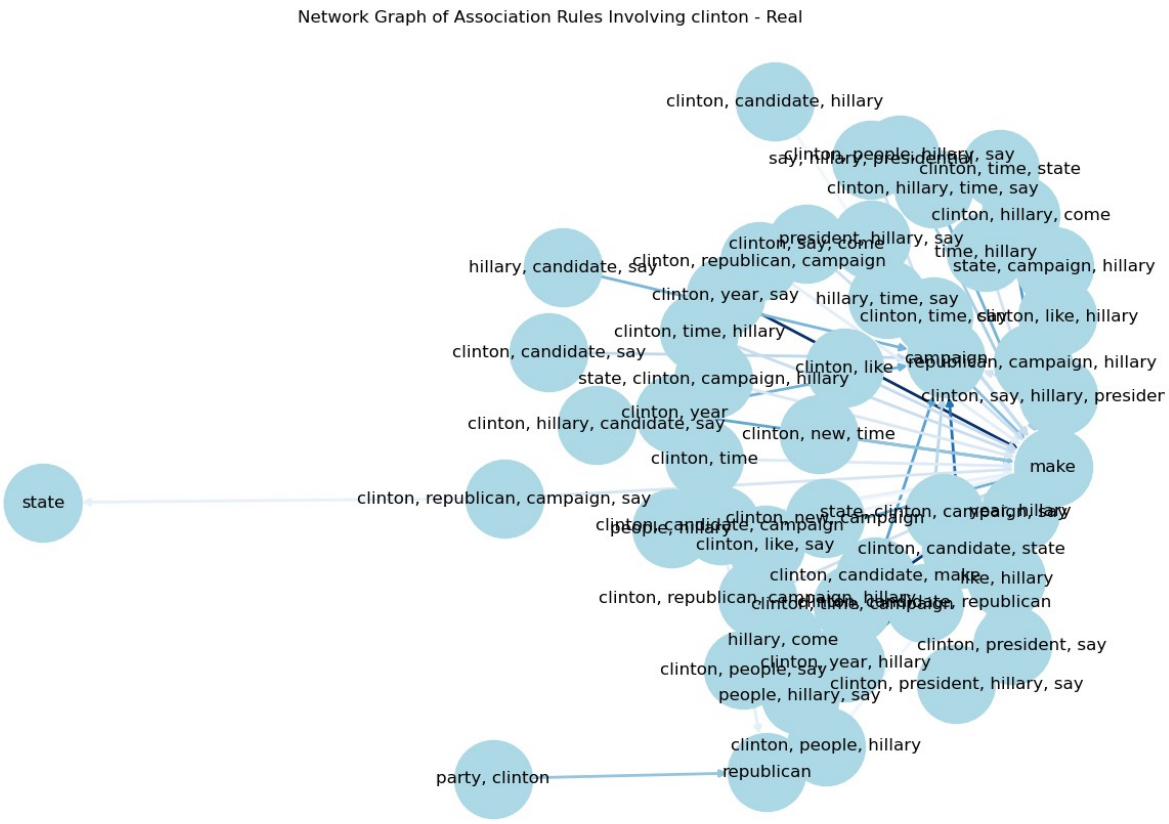
```
import pandas as pd
```

```
rules_with_clinton = rules_real[rules_real['antecedents'].apply(lambda
x: 'clinton' in x or 'hillary' in x)]

# Step 2: Exclude rules where 'clinton' or 'hillary' is in the
antecedents
rules_with_clinton =
rules_with_clinton[~rules_with_clinton['antecedents'].apply(lambda x:
'donald' in x or 'trump' in x)]

# Step 3: Exclude rules where 'trump' is in the consequents
rules_with_clinton =
rules_with_clinton[~rules_with_clinton['consequents'].apply(lambda x:
'clinton' in x or 'hillary' in x)]


# Sort the resulting rules by confidence in descending order
rules_with_clinton = rules_with_clinton.sort_values('confidence',
ascending=False)

# Display the filtered rules
rules_with_clinton
```

```
       Unnamed: 0                 antecedents
consequents  \
54488       25706         clinton, year, say
make
55014       19457    clinton, time, campaign
make
4317        18592  clinton, candidate, state
campaign
58787       25200     clinton, year, hillary
make
59119        9020               year, hillary
make
...           ...                        ...
...
6106        19506                    clinton         state, new,
campaign
3926        22983                    clinton        trump, candidate,
say
4228         4975                    clinton                state,
donald
7414        22887                    clinton  candidate, republican,
make
56622        5109                    clinton                  like,
time

       antecedent support  consequent support   support  confidence
lift  \
54488            0.239062            0.699746  0.211477    0.884615
```

```
1.264194
55014             0.228916             0.699746  0.202283    0.883657
1.262824
4317              0.229550             0.484781  0.201332    0.877072
1.809212
58787             0.231452             0.699746  0.202917    0.876712
1.252900
59119             0.233037             0.699746  0.204185    0.876190
1.252154
...                    ...                  ...       ...         ...
...
6106              0.398542             0.292961  0.200380    0.502784
1.716214
3926              0.398542             0.273304  0.200380    0.502784
1.839654
4228              0.398542             0.276474  0.200063    0.501989
1.815680
7414              0.398542             0.301839  0.200063    0.501989
1.663102
56622             0.398542             0.398859  0.200063    0.501989
1.258563

       leverage  conviction  zhangs_metric  antecedents_length  \
54488  0.044195    2.602198       0.274638                   3
55014  0.042100    2.580752       0.269911                   3
4317   0.090050    4.191218       0.580535                   3
58787  0.040959    2.435391       0.262641                   3
59119  0.041118    2.425126       0.262564                   2
...         ...         ...            ...                 ...
6106   0.083623    1.421996       0.693850                   1
3926   0.091458    1.461532       0.758855                   1
4228   0.089877    1.452830       0.746921                   1
7414   0.079768    1.401898       0.662912                   1
56622  0.041102    1.207084       0.341575                   1

       consequents_length
54488                   1
55014                   1
4317                    1
58787                   1
59119                   1
...                   ...
6106                    3
3926                    3
4228                    2
7414                    3
56622                   2

[1559 rows x 13 columns]
```

```
# Visualizing the filtered rules using a network graph
# Extracting top 20 rules for visualization
top_rules_with_clinton = rules_with_clinton.head(50)

# Create a directed graph
G = nx.DiGraph()

# Add edges with weights
for _, row in top_rules_with_clinton.iterrows():
    G.add_edge(''.join(row['antecedents']),
''.join(row['consequents']), weight=row['confidence'])

# Draw the graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, k=0.5)
edges = G.edges(data=True)
weights = [edge[2]['weight'] for edge in edges]
nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=3000, edge_color=weights, width=2.0, edge_cmap=plt.cm.Blues)
plt.title('Network Graph of Association Rules Involving clinton -
Real')
plt.show()
```



Interpretation of the Real Network Graph

1. **Nodes and Edges**: Similar to the previous graphs, nodes represent terms and edges represent the associations between them. The direction of the edges indicates the flow from antecedents to consequents in association rules.

2. **Key Nodes**:
   - **"clinton"**: This node has many connections, indicating strong associations with other terms.
   - **"state"**: There is a notable connection between "state" and "clinton, republican, campaign, say," suggesting frequent mention of states in the context of Clinton's campaign and Republicans.
   - **"party, clinton"**: This isolated cluster suggests discussions about Clinton in relation to the party.
3. **Dense Clusters**:
   - The central cluster includes terms like "clinton, candidate, hillary," "hillary, time, say," and "clinton, time, hillary." This shows a strong association among these terms, indicating frequent co-occurrence in the data.
4. **Isolated Nodes and Clusters**:
   - Terms like "party, clinton" and "clinton, republican, campaign" form smaller clusters, indicating less frequent but still significant associations.

Comparison Between Real and Fake Graphs

General Observations:

- **Node Density**: The real graph appears to have fewer, more focused clusters compared to the fake graph, which is more dispersed.
- **Edge Weights**: The real graph edges appear lighter, indicating potentially weaker associations compared to the fake graph.

Specific Comparisons:

1. **Central Clusters**:
   - **Real**: Central terms are focused around specific campaign-related terms like "clinton, candidate, hillary" and "clinton, republican, campaign."
   - **Fake**: The central cluster is larger and includes broader terms like "clinton, know," "clinton, make," "clinton, people."
2. **Association Strength**:
   - **Real**: The associations seem more specific and targeted, with terms like "clinton, republican, campaign" being prominent.
   - **Fake**: Associations are more varied, with terms like "clinton, campaign, make" and "clinton, new, come" being prominent, suggesting a broader but less specific set of associations.
3. **Isolated Clusters**:

- **Real**: There are distinct smaller clusters like "party, clinton" and "clinton, republican, campaign."
- **Fake**: The fake graph has more interconnected clusters without many isolated terms.

4. **Key Terms**:
   - **Real**: Terms related to political processes and campaigns are more central.
   - **Fake**: There are broader terms including personal attributes and actions, indicating a wider range of topics.

Conclusion

The real network graph shows more specific and targeted associations around Clinton's campaign and political activities, whereas the fake graph includes a broader and more varied set of associations. This suggests that real data tends to focus on more concrete and specific events or contexts, while fake data might mix a wider range of terms, leading to less specific but more numerous associations.

```
import pandas as pd



# Function to filter rules based on keywords
def filter_rules_by_keywords(rules, keywords):
    filtered_rules = rules[(rules['antecedents'].apply(lambda x:
any(keyword in x for keyword in keywords))) |
                           (rules['consequents'].apply(lambda x:
any(keyword in x for keyword in keywords)))]
    filtered_rules = filtered_rules.sort_values('confidence',
ascending=False)
    return filtered_rules

# Keywords to filter by
keywords = ['email', 'mail', 'FBI']


# Applying the function to each DataFrame
rules_fake_filtered = filter_rules_by_keywords(rules_fake, keywords)
rules_fake1_filtered = filter_rules_by_keywords(rules_fake1, keywords)
rules_real_filtered = filter_rules_by_keywords(rules_real, keywords)

# Display the filtered DataFrames
print("Filtered Rules with Email, Mail, or FBI - Fake")
print(rules_fake_filtered)

print("\nFiltered Rules with Email, Mail, or FBI - Fake1")
print(rules_fake1_filtered)

print("\nFiltered Rules with Email, Mail, or FBI - Real")
print(rules_real_filtered)
```

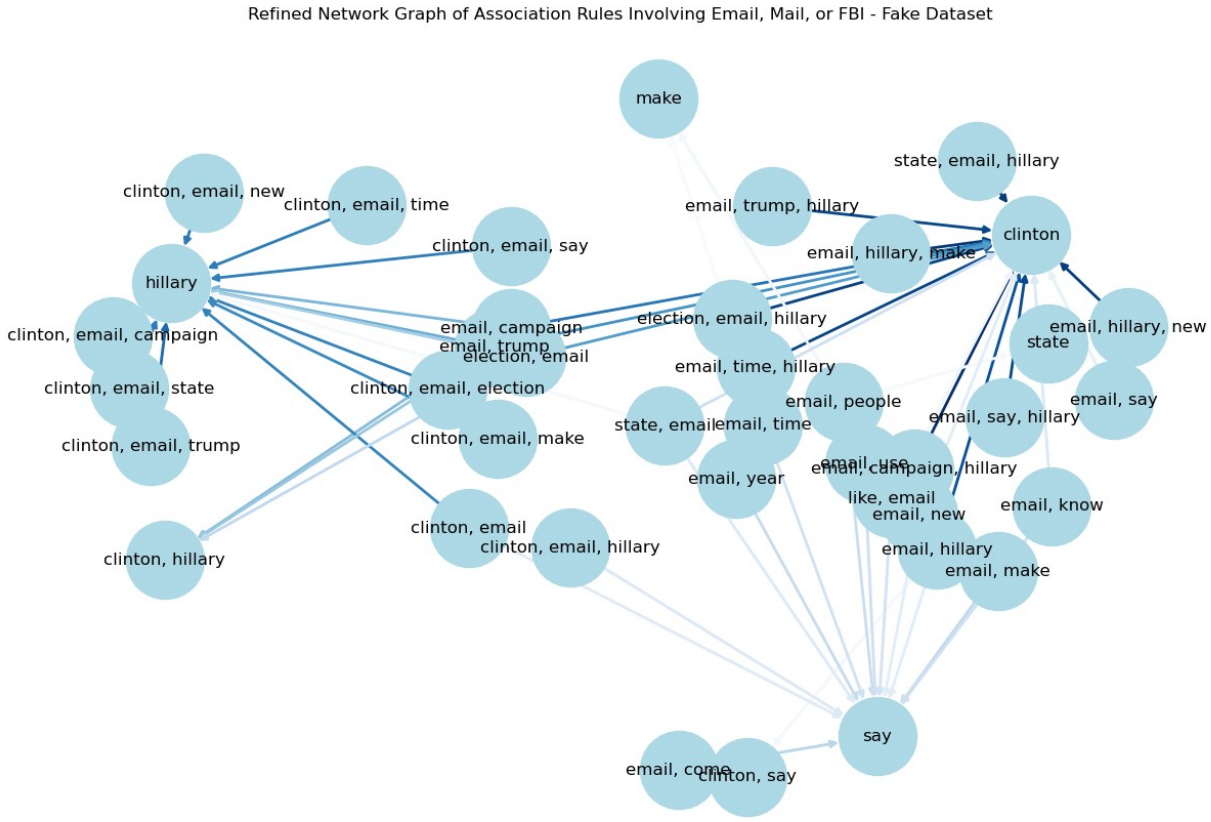
```
Filtered Rules with Email, Mail, or FBI - Fake
        Unnamed: 0               antecedents  consequents  antecedent
support  \
3218         45291  email, campaign, hillary      clinton
0.104689
3342         48035      email, hillary, make      clinton
0.102441
3381         48077     state, email, hillary      clinton
0.118176
3437         48049       email, hillary, new      clinton
0.105010
3513         47336  election, email, hillary      clinton
0.103725
...            ...                       ...          ...
...
108493       20421                     say  like, email
0.625562
108151       20491                     say   email, use
0.625562
108233       20415                     say  email, know
0.625562
106527       14137                     say  email, come
0.625562
107559       20494                     say  email, year
0.625562

        consequent support   support  confidence      lift
leverage  \
3218              0.374438  0.103725    0.990798  2.646092  0.064526

3342              0.374438  0.100514    0.981191  2.620437  0.062156

3381              0.374438  0.115607    0.978261  2.612611  0.071357

3437              0.374438  0.102441    0.975535  2.605331  0.063121

3513              0.374438  0.100835    0.972136  2.596254  0.061996

...                    ...       ...         ...       ...       ...

108493            0.139692  0.108542    0.173511  1.242102  0.021156

108151            0.137765  0.107900    0.172485  1.252021  0.021719

108233            0.136802  0.106936    0.170945  1.249581  0.021359

106527            0.126525  0.101798    0.162731  1.286153  0.022649

107559            0.126846  0.100514    0.160678  1.266709  0.021163
```

```
        conviction  zhangs_metric  antecedents_length
consequents_length
3218     67.977735       0.694824                   3
1
3342     33.259045       0.688962                   3
1
3381     28.775851       0.699960                   3
1
3437     25.569846       0.688467                   3
1
3513     22.450724       0.685983                   3
1
...            ...            ...                 ...
...
108493    1.040920       0.520548                   1
2
108151    1.041957       0.537583                   1
2
108233    1.041183       0.533417                   1
2
106527    1.043243       0.594191                   1
2
107559    1.040308       0.562317                   1
2

[372 rows x 13 columns]

Filtered Rules with Email, Mail, or FBI - Fake1
Empty DataFrame
Columns: [Unnamed: 0, antecedents, consequents, antecedent support,
consequent support, support, confidence, lift, leverage, conviction,
zhangs_metric, antecedents_length, consequents_length]
Index: []

Filtered Rules with Email, Mail, or FBI - Real
Empty DataFrame
Columns: [Unnamed: 0, antecedents, consequents, antecedent support,
consequent support, support, confidence, lift, leverage, conviction,
zhangs_metric, antecedents_length, consequents_length]
Index: []
```

rules_fake Dataset: The rules_fake dataset contains several rules involving "email," "mail," or "FBI" in the antecedents or consequents. The top rules, based on confidence, show strong associations between "email, campaign, hillary" and "clinton," with confidence values close to 1.0. The lift values are greater than 1, indicating a positive correlation between the antecedents and consequents. These rules highlight the significant role of "email" and "hillary" in the context of "clinton." rules_fake1 and rules_real Datasets: Both rules_fake1 and rules_real datasets did not contain any rules involving "email," "mail," or "FBI" in either the antecedents or consequents after applying the filtering criteria. This suggests that, under the specific conditions and support

thresholds of these datasets, the keywords of interest did not form any significant association rules.

```
import networkx as nx
import matplotlib.pyplot as plt

# Extracting top 20 rules for visualization
top_rules_fake = rules_fake_filtered.head(50)

# Create a directed graph
G = nx.DiGraph()

# Add edges with weights
for _, row in top_rules_fake.iterrows():
    G.add_edge(''.join(row['antecedents']),
''.join(row['consequents']), weight=row['confidence'])

# Draw the graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, k=0.5)
edges = G.edges(data=True)
weights = [edge[2]['weight'] for edge in edges]
nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=3000, edge_color=weights, width=2.0, edge_cmap=plt.cm.Blues)
plt.title('Refined Network Graph of Association Rules Involving Email,
Mail, or FBI - Fake Dataset')
plt.show()
```



Key Points to Note:

1. **Nodes**:
   - Each node represents an item (e.g., "clinton," "email," "hillary," "campaign," etc.).
   - Larger nodes often represent more frequently occurring items in the association rules.
2. **Edges**:
   - Each edge represents an association rule where the items in the antecedents lead to the items in the consequents.
   - The direction of the arrow indicates the direction of the rule (from antecedent to consequent).
   - The thickness and color intensity of the edges indicate the strength (confidence) of the association rule. Thicker and darker edges represent stronger associations.
3. **Clusters**:
   - Groups of closely connected nodes indicate clusters of items that frequently appear together in the association rules.
   - For example, nodes connected to "clinton" and "hillary" often form a distinct cluster indicating strong associations between these items and other related items like "email" and "campaign."

Specific Observations:

1. **Central Nodes**:

   - "clinton" and "hillary" are central nodes, indicating they frequently appear in the rules.
   - "email" is another central node, suggesting it plays a significant role in the associations.
2. **Strong Associations**:
   - Rules like "clinton, email, campaign -> clinton" and "email, campaign, hillary -> clinton" show strong associations with high confidence, as indicated by the thick edges.
   - "email, hillary, make -> clinton" and "email, state, hillary -> clinton" are also strong associations.
3. **Clusters**:
   - There is a noticeable cluster around "clinton" and "hillary," indicating frequent co-occurrence of these items with "email" and other related terms.
   - Another cluster can be observed around "email" connecting to various items like "state," "campaign," "election," etc.

Insights and Further Analysis:

- **Email and Campaign Context**: The frequent appearance of "email" with "clinton" and "hillary" suggests that emails related to their campaigns are significant in the dataset.
- **State and Election Context**: Associations involving "state" and "election" with "email" indicate discussions or rules involving these terms are prevalent.
- **Investigate Strong Rules**: Focus on the strongest rules (thickest edges) for deeper analysis. For instance, understanding the context of "email, campaign, hillary -> clinton" can provide insights into the dataset's narrative.

Next Steps:

1. **Deep Dive into Specific Rules**: Investigate the top rules by confidence to understand the underlying data and context better.
2. **Compare with Other Datasets**: Compare these findings with `rules_fake1` and `rules_real` (even though they are empty for these keywords) to see if similar patterns emerge with different support thresholds.
3. **Contextual Analysis**: Analyze the text or data surrounding these rules to derive more specific insights into why these associations exist.

Summary of Association Rules with Topic Modeling

Context and Goal

The analysis aimed to extract and compare association rules from real and fake news datasets to identify patterns and themes involving specific keywords.

Key Findings

1. **Real News Dataset**:
   - **Themes**: Primarily political, focusing on Hillary Clinton and governmental activities.
   - **Key Rule**: (use, say) → (email, hillary, department, state, president)

   - **Insight**: Strong and reliable associations within the political theme, indicated by high confidence and lift values.
2. **Fake News Dataset**:
   - **Themes**: Varied, often sensational or controversial topics covering social justice, law enforcement, and racial issues.
   - **Key Rule**: (think, make, say) → (thing, people, time, world)
   - **Insight**: High confidence and lift values despite thematic diversity, reflecting strong associations.
3. **Network Graph of Donald/Trump - Fake**:
   - **Key Nodes**: Central nodes like "trump" and "donald" with strong associations to terms like "candidate" and "campaign".
   - **Insight**: Frequent discussions around political contexts and campaigns involving Trump.
4. **Network Graph of Donald/Trump - Real**:
   - **Key Nodes**: Central nodes "republican" and "candidate" indicating strong political context.
   - **Key Rule**: (donald, candidate) → (republican)
   - **Insight**: High-confidence associations reflect discussions around the Republican party and electoral campaigns.
5. **Network Graph of Clinton - Fake**:
   - **Key Nodes**: Central nodes "clinton" and "hillary" frequently associated with "email" and "campaign".
   - **Key Rule**: (email, campaign, hillary) → (clinton)
   - **Insight**: Strong associations indicate significant discussions about email-related controversies involving Clinton.

Conclusion

Real news datasets focus on specific political narratives, while fake news datasets cover a broader range of socially charged topics. Both datasets exhibit strong, reliable associations reflecting their thematic focuses.

Summary:

- **Without Topic Modeling:** The analysis focuses on direct associations and frequent co-occurrences of "clinton," "say," and "dont" in both real and fake news datasets. It sheds light on typical narrative structures and thematic clusters, revealing that associations primarily revolve around general terms and specific political figures like "Hillary" and "Trump." Real news tends to concentrate on political actions, while fake news covers a wide array of sensational topics.

- **With Topic Modeling:** Incorporating topic modeling enriches the analysis by uncovering underlying themes or topics where these terms coalesce. This approach provides a deeper understanding of how these terms contribute to broader narratives in both genuine and deceptive news contexts. In the real news dataset, there is a pronounced focus on political discourse, particularly around Hillary

Clinton. Conversely, fake news explores a broader spectrum of socially charged issues.

Summary of EDA

Descriptive Statistics

1. **Title Column:**
   - **Count:** 6335 entries
   - **Unique:** 6256 unique titles
   - **Most Frequent Title:** "OnPolitics | 's politics blog" (5 occurrences)
2. **Text Column:**
   - **Count:** 6335 entries
   - **Unique:** 6060 unique text entries
   - **Most Frequent Text:** Related to "Killing Obama administration rules, dismantlin..." (frequency not specified)
3. **Label Column:**
   - **Count:** 6335 entries
   - **Unique:** 2 unique labels (binary classification: REAL, FAKE)
   - **Most Frequent Label:** "REAL" (3171 occurrences)

Text Analysis

1. **TF-IDF Vectorization:**
   - Converts text data into numerical vectors representing word importance.
   - TF-IDF vectors created separately for real and fake news articles.
2. **Semantic Differences Analysis:**
   - Mean TF-IDF values calculated for real and fake news.
   - Significant differences in term usage identified.
3. **T-Test on TF-IDF Vectors:**
   - Independent t-test performed on TF-IDF vectors.
   - Statistically significant differences found between real and fake news terms.
4. **Top Features Analysis:**
   - Identified top terms with the most significant differences.
   - Examples of significant features with the lowest p-values provided.
5. **PCA Visualization:**
   - PCA applied to reduce dimensionality of TF-IDF vectors.
   - Real and fake news articles show distinct clustering in the 2D PCA plot.

Sentiment Analysis

1. **Aspect-Based Sentiment Analysis:**
   - Sentiment analysis performed on various aspects (politician, organization, location, etc.).
   - Real news generally more positive; fake news exhibits higher negative sentiment.

2. **Chi-Square Test:**
   - Significant association between news type and sentiment.

Co-occurrence Analysis

1. **Patterns in Real and Fake News:**
   - Real news focuses on credible sources, political figures, and national topics.
   - Fake news emphasizes negative associations, sensational events, and conspiratorial narratives.

Named Entity Recognition (NER)

1. **Entity Counts:**
   - Real news articles have higher named entity counts across most categories.
   - Real news more detailed and specific in references to entities.

POS Tag Analysis

1. **POS Tag Communities:**
   - Real news has more complex and densely connected POS tag communities.
   - Fake news shows simpler and less connected communities, indicating less varied grammatical structures.

Topic Modeling and Visualization

1. **Topic Networks:**
   - LSA applied to identify topics in real and fake news.
   - Visualization shows distinct themes in real and fake news articles.
2. **Comparison of Topics:**
   - Real news covers a wider range of subjects and focuses on political discourse.
   - Fake news explores broader sensational topics and socially charged issues.

Association Rules

1. **Key Findings:**
   - Real news exhibits strong and reliable associations within political themes.
   - Fake news covers a wide array of sensational topics with high confidence and lift values in associations.

Conclusion

- Real and fake news articles exhibit significant semantic differences, which can be leveraged to improve classification accuracy.
- Sentiment analysis, co-occurrence patterns, NER, POS tag analysis, and topic modeling provide comprehensive insights into the linguistic and thematic structures of real and fake news.
- These findings can guide feature engineering, model training, and further analysis to develop robust fake news detection systems.

Achievements

- **Preprocessed and transformed** the dataset into a structured format suitable for **association rule mining**, converting textual data into a **binary transactional format** to identify key word associations in fake and real news.
- **Applied association rule mining** using the **Apriori algorithm**, uncovering significant word combinations and linguistic patterns that differentiate fake news from real news.
- **Performed feature engineering**, including **TF-IDF vectorization, stopword removal, and bigram/trigram extraction**, to enhance the representation of textual data for better rule discovery.
- **Analyzed sentiment polarity** and linguistic tendencies in fake news vs. real news, revealing the prevalence of **sensationalized or emotionally charged language** in misleading articles.
- **Implemented clustering techniques**, such as **Topic Modeling (LDA)**, to group similar news articles and enhance the interpretability of frequent itemsets and co-occurring keywords.
- **Utilized dimensionality reduction (PCA & t-SNE)** for visualizing high-dimensional feature spaces, helping to distinguish linguistic structures between fake and real news.
- **Generated comprehensive visualizations**, including **word clouds, association graphs, heatmaps, and lift-based network plots**, to present key insights from association rule mining and sentiment analysis.
- **Validated findings through statistical tests**, ensuring that discovered associations and linguistic trends hold significance for detecting fake news patterns.

References

The following is the table of References for each section of our project.

Topic	Reference Website	Description
Data Overview	Kaggle	A platform with datasets and kernels for data analysis and machine learning.
Data Cleaning	Towards Data Science	Articles on data cleaning techniques and best practices in Python.
Univariate Analysis	Analytics Vidhya	Tutorials on univariate analysis methods and visualizations.
Bivariate Analysis	DataCamp	Courses on statistical analysis and visualization techniques for exploring

Topic	Reference Website	Description
		relationships between variables.
Hypothesis Testing	StatQuest	Clear explanations of hypothesis testing concepts and methods.
Sentiment Analysis	NLTK Documentation	Resources for performing sentiment analysis using the Natural Language Toolkit (NLTK).
Named Entity Recognition (NER)	spaCy Documentation	Comprehensive guide on using spaCy for NER tasks.
Topic Modeling	Gensim Documentation	Information on topic modeling techniques like LDA and NMF using Gensim library.
Association Rule Mining	Towards Data Science - Association Rules	Articles explaining association rule mining techniques like Apriori algorithm.
Co-occurrence Analysis	Medium - Co-occurrence Analysis	Insights into co-occurrence analysis methods in text data.

Thank You!

We express our sincere gratitude to the data providers on Kaggle. Most importantly, we extend our heartfelt thanks to our professors for inspiring and fostering the curiosity that drove us to explore this project.