

Setup

```
In [1]: from bs4 import BeautifulSoup
import json
import logging
from newsapi import NewsApiClient
import numpy as np
import os
import pandas as pd
import random
import re
import requests
import string
import time
```

Source URLs

```
In [ ]: api_key = os.environ['NewsAPIKey']
newsapi = NewsApiClient(api_key=api_key)

sources = newsapi.get_sources(language='en', country='us')
print(' - '.join([source['id'] for source in sources['sources']]))
```

```
In [ ]: def news_api_urls(q=None,
                        s=None,
                        d_from='2023-05-01',
                        d_to='2023-05-31',
                        api_lst=[]):
    all_articles = newsapi.get_everything(q=q,
                                          sources=s,
                                          from_param=d_from,
                                          to=d_to,
                                          language='en',
                                          sort_by='relevancy',
                                          page=1)

    for article in all_articles['articles']:
        print('Title:', article['title'])

    source_data01 = [(a['source']['name'],
                      a['author'],
                      a['title'],
                      a['url'],
                      a['publishedAt'],
                      a['content'])
                     for a in all_articles['articles']]

    api_lst.extend(source_data01)
    print(len(api_lst))
```

```
In [ ]: # Already executed:
# run 'A': 'the-washington-post', '2023-05-30'
# run 'B': 'the-washington-post', '2023-05-20/29'
# run 'C': 'the-washington-post', '2023-05-21/22/23/24/25/26/27/28'
# run 'D': 'the-washington-post', '2023-05-10/11/12/13/14/15/16/17/18/19'
# run 'E': 'the-washington-post', '2023-05-05/06/07/08/09/31', '2023-06-01/02/03'
# run 'F': 'fox-news', '2023-05-24/25/26/27/28'
# run 'G': 'breitbart-news', '2023-05-24/25/26/27/28'
```

```
# run 'H': 'cnn', '2023-05-24/25/26/27/28'

# Last to execute:
#run = 'H'
#source_lst = ['cnn']
#date_lst = ['2023-05-24', '2023-05-25', '2023-05-26', '2023-05-27', '2023-05-28']

q_word_lst = ['justice OR surveillance', 'healthcare OR "health care"',
              '(political AND (bias OR party)) OR republican OR democrat OR election',
              'security AND (social OR national)']
```

In []:

```
api_record_lst01 = []
for s in source_lst:
    print(f'Source: {s}')
    for d in date_lst:
        print(f'Date: {d}')
        for q in q_word_lst:
            print(f'Query word: {q}')
            time.sleep(5 * random.random())
            news_api_urls(q=q,
                          s=s,
                          d_from=d,
                          d_to=d,
                          api_lst=api_record_lst01)
```

Scrape articles

In []:

```
api_df = pd.DataFrame(api_record_lst01, columns=['source_name', 'author', 'title', 'url',
api_df['article_text'] = ''

total_urls = len(api_df)
for i, row in enumerate(api_df.itertuples(), 1):
    print(f'Retrieving url {i} of {total_urls}...', end='')
    response = requests.get(row.url)

    if response.status_code == 200:
        print('; now Scraping...', end='')

        soup = BeautifulSoup(response.content, 'html.parser')

        # The Washington Post
        try:
            script_tag = soup.find('script', {'type': 'application/ld+json'})
            article_json = json.loads(script_tag.string)
            article_content = article_json['hasPart']['value']
            api_df.at[row.Index, 'article_text'] = article_content

        # Fox News
        try:
            script_tag = soup.find('script', {'type': 'application/ld+json'})
            article_json = json.loads(script_tag.string)
            article_content = article_json['articleBody']
            api_df.at[row.Index, 'article_text'] = article_content

        # Breitbart News
        try:
            title_tag = soup.find('h1')
            if title_tag is not None:
                title = title_tag.text
            content_tags = soup.find_all(['p', 'blockquote'])
            content = " ".join([tag.text for tag in content_tags if tag.text.strip() !=
            api_df.at[row.Index, 'article_text'] = content
```

```

# CNN News
try:
    script_tag = soup.find('script', {'type': 'application/ld+json'})
    article_json = json.loads(script_tag.string)
    article_content = article_json['articleBody']
    api_df.at[row.Index, 'article_text'] = article_content

except KeyError:
    print('; missing key in article JSON!', end='')

    time.sleep(5 * random.random())
    print('; done!')
else:
    print(f' response is {response.status_code}!')

api_df.to_csv(f'509_final_proj-{run}.csv', index=False)

```

Combine output files (*only at end of iterative process*)

In []:

```

# Combine all the output files
master_df = pd.DataFrame()
for letter in string.ascii_lowercase:
    file_name = f'509_final_proj-{letter.upper()}.csv'

    if os.path.isfile(file_name):
        df = pd.read_csv(file_name)

        master_df = pd.concat([master_df, df], ignore_index=True)
    else:
        break
print(master_df.info())

# Get rid of what appear to be very cluttered (misread) article text rows
pattern = re.compile('\n{3,}')
rows_list = []
for index, row in master_df.iterrows():
    try:
        processed_row = row
        if not pattern.search(str(processed_row[6])):
            rows_list.append(processed_row)
    except Exception as e:
        print(f'Error processing row {index}: {e}')
print(f'Rows processed = {len(rows_list)}')
new_df = pd.concat(rows_list, axis=1).transpose()
new_df.columns = master_df.columns

# Save the new file
new_df.to_csv('509_final_proj.csv', index=False)

```

Combine all final files

In [2]:

```

df0 = pd.read_csv('../data/509_final_proj.csv')
df1 = pd.read_csv('../data/data_parsed_amc.csv')
df2 = pd.read_csv('../data/News_API_FOX_CNN_Breitbart_May18_23_May31_3.csv')

```

In [3]:

```

print(df0.info())
print(df1.info())
print(df2.info())

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1594 entries, 0 to 1593
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   source_name     1594 non-null   object
1   author          1564 non-null   object
2   title           1593 non-null   object
3   url             1593 non-null   object
4   publishedAt     1593 non-null   object
5   content         1593 non-null   object
6   article_text    1561 non-null   object
dtypes: object(7)
memory usage: 87.3+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3390 entries, 0 to 3389
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   source_name     3390 non-null   object
1   author          3375 non-null   object
2   title           3390 non-null   object
3   url             3390 non-null   object
4   publish_date    3390 non-null   object
5   article_parsed  3375 non-null   object
dtypes: object(6)
memory usage: 159.0+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1302 entries, 0 to 1301
Data columns (total 9 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   Unnamed: 0.1    1302 non-null   int64
1   Unnamed: 0      1302 non-null   int64
2   Source          1302 non-null   object
3   Author         1302 non-null   object
4   Title           1302 non-null   object
5   URL             1302 non-null   object
6   date            1302 non-null   object
7   content         1302 non-null   object
8   article_parsed  1295 non-null   object
dtypes: int64(2), object(7)
memory usage: 91.7+ KB
None

```

In [4]:

```

df0 = df0.rename(columns={'publishedAt': 'publish_date'})

df1 = df1.rename(columns={'article_parsed': 'article_text'})
df1['content'] = np.nan

df2 = df2.drop(columns=['Unnamed: 0.1', 'Unnamed: 0'])
df2 = df1.rename(columns={'Source': 'source_name',
                          'Author': 'author',
                          'Title': 'title',
                          'URL': 'url',
                          'date': 'publish_date',
                          'article_parsed': 'article_text'})

df = pd.concat([df0, df1, df2], ignore_index=True)

df = df.drop_duplicates(subset='article_text')
df.info()

```

```
df.to_csv('../data/master.csv', index=False)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4509 entries, 0 to 4983
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  -
0   source_name     4509 non-null   object
1   author          4472 non-null   object
2   title           4509 non-null   object
3   url             4509 non-null   object
4   publish_date    4509 non-null   object
5   content         1158 non-null   object
6   article_text    4508 non-null   object
dtypes: object(7)
memory usage: 281.8+ KB
```

In []:

Illuminating Mainstream Media Political Bias through Text Mining

Aaron Carr, Azucena Faus, and Dave Friesen - ADS-509-01-SU23

```
In [1]: __author__ = 'Aaron Carr, Azucena Faus, Dave Friesen'
__email__ = 'acarr@sandiego.edu, afaus@sandiego.edu, dfriesen@sandiego.edu'
__version__ = '1.0'
__date__ = 'June 2023'
```

Setup

```
In [2]: # Import basic and data access libraries
import numpy as np
import pandas as pd
from profiler import profile, profile_cat

# Import pre-processing, model and performance evaluation libraries
from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from nltk.tokenize import sent_tokenize, word_tokenize
from sklearn.decomposition import LatentDirichletAllocation

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import NearestCentroid
from model_process import ModelProcess

from transformers import pipeline as tpipeline

# Import lexicons
import nltk
#nltk.download('opinion_lexicon')
from nltk.corpus import opinion_lexicon

# Import visualization libraries
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
from wordcloud import WordCloud

# Import utility libraries
import math
from collections import Counter, defaultdict
from tqdm import tqdm; tqdm.pandas()
```

```
In [3]: # Set basic np, pd, and plt output defaults (keeping this code 'clean')
%run -i 'defaults.py'
```

Data Ingestion

```
In [4]: # Instantiate and confirm master dataframe
master_df = pd.read_csv('../data/master.csv')

master_AP_df = pd.read_csv('../data/master_tokenized_AP.csv')
```

```

master_TheHill_df = pd.read_csv('../data/master_business_TheHill.csv')

print(master_df.info())

# Instantiate and confirm test dataframes
master_TheHill_df = master_TheHill_df.rename(columns={
    'Source': 'source_name',
    'Author': 'author',
    'Title': 'title',
    'URL': 'url',
    'date': 'publish_date',
})

master_TheHill_df = master_TheHill_df.drop(
    ['Unnamed: 0.1', 'Unnamed: 0', 'word_count', 'tokens', 'cleaner_text'], axis=1)

master_AP_df = master_AP_df.rename(columns={
    'Source': 'source_name',
    'Author': 'author',
    'Title': 'title',
    'URL': 'url',
    'date': 'publish_date',
    'article_parsed': 'article_text',
})

master_AP_df = master_AP_df.drop(
    ['Unnamed: 0.1', 'Unnamed: 0', 'word_count', 'tokens', 'cleaner_text', 'word_count_to

master_ex_df = pd.concat([master_TheHill_df, master_AP_df])
print(master_ex_df.info())

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4509 entries, 0 to 4508
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  -
0   source_name     4509 non-null  object
1   author          4472 non-null  object
2   title           4509 non-null  object
3   url             4509 non-null  object
4   publish_date    4509 non-null  object
5   content         1158 non-null  object
6   article_text    4508 non-null  object
dtypes: object(7)

```

memory usage: 246.7+ KB

None

```

<class 'pandas.core.frame.DataFrame'>
Index: 369 entries, 0 to 187
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  -
0   source_name     369 non-null  object
1   author          364 non-null  object
2   title           369 non-null  object
3   url             369 non-null  object
4   publish_date    369 non-null  object
5   content         369 non-null  object
6   article_text    369 non-null  object
dtypes: object(7)

```

memory usage: 23.1+ KB

None

In [5]:

```

# "Blanket" label data based on purported source leaning
target_cls_col = 'lean'

def assign_lean(source_name):
    if source_name == 'Breitbart News' or source_name == 'Fox News':

```

```

        return 'right'
    elif source_name == 'CNN' or source_name == 'The Washington Post':
        return 'left'
    else:
        return np.nan
master_df[target_cls_col] = master_df['source_name'].apply(lambda x: assign_lean(x))

```

Tokenization and Cleaning

In [6]:

```

# Stopword removal function, with related initialization
from nltk.corpus import stopwords
sw = stopwords.words('english')
def remove_stop(tokens):
    filtered_tokens = [word for word in tokens if word not in sw]
    return(filtered_tokens)

# Token join back to string
def join_tokens(tokens):
    return ' '.join(tokens)

# Tokenizing function
def tokenize(text):
    return(text.split()) # Tokenize on white space

# Emoji-to-text conversion function
import emoji
def convert_emojis(text):
    # return emoji.demojize(text)
    return emoji.demojize(text).replace('_', ' ')

# Contains-emojis function, with related initialization
all_language_emojis = set()
for country in emoji.EMOJI_DATA :
    for em in emoji.EMOJI_DATA[country]:
        all_language_emojis.add(em)
def contains_emoji(s):
    s = str(s)
    emojis = [ch for ch in s if ch in all_language_emojis]
    return(len(emojis) > 0)

# Punctuation removal function, with related initialization
from string import punctuation
tw_punct = set(punctuation + '\'') - {'#'}
def remove_punct(text, punct_set=tw_punct):
    return(''.join([ch for ch in text if ch not in punct_set]))

# Preparation (pipeline) function
def prepare(text, pipeline):
    tokens = str(text)
    for transform in pipeline:
        tokens = transform(tokens)
    return(tokens)

```

In [7]:

```

# Set pipeline
pipeline = [str.lower, remove_punct, convert_emojis, tokenize, remove_stop]

# Clean and tokenize master dataframe
master_df['article_tokens'] = master_df['article_text'].progress_apply(lambda x: prepare(x, pipeline))
master_df['article_text_tokenized'] = master_df['article_tokens'].progress_apply(lambda x: join_tokens(x))
print(master_df['article_tokens'])
print(master_df['article_text_tokenized'])

```



```
# Also tokenize test dataframes
master_ex_df['article_tokens'] = master_ex_df['article_text'].progress_apply(lambda x: pre
master_ex_df['article_text_tokenized'] = master_ex_df['article_tokens'].progress_apply(lan

100%|████████████████████████████████████████| 4509/4509 [00:09<00:00, 488.19it/s]
100%|████████████████████████████████████████| 4509/4509 [00:00<00:00, 79830.63it/s]
0      [travelers, alabama, driving, interstate, 65, ...
1      [federal, prosecutor, may, nearing, decision, ...
2      [federal, appeals, court, tuesday, cleared, wa...
3      [speaking, orlando, november, 2015, republican...
4                                             [nan]

...
4504    [germanys, populist, alternative, germany, afd...
4505    [president, bidens, justice, department, seemi...
4506    [incumbent, turkish, president, recep, tayyip,...
4507    [throughout, month, may, farleft, cnn, attract...
4508    [disney, known, fighting, antigrooming, legisl...
Name: article_tokens, Length: 4509, dtype: object
0      travelers alabama driving interstate 65 partie...
1      federal prosecutor may nearing decision whethe...
2      federal appeals court tuesday cleared way drug...
3      speaking orlando november 2015 republican pres...
4                                             nan

...
4504    germanys populist alternative germany afd surg...
4505    president bidens justice department seemingly ...
4506    incumbent turkish president recep tayyip erdog...
4507    throughout month may farleft cnn attracted mea...
4508    disney known fighting antigrooming legislation...
Name: article_text_tokenized, Length: 4509, dtype: object
100%|████████████████████████████████████████| 369/369 [00:00<00:00, 428.42it/s]
100%|████████████████████████████████████████| 369/369 [00:00<00:00, 64185.22it/s]
```

Descriptive Stats

In [8]:

```
# Descriptive stats function
def descriptive_stats(tokens, num_tokens=5, verbose=False):
    num_tokens = len(tokens)
    num_unique_tokens = len(set(tokens)) # set() creates unordered set of unique elements
    num_characters = sum(len(token) for token in tokens) # Finds characters sans spaces
    lexical_diversity = num_unique_tokens / num_tokens

    if verbose:
        print(f'There are {num_tokens} tokens in the data.')
        print(f'There are {num_unique_tokens} unique tokens in the data.')
        print(f'There are {num_characters} characters in the data.')
        print(f'The lexical diversity is {lexical_diversity:.3f} in the data.')

    return([num_tokens, num_unique_tokens, lexical_diversity, num_characters])
```

In [9]:

```
# Descriptive stats across all sources
descriptive_stats([token for sublist in master_df['article_tokens'] for token in sublist])
```

Out[9]:

```
[1977106, 84569, 0.0427741355294051, 12724251]
```

In [10]:

```
# Standard dataframe profile for confirmation
profile(master_df)
```

```
100%|████████████████████████████████████████| 4509/4509 [00:00<00:00, 1287677.32it/s]
```

	Dtype	count	unique	na	na%	mean	std	min	max	skew(>=3)	<v0.01	\
source_name	object	4509.0	4.0									
author	object	4472.0	956.0	37.0	0.8							
title	object	4509.0	4509.0									
url	object	4509.0	4509.0									
publish_date	object	4509.0	4487.0									
content	object	1158.0	1158.0	3351.0	74.3							
article_text	object	4508.0	4508.0	1.0								
lean	object	4509.0	2.0									
article_tokens	object	1977106.0	84569.0									
article_text_tokenized	object	4509.0	4509.0									

```
In [11]: # Descriptive stats aggregating function
def aggregate_and_describe(group):
    aggregate_tokens = [token for sublist in group['article_tokens'].tolist() for token in sublist]
    return descriptive_stats(aggregate_tokens)

# Aggregate descriptive stats by source; convert to dataframe; sort and output
grouped_stats = master_df.groupby('source_name').apply(aggregate_and_describe)
grouped_stats_df = pd.DataFrame(grouped_stats.tolist(), index=grouped_stats.index,
                                columns=['num_tokens', 'num_unique_tokens', 'lexical_diversity', 'num_characters'])
grouped_stats_df = grouped_stats_df.sort_index(ascending=False)
print(grouped_stats_df)
```

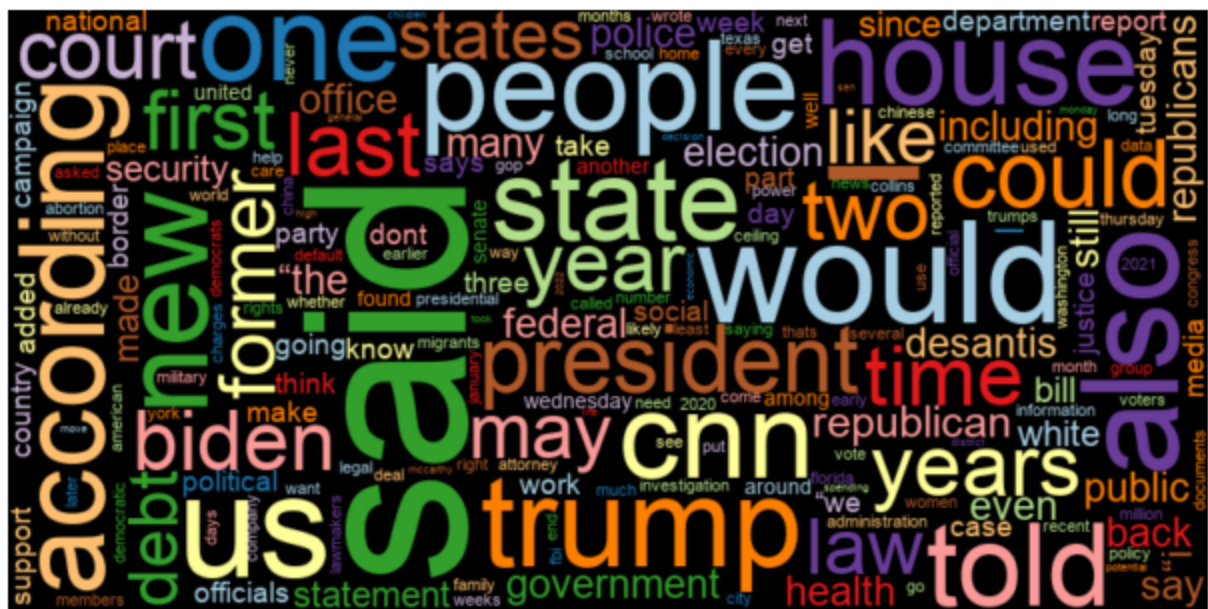
source_name	num_tokens	num_unique_tokens	lexical_diversity	num_characters
The Washington Post	366707	32341	0.09	2370171
Fox News	828739	47097	0.06	5326935
CNN	409422	34724	0.08	2628951
Breitbart News	372238	36815	0.10	2398194

WordCloud

```
In [12]: # Word cloud function
def wordcloud(word_freq, title=None, max_words=200, stopwords=None):
    wc = WordCloud(font_path='/Library/Fonts/Arial.ttf',
                   width=800, height=400,
                   background_color="black", colormap="Paired",
                   max_font_size=150, max_words=max_words)

    # Convert data frame into dict
    if type(word_freq) == pd.Series:
        counter = Counter(word_freq.fillna(0).to_dict())
    else:
        counter = word_freq

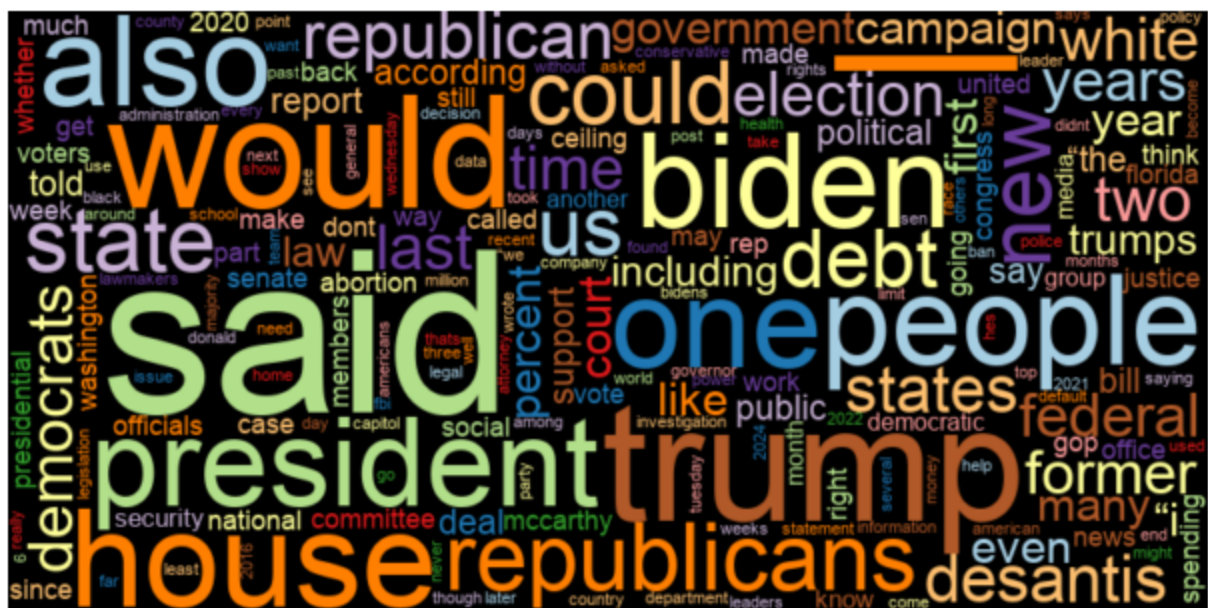
    # filter stop words in frequency counter
    if stopwords is not None:
        counter = {token:freq for (token, freq) in counter.items()
                   if token not in stopwords}
    wc.generate_from_frequencies(counter)
```

Wordcloud for source: Fox News



Wordcloud for source: The Washington Post



Partitioned Data


```
In [14]: # Set splits
train_ratio = 0.7; val_ratio = 0.20; test_ratio = 0.10

# Split and profile
train_df, test_df = train_test_split(master_df, test_size=1-train_ratio,
                                     random_state=42, stratify=master_df[target_cls_col])
val_df, test_df = train_test_split(test_df, test_size=test_ratio/(test_ratio+val_ratio),
                                   random_state=42, stratify=test_df[target_cls_col])
profile_cat(train_df, [target_cls_col])

lean -
right 71.17
left 28.83
```

Topic Modeling

```
In [15]: # Topic summarization function, from BTAP repo
def display_topics(model, features, no_top_words=5):
    for topic, words in enumerate(model.components_):
        total = words.sum()
        largest = words.argsort()[::-1] # invert sort order

        print('\nTopic %02d' % topic, end=':')

        out = []
        for i in range(0, no_top_words):
            out.append(' %s (%2.2f)' % (features[largest[i]], abs(words[largest[i]]*100.0))
        print(';'.join(out), end='')
```

```
In [16]: # Model topics by source
for source_name, group in train_df.groupby('source_name'):
    print(f'Topic modeling for source: {source_name}')

    # Transform article tokens into bag-of-words document-term sparse matrix
    count_vectorizer = CountVectorizer(min_df=0.05, max_df=0.75)
    count_vectors = count_vectorizer.fit_transform(group['article_text_tokenized'])
    # print('Vector shape:', count_vectors.shape)

    lda_model = LatentDirichletAllocation(n_components=5, random_state=42)
    W_lda_matrix = lda_model.fit_transform(count_vectors)
    H_lda_matrix = lda_model.components_

    display_topics(lda_model, count_vectorizer.get_feature_names_out())
    print('\n')

    # Now plot
    num_topics = H_lda_matrix.shape[0]
    words = count_vectorizer.get_feature_names_out()

    # Determine grid size
    grid_size = math.ceil(math.sqrt(num_topics))

    fig, axs = plt.subplots(grid_size, grid_size, figsize=(5, 3))

    # Ensure 1-D array to easily index into subplot grid
    axs = axs.flatten()

    for topic_idx, topic in enumerate(H_lda_matrix):
        # Create dataframe for current topic and sort by importance
        topic_df = pd.DataFrame({'word': words, 'importance': topic})
        topic_df = topic_df.sort_values('importance', ascending=False)
```

```

# Get top 5 words of topic and create word cloud
top_words_dict = {row['word']: row['importance'] for idx, row in topic_df.head(5).iteritems()}
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(top_words_dict)

# Display word cloud for each topic
fig, axs = plt.subplots(5, 1, figsize=(10, 10))

for topic_idx in range(5):
    axs[topic_idx].imshow(wordcloud, interpolation='bilinear')
    axs[topic_idx].axis('off')
    # Set title for each subplot
    axs[topic_idx].set_title(f"Topic #{topic_idx+1}", fontsize=20)

# Turn off unused subplots
for idx in range(num_topics, len(axs)):
    axs[idx].axis('off')

plt.tight_layout()
plt.show()

```

Topic modeling for source: Breitbart News

Topic 00: percent (5.31); desantis (2.53); trump (2.50); news (1.30); president (1.29)
 Topic 01: biden (2.85); border (1.88); house (1.82); migrants (1.29); debt (1.23)
 Topic 02: chinese (0.98); people (0.97); government (0.94); china (0.94); may (0.89)
 Topic 03: 2023 (1.45); women (1.06); children (1.05); may (1.02); news (0.95)
 Topic 04: trump (2.19); president (1.46); think (1.27); thats (1.14); people (1.03)



Topic modeling for source: CNN

Topic 00: people (1.18); health (0.99); new (0.98); one (0.77); like (0.73)
 Topic 01: us (1.97); government (0.81); china (0.71); chinese (0.65); security (0.64)
 Topic 02: police (1.67); according (1.33); cnn (1.17); told (1.02); people (0.79)
 Topic 03: trump (2.55); desantis (1.70); former (1.29); president (1.16); court (1.08)
 Topic 04: house (2.23); debt (1.78); would (1.68); biden (1.12); bill (1.12)



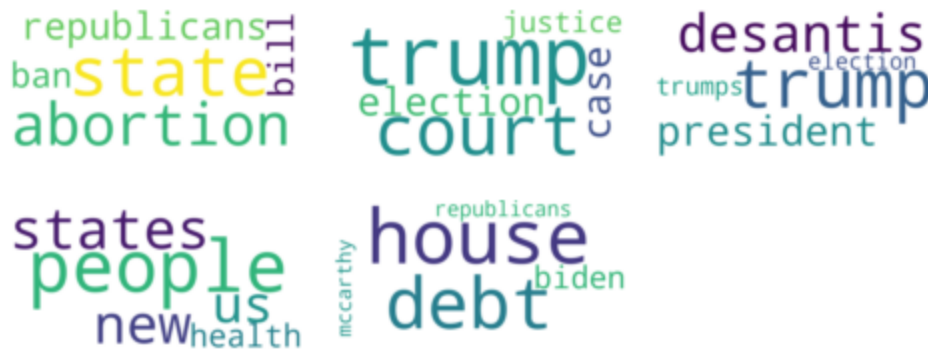
Topic modeling for source: Fox News

Topic 00: ai (1.14); people (1.09); also (0.85); us (0.80); like (0.77)
 Topic 01: trump (2.19); president (1.75); desantis (1.59); former (1.31); campaign (1.06)
 Topic 02: biden (2.87); house (2.38); president (1.46); debt (1.14); fbi (1.12)
 Topic 03: border (2.02); state (1.98); school (1.41); law (1.38); migrants (1.21)
 Topic 04: police (1.77); according (0.92); told (0.90); two (0.79); one (0.78)



Topic modeling for source: The Washington Post

Topic 00: state (1.91); abortion (1.39); republicans (0.94); bill (0.84); ban (0.81)
 Topic 01: trump (1.31); court (0.92); election (0.76); case (0.74); justice (0.72)
 Topic 02: trump (3.52); desantis (1.91); president (1.06); trumps (0.85); election (0.83)
 Topic 03: people (1.09); states (0.68); new (0.63); us (0.56); health (0.54)
 Topic 04: house (2.33); debt (2.05); biden (1.77); republicans (1.29); mccarthy (1.19)



In [17]:

```
# Now do topic modeling on full dataset to find latent groupings
count_vectorizer = CountVectorizer(min_df=0.05, max_df=0.75)
count_vectors = count_vectorizer.fit_transform(train_df['article_text_tokenized'])

n_sources = train_df['source_name'].nunique()
lda_model = LatentDirichletAllocation(n_components=n_sources, random_state=42)
lda_output = lda_model.fit_transform(count_vectors) # shape: (n_documents, n_topics)

num_topics = lda_model.components_.shape[0]
words = count_vectorizer.get_feature_names_out()

# Plot
grid_size = math.ceil(math.sqrt(num_topics))
fig, axs = plt.subplots(grid_size, grid_size)

# 1-D array to easily index into subplot grid
axs = axs.flatten()

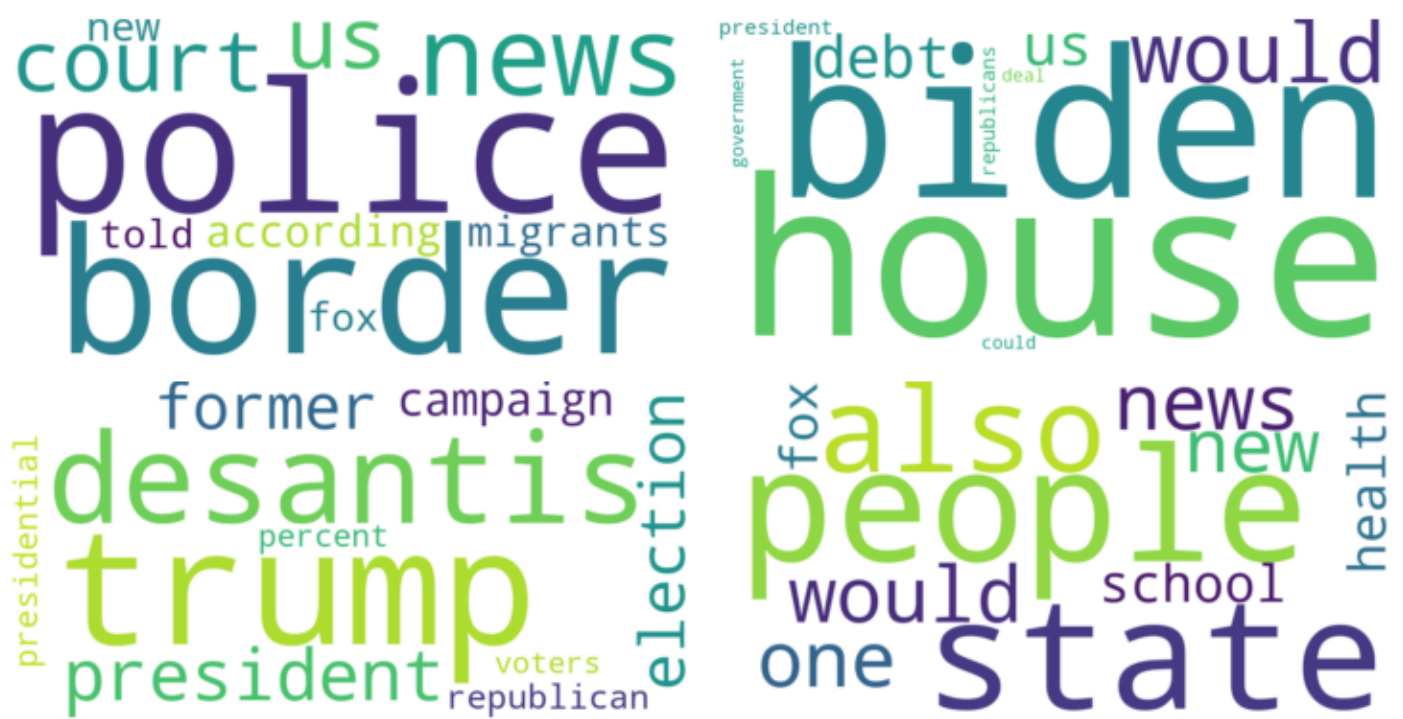
for topic_idx, topic in enumerate(lda_model.components_):
    # Create dataframe for current topic and sort by importance
    topic_df = pd.DataFrame({'word': words, 'importance': topic})
    topic_df = topic_df.sort_values('importance', ascending=False)

    # Get top 10 words of the topic and create a word cloud
    top_words_dict = {row['word']: row['importance'] for idx, row in topic_df.head(10).iterrows()}
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(top_words_dict)

    axs[topic_idx].imshow(wordcloud, interpolation='bilinear')
    axs[topic_idx].axis('off')

# Turn off unused cells in plot
for idx in range(num_topics, len(axs)):
    axs[idx].axis('off')

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

Text Summarization and Sentiment Analysis

```
In [18]: # NLTK opinion lexicon
positive_words = set(opinion_lexicon.positive())
negative_words = set(opinion_lexicon.negative())
```

```
In [19]: # List of "assumed" political phrases
political_phrases = ['gun rights', 'voting rights', 'climate change', 'immigration reform',
                    'tax cuts', 'universal healthcare',
                    'criminal justice reform', 'income inequality']
```

```
In [20]: # Group train_df by 'source_name' for source-level comparison
grouped_df = train_df.groupby('source_name')
#grouped_df = master_ex_df.groupby('source_name')

# Instantiate BERT pipeline
nlp = pipeline('sentiment-analysis')

# Create dictionaries for scores
political_phrase_scores = {}
sentiment_scores = {}
sentiment_scores_T = {}

# Iterate over sources and calc TF-IDF scores vs. political phrases
for source, group in tqdm(grouped_df):
    tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 3))
    tfidf_vectors = tfidf_vectorizer.fit_transform(group['article_text_tokenized'])

    # Calc TF-IDF sum (scores) where political phrases found
    scores = {}
    sentiment = defaultdict(lambda: defaultdict(int))
    sentiment_T = defaultdict(lambda: defaultdict(int))

    # Iterate over political phrases
    for phrase in political_phrases:
        try:
            index = tfidf_vectorizer.get_feature_names_out().tolist().index(phrase) # try
```



```

# Categorize based on relative_to_median (otherwise arbitrary)
if relative_to_median > 10:
    category = 'high'
elif 5 < relative_to_median <= 10:
    category = 'medium'
else:
    category = 'low'

sentiment_phrase = sentiment.get(phrase, {'positive': 0, 'negative': 0})
sentiment_phrase_T = sentiment_T.get(phrase, {'positive': 0, 'negative': 0})
results.append({
    'source_name': source_name,
    'phrase': phrase,
    'score': score,
    'relative_to_mean': relative_to_mean,
    'relative_to_median': relative_to_median,
    'category': category,
    'p_sentiment': sentiment_phrase['positive'],
    'p_sentiment_T': sentiment_phrase_T['positive'],
    'n_sentiment': sentiment_phrase['negative'],
    'n_sentiment_T': sentiment_phrase_T['negative'],
    'sentiment': sentiment_phrase['positive'] + (sentiment_phrase['negative'] * -1),
    'sentiment_T': sentiment_phrase_T['positive'] + (sentiment_phrase_T['negative'] * -1)
})

for result in results:
    result['sentiment_label'] = 'positive' \
        if result['sentiment'] > 0 else 'negative' if result['sentiment'] < 0 else 'neutral'
    result['sentiment_label_T'] = 'positive' \
        if result['sentiment_T'] > 0 else 'negative' if result['sentiment_T'] < 0 else 'neutral'

# Sort results by score
results.sort(key=lambda x: x['score'], reverse=True)

# Print sorted results
for result in results:
    print(f"{result['phrase']}: {result['category']} importance ",
          f"{result['sentiment_label']} sentiment")

results_df = pd.concat([results_df, pd.DataFrame(results)])

```

Scores for Breitbart News:

```

climate change: high importance negative sentiment
tax cuts: high importance positive sentiment
gun rights: high importance negative sentiment
immigration reform: high importance positive sentiment
criminal justice reform: low importance negative sentiment
universal healthcare: low importance negative sentiment
voting rights: low importance neutral sentiment
income inequality: low importance neutral sentiment

```

Scores for CNN:

```

climate change: high importance negative sentiment
voting rights: medium importance negative sentiment
tax cuts: medium importance negative sentiment
criminal justice reform: low importance negative sentiment
gun rights: low importance negative sentiment
income inequality: low importance negative sentiment
immigration reform: low importance neutral sentiment
universal healthcare: low importance neutral sentiment

```

Scores for Fox News:

```

climate change: high importance negative sentiment
voting rights: high importance positive sentiment

```

immigration reform: high importance positive sentiment
gun rights: high importance negative sentiment
tax cuts: high importance negative sentiment
criminal justice reform: high importance negative sentiment
income inequality: medium importance positive sentiment
universal healthcare: low importance neutral sentiment

Scores for The Washington Post:

voting rights: high importance negative sentiment
tax cuts: high importance negative sentiment
climate change: high importance negative sentiment
immigration reform: low importance positive sentiment
criminal justice reform: low importance positive sentiment
gun rights: low importance negative sentiment
universal healthcare: low importance neutral sentiment
income inequality: low importance neutral sentiment

In [22]:

```
# Sort DataFrame by 'source' and 'phrase' to match order of bars in plot
sorted_df = results_df.sort_values(['source_name', 'phrase'])

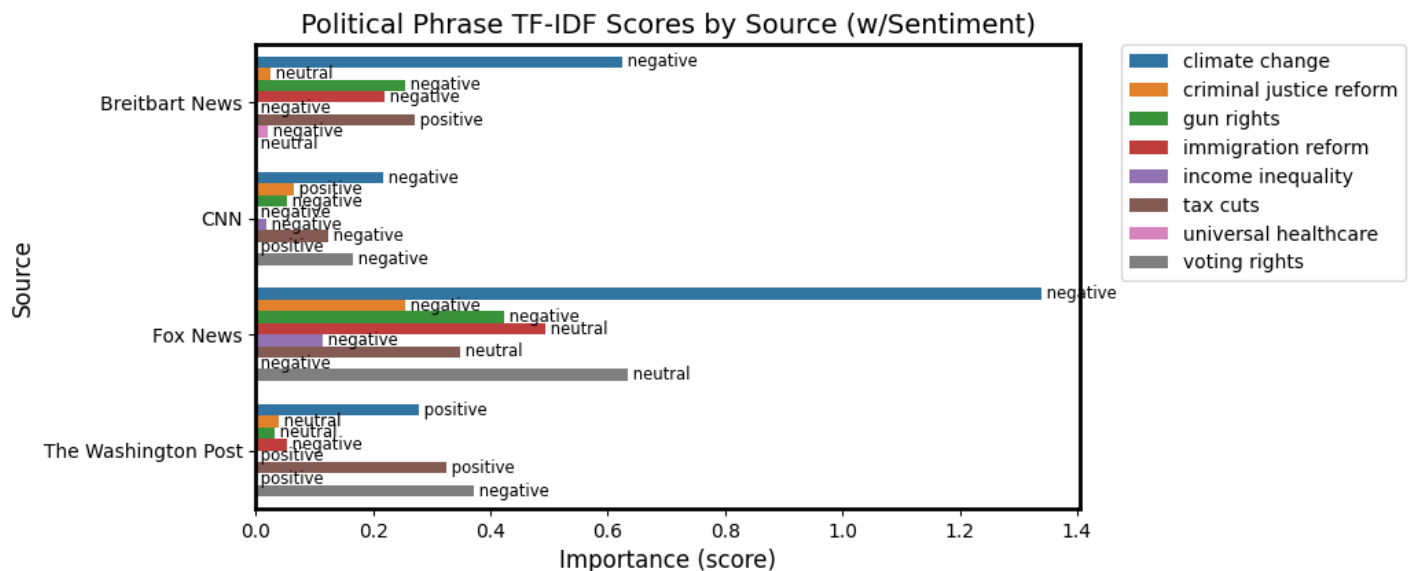
# Create barplot
fig, ax = plt.subplots()
sns.barplot(data=sorted_df, y='source_name', x='score',
            ax=ax, hue='phrase', errorbar=None)

# Iterate over bars and dataframe rows to add sentiment
for p, (_, row) in zip(ax.patches, sorted_df.iterrows()):
    plt.text(p.get_width(), p.get_y() + p.get_height()/2,
            f' {row["sentiment_label"]}',
            ha='left', va='center', fontsize='smaller')

ax.set_title('Political Phrase TF-IDF Scores by Source (w/Sentiment)')
ax.set_xlabel('Importance (score)')
ax.set_ylabel('Source')

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()
```



In [23]:

```
# Sentiment comparison plot function
def create_plot(df, sentiment_label, title):
    # Sort DataFrame by 'source' and 'phrase' to match order of bars in plot
    sorted_df = df.sort_values(['source_name', 'phrase'])

    # Create plot
```

```

fig, ax = plt.subplots()

# Define a custom color palette
custom_palette = ['#FF7F0E', '#1F77B4', '#2CA02C', '#D62728', '#9467BD',
                  '#8C564B', '#E377C2', '#7F7F7F', '#BCBD22', '#17BECF']

# Ensure the colors are consistent
hue_order = sorted_df['phrase'].unique()
palette = [custom_palette[i % len(custom_palette)] for i in range(len(hue_order))]

# List to store the created patches
patches = []

# Iterate over dataframe rows to add bars
for i, (_, row) in enumerate(sorted_df.iterrows()):
    # Adjust score so that 'negative' sentiments display to the left of the y-axis
    score = row['score'] if row[sentiment_label] != 'negative' else row['score'] * -1

    color = palette[np.where(hue_order == row['phrase'])[0][0]]
    patch = ax.barh(row['source_name'], score, color=color)
    patches.append(patch)

ax.set_title(title)
ax.set_ylabel('Source')

# Create legend
handles = [plt.Rectangle((0,0),1,1, color=palette[i]) for i in range(len(hue_order))]
plt.legend(handles, hue_order, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

# Center x-axis
max_abs_score = abs(sorted_df['score']).max()
ax.set_xlim([-max_abs_score, max_abs_score])
ax.axvline(0, color='black') # draw y-axis line

# Remove x-axis labels and add custom labels
ax.xaxis.set_ticklabels([])

ax.annotate('Negative Sentiment', xy=(-max_abs_score, 0), xytext=(0,-10),
           xycoords=('data', 'axes fraction'), textcoords='offset points',
           ha='left', va='top', fontsize='smaller')

ax.annotate('Positive Sentiment', xy=(max_abs_score, 0), xytext=(0,-10),
           xycoords=('data', 'axes fraction'), textcoords='offset points',
           ha='right', va='top', fontsize='smaller')

plt.show()

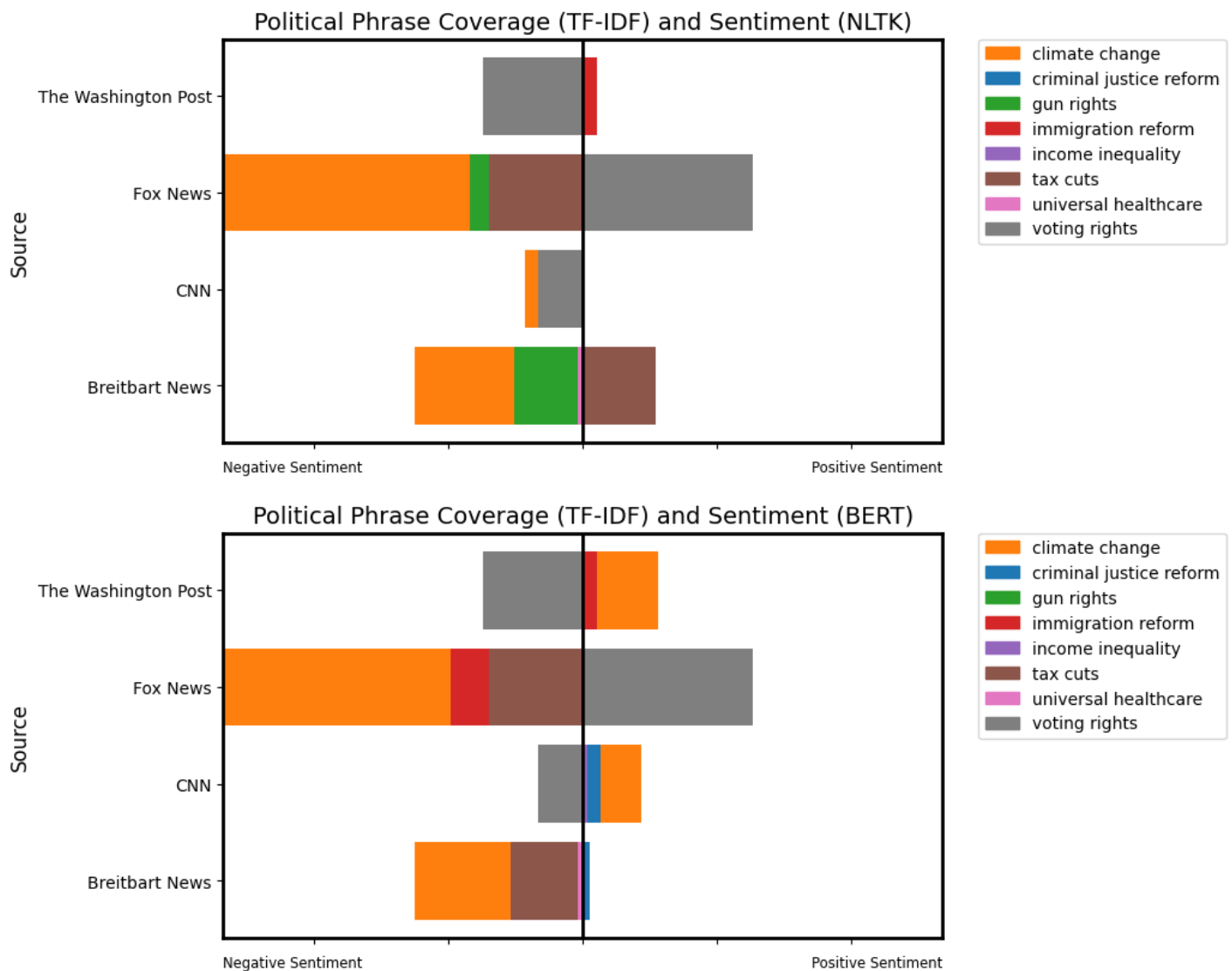
```

In [24]:

```

create_plot(results_df, 'sentiment_label', 'Political Phrase Coverage (TF-IDF) and Sentiment')
create_plot(results_df, 'sentiment_label_T', 'Political Phrase Coverage (TF-IDF) and Sentiment')

```



In [25]:

```
# Sort DataFrame by 'source' and 'phrase' to match order of bars in plot
sorted_df = results_df.sort_values(['source_name', 'phrase'])

# Create barplot
fig, ax = plt.subplots()
sns.barplot(data=sorted_df, y='source_name', x='score',
            ax=ax, hue='phrase', errorbar=None)

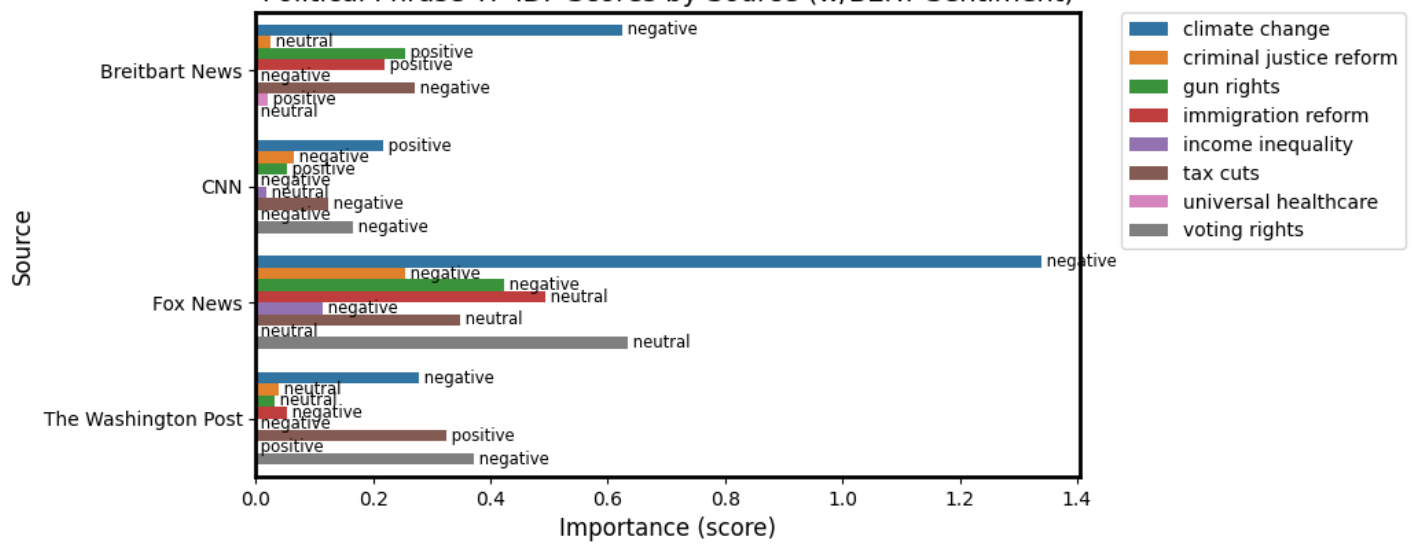
# Iterate over bars and dataframe rows to add sentiment
for p, (_, row) in zip(ax.patches, sorted_df.iterrows()):
    plt.text(p.get_width(), p.get_y() + p.get_height()/2,
            f' {row["sentiment_label_T"]}',
            ha='left', va='center', fontsize='smaller')

ax.set_title('Political Phrase TF-IDF Scores by Source (w/BERT Sentiment)')
ax.set_xlabel('Importance (score)')
ax.set_ylabel('Source')

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()
```

Political Phrase TF-IDF Scores by Source (w/BERT Sentiment)



In [26]:

```
# Sort DataFrame by political 'lean' and 'phrase' to match order of bars in plot
lean_df = train_df[['source_name', 'lean']].drop_duplicates()
results_df = pd.merge(results_df, lean_df, on='source_name', how='left')
sorted_df = results_df.sort_values(['lean', 'phrase'])

# Create barplot
fig, ax = plt.subplots()
sns.barplot(data=sorted_df, y='lean', x='score',
            ax=ax, hue='phrase', errorbar=None)

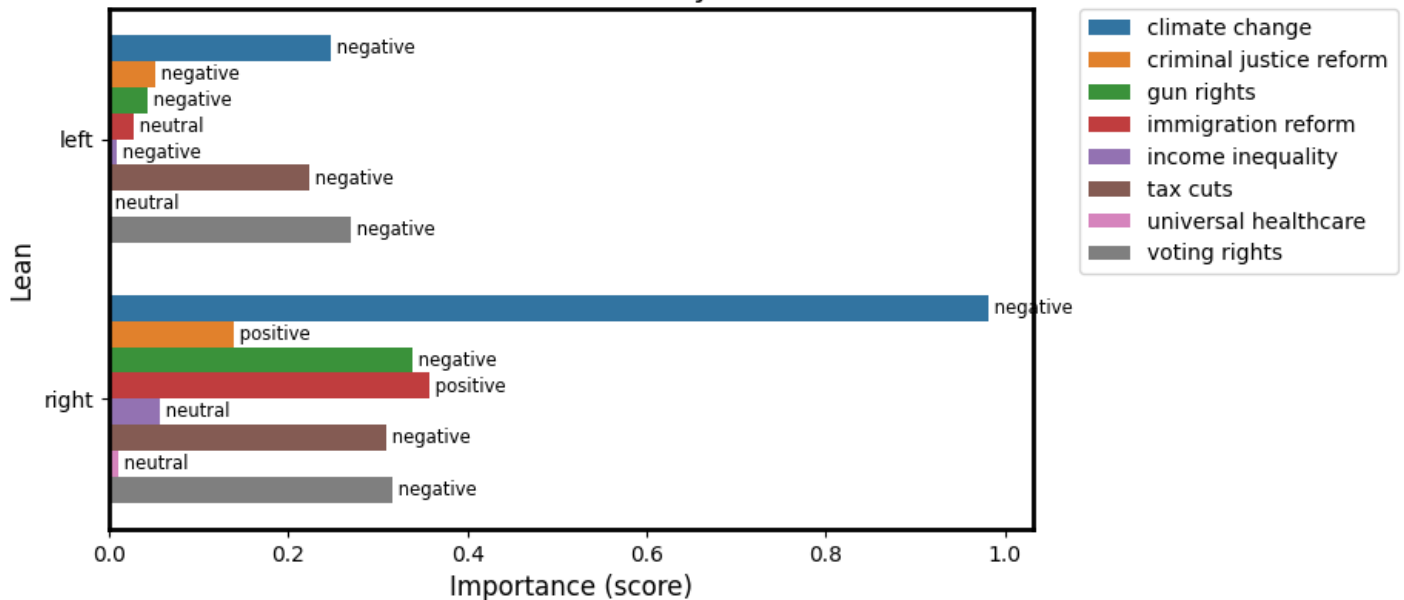
# Iterate over bars and dataframe rows to add sentiment
for p, (_, row) in zip(ax.patches, sorted_df.iterrows()):
    plt.text(p.get_width(), p.get_y() + p.get_height()/2,
            f' {row["sentiment_label"]}',
            ha='left', va='center', fontsize='smaller')

ax.set_title('Political Phrase TF-IDF Scores by Political Lean')
ax.set_xlabel('Importance (score)')
ax.set_ylabel('Lean')

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()
```

Political Phrase TF-IDF Scores by Political Lean



Modeling

In [27]:

```
import warnings
warnings.filterwarnings('ignore')

# Define labels
target_labels = master_df[target_cls_col].unique().tolist()

# Dummy function to leverage existing tokenization; and tokenize
def identity_tokenizer(tokens):
    return tokens
tfidf_vectorizer = TfidfVectorizer(tokenizer=identity_tokenizer, preprocessor=lambda x: x,
                                   lowercase=False, ngram_range=(1, 3))

# Define train/val/test per original split
X_train = tfidf_vectorizer.fit_transform(train_df['article_text_tokenized'])
y_train = train_df[target_cls_col]
X_val = tfidf_vectorizer.transform(val_df['article_text_tokenized'])
y_val = val_df[target_cls_col]
X_test = tfidf_vectorizer.transform(test_df['article_text_tokenized'])
y_test = test_df[target_cls_col]

# Finally, convert sparse matrices to dataframes as required by ModelProcess class
X_train = pd.DataFrame.sparse.from_spmatrix(X_train, columns=tfidf_vectorizer.get_feature_names())
X_val = pd.DataFrame.sparse.from_spmatrix(X_val, columns=tfidf_vectorizer.get_feature_names())
X_test = pd.DataFrame.sparse.from_spmatrix(X_test, columns=tfidf_vectorizer.get_feature_names())
```

In [28]:

```
# Set model list
mp_queue = (
    (LogisticRegression(), {'C': 0.01, 'class_weight': 'balanced', 'random_state': 42}),
    (NearestCentroid(), {}),
)
```

In [29]:

```
import copy

ModelProcess.show_progress = True

# Iterate models (note use of 'copy' is to preserve mutable elements
# of model_queue tuple for possible later use)
mp_df = pd.DataFrame(mp_queue, columns=['algorithm', 'params'])
mp_df['mp'] = mp_df.apply(
    lambda mp: ModelProcess(copy.deepcopy(mp['algorithm']), None,
                             copy.copy(mp['params']),
                             X_train, y_train,
                             X_val, y_val,
                             X_test, y_test,
                             labels=target_labels,
                             cat_cols=None, num_cols=None,
                             balance_target=True).train_validate_test(), axis=1)

# Compile, sort, and display results
mp_df[['train_acc', 'train_f1', 'train_time',
        'val_acc', 'val_f1', 'val_time',
        'test_acc', 'test_f1', 'test_time']] = \
    mp_df['mp'].apply(
        lambda mp: sum(list(map(
            lambda dataset: mp.score[dataset] + [mp.time[dataset]], ['train', 'val', 'test']
        )))
mp_df.sort_values(by=['train_f1', 'val_f1', 'test_f1'],
                  ascending=[False, False, False], inplace=True)
mp_df.loc[:, mp_df.columns != 'mp'].to_csv('results_table.csv')
```



```
print()  
mp_df
```

NearestCentroid: train... done in 7.64s.
NearestCentroid: val... done in 0.82s.
NearestCentroid: test... done in 0.68s.

Out[29]:

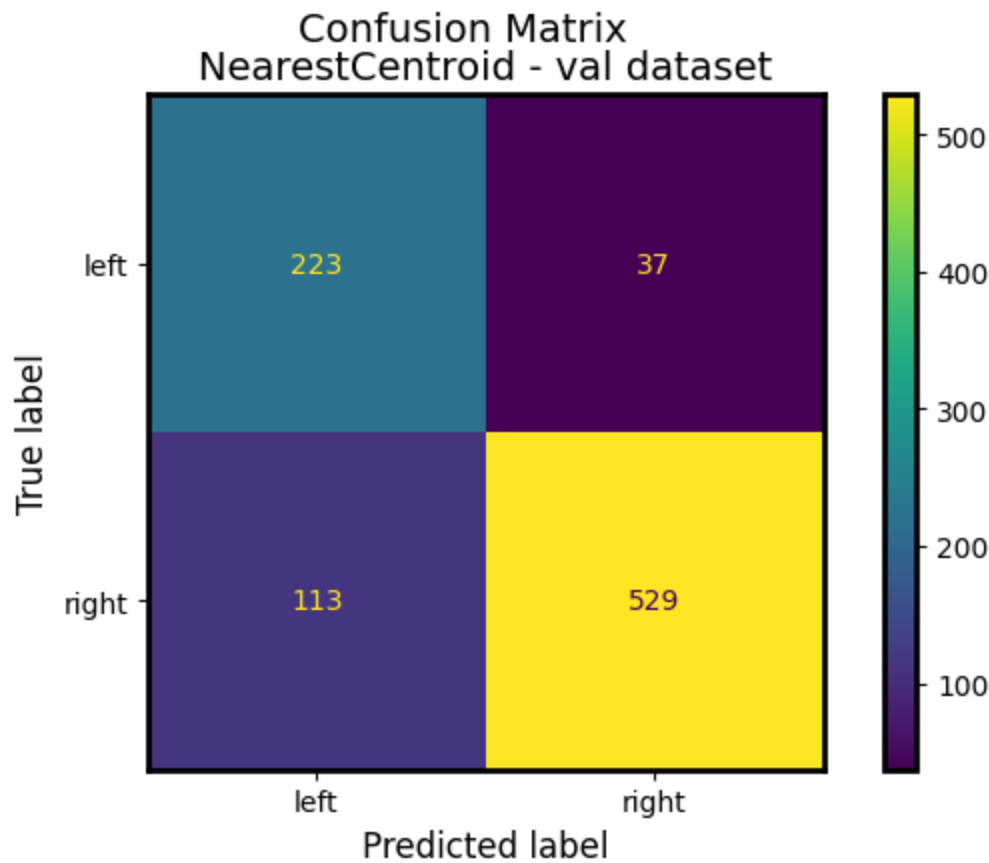
	algorithm	params	mp	train_acc	train_f1	train_time	val_acc	val_f1	va
0	NearestCentroid()	{}	<model_process.ModelProcess object at 0x7fd439...	0.84	0.76	7.64	0.83	0.75	

In [30]:

```
# Select model (process) for focus and tuning  
select_mp = 0
```

In [31]:

```
# Show confusion matrix and summary for top Neural Network  
mp_df.loc[select_mp]['mp'].confusion_matrix('val')  
mp_df.loc[select_mp]['mp'].summary('val')
```



NearestCentroid - val dataset

	precision	recall	f1-score	support
left	0.66	0.86	0.75	260
right	0.93	0.82	0.88	642
accuracy			0.83	902
macro avg	0.80	0.84	0.81	902
weighted avg	0.86	0.83	0.84	902

In []:

