Sentiment and Thematic Analyses of British Politician Speeches

Kanav Bhagat 730013149

importaing important libraries that will be used for the experimentations

In [15]:
```
import pandas as pd
import nltk

from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.util import ngrams

import spacy
from gensim import corpora
from gensim.models import LdaModel
import gensim
from nltk import pos_tag


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

Downloading important modules from nltk library

In [ ]:
```
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('vader_lexicon')
nltk.download('averaged_perceptron_tagger')
```

Loading the Dataset into notebook

In [ ]:
```
#Load the Dataset into datafram
Speech_data = pd.read_csv('SpeechData_with_Party.csv')
print(Speech_data.head())

'''
# Counting the number of speeches by Conservative, Labour, and Other p
speech_count_by_party = Speech_data['Party'].value_counts()
print("Number of speeches by each party:\n", speech_count_by_party)
'''
```

Speeches Dataset Overview

Below is a preview of the dataset containing speeches from British politicians, categorized by their name, title, URL, text, and party affiliation.

Politician Name	Speech Title	Speech URL	Party
Scott Benton	2024 Resignation Statement	Link (https://www.ukpol.co.uk/scott-benton-2024-resignation...)	Other
Rishi Sunak	2024 Speech on Extremism	Link (https://www.ukpol.co.uk/rishi-sunak-2024-speech-on-ext...)	Conservative
Michael Gove	2024 Speech at the Convention of the North	Link (https://www.ukpol.co.uk/michael-gove-2024-speech-at-th...)	Conservative
Oliver Dowden	2024 Speech on AI for Public Good	Link (https://www.ukpol.co.uk/oliver-dowden-2024-speech-on-a...)	Conservative
Stuart Andrew	2024 Speech at the Beacon Philanthropy and Impact Forum	Link (https://www.ukpol.co.uk/stuart-andrew-2024-speech-at-t...)	Conservative

This table provides a quick look at the diversity of topics and party representation in our dataset. Each speech's URL is linked for easy access to the full text.

Performing Sentiment Analysis

In [18]:
```
#Calling the sentiment Analyser
sia = SentimentIntensityAnalyzer()

def analyze_sentiment_vader(text):
    sentiment_score = sia.polarity_scores(text)
    return sentiment_score
```

In [19]:
```
#Applying the Sentiment Analysis over the speeches.
#No need to apply any pre-processing since Vader is efficient to work
sentiment_df = Speech_data['Speech Text'].apply(analyze_sentiment_vade
Speech_data[['Neg', 'Neu', 'Pos', 'Compound']] = pd.json_normalize(sen
```

In [ ]:
```
#Finding Average compound of all the speeches
average_compound = Speech_data['Compound'].mean()
print(f"Average Compound Sentiment Score: {average_compound:.3f}")
```

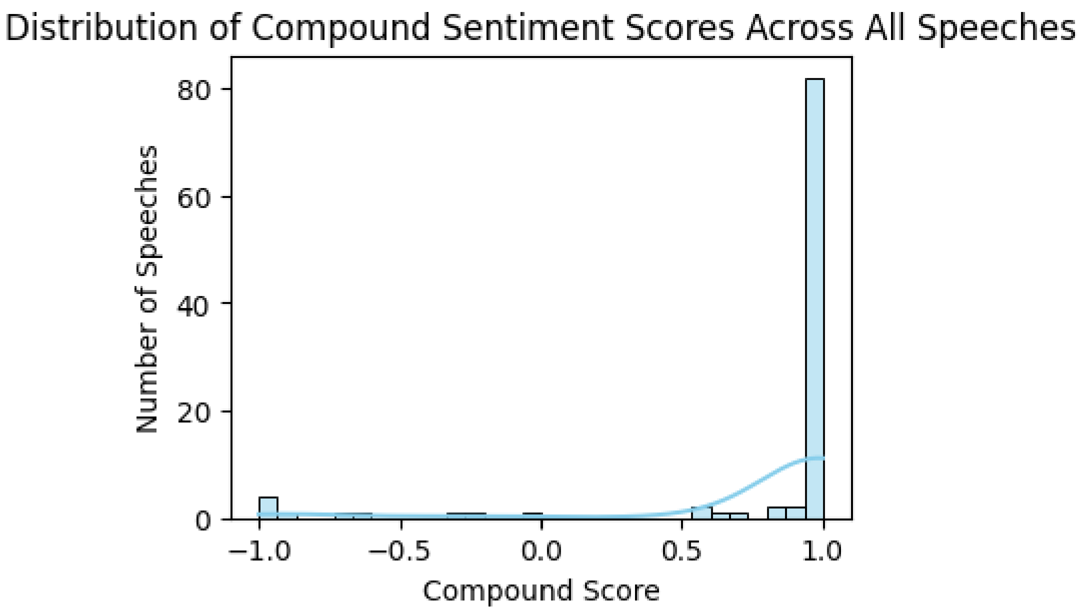
Average Compound Sentiment Score

The overall average compound sentiment score across all analyzed speeches is **0.808**. This score indicates a generally positive sentiment within the political speeches in our dataset.

Analysing the data using the graphs

In [ ]:
```
#Plotting Sentiment distribution across all the speeches.
plt.figure(figsize=(4, 3))
sns.histplot(Speech_data['Compound'], bins=30, kde=True, color='skyblu
plt.title('Distribution of Compound Sentiment Scores Across All Speech
plt.xlabel('Compound Score')
plt.ylabel('Number of Speeches')
plt.show()
```

Graph representing Distribution of Compound Sentiment Scores Across All Speeches



In [ ]:
```
# Calculating average sentiment scores for each political party
avg_sentiment_by_party = Speech_data.groupby('Party')[['Compound']].me

print(avg_sentiment_by_party)

plt.figure(figsize=(9, 3))
plt.subplot(1, 2, 2)
sns.barplot(x='Party', y='Compound', data=avg_sentiment_by_party, pale
plt.title('Average Compound Sentiment')
plt.xlabel('Party')
plt.ylabel('Average Score')
```

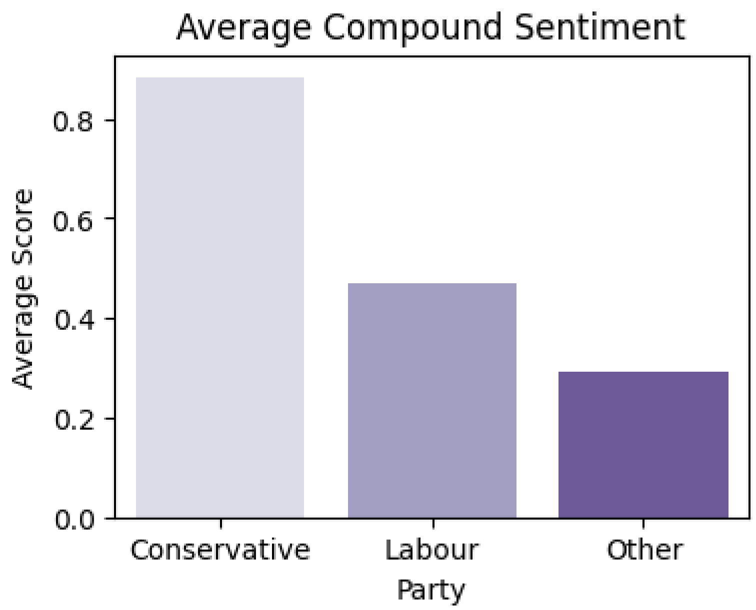
Results for sentiments based upon different parties

Below is the summary of average compound sentiment scores by party:

Party	Compound
Conservative	0.884162
Labour	0.467271
Other	0.290737

Visual Representation

For a more detailed analysis, refer to the graph below:



In [ ]:
```
# Identifying speeches with the highest and lowest compound scores
highest_sentiment_speech = Speech_data.loc[Speech_data['Compound'].idx
lowest_sentiment_speech = Speech_data.loc[Speech_data['Compound'].idxm

print(f"Highest Sentiment Speech: '{highest_sentiment_speech['Speech T
print(f"Lowest Sentiment Speech: '{lowest_sentiment_speech['Speech Ti
```

Speech Sentiment Extremes

Highest Sentiment Speech

- **Title:** 2024 Budget Speech
- **Politician:** Jeremy Hunt
- **Party:** Conservative
- **Sentiment Score (Compound):** 1.0

Lowest Sentiment Speech

- **Title:** 2024 Speech on Freedom and Democracy in Iran
- **Politician:** Bob Blackman
- **Party:** Conservative
- **Sentiment Score (Compound):** -0.9998

Performing Themetic Analysis

In [24]:
```
nlp = spacy.load("en_core_web_sm")

#Funtion preprocess_text pre-processes the text data by various diffe
"""
- It lowercase the text,
- Remove the first sentence since first sentence of each speech tells
- Tokenize the text
- Remove non-alphanumeric characters and stopwords. Added a list of co
- Lemmatize the text
- Generate bigrams and multi-word texts

"""
def preprocess_text(text):
    text = text.lower()
    doc = nlp(text)
    entities = [entity.text for entity in doc.ents]
    # Combine entity tokens back into the text
    text_with_entities = ' '.join(entities) + ' ' + text

    # Remove the first sentence (usually metadata in political speeche
    sentences = nltk.sent_tokenize(text_with_entities)
    if len(sentences) > 1:
        text = ' '.join(sentences[1:])

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove non-alphanumeric characters and stopwords
    common_stop_words = {"also","new","year","0o", "0s", "3a", "3b", '
    stop_words = set(stopwords.words('english')).union(common_stop_wo
    tokens = [word for word in tokens if word.isalnum() and word not i

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

    # Generate and add bigrams to capture multi-word themes
    bigrams = ['_'.join(gram) for gram in ngrams(tokens, 2)]
    tokens.extend(bigrams)

    return ' '.join(tokens)
```

Preprocessing the speeches based upon the political party

In [25]:
```
#Pre-process the speeches based upon the political party
Speech_data['Titile and Speech'] = Speech_data['Speech Title'] + ". "

pre_processed_conservative= Speech_data[Speech_data['Party'] == 'Conse
pre_processed_labour= Speech_data[Speech_data['Party'] == 'Labour']['T
pre_processed_other= Speech_data[Speech_data['Party'] == 'Other']['Ti
```

Function to process texts before we can apply LDA.

In [26]:
```
#Funtion to process texts before we can apply LDA.
#This function builds the dictionary and a corpus from the speech data
def process_speeches(preprocessed_speeches):
    # Tokenize the preprocessed speeches
    # Initialize an empty list to hold the filtered tokens
    texts = []

    for speech in preprocessed_speeches:
        # Tokenize the preprocessed speech
        tokens = word_tokenize(speech)

        # Perform POS tagging on the tokens
        tagged_tokens = pos_tag(tokens)

        # Filter tokens based on POS tags (keeping only nouns and verb
        allowed_pos_tags = ['NN', 'NNS', 'NNP', 'NNPS', 'VB', 'VBD',
        important_words = [word for word, tag in tagged_tokens if tag

        texts.append(important_words)

    # Create a dictionary representation of the documents, and filter
    dictionary = corpora.Dictionary(texts)
    dictionary.filter_extremes(no_below=2, no_above=0.9)

    # Convert document into the bag-of-words (BoW) format
    corpus = [dictionary.doc2bow(text) for text in texts]

    if len(corpus) == 0 or len(dictionary) == 0:
        print("Warning: Corpus or dictionary is empty. Consider adjus


    return corpus, dictionary
```

LDA function to perform themetic Analysis

In [27]:
```
#Function to create LDA model with corpus and dictionary. We will gene
def create_lda_model(corpus, dictionary, num_topics=5, passes=15):
    model = LdaModel(
        corpus=corpus,
        id2word=dictionary,
        chunksize=100,
        alpha='auto',
        eta='auto',
        iterations=400,
        num_topics=num_topics,
        passes=passes
    )

    # Get topics from the model
    topics = model.print_topics(num_words=4)

    return topics
```

Getting the themes for different parties using LDA

In [28]:
```
#Applying the LDA to our speeches filtered upon parties.

corpus_conservative, dictionary_conservative, = process_speeches(pre_p
Keywords_conservative = create_lda_model(corpus_conservative, dictiona

corpus_labour, dictionary_labour, = process_speeches(pre_processed_lab
Keywords_labour = create_lda_model(corpus_labour, dictionary_labour, n

corpus_other, dictionary_other, = process_speeches(pre_processed_other
Keywords_other = create_lda_model(corpus_other, dictionary_other, num_
```

In [ ]:
```
#Printing the results
print("Conservative")
for i in Keywords_conservative:
    print(i)

print("Labour")
for i in Keywords_labour:
    print(i)

print("Other")
for i in Keywords_other:
    print(i)
```

Thematic Analysis Results

Conservative Themes

1. 0.012*"people" + 0.010*"investment" + 0.010*"business" + 0.009*"work"
2. 0.010*"ireland" + 0.007*"northern_ireland" + 0.007*"government" + 0.007*"defence"
3. 0.014*"gambling" + 0.008*"government" + 0.008*"ireland" + 0.008*"consultation"
4. 0.016*"woman" + 0.013*"government" + 0.012*"people" + 0.009*"service"
5. 0.013*"food" + 0.009*"country" + 0.008*"child" + 0.008*"today"

Labour Themes

1. 0.085*"government" + 0.055*"authority" + 0.047*"funding" + 0.036*"care"
2. 0.060*"people" + 0.058*"politics" + 0.046*"country" + 0.042*"britain"
3. 0.077*"ireland" + 0.060*"northern_ireland" + 0.042*"minister" + 0.030*"market"
4. 0.039*"member" + 0.029*"government" + 0.029*"hon_member" + 0.026*"house"

5. `0.005*"house" + 0.005*"member" + 0.005*"government" + 0.005*"hope"`

Other Party Themes

1. `0.045*"continue" + 0.045*"return" + 0.045*"comment" + 0.024*"life"`
2. `0.051*"community" + 0.051*"mp" + 0.041*"government" + 0.041*"country"`
3. `0.064*"house" + 0.045*"party" + 0.039*"opposition" + 0.039*"leader"`
4. `0.064*"office" + 0.049*"meet" + 0.049*"minister" + 0.034*"court"`