TEAM 8 ANALYTIQ BANK OF ENGLAND
SPEECH SENTIMENT ANALYSIS – TECHNICAL REPORT

Lalitha Vemuri, Christina Tsoulfa, Reka Bodo, Yann Hirsig, Louis Pang, Dr. Karin Agius Ferrante June 2nd, 2025

BUSINESS SCENARIO

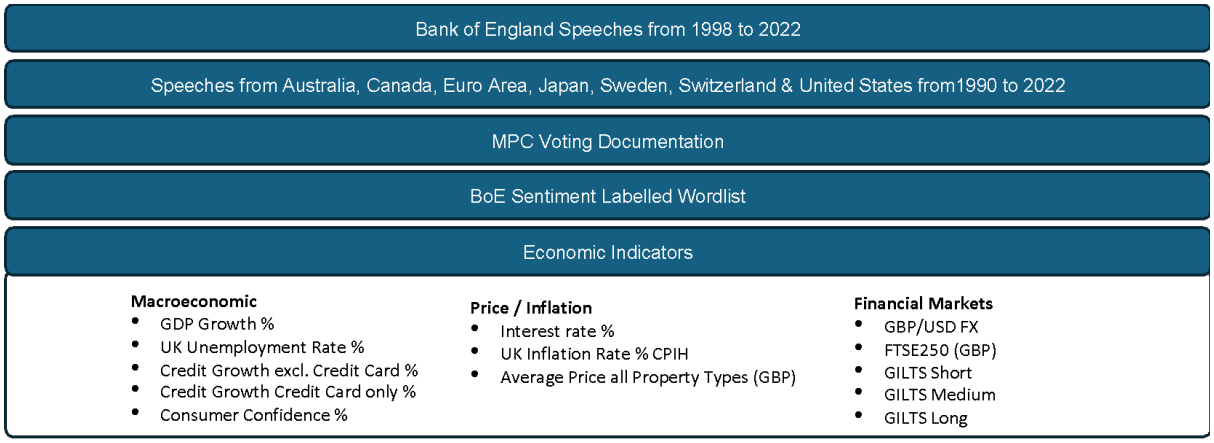
Bank of England (BoE): UK's central bank and one of the world's leading financial institutions.
Objective: Maintain economic and financial stability. Support the UK government's economic policies. Communicate with the public and markets through formal speeches to offer guidance, manage expectations, and provide clarity in times of uncertainty.
Issue: Impact of speeches on economic indicators and market behaviour not fully understood.
Impact: Effectiveness of the speeches cannot be assessed.
Approach: Explore whether sentiment and timing of BoE speeches hold analytical or predictive value compared with key events.

Business questions

- How do speeches and sentiment develop over time?
- How does speech sentiment relate to economic events?
- How is speech sentiment correlated with economic indicators?
- Does sentiment influence economic indicators or react to them?
- Can sentiment trends predict market behaviour?
- What broader insights can improve BoE's communication strategy?

ANALYTICAL APPROACH

Datasets



- *all_speeches.csv* - focus on BoE speeches filtered in country for 'united kingdom'
- *LSE_DA_BoE_Employer_project_Sentiment-labelled_wordlist-2.xlsx*
- *mpcvoting.xlsx*
- *Consolidated_Eco_KPI_V3.xlsx*: sources in appendix

Tools

Python/ Jupyter notebook: data merging, sentiment and correlation analysis, charts

Data Import & Validation

- Import libraries & define functions (see Appendix)
- Import csv & Excel files
- Data Validation
  - no null values or full duplicates identified in speeches
  - 29 entries identified with duplicated speeches. Original authors cannot be identified. Share is small ⇒ not removed.

```
# Create a normalized version of the 'text' column
speeches_original['text_norm'] = speeches_original['text'].str.strip().str.lower()

# Find duplicate 'text_norm' entries
duplicate_mask = speeches_original['text_norm'].duplicated(keep=False)

# Extract all duplicates based on normalized text
duplicates = speeches_original[duplicate_mask]
```

Data Cleaning & Transformation

- Edward George not stated as governor ⇒ speeches from '1 July 1993 to 30 June 2003' by 'george' changed to '1' in column 'is_gov' column.

```
# Speeches given by Edward George are wrongly not stated as is_gov
def correct_is_gov_column(speeches_df: pd.DataFrame):
    # Make sure date is datetime first
    speeches_original['date'] = pd.to_datetime(speeches_original['date'], errors='coerce')

    # Apply correction
    condition = (
        (speeches_original['author'].str.lower() == 'george') &
        (speeches_original['date'].dt.year > 1993) &
        (speeches_original['date'].dt.year < 2004)
    )
    speeches_original.loc[condition, 'is_gov'] = 1  # 1 means Governor

    return speeches_original

# Correct the is_gov column
speeches_original = correct_is_gov_column(speeches_original)

# View the DataFrame
display(speeches_original[speeches_original['author'].str.lower() == 'george'].head())
```

- Speech "r000101a_SRB" wrongly entered as year 1900 ⇒ removed

```
# Filter for speeches in year 1900
speeches_1900 = speeches_original[speeches_original['year'] == 1900]

# View the DataFrame
speeches_1900
```

	reference	country	date	title	author	is_gov	text	date_format	year_month	year	year_month_dt
4010	r000101a_SRB	sweden	1900-01-01	Swedish economy in transition	backstrom	1	Thank you for inviting me to talk with you thi...	1900-01-01	1900-01	1900	1900-01-01

```
# Remove the specific row with index 4010, as the date must be wrongly entered, as dates on start in 1990
# Backstrom was governor of the Bank of Sweden from 1994 to 2002.
speeches_original = speeches_original.drop(4010)
```

- Speech "r181025a_BOC" no speech content ⇒ removed

```
# Find the row that is missing the sentiment score
row_missing_score = speeches[speeches['sentiment_lexicon_weighted'].isnull()]

# View the row
row_missing_score
```

	reference	country	date	title	author	is_gov	text	text_norm
795	r181025a_BOC	canada	2018-10-25	Money for Nothing? A Central Banker's Take on ...	wilkins	0	I - a FI - b FI - n bankofcanada.ca bankofcana...	i - a fi - b fi - n bankofcanada.ca bankofcana...

1 rows × 28 columns

Observation: no clear speech text => line can be dropped from dataset

```
speeches = speeches.drop(index=795)
```

- Date transformation for chart design and data merging: new columns for year and year_month

```
# Change date format from 'object' to 'datetime64' and display in a new column
speeches_original['date_format'] = speeches_original['date'].astype('datetime64[ns]')
```

```
# Add a new column for year and month
speeches_original['year_month'] = pd.to_datetime(speeches_original['date_format']).dt.to_period('M')
```

```
# Add a new column for year only
speeches_original['year'] = pd.to_datetime(speeches_original.date).dt.year
```

```
# Add a column for year_month in date format
speeches_original['year_month_dt'] = speeches_original['year_month'].dt.to_timestamp()
```

```
# Add a new column for year and month.
uk_economic_indicators['year_month'] = pd.to_datetime(uk_economic_indicators['year_month']).dt.to_period('M')
```

Analysis based on 7719 unique speeches across the world, and 1209 speeches from the UK.

	country	number_speeches
0	euro area	2351
1	united states	1551
2	united kingdom	1209
3	japan	755
4	canada	619
5	sweden	599
6	switzerland	351
7	australia	284
8	Total	7719

Restrictions

- BoE speech data only from 1997 to 2022
- Limited speech data from some countries allowing only for limited comparison

SENTIMENT SCORE ANALYSIS

To select the most approriate sentiment model, various sentiment scores were calculated and compared.

- **BoE Wordlist simple sentiment score** (weighted by sentiment word count)
  Process to apply lexicon to text followed for all lexicon models

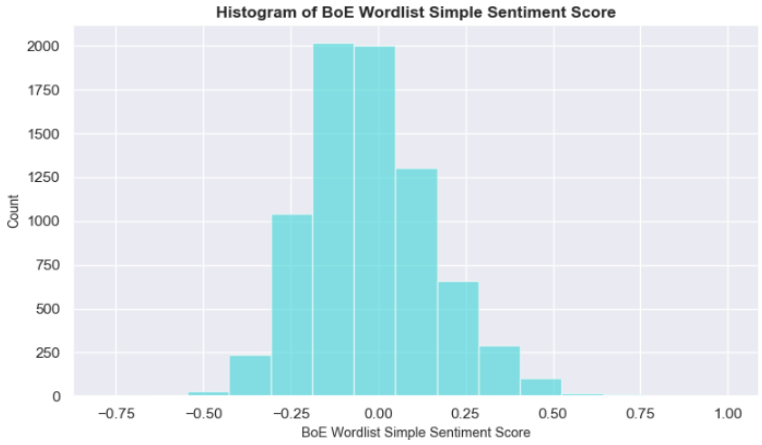
```
# Define function to apply the lexicon to the text
def lexicon_counts(tokens):
    return pd.Series({
        cat: sum(t in word_sets[cat] for t in tokens)
        for cat in categories
    })
```

```
# Prepare the lexicon
sentiment_lexicon = sentiment_lexicon.copy()

# Define categories
categories = [
     'Negative',
     'Positive',
     'Uncertainty',
     'Litigious',
     'Strong',
     'Weak',
     'Constraining',
 ]

# Create dictionary of categories, containing words that belong to that category based on your sentiment lexicon.
word_sets = {
    cat: set(sentiment_lexicon.loc[sentiment_lexicon[cat] == 1, 'Word'].str.lower())
    for cat in categories
}
```

```
# Apply the lexicon_counts function to 'text_lemmatised' column
category_counts = boe_speeches['text_lemmatised'].apply(lexicon_counts)

# Concatenate category counts to the original DataFrame
boe_speeches = pd.concat([boe_speeches, category_counts], axis=1)

# Calculate total sentiment words across all categories for each speech
boe_speeches['word_count_sentiment'] = category_counts.sum(axis=1)

# View the DataFrame
boe_speeches.head()
```

Calculation of sentiment score using positive & negative categories only and sentiment word count.

```
# Calculate the sentiment score by subtracting the negative score from the positive score
# abd dividing by the total number of words in the lemmatised text
speeches['sentiment_lexicon_simple'] = (speeches['positive'] - speeches['negative'])/ speeches['word_count_sentiment']
```



Slight negative skew due to negative skew in wordlist (57% of words negative, 9% positive). Not using all categories does not take into account the nuance the dictionary has to offer.

- **BoE Wordlist weighted sentiment score**

  Calculation using all categories by applying weightings and sentiment word count.

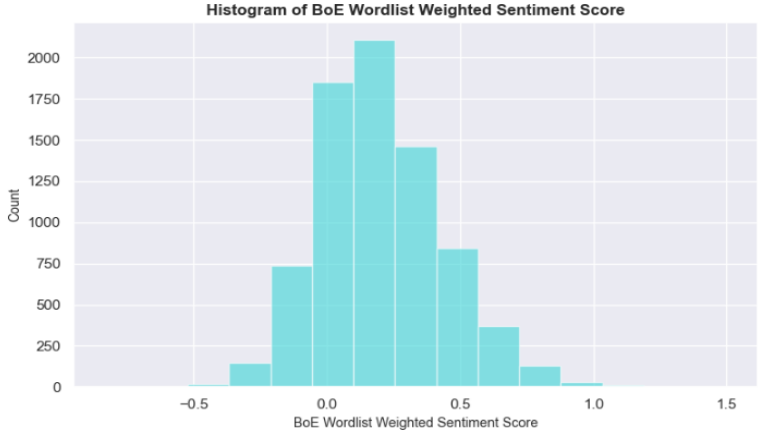
```
# Define function to apply the lexicon to the text
def lexicon_score_weighted(tokens):
    score = 0
    for cat in categories:
        count = sum(t in word_sets[cat] for t in tokens)
        score += count * category_weights[cat]
    return score
```

Category weights based on importance:

- Negative -1 Standard negative words
- Positive +1.5 Standard positive words - compensate for negatively skewed wordlist
- Uncertainty 0.2 Words expressing doubt or ambiguity, less impactful than outright negative or positive words
- Litigious -0.2 Words related to lawsuits or legal issues, potentially negative or impactful depending on context
- Strong +1.5 Words with high intensity or impact, thus given more weight
- Weak +0.5 Words with less impact, so given lesser weight than 'Strong' words
- Constraining -0.5 Words implying restriction or limitations, generally negative

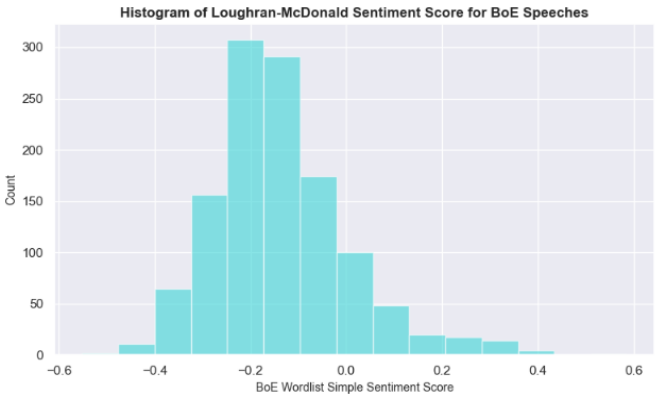
```
# Assign weights to the categories
category_weights = {
    'negative': -1,
    'positive': 1.5,
    'uncertainty': 0.2,
    'litigious': -0.2,
    'strong': 1.5,
    'weak': 0.5,
    'constraining': -0.5
}
```

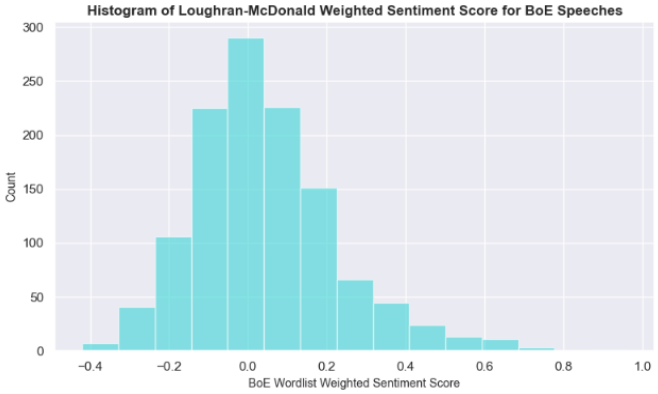
```
# Compute counts and store as a new column
speeches['sentiment_lexicon_weighted'] = speeches['text_lemmatised'].apply(lexicon_score_weighted) \
                                         / speeches['word_count_sentiment']
```



Applying greater weight to positive words, removes negative skew.

- **Loughran-McDonald Master Dictionary**
  Simple and weighted scores calculated as for BoE wordlist



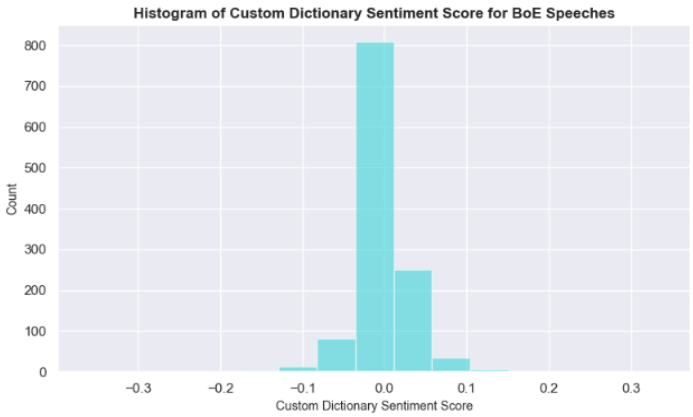


  BoE wordlist based on Loughran-McDonald dictionary ⇒ no added value.

- **Custom Dictionary** (35 words)

```
# Prepare the lexicon
custom = custom_dict.copy()

categories = [
     'negative_custom',
     'positive_custom',
     'uncertainty_custom'
 ]

word_sets = {
    cat: set(custom.loc[custom[cat] == 1, 'Word'].str.lower())
    for cat in categories
}
```

```
# Apply the lexicon_counts function to 'text_lemmatised' column
category_counts_custom = boe_speeches['text_lemmatised'].apply(lexicon_counts)

# Concatenate category counts to the original DataFrame
boe_speeches = pd.concat([boe_speeches, category_counts_custom], axis=1)

# Calculate total sentiment words across all categories for each speech
boe_speeches['word_count_custom'] = category_counts.sum(axis=1)
```



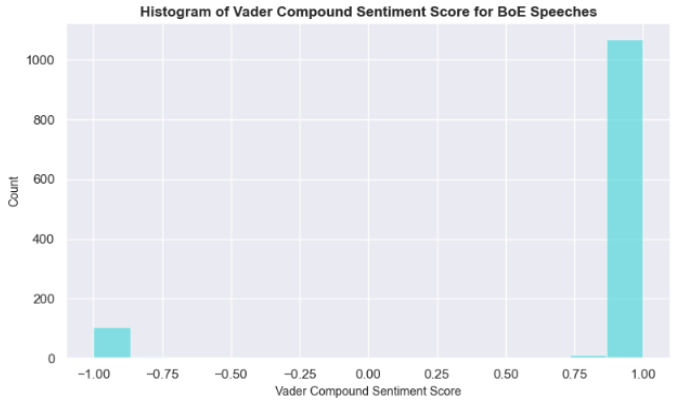
  Custom dictionary not complete enough for nuanced results.

- **VADER**

```
# VADER Sentiment Intensity Analyzer.
analyzer = SentimentIntensityAnalyzer()

# Define the function to compute and return sentiment scores.
def analyse_sentiment(text):
    return analyzer.polarity_scores(' '.join(text))
```

```
# Apply sentiment analysis to the columns using the lemmatised data converted into strings.
boe_speeches['sentiment_score_vader'] = boe_speeches['text_lemmatised'].apply(analyse_sentiment)
```

```
# Extract individual sentiment scores for speeches.
boe_speeches['text_neg'] = boe_speeches['sentiment_score_vader'].apply(lambda x: x['neg'])
boe_speeches['text_neu'] = boe_speeches['sentiment_score_vader'].apply(lambda x: x['neu'])
boe_speeches['text_pos'] = boe_speeches['sentiment_score_vader'].apply(lambda x: x['pos'])
boe_speeches['text_compound'] = boe_speeches['sentiment_score_vader'].apply(lambda x: x['compound'])
```



  Vader compound score gives polarised view ⇒ less nuanced sentiment analysis.

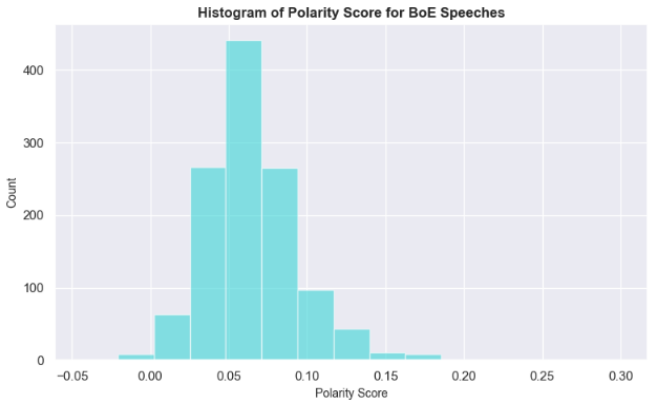
- **Polarity & Subjectivity**

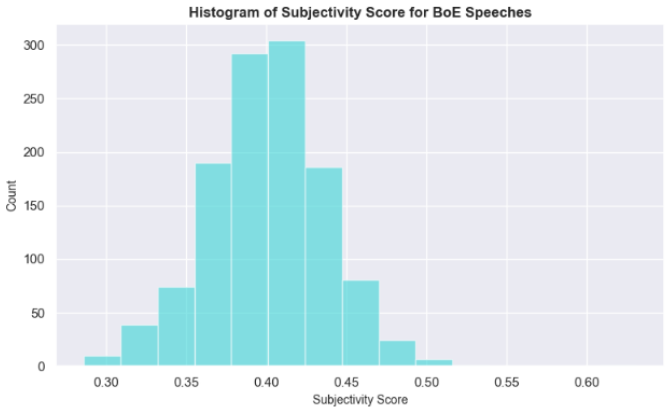
```python
# Define a function to extract a polarity score using TextBlob.
def generate_polarity(comment):
    return TextBlob(comment).sentiment[0]
```

```python
# Define a function to extract a subjectivity score using TextBlob.
def generate_subjectivity(comment):
    return TextBlob(comment).sentiment[1]
```

```python
# Populate a new column with polarity scores for each comment.
boe_speeches['polarity_score'] = boe_speeches['text_lemmatised_str'].apply(generate_polarity)
```

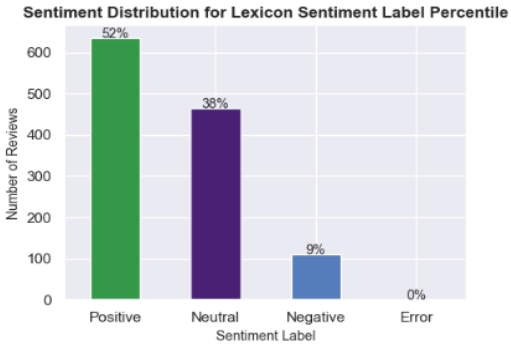
```python
# Populate a new column with polarity scores for each comment.
boe_speeches['subjectivity_score'] = boe_speeches['text_lemmatised_str'].apply(generate_subjectivity)
```





Can be used as compliment to sentiment analysis.

- **OpenAI GPT**



Categorical results only.

- **Finbert ProsusAI and Yiyang models**

```python
# FinBERT Models: Load model 1: ProsusAI/finbert
tokenizer_prosus = AutoTokenizer.from_pretrained('ProsusAI/finbert')
model_prosus = AutoModelForSequenceClassification.from_pretrained('ProsusAI/finbert').to(device)
```

```python
# FinBERT Models: Load model 2: yiyanghkust/finbert-tone
tokenizer_yiyang = AutoTokenizer.from_pretrained('yiyanghkust/finbert-tone')
model_yiyang = AutoModelForSequenceClassification.from_pretrained('yiyanghkust/finbert-tone').to(device)
```

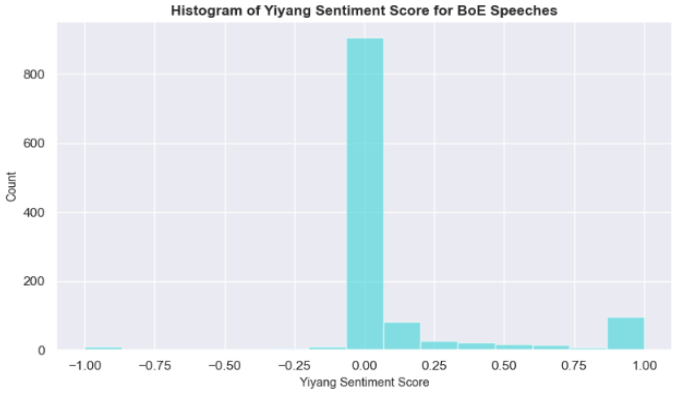
```python
# Define a function to predict probabilities in batches
def predict_batch(texts, tokenizer, model, max_length=128):
    inputs = tokenizer(texts, padding=True, truncation=True, max_length=max_length, return_tensors='pt')
    inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    probs = F.softmax(outputs.logits, dim=1)
    return probs.cpu().numpy()
```

```python
# Define function to calculate one sentiment score
def compute_tone_score(probs):
    class_labels = ['Neutral', 'Positive', 'Negative']
    prob_dict = dict(zip(class_labels, probs))
    return (
        prob_dict['Positive'] * weights['Positive'] +
        prob_dict['Neutral'] * weights['Neutral'] +
        prob_dict['Negative'] * weights['Negative']
    )
```

```
# Check the order of labels in the model
model_yiyang.config.id2label
```

```
{0: 'Neutral', 1: 'Positive', 2: 'Negative'}
```

```
# Column that the model should be applied to
texts = boe_speeches['text_lemmatised'].astype(str).tolist()

# Specify batch size for efficiency
batch_size = 32
all_probs = []

for i in range(0, len(texts), batch_size):
    batch_texts = texts[i:i + batch_size]
    batch_probs = predict_batch(batch_texts, tokenizer_yiyang, model_yiyang)
    all_probs.extend(batch_probs)

# Store the predicted probabilities back into your DataFrame
boe_speeches['yiyang_probs'] = all_probs

# Extract top labels using the order established above
labels = ['Neutral', 'Positive', 'Negative']

def get_probs_yiyang_dict(probs):
    # Assumes probs is an array/list like [neutral_score, positive_score, negative_score]
    return {
        'yiyang_neutral': probs[0],
        'yiyang_positive': probs[1],
        'yiyang_negative': probs[2]
    }

def get_top_label(probs):
    idx = probs.argmax()
    return labels[idx], probs.max()
```

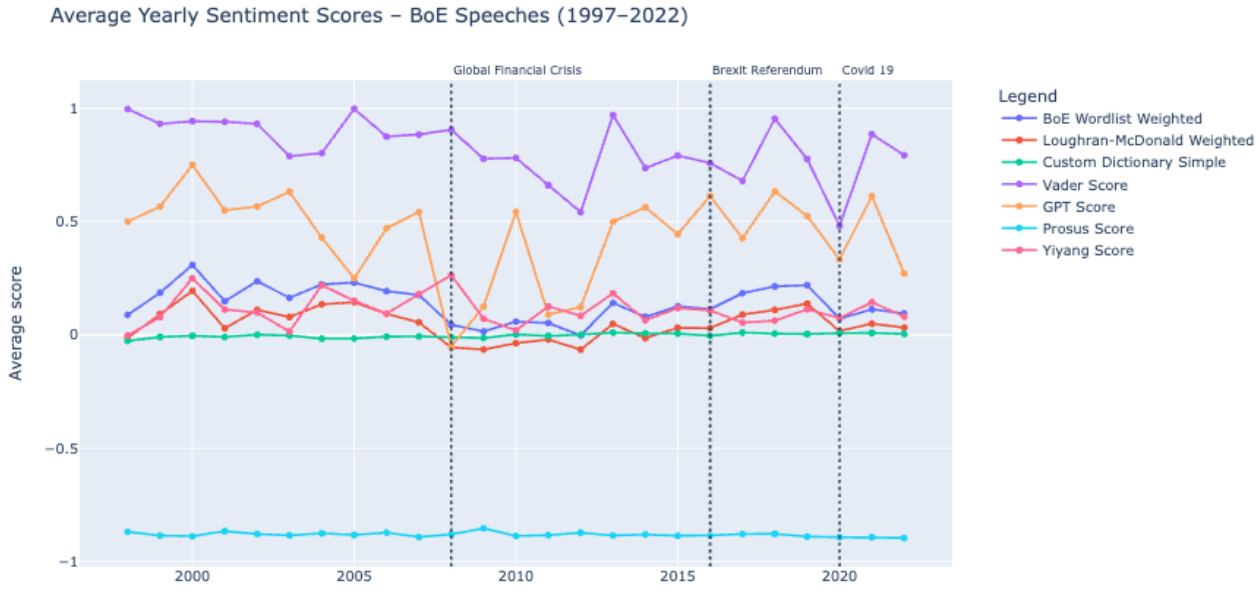



```
# Create a DataFrame with all class probabilities
probs_yiyang = boe_speeches['yiyang_probs'].apply(lambda x: get_probs_yiyang_dict(x)).apply(pd.Series)

# Assign back to your main DataFrame
boe_speeches = pd.concat([boe_speeches, probs_yiyang], axis=1)

# Apply and extract label + confidence
boe_speeches[['yiyang_label', 'yiyang_confidence']] = boe_speeches['yiyang_probs'].apply(lambda x: get_top_label(x)).apply(pd.Series)
```

Finbert models require significant finetuning.
Basic results very neutral and less effective for nuanced sentiment analysis.

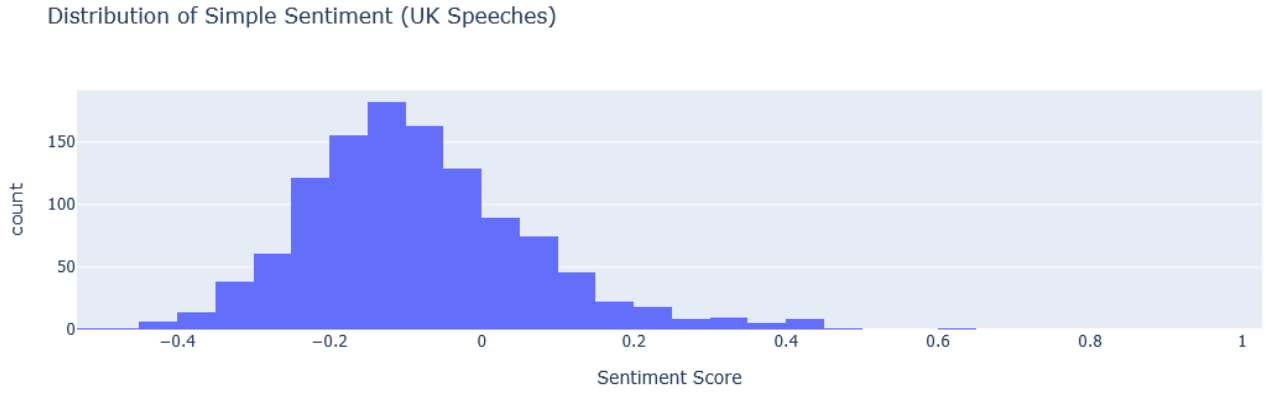
- **Comparison**

Model	Benefit	Drawback
BoE Wordlist	domain specific	negative word skew
Open AI Analysis	flexible	categorical, requires numerical conversion
FinBert Prosus & YiYang	domain specific	very neutral, less nuanced
VADER	easy & fast	limited in capturing complex sentiments
Polarity & Subjectivity	easy & fast	sensitive to context
Loughran-McDonald	domain specific	no additional benefit to BoE wordlist
Custom Dictionary	domain specific	not representative yet



- **Conclusion**
  Sentiment analysis based on dictionaries/ wordlists gives most nuanced analysis of the applied models. BoE Wordlist is tailored to central bank speeches and provides solid base for further analysis.

  Simple sentiment score without applying custom weights



  - Distribution centered slightly below zero, with negative skew
  - Negative and positive sentiment words treated equally ⇒ tone mildly negative/ cautious

  Weighted sentiment score

  Category weights based on importance:
  - Negative: -1.0 Standard negative words
  - Positive: +1.5 Standard positive words – compensate for negatively skewed wordlist
  - Uncertainty: +0.2 Words expressing doubt or ambiguity, less impactful than outright negative or positive words
  - Litigious: -0.2 Words related to legal issues, potentially negative or impactful depending on context
  - Strong: +1.5 Words with high intensity or impact, thus given more weight
  - Weak: +0.5 Words with less impact, so given lesser weight than 'Strong' words
  - Constraining: -0.5 Words implying restriction or limitations, generally negative

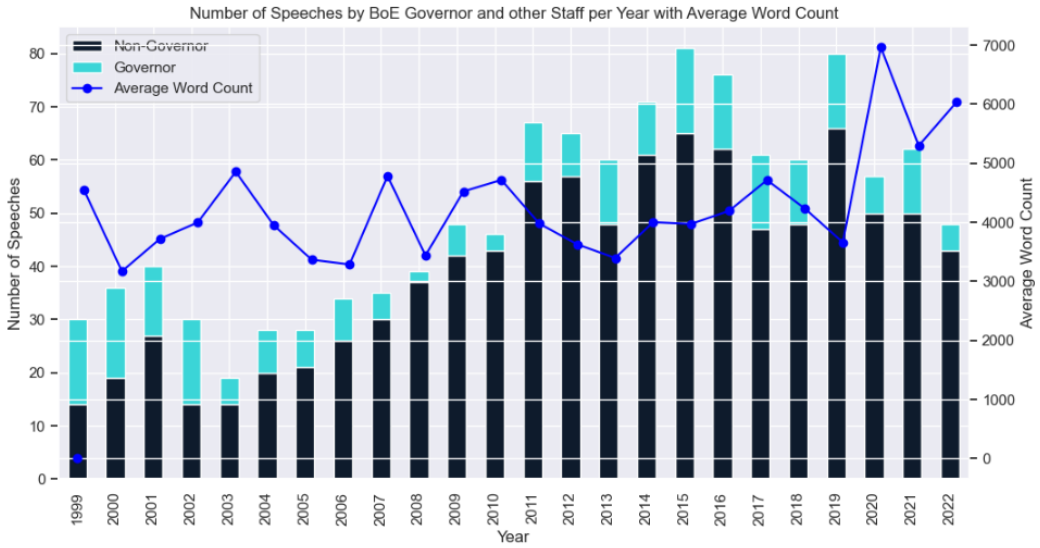
  - Distribution clustered between 0 and 0.2 ⇒ mildly positive tone
  - Aligns better with BoE's role in promoting economic stability and confidence, especially during uncertainty

For the remainder of the analysis weighted sentiment scores taking into account all categories from provided BoE wordlist is used.

SPEECH & SENTIMENT ANALYSIS – BoE Speeches only

Analysis of BoE speech development and relation with key events

- **Speech Analysis**
  - more frequent, shorter speeches since 2010 to increase confidence in central banks after financial crisis and to reach wider audiences



  - Governor speeches generally more positive than non-governor speeches to bring stability

- **MPC Voting Results** (see Appendix for additional details)
  - Creating cleaned MPC voting data set

```
# Create subset of bank rates voted
mpc_df_og_cleaned.columns = [f"col_{i}" for i in range(len(mpc_df_og_cleaned.columns))]

# Create header and drop unnecessary rows
mpc_bank_rates_voted = mpc_df_og_cleaned.iloc[:,:2].drop(index=range(9)).reset_index(drop=True)

# Name columns
mpc_bank_rates_voted.rename(columns={
    "col_0": "date",
    "col_1": "bank_rate"
}, inplace=True)

# Convert vote date to datetime  with mixed formats and handle bad values
mpc_bank_rates_voted['date'] = pd.to_datetime(
    mpc_bank_rates_voted['date'],
    errors='coerce',          # Handle errors
    dayfirst=True,            # most looks like day month year
    format='mixed'            # check by row
)

# Now extract year and month
mpc_bank_rates_voted['vote_year'] = mpc_bank_rates_voted['date'].dt.year
mpc_bank_rates_voted['vote_month'] = mpc_bank_rates_voted['date'].dt.month

# Calculate the difference in bank rate from the previous row
mpc_bank_rates_voted['rate_diff'] = mpc_bank_rates_voted['bank_rate'].diff()

# Categorize the difference
mpc_bank_rates_voted['rate_change'] = mpc_bank_rates_voted['rate_diff'].apply(
    lambda x: 'increase' if x > 0 else ('reduce' if x < 0 else 'maintain')
)

display(mpc_bank_rates_voted.head(5))

max_diff = mpc_bank_rates_voted['rate_diff'].max()
min_diff = mpc_bank_rates_voted['rate_diff'].min()

print(f"Max rate change: {max_diff}")
print(f"Min rate change: {min_diff}")
```

	date	bank_rate	vote_year	vote_month	rate_diff	rate_change
0	1997-06-06	0.065	1997	6	NaN	maintain
1	1997-07-10	0.0675	1997	7	0.0025	increase
2	1997-08-07	0.07	1997	8	0.0025	increase
3	1997-09-11	0.07	1997	9	0.0	maintain
4	1997-10-09	0.07	1997	10	0.0	maintain

```
Max rate change: 0.0075
Min rate change: -0.015
```

```
# Check if there were multiple votes per month
# Group by year and month, count how many rate votes occurred
rate_vote_summary = mpc_bank_rates_voted.groupby(['vote_year', 'vote_month']).size().reset_index(name='vote_count')

# Select  multi vote months
multiple_votes = rate_vote_summary[rate_vote_summary['vote_count'] > 1]

print(multiple_votes)
```

```
     vote_year  vote_month  vote_count
51        2001           9           2
259       2020           3           3
```

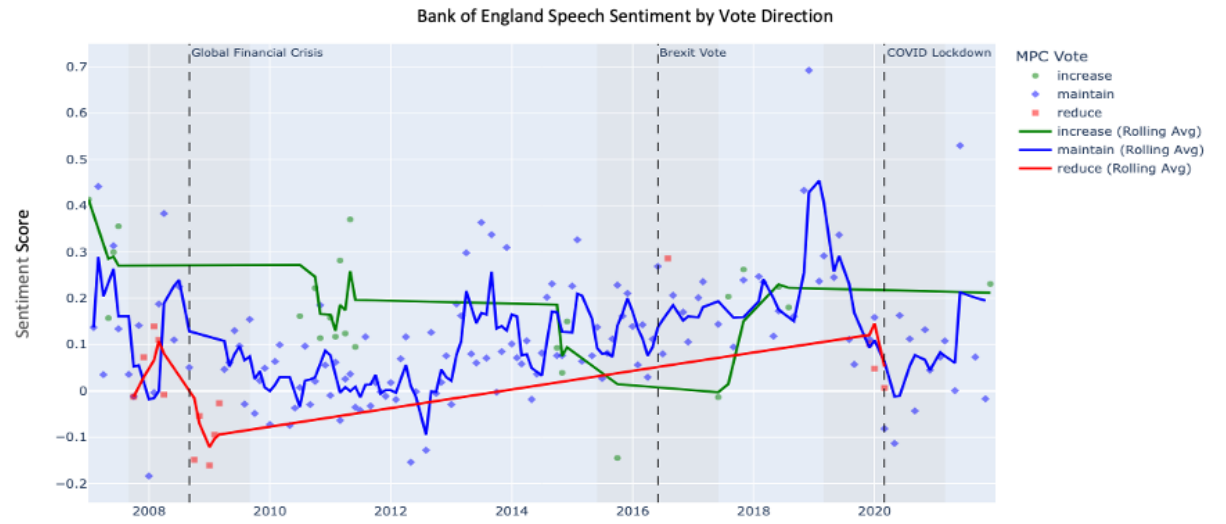
```
# Keep only last vote of the month
mpc_bank_rates_voted_cleaned = mpc_bank_rates_voted.sort_values('date').drop_duplicates(
    subset=['vote_year', 'vote_month'],
    keep='last'
).reset_index(drop=True)

mpc_bank_rates_voted_cleaned.head(5)
```

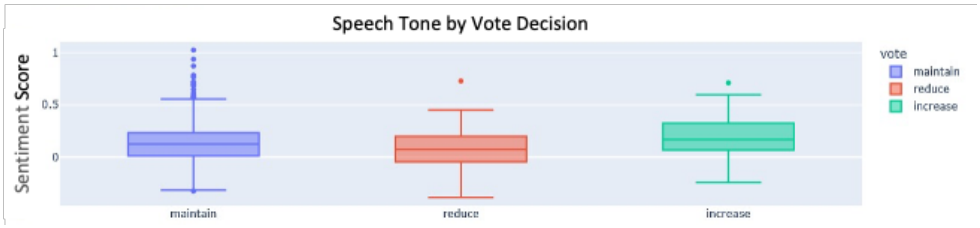
	date	bank_rate	vote_year	vote_month	rate_diff	rate_change
0	1997-06-06	0.065	1997	6	NaN	maintain
1	1997-07-10	0.0675	1997	7	0.0025	increase
2	1997-08-07	0.07	1997	8	0.0025	increase
3	1997-09-11	0.07	1997	9	0.0	maintain
4	1997-10-09	0.07	1997	10	0.0	maintain

- Clean vote data merged with BoE speeches

```
# Join this merged data to speeches on author + date
uk_speeches['year'] = uk_speeches['date'].dt.year
uk_speeches['month'] = uk_speeches['date'].dt.month

# Normalise author names (title-case or lowercase match)
uk_speeches['author_clean'] = uk_speeches['author'].str.lower()
votes_merged['author_clean'] = votes_merged['author'].str.lower()

# Merge votes into speeches
speeches_with_votes = uk_speeches.merge(
    votes_merged,
    on=['author_clean', 'year', 'month'],
    how='left'
)
```

```
# Rolling Averages
# Settings
window_months = 3
rolling_window = 3

# Custom vote colors
vote_color_map = {
    'increase': 'green',
    'maintain': 'blue',
    'reduce': 'red'
}

# Monthly aggregation
monthly_sentiment = (
    mpc_df
    .groupby(['month', 'vote'])['sentiment_lexicon_weighted']
    .mean()
    .reset_index()
    .sort_values(by=['vote', 'month'])
)

# Compute rolling average per vote direction
monthly_sentiment['rolling_sentiment'] = (
    monthly_sentiment
    .groupby('vote')['sentiment_lexicon_weighted']
    .transform(lambda x: x.rolling(window=rolling_window, min_periods=1).mean())
)
```





- Results
  - Decision “increase”: sentiment more positive. Increase used as countermeasure to control inflation ⇒ reflected in positive sentiment.
  - Decision “maintain”: sentiment more moderate ⇒ depends more on other economic factors.
  - Decision “reduce”: often taken in times of crisis ⇒ more negative sentiment.

  BoE communication tone aligns with key events to maintain financial stability.

- **Sentiment Analysis**
  - Timeframe 1999 to 2022, based on introduction of Euro to make data comparable across countries.
    BoE speeches = 1201

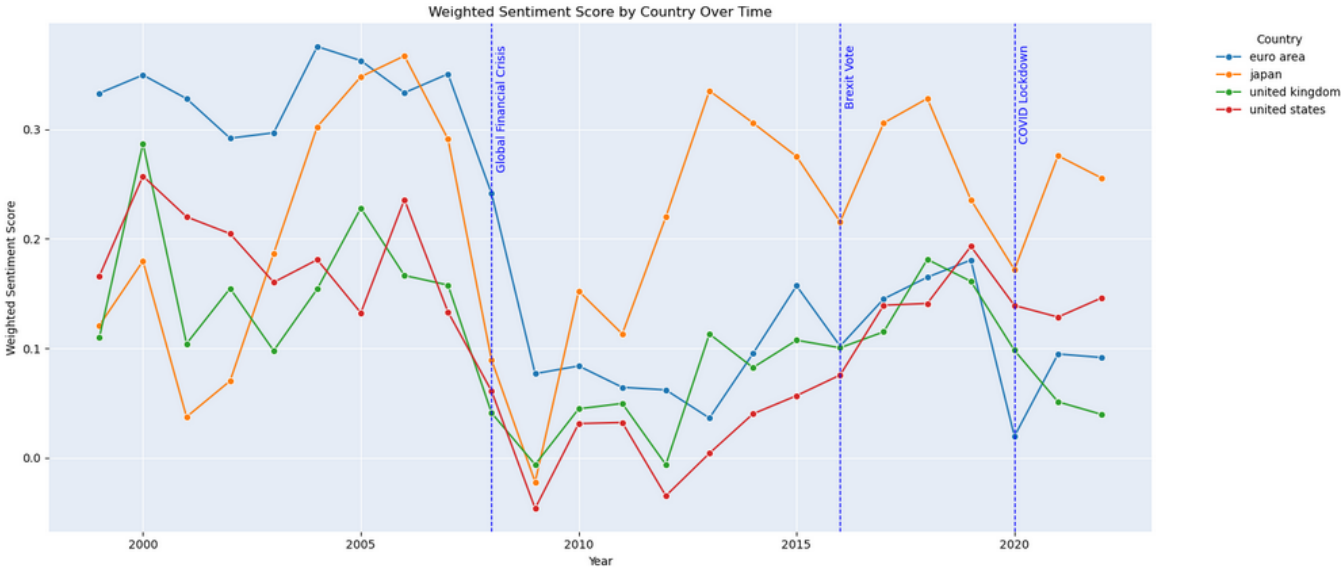
```
# Create separate DataFrame to filter for 2000 to 2022
speeches_1999_2022 = speeches_short[(speeches_short['year'] >= 1999) & (speeches_short['year'] <= 2022)].reset_index()
```

  - Country Comparison: since 2010, UK, Euro Area and US sentiment scores move in line; Japan is an outlier

```
# Weighted sentiment score 2 - compound
def compute_weighted_sentiment_grouped(df, groupby_cols, weights, normalize=True):
    """
    Create aggregated sentiment score based on weighted average of sentiment-related word categories, normalized by the number of sentiment-bearing words

    Parameters:
    - df: DataFrame with sentiment data
    - groupby_cols: Columns to group by sentiment scores (country, author, year, quarter, month)
    - weights: Weights per sentiment category
    """
    # Aggregate all category counts
    agg_dict = {cat: 'sum' for cat in weights if cat in df.columns}
    agg_dict['word_count_sentiment'] = 'sum'

    grouped = df.groupby(groupby_cols).agg(agg_dict).reset_index()

    # Compute weighted score
    weighted_score_2 = sum(
        grouped[cat] * weight for cat, weight in weights.items() if cat in grouped.columns
    )

    if normalize:
        grouped['sentiment_weighted_2'] = weighted_score_2 / grouped['word_count_sentiment']
    else:
        grouped['sentiment_weighted_2'] = weighted_score_2

    return grouped
```

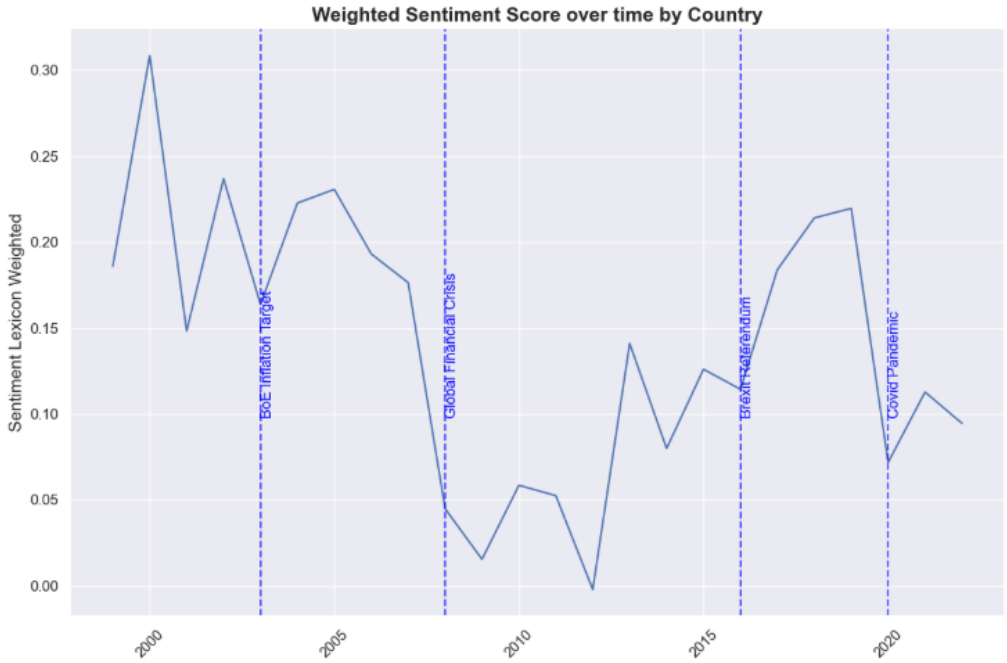
```
# New weighted sentiment per country per year
country_year_sentiment = compute_weighted_sentiment_grouped(
    speeches_sentiment_filtered,
    groupby_cols=['country', 'year'],
    weights=category_weights
)
```

```
# Filter to only include selected countries
selected_countries = ['united kingdom', 'united states', 'japan', 'euro area']
filtered_df = country_year_sentiment[country_year_sentiment['country'].isin(selected_countries)]
```



- Alignment with Key Events

```
generate_multi_lineplot(speeches_1999_2022_yearly[speeches_1999_2022_yearly['country'].isin(country_to_display)], \
                        x_axis='year', y_axis='sentiment_lexicon_weighted', hue='country', \
                        title= 'Weighted Sentiment Score over time by Country', date=None, ylim=None, \
                            save_path=None, errorbar=None, events=None, event_years=events)
```



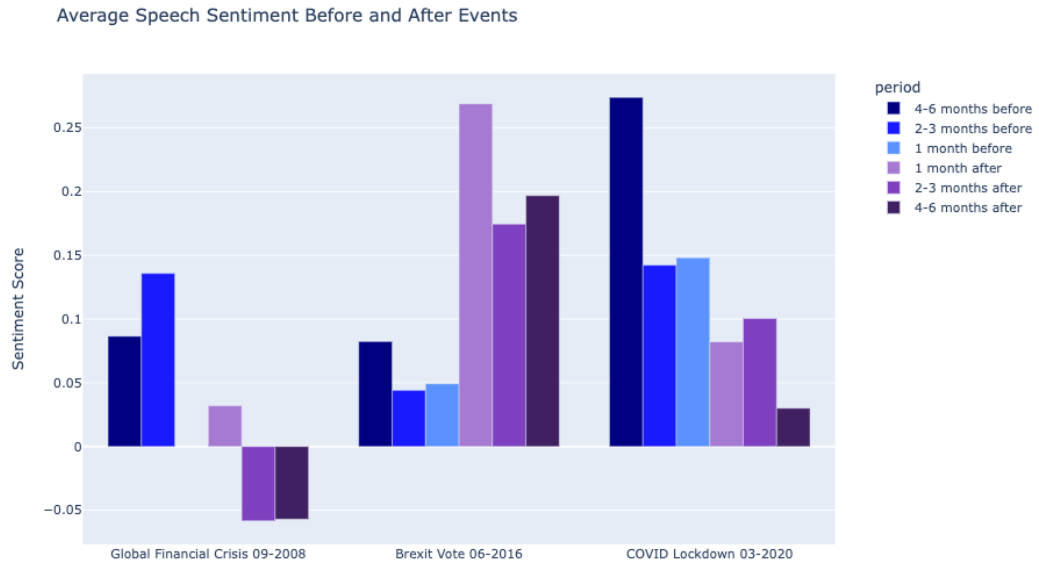
<u>Analysis 6 months before & after an event</u>

Sentiment scores fall after surprise rather than anticipated events.

Global fnancial crisis and Covid lockdown ⇒ sentiment scores fall, as severity was unexpected.

Brexit referendum ⇒ speech sentiment scores afterwards more positive ⇒ anticipated event. Policies put in place to counteract market reactions ⇒ speech sentiment reflects more positive view.

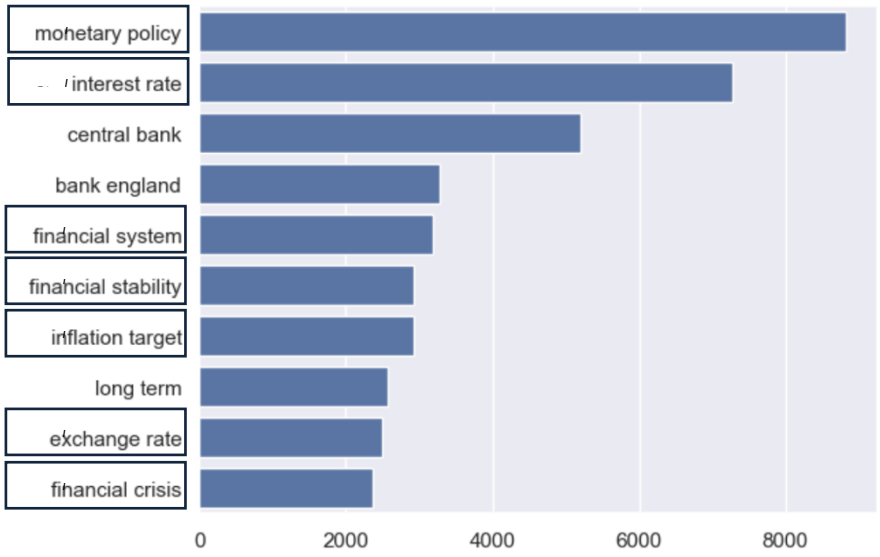
```python
# Initialize list for sentiment comparison results
sentiment_comparison = []

for _, row in event_df.iterrows():
    event_name = row['event']
    event_date = row['month']

    # Define time windows around the event
    windows = {
        '4-6 months before': (event_date - pd.DateOffset(months=6), event_date - pd.DateOffset(months=3)),
        '2-3 months before': (event_date - pd.DateOffset(months=3), event_date - pd.DateOffset(months=1)),
        '1 month before': (event_date - pd.DateOffset(months=1), event_date - pd.DateOffset(days=1)),
        '1 month after': (event_date, event_date + pd.DateOffset(months=1) - pd.DateOffset(days=1)),
        '2-3 months after': (event_date, event_date + pd.DateOffset(months=3) - pd.DateOffset(months=1)),
        '4-6 months after': (event_date, event_date + pd.DateOffset(months=6) - pd.DateOffset(months=3))
    }

    for period_name, (start_date, end_date) in windows.items():
        speeches = boe_indicators_1999_2022[
            (boe_indicators_1999_2022['date_format'] >= start_date) &
            (boe_indicators_1999_2022['date_format'] <= end_date)
        ]

        # Save results
        sentiment_comparison.append({
            'event': event_name,
            'period': period_name,
            'avg_sentiment': speeches['sentiment_lexicon_weighted'].mean(),
            'count': speeches.shape[0]
        })

# Convert to DataFrame
sentiment_df = pd.DataFrame(sentiment_comparison)

# Define the order of periods
order = ['4-6 months before', '2-3 months before', '1 month before', '1 month after', '2-3 months after', '4-6 months after']
sentiment_df['period'] = pd.Categorical(sentiment_df['period'], categories=order, ordered=True)
sentiment_df = sentiment_df.sort_values(['period'])
```



- Top Word Counts per speech

```python
# Define bar chart for top word groups
def top_word_group_barchart(text, n=2):
    stop=set(stopwords.words('english'))

    new= text.str.split()
    new=new.values.tolist()
    corpus=[word for i in new for word in i]

    def _get_top_ngram(corpus, n=None):
        vec = CountVectorizer(ngram_range=(n, n)).fit(corpus)
        bag_of_words = vec.transform(corpus)
        sum_words = bag_of_words.sum(axis=0)
        words_freq = [(word, sum_words[0, idx])
                      for word, idx in vec.vocabulary_.items()]
        words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
        return words_freq[:10]

    top_n_bigrams=_get_top_ngram(text,n)[:10]
    x,y=map(list,zip(*top_n_bigrams))
    sns.barplot(x=y,y=x)
```

```
# Plot top phrases with 2 words
top_word_group_barchart(boe_speeches['text_lemmatised_str'],2)
```



Calculating number of bigrams per speech

```
# Bigrams to search for
target_bigrams = ['monetary policy', 'interest rate', 'financial stability', 'inflation target', 'financial crisis', 'exchange rate']

# Fit CountVectorizer on all speeches
vectorizer = CountVectorizer(ngram_range=(2, 2))
X = vectorizer.fit_transform(boe_indicators_1999_2022['text_lemmatised_str'].str.lower())

# Create a DataFrame of counts
bigram_counts_df = pd.DataFrame(X.toarray(), index=boe_indicators_1999_2022.index, columns=vectorizer.get_feature_names_out())

# For each target bigram, add a column with its count
for bigram in target_bigrams:
    if bigram in bigram_counts_df.columns:
        boe_indicators_1999_2022[f'{bigram}'] = bigram_counts_df[bigram]
    else:
        # Bigram not found in the data
        boe_indicators_1999_2022[f'{bigram}'] = 0
```

Bigrams in speech more than 10 times

```
# Filter speeches with 'monetary policy' > 10
boe_monetary_policy_10 = boe_indicators_1999_2022[boe_indicators_1999_2022['monetary policy'] > 10]
```

```
# Filter speeches with 'interest rate' > 10
boe_interest_rate_10 = boe_indicators_1999_2022[boe_indicators_1999_2022['interest rate'] > 10]
```

```
# Filter speeches with 'interest rate' > 10
boe_financial_stability_10 = boe_indicators_1999_2022[boe_indicators_1999_2022['financial stability'] >10]
```

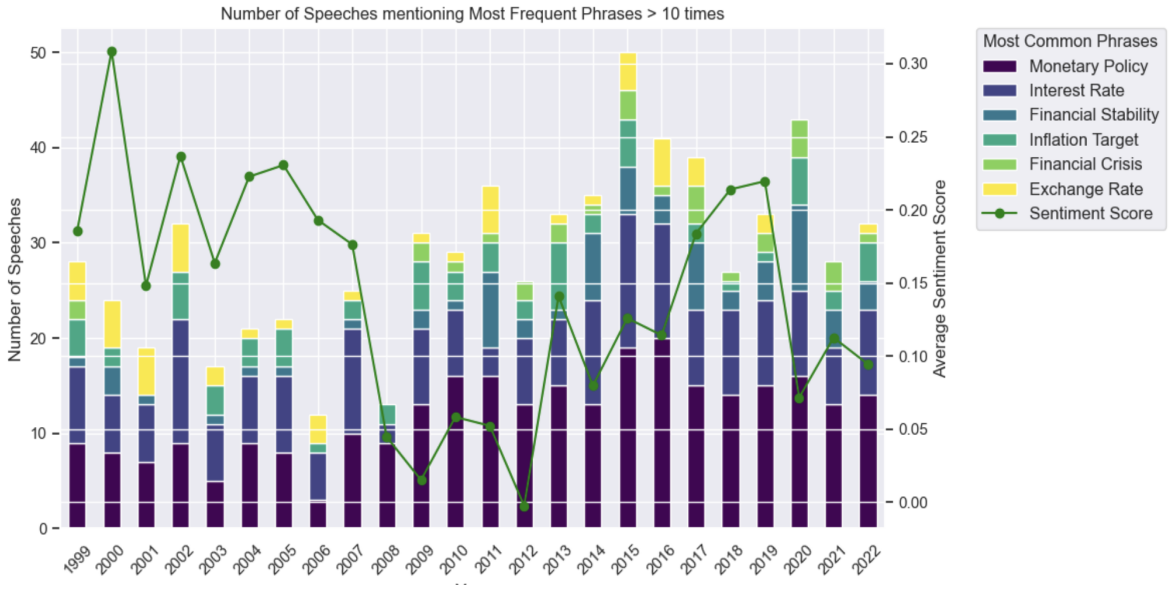
```
# Filter speeches with 'inflation target' > 10
boe_inflation_target_10 = boe_indicators_1999_2022[boe_indicators_1999_2022['inflation target'] > 10]
```

```
# Filter speeches with 'inflation target' > 10
boe_financial_crisis_10 = boe_indicators_1999_2022[boe_indicators_1999_2022['financial crisis'] > 10]
```

```
# Filter speeches with 'inflation target' > 10
boe_exchange_rate_10 = boe_indicators_1999_2022[boe_indicators_1999_2022['exchange rate'] > 10]
```

Number of speeches per year in which a word group (bigram) was used more than 10 times.

- monetary policy and interest rates greatest share and increasing
- financial crisis only used after crisis of 2008
- number of speeches using most common phrases > 10 times falls in years with more positive sentiment scores ⇒ other topics are discussed.

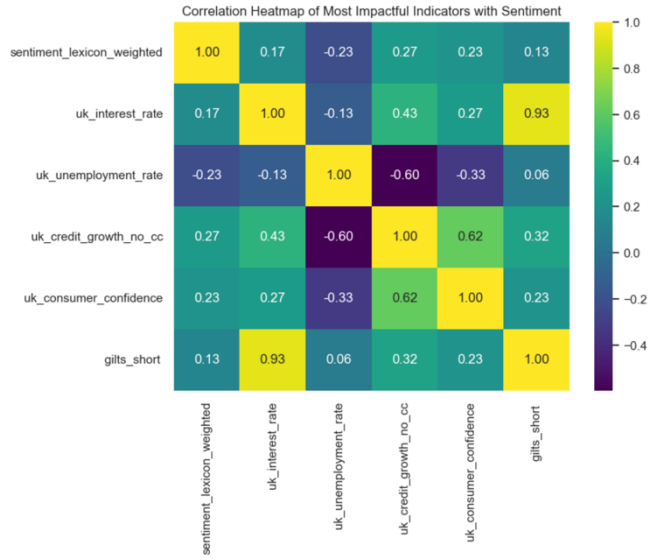


CORRELATION ANALYSIS – BoE Speeches only

- **Correlation Matrix**

  Check for linear correlation between economic indicators and sentiment score – show indicators with best fit:

```
# Create a pairplot for sentiment score and most impactful indicators
columns_sentiment_top = ['sentiment_lexicon_weighted', 'uk_interest_rate', 'uk_unemployment_rate',
                         'uk_credit_growth_no_cc', 'uk_consumer_confidence', 'gilts_short ']

# Correlation matrix for sentiment score and most impactful indicators
correlation_matrix_top = boe_indicators_1999_2022[columns_sentiment_top].corr()
```



- **Cross Correlation** of sentiment scores with economic indicators

  Check for stationarity to ensure modeling is valid. Transform the data, if not stationary.

```
# List of variables you want to test
variables = ['sentiment_lexicon_weighted',
             'uk_consumer_confidence',
             'uk_inflation_rate_CPIH',
             'uk_unemployment_rate',
             'uk_gdp_growth',
             'uk_interest_rate',
             'gbp_usd_fx',
             'ftse_250',
             'gilts_short ',
             'gilts_medium ',
             'gilts_long ',
             'uk_credit_growth_no_cc',
             'uk_credit_growth_only_cc',
             'avg_price_all_property_types']

# Loop over each variable
for var in variables:
    series = boe_rf[var]
    result = adfuller(series.dropna())  # drop NaNs if any
    print(f"{var}: p-value = {result[1]:.4f}")
```

Observation: the following variables are not stationary and need to be transformed (p>0.05):

- uk_consumer_confidence: p-value = 0.6762
- uk_inflation_rate_CPIH: p-value = 0.9101
- uk_unemployment_rate: p-value = 0.6788
- uk_interest_rate: p-value = 0.3117
- gbp_usd_fx: p-value = 0.4019
- ftse_250: p-value = 0.5079
- gilts_short : p-value = 0.2794
- gilts_medium : p-value = 0.6901
- gilts_long : p-value = 0.7010
- uk_credit_growth_no_cc: p-value = 0.4252
- uk_credit_growth_only_cc: p-value = 0.0916
- avg_price_all_property_types: p-value = 0.8011

```python
for i, var in enumerate(indicators_diff):
    cc = []
    p_values = []

    for lag in lags:
        x = boe_rf_diff['sentiment_lexicon_weighted']
        y = boe_rf_diff[var].shift(lag)

        # Drop NaNs that might be introduced after shift
        valid_idx = x.dropna().index.intersection(y.dropna().index)
        n = len(valid_idx)

        if n > 2:
            x_valid = x.loc[valid_idx]
            y_valid = y.loc[valid_idx]
            r = x_valid.corr(y_valid)
            cc.append(r)
            # Convert r to t-stat and p-value
            t_stat = r * np.sqrt((n - 2) / (1 - r**2))
            p = 2 * (1 - stats.t.cdf(abs(t_stat), df=n - 2))
        else:
            cc.append(np.nan)
            p = np.nan
        p_values.append(p)

    # Plotting the correlation over lags
    plt.figure(figsize=(8, 4))
    plt.plot(lags, cc, label='Cross-correlation')
    for j, p in enumerate(p_values):
        if not np.isnan(p) and p < 0.05:
            plt.scatter(lags[j], cc[j], color='red', marker='o')
            plt.annotate('*', (lags[j], cc[j]), textcoords="offset points", xytext=(0,10), ha='center', color='red')

    plt.title(f'Cross-correlation of sentiment with {var}')
    plt.xlabel('Lag')
    plt.ylabel('Correlation coefficient')
    plt.grid(True)
    plt.legend()
    plt.show()

    # Print significant lags with both p-value and correlation coefficient
    print(f"\nSignificant lags for {var}:")
    for lag_idx, p in enumerate(p_values):
        if not np.isnan(p) and p < 0.05:
            corr_coeff = cc[lag_idx]
            print(f"Lag {lags[lag_idx]}: p-value={p:.4f}, correlation={corr_coeff:.3f}")
```

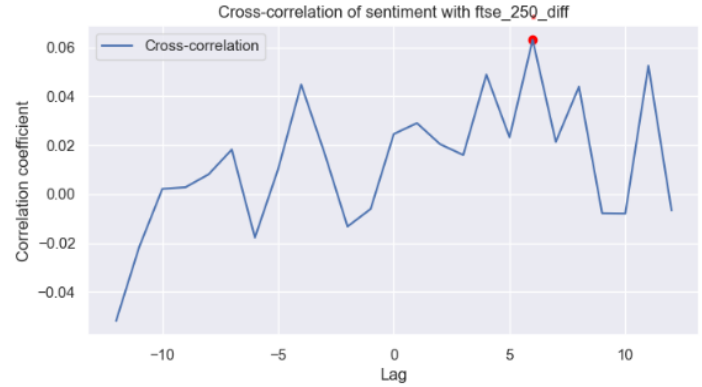
Results: red dots show peaks with statistical significance

- Unemployment Rate


```
Significant lags for uk_unemployment_rate_diff:
Lag -11: p-value=0.0297, correlation=-0.066
Lag -3: p-value=0.0343, correlation=-0.064
Lag 0: p-value=0.0292, correlation=-0.066
Lag 1: p-value=0.0122, correlation=-0.076
Lag 4: p-value=0.0020, correlation=-0.094
Lag 6: p-value=0.0391, correlation=-0.063
Lag 8: p-value=0.0021, correlation=-0.094
```

significant negative correlations ⇒ more positive speeches followed by decline in unemployment rate changes after about 4-8 months.

- FTSE 250




```
Significant lags for ftse_250_diff:
Lag 6: p-value=0.0384, correlation=0.063
```

Central bank speeches more positive ⇒ small but significant rise in FTSE250 changes after 6 months.

- **Random Forests with lagged sentiment scores** (monthly analysis)

```python
# List of targets
targets = [
    'uk_consumer_confidence',
    'uk_inflation_rate_CPIH',
    'uk_unemployment_rate',
    'uk_gdp_growth',
    'uk_interest_rate',
    'gbp_usd_fx',
    'ftse_250',
    'gilts_short ',
    'gilts_medium ',
    'gilts_long ',
    'uk_credit_growth_no_cc',
    'uk_credit_growth_only_cc',
    'avg_price_all_property_types'
]

# Features
feature_cols = [
    'sentiment_lexicon_weighted',
    'sentiment_lexicon_weighted_lag_1m',
    'sentiment_lexicon_weighted_lag_2m',
    'sentiment_lexicon_weighted_lag_3m',
    'month', 'quarter', 'year'
]
```

```python
# Loop through each target
for target in targets:
    print(f"\nTraining model for: {target}")
    y = boe_rf_monthly[target]
    X = boe_rf_monthly[feature_cols]

    # Time-series aware split without shuffling
    split_idx = int(len(boe_rf_monthly) * 0.8)
    X_train, X_test = X.iloc[:split_idx], X.iloc[split_idx:]
    y_train, y_test = y.iloc[:split_idx], y.iloc[split_idx:]

    # Initialize and train the model
    rf = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42)
    rf.fit(X_train, y_train)

    # Make predictions
    y_pred = rf.predict(X_test)

    # Evaluate
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    print(f"RMSE: {rmse:.3f}")
    print(f"R^2: {r2:.3f}")
```

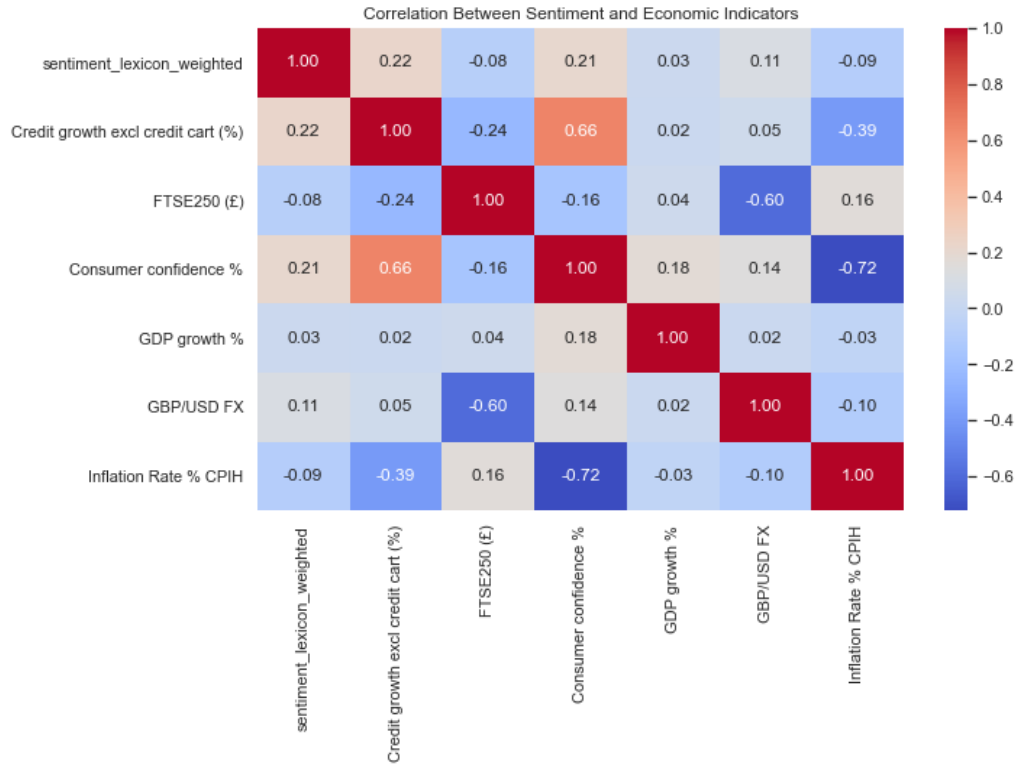
Results

```
Training model for: uk_consumer_confidence
RMSE: 11.694
R^2: -0.561

Training model for: uk_inflation_rate_CPIH
RMSE: 2.561
R^2: 0.011

Training model for: uk_unemployment_rate
RMSE: 0.522
R^2: -0.147

Training model for: uk_gdp_growth
RMSE: 7.186
R^2: -0.006

Training model for: uk_interest_rate
RMSE: 0.395
R^2: -0.307

Training model for: uk_credit_growth_no_cc
RMSE: 8.122
R^2: -2.254

Training model for: uk_credit_growth_only_cc
RMSE: 12.870
R^2: -0.623

Training model for: avg_price_all_property_types
RMSE: 25856.817
R^2: -1.765

Training model for: gbp_usd_fx
RMSE: 0.073
R^2: -0.792

Training model for: ftse_250
RMSE: 1871.267
R^2: -0.068

Training model for: gilts_short
RMSE: 0.568
R^2: -0.118

Training model for: gilts_medium
RMSE: 0.658
R^2: -0.356

Training model for: gilts_long
RMSE: 0.785
R^2: -1.424
```

- no meaningful correlations ⇒ sentiment scores do not predict economic indicators alone.
- Negative R2 suggests model performs worse than predicting mean of target variable for all observations.
- Not improved by using quarterly data.
- Economic indicators do not predict sentiment score either (R2 remains negative).

```
Training model for: sentiment_lexicon_weighted
RMSE: 0.133
R^2: -0.435
```

- **Correlation Analysis**



Correlation Heatmap Summary Consumer confidence & credit growth (0.66): Strong positive link.

Consumer confidence & inflation (-0.72): Confidence drops as inflation rises.

FTSE250 & credit growth (-0.24): Rapid credit growth may signal market concerns.

Sentiment vs. indicators (-0.09 to 0.22): Weak predictive power.

GBP/USD & FTSE250 (-0.60): Strong pound affects exports.

Insight: Consumer confidence bridges financial and psychological indicators; sentiment analysis alone has limited predictive value.

- **Multivariate Regression Analysis**

Application of Random Forest and XGBoost Models

```
Random Forest (Credit growth excl credit card (%)): MAE = 1.9753036110023126, R² = 0.7435242446351552
Random Forest (FTSE250 (£)): MAE = 2357.7750331061484, R² = 0.6445719817911746
Random Forest (Consumer confidence %): MAE = 0.4152026099569061, R² = 0.9900855742387913
Random Forest (GDP growth %): MAE = 0.5343122193436963, R² = 0.6077661037640882
Random Forest (GBP/USD FX): MAE = 0.055399685776907306, R² = 0.7997742306186136
Random Forest (Inflation Rate % CPIH): MAE = 0.0013869016684105516, R² = 0.9627956396838859
```

```
XGBoost (Credit growth excl credit card (%)): MAE = 2.118712333933517, R² = 0.7259484028992089
XGBoost (FTSE250 (£)): MAE = 2402.7291164831063, R² = 0.6738256138241467
XGBoost (Consumer confidence %): MAE = 0.5201183928660205, R² = 0.9876055789677534
XGBoost (GDP growth %): MAE = 0.6294962596631094, R² = 0.44558906283504485
XGBoost (GBP/USD FX): MAE = 0.056574258776940264, R² = 0.8297494638049907
XGBoost (Inflation Rate % CPIH): MAE = 0.001736582689423289, R² = 0.9651820141780643
```

- **Sentiment and Market Predictability with Feature Engineering**

Modeling sentiment with lags, averages, interactions, and trends to capture delayed effects

Key Economic Indicators and Their Importance:

- Sentiment_Lag_3M – Economic sentiment influences growth with a delay, reflecting firms' reaction time.
- Credit_Growth_Lag_3M – Credit expansion drives spending and investment with a lag.
- Sentiment_MA_3M – Averages sentiment over 3 months to reduce noise and highlight trends.
- Consumer_Confidence_MA_3M – Smooths volatile consumer confidence to show persistent optimism/pessimism.
- Inflation_Sentiment_Interaction – Sentiment moderates inflation's impact on purchasing power.
- FX_Sentiment_Interaction – Market sentiment amplifies or mutes exchange rate effects.
- Sentiment_Trend – Tracks changes in sentiment to signal economic shifts.
- GDP_Trend – Captures momentum in GDP growth, beyond static rates.

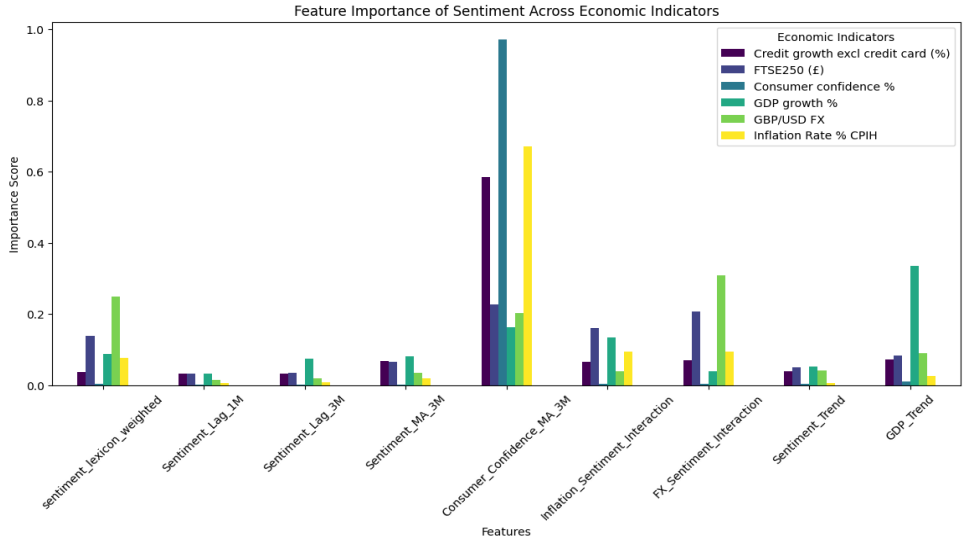
```
merged_df['Sentiment_Lag_3M'] = merged_df['sentiment_score'].shift(3)  # 3-month lag
merged_df['Credit_Growth_Lag_3M'] = merged_df['Credit growth excl credit cart (%)'].shift(3)
```

In [75]:

```
merged_df['Sentiment_MA_3M'] = merged_df['sentiment_score'].rolling(window=3).mean()  # 3-month a
merged_df['Consumer_Confidence_MA_3M'] = merged_df['Consumer confidence %'].rolling(window=3).mea
```

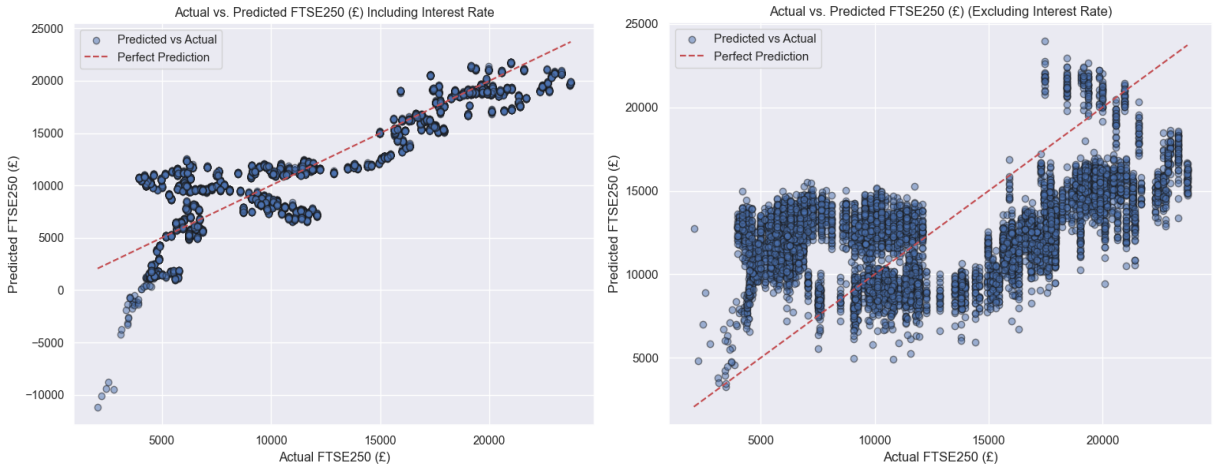
In [77]:

```
merged_df['Inflation_Sentiment_Interaction'] = merged_df['Inflation Rate % CPIH'] * merged_df['se
merged_df['FX_Sentiment_Interaction'] = merged_df['GBP/USD FX'] * merged_df['sentiment_score']
```



Sentiment models show consumer confidence (3 month average) most predictive, with moderate input from inflation. Sentiment impacts FTSE250 and FX, and minimally affects GDP growth.

- **Connecting Sentiment, Interest Rates, and Market Predictability**
  Using regressions with sentiment and economic indicators (e.g. interest rates) to assess impact on FTSE250 (UK market proxy reflecting domestic sentiment).



Excluding interest rates worsened model performance, highlighting vital role in shaping market expectations.

Key findings:

Updated FTSE250 Model: Key Takeaways R^2 = 0.474 Sharp drop from 0.848 : Interest Rate % was a key driver Multicollinearity reduced, but at a cost to model performance

Coefficient Insights Sentiment Score: -3.58 : Now negative, weak influence

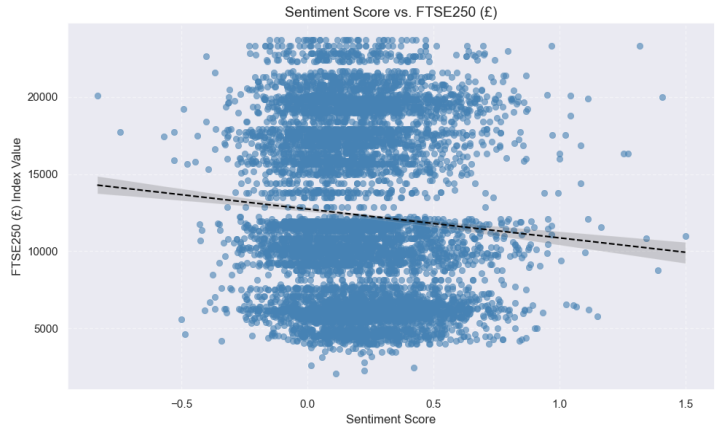
Inflation Rate: -9782.93 : Now negative, opposite of prior model

GDP Growth: +183.72: Still positively linked to FTSE250

Unemployment Rate: -211,000 : Strongest negative impact remains

Multicollinearity Status Condition No. = 3.36e+03 Slightly improved, but not significant Dropping Interest Rate % reduced predictive power more than redundancy

Sentiment alone weakly and negatively correlates with FTSE250. Interest rates needed to explain markets, aligning with BoE's financial stability mandate.



- **Focus on macro vs microeconomic factors**
  - Macroeconomic (GDP growth, inflation, interest rates, consumer confidence, umployment): broad indicators of overall economic conditions. Influence entire markets, including stock indices (FTSE250). Closely tied to market expectations and investor sentiment.
  - Microeconomic (individual behaviours, earnings, management decisions, product launches): highly volatile and noisy. Often unpredictable, influenced by news, competitor actions and one-off events.

PATTERNS, TRENDS & INSIGHTS

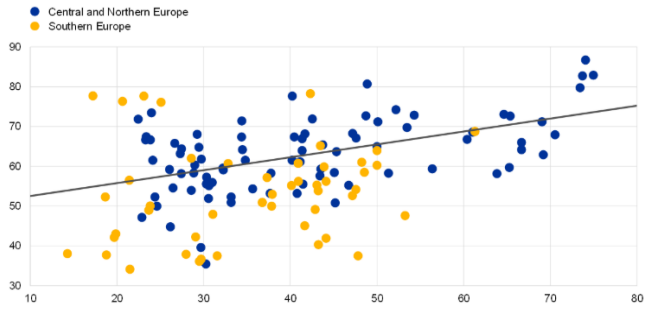
- Speeches are increasingly concise & frequent.
- Governor tone is more optimistic.
- Major events and MPC voting results drive shifts in speech sentiment.
- Weak standalone correlation of speech sentiment with economic indicators.
- Hybrid models offer stronger predictive insights.
- Most influential features are consumer confidence, credit growth and interest rates.

Measure speech complexity using models like Flesch-Kincaid score, based on sentence length and words.[1]

Chart 9

The relationship between trust in the ECB and social trust

(x-axis: regional share of respondents stating that other people can generally be trusted; y-axis: regional share of respondents stating that they trust the ECB)

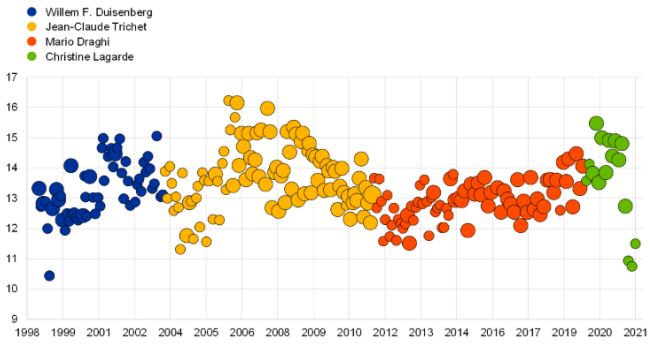


Source: Angino et al. (2021).

Note: The chart displays the share of European Social Survey (ESS) respondents who state that most people can be trusted and the share of K&A respondents who state they trust the ECB. To measure social trust, data from the ESS between 2002 and 2016 are used. To measure trust in the ECB, data from the K&A survey collected in the 19 euro area countries in 2016, 2017 and 2018 are used.

Chart 15

Complexity of ECB monetary policy statements



Source: Updated data (October 2021) from analysis in Coenen et al. (2017).

Notes: The figure depicts the length and the complexity of the ECB's monetary policy statements (until June 2021 "introductory statement"). The length is measured by the number of words (indicated by circle size). The difficulty of the language employed is measured using the Flesch-Kincaid Grade Level score, which indicates how many years of formal training are required to understand the text, based on the length of its sentences and words.

[1] Marius Gardt, Siria Angino, Simon Mee, Gabriel Glöckler, ECB communication with the wider public, ECB Economic Bulletin, Issue 8/2021 https://www.ecb.europa.eu/press/economic-bulletin/articles/2022/html/ecb.ebart202108_02~5c1e5a116d.en.html

RECOMMENDATIONS

- **Strengthen Communication Impact**
  Expand BoE wordlist for consistent sentiment analysis over time
  - fully reflect UK-specific financial and policy language
  - domain-relevant terms for sentiment nuance
  - reduce negative skew by accurately capturing cautious vs optimistic tone

  Include language complexity
  - define measures of speech complexity
  - change over time
  - impact on consumer confidence

  Leverage sentiment in policy communication
  - clear, concise and transparent language to boost transparency and audience reach
  - reinforce market stability, especially during times of crisis

  Analyse external sources, such as (re)tweets, to understand public receives sentiment

- **Strengthen Modelling Precision**
  Speech sentiment limited predictive power ⇒ hybrid model to simulate complex scenarios
  - combine sentiment with additional macroeconomic variables
  - use rolling sentiment as leading indicator
  - deepen analysis around key events, like interest rate votes, and surprise vs anticipated events

- **Strengthen Policy Accountability**
  Policy Risk Radar Dashboard to track economic uncertainty and impact on consumer confidence
  - track nuanced speech tone and complexity by vote type, macroeconomic trends, and key events
  - visualise tone shifts relative to key policy milestones

  Regular briefings for executives to support strategic planning
  - highlight tone and complexity trends, author variation and market reactions
  - support strategy and crisis planning with sentiment-driven insights

OPPORTUNTIES / NEXT STEPS

- Broader event analysis around MPC vote decisions
- Speech sentiment variation ahead of anticipated versus surprise events
- Thematic context: e.g. correlation of speech sentiment for speeches about interest rates with interest rates
- Analyse speech content, including speech complexity and community engagement

⇒ use improved sentiment lexicon scores with cross statistical analysis to unlock predictive power
⇒ make data driven decisions and use communication strategy to engage with wider public and strengthen consumer confidence

APPENDIX

To support the technical report, some additional context has been provided.

Content

- Sources Economic Indicators
- Define functions
- MPC Voting Data

Sources for Economic Indicators (*Consolidated_Eco_KPI_V3.xlsx & Consolidated_Eco_KPI_Sources.xlsx*)

KPI	Source	Link	Data Manipulation
Inflation CPIH	ONS	CPIH ANNUAL RATE 00: ALL ITEMS 2015=100 - Office for National Statistics	None
Unemployment	ONS	Unemployment rate (aged 16 and over, seasonally adjusted): % - Office for National Statistics	None
GDP Growth	ONS	Gross Domestic Product: Quarter on Quarter growth: CVM SA % - Office for National Statistics	Quarterly to monthly
Interest rate	BoE	Bank Rate history and data | Bank of England Database	Date of change to monthly
Consumer confidence	FED	Consumer Opinion Surveys: Composite Consumer Confidence for United Kingdom (CSCICP02GBM460S) | FRED | St. Louis Fed	None
FTSE	YahooFinance	FTSE 250 (^FTMC) historical data - Yahoo Finance	Daily to monthly average
FX USD	BoE	Bank of England | Database	Daily to monthly average
GILTS	DMO	Historical Average Daily Conventional Gilt Yields	None
Credit	BoE	Bank of England | Database	None
House price	UK House Price Index	House Price Statistics - UK House Price Index	None

Note: FTSE250 was selected over FTSE100, because FTSE100 includes mainly global companies, while the FTSE250 also includes UK companies more generally impacted by UK policy.

Define functions Charts

```
def clean_label(label):
    # If label is a Series, return its name.
    if isinstance(label, pd.Series):
        return label.name.replace('_', ' ').title() if label.name else ' '
    elif isinstance(label, str):
        return label.replace('_', ' ').title()
    return ' '
```

```
# Define function for scatterplot.
def generate_scatterplot(df, x_axis, y_axis, title, hue, save_path=None):

    # Set figure size & style for seaborn.
    sns.set_theme(style='darkgrid')
    sns.set(rc={'figure.figsize':(8, 6)})

    # Plot the scatterplot.
    sns.scatterplot(data=df, x=x_axis, y=y_axis, hue=hue, color='#0e1b2c')

    # Customize the plot.
    plt.title(title, fontsize=12, fontweight='bold')
    plt.xlabel(clean_label(x_axis), fontsize=10)
    plt.ylabel(clean_label(y_axis), fontsize=10)

    # Add legend ONLY if hue is not None.
    if hue is not None:
        plt.legend(title='Legend', fontsize=10, bbox_to_anchor=(1.05,1), loc='upper left')

    # Save the plot, if save_path is provided.
    if save_path:
        plt.savefig(save_path, dpi=500, bbox_inches='tight')

    # Display the chart.
    plt.tight_layout()
    plt.show()
```

```python
# Define function to plot a lineplot with various categories.
def generate_multi_lineplot(df, x_axis, y_axis, hue, title, date=None, ylim=None, \
                            save_path=None, errorbar=None, events=None, event_years=None, **kwargs):

    # Set figure size & style for seaborn.
    sns.set_theme(style='darkgrid')
    sns.set(rc={'figure.figsize':(14, 8)})

    # Sort DataFrame by the time column.
    df.sort_values(by=x_axis, inplace=True)

    # Plot the lineplot.
    sns.lineplot(data=df, x=x_axis, y=y_axis, hue=hue, ci=None, **kwargs)

    # Get current axes
    ax = plt.gca()

    # Customize the plot.
    plt.title(title, fontsize=16, fontweight='bold')
    plt.xlabel(clean_label(x_axis), fontsize=14)
    plt.ylabel(clean_label(y_axis), fontsize=14)
    plt.legend(title='Legend', fontsize=12, bbox_to_anchor=(1.05,1), loc='upper left')
    plt.tick_params(axis='both', labelsize=12)

    # Add annotation lines, if provided.
    if date:
        plt.axvline(x=date, color='k', linestyle='--')

    # Add shading for each event period
    if events:
        for start_date, end_date in events:
            ax.axvspan(start_date, end_date, color='yellow', alpha=0.3)

    # Add event lines if provided
    if event_years:
        for year, label in event_years:
            ax.axvline(x=year, color='blue', linestyle='--', alpha=0.7)
            # Add a label or annotation
            y_position = plt.ylim()[1] * 0.3  # 30% of max y for label placement
            ax.text(year, y_position, label, rotation=90, verticalalignment='bottom', color='blue')

    # Rotate labels on the x-axis.
    plt.xticks(rotation=45)

    # Set y-axis limits, if provided.
    if ylim:
        plt.ylim(ylim)

    # Save the plot, if save_path is provided.
    if save_path:
        plt.savefig(save_path, dpi=500, bbox_inches='tight')

    # Display the chart.
    plt.tight_layout()
    plt.show()
```

```python
# Define function to plot a lineplot with a trendline.
def generate_lineplot_with_trendline(df, x_axis, y_axis, title, date=None, ylim=None, \
                                     rotate_xticks=False, save_path=None, errorbar=None):

    # Set figure size & style for seaborn.
    sns.set_theme(style='darkgrid')
    sns.set(rc={'figure.figsize':(14, 8)})

    # Ensure time column is in datetime format.
    df[x_axis] = pd.to_datetime(df[x_axis])

    # Sort DataFrame by the time column.
    df.sort_values(by=x_axis, inplace=True)

    # Calculate trendline.
    x = np.arange(len(df))
    y = df[y_axis].values
    z = np.polyfit(x, y, 1)
    p = np.poly1d(z)

    # Plot the lineplot.
    sns.lineplot(data=df, x=x_axis, y=y_axis, label=clean_label(y_axis))

    # Plot the trendline.
    plt.plot(df[x_axis], p(x), linestyle='--', color='r', label='Trendline')

    # Customize the plot.
    plt.title(title, fontsize=16, fontweight='bold')
    plt.xlabel(clean_label(x_axis), fontsize=14)
    plt.ylabel(clean_label(y_axis), fontsize=14)
    plt.legend(title='Legend', fontsize=12, bbox_to_anchor=(1.05,1), loc='upper left')
    plt.tick_params(axis='both', labelsize=12)

    # Add annotation lines, if provided.
    if date:
        plt.axvline(x=date, color='k', linestyle='--')

    # Rotate x-tick labels by 45 degrees, if specified.
    if rotate_xticks:
        plt.xticks(rotation=45)

    # Set y-axis limits, if provided.
    if ylim:
        plt.ylim(ylim)

    # Save the plot, if save_path is provided.
    if save_path:
        plt.savefig(save_path, dpi=500, bbox_inches='tight')

    # Display the chart.
    plt.tight_layout()
    plt.show()
```

Define Functions Natural Language Processing

Cleaning the data

```
# Load English stop words for text preprocessing tasks
stop_words = set(stopwords.words('english'))
```

```
# Preprocessing function
def preprocess_text(text):
    text = contractions.fix(text)  # Expand contractions i.e I'm not good goes to I am not good
    text = re.sub(r'http\S+', '', text)  # Remove URLs
    text = re.sub('#', '', text)         # Remove hashtags
    text = re.sub(r'\W', ' ', text)      # Remove special characters
    text = text.lower()                  # Convert to lowercase
    #Below is to create a set of stop words from the NLTK library's predefined list but not is excluded.
    stop_words = set(stopwords.words('english')) - {'not'}
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text
```

```
# Define the tag map for POS tagging.
tag_map = defaultdict(lambda: wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV

# Lemmatise the tokens with correct POS tags.
lemma_function = WordNetLemmatizer()

# Lemmatisation function.
def lemmatize_tokens(tokens):
    #For each word in the token list, it lemmatizes the word with the correct part-of-speech
    lemmatized_tokens = [lemma_function.lemmatize(token, tag_map[tag[0]]) for token, tag in pos_tag(tokens)]
    return lemmatized_tokens
```

Vader Sentiment

```
# VADER Sentiment Intensity Analyzer.
analyzer = SentimentIntensityAnalyzer()

# Define the function to compute and return sentiment scores.
def analyse_sentiment(text):
    return analyzer.polarity_scores(' '.join(text))
```

```
# Categories VADER sentiment according to compound_score
def vader_sentiment(compound_score):
    if compound_score >= 0.05:
        return 'Positive'
    elif compound_score <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'
```

Polarity & Subjectivity

```
# Define a function to extract a polarity score using TextBlob.
def generate_polarity(comment):
    return TextBlob(comment).sentiment[0]
```

```
# Define a function to extract a subjectivity score using TextBlob.
def generate_subjectivity(comment):
    return TextBlob(comment).sentiment[1]
```

Dictionaries

```
# Define function to apply the lexicon to the text
def lexicon_counts(tokens):
    return pd.Series({
        cat: sum(t in word_sets[cat] for t in tokens)
        for cat in categories
    })
```

```
# Define function to apply the lexicon to the text
def lexicon_score_weighted(tokens):
    score = 0
    for cat in categories:
        count = sum(t in word_sets[cat] for t in tokens)
        score += count * category_weights[cat]
    return score
```

Finbert

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
# FinBERT Models: Load model 1: ProsusAI/finbert
tokenizer_prosus = AutoTokenizer.from_pretrained('ProsusAI/finbert')
model_prosus = AutoModelForSequenceClassification.from_pretrained('ProsusAI/finbert').to(device)
```

```
# FinBERT Models: Load model 2: yiyanghkust/finbert-tone
tokenizer_yiyang = AutoTokenizer.from_pretrained('yiyanghkust/finbert-tone')
model_yiyang = AutoModelForSequenceClassification.from_pretrained('yiyanghkust/finbert-tone').to(device)
```

```
# Define a function to predict probabilities in batches
def predict_batch(texts, tokenizer, model, max_length=128):
    inputs = tokenizer(texts, padding=True, truncation=True, max_length=max_length, return_tensors='pt')
    inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    probs = F.softmax(outputs.logits, dim=1)
    return probs.cpu().numpy()
```

```
# Define function to calculate one sentiment score
def compute_tone_score(probs):
    class_labels = ['Neutral', 'Positive', 'Negative']
    prob_dict = dict(zip(class_labels, probs))
    return (
        prob_dict['Positive'] * weights['Positive'] +
        prob_dict['Neutral'] * weights['Neutral'] +
        prob_dict['Negative'] * weights['Negative']
    )
```

MPC Voting Dataset: Process of data preparation for further analysis

- Clean and transpose headers to retrieve list of current and past members names
  - split names to given and family (all low case) to match author names in sentiment dataset
- Clean and transpose member names and their votes to create a table with vote date, voted rate member names and their voted rate
- Calculate whether the member voted to increase, reduce or maintain the actual rate
  - Calculate whether rate was increased, reduced or maintained as a result of the vote
- Outputs
  - Table of vote dates and vote result: increase, reduce, maintain ⇒ used to match with speech data to analyse sentiment around votes
  - Table of vote dates, member's family names and vote direction: increase, reduce, maintain