Classical NLP Approach - Sentiment Analysis

Load cleaned & merged dataset

```
import pandas as pd

from google.colab import drive
drive.mount('/content/drive')

# Load the dataset that has transcripts + economic indicators
df_merged = pd.read_csv('/content/drive/My Drive/NLP/Assignment_3/merged_fomc_with_indica
print("✅ Dataset loaded.")
```

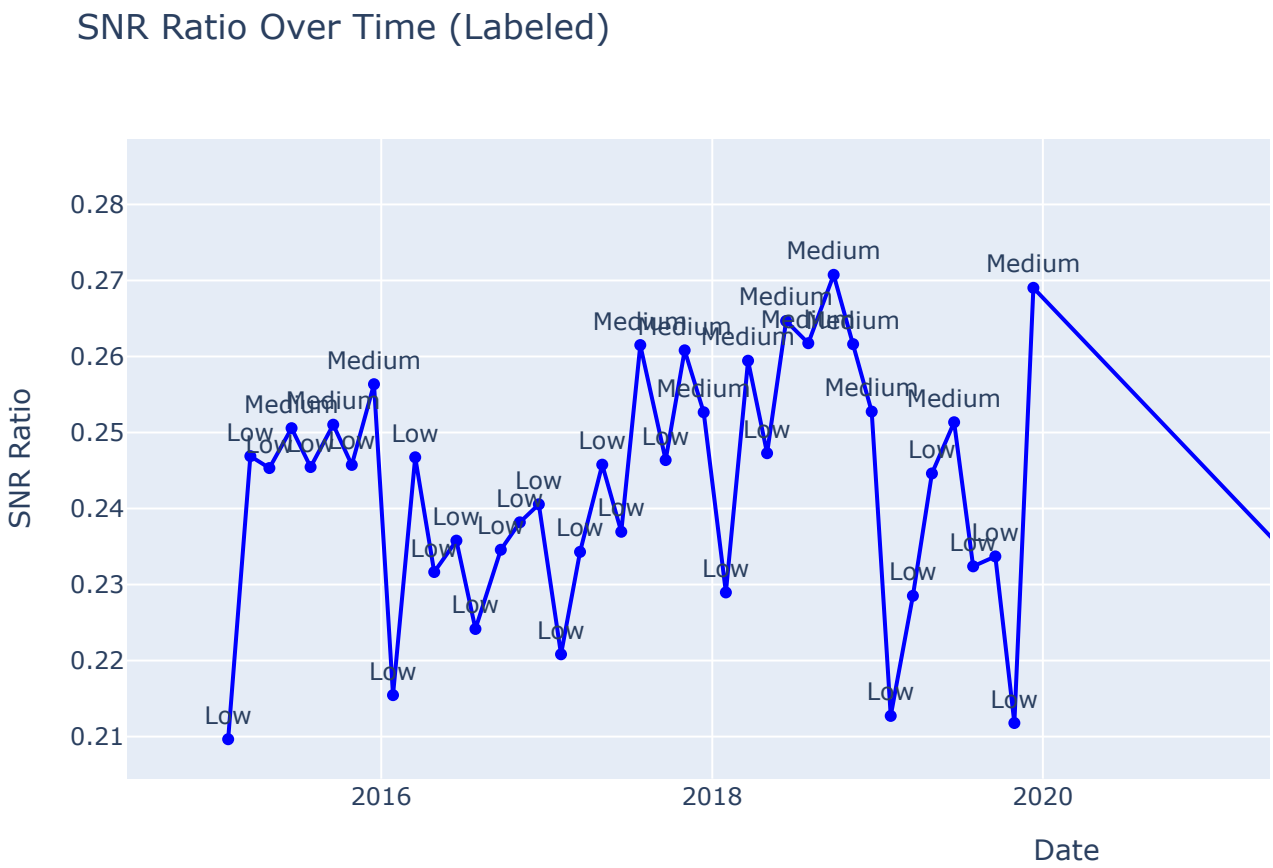
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
✅ Dataset loaded.
```

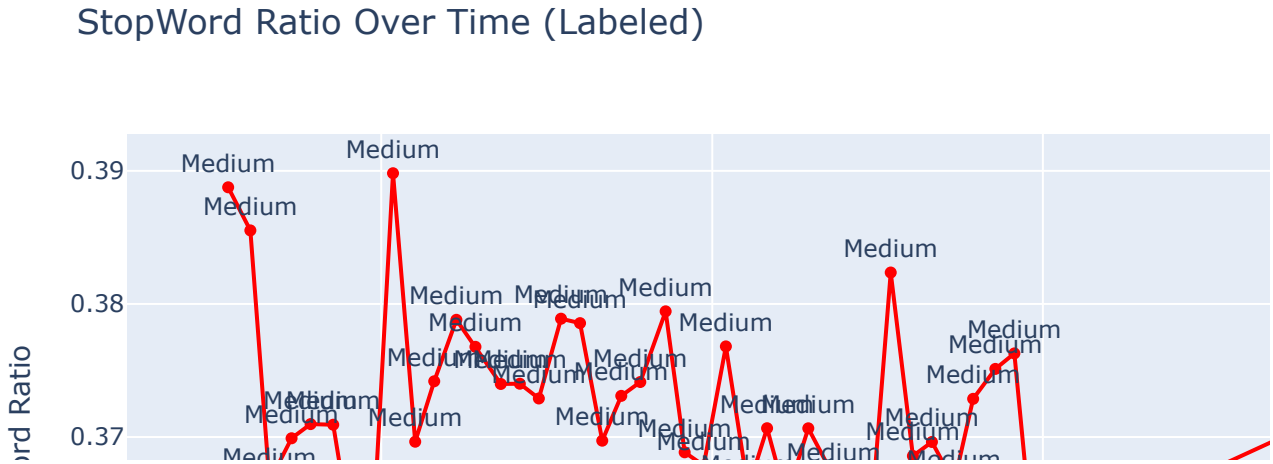
NLP + Quality Analysis before Cleaning

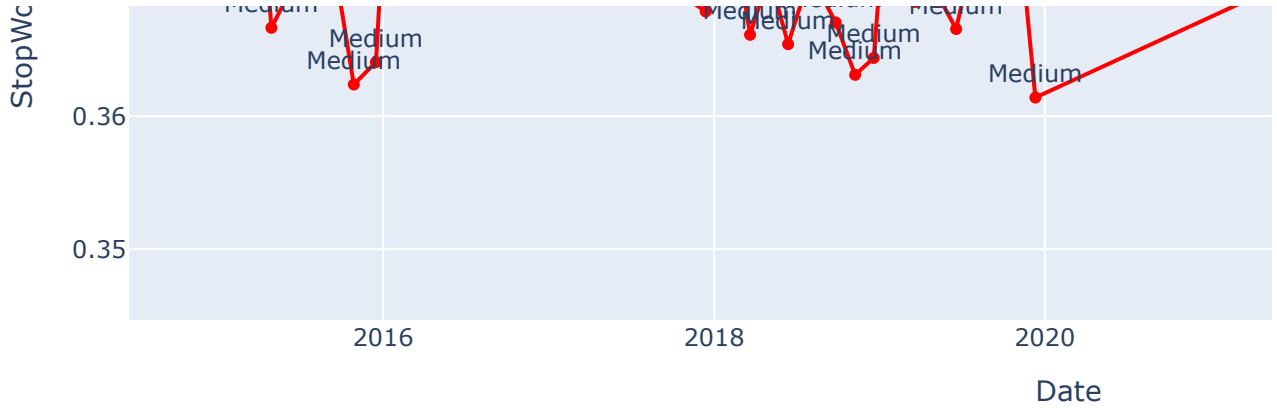
```
# --- 3. Noise + Quality Metric Analysis (Pre-cleaning) ---
import numpy as np
import string
from wordcloud import STOPWORDS
import plotly.graph_objs as go
import spacy  # Import spaCy

# Load the spaCy model (you might need to download it first)
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    !python -m spacy download en_core_web_sm
    nlp = spacy.load("en_core_web_sm")

# Metric functions
def calc_snr_ratio(text):
    words = text.split()
    return len(set(words)) / (len(words) + 1e-5) if words else 0

def calc_stopword_ratio(text):
    words = text.split()
    return sum(1 for w in words if w.lower() in STOPWORDS) / len(words) if words else 0

def calc_redundancy_ratio(text):
```

```
    words = text.split()
    return 1 - (len(set(words)) / len(words)) if words else 0

def calc_special_char_density(text):
    return sum(1 for c in text if c in string.punctuation) / (len(text) + 1e-5)

def calc_semantic_density(text):
    doc = nlp(text)
    return sum(1 for t in doc if t.pos_ in ['NOUN', 'VERB', 'ADJ']) / len(doc) if len(doc

def calc_digit_ratio(text):
    return sum(1 for c in text if c.isdigit()) / (len(text) + 1e-5)

def calc_noise_score(row):
    return row['StopWord_Ratio'] + row['Digit_Ratio'] + row['SpecialChar_Density']

def calc_quality_score(row):
    return row['SNR_Ratio'] + row['Semantic_Density'] - row['Noise_Score']

# Apply metrics on raw Content
df_merged['SNR_Ratio'] = df_merged['Content'].apply(calc_snr_ratio)
df_merged['StopWord_Ratio'] = df_merged['Content'].apply(calc_stopword_ratio)
df_merged['Redundancy_Ratio'] = df_merged['Content'].apply(calc_redundancy_ratio)
df_merged['SpecialChar_Density'] = df_merged['Content'].apply(calc_special_char_density)
df_merged['Semantic_Density'] = df_merged['Content'].apply(calc_semantic_density)
df_merged['Digit_Ratio'] = df_merged['Content'].apply(calc_digit_ratio)
df_merged['Noise_Score'] = df_merged.apply(calc_noise_score, axis=1)
df_merged['Quality_Score'] = df_merged.apply(calc_quality_score, axis=1)

# Plot each metric individually
metric_colors = {
    'SNR_Ratio': 'blue',
    'StopWord_Ratio': 'red',
    'Redundancy_Ratio': 'green',
    'SpecialChar_Density': 'purple',
    'Semantic_Density': 'orange',
    'Digit_Ratio': 'brown',
    'Noise_Score': 'teal',
    'Quality_Score': 'maroon'
}

for metric, color in metric_colors.items():
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=df_merged['Date'],
        y=df_merged[metric],
        mode='lines+markers+text',
        text=["Low" if v < 0.25 else "Medium" if v < 0.5 else "High" for v in df_merged[m
        textposition="top center",
        marker=dict(color=color),
        name=metric
```

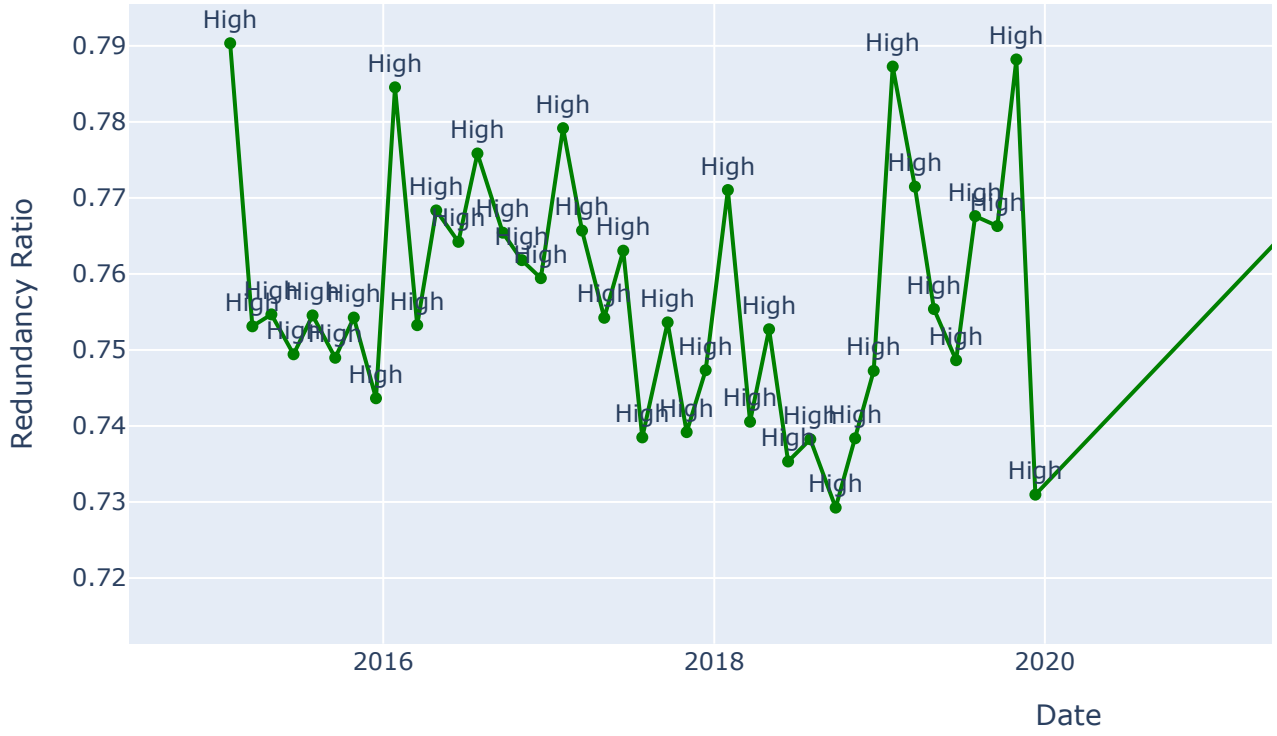
```
    name=metric
))
fig.update_layout(
    title=f"{metric.replace('_', ' ')} Over Time (Labeled)",
    xaxis_title='Date',
    yaxis_title=metric.replace('_', ' '),
    height=500,
    width=1100
)
fig.show()
```



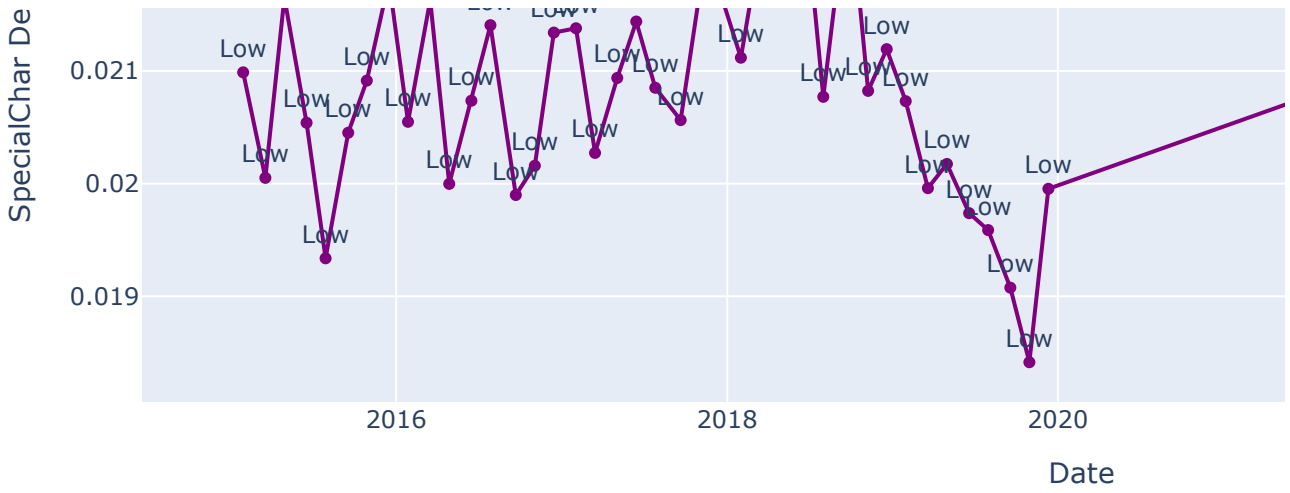




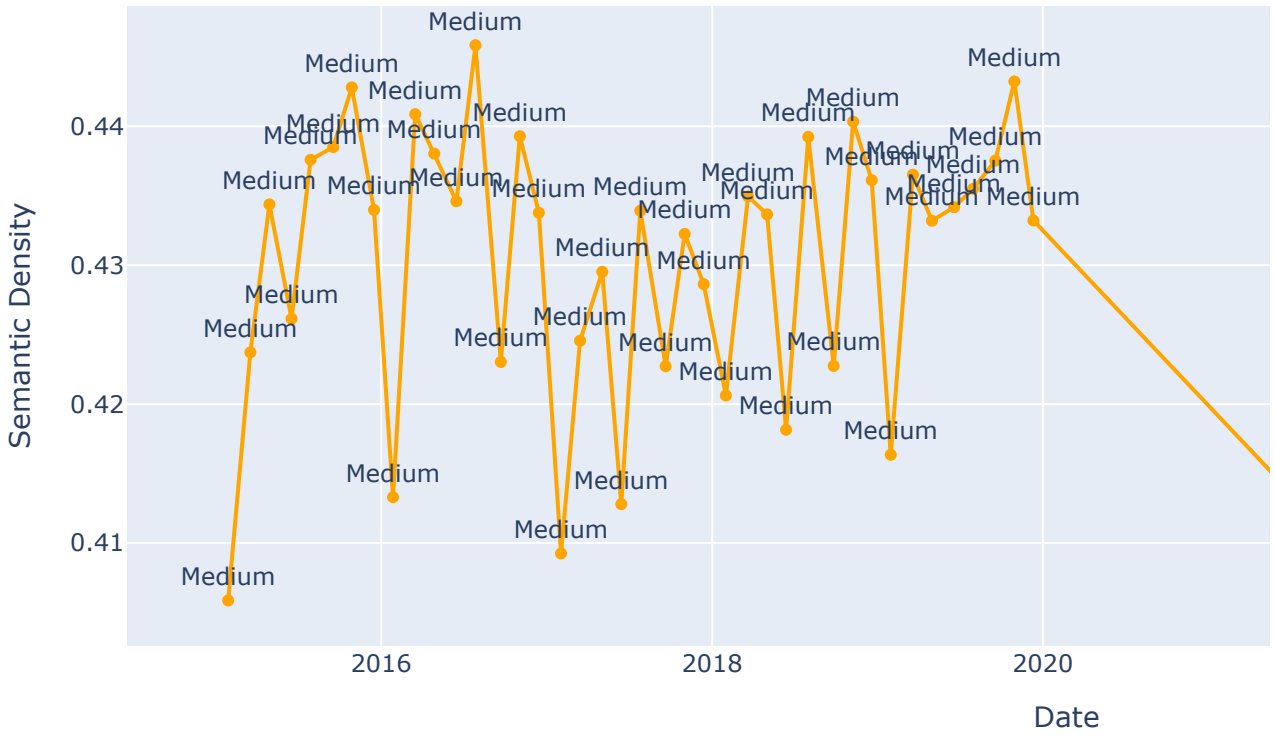
Redundancy Ratio Over Time (Labeled)



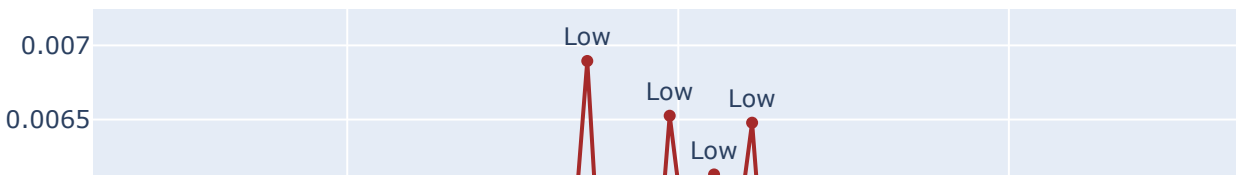
SpecialChar Density Over Time (Labeled)

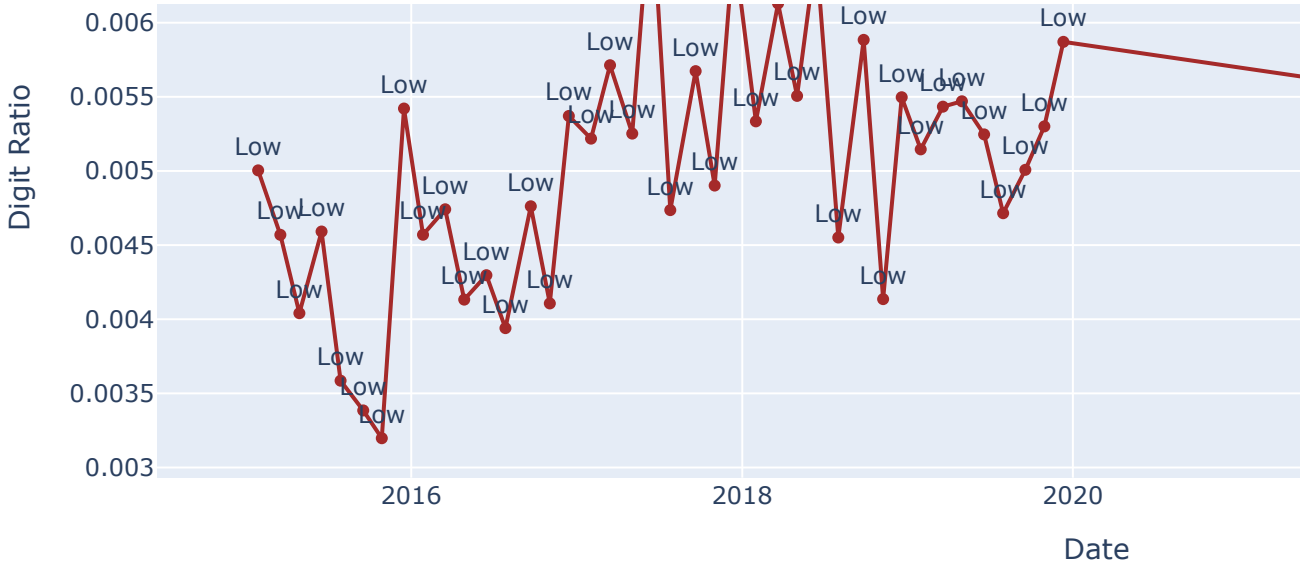


Semantic Density Over Time (Labeled)

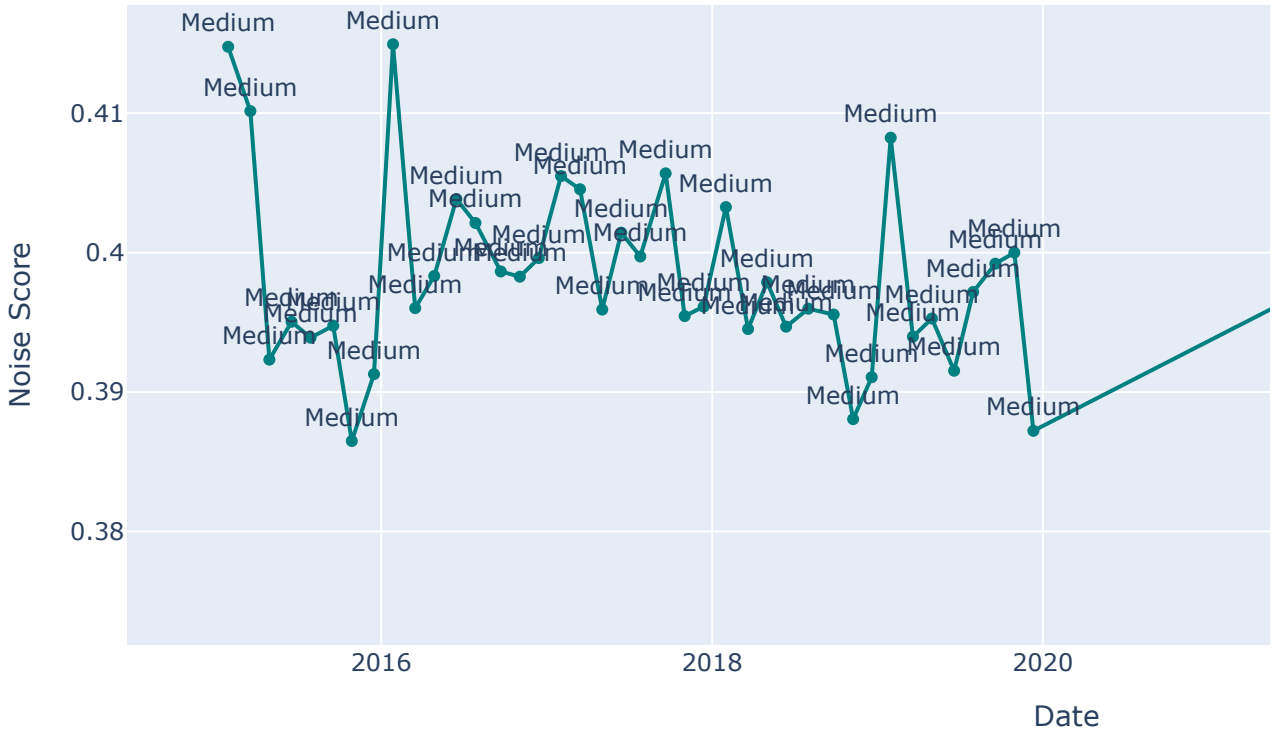


Digit Ratio Over Time (Labeled)

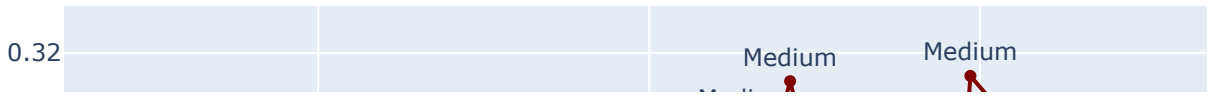


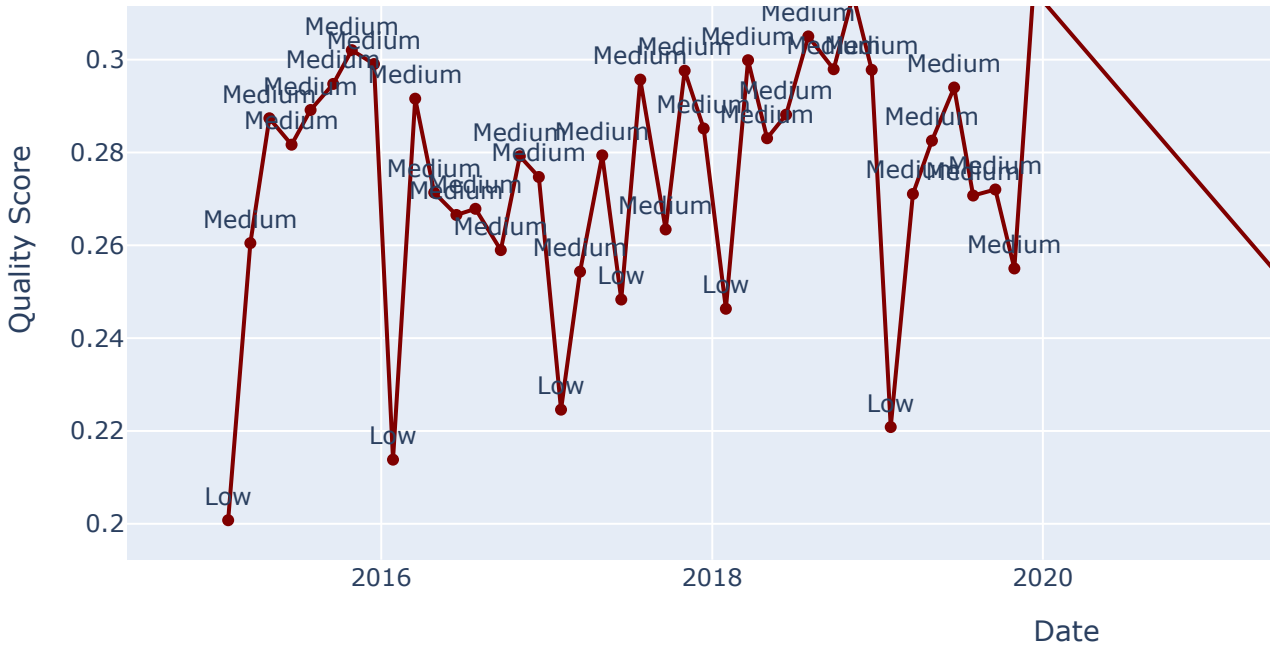


Noise Score Over Time (Labeled)



Quality Score Over Time (Labeled)






```
import os
import spacy
from bs4 import BeautifulSoup
import re
import string


# Cleaning pipeline
try:
    nlp = spacy.load("en_core_web_sm")
except:
    os.system("python -m spacy download en_core_web_sm")
    nlp = spacy.load("en_core_web_sm")

def minimal_clean(text):
    text = BeautifulSoup(text, "html.parser").get_text()
    text = re.sub(r'http\S+|www\S+', '', text)  # Remove URLs
    text = re.sub(r'\S+@\S+', '', text)         # Remove emails
    text = re.sub(r'\s+', ' ', text).strip()    # Normalize whitespace
    return text

df_merged['cleaned_text'] = df_merged['Content'].fillna('').apply(minimal_clean)
print("✅ Texts cleaned.")
```

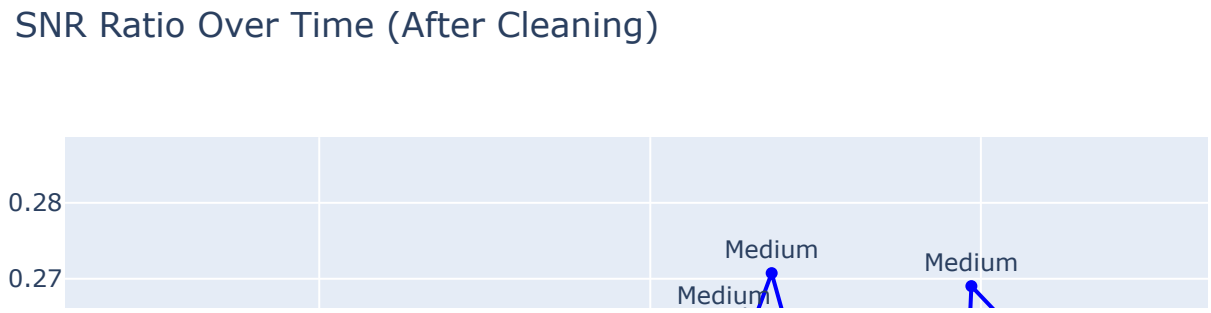
```
✅ Texts cleaned.
```

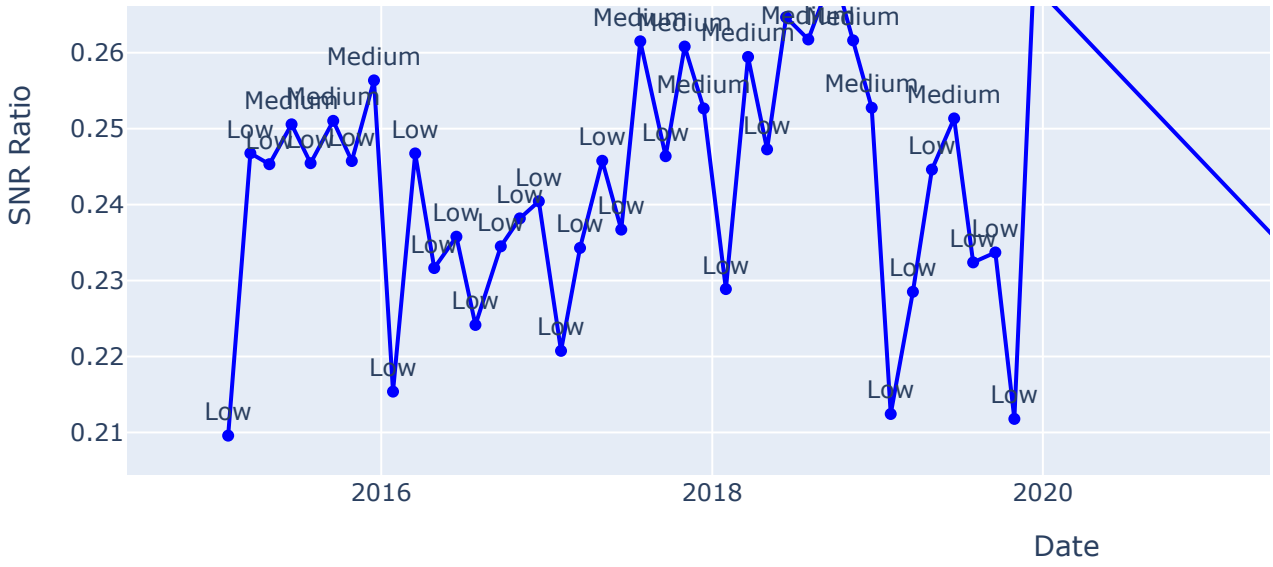
```
# Save checkpoint
output_path_cleaned = "/content/drive/My Drive/NLP/Assignment_3/fomc_transcripts_spacy_cl
df_merged.to_csv(output_path_cleaned, index=False)
print(f"\n✅ Cleaned data saved to {output_path_cleaned}")
```

```
✅ Cleaned data saved to /content/drive/My Drive/NLP/Assignment_3/fomc_transcripts_s
```

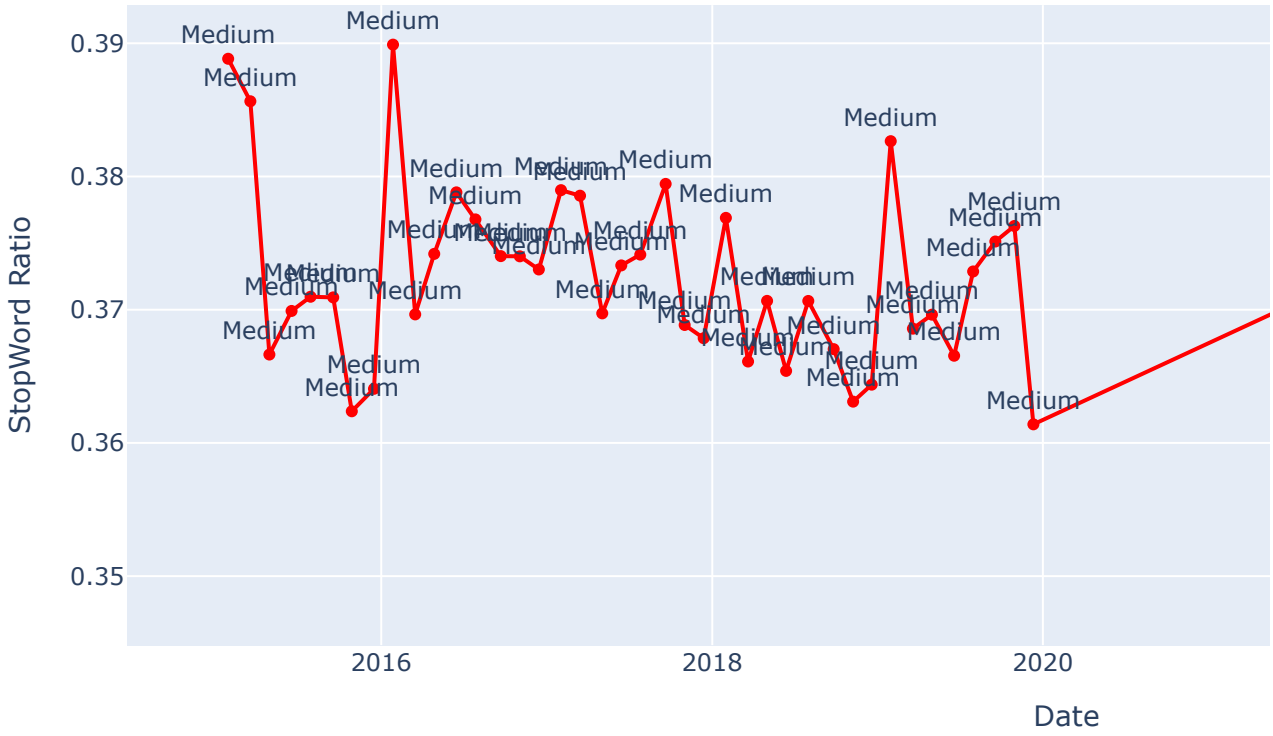
NLP + Quality Analysis after Cleaning

```
# --- 4. Noise Metrics After Cleaning ---
df_merged['SNR_Ratio_Cleaned'] = df_merged['cleaned_text'].apply(calc_snr_ratio)
df_merged['StopWord_Ratio_Cleaned'] = df_merged['cleaned_text'].apply(calc_stopword_ratio
df_merged['Redundancy_Ratio_Cleaned'] = df_merged['cleaned_text'].apply(calc_redundancy_r
df_merged['SpecialChar_Density_Cleaned'] = df_merged['cleaned_text'].apply(calc_special_c
df_merged['Semantic_Density_Cleaned'] = df_merged['cleaned_text'].apply(calc_semantic_den
df_merged['Digit_Ratio_Cleaned'] = df_merged['cleaned_text'].apply(calc_digit_ratio)
df_merged['Noise_Score_Cleaned'] = df_merged.apply(lambda row: row['StopWord_Ratio_Cleane
df_merged['Quality_Score_Cleaned'] = df_merged.apply(lambda row: row['SNR_Ratio_Cleaned']

# Plot each metric individually (After Cleaning)
for metric, color in metric_colors.items():
    metric_clean = metric + '_Cleaned'
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=df_merged['Date'],
        y=df_merged[metric_clean],
        mode='lines+markers+text',
        text=["Low" if v < 0.25 else "Medium" if v < 0.5 else "High" for v in df_merged[m
        textposition="top center",
        marker=dict(color=color),
        name=metric_clean
    ))
    fig.update_layout(
        title=f"{metric.replace('_', ' ')} Over Time (After Cleaning)",
        xaxis_title='Date',
        yaxis_title=metric.replace('_', ' '),
        height=500,
        width=1100
    )
    fig.show()
```

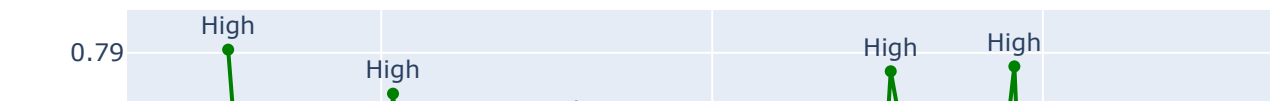


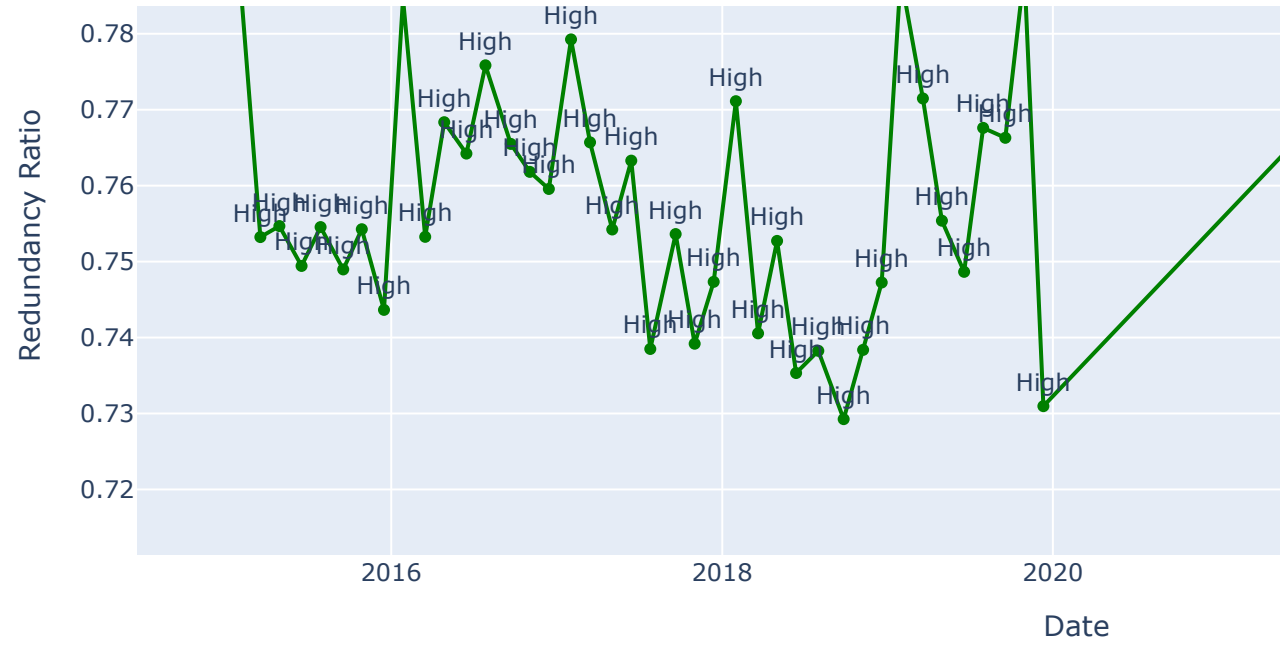


StopWord Ratio Over Time (After Cleaning)

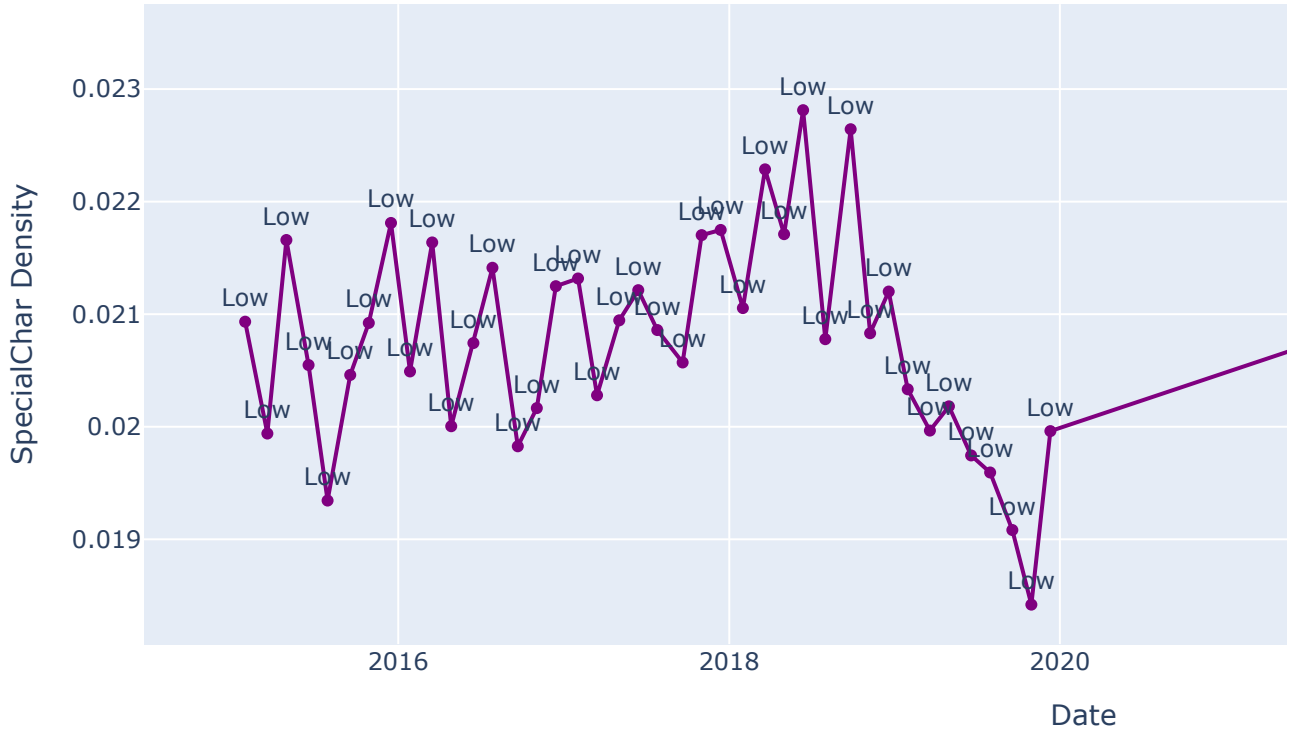


Redundancy Ratio Over Time (After Cleaning)

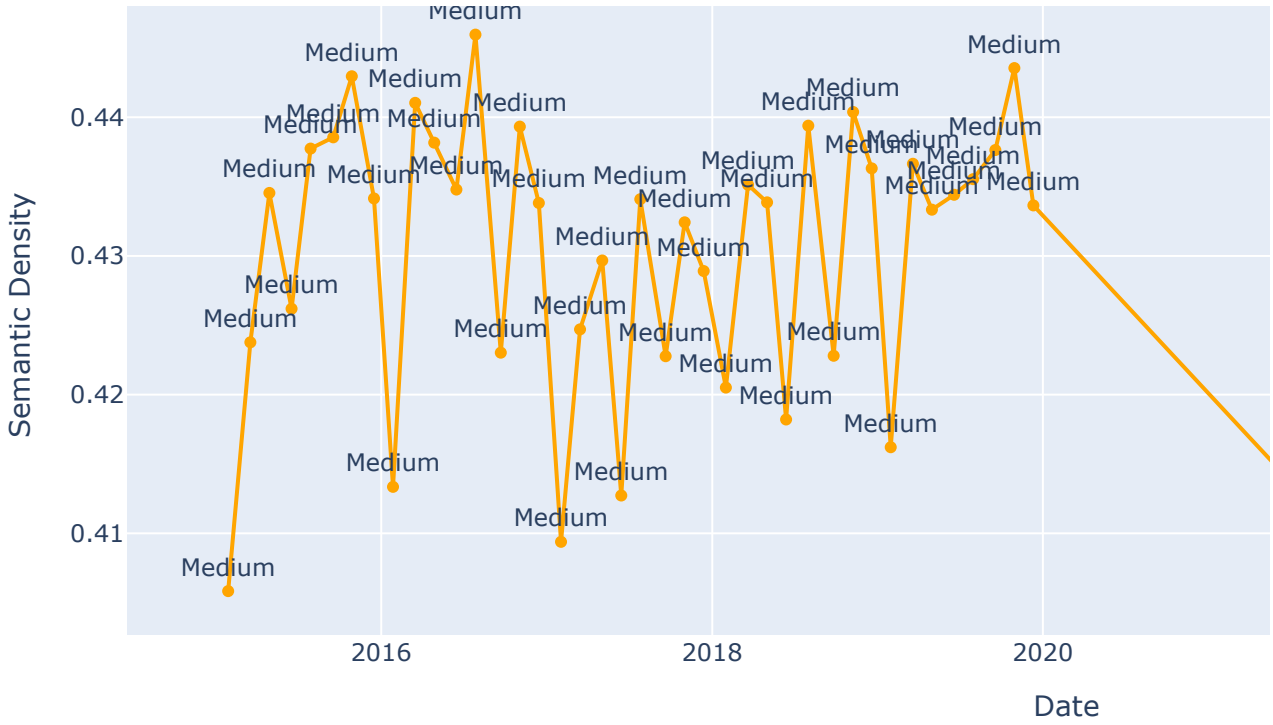




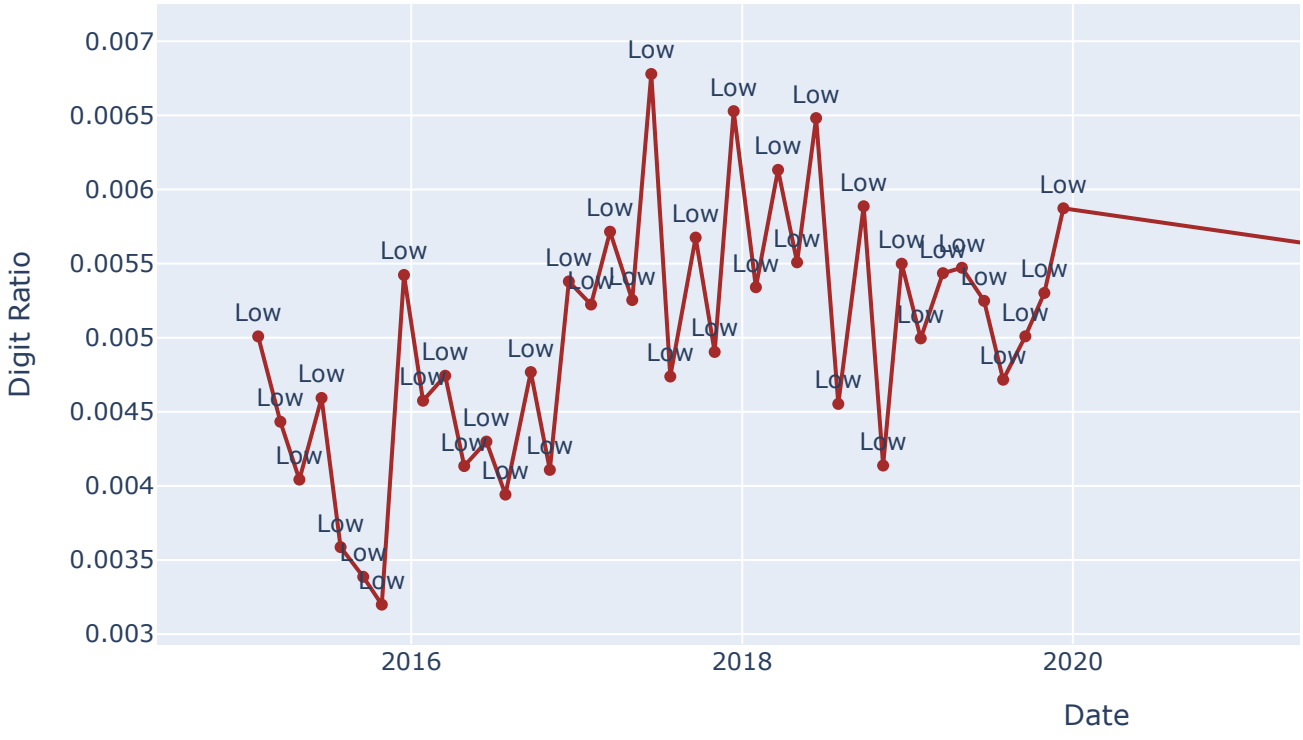
SpecialChar Density Over Time (After Cleaning)



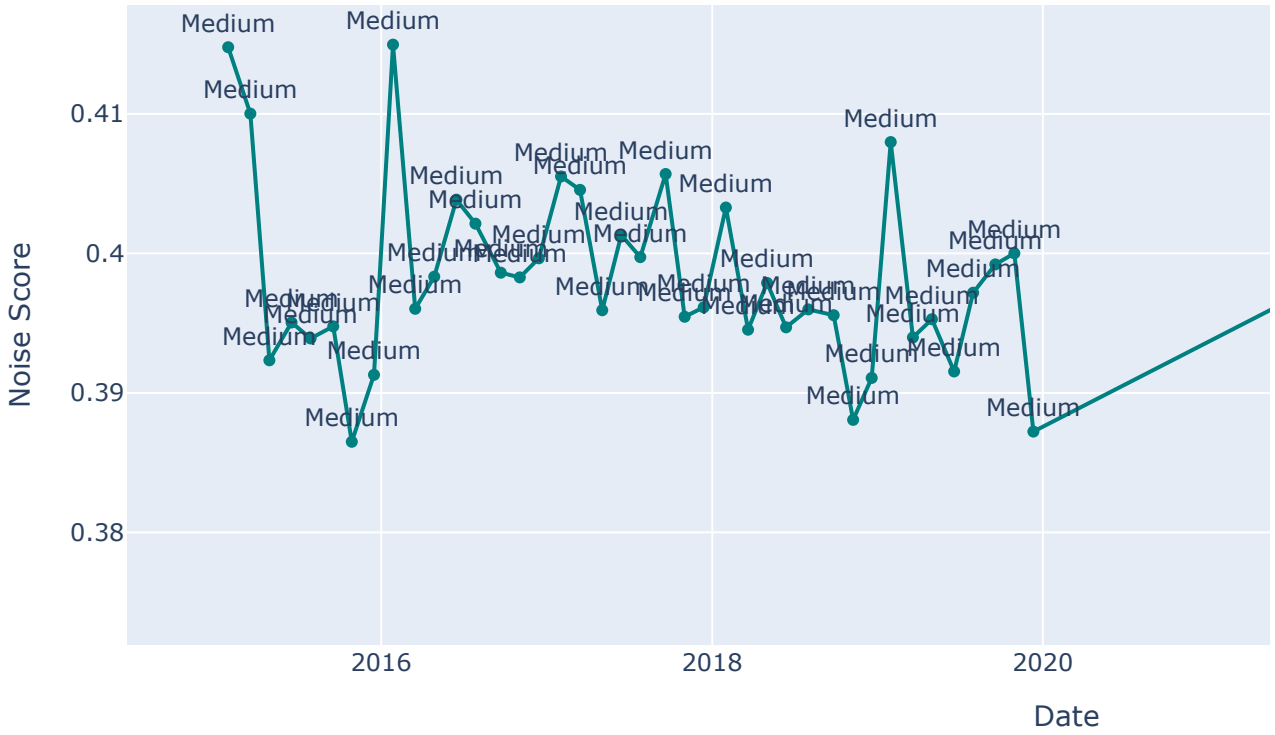
Semantic Density Over Time (After Cleaning)



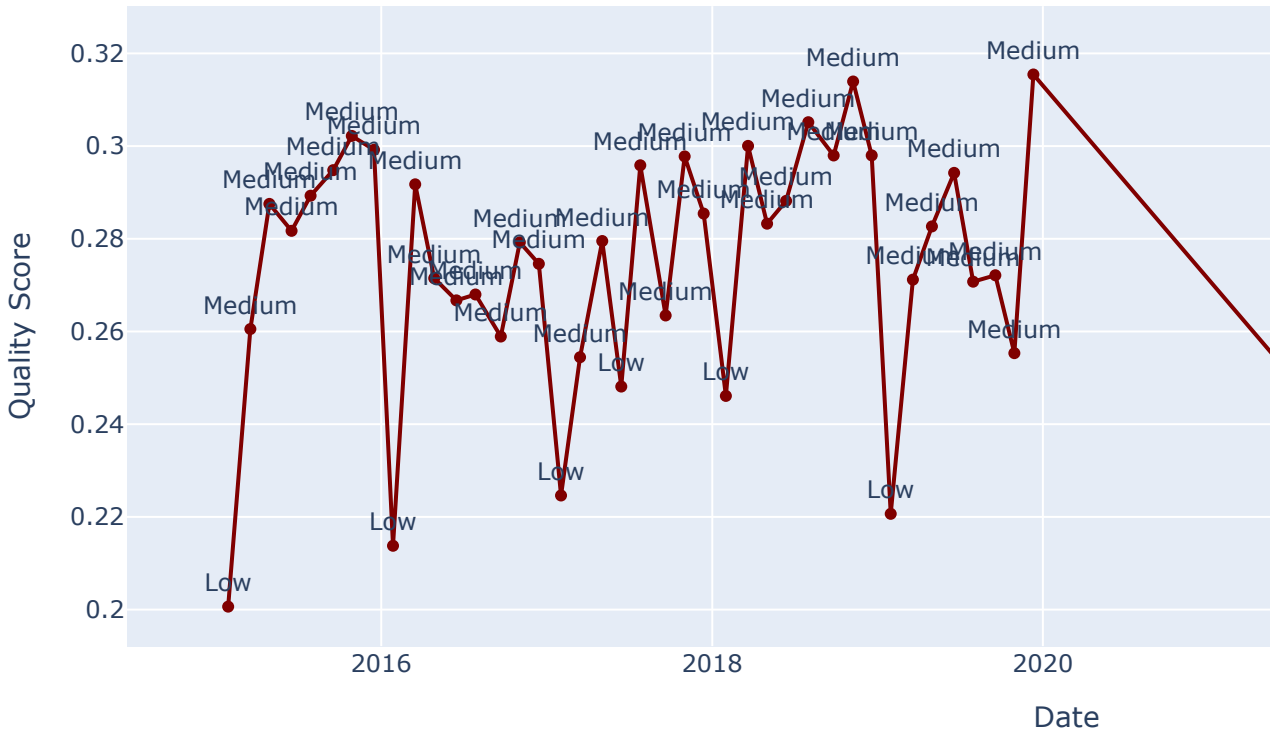
Digit Ratio Over Time (After Cleaning)



Noise Score Over Time (After Cleaning)



Quality Score Over Time (After Cleaning)



VADER Sentiment Analysis on FOMC Transcripts

Used the vaderSentiment library to calculate:

pos: positive score

neg: negative score

neu: neutral score

compound: overall sentiment score

```
# Load Cleaned dataset
df_merged = pd.read_csv('/content/drive/My Drive/NLP/Assignment_3/fomc_transcripts_spacy_
print("✅Cleaned Dataset loaded.")
```

```
✅Cleaned Dataset loaded.
```

```
!pip install vaderSentiment
```

```
Requirement already satisfied: vaderSentiment in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-p
```

```
df_merged['cleaned_text'].str.len().describe()
```

	cleaned_text
count	66.000000
mean	57459.045455
std	12437.504895
min	44054.000000
25%	49508.500000
50%	54478.500000
75%	58313.750000
max	94634.000000

dtype: float64

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download VADER
nltk.download('vader_lexicon')

# Initialize Analyzer
analyzer = SentimentIntensityAnalyzer()

# Apply VADER
df_merged[['neg', 'neu', 'pos', 'compound']] = df_merged['cleaned_text'].apply(lambda x:

print("✅ VADER Sentiment Scores Extracted!")
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
✅ VADER Sentiment Scores Extracted!
```

```
analyzer.polarity_scores(df_merged['cleaned_text'].iloc[0])
```

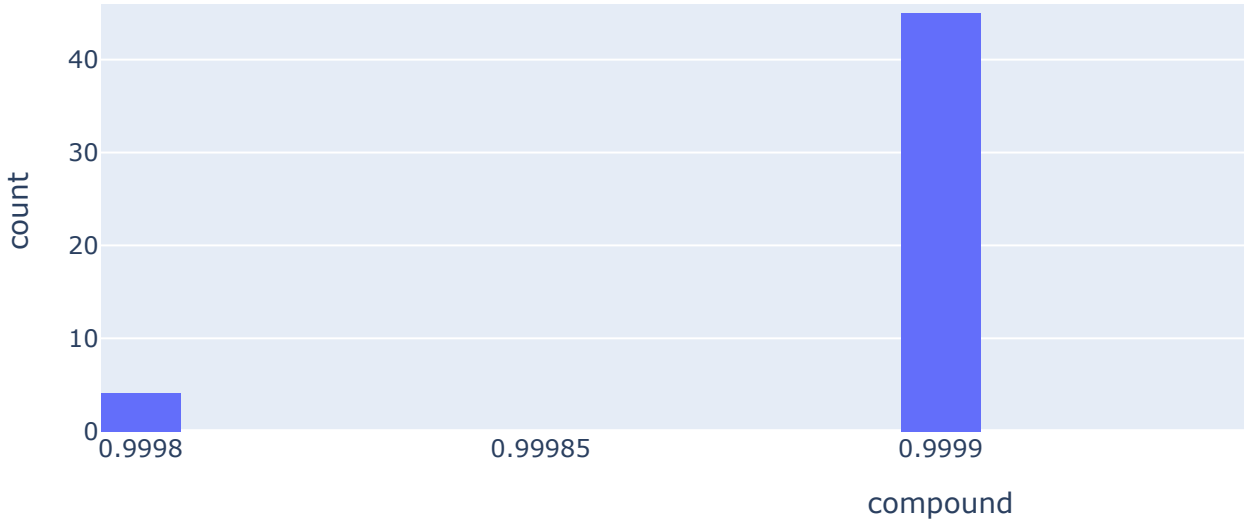
```
{'neg': 0.055, 'neu': 0.849, 'pos': 0.095, 'compound': 0.9998}
```

```
df_merged['cleaned_text'].iloc[0][:1000]
```

```
'HomeMonetary PolicyFederal Open Market CommitteeFederal Open Market CommitteePDFPle
ase enable JavaScript if it is disabled in your browser or access the information th
rough the links provided below.FOMC MinutesMinutes of the Federal Open Market Commit
teeMarch 18â\x80\x9319, 2025A joint meeting of the Federal Open Market Committee and
the Board of Governors of the Federal Reserve System was held in the offices of the
Board of Governors on Tuesday, March 18, 2025, at 9:00 a.m. and continued on Wednesd
ay, March 19, 2025, at 9:00 a.m. 1Review of Monetary Policy Strategy, Tools, and Comm
```

```
# --- Distribution of Compound Scores ---
import plotly.express as px

# Plot distribution of compound scores
fig = px.histogram(df_merged, x='compound', nbins=20, title='Distribution of Compound Sen
fig.update_layout(height=400, width=1000)
fig.show()
```

Distribution of Compound Sentiment Scores



📊 Plot Sentiment Trends Over Time

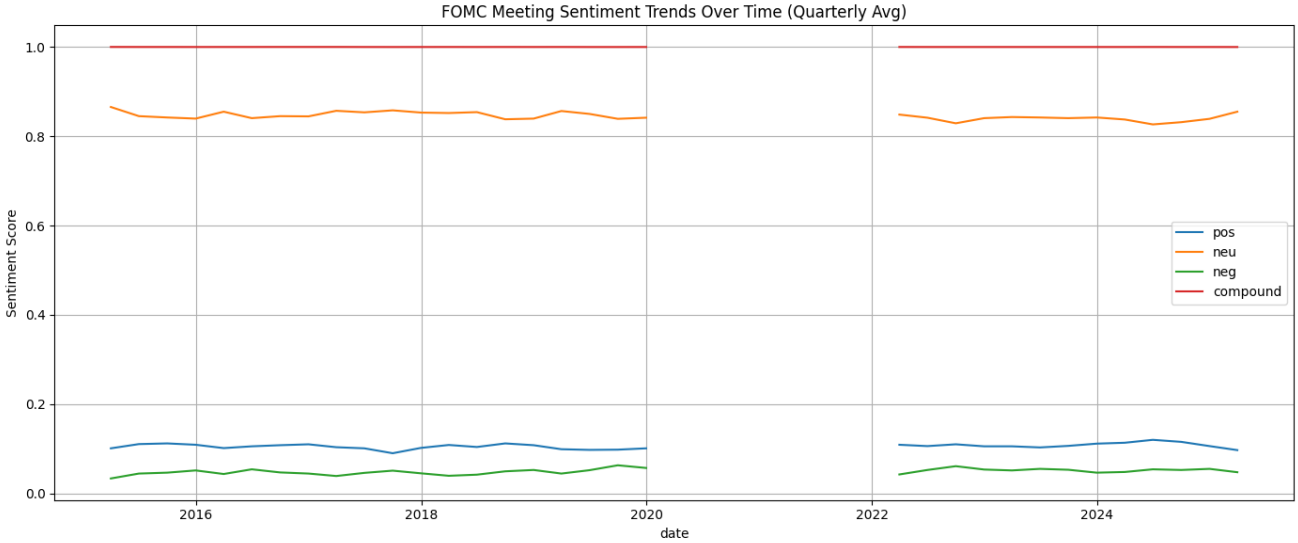
```
# Convert the date column and group by year or month
df_merged['Date'] = pd.to_datetime(df_merged['Date'])
df_merged['Year'] = df_merged['Date'].dt.year
df_merged['Month'] = df_merged['Date'].dt.to_period('M').dt.to_timestamp()

# Group by month for trend overview
sentiment_by_year = df_merged.groupby('Year')[['compound', 'pos', 'neu', 'neg']].mean()
```

Plot the trends

```
import matplotlib.pyplot as plt

# Set datetime index
df_merged['date'] = pd.to_datetime(df_merged['date'])
df_merged.sort_values('date', inplace=True)

# Rolling sentiment (quarterly average)
sentiment_trend = df_merged.set_index('date').resample('Q')[['pos', 'neu', 'neg', 'compou

# Plot sentiment over time
plt.figure(figsize=(14, 6))
for col in ['pos', 'neu', 'neg', 'compound']:
    plt.plot(sentiment_trend.index, sentiment_trend[col], label=col)

plt.title("FOMC Meeting Sentiment Trends Over Time (Quarterly Avg)")
plt.xlabel("date")
plt.ylabel("Sentiment Score")
```

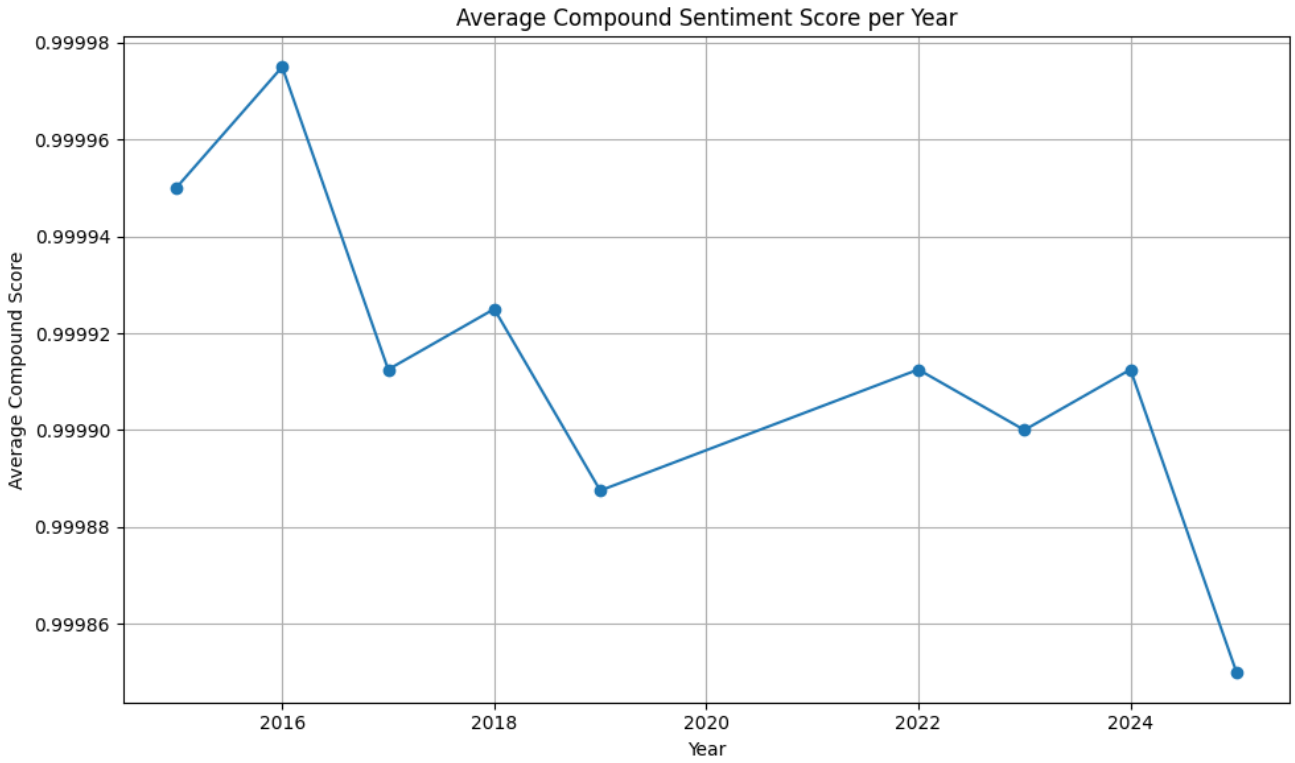
```
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-102-a3b5480ee5c1>:8: FutureWarning:

'Q' is deprecated and will be removed in a future version, please use 'QE' instead.
```




```
import matplotlib.pyplot as plt

# Group by year and compute average compound sentiment score
avg_compound_by_year = df_merged.groupby('Year')['compound'].mean()

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(avg_compound_by_year.index, avg_compound_by_year.values, marker='o', linestyle='
```

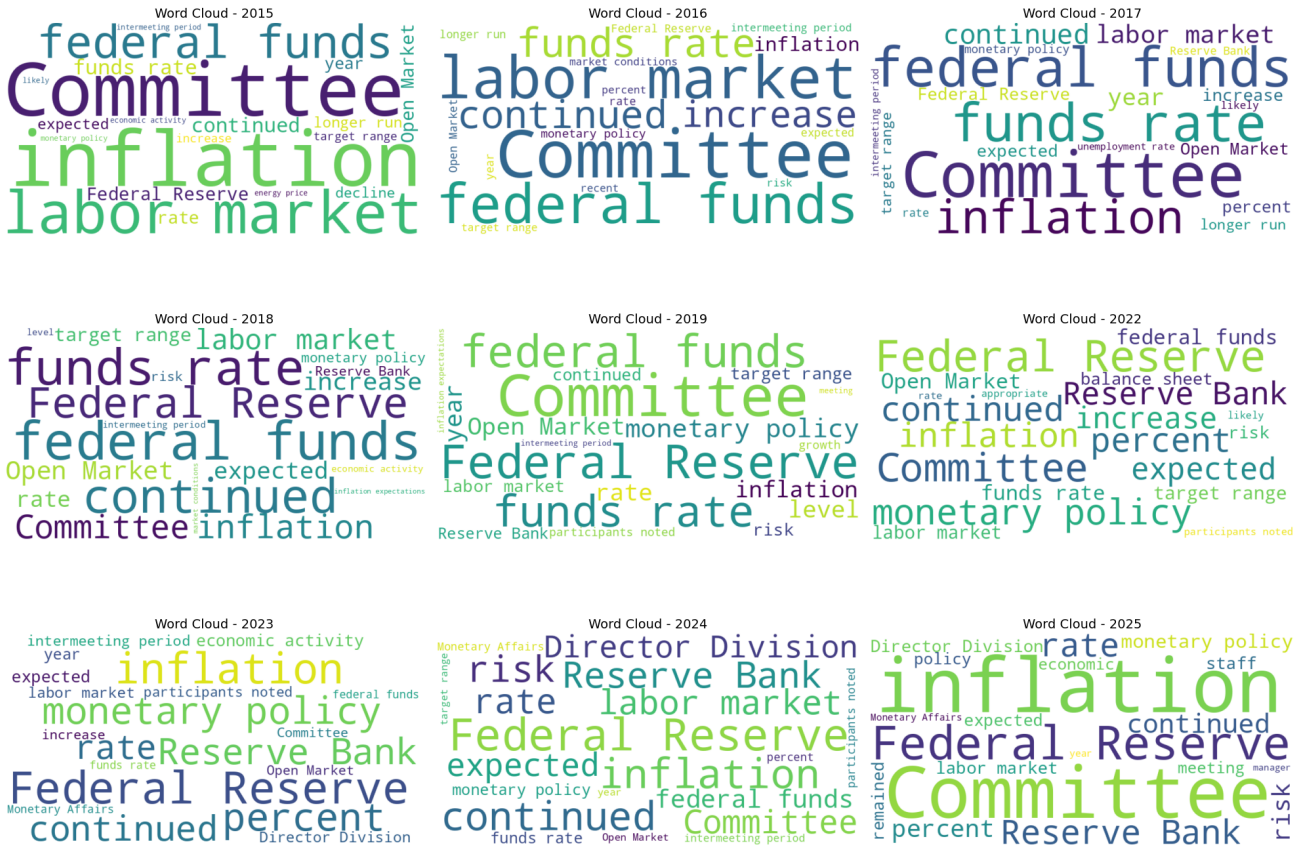
```
plt.title('Average Compound Sentiment Score per Year')
plt.xlabel('Year')
plt.ylabel('Average Compound Score')
plt.grid(True)
plt.tight_layout()
plt.show()
```



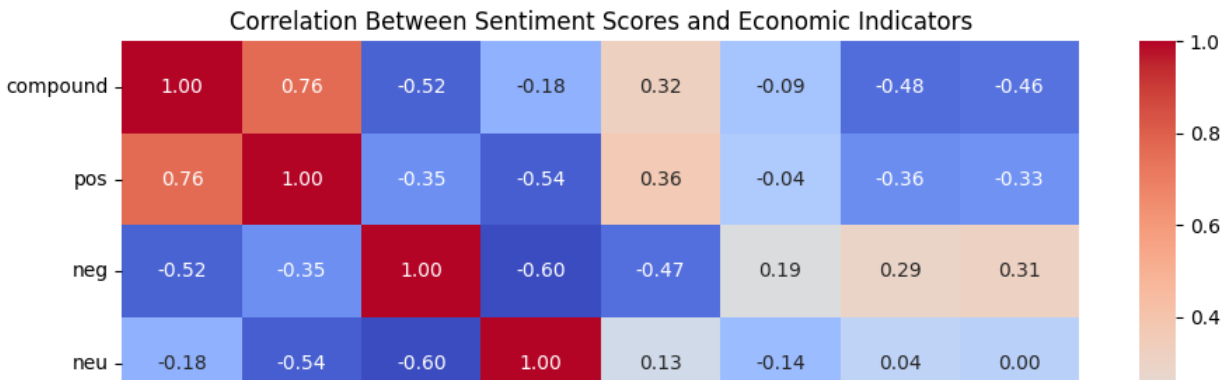
Wordclouds

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud

# Make sure you load the cleaned data that contains the 'cleaned_text' column
df_merged = pd.read_csv('/content/drive/My Drive/NLP/Assignment_3/fomc_transcripts_spacy_

# Generate word clouds for each year
```

```
years = sorted(df_merged['Year'].dropna().unique())
ncols = 3
nrows = (len(years) + ncols - 1) // ncols

plt.figure(figsize=(20, 5 * nrows))

for i, year in enumerate(years):
    # Use 'cleaned_text' column instead of 'cleaned_content'
    year_text = ' '.join(df_merged[df_merged['Year'] == year]['cleaned_text'].astype(str)
    wordcloud = WordCloud(width=800, height=400, max_words=20, background_color='white').

    plt.subplot(nrows, ncols, i + 1)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f'Word Cloud - {year}', fontsize=14)

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



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

# Load the dataframe with sentiment scores
df_merged = pd.read_csv('/content/drive/My Drive/NLP/Assignment_3/fomc_with_sentiment.csv

# Select relevant columns
correlation_df = df_merged[['compound', 'pos', 'neg', 'neu', 'Unemployment', 'GDP', 'CPI'

# Drop any rows with NaNs
correlation_df = correlation_df.dropna()

# Compute correlation
corr = correlation_df.corr()

# Plot heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Between Sentiment Scores and Economic Indicators")
plt.tight_layout()
plt.show()
```



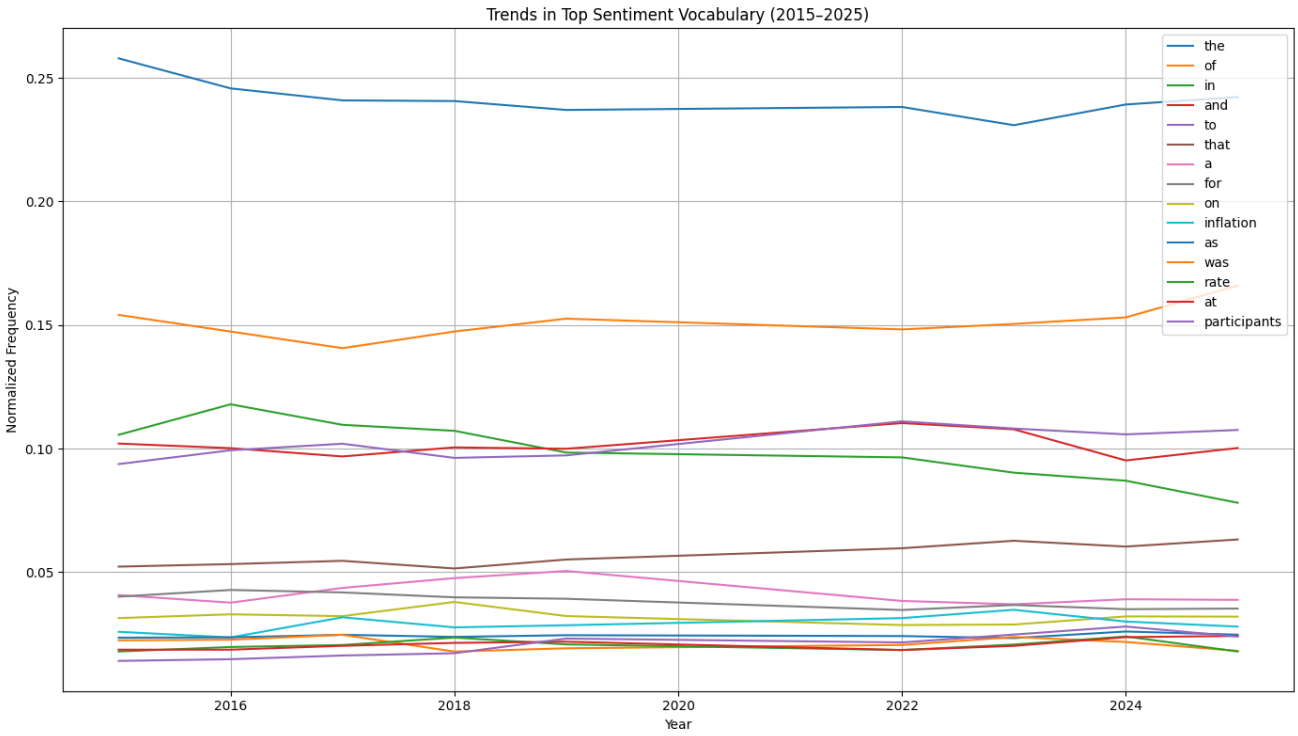
Analyze changes in sentiment vocabulary over time

```
from collections import Counter

# Dictionary to hold word frequencies per year
yearly_word_freq = {}

# Collect top words per year
for year in sorted(df_merged['Year'].dropna().unique()):
    texts = df_merged[df_merged['Year'] == year]['cleaned_text'].dropna().tolist()
    all_words = ' '.join(texts).split()
    word_counts = Counter(all_words)
    yearly_word_freq[year] = word_counts

# Aggregate across all years
total_word_counts = Counter()
for year_counts in yearly_word_freq.values():
    total_word_counts += year_counts

# Get top N words overall
top_words = [word for word, _ in total_word_counts.most_common(15)]
print("Top sentiment-related words:", top_words)

# Build a DataFrame with frequency of top words over years
word_trends = pd.DataFrame(index=sorted(yearly_word_freq.keys()), columns=top_words).fill

for year, counts in yearly_word_freq.items():
```

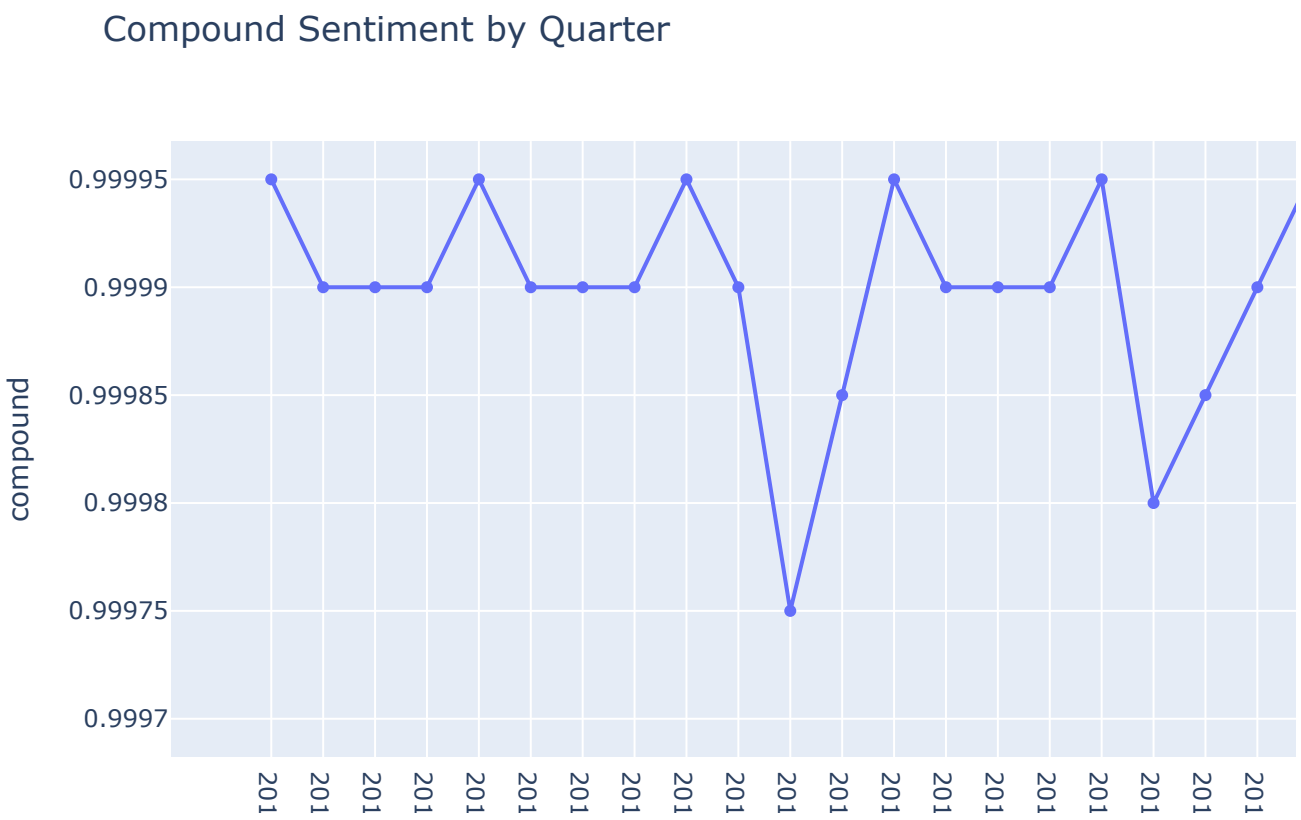
```
    for word in top_words:
        word_trends.loc[year, word] = counts.get(word, 0)

# Normalize by total words per year (optional)
word_trends = word_trends.div(word_trends.sum(axis=1), axis=0)
```

```
Top sentiment-related words: ['the', 'of', 'in', 'and', 'to', 'that', 'a', 'for', 'on
<ipython-input-106-f2e7672f4659>:23: FutureWarning:

Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will cha
```

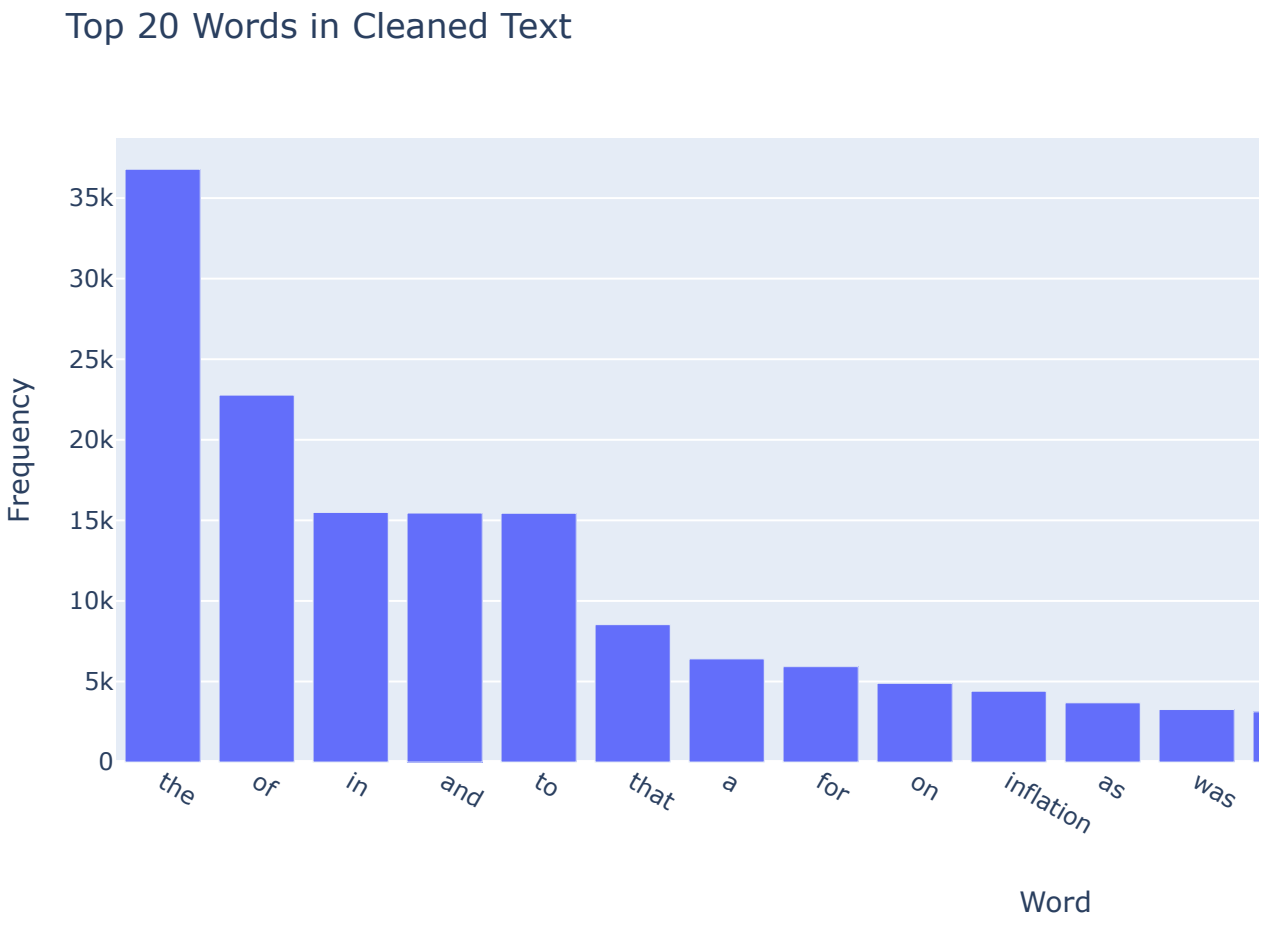
```
plt.figure(figsize=(14, 8))
for word in top_words:
    plt.plot(word_trends.index, word_trends[word], label=word)

plt.title('Trends in Top Sentiment Vocabulary (2015–2025)')
plt.xlabel('Year')
plt.ylabel('Normalized Frequency')
plt.legend(loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
# --- 7. Sentiment by Quarter ---
df_merged['Date'] = pd.to_datetime(df_merged['Date'])

df_merged['Quarter'] = df_merged['Date'].dt.to_period('Q').astype(str)
sentiment_by_quarter = df_merged.groupby('Quarter')[['pos', 'neg', 'neu', 'compound']].me
fig = px.line(sentiment_by_quarter, x='Quarter', y='compound', markers=True, title='Compo
fig.update_layout(height=500, width=1100)
fig.show()
```




```
# --- 9. Word Frequency (Top Words Overall) ---
from collections import Counter
import plotly.graph_objs as go

all_words = ' '.join(df_merged['cleaned_text'].fillna('')).split()
word_freq = Counter(all_words)
top_words = word_freq.most_common(20)

words, counts = zip(*top_words)
fig = go.Figure([go.Bar(x=words, y=counts)])
fig.update_layout(title='Top 20 Words in Cleaned Text', xaxis_title='Word', yaxis_title='
fig.show()

# --- Save final dataset ---
final_output_path = "/content/fomc_transcripts_with_sentiment.csv"
df_merged.to_csv(final_output_path, index=False)
print(f"\n✅ Final dataset with sentiment and plots saved to {final_output_path}")
```



```
✅ Final dataset with sentiment and plots saved to /content/fomc_transcripts_with_se
```

Classical NLP Approach - Topic Modeling

Clean and Tokenize text

```
# --- 1. Clean & Tokenize ---
import re
import pandas as pd
import spacy
from gensim.corpora import Dictionary
from gensim.models import LdaModel, CoherenceModel

# Load spaCy
try:
    nlp = spacy.load("en_core_web_sm")
except:
    import os
    os.system("python -m spacy download en_core_web_sm")
    nlp = spacy.load("en_core_web_sm")

# Cleaning
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

df_merged['cleaned_content'] = df_merged['Content'].fillna('').apply(clean_text)

# Tokenize and remove stopwords
def tokenize(text):
    doc = nlp(text)
    return [token.lemma_ for token in doc if token.is_alpha and not token.is_stop]

df_merged['Tokens'] = df_merged['cleaned_content'].apply(tokenize)
print("✅ Text cleaned and tokenized.")
```

```
✅ Text cleaned and tokenized.
```

Coherence Score Optimization

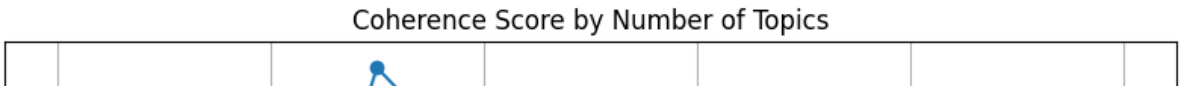
```
def find_optimal_topics_gensim(df_tokens, max_topics=12):
    dictionary = Dictionary(df_tokens)
    corpus = [dictionary.doc2bow(text) for text in df_tokens]
```

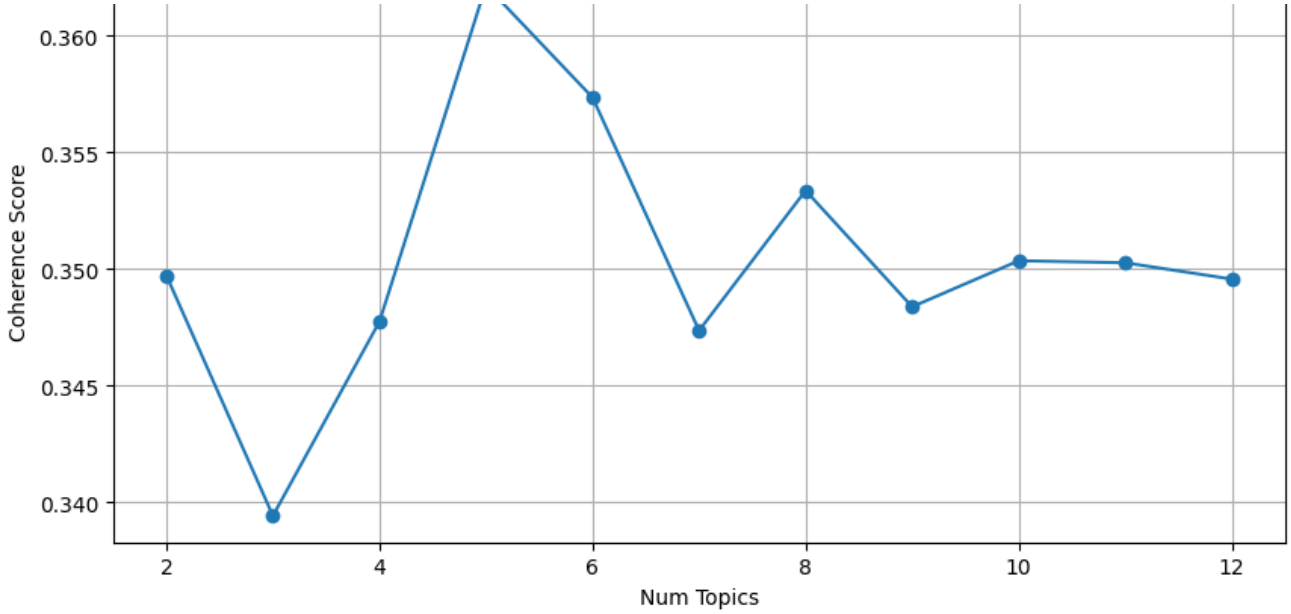
```
    coherence_scores = {}
    for k in range(2, max_topics + 1):
        lda_model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=k, random_stat
        cm = CoherenceModel(model=lda_model, texts=df_tokens, dictionary=dictionary, cohe
        coherence_scores[k] = cm.get_coherence()
        print(f"Topics: {k}, Coherence Score: {coherence_scores[k]:.4f}")

    # Plot scores
    import matplotlib.pyplot as plt
    plt.figure(figsize=(10, 5))
    plt.plot(list(coherence_scores.keys()), list(coherence_scores.values()), marker='o')
    plt.title("Coherence Score by Number of Topics")
    plt.xlabel("Num Topics"); plt.ylabel("Coherence Score")
    plt.grid(True)
    plt.show()

    best_k = max(coherence_scores, key=coherence_scores.get)
    return best_k, dictionary, corpus
```

Run this to find best k

```
best_k, dictionary, corpus = find_optimal_topics_gensim(df_merged['Tokens'])
```

```
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 2, Coherence Score: 0.3497
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 3, Coherence Score: 0.3394
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 4, Coherence Score: 0.3478
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 5, Coherence Score: 0.3622
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 6, Coherence Score: 0.3574
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 7, Coherence Score: 0.3474
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 8, Coherence Score: 0.3534
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 9, Coherence Score: 0.3484
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 10, Coherence Score: 0.3504
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider
Topics: 11, Coherence Score: 0.3503
Topics: 12, Coherence Score: 0.3496
```





Train Final LDA Model using (gensim)

```
from gensim.models import LdaModel

lda_model = LdaModel(
    corpus=corpus,
    id2word=dictionary,
    num_topics=best_k,
    random_state=42,
    passes=10,          # full corpus iterations
    iterations=100      # per-document iterations
)

print(f"✅ Final LDA model with {best_k} topics trained.")
```

```
✅ Final LDA model with 5 topics trained.
```

Re-evaluate Coherence Score After Training

```
from gensim.models.coherencemodel import CoherenceModel

# Evaluate the model using coherence score
coherence_model_lda = CoherenceModel(
    model=lda_model,
```

```
    texts=df_merged['Tokens'],
    dictionary=dictionary,
    coherence='c_v'
)
coherence_score = coherence_model_lda.get_coherence()
print(f"📈 Coherence Score (after passes): {coherence_score:.4f}")
```

```
📈 Coherence Score (after passes): 0.3559
```

Save the Trained Model

```
lda_model.save("/content/drive/My Drive/NLP/Assignment_3/final_lda_model.gensim")
dictionary.save("/content/drive/My Drive/NLP/Assignment_3/final_dictionary.dict")
print("✅ Final LDA model and dictionary saved.")
```

```
✅ Final LDA model and dictionary saved.
```

And to reload:

```
from gensim.models import LdaModel
from gensim.corpora import Dictionary

lda_model = LdaModel.load("/content/drive/My Drive/NLP/Assignment_3/final_lda_model.gensi
dictionary = Dictionary.load("/content/drive/My Drive/NLP/Assignment_3/final_dictionary.d
print("✅ Final LDA model and dictionary loaded.")
```

```
✅ Final LDA model and dictionary loaded.
```

Assign Topics to Each Document

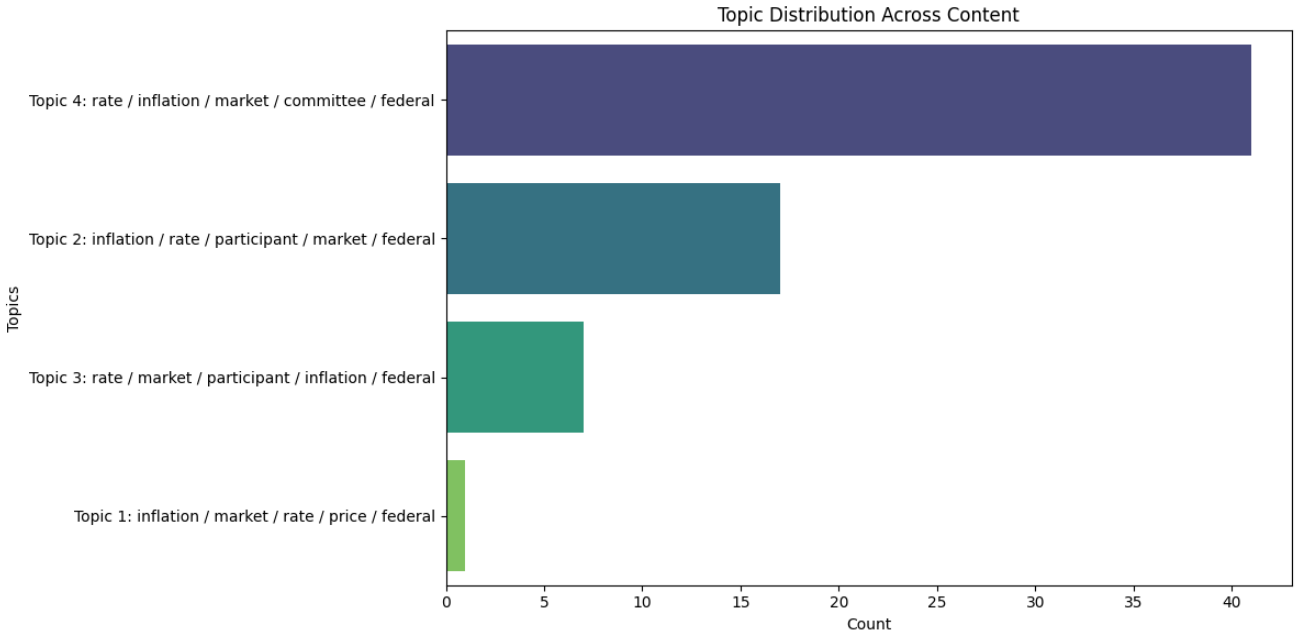
```
def assign_gensim_topics(df, corpus, lda_model):
    dominant_topics = []
    topic_keywords = {}

    for bow in corpus:
        topic_probs = lda_model.get_document_topics(bow, minimum_probability=0.0)
        dominant_topic = max(topic_probs, key=lambda x: x[1])[0]
        dominant_topics.append(dominant_topic)

    for topic_num in range(lda_model.num_topics):
        words = lda_model.show_topic(topic_num, topn=5)
        topic_keywords[topic_num] = " / ".join([w for w, _ in words])

    df['topics'] = [f"Topic {i}: {topic_keywords[i]}" for i in dominant_topics]
    return df, topic_keywords
```

```
df_merged, topic_map = assign_gensim_topics(df_merged, corpus, lda_model)
```

📊 Visualize Topic Distribution

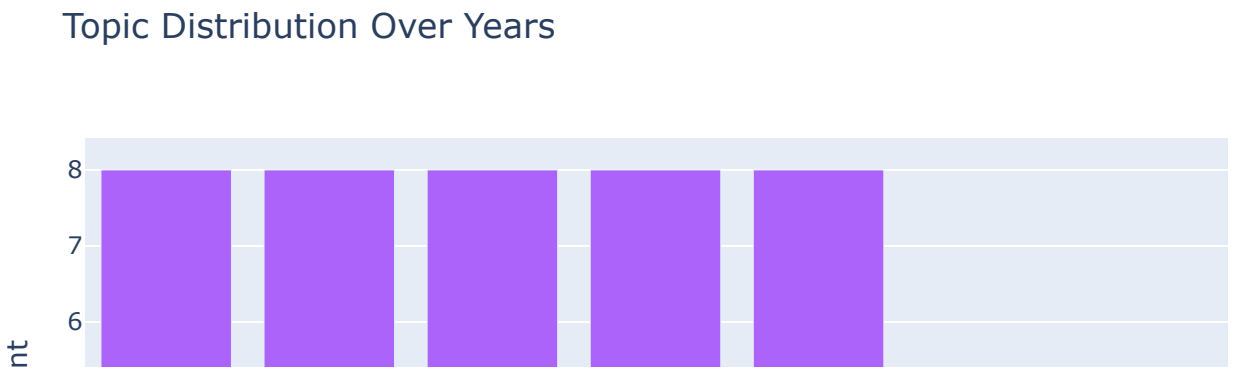
```
import seaborn as sns
import matplotlib.pyplot as plt

def plot_topic_distribution(df):
    plt.figure(figsize=(12, 6))
    sns.countplot(y=df['topics'], order=df['topics'].value_counts().index, palette="virid
    plt.title("Topic Distribution Across Content")
    plt.xlabel("Count"); plt.ylabel("Topics")
    plt.tight_layout()
    plt.show()

plot_topic_distribution(df_merged)
```

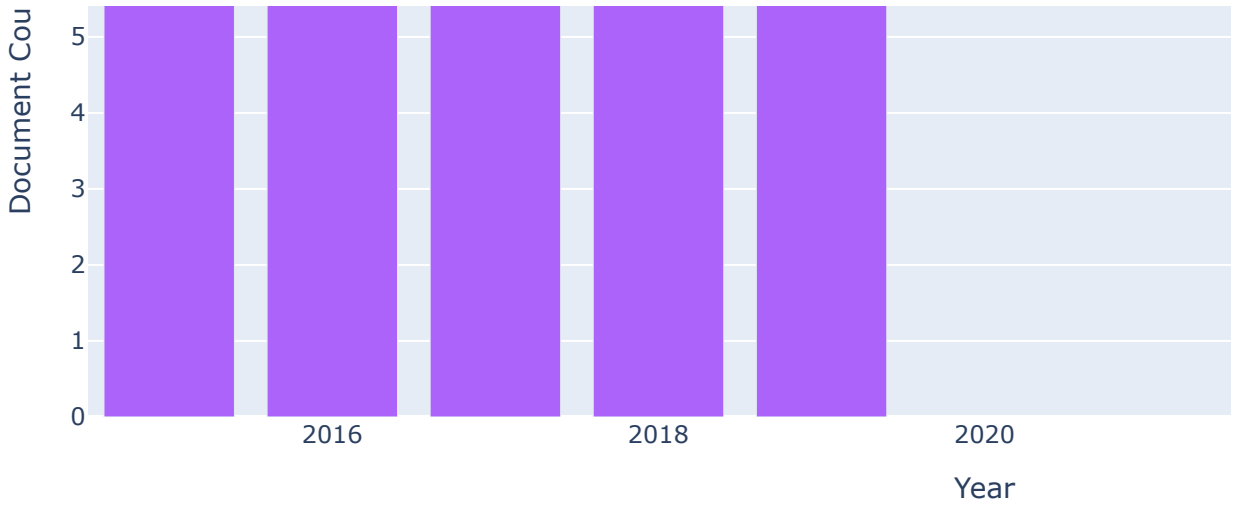
```
<ipython-input-119-5531f6baec09>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
```



Topic Trend Over Time (by Year)

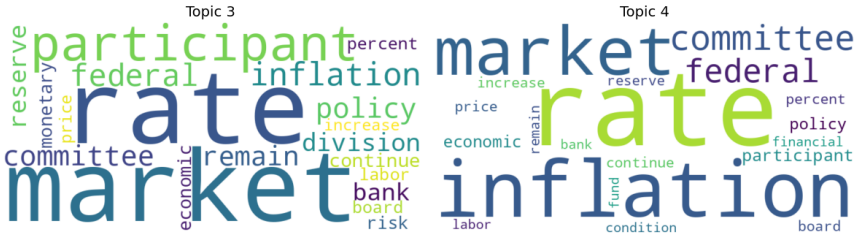
```
# Make sure 'Date' is datetime and 'Year' column exists
df_merged['Date'] = pd.to_datetime(df_merged['Date'])
df_merged['Year'] = df_merged['Date'].dt.year

# Count topic occurrences per year
topic_year_dist = df_merged.groupby(['Year', 'topics']).size().reset_index(name='Count')

# Pivot the data for stacked bar plot
topic_year_pivot = topic_year_dist.pivot(index='Year', columns='topics', values='Count').

# Plot using plotly
import plotly.express as px

fig = px.bar(topic_year_pivot,
             x=topic_year_pivot.index,
             y=topic_year_pivot.columns,
             title="Topic Distribution Over Years",
             labels={'value': 'Count', 'Year': 'Year'},
             height=500)

fig.update_layout(barmode='stack', xaxis_title="Year", yaxis_title="Document Count")
fig.show()
```





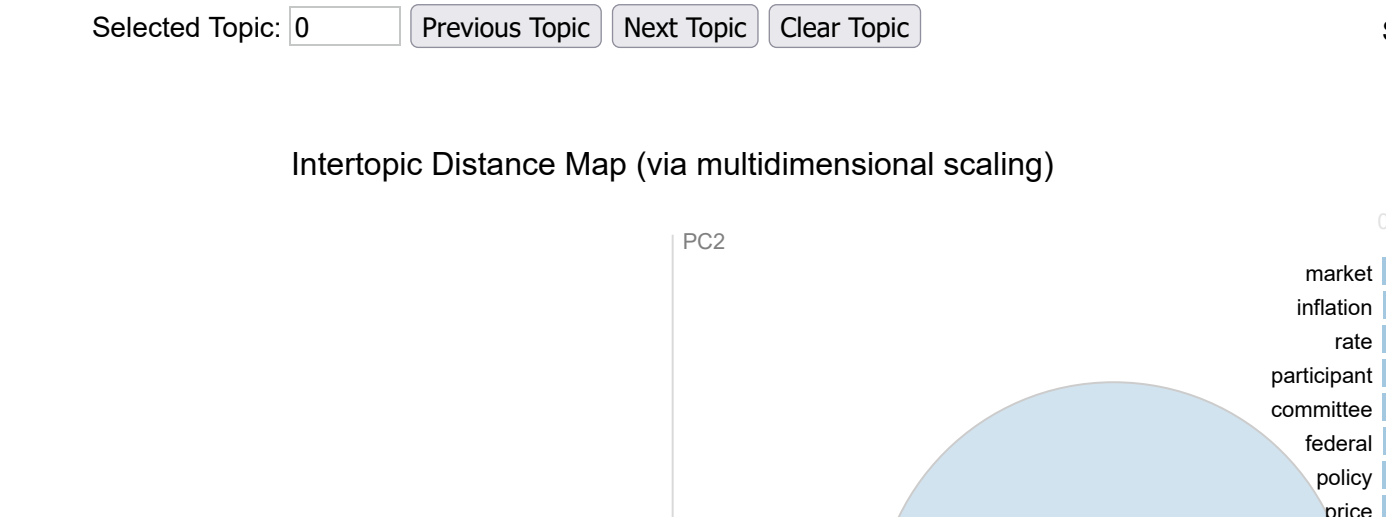
Word Cloud Per Topic

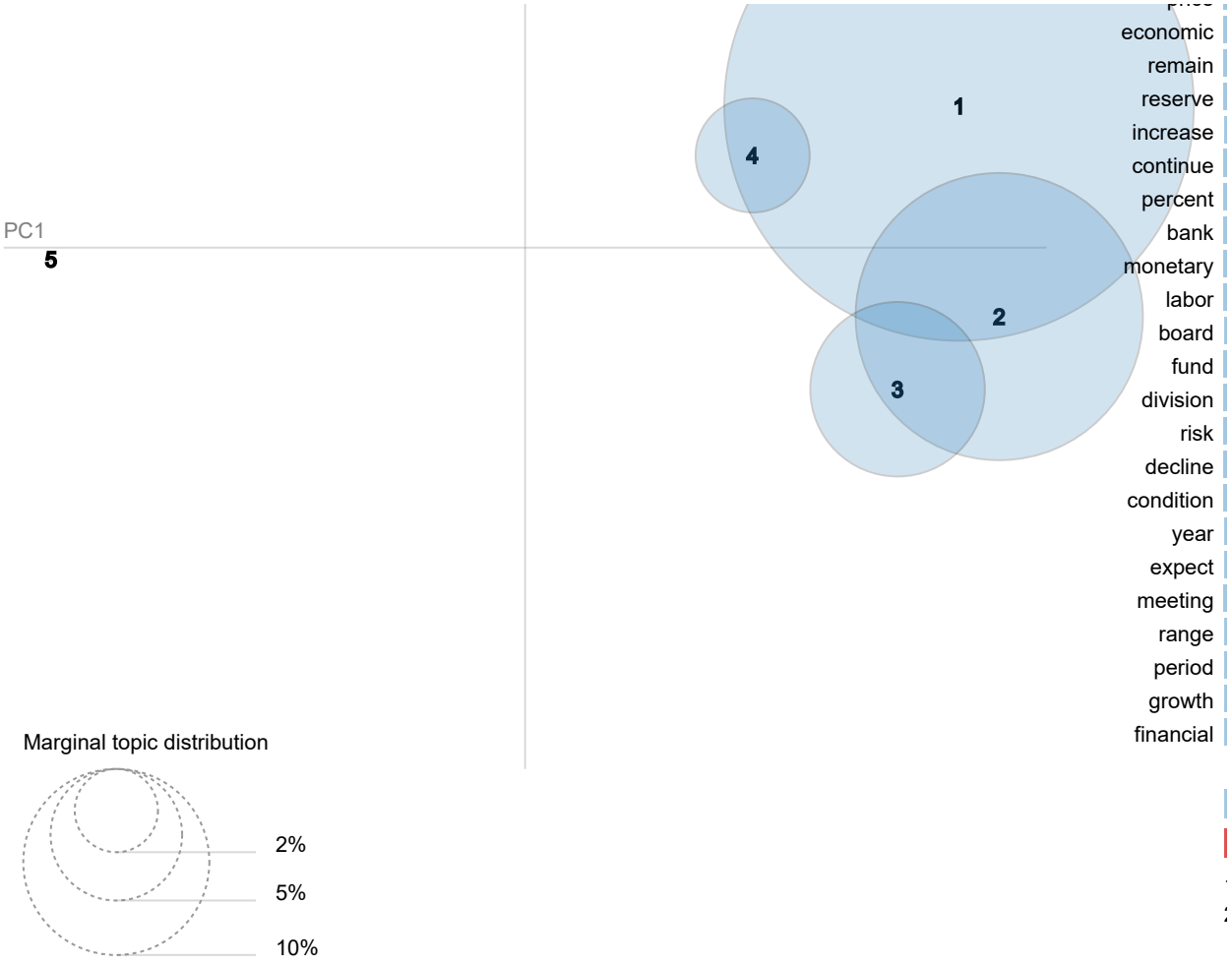
```
from wordcloud import WordCloud

def generate_wordclouds_for_topics(lda_model, num_words=20):
    cols = 3
    rows = int(np.ceil(lda_model.num_topics / cols))
    plt.figure(figsize=(6 * cols, 5 * rows))

    for topic_id in range(lda_model.num_topics):
        plt.subplot(rows, cols, topic_id + 1)
        topic_terms = lda_model.show_topic(topic_id, topn=num_words)
        wordcloud = WordCloud(width=800, height=400, background_color='white').generate_f
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
        plt.title(f"Topic {topic_id}", fontsize=14)

    plt.tight_layout()
    plt.show()

generate_wordclouds_for_topics(lda_model)
```



Interactive pyLDAvis

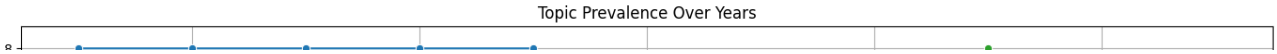
```
import pyLDAvis.gensim_models as gensimvis
import pyLDAvis

pyLDAvis.enable_notebook()
vis = gensimvis.prepare(lda_model, corpus, dictionary)
vis  # This will open the interactive panel inside the notebook
```

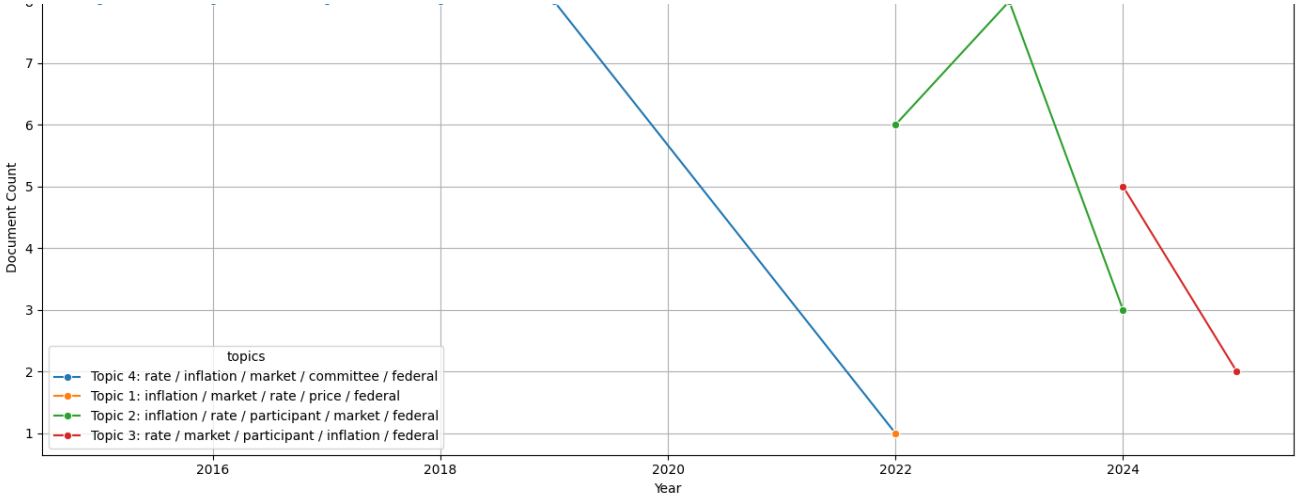




Topic Trends Over Time

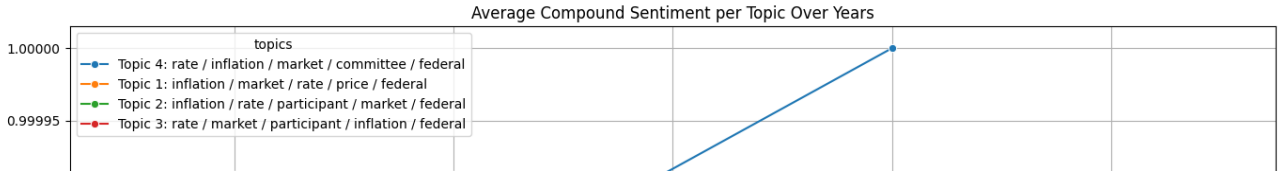
```
df_merged['Year'] = pd.to_datetime(df_merged['Date']).dt.year

topic_trend = df_merged.groupby(['Year', 'topics']).size().reset_index(name='Count')

plt.figure(figsize=(14, 6))
sns.lineplot(data=topic_trend, x='Year', y='Count', hue='topics', marker='o')
plt.title("Topic Prevalence Over Years")
plt.xlabel("Year")
plt.ylabel("Document Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```

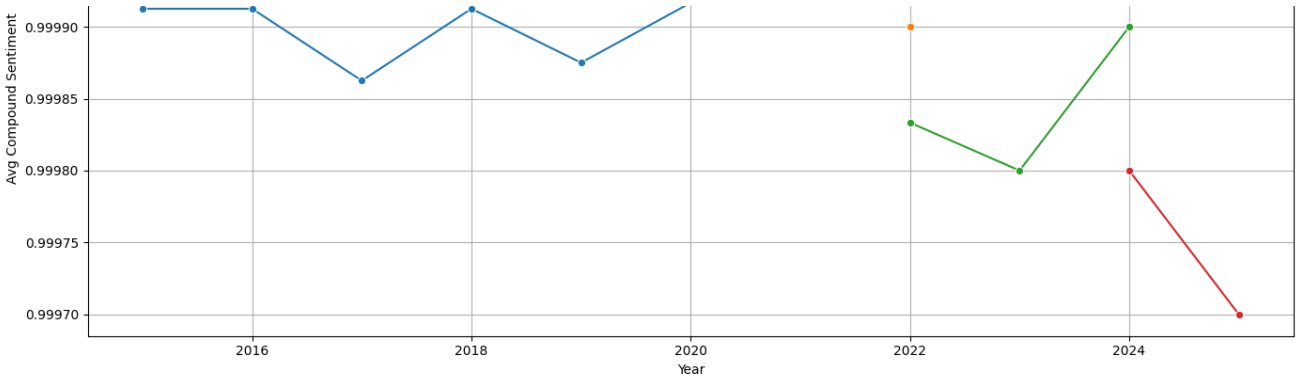




Overlay Sentiment with Topics

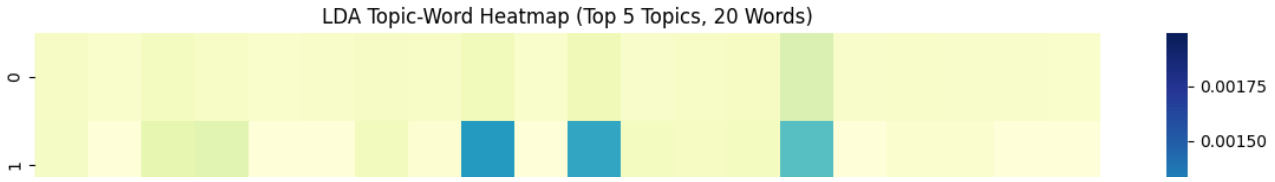
```
# Example: Average sentiment per topic per year
sentiment_overlay = df_merged.groupby(['Year', 'topics'])['compound'].mean().reset_index()

plt.figure(figsize=(14, 6))
sns.lineplot(data=sentiment_overlay, x='Year', y='compound', hue='topics', marker='o')
plt.title("Average Compound Sentiment per Topic Over Years")
plt.ylabel("Avg Compound Sentiment")
plt.grid(True)
plt.tight_layout()
plt.show()
```

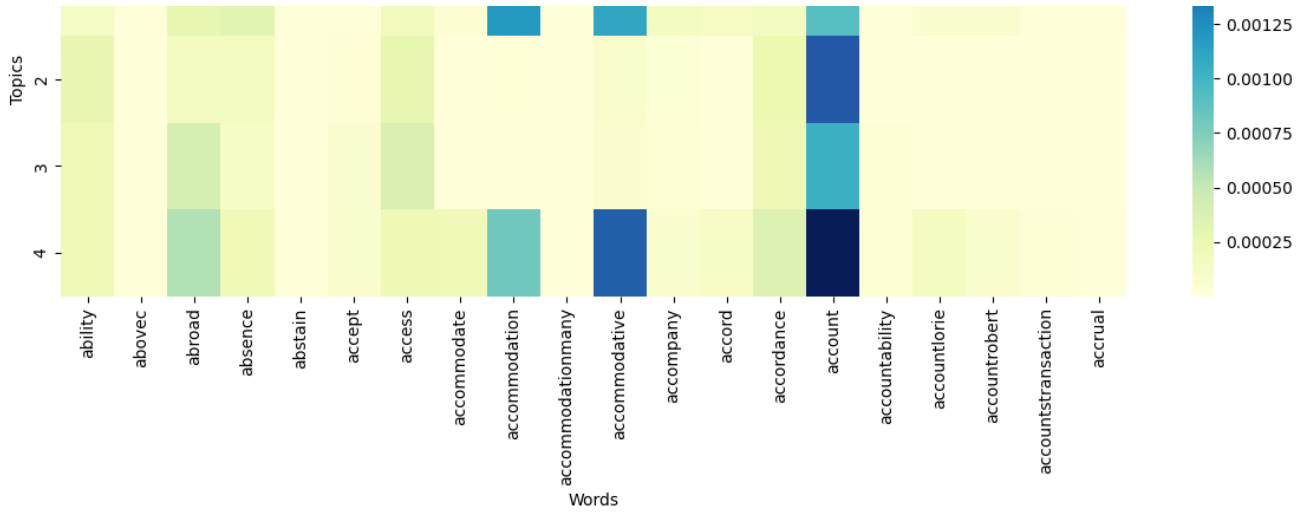




Topic–Word Heatmap

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

# Create matrix of topic-word weights
topic_word_df = pd.DataFrame(lda_model.get_topics(), columns=[dictionary[i] for i in rang

plt.figure(figsize=(12, 6))
sns.heatmap(topic_word_df.iloc[:5, :20], cmap='YlGnBu')
plt.title("LDA Topic-Word Heatmap (Top 5 Topics, 20 Words)")
plt.xlabel("Words")
plt.ylabel("Topics")
plt.tight_layout()
plt.show()
```




Topics
2
3
4
ability
abovec
abroad
absence
abstain
accept
access
accommodate
accommodation
accommodationmany
accommodative
accompany
accord
accordance
account
accountability
accountlorie
accountrobert
accountstransaction
accrual
Words
0.00125
0.00100
0.00075
0.00050
0.00025