NLP Analysis and Forecasting using the Federal Reserve Meeting Minutes

Background

Natural Language Processing (NLP) plays a crucial role in extracting insights from institutional communications, particularly in the financial sector. The Federal Reserve's meeting minutes contain valuable information about monetary policy, economic outlook, and market sentiment.

Financial analysts, economists, policymakers, and investors use NLP to analyze trends in central bank communications to anticipate policy changes and economic shifts.

Organizations utilize central bank communication analysis for: ● Market Predictions – Financial institutions monitor central bank language for signals about future interest rate changes and economic outlook.

● Policy Analysis – Economists track shifts in central bank rhetoric to understand evolving monetary policy frameworks.

● Sentiment Tracking – Tracking the sentiment in monetary policy statements helps gauge the central bank's confidence in economic conditions.

● Risk Assessment – Financial institutions use central bank communication analysis to identify potential economic risks and policy uncertainties.

● Economic Forecasting – Analyzing the relationship between central bank language and economic indicators helps improve economic forecasts.

Build a Dataset Using Federal Reserve Meeting Minutes

```
import pandas as pd
import requests
from bs4 import BeautifulSoup
import re
from tqdm import tqdm
import plotly.express as px

# Step 1: Define the date range with input
start_year = int(input("Enter the start year: "))
start_month = int(input("Enter the start month (1-12): "))
end_year = int(input("Enter the end year: "))
```

```
end_month = int(input("Enter the end month (1-12): "))
covid_years = [2020, 2021]

start_date = pd.Timestamp(year=start_year, month=start_month, day=1)
end_date = pd.Timestamp(year=end_year, month=end_month, day=1) + pd.offsets.MonthEnd(1)

# ✅ Step 2: Get current and recursive historical links
def get_current_links():
    url = "https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm"
    res = requests.get(url)
    soup = BeautifulSoup(res.text, 'html.parser')
    return [a['href'] for a in soup.find_all('a', href=True) if re.search(r'fomcminutes\d{8

def get_all_historical_links():
    base_url = "https://www.federalreserve.gov"
    archive_page = f"{base_url}/monetarypolicy/fomc_historical_year.htm"
    res = requests.get(archive_page)
    soup = BeautifulSoup(res.text, 'html.parser')

    year_pages = [a['href'] for a in soup.find_all('a', href=True) if 'fomchistorical' in a
    historical_links = []

    for yp in year_pages:
        year_url = yp if yp.startswith("http") else base_url + yp
        try:
            res_year = requests.get(year_url)
            res_year.raise_for_status()
            soup_year = BeautifulSoup(res_year.text, 'html.parser')
            links = [a['href'] for a in soup_year.find_all('a', href=True) if re.search(r'r
            historical_links.extend(links)
        except Exception as e:
            print(f"⚠️ Failed to process {year_url}: {e}")
            continue

    return historical_links

# ✅ Step 3: Get all links and scrape
current_links = get_current_links()
historical_links = get_all_historical_links()
all_links = current_links + historical_links

minutes_data = []

for link in tqdm(all_links, desc="Scraping FOMC Minutes"):
    full_url = link if link.startswith("http") else f"https://www.federalreserve.gov{link}"
    date_match = re.search(r'(\d{8})', link)

    if date_match:
        date = pd.to_datetime(date_match.group(1), format="%Y%m%d")
        if not (start_date <= date <= end_date) or date.year in covid_years:
            continue
```

```
        try:
            res = requests.get(full_url)
            res.raise_for_status()
            soup = BeautifulSoup(res.text, 'html.parser')
            content_div = soup.find('div', id='content')
            content_text = content_div.get_text(strip=True) if content_div else "CONTENT M

            minutes_data.append({
                "URL": full_url,
                "Date": date,
                "Year": date.year,
                "Month": date.month,
                "Day": date.day,
                "Content": content_text
            })

        except Exception as e:
            print(f"⚠️ Error fetching {full_url}: {e}")
            continue
```

```
Enter the start year: 2015
Enter the start month (1–12): 1
Enter the end year: 2025
Enter the end month (1–12): 12
Scraping FOMC Minutes: 100%|██████████| 139/139 [00:17<00:00,  7.74it/s]
```

```
# ✅ Step 4: Create DataFrame and sort
df = pd.DataFrame(minutes_data)
df['Date'] = pd.to_datetime(df['Date'])
df_transcripts = df.sort_values(by='Date', ascending=False).reset_index(drop=True)
```

```
df_transcripts.shape
```

```
(66, 6)
```

```
# ✅ Save to CSV
from google.colab import drive
drive.mount('/content/drive')

output_path = '/content/drive/My Drive/NLP/Assignment_3/fomc_transcripts_2015_2025_cleane
df_transcripts.to_csv(output_path, index=False)
print(f"✅ Saved sorted dataset to {output_path}")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
✅ Saved sorted dataset to /content/drive/My Drive/NLP/Assignment_3/fomc_transcripts_
```

```
df_transcripts.head()
```

	URL	Date	Year	Month	Day	Content
0	https://www.federalreserve.gov/monetarypolicy/...	2025-03-19	2025	3	19	HomeMonetary PolicyFederal Open Market Committ...
1	https://www.federalreserve.gov/monetarypolicy/...	2025-01-29	2025	1	29	HomeMonetary PolicyFederal Open Market Committ...
2	https://www.federalreserve.gov/	2024-12-18	2024	12	18	HomeMonetary PolicyFederal Open Market

```
df_sorted_asc = df.sort_values(by='Date', ascending=True).reset_index(drop=True)
df_sorted_asc.head(10)
```

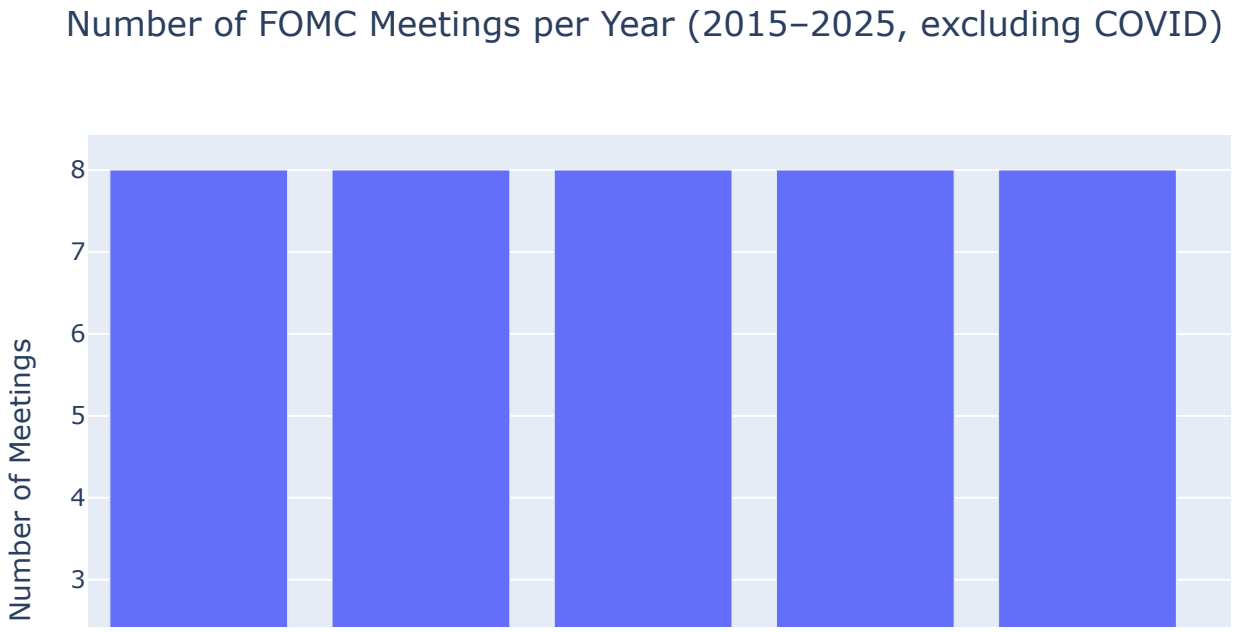
	URL	Date	Year	Month	Day	Content
0	https://www.federalreserve.gov/monetarypolicy/...	2015-01-28	2015	1	28	HomeMonetary PolicyFederal Open Market Committ...
1	https://www.federalreserve.gov/monetarypolicy/...	2015-03-18	2015	3	18	HomeMonetary PolicyFederal Open Market Committ...
2	https://www.federalreserve.gov/monetarypolicy/...	2015-04-29	2015	4	29	HomeMonetary PolicyFederal Open Market Committ...
3	https://www.federalreserve.gov/monetarypolicy/...	2015-06-17	2015	6	17	HomeMonetary PolicyFederal Open Market Committ...
4	https://www.federalreserve.gov/monetarypolicy/...	2015-07-29	2015	7	29	HomeMonetary PolicyFederal Open Market Committ...

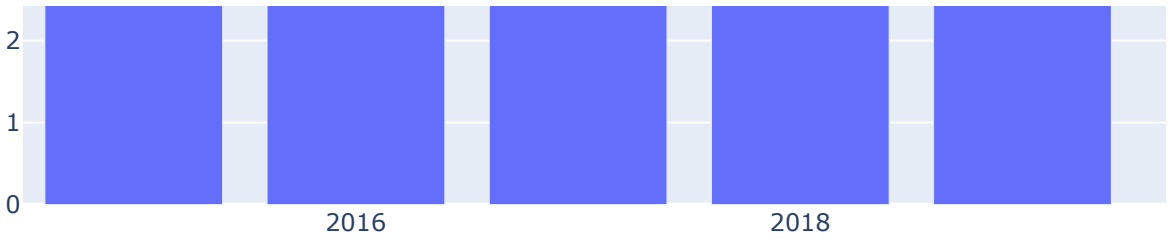
```
df_sorted_asc.tail(10)
```

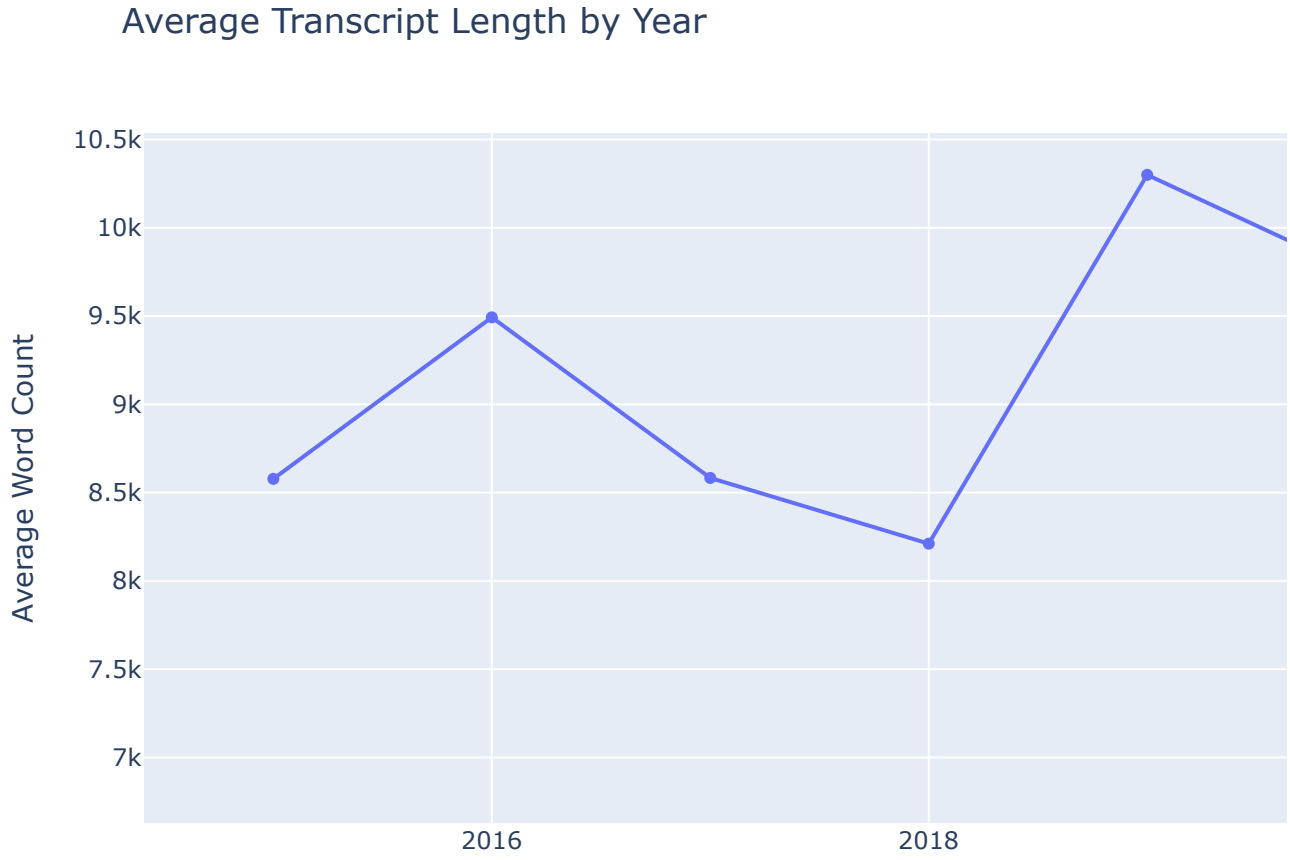
	URL	Date	Year	Month	Day	Content
56	https://www.federalreserve.gov/monetarypolicy/...	2024-01-31	2024	1	31	HomeMonetary PolicyFederal Open Market Committ...
57	https://www.federalreserve.gov/monetarypolicy/...	2024-03-20	2024	3	20	HomeMonetary PolicyFederal Open Market Committ...
58	https://www.federalreserve.gov/monetarypolicy/...	2024-05-01	2024	5	1	HomeMonetary PolicyFederal Open Market Committ...

59	https://www.federalreserve.gov/monetarypolicy/...	2024-06-12	2024	6	12	HomeMonetary PolicyFederal Open Market Committ...
60	https://www.federalreserve.gov/monetarypolicy/...	2024-07-31	2024	7	31	HomeMonetary PolicyFederal Open Market Committ...

```
# ✅ Visualizations using Plotly
df_transcripts['Content Length'] = df_transcripts['Content'].str.split().str.len()

year_counts = df_transcripts['Year'].value_counts().sort_index().reset_index()
year_counts.columns = ['Year', 'Count']
fig1 = px.bar(
    year_counts,
    x='Year', y='Count',
    labels={'Year': 'Year', 'Count': 'Number of Meetings'},
    title='Number of FOMC Meetings per Year (2015–2025, excluding COVID)'
)
fig1.show()

avg_lengths = df_transcripts.groupby('Year')['Content Length'].mean().reset_index()
fig2 = px.line(
    avg_lengths, x='Year', y='Content Length',
    markers=True,
    title='Average Transcript Length by Year',
    labels={'Content Length': 'Average Word Count'}
)
fig2.show()

print("\nSample preview:")
df_transcripts[['Date', 'Year', 'Content']].head(3)
```








```
!pip install fredapi
```

```
Requirement already satisfied: fredapi in /usr/local/lib/python3.11/dist-packages (0.
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/di
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (f
```

Sample preview:

	Date	Year	Content
0	2025-03-19	2025	HomeMonetary Policy Federal Open Market Committ...
1	2025-01-29	2025	HomeMonetary Policy Federal Open Market Committ...
2	2024-12-18	2024	HomeMonetary Policy Federal Open Market Committ...

```
from fredapi import Fred
import pandas as pd

# Initialize FRED API
fred = Fred(api_key='c1bf18b836872c769113d9189021caee')
```

```
series = {
    'GDP': 'A191RL1Q225SBEA',         # Real GDP % Change (Quarterly)
    'CPI': 'CPIAUCSL',                # Consumer Price Index (Monthly)
    'Unemployment': 'UNRATE',        # Unemployment Rate (Monthly)
    'FedFundsRate': 'FEDFUNDS'       # Federal Funds Rate (Monthly)
}

# Download the Data (2015–2025)
start = '2015-01-01'
end = '2025-12-31'

data = {}
for name, series_id in series.items():
    data[name] = fred.get_series(series_id, start, end)

df_econ = pd.concat(data, axis=1)
df_econ.index = pd.to_datetime(df_econ.index)

# Format & Resample to Monthly (fill GDP forward)
# Resample to monthly average
df_econ = df_econ.resample('M').mean()

# Forward-fill GDP since it's quarterly
df_econ['GDP'] = df_econ['GDP'].fillna(method='ffill')

# Reset index for saving
df_econ.reset_index(inplace=True)
df_econ.rename(columns={'index': 'date'}, inplace=True)

# Save to CSV
df_econ.to_csv("economic_indicators_2015_2025.csv", index=False)
print("Economic indicators saved to 'economic_indicators_2015_2025.csv'")
```

```
Economic indicators saved to 'economic_indicators_2015_2025.csv'
<ipython-input-52-a6ac003e3e34>:27: FutureWarning:

'M' is deprecated and will be removed in a future version, please use 'ME' instead.

<ipython-input-52-a6ac003e3e34>:30: FutureWarning:

Series.fillna with 'method' is deprecated and will raise in a future version. Use obj
```

```
# ✅ Save economic indicators
econ_path = "/content/drive/My Drive/NLP/Assignment_3/economic_indicators_2015_2025.csv"
df_econ.to_csv(econ_path, index=False)
print(f"✅ Economic indicators saved to {econ_path}")
```

```
✅ Economic indicators saved to /content/drive/My Drive/NLP/Assignment_3/economic_in
```

```
df_econ.sample(5)
```

	date	GDP	CPI	Unemployment	FedFundsRate
70	2020-11-30	4.4	260.911	6.7	0.09
23	2016-12-31	2.2	242.637	4.7	0.54
57	2019-10-31	2.8	257.155	3.6	1.83
78	2021-07-31	3.5	271.965	5.4	0.10
48	2019-01-31	2.5	252.561	4.0	2.40

```
print(df_transcripts.columns)
```

```
Index(['URL', 'Date', 'Year', 'Month', 'Day', 'Content', 'Content Length'], dtype='ob
```

```
# Load CSVs
df_transcripts = pd.read_csv('/content/drive/My Drive/NLP/Assignment_3/fomc_transcripts_2
df_econ = pd.read_csv('/content/drive/My Drive/NLP/Assignment_3/economic_indicators_2015_
```

Convert Date Columns and Align by Month

```
# Convert Meeting Date and date columns to datetime
# For transcripts
df_transcripts['Date'] = pd.to_datetime(df_transcripts['Date'], errors='coerce')

# Step 1: Read the CSV and check column types
df_econ = pd.read_csv('economic_indicators_2015_2025.csv')
df_econ['date'] = pd.to_datetime(df_econ['date'], errors='coerce')

# Create a 'Month' column in both DataFrames
# Align transcript dates to the first day of each month
df_transcripts['Month'] = df_transcripts['Date'].dt.to_period('M').dt.to_timestamp()

# Step 2: Align by month
df_econ['Month'] = df_econ['date'].dt.to_period('M').dt.to_timestamp()

# Optional: Preview
df_econ[['date', 'Month']].head()
```

	date	Month
0	2015-01-31	2015-01-01
1	2015-02-28	2015-02-01
2	2015-03-31	2015-03-01

3	2015-04-30	2015-04-01
4	2015-05-31	2015-05-01

```
# Merge on the 'Month' column
df_merged = pd.merge(df_transcripts, df_econ, on='Month', how='left')

# Verify if the column exists with the correct name:
print(df_merged.columns)  # Check if 'Content' is present
```

```
Index(['URL', 'Date', 'Year', 'Month', 'Day', 'Content', 'date', 'GDP', 'CPI',
       'Unemployment', 'FedFundsRate'],
      dtype='object')
```

```
# Save the final aligned dataset
df_merged.to_csv('/content/drive/My Drive/NLP/Assignment_3/merged_fomc_with_indicators.cs
print("✅ Final merged dataset saved.")
```

```
✅ Final merged dataset saved.
```

```
df_merged.head()
```

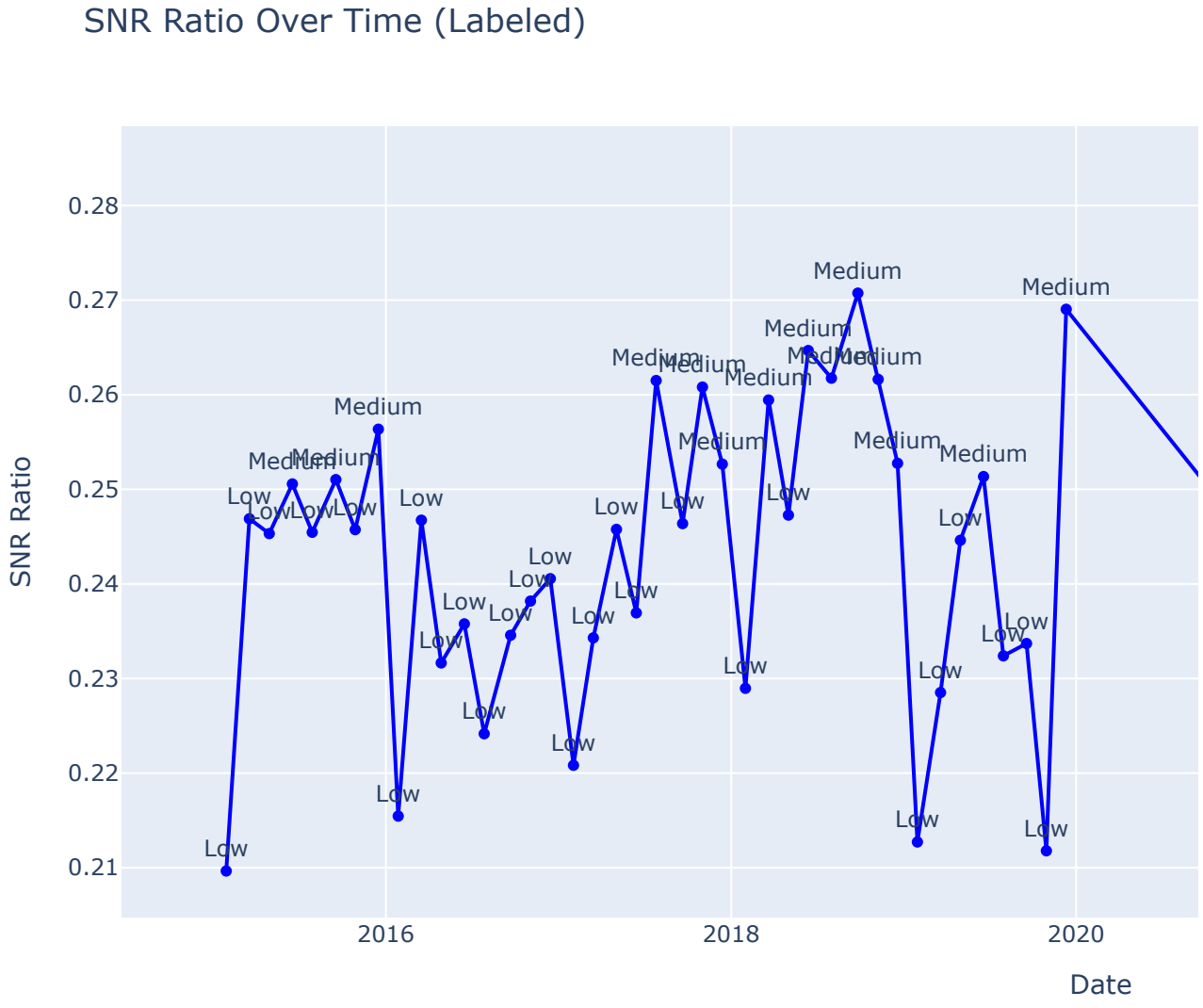
	URL	Date	Year	Month	Day	Content	date	G
0	https://www.federalreserve.gov/monetarypolicy/...	2025-03-19	2025	2025-03-01	19	HomeMonetary PolicyFederal Open Market Committ...	2025-03-31	-(
1	https://www.federalreserve.gov/monetarypolicy/...	2025-01-29	2025	2025-01-01	29	HomeMonetary PolicyFederal Open Market Committ...	2025-01-31	-(

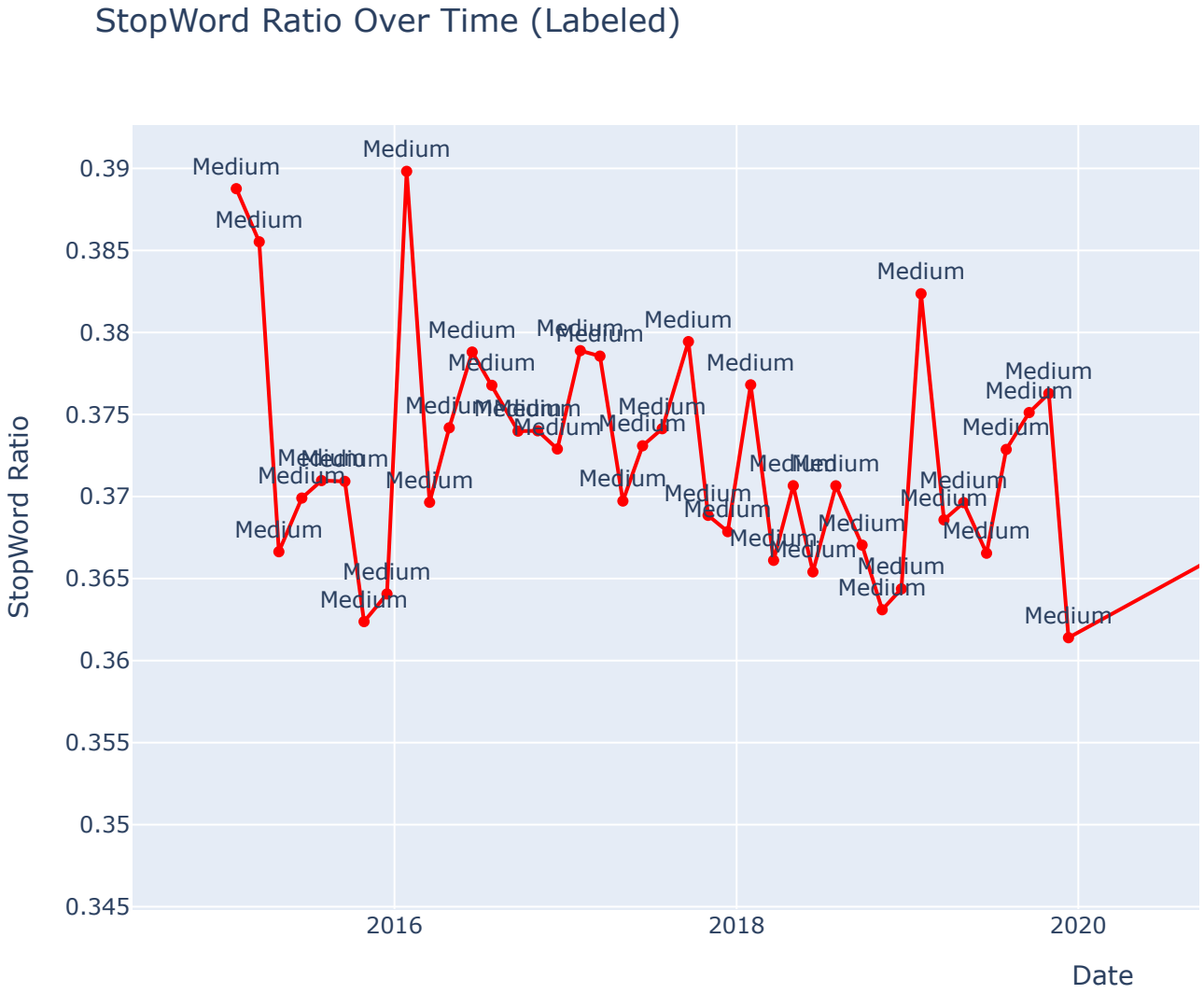
NLP + Quality Analysis before Cleaning

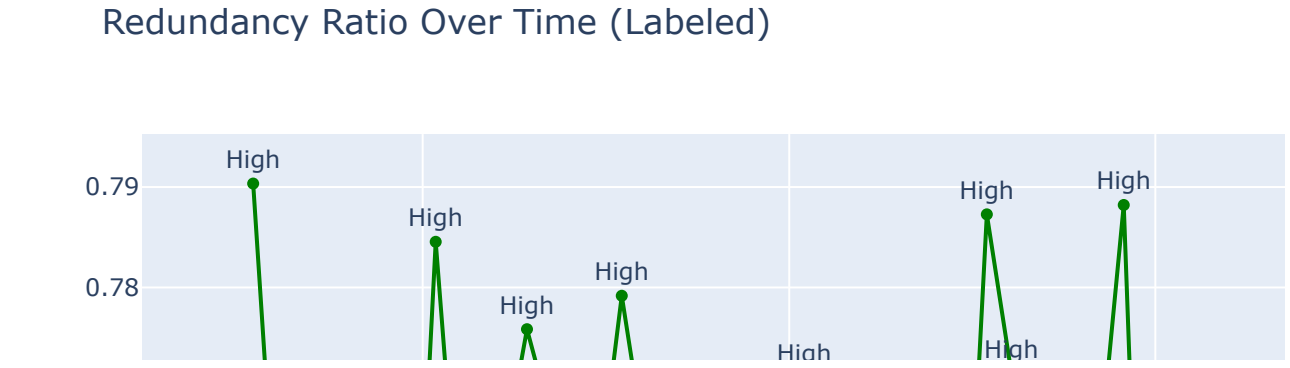
```
# --- 3. Noise + Quality Metric Analysis (Pre-cleaning) ---
import numpy as np
import string
from wordcloud import STOPWORDS
import plotly.graph_objs as go
```

```
# 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
    ))
    fig.update_layout(
        title=f"{metric.replace('_', ' ')} Over Time (Labeled)",
        xaxis_title='Date',
        yaxis_title=metric.replace('_', ' '),
        height=600,
        width=1200
    )
    fig.show()
```




StopWord Ratio Over Time (Labeled)
StopWord Ratio
0.39
0.385
0.38
0.375
0.37
0.365
0.36
0.355
0.35
0.345
2016
2018
2020
Date
Medium


Redundancy Ratio Over Time (Labeled)
0.79
0.78
High