```
from pathlib import Path
Path("fed_data").mkdir(parents=True, exist_ok=True)
```

```
!pip install PyPDF2
!pip install transformers torch scikit-learn
!pip install FedTools
!pip install nltk
```

```
Requirement already satisfied: PyPDF2 in /usr/local/lib/python3.10/dist-packages (3.0.1)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.44.2)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.4.1+cu121)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformer
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.20,>=0.19 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.5)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.6.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->trans
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformer
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformer
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Requirement already satisfied: FedTools in /usr/local/lib/python3.10/dist-packages (0.0.7)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from FedTools) (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from FedTools) (2.2.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from FedTools) (2.32.3)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from FedTools) (4.12.3)
Requirement already satisfied: fake-useragent in /usr/local/lib/python3.10/dist-packages (from FedTools) (1.5.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from FedTools) (2.8.2)
Requirement already satisfied: pytz in /usr/local/lib/python3.10/dist-packages (from FedTools) (2024.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from FedTools) (1.16.0)
Requirement already satisfied: soupsieve in /usr/local/lib/python3.10/dist-packages (from FedTools) (2.6)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->FedTools) (2024.2
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->FedTo
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->FedTools) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->FedTools) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->FedTools) (
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.5)
```

```
from FedTools import MonetaryPolicyCommittee

dataset = MonetaryPolicyCommittee().find_statements()
```

```
Constructing links between 1994 and 2024
Extracting the past 200 FOMC Statements.
Retrieving articles.
.....................................................................................................
```

```
dataset.index.values
```

```
 '2011-08-22T00:00:00.000000000', '2011-08-09T00:00:00.000000000',
 '2011-09-21T00:00:00.000000000', '2011-11-02T00:00:00.000000000',
 '2011-12-13T00:00:00.000000000', '2012-01-25T00:00:00.000000000',
 '2012-03-13T00:00:00.000000000', '2012-04-25T00:00:00.000000000',
 '2012-06-20T00:00:00.000000000', '2012-08-01T00:00:00.000000000',
 '2012-09-13T00:00:00.000000000', '2012-10-24T00:00:00.000000000',
 '2012-12-12T00:00:00.000000000', '2013-01-30T00:00:00.000000000',
 '2013-03-20T00:00:00.000000000', '2013-05-01T00:00:00.000000000',
 '2013-06-19T00:00:00.000000000', '2013-07-31T00:00:00.000000000',
 '2013-09-18T00:00:00.000000000', '2013-10-30T00:00:00.000000000',
 '2013-12-18T00:00:00.000000000', '2014-01-29T00:00:00.000000000',
 '2014-03-19T00:00:00.000000000', '2014-04-30T00:00:00.000000000',
 '2014-06-18T00:00:00.000000000', '2014-07-30T00:00:00.000000000',
 '2014-09-17T00:00:00.000000000', '2014-10-29T00:00:00.000000000',
 '2014-12-17T00:00:00.000000000', '2019-01-30T00:00:00.000000000',
 '2019-03-20T00:00:00.000000000', '2019-05-01T00:00:00.000000000',
 '2019-06-19T00:00:00.000000000', '2019-07-31T00:00:00.000000000',
 '2019-09-18T00:00:00.000000000', '2019-10-11T00:00:00.000000000',
 '2019-10-30T00:00:00.000000000', '2019-12-11T00:00:00.000000000',
 '2020-01-29T00:00:00.000000000', '2020-03-03T00:00:00.000000000',
 '2020-03-15T00:00:00.000000000', '2020-03-23T00:00:00.000000000',
 '2020-03-31T00:00:00.000000000', '2020-04-29T00:00:00.000000000',
 '2020-06-10T00:00:00.000000000', '2020-07-29T00:00:00.000000000',
 '2020-08-27T00:00:00.000000000', '2020-09-16T00:00:00.000000000',
 '2020-11-05T00:00:00.000000000', '2020-12-16T00:00:00.000000000',
 '2021-01-27T00:00:00.000000000', '2021-03-17T00:00:00.000000000',
 '2021-04-28T00:00:00.000000000', '2021-06-16T00:00:00.000000000',
 '2021-07-28T00:00:00.000000000', '2021-09-22T00:00:00.000000000',
 '2021-11-03T00:00:00.000000000', '2021-12-15T00:00:00.000000000',
 '2022-01-26T00:00:00.000000000', '2022-03-16T00:00:00.000000000',
 '2022-05-04T00:00:00.000000000', '2022-06-15T00:00:00.000000000',
 '2022-07-27T00:00:00.000000000', '2022-09-21T00:00:00.000000000',
 '2022-11-02T00:00:00.000000000', '2022-12-14T00:00:00.000000000',
 '2023-02-01T00:00:00.000000000', '2023-03-22T00:00:00.000000000',
 '2023-05-03T00:00:00.000000000', '2023-06-14T00:00:00.000000000',
 '2023-07-26T00:00:00.000000000', '2023-09-20T00:00:00.000000000',
 '2023-11-01T00:00:00.000000000', '2023-12-13T00:00:00.000000000',
 '2024-01-31T00:00:00.000000000', '2024-03-20T00:00:00.000000000',
 '2024-05-01T00:00:00.000000000', '2024-06-12T00:00:00.000000000',
 '2024-07-31T00:00:00.000000000', '2024-09-18T00:00:00.000000000'],
dtype='datetime64[ns]')
```

```
[date.replace('-', '') for date in fomc_dates]
```

```
'20140319',
'20140430',
'20140618',
'20140730',
'20140917',
'20141029',
'20141217',
'20150128',
'20150318',
'20150429',
'20150617',
'20150729',
'20150917',
'20151028',
'20151216',
'20160127',
'20160316',
```

```
        '20191030',
        '20191211',
        '20200129',
        '20200303',
        '20200315',
        '20200323',
        '20200429',
        '20200610',
        '20200729',
        '20200916',
        '20201105',
        '20201216',
```

```
fomc_dates = ['20110126',
 '20110315',
 '20110427',
 '20110622',
 '20110801',
 '20110809',
 '20110921',
 '20111102',
 '20111128',
 '20111213',
 '20120125',
 '20120313',
 '20120425',
 '20120620',
 '20120801',
 '20120913',
 '20121023',
 '20121024',
 '20121212',
 '20130130',
 '20130320',
 '20130501',
 '20130619',
 '20130731',
 '20130918',
 '20131016',
 '20131030',
 '20131218',
 '20140129',
 '20140304',
 '20140319',
 '20140430',
 '20140618',
 '20140730',
 '20140917',
 '20141029',
 '20141217',
 '20150128',
 '20150318',
 '20150429',
 '20150617',
 '20150729',
 '20150917',
 '20151028',
 '20151216',
 '20160127',
 '20160316',
 '20160427',
 '20160615',
 '20160727',
 '20160921',
 '20161102',
 '20161214',
 '20170201',
 '20170315',
 '20170503',
 '20170614',
 '20170726',
 '20170920',
 '20171101',
 '20171213',
 '20180131',
 '20180321',
 '20180502',
 '20180613',
 '20180801',
 '20180926',
 '20181108',
 '20181219',
 '20190130',
 '20190320',
 '20190501',
 '20190619',
 '20190731',
 '20190918',
 '20191004',
 '20191030',
 '20191211',
 '20200129',
 '20200303',
 '20200315',
 '20200323',
```

```
 '20200429',
 '20200610',
 '20200729',
 '20200916',
 '20201105',
 '20201216',
 '20210127',
 '20210317',
 '20210428',
 '20210616',
 '20210728',
 '20210922',
 '20211103',
 '20211215',
 '20220126',
 '20220316',
 '20220504',
 '20220615',
 '20220727',
 '20220921',
 '20221102',
 '20221214',
 '20230201',
 '20230322',
 '20230503',
 '20230614',
 '20230726',
 '20230920',
 '20231101',
 '20231213',
 '20240131',
 '20240320',
 '20240501',
 '20240612',
 '20240731',
 '20240918']
```

Data Acquisition

```
import requests
from bs4 import BeautifulSoup

def fetch_fed_press_release(url):
    """Fetch and parse the Federal Reserve press release from the given URL."""
    # Send a GET request to the URL
    response = requests.get(url)

    # Check if the request was successful
    if response.status_code == 200:
        # Parse the HTML content
        soup = BeautifulSoup(response.content, 'html.parser')

        # Find the main content section; inspect the HTML to find the correct selector
        # In this case, just look at <div id="article"> and its <p> children.
        main_content = soup.find("div", id="article")

        if main_content:
            # Extract text from each paragraph within the main content
            paragraphs = main_content.find_all('p')
            content_text = "\n".join([p.get_text(strip=True) for p in paragraphs])
            return content_text.strip()
        else:
            return ""
    else:
        return ""

# # URL of the FOMC press release
# url = "https://www.federalreserve.gov/newsevents/pressreleases/monetary20110126a.htm"

# # Fetch and print the Federal Reserve press release content
# press_release_content = fetch_fed_press_release(url)
# print(press_release_content)
```

```
January 26, 2011
For immediate releaseShare
Information received since the Federal Open Market Committee met in December confirms that the economic recovery is cont
Consistent with its statutory mandate, the Committee seeks to foster maximum employment and price stability. Currently,
To promote a stronger pace of economic recovery and to help ensure that inflation, over time, is at levels consistent wi
```

```
The Committee will maintain the target range for the federal funds rate at 0 to 1/4 percent and continues to anticipate
The Committee will continue to monitor the economic outlook and financial developments and will employ its policy tools
Voting for the FOMC monetary policy action were: Ben S. Bernanke, Chairman; William C. Dudley, Vice Chairman; Elizabeth
```

```
import pandas as pd
import pickle
import os
import PyPDF2

def read_pdf(file_path):
    # Open the PDF file
    with open(file_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        text = ""

        # Iterate through each page
        for page in reader.pages:
            text += page.extract_text()  # Extract text from each page

    return text

statementurl = 'https://www.federalreserve.gov/newsevents/pressreleases/monetary'#20180321a1.pdf
pressurl = 'https://www.federalreserve.gov/mediacenter/files/FOMCpresconf'#20180321.pdf
minutesurl = 'https://www.federalreserve.gov/monetarypolicy/files/fomcminutes'#20180321.pdf
press_content = {'date':[], 'press':[]}
statement_content = {'date':[], 'statement':[]}
minutes_content = {'date':[], 'minutes':[]}

for dateStr in fomc_dates:
  try:
    pdf_text_statement = fetch_fed_press_release(f"{statementurl}{dateStr}a.htm")
    if pdf_text_statement == "":
      continue
    statement_content['date'].append(dateStr)
    statement_content['statement'].append(pdf_text_statement)
    print(f"Processing FOMC Statement at {dateStr}")
  except:
    print(f"{statementurl}{dateStr}a1.htm")
  try:
    os.system(f"wget {pressurl}{dateStr}.pdf")
    press_pdf_file_path = f'FOMCpresconf{dateStr}.pdf'  # Update to your PDF path
    pdf_text_press = read_pdf(press_pdf_file_path)
    press_content['date'].append(dateStr)
    press_content['press'].append(pdf_text_press)
    print(f"Processing FOMC Press at {dateStr}")
  except:
    print(f"{pressurl}{dateStr}.pdf")
  try:
    os.system(f"wget {minutesurl}{dateStr}.pdf")
    minutes_pdf_file_path = f'fomcminutes{dateStr}.pdf'  # Update to your PDF path
    pdf_text_minutes = read_pdf(minutes_pdf_file_path)
    minutes_content['date'].append(dateStr)
    minutes_content['minutes'].append(pdf_text_minutes)
    print(f"Processing FOMC Minutes at {dateStr}")
  except:
    print(f"{minutesurl}{dateStr}.pdf")

press_content = pd.DataFrame(press_content)
statement_content = pd.DataFrame(statement_content)
minutes_content = pd.DataFrame(minutes_content)

with open("fed_data/press.pkl", "wb") as f:
    pickle.dump(press_content, f)
with open("fed_data/statement.pkl", "wb") as f:
    pickle.dump(statement_content, f)
with open("fed_data/minutes.pkl", "wb") as f:
    pickle.dump(minutes_content, f)
```

```
Processing FOMC Press at 20221214
Processing FOMC Minutes at 20221214
Processing FOMC Statement at 20230201
Processing FOMC Press at 20230201
Processing FOMC Minutes at 20230201
Processing FOMC Statement at 20230322
Processing FOMC Press at 20230322
Processing FOMC Minutes at 20230322
Processing FOMC Statement at 20230503
Processing FOMC Press at 20230503
Processing FOMC Minutes at 20230503
Processing FOMC Statement at 20230614
Processing FOMC Press at 20230614
Processing FOMC Minutes at 20230614
Processing FOMC Statement at 20230726
Processing FOMC Press at 20230726
Processing FOMC Minutes at 20230726
Processing FOMC Statement at 20230920
Processing FOMC Press at 20230920
Processing FOMC Minutes at 20230920
Processing FOMC Statement at 20231101
Processing FOMC Press at 20231101
Processing FOMC Minutes at 20231101
Processing FOMC Statement at 20231213
Processing FOMC Press at 20231213
Processing FOMC Minutes at 20231213
Processing FOMC Statement at 20240131
Processing FOMC Press at 20240131
Processing FOMC Minutes at 20240131
Processing FOMC Statement at 20240320
Processing FOMC Press at 20240320
Processing FOMC Minutes at 20240320
Processing FOMC Statement at 20240501
Processing FOMC Press at 20240501
Processing FOMC Minutes at 20240501
Processing FOMC Statement at 20240612
Processing FOMC Press at 20240612
Processing FOMC Minutes at 20240612
Processing FOMC Statement at 20240731
Processing FOMC Press at 20240731
Processing FOMC Minutes at 20240731
Processing FOMC Statement at 20240918
Processing FOMC Press at 20240918
Processing FOMC Minutes at 20240918
```

```
with open("fed_data/press.pkl", "wb") as f:
    pickle.dump(press_content, f)
with open("fed_data/statements.pkl", "wb") as f:
    pickle.dump(statement_content, f)
with open("fed_data/minutes.pkl", "wb") as f:
    pickle.dump(minutes_content, f)
```

Data Cleaning

- Clean textual data, e.g. remove stop words.
- Normalize data to lower cases.

```
import os
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
import string

# Download necessary resources if you haven't already
nltk.download('punkt')
nltk.download('stopwords')


# Function to clean and tokenize text into words
def clean_text_to_words(input_text):
    """Cleans the text by removing stop words and punctuation, then tokenizes it."""

    # Normalize to lowercase
    text = input_text.lower()

    # Tokenize the text into words
    tokens = word_tokenize(text)

    # Remove punctuation and keep only alphabetic tokens
    tokens = [word for word in tokens if word.isalpha()]  # filters out punctuation

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    return " ".join(tokens)


def clean_and_tokenize_sentences(input_text):
    """
    Tokenizes the input text into sentences, cleans each sentence by removing stop words and punctuation,
    and normalizes to lower case.

    Parameters:
    input_text (str): The text to be processed.

    Returns:
    list: A list of cleaned sentences.
    """
    # Normalize to lowercase
    input_text = input_text.lower()

    # Tokenize the input text into sentences
    sentences = sent_tokenize(input_text)

    stop_words = set(stopwords.words('english'))  # Set of English stop words
    cleaned_sentences = []

    for sentence in sentences:
        # Remove punctuation
        cleaned_sentence = sentence.translate(str.maketrans('', '', string.punctuation))

        # Tokenize the cleaned sentence into words
        words = word_tokenize(cleaned_sentence)

        # Remove stop words
        words = [word for word in words if word not in stop_words and word.isalpha()]

        # Join the cleaned words back into a sentence
        cleaned_sentences.append(" ".join(words))  # Optionally, you can keep words in tokens instead of joining them as a s

    return cleaned_sentences
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
with open("fed_data/minutes.pkl", "rb") as f:
  minutes = pickle.load(f)

with open("fed_data/statements.pkl", "rb") as f:
  statements = pickle.load(f)

with open("fed_data/press.pkl", "rb") as f:
  press_content = pickle.load(f)
```

```
minutes['minutes_cleaned'] = minutes['minutes'].apply(clean_text_to_words)
statements['statement_cleaned'] = statements['statement'].apply(clean_text_to_words)
press_content['press_cleaned'] = press_content['press'].apply(clean_text_to_words)


minutes['minutes_Sentence'] = minutes['minutes'].apply(clean_and_tokenize_sentences)
statements['statement_Sentence'] = statements['statement'].apply(clean_and_tokenize_sentences)
press_content['press_Sentence'] = press_content['press'].apply(clean_and_tokenize_sentences)


with open("fed_data/minutes_cleaned.pkl", "wb") as f:
  pickle.dump(minutes, f)

with open("fed_data/statements_cleaned.pkl", "wb") as f:
  pickle.dump(statements, f)

with open("fed_data/press_content_cleaned.pkl", "wb") as f:
  pickle.dump(press_content, f)
```

Data Parsing and Tagging

POS Tagging: Assign POS tags to each word in the tokenized sentences.

```
nltk.download('averaged_perceptron_tagger')  # For POS tagging
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
minutes['minutes_cleaned_POS'] = minutes['minutes_cleaned'].apply(lambda x : nltk.pos_tag(x.split(" ")))
statements['statement_cleaned_POS'] = statements['statement_cleaned'].apply(lambda x : nltk.pos_tag(x.split(" ")))
press_content['press_cleaned_POS'] = press_content['press_cleaned'].apply(lambda x : nltk.pos_tag(x.split(" ")))
```

Please create a corpus of the FOMC Meeting Minutes, Fed speeches and Press Conference transcripts and evaluate the level of hawkishness and dovishness of those statements using an appropriate polarity score for each document.

- Build Corpus
- WordList Method +tf-idf method
- Using FinBert to vectorize sentences and calculate the factor similarity score

```
##Factor Similarity Method

import torch
from transformers import BertTokenizer, BertModel
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Load the FinBERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained("yiyanghkust/finbert-tone")
model = BertModel.from_pretrained("yiyanghkust/finbert-tone").to(device)

# Check if GPU is available and get the device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def calculate_similarity(vec1, vec2):
    """Calculates the cosine similarity between two vectors."""
    vec1 = vec1.detach().cpu().numpy().reshape(1, -1)  # Reshape for sklearn compatibility
    vec2 = vec2.detach().cpu().numpy().reshape(1, -1)  # Use `detach()` to remove from the computation graph
    return cosine_similarity(vec1, vec2)[0][0]  # Return the similarity score


def get_sentence_embedding(sentence):
    """Generates the vector embedding for a given sentence using FinBERT."""
    # Tokenize the sentence
    inputs = tokenizer(sentence, return_tensors="pt", truncation=True, padding=True, max_length=512).to(device)

    # Get the outputs from FinBERT
    with torch.no_grad():
        outputs = model(**inputs)

    # Return the mean pooling of the last hidden states as the sentence embedding
    embeddings = outputs.last_hidden_state
    return torch.mean(embeddings, dim=1).squeeze()  # take the mean along the sequence dimension# Sample FOMC meeting statem

hawkish = "Federal Reserve need to raise interest rates sooner than expected if inflation continues to rise above our target
dovish = "Federal Reserve will maintain the current low interest rates to support the ongoing economic recovery."

# Generate embeddings for each statement
hawkish_emb = get_sentence_embedding(hawkish)
dovish_emb = get_sentence_embedding(dovish)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenizat
  warnings.warn(
```

Generate hawkish and dovish score per document.

- Hawkish Score on Doc[i] = Average Similarity(Sentence in Doc[i], Hawkish)
- Dovish Score on Doc[i] = Average Similarity(Sentence in Doc[i], Hawkish)

```
def calculate_hawkish_scores(sentences, hawkish_emb):
    hawkish_scores = []
    for sentence in sentences:
        sentence_emb = get_sentence_embedding(sentence)
        similarity = calculate_similarity(sentence_emb, hawkish_emb)
        hawkish_scores.append(similarity)
    print("finished calculate_hawkish_scores")
    return np.mean(hawkish_scores)

def calcuate_dovish_scores(sentences, dovish_emb):
    dovish_scores = []
    for sentence in sentences:
        sentence_emb = get_sentence_embedding(sentence)
        similarity = calculate_similarity(sentence_emb, dovish_emb)
        dovish_scores.append(similarity)
    print("finished calcuate_dovish_scores")
    return np.mean(dovish_scores)
```

```
press_content['hawkish_score_factor_similarity'] = press_content['press_Sentence'].apply(lambda x : calculate_hawkish_scores
press_content['dovish_score_factor_similarity'] = press_content['press_Sentence'].apply(lambda x : calcuate_dovish_scores(x,
```

```
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
```

```
statements['hawkish_score_factor_similarity'] = statements['statement_Sentence'].apply(lambda x : calculate_hawkish_scores(
statements['dovish_score_factor_similarity'] = statements['statement_Sentence'].apply(lambda x : calcuate_dovish_scores(x, 
```

```
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
finished calcuate_dovish_scores
```

```
minutes['hawkish_score_factor_similarity'] = minutes['minutes_Sentence'].apply(lambda x : calculate_hawkish_scores(x, hawkis
minutes['dovish_score_factor_similarity'] = minutes['minutes_Sentence'].apply(lambda x : calcuate_dovish_scores(x, dovish_em
```

```
Finished calcuate dovish scores

minutes = minutes.rename(columns={'hawkish_score':'hawkish_score_factor_similarity',
                        'dovish_score':'dovish_score_factor_similarity'})
statements = statements.rename(columns={'hawkish_score':'hawkish_score_factor_similarity',
                        'dovish_score':'dovish_score_factor_similarity'})
press_content = press_content.rename(columns={'hawkish_score':'hawkish_score_factor_similarity',
                        'dovish_score':'dovish_score_factor_similarity'})


with open("fed_data/minutes_score.pkl", "wb") as f:
  pickle.dump(minutes, f)

with open("fed_data/statements_score.pkl", "wb") as f:
  pickle.dump(statements, f)

with open("fed_data/press_content_score.pkl", "wb") as f:
  pickle.dump(press_content, f)
```

WordList Method

For Document k, the sentiment score is,

$$x_k(t) = \frac{1}{n_k} \sum_{i=1}^{n} sign[\sum_{p=1}^{m} L(p)S(p)\mathbf{1_k(i)}]$$

instead of using phrases, we use words directly, where

- S: sentiment of word
- L: tf-idf weight with keywords as vocabulary
- 1: presence of topics(Hawkish/Dovish) in word y

```
data
```

	date	statements_hawkish_score_factor_similarity	statements_dovish_score_factor_similarity	statements_hawkish_
0	20110126	0.472397	0.567278	
1	20110315	0.437926	0.602098	
2	20110427	0.433731	0.590298	
3	20110622	0.465729	0.542188	
4	20110809	0.478743	0.526855	
...	...	...	...	
108	20240320	0.440875	0.512803	
109	20240501	0.444065	0.507015	
110	20240612	0.441276	0.519891	
111	20240731	0.435557	0.510684	
112	20240918	0.457373	0.498214	

112 rows × 15 columns

后续步骤： 使用 data生成代码 查看推荐的图表 New interactive sheet

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

minutes_corpus = minutes['minutes_cleaned'].values
statements_corpus = statements['statement_cleaned'].values
press_content_corpus = press_content['press_cleaned'].values

# Define hawkish and dovish terms with their sentiment scores
hawkish_sentiment_scores = {
    "increase": 1,  # Hawkish
    "tighten": 1,   # Hawkish
    "hawkish": 1,   # Hawkish
    "strong": 1,    # Hawkish
    "concerning": 1, # Hawkish, can be subjective
    "stabilize": 1, # Hawkish
    "ratios": 1,    # Hawkish
    "aggressive": 1, # Hawkish
    "recession": -1, # Hawkish but negative context
    "tightening": 1,  # Hawkish
    "balance": 1,    # Hawkish
    "inflation": 1,  # Hawkish
    "tight": 1,      # Hawkish
    "policy": 0,     # Neutral
}

dovish_sentiment_scores = {
    "support": -1,   # Dovish
    "growth": -1,    # Dovish
    "slow": -1,      # Dovish
    "fragile": -1,   # Dovish
    "patience": -1,  # Dovish
    "employment": -1, # Dovish
    "stimulus": -1,  # Dovish
    "encourage": -1, # Dovish
    "loose": -1,     # Dovish
    "accommodative": -1, # Dovish
    "reduce": -1,    # Dovish
    "help": -1,      # Dovish
    "flexible": -1,  # Dovish
    "consider": 0    # Neutral
}
def calculate_sentiment_score(fomc_documents, sentiment_scores):
    # Convert the sentiment scores to numpy arrays for easier manipulation
    sentiment_terms = list(sentiment_scores.keys())
    sentiment_values = np.array([sentiment_scores[term] for term in sentiment_terms])

    # Initialize TF-IDF Vectorizer
    tfidf_vectorizer = TfidfVectorizer(vocabulary=sentiment_terms)

    # Fit and transform the FOMC documents
    tfidf_matrix = tfidf_vectorizer.fit_transform(fomc_documents)

    # Convert the TF-IDF matrix to an array
    tfidf_array = tfidf_matrix.toarray()

    # Initialize sentiment scores
    sentiment_scores_list = []

    # Calculating sentiment score for each document
    for i in range(len(fomc_documents)):
        doc_score = 0
        for j in range(len(sentiment_terms)):
            term_score = sentiment_values[j] * tfidf_array[i][j]
            doc_score += term_score

        sentiment_scores_list.append(doc_score)

    return sentiment_scores_list

minutes['hawkish_sentiment_score_wordList'] =calculate_sentiment_score(minutes['minutes_cleaned'].values, hawkish_sentiment_
minutes['dovish_sentiment_score_wordList'] =calculate_sentiment_score(minutes['minutes_cleaned'].values, dovish_sentiment_sc

statements['hawkish_sentiment_score_wordList'] =calculate_sentiment_score(statements['statement_cleaned'].values, hawkish_se
statements['dovish_sentiment_score_wordList'] =calculate_sentiment_score(statements['statement_cleaned'].values, dovish_sent

press_content['hawkish_sentiment_score_wordList'] =calculate_sentiment_score(press_content['press_cleaned'].values, hawkish_
press_content['dovish_sentiment_score_wordList'] =calculate_sentiment_score(press_content['press_cleaned'].values, dovish_se
```

Statistical Significance test

```
fomc_minutes = minutes[['date','hawkish_score_factor_similarity', 'dovish_score_factor_similarity', 'hawkish_sentiment_score
fomc_minutes.rename(columns = {'hawkish_score_factor_similarity':'minutes_hawkish_score_factor_similarity',
                      'dovish_score_factor_similarity':'minutes_dovish_score_factor_similarity',
                      'hawkish_sentiment_score_wordList':'minutes_hawkish_sentiment_score_wordList',
                      'dovish_sentiment_score_wordList':'minutes_dovish_sentiment_score_wordList'}, inplace = True)

fomc_statements = statements[['date','hawkish_score_factor_similarity', 'dovish_score_factor_similarity', 'hawkish_sentiment
fomc_statements.rename(columns = {'hawkish_score_factor_similarity':'statements_hawkish_score_factor_similarity',
                      'dovish_score_factor_similarity':'statements_dovish_score_factor_similarity',
                      'hawkish_sentiment_score_wordList':'statements_hawkish_sentiment_score_wordList',
                      'dovish_sentiment_score_wordList':'statements_dovish_sentiment_score_wordList'}, inplace = True)

fomc_press = press_content[['date','hawkish_score_factor_similarity', 'dovish_score_factor_similarity', 'hawkish_sentiment_s
fomc_press.rename(columns = {'hawkish_score_factor_similarity':'press_hawkish_score_factor_similarity',
                      'dovish_score_factor_similarity':'press_dovish_score_factor_similarity',
                      'hawkish_sentiment_score_wordList':'press_hawkish_sentiment_score_wordList',
                      'dovish_sentiment_score_wordList':'press_dovish_sentiment_score_wordList'}, inplace = True)
```

```
<ipython-input-137-6a997bd554b5>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  fomc_minutes.rename(columns = {'hawkish_score_factor_similarity':'minutes_hawkish_score_factor_similarity',
<ipython-input-137-6a997bd554b5>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  fomc_statements.rename(columns = {'hawkish_score_factor_similarity':'statements_hawkish_score_factor_similarity',
<ipython-input-137-6a997bd554b5>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  fomc_press.rename(columns = {'hawkish_score_factor_similarity':'press_hawkish_score_factor_similarity',
```

```
import yfinance as yf
import pandas as pd

# Define the ticker symbols for the 10Y and 2Y Treasury yields
# 10Y Treasury yield symbol
ticker_10y = '^TNX'  # This represents the 10-Year Treasury yield
# 2Y Treasury yield symbol
ticker_2y = '^IRX'   # This represents the 2-Year Treasury yield

# Fetch historical data for 10Y and 2Y Treasury yields
# We will get the last 30 days of yield data
start_date = '2011-01-01'  # Define start date for historical data
end_date = '2024-10-10'     # Today's date

# Get the data
yield_10y = yf.download(ticker_10y, start=start_date, end=end_date)
yield_2y = yf.download(ticker_2y, start=start_date, end=end_date)

# Make sure to reset the index to convert dates into a column
yield_10y.reset_index(inplace=True)
yield_2y.reset_index(inplace=True)
spread = (yield_10y.set_index("Date") - yield_2y.set_index("Date"))
spread['TsyYldSprd'] = spread['Close'] - spread['Open']
spread = spread.reset_index()
spread['FOMC_JOINT_DATE'] = spread['Date'].shift(1)
spread = spread.dropna()
spread['FOMC_JOINT_DATE'] = spread['FOMC_JOINT_DATE'].apply(lambda x : x.strftime('%Y%m%d'))
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

```
data = pd.merge(pd.merge(fomc_statements,fomc_press,on='date', how='left'),fomc_minutes, on = 'date', how = 'left').fillna(0

data = pd.merge(data,spread[['FOMC_JOINT_DATE','TsyYldSprd']],left_on='date',right_on='FOMC_JOINT_DATE',how='left').dropna()
```

```
fomc_statements['statements_hawkish_score_factor_similarity'].median()
```

```
0.438028484582901
```

```
### Statistical Analysis on Factor Similarity Scores of FOMC Statements
import pandas as pd
import statsmodels.api as sm
data = data.dropna()

# Define the dependent variable (Y) and independent variables (X)
```

```
# Define the dependent variable (Y) and independent variables (X)
X = data[['statements_hawkish_score_factor_similarity', 'statements_dovish_score_factor_similarity']]
Y = data['TsyYldSprd']

# Add a constant to the model (intercept)
X = sm.add_constant(X)

# Fit the regression model
model = sm.OLS(Y, X).fit()

# Print the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             TsyYldSprd   R-squared:                       0.011
Model:                            OLS   Adj. R-squared:                 -0.007
Method:                 Least Squares   F-statistic:                    0.6097
Date:                Sun, 13 Oct 2024   Prob (F-statistic):              0.545
Time:                        06:06:07   Log-Likelihood:                 172.76
No. Observations:                 111   AIC:                            -339.5
Df Residuals:                     108   BIC:                            -331.4
Df Model:                           2
Covariance Type:            nonrobust
==================================================================================================================
                                                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------------------------
const                                             0.1286      0.134      0.963      0.338      -0.136       0.393
statements_hawkish_score_factor_similarity       -0.0570      0.216     -0.264      0.792      -0.484       0.370
statements_dovish_score_factor_similarity        -0.1855      0.173     -1.072      0.286      -0.528       0.157
==============================================================================
Omnibus:                       23.466   Durbin-Watson:                   2.075
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.206
Skew:                           0.864   Prob(JB):                     1.53e-10
Kurtosis:                       5.605   Cond. No.                         60.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
### Statistical Analysis on Factor Similarity Scores of FOMC Press

# Define the dependent variable (Y) and independent variables (X)
X = data[['press_hawkish_score_factor_similarity', 'press_dovish_score_factor_similarity']]
Y = data['TsyYldSprd']

# Add a constant to the model (intercept)
X = sm.add_constant(X)

# Fit the regression model
model = sm.OLS(Y, X).fit()

# Print the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             TsyYldSprd   R-squared:                       0.045
Model:                            OLS   Adj. R-squared:                  0.027
Method:                 Least Squares   F-statistic:                     2.565
Date:                Sun, 13 Oct 2024   Prob (F-statistic):             0.0816
Time:                        05:59:03   Log-Likelihood:                 176.77
No. Observations:                 112   AIC:                            -347.5
Df Residuals:                     109   BIC:                            -339.4
Df Model:                           2
Covariance Type:            nonrobust
==================================================================================================================
                                                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------------------------
const                                           -0.0063      0.009     -0.729      0.468      -0.024       0.011
press_hawkish_score_factor_similarity           -1.2403      0.632     -1.962      0.052      -2.493       0.013
press_dovish_score_factor_similarity             1.1497      0.572      2.009      0.047       0.016       2.284
==============================================================================
Omnibus:                       26.432   Durbin-Watson:                   2.099
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               54.466
Skew:                           0.939   Prob(JB):                     1.49e-12
Kurtosis:                       5.853   Cond. No.                         192.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
### Statistical Analysis on Factor Similarity Scores of FOMC Minutes
```

```
### Statistical Analysis on Factor Similarity Scores of FOMC Minutes

# Define the dependent variable (Y) and independent variables (X)
X = data[['minutes_hawkish_score_factor_similarity', 'minutes_dovish_score_factor_similarity']]
Y = data['TsyYldSprd']

# Add a constant to the model (intercept)
X = sm.add_constant(X)

# Fit the regression model
model = sm.OLS(Y, X).fit()

# Print the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             TsyYldSprd   R-squared:                       0.027
Model:                            OLS   Adj. R-squared:                  0.009
Method:                 Least Squares   F-statistic:                     1.524
Date:                Sun, 13 Oct 2024   Prob (F-statistic):              0.222
Time:                        05:59:56   Log-Likelihood:                 175.74
No. Observations:                 112   AIC:                            -345.5
Df Residuals:                     109   BIC:                            -337.3
Df Model:                           2
Covariance Type:            nonrobust
==========================================================================================================
                                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------------------
const                                      0.0450      0.025      1.793      0.076      -0.005       0.095
minutes_hawkish_score_factor_similarity   -0.0071      0.288     -0.025      0.980      -0.578       0.564
minutes_dovish_score_factor_similarity    -0.0826      0.257     -0.321      0.749      -0.592       0.427
==============================================================================
Omnibus:                       29.261   Durbin-Watson:                   2.129
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               62.457
Skew:                           1.028   Prob(JB):                     2.74e-14
Kurtosis:                       6.026   Cond. No.                         95.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```