import networkx as nx

from collections import Counter

from itertools import combinations

from nltk.tokenize import word\_tokenize

from matplotlib.cm import ScalarMappable

from sklearn.metrics.pairwise import cosine\_similarity

from gensim.models import Word2Vec

from scipy.cluster.hierarchy import linkage, dendrogram

from collections import Counter

**Python script for performing hierarchical clustering**

top\_words = dict(fdist.most\_common(20)[1:])  # Excluding first term

word\_frequencies = [[freq] for freq in top\_words.values()]

# Assuming 'terms' is defined or replace it with appropriate labels

terms = list(top\_words.keys())

# Perform hierarchical clustering

linkage\_matrix = linkage(word\_frequencies, method ='ward')

# Plot the dendrogram

plt.figure(figsize=(10, 14))

dendrogram(linkage\_matrix, labels=terms, orientation='top')

plt.xlabel('Most frequent words')

plt.ylabel('Linkage distance')

plt.gca().spines['top'].set\_visible(False)

plt.gca().spines['right'].set\_visible(False)

plt.xticks(rotation=45)

plt.show()

**Python script for co-occurrence analysis**

corpus = df["clean\_tokens2"].apply(word\_tokenize)  # Tokenize each sentence in the corpus

fdist = Counter()

for sentence in corpus:

    fdist.update(sentence)

words\_wn = fdist.most\_common(30)

common\_terms = [word for word, \_ in words\_wn]

term\_network = nx.Graph()

term\_network.add\_nodes\_from(common\_terms)

co\_occurrences = Counter()

for sentence in corpus:

    combinations\_list = list(combinations(set(sentence), 2))

    co\_occurrences.update(combinations\_list)

for (term1, term2), weight in co\_occurrences.items():

    if term1 in common\_terms and term2 in common\_terms:

        term\_network.add\_edge(term1, term2, weight=weight)

edge\_colors = [0.01 \* term\_network[u][v]['weight'] for u, v in term\_network.edges()]

plt.figure(figsize=(15, 10))

pos = nx.spring\_layout(term\_network)  # Position nodes using a spring layout algorithm

labels = nx.get\_edge\_attributes(term\_network, 'weight')

nx.draw(term\_network, pos, with\_labels=True, node\_color='green', font\_size=16, node\_size=10,

        edge\_color=edge\_colors, edge\_cmap=plt.cm.gray, width = edge\_widths)

sm = plt.cm.ScalarMappable(cmap=plt.cm.gray, norm=plt.Normalize(vmin=min(edge\_colors),

                                                                vmax=max(edge\_colors)))

sm.set\_array([])

# Set the font family and size

font = {'family': 'serif', 'weight': 'normal', 'size': 16}

plt.rc('font', \*\*font)

plt.colorbar(sm, label='Edge weight')

plt.show()

**Python script for co-association analysis**

data = df['clean\_tokens2'].apply(lambda x: x.split()).tolist()

model = Word2Vec(data,

vector\_size = 50,

window = 5,

min\_count = 3,

workers = 3,

sg = 1,

hs = 1,

negative = 0,

epochs = 10,

seed = 50,

)

word\_vectors = model.wv

all\_words = [word for sublist in data for word in sublist]

top\_30\_words = [word for word, \_ in Counter(all\_words).most\_common(30)]

filtered\_words = [word for word in top\_30\_words if word in word\_vectors.index\_to\_key]

filtered\_vectors = [word\_vectors[word] for word in filtered\_words]

cosine\_sim = cosine\_similarity(filtered\_vectors)

for i, word1 in enumerate(filtered\_words):

for j, word2 in enumerate(filtered\_words):

if i < j:

print(f"Similarity between {word1} and {word2}: {cosine\_sim[i, j]}")

G = nx.Graph()

for word in filtered\_words:

G.add\_node(word)

for i, word1 in enumerate(filtered\_words):

for j, word2 in enumerate(filtered\_words):

if i < j: # Avoid duplicates

similarity = cosine\_sim[i, j]

if similarity > 0.3: # Lowered the threshold for more connections

G.add\_edge(word1, word2, weight=similarity)

plt.figure(figsize=(10, 8))

pos = nx.spring\_layout(G)  # You can use different layout algorithms

nx.draw(G, pos, with\_labels=True, node\_size=300, font\_size=10, width=0.5, edge\_color='teal')

plt.show()

**Python script for collocation analysis**

import matplotlib.pyplot as plt

df["tokenized\_words"] = df["clean\_tokens2"].apply(word\_tokenize)

all\_tokens = [token for sublist in df["tokenized\_words"] for token in sublist]

bigram\_finder = BigramCollocationFinder.from\_words(all\_tokens)

def filter\_issue\_bigram(bigram):

    return 'safe' in bigram

issue\_bigrams = bigram\_finder.nbest(BigramAssocMeasures.likelihood\_ratio, 20000)

issue\_bigrams = [bigram for bigram in issue\_bigrams if filter\_issue\_bigram(bigram)]

if not issue\_bigrams:

    print("No bigrams found involving the word 'issue'.")

else:

    issue\_bigram\_freq = Counter(issue\_bigrams)

    # Create a dictionary for the word cloud

    bigram\_dict = {' '.join(bigram): freq for bigram, freq in issue\_bigram\_freq.items()}

    # Generate the word cloud

    if bigram\_dict:

        wordcloud = WordCloud(width=800, height=400, background\_color='white').generate\_from\_frequencies(bigram\_dict)

        plt.figure(figsize=(10, 5))

        plt.imshow(wordcloud, interpolation='bilinear')

        plt.axis('off')

        plt.show()

    else:

        print("No bigrams to display in the word cloud.")

wordcloud = WordCloud(width=800, height=400, background\_color='white', colormap='inferno').generate\_from\_frequencies(bigram\_dict)

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.show()