Preparation for Distant Reading

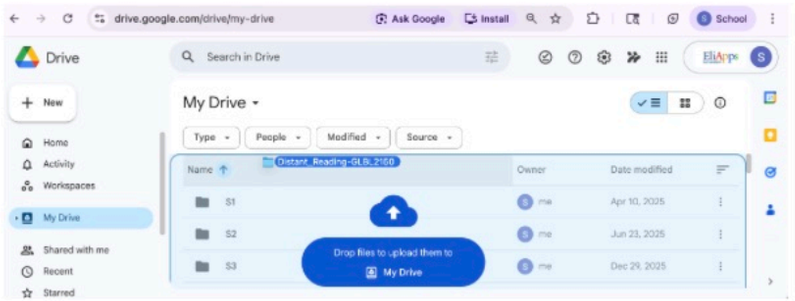
For our class this week, we will be meeting at the Poorvu Center (CTL 121) for an activity that will involve distant reading with Google Colab. Please complete the steps outlined below before the class session.

1- Make sure you have Google Colab installed on your Yale Google account. You can install it here: link

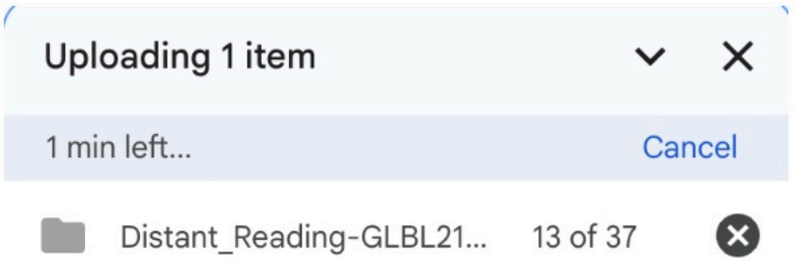
2- Download the **Distant_Reading-GLBL2150.zip** file to your computer.

3- Unzip the file on your computer. You'll get a folder named **Distant_Reading-GLBL2150**.

4- On your browser, open your **Yale** Google Drive and go to the section titled "My Drive"

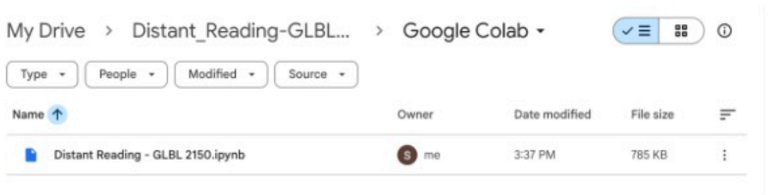


5- Drag and drop the unzipped folder directly into your drive. **Do not add it to any folders/subfolders you may already have.**



6-Wait for all the files to upload (you'll see the upload progress on the lower right side of your screen)

7-Go into the folder and click inside the "Google Colab" folder. Look for the file named "Distant Reading - GLBL 2150.ipynb". We will be working with this Google Colab file during Wednesday's class.



Set-Up & Introduction to Google Colab

Welcome to Google Colab, an online environment where you can run Python code. For this class session, we will be working with distant reading to derive insights from the *Foreign Relations of the United States* documents from the Kennedy administration.

The U.S. Office of the Historian made the files available online in an xml format, which will allow us to work with them today. You can access all the FRUS XML files that the Office of the Historian has made available here: link

Let's start with the basics! Google Colab allows you to write code in Python and print out outputs to your notebook.

Let's start with printing something out to your Colab notebook. Run the cell below by hovering on the cell then clicking on the "run cell" icon

```
#Let's print something out! --- this is a comment (comments start witl
print("Hello World")
```

Now, try printing out your name in the code cell below!

```
#Click on this cell and write the code to print out your name!
```

We can also store information in variables! Store your name in the 'your_name' variable below.

```
your_name = "your name here"
print(your_name)
```

Great! Now you know how to print output and create variables using Python.

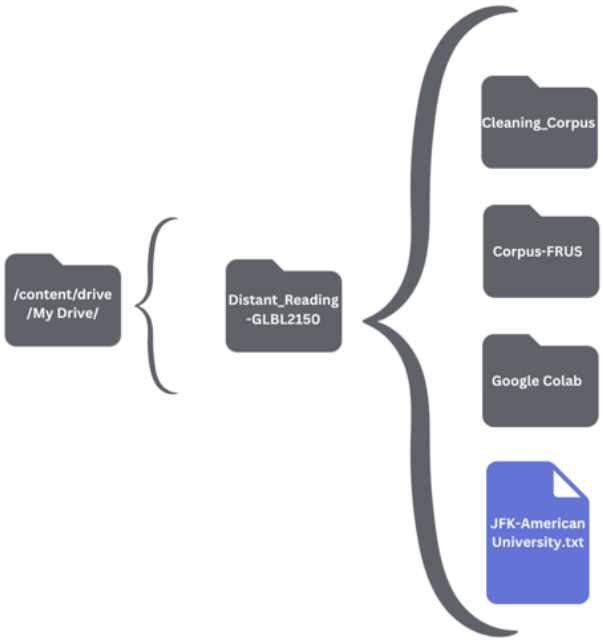
For this activity, we will be modifying/creating documents in our Google Drive.

We can start by connecting our Google Colab notebook to our Google Drive. Run the cell below!

```
#Connect google colab to your google drive
from google.colab import drive
drive.mount('/content/drive')
```

To access information in our Google Drive, we have to provide Google Colab our File Path.

This is what our folder currently looks like:



For every subfolder, we add it to our file path, adding a "/" between folder names.

Now, let's try printing out one of our documents. Let's print out JFK's American University speech!

If our file is named

```
JFK–AmericanUniversity.txt
```

What would the file path be?

Try generating code below! Ask Gemini to print out the contents of JFK's speech! Provide the file path.

```
Start coding or generate with AI.
```

Getting Started with our Corpus

We will be working with XML files. XML files have data stored in tags!

<p>is a common tag indicating a paragraph, while <head> is often used for chapter titles, and <title> is used to enclose a title.

What tags do you notice below?

```
<head>1. Paper Prepared by the Country Team Staff Committee<note n="1"
        type="source" xml:id="d1fn1">Source: Department of State, Central
        Files, 751K.5-<gloss target="#t_MSP1">MSP</gloss>/1-461. Secret. The
        Country Team Staff Committee was chaired by
            <persName>Mendenhall</persName> and composed of officers from
            <gloss target="#t_MAAG1">MAAG</gloss>, <gloss target="#t_USOM1"
            >USOM</gloss>, <gloss target="#t_USIS1">USIS</gloss>, <gloss
            target="#t_OSA1">OSA</gloss>, and the Embassy. Transmitted as
        enclosure 1 to despatch 276 from Saigon, January 4.</note></head>
<opener><dateline rendition="#right"><hi rend="italic"
                ><placeName>Saigon</placeName>, [<date calendar="gregorian"
                when="1961-01-04">January 4,
    1961</date>.]</hi></dateline></opener>
<p>The plan consisted of a table of contents, the basic plan, 3 annexes, 15
    appendices, and 5 tabs. Only the basic plan is printed here. For the
    draft plan transmitted to Saigon in 1960, see Def 982994, September 16,
        <ref target="frus1958-60v01#pg_572"><hi rend="italic">Foreign
            Relations</hi>, 1958–1960, vol. I, p. 572</ref>. In despatch
    486, Ambassador <persName corresp="#p_DE1">Durbrow</persName> commented
    that a number of the indispensable recommendations would “probably not
    be particularly palatable” to the Government of Vietnam, especially
    certain political actions and ideas about military-civilian
    relationships. The Ambassador completed his comments by stating that
    consideration should be given to what steps “we are prepared to take to
    encourage, or if necessary to force, acceptance of all essential
    elements of the plan.”</p>
```

For example: if my speech above was in xml format, the title would be written as:

<title> COMMENCEMENT ADDRESS AT AMERICAN UNIVERSITY, WASHINGTON, D.C., JUNE 10, 1963 </title>

the body of the text would be enclosed between and

We can use our new knowledge of XML files to our advantage. Let's clean up our data and extract the relevant information! From our first xml file, we can notice the volume title is enclosed in this tag:

```
<title type="complete">Foreign Relations of the United States, 1961–1963, Volume I,
    Vietnam, 1961</title>
```

Let's try to find our title in our xml document and print it out.

```
#Extracting Volume 1 title
import xml.etree.ElementTree as ET

#Selecting the xml file we will be looking at
tree = ET.parse('/content/drive/My Drive/Distant_Reading-GLBL2150/Cor
root = tree.getroot()

#Selecting the tag we want to search for
tag = 'title[@type="complete"]'

#We are going to parse (aka search) through our XML file to look for
#So for every XML element in our file, we are checking if it matches
for title_element in root.findall('.//{http://www.tei-c.org/ns/1.0}'+
    print(title_element.text)
```

Great! We were able to print out the title for our volume! But what if we want to know more about the contents of our volume?

Let's print out the chapter titles. **Find the xml tag that corresponds to the chapter titles and add it below.**

Go to Distant_Reading-GLBL2150 -> Corpus-FRUS -> and open the frus1961-63v01.xml file

```
#Volume 1 chapters
tag = 'Add here' #TO DO: Find the relevant tag for the chapter titles
for title_element in root.findall('.//{http://www.tei-c.org/ns/1.0}'+
    print(title_element.text)
```

Now that we know some of the basics, let's try looking at all the **titles** for the files in our corpus

```
#For all files in Corpus-FRUS, print out titles
import os
import glob
import xml.etree.ElementTree as ET

#This is the folder where we will search for our files
folder_path = '/content/drive/My Drive/Distant_Reading-GLBL2150/Corpus

#Within this folder, we are looking at all files with the ending '.xml
file_pattern = os.path.join(folder_path, '*.xml')

#For every file in our folder, we will search all the xml elements to
for filename in glob.glob(file_pattern):
    tree = ET.parse(filename)
    root = tree.getroot()
    for title_element in root.findall('.//{http://www.tei-c.org/ns/1.0
      print(title_element.text)
```

Now that we know what files are in our corpus, let's start processing our files!

Cleaning our Files

First, let's see how can we write text to a file. Let's try creating a file called "output.txt" in our 'Distant_Reading-GLBL2150' folder.

Write "Hello World" in the file.

Let's try asking Gemini to generate this code!

```
Start coding or generate with AI.
```

Now that we know how to save output as a plain text file (.txt) let's convert our .xml files to .txt files so we can begin processing them.

We also want to isolate relevant parts of our files for text processing later on, so we can use the <title>, <head>, and <p> tags we talked about earlier.

Run the code below!

```
#For all files in Corpus-FRUS, isolate the relevant tags,and add it t
import os
import glob
import xml.etree.ElementTree as ET

folder_path = '/content/drive/My Drive/Distant_Reading-GLBL2150/Corpu
file_pattern = os.path.join(folder_path, '*.xml') #What file type is
output_dir = '/content/drive/My Drive/Distant_Reading-GLBL2150/Cleani

# Make sure the output directory exists
os.makedirs(output_dir, exist_ok=True)

#For each xml file
for filename in glob.glob(file_pattern):
    tree = ET.parse(filename)
    root = tree.getroot()

    title_element = root.find('.//{http://www.tei-c.org/ns/1.0}title[

    isolate_filename = (os.path.basename(filename))
    output_file_path = os.path.join(output_dir, isolate_filename + '.

    with open(output_file_path, 'w') as f:
      f.write(title_element.text + '\n')  # write the title

      for element in root.iter():
        #find and write the body tags
        if element.tag == '{http://www.tei-c.org/ns/1.0}p':
            if element.text is not None:
              f.write(element.text + '\n')

        #find and write the heading tags
        if element.tag == '{http://www.tei-c.org/ns/1.0}head':
            if element.text is not None:
              f.write(element.text + '\n')
```

Tokenization, Lemmatization, Stop Words & Topic Modeling

Let's start with **topic modeling**.

Topic modeling is "a way of identifying patterns in a corpus. You take your corpus and run it through a tool which groups words across the corpus into 'topics'." (Brett, 2012). Essentially, we are going to group words that occur together frequently across our documents and display these as our 'topics.'

To be able to do this, we need to **tokenize** and **lemmatize** our corpus and remove **stop words**.

Tokenization splits our corpus into more managable "token" subunits (in our case, words). Lemmatization is a technique in Natural Language that reduces words to their base form (ex: learning becomes learn). You can learn more here!

Stop words are commonly used words ("the", "and", "an", "in", etc.) that we will be removing from our corpus before doing topic modeling. (We will also be removing punctuation!)

We will run the cell below first to download essential modules first! Then, run the cell below it to remove stop words and lemmatize the text. Let's start with JFK's speech before diving into our larger corpus:

```
#Download stopwords and punkt_tab (tokenizer) from the nltk library
import nltk
nltk.download('stopwords')
nltk.download('punkt_tab')
nltk.download('wordnet')

#Install gensim if not already installed
try:
    import gensim
except ModuleNotFoundError:
    %pip install gensim
```

```
#AMERICAN UNIVERSTIY SPEECH
import os
import glob
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
from nltk.stem import WordNetLemmatizer
import gensim
from gensim import corpora

filename = '/content/drive/My Drive/Distant_Reading-GLBL2150/JFK-Amer
output_dir = '/content/drive/My Drive/Distant_Reading-GLBL2150/JFK_Ou

os.makedirs(output_dir, exist_ok=True)

# 1. SETUP STOP WORDS
stop_words = set(stopwords.words('english'))
custom_stop_words = {'us'} ##These are custom words you can add depen
stop_words = stop_words.union(custom_stop_words)

lemmatizer = WordNetLemmatizer()
processed_documents = []


with open(filename, 'r', encoding='utf-8') as f_in:
    text = f_in.read()

    # 2. TOKENIZE
    word_tokens = word_tokenize(text)

    # 3. Filter punctuation, force lowercase, filter stopwords, then l
    lemmatized_words = [
        lemmatizer.lemmatize(w.lower())
        for w in word_tokens
        if w.isalpha() and w.lower() not in stop_words
    ]

    processed_documents.append(lemmatized_words)

    # Save lemmatized text to a new file
    isolate_filename = os.path.basename(filename)
    output_file_path = os.path.join(output_dir, "lemmatized_" + isola
    with open(output_file_path, 'w', encoding='utf-8') as f_out:
        f_out.write(' '.join(lemmatized_words))

print(f"Lemmatization complete. Lemmatized files saved to: {output_di

# Prepare data for topic modeling
dictionary = corpora.Dictionary(processed_documents)

corpus = [dictionary.doc2bow(doc) for doc in processed_documents]
```

```
# Perform LDA Topic Modeling
num_topics = 1
lda_model = gensim.models.LdaModel(corpus, num_topics=num_topics, id2w

print("\nTopic Modeling Results:")
for idx, topic in lda_model.print_topics(-1):
    print(f"Topic: {idx} \nWords: {topic}\n")
```

Did you expect the topic given above? How does this compare to your close reading of JFK's speech?

Now that we've tested out topic modeling, let's try it out on our larger corpus!

```
#Remove stop words, lemmatize, and preform topic modeling.

import os
import glob
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import gensim
from gensim import corpora

folder_path = '/content/drive/My Drive/Distant_Reading-GLBL2150/Clean
file_pattern = os.path.join(folder_path, '*.txt')
output_dir = '/content/drive/My Drive/Distant_Reading-GLBL2150/Cleani

os.makedirs(output_dir, exist_ok=True)

# 1. Set-up STOP WORDS
stop_words = set(stopwords.words('english'))
custom_stop_words = {'would', 'could', 'said', 'also', 'upon', 'may',
stop_words = stop_words.union(custom_stop_words)

lemmatizer = WordNetLemmatizer()
processed_documents = []

for filename in glob.glob(file_pattern):
    with open(filename, 'r', encoding='utf-8') as f_in:
```

```
            text = f_in.read()

        # 2. TOKENIZE
        word_tokens = word_tokenize(text)

        # 3. Filter punctuation, force lowercase, filter stopwords, then l
        lemmatized_words = [
            lemmatizer.lemmatize(w.lower())
            for w in word_tokens
            if w.isalpha() and w.lower() not in stop_words
        ]

        processed_documents.append(lemmatized_words)

        # Save lemmatized text to a new file
        isolate_filename = os.path.basename(filename)
        output_file_path = os.path.join(output_dir, "lemmatized_" + isola
        with open(output_file_path, 'w', encoding='utf-8') as f_out:
            f_out.write(' '.join(lemmatized_words))

    print(f"Lemmatization complete. Lemmatized files saved to: {output_di

    # Prepare data for topic modeling
    dictionary = corpora.Dictionary(processed_documents)

    corpus = [dictionary.doc2bow(doc) for doc in processed_documents]

    # Perform LDA Topic Modeling
    num_topics = 5
    lda_model = gensim.models.LdaModel(corpus, num_topics=num_topics, id2

    print("\nTopic Modeling Results:")
    for idx, topic in lda_model.print_topics(-1):
        print(f"Topic: {idx} \nWords: {topic}\n")
```

There are multiple repeated words across our topics. What do words like "president" mean in the context of our FRUS corpus? Would you consider these stop words in this corpus?

Discuss and add relevant stop words to your **custom_stop_words** list below. Then, try re-running the code. Are the topics different?

You'll also notice that some of our words are being split apart: "United States" becomes "united", "states" through tokenization, and "U.S." is split apart when we remove punctuation. Let's fix that too!

```
#Remove stop words, lemmatize, and preform topic modeling - improved!

import os
import glob
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import gensim
from gensim import corpora

folder_path = '/content/drive/My Drive/Distant_Reading-GLBL2150/Clean
file_pattern = os.path.join(folder_path, '*.txt')
output_dir = '/content/drive/My Drive/Distant_Reading-GLBL2150/Cleani

os.makedirs(output_dir, exist_ok=True)

# 1. Set-up STOP WORDS
stop_words = set(stopwords.words('english'))
#Add more stop words to the list below!
custom_stop_words = {'would', 'could', 'said', 'also', 'upon', 'may',
stop_words = stop_words.union(custom_stop_words)

# Let's define custom phrases to be treated as single tokens (e.g., 'u
custom_phrases = {
    'United States': 'united_states',
    'U.S.': 'united_states',
    'U.S.A.': 'united_states'
}

lemmatizer = WordNetLemmatizer()
```

```
processed_documents = []

for filename in glob.glob(file_pattern):
    with open(filename, 'r', encoding='utf-8') as f_in:
        text = f_in.read()

        # Replace custom phrases in the text before tokenization
        for phrase, replacement in custom_phrases.items():
          text = text.replace(phrase, replacement)
          text = text.replace(phrase.capitalize(), replacement) # Han
          text = text.replace(phrase.upper(), replacement) # Handle u

    # 2. TOKENIZE
    word_tokens = word_tokenize(text)

    # 3. Filter punctuation, force lowercase, filter stopwords, then l
    lemmatized_words = [
        lemmatizer.lemmatize(w.lower())
        for w in word_tokens
        if any(char.isalpha() for char in w) and w.lower() not in sto
    ]

    processed_documents.append(lemmatized_words)

    # Save lemmatized text to a new file
    isolate_filename = os.path.basename(filename)
    output_file_path = os.path.join(output_dir, "lemmatized_" + isola
    with open(output_file_path, 'w', encoding='utf-8') as f_out:
        f_out.write(' '.join(lemmatized_words))

print(f"Lemmatization complete. Lemmatized files saved to: {output_di

# Prepare data for topic modeling
dictionary = corpora.Dictionary(processed_documents)

corpus = [dictionary.doc2bow(doc) for doc in processed_documents]

# Perform LDA Topic Modeling
num_topics = 5
lda_model = gensim.models.LdaModel(corpus, num_topics=num_topics, id2

print("\nTopic Modeling Results:")
for idx, topic in lda_model.print_topics(-1):
    print(f"Topic: {idx} \nWords: {topic}\n")
```

Let's visualize our topics! Run the cell below:

```
import matplotlib.pyplot as plt

def generate_topic_bar_charts(lda_model, num_topics=5):
    for i in range(num_topics):
        plt.figure(figsize=(10, 6))
        # Get topic words and their probabilities
        topic_words = lda_model.show_topic(i, topn=10) # Get top 10 w
        words = [word for word, prob in topic_words]
        probs = [prob for word, prob in topic_words]

        # Create horizontal bar chart
        plt.barh(words, probs, color='skyblue')
        plt.xlabel('Probability')
        plt.ylabel('Word')
        plt.title(f'Topic {i} - Top Words')
        plt.gca().invert_yaxis() # Put the highest probability word a
        plt.tight_layout()
        plt.show()

generate_topic_bar_charts(lda_model, num_topics=num_topics)
```

```
#We can also make word clouds
import matplotlib.pyplot as plt
from wordcloud import WordCloud

def generate_wordclouds(lda_model, num_topics=5):
    for i in range(num_topics):
        plt.figure(figsize=(10, 5))
        # Create a dictionary of word: probability
        word_freq = dict(lda_model.show_topic(i, 30))
        wc = WordCloud(background_color="white").generate_from_freque
        plt.imshow(wc)
        plt.title(f"Topic {i}")
        plt.axis("off")
        plt.show()

generate_wordclouds(lda_model, num_topics=num_topics)
```

What do these topics tell us about our corpus?

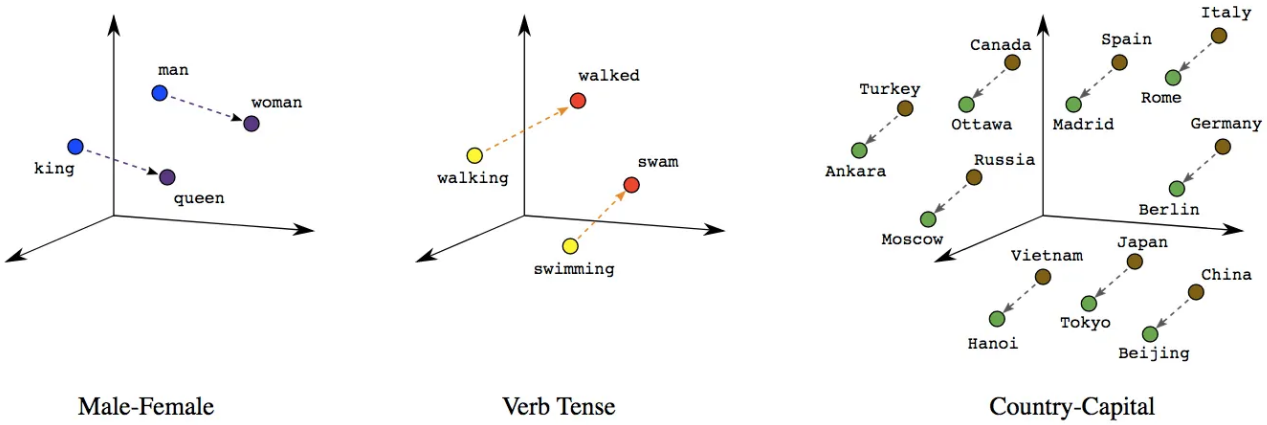
What if we want to examine our speech in the context of the FRUS documents?

```
# 1. Preprocess the speech
speech_file_path = '/content/drive/My Drive/Distant_Reading-GLBL2150/
# Read the lemmatized speech content from the file
with open(speech_file_path, 'r', encoding='utf-8') as f:
    lemmatized_speech_text = f.read()

# Split the text into a list of words (tokens)
speech_tokens = lemmatized_speech_text.split()

# 2. Convert to Bag-of-Words using your our topics (from the FRUS cor
new_bow = dictionary.doc2bow(speech_tokens) #we defined the dictionar

# 3. Get the topic distribution
new_topics = lda_model.get_document_topics(new_bow)

for topic_id, score in sorted(new_topics, key=lambda x: x[1], reverse=
    print(f"Topic {topic_id}: {score*100:.1f}%")
```

What does this tell us about this speech in the broader context of the JFK administration? What questions does this make you think of?

Vector Embeddings

Vector embeddings are numerical representations of our words. When we plot these numerical vectors in a multi-dimensional space, similar words will be grouped together.



Male-Female Verb Tense Country-Capital

Image from: [Anala, 2020]

Let's train our word2vec model to generate word vectors for our corpus

```
import gensim
import os
import glob

#Let's get our proccessed files for the FRUS corpus
folder_path = '/content/drive/My Drive/Distant_Reading-GLBL2150/Clean
file_pattern = os.path.join(folder_path, '*.txt')

processed_documents = []

for filename in glob.glob(file_pattern):
    with open(filename, 'r', encoding='utf-8') as f_in:
        text = f_in.read()
        words = text.split()
        processed_documents.append(words)

word2vec_model = gensim.models.Word2Vec(
    processed_documents,
    vector_size=100,
    window=5,
    min_count=1,
    workers=4
)

print("Word2Vec model trained successfully!")
```

Now, we will find words that are used in similar contexts in our corpus.

```
# Example 1: Find words most similar to a given word
word = 'war' #You can modify this!
print("Words similar to '" + word + "': \n")
try:
    similar_words = word2vec_model.wv.most_similar(word)
    for word, similarity in similar_words:
        print(f"  {word}: {similarity:.4f}")
except KeyError:
    print("word not found in the vocabulary.")
```

What other words might be interesting to try out? Try changing 'war' for another word. What insights can this give us about our corpus?

Another thing we can do is calculate the 'similarity' between words. Essentially, we are calculating the cosine similarity between our word vectors. Our result will range from -1 to 1, where -1 means words are opposites and 1 means words are very similar

```
#the words we are comparing! feel free to change them
word1 = 'soviet'
word2 = 'cuba'

print("Similarity between '"+ word1 + "' and '"+ word2 +"':")
try:
    similarity_score = word2vec_model.wv.similarity(word1, word2)
    print(f"  {similarity_score:.4f}")
except KeyError:
    print("  One or both words not found in the vocabulary.")
```

Feel free to experiment below!

```
Start coding or generate with AI.
```