In [ ]:

```python
#Import packages

import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import fitz
import re
import string
import datetime
import math
from glob import glob
from zipfile import ZipFile
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud
from gensim.models.word2vec import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity
from operator import itemgetter
```

In [ ]:

```python
def retrieve(n):
    zip_file = ZipFile('pdf_json.zip','r')
    random_files = np.random.choice(zip_file.namelist(), n)
    for names in zip_file.namelist():
        if names in random_files:
            zip_file.extract(names)
    #Transform json files to dataframe
    json_path = 'pdf_json/*.json'
    files = glob(json_path)
    data = []
    for file in files:
        with open(file) as pdf:
            json_data = json.loads(pdf.read())
            data.append(json_data)
    df = pd.DataFrame(data)
    #Drop columns that will not be used
    df = df.drop(columns = ['bib_entries', 'ref_entries', 'back_matter'])
    with open('metadata.csv', 'rb') as file:
        metadata = pd.read_csv(file)
    #Drop useless columns
    metadata = metadata.drop(columns = ['cord_uid', 'source_x','pmcid','pubmed_id', 'license','mag_id', 'who_covidence_id', 'pmc_json_files', 'arxiv_id','s2_id', 'pdf_json_files']
    #Joining json dataframe and metatdata.csv
    df_merge = df.merge(metadata, how = 'left', left_on = 'paper_id', right_on = 'sha')
    #Remane abstract column
    df_merge = df_merge.rename(columns = {"abstract_y": "abstract"})
    #Merge all body text into one string and store in main_content column
    #Remove body_text afterwards
    for i, row in df_merge.iterrows():
        try:
            content = ""
            body_text = pd.DataFrame(df_merge.loc[i]['body_text'])
            for text in body_text['text']:
                content += text
            df_merge.at[i, 'main_content'] = content
        except IndexError:
            df_merge.at[i, 'main_content'] = np.nan
    df_merge = df_merge.drop(columns = ['body_text'])
    #Set the country of first author with non-empty country as the country of the article is from
    #Remove metatdata afterwards
    for i, row in df_merge.iterrows():
        try:
            c = 0
            country_name = df_merge.loc[i]['metadata']['authors'][c]['affiliation']['location']['country']
            while(country_name == np.nan):
                c += 1
                country_name = df_merge.loc[i]['metadata']['authors'][c]['affiliation']['location']['country']
            df_merge.at[i, 'country'] = country_name
        except:
            continue
    df_merge = df_merge.drop(columns = ['metadata'])
    #Replace null under abstract with empty string
    df_merge['abstract'].fillna("", inplace = True)
    #Remove duplicated columns to reduce dimension
    df_merge = df_merge.drop(columns = ['abstract_x', 'sha'])
    df_merge = df_merge[~df_merge['title'].isna()]
    df_merge = df_merge.reset_index(drop = True)
    #Save cleaned dataframe
    df_merge.to_pickle('merged_df')
```

In [ ]:

```
def text_process1(text):
    #lowercase all characters
    text = text.lower()
    #remove punctuation
    text = re.sub(r'[%s]' % re.escape(punc), ' ', text)
    #remove unicode text
    text = re.sub(r'[^\x00-\x7F]+', ' ', text)
    #remove the numbers
    text = re.sub(r'[0-9]', '', text)
    #remove double space
    text = re.sub(r'\s{2,}', ' ', text)
    return text

def text_process2(text):
    #lowercase all characters
    text = text.lower()
    #remove punctuation
    text = re.sub(r'[%s]' % re.escape(string.punctuation), ' ', text)
    #remove unicode text
    text = re.sub(r'[^\x00-\x7F]+', ' ', text)
    #remove the numbers
    text = re.sub(r'[0-9]', '', text)
    #remove double space
    text = re.sub(r'\s{2,}', ' ', text)
    return text
```

In [ ]:

```
#Edit the two lines below according to the comment to retrieve a new set of articles

retrieve_new = False #Set to True to retrieve a new set of articles
n = 6000 #Set this value to number of articles to be retrieved

if(retrieve_new):
    retrieve(n)

#Load clean dataframe
```

```
df_merge = pd.read_pickle('merged_df')

#Check
df_merge
```

In [ ]:
```
###EDA before training model

#Plot top 10 count of articles by country
df_merge['country'].value_counts().head(10).plot(kind = 'bar', figsize = (12, 7), title = 'Number of articles by country')
plt.xticks(rotation = 45)
plt.savefig('Number of articles by country.png')
```

In [ ]:
```
#Plot number of articles published after 2003 in the sample

df_plot = df_merge[["publish_time"]].copy()
df_plot['publish_time'] = pd.to_datetime(df_plot['publish_time'])
minimum = pd.to_datetime(datetime.date(2003, 1, 1))
df = df_plot[df_plot['publish_time'] >= minimum]
p = df['publish_time'].groupby(df['publish_time'].dt.to_period("M")).agg('count')
#Plot top 10 count of articles by country
p.plot(kind = 'line', figsize = (12, 7), title = 'Number of articles published after 2003')
plt.xlabel('Year of publication')
plt.savefig('Number of articles published after 2003.png')
```

In [ ]:
```
###Text preprocessing and normalization

abstract_tokenized = []
main_content_tokenized = []
sentence_list = []

#Load stopwords
stop_words = set(stopwords.words('english'))
stop_words.add("et")
stop_words.add("al")
stop_words.add("also")
stop_words.add("may")

punc = string.punctuation
punc = punc.replace(".", "")

#Read abstract
for i, row in df_merge.iterrows():
    abstract_sent = []
    try:
        text = df_merge.at[i, 'abstract']
        splitted_text = text.split(sep = '.')
        for sentence in splitted_text:
            if(sentence != ''):
                sentence = text_process1(sentence)
                #tokenize and lemmatize the sentence
                word_tokens = word_tokenize(sentence)
                token_text = [w for w in word_tokens if not w.lower() in stop_words]
                lemmatizer = WordNetLemmatizer()
                lemmatized_text = []
                for word in token_text:
                    lemmatized_text.append(lemmatizer.lemmatize(word))
                abstract_sent.append(lemmatized_text)
                sentence_list.append(lemmatized_text)
            abstract_tokenized.append(abstract_sent)
        except:
            continue

#Read main content
for i, row in df_merge.iterrows():
    body_sent = []
    try:
        text = df_merge.at[i, 'main_content']
        splitted_text = text.split(sep = '.')
        for sentence in splitted_text:
            if(sentence != ''):
                sentence = text_process1(sentence)
                #tokenize and lemmatize the file
                word_tokens = word_tokenize(sentence)
                token_text = [w for w in word_tokens if not w.lower() in stop_words]
                lemmatizer = WordNetLemmatizer()
                lemmatized_text = []
                for word in token_text:
                    lemmatized_text.append(lemmatizer.lemmatize(word))
                body_sent.append(lemmatized_text)
                sentence_list.append(lemmatized_text)
        main_content_tokenized.append(body_sent)
    except:
        continue

sentence_list = [s for s in sentence_list if len(s) > 0]
```

In [ ]:

```
#WordCloud

freq = pd.Series(np.concatenate(sentence_list)).value_counts()
wordcloud = WordCloud(
            width = 300, height = 200,
            background_color = "white",
            max_words = 200,
            max_font_size = 40,
            stopwords = stop_words,
            scale = 5,
            random_state = 0).generate_from_frequencies(freq)
fig = plt.figure(1, figsize = (9,6))
plt.axis('off')
plt.imshow(wordcloud)
plt.savefig('WordCloud.png')
```

In [ ]:

```
#Word2Vec

#Define model
model = Word2Vec(sentence_list, vector_size = 100, window = 8, min_count = 1, sg = 0, workers = 4)

#Save model
model.save("Word2Vec.model")
```

In [ ]:

```
#Load model
model = Word2Vec.load("Word2Vec.model")
```

In [ ]:

```
##Take and clean query

#Read input
```

```
text = input("Please enter your query: ")
text = text_process2(text)
#tokenize and lemmatize the file
word_tokens = word_tokenize(text)
token_text = [w for w in word_tokens if not w.lower() in stop_words]
lemmatizer = WordNetLemmatizer()
query = []
for word in token_text:
    query.append(lemmatizer.lemmatize(word))
```

In [ ]:

```
#Vectorize query

query_vec = [0.0 for i in range(100)]
num = 0
for i in query:
    try:
        vec = model.wv[i]
    except:
        continue
    else:
        query_vec += vec
        num +=1
if(num != 0):
    query_vec /= num
```

In [ ]:

```
sims = [] #contains top 20 of most similary vector and corresponding sentence index
count = 0

for i in range(len(sentence_list)):
    sent_vec = [0.0 for i in range(100)]
    num = 0
    for j in sentence_list[i]:
        try:
            vec = model.wv[j]
        except:
            continue
        else:
            sent_vec += vec
            num +=1
    if(num != 0):
        sent_vec /= num
    similarity = cosine_similarity(np.expand_dims(query_vec, 0), np.expand_dims(sent_vec, 0))
    if(count < 20):
        sims.append(tuple((i, similarity)))
        count += 1
    else:
        if(similarity > min(sims, key = itemgetter(1))[1]):
            sims[sims.index(min(sims, key = itemgetter(1)))] = tuple((i, similarity))

sims.sort(key = lambda x: x[1], reverse = True)
```

In [ ]:

```
#Generate most relevant sentence with source

article = []

print("Showing top 10 relevant articles")
print("-----------------------------------------------------------------------------------------------------------------------")
```

```
for n in range(len(sims)):
    if(len(article) == 10):
        break
    for i in range(len(abstract_tokenized)):
        if(i not in article):
            for j in range(len(abstract_tokenized[i])):
                if(abstract_tokenized[i][j] == sentence_list[sims[n][0]]):
                    text = df_merge.loc[i]['abstract']
                    splitted_text = text.split(sep = '.')
                    location = text.find(splitted_text[j])
                    s = 0
                    e = len(text) - 1
                    if(location - 150 > 0):
                        s = location - 150
                    if(location + 450 < len(text)):
                        e = location + 450
                    while(text[e].isalpha()):
                        e += 1
                    print(df_merge.loc[i]['title'] + str(i))
                    print("\n")
                    while(text[s].isalpha()):
                        s += 1
                    print("... " + text[s:e] + " ...")
                    if(df_merge.loc[i]['url'] != np.nan):
                        print("Original document at: " + df_merge.loc[i]['url'])
                    else:
                        print("Original document at: Not available")
                    article.append(i)
                    print("----------------------------------------------------------------------------------------------------------------------")
    for i in range(len(main_content_tokenized)):
        if(i not in article):
            for j in range(len(main_content_tokenized[i])):
                if(main_content_tokenized[i][j] == sentence_list[sims[n][0]]):
                    text = df_merge.loc[i]['main_content']
                    splitted_text = text.split(sep = '.')
                    location = text.find(splitted_text[j])
                    s = 0
                    e = len(text) - 1
                    if(location - 150 > 0):
                        s = location - 150
                    if(location + 450 < len(text)):
                        e = location + 450
                    while(text[e].isalpha()):
                        e += 1
                    print(df_merge.loc[i]['title'] + str(i))
                    print("\n")
                    while(text[s].isalpha()):
                        s += 1
                    print("... " + text[s:e] + " ...")
                    if(df_merge.loc[i]['url'] != np.nan):
                        print("Original document at: " + df_merge.loc[i]['url'])
                    else:
                        print("Original document at: Not available")
                    article.append(i)
                    print("----------------------------------------------------------------------------------------------------------------------")
```