**CIS 593 – BIG DATA**

**LAB ASSIGNMENT – 4**

**Name: Pooja A. Khatri**

**CSU ID: 2783752**

**Extra credit work:**

**1.Inverted Index (Term Dictionary) Construction with TF-IDF**

**2. Cosine similarity matrix**

1. **Platforms**

Language used: Python

Python IDE: PyCharm

1. **Important Libraries**

Nltk : for natural language processing

BeautifulSoup : for extracting text between the tags

Pandas : to create dataframe for TF-IDF

Scipy : for cosine similarity

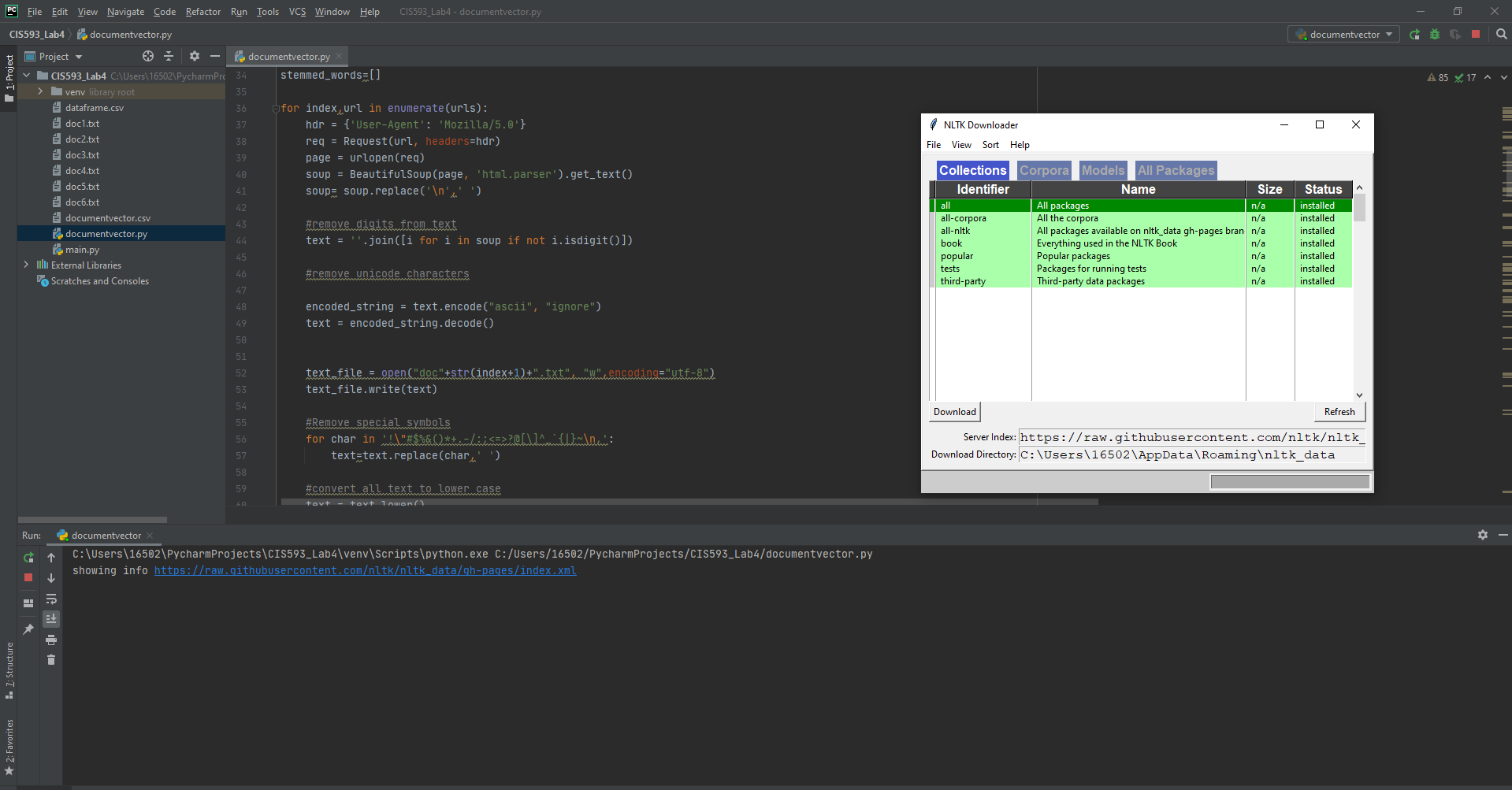
1. **Steps performed**
   1. Fetch content from all 6 given webpages
   2. Perform preprocessing of text:
2. Remove Unicode characters and digits
3. Remove special symbols
4. Convert tokens to lower case
5. Remove stop words
6. Perform stemming
7. Remove single quotes from token(This step has to be performed after removing stop words so that it doesn’t remove any stop words having ‘ just like in words don’t doesn’t ]
   1. Generate tokens of text fetched from webpages
   2. Count how many times each word occur in each document and generate TF-IDF considering to count only the words appeared in the Webpage Text as the Content of the page
   3. Look for given single word terms in TF-IDF for all documents(To search, I have searched for stemmed word as all tokens are already in stemmed term but in CSV , I have printed original term for display)
   4. Generate bigrams for all documents
   5. Look for given bigrams in all documents
   6. Store in a list and write it to csv as document vector
   7. For cosinesimilarity, read the columns from documentvector csv and calculate the consine similarity using formula from scipy library

1. **Files included**

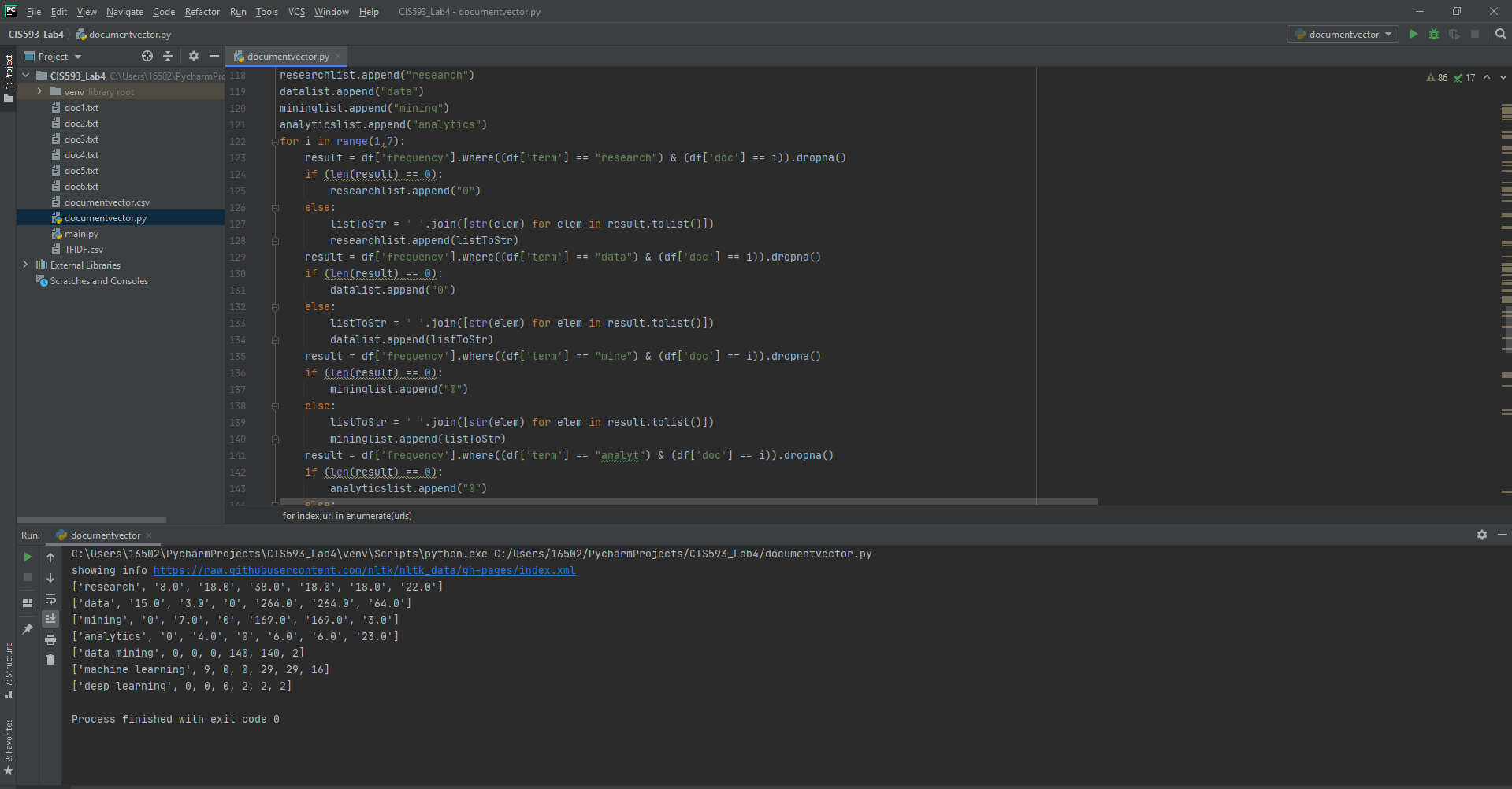
documentvector.py, TFIDF.csv. documentvector.csv, doc1.txt to doc6.txt, cosinesimilary.csv, cosinesimilarity.py

1. **Screenshots**

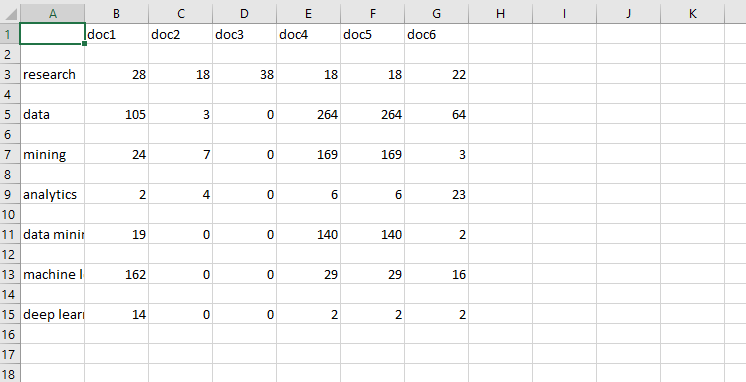
All NLTK collections are installed successfully and execution is started



Program is executed successfully

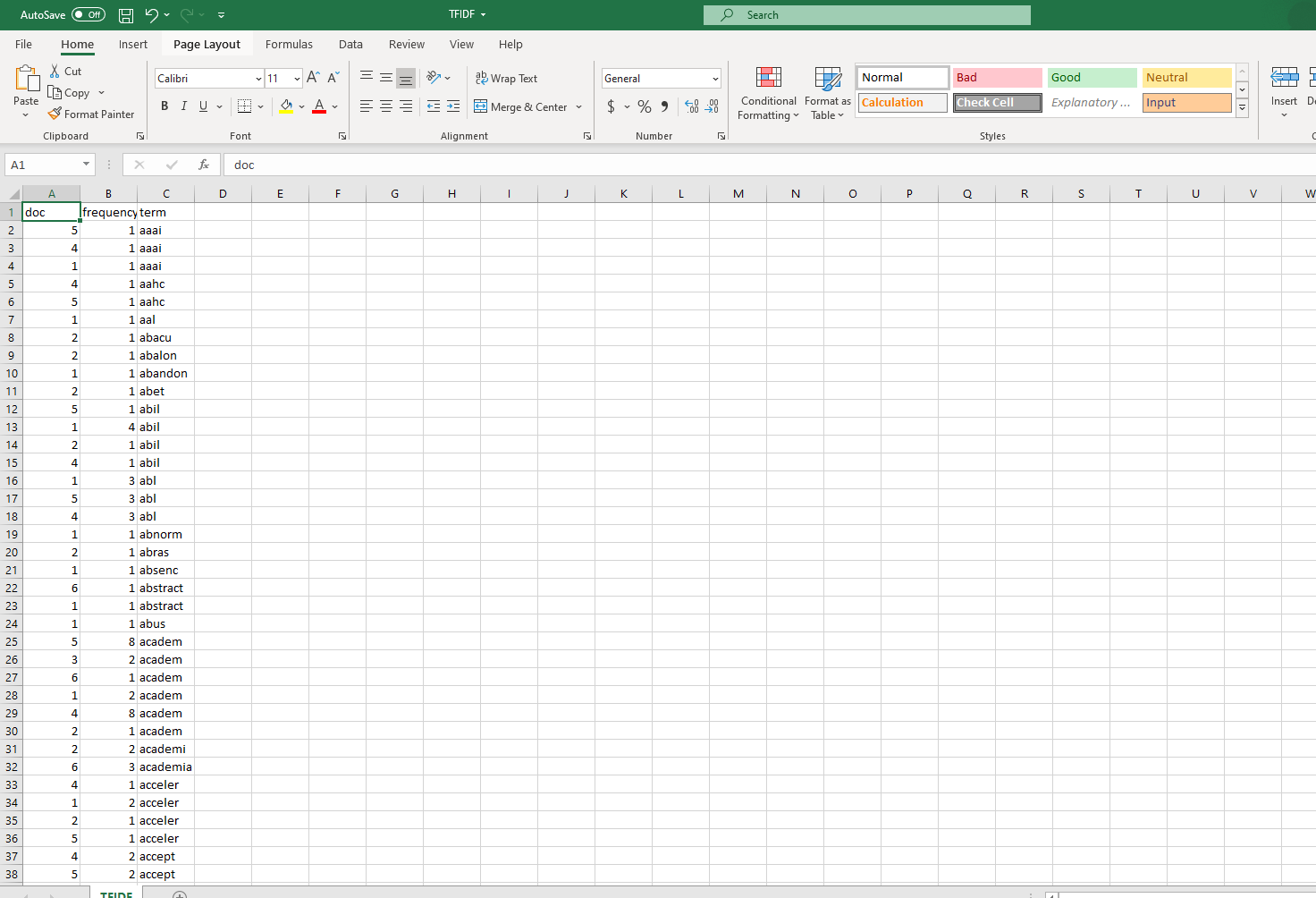


CSV file documentvector.csv showing the required output- document vector.

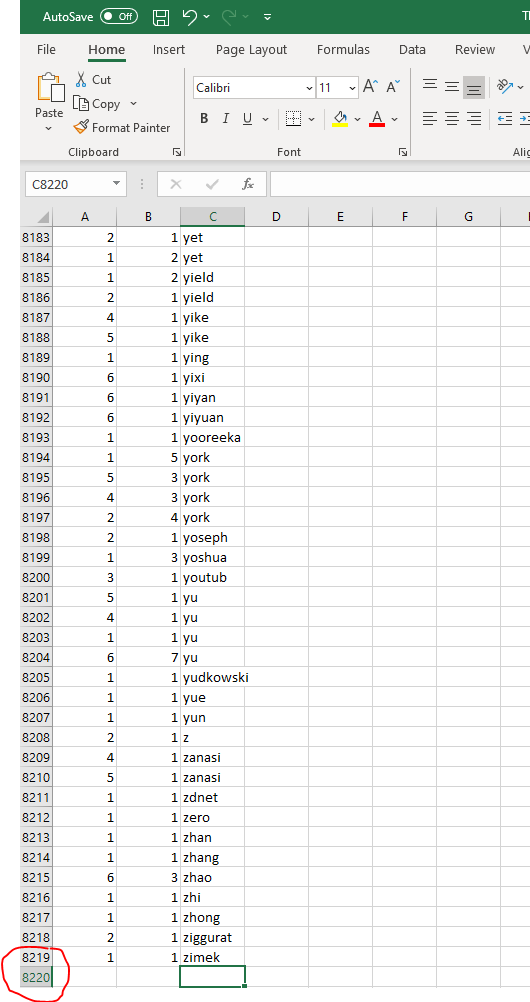


**Hence, from document vector- we can say that doc4 and doc5 which is basically a same webpage is more related to terms given.**

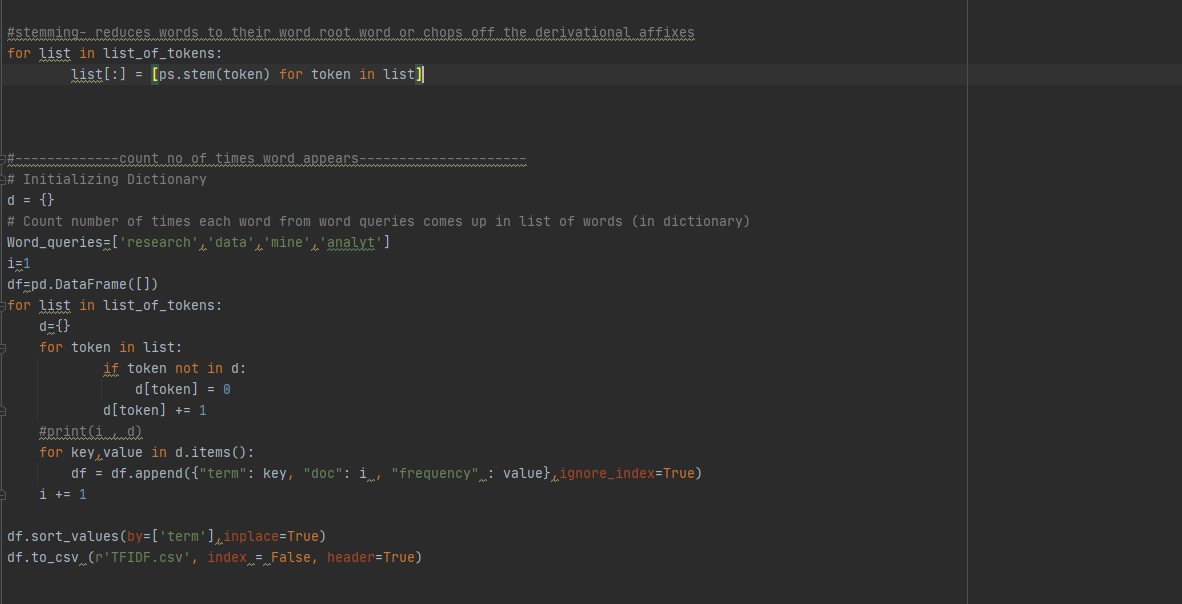
CSV file TFIDF.csv showing the inverted index – Term dictionary



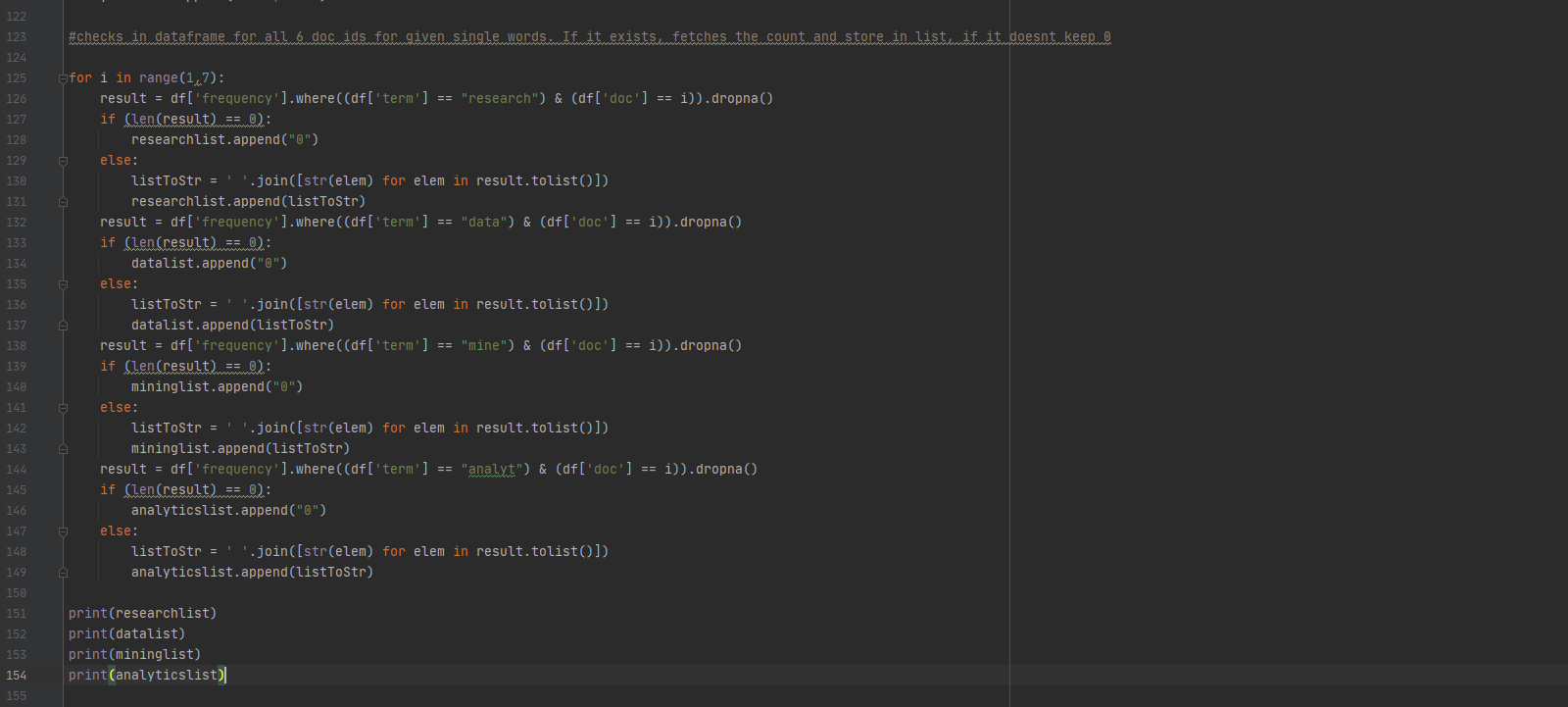
Term dictionary has 8220 records



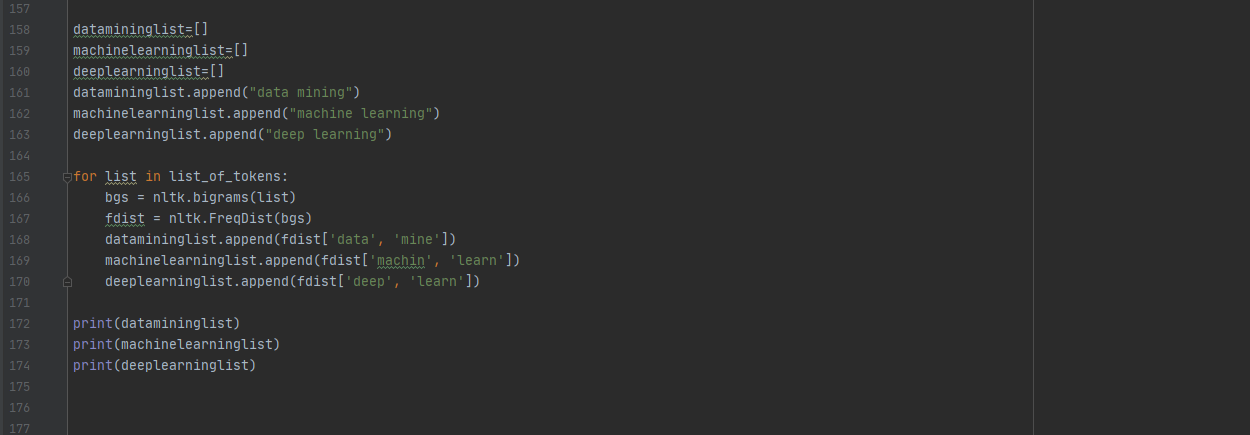
Main logic to generate inverted index



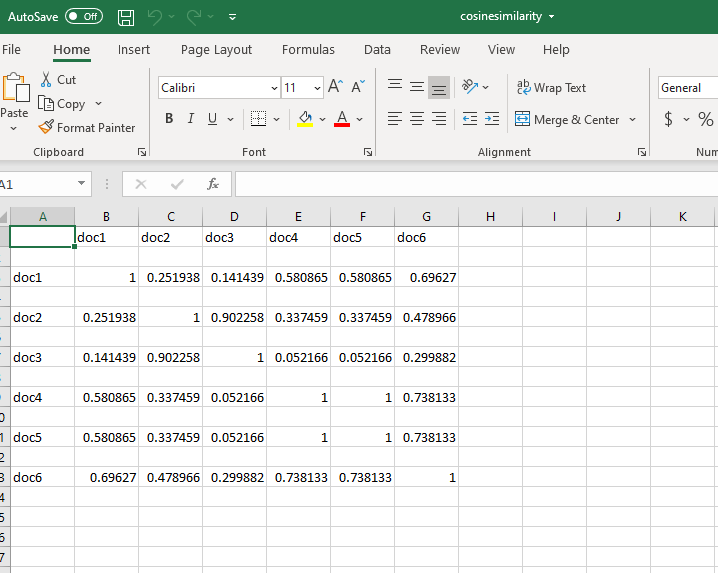
Main logic to look for given terms in generated dataframe[inverted index]

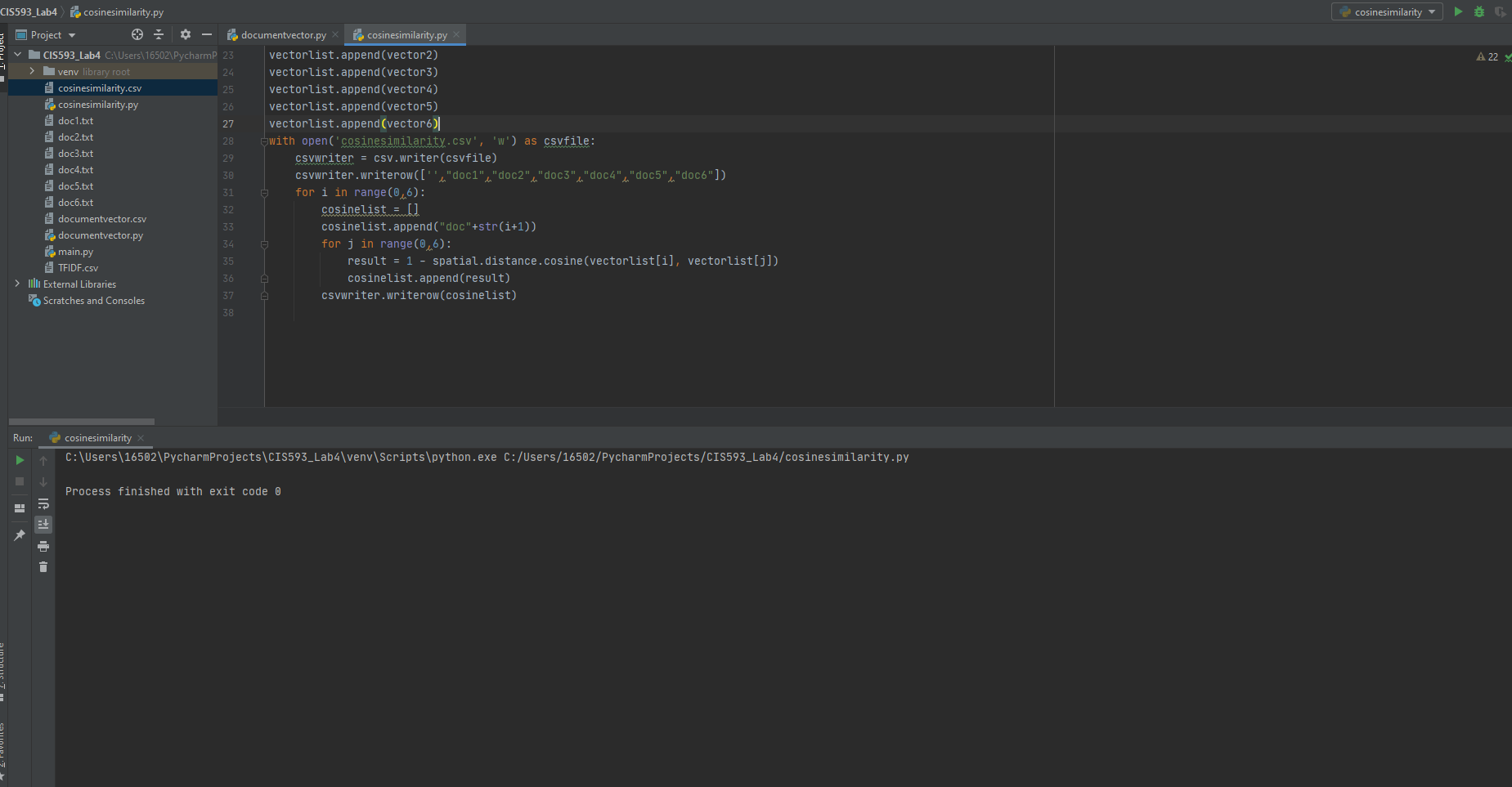


Main logic to generate bigrams and get count of given bigrams



**Cosine similarity Matrix**





1. **Cosine similarity Analysis and Discussion**

Cosine measure: If d1 and d2 are two vectors (e.g., term-frequency vectors), then cos(d1, d2) = (d1 • d2) /||d1|| ||d2|| , where • indicates vector dot product, ||d||: the length of vector d

• Discuss briefly about your topic analysis from your cosine similarity focusing on whether or not the indications by the values of your Cosine Sim are all correct?

Values of cosine similarity are correct as:

1. it gives the same values as calculating the cosine similarity manually using the method shown in ppt
2. all values are below 1
3. values are 1 for cosine similarity between two same documents

• Which 2 docs are most similar in terms of 7 given topics?

Document 4 and 5 are most similar in terms of 7 given topics

• The Topics of Doc6 is similar to the Topics of Doc 4 and 5? Explain Why or Why Not? If not, what are the reasons?

Topics of Doc6 is not similar to Topics of Doc 4 and Doc 5 as Doc 4 and Doc 5 has high values for Data Mining words [individual words or as bigrams] so we can say that those two docs are more about data mining.

1. **Code**

from bs4 import BeautifulSoup

from urllib.request import Request, urlopen

from nltk import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

import nltk

import csv

import pandas as pd

#nltk.download()

list1=[]

list2=[]

list3=[]

list4=[]

list5=[]

list6=[]

list\_of\_tokens=[]

list\_of\_tokens.append(list1)

list\_of\_tokens.append(list2)

list\_of\_tokens.append(list3)

list\_of\_tokens.append(list4)

list\_of\_tokens.append(list5)

list\_of\_tokens.append(list6)

bigram\_token\_list=[]

stop\_words=set(stopwords.words("english"))

list\_stopwords= list(stop\_words)

ps = PorterStemmer()

urls=['https://en.wikipedia.org/wiki/Machine\_learning’,'https://en.wikipedia.org/wiki/Engineering','https://my.clevelandclinic.org/research','https://en.wikipedia.org/wiki/Data\_mining','https://en.wikipedia.org/wiki/Data\_mining#Data\_mining','http://cis.csuohio.edu/~sschung/']

stemmed\_words=[]

for index,url in enumerate(urls):

hdr = {'User-Agent': 'Mozilla/5.0'}

req = Request(url, headers=hdr)

page = urlopen(req)

soup = BeautifulSoup(page, 'html.parser').get\_text()

soup= soup.replace('\n',' ')

#remove digits from text

text = ''.join([i for i in soup if not i.isdigit()])

#remove unicode characters

encoded\_string = text.encode("ascii", "ignore")

text = encoded\_string.decode()

text\_file = open("doc"+str(index+1)+".txt", "w",encoding="utf-8")

text\_file.write(text)

#Remove special symbols

for char in '!\"#$%&()\*+.-/:;<=>?@[\]^\_`{|}~\n,':

text=text.replace(char,' ')

#convert all text to lower case

text = text.lower()

#generate tokens

tokens = word\_tokenize(text)

for token in tokens:

list\_of\_tokens[index].append(token)

#remove stop words

for list in list\_of\_tokens:

for token in list:

if token in list\_stopwords:

while token in list:

list.remove(token)

#remove ' single quote

for list in list\_of\_tokens:

for token in list:

if "'" in token:

list.remove(token)

#stemming- reduces words to their word root word or chops off the derivational affixes

for list in list\_of\_tokens:

list[:] = [ps.stem(token) for token in list]

#-------------count no of times word appears---------------------

# Initializing Dictionary

d = {}

# Count number of times each word from word queries comes up in list of words (in dictionary)

Word\_queries=['research','data','mine','analyt']

i=1

df=pd.DataFrame([])

for list in list\_of\_tokens:

d={}

for token in list:

if token not in d:

d[token] = 0

d[token] += 1

#print(i , d)

for key,value in d.items():

df = df.append({"term": key, "doc": i , "frequency" : value},ignore\_index=True)

i += 1

df.sort\_values(by=['term'],inplace=True)

df.to\_csv (r'TFIDF.csv', index = False, header=True)

researchlist=[]

datalist=[]

mininglist=[]

analyticslist=[]

researchlist.append("research")

datalist.append("data")

mininglist.append("mining")

analyticslist.append("analytics")

#checks in dataframe for all 6 doc ids for given single words. If it exists, fetches the count and store in list, if it doesnt keep 0

for i in range(1,7):

result = df['frequency'].where((df['term'] == "research") & (df['doc'] == i)).dropna()

if (len(result) == 0):

researchlist.append("0")

else:

listToStr = ' '.join([str(elem) for elem in result.tolist()])

researchlist.append(listToStr)

result = df['frequency'].where((df['term'] == "data") & (df['doc'] == i)).dropna()

if (len(result) == 0):

datalist.append("0")

else:

listToStr = ' '.join([str(elem) for elem in result.tolist()])

datalist.append(listToStr)

result = df['frequency'].where((df['term'] == "mine") & (df['doc'] == i)).dropna()

if (len(result) == 0):

mininglist.append("0")

else:

listToStr = ' '.join([str(elem) for elem in result.tolist()])

mininglist.append(listToStr)

result = df['frequency'].where((df['term'] == "analyt") & (df['doc'] == i)).dropna()

if (len(result) == 0):

analyticslist.append("0")

else:

listToStr = ' '.join([str(elem) for elem in result.tolist()])

analyticslist.append(listToStr)

'''

print(researchlist)

print(datalist)

print(mininglist)

print(analyticslist)'''

#generate bigrams with frequency distributions and look for given bigrams in generated list

datamininglist=[]

machinelearninglist=[]

deeplearninglist=[]

datamininglist.append("data mining")

machinelearninglist.append("machine learning")

deeplearninglist.append("deep learning")

for list in list\_of\_tokens:

bgs = nltk.bigrams(list)

fdist = nltk.FreqDist(bgs)

datamininglist.append(fdist['data', 'mine'])

machinelearninglist.append(fdist['machin', 'learn'])

deeplearninglist.append(fdist['deep', 'learn'])

'''

print(datamininglist)

print(machinelearninglist)

print(deeplearninglist)

'''

#write to csv

listofdocs=['',"doc1","doc2","doc3","doc4","doc5","doc6"]

with open('documentvector.csv', 'w') as csvfile:

# creating a csv writer object

csvwriter = csv.writer(csvfile)

csvwriter.writerow(listofdocs)

csvwriter.writerow(researchlist)

csvwriter.writerow(datalist)

csvwriter.writerow(mininglist)

csvwriter.writerow(analyticslist)

csvwriter.writerow(datamininglist)

csvwriter.writerow(machinelearninglist)

csvwriter.writerow(deeplearninglist)

**cosinesimilary.py**

from scipy import spatial

import pandas as pd

import csv

vector1=[]

vector2=[]

vector3=[]

vector4=[]

vector5=[]

vector6=[]

col\_list = ["doc1","doc2","doc3","doc4","doc5","doc6"]

df = pd.read\_csv("documentvector.csv", usecols=col\_list)

vector1.append(df["doc1"].values)

vector2.append(df["doc2"].values)

vector3.append(df["doc3"].values)

vector4.append(df["doc4"].values)

vector5.append(df["doc5"].values)

vector6.append(df["doc6"].values)

vectorlist=[]

vectorlist.append(vector1)

vectorlist.append(vector2)

vectorlist.append(vector3)

vectorlist.append(vector4)

vectorlist.append(vector5)

vectorlist.append(vector6)

with open('cosinesimilarity.csv', 'w') as csvfile:

csvwriter = csv.writer(csvfile)

csvwriter.writerow(['',"doc1","doc2","doc3","doc4","doc5","doc6"])

for i in range(0,6):

cosinelist = []

cosinelist.append("doc"+str(i+1))

for j in range(0,6):

result = 1 - spatial.distance.cosine(vectorlist[i], vectorlist[j])

cosinelist.append(result)

csvwriter.writerow(cosinelist)