19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

Q1

a)

Code:

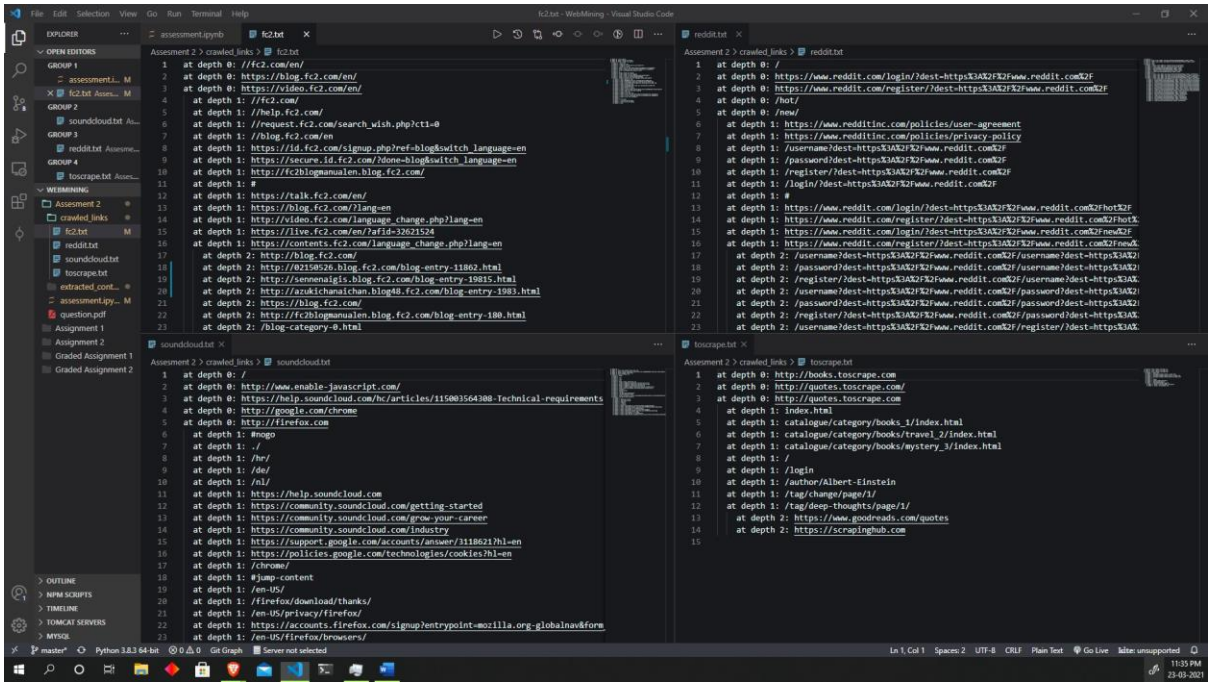
```
# bfs concurrency
import requests
from bs4 import BeautifulSoup
from collections import deque
import re
import concurrent.futures
from urllib import request

def bfs(url):
    visited = set([url])
    dq = deque([[url, "", 0]])
    max_depth = 3
    max_breadth = 5
    string = ""
    return_string =""
    while dq:
        base, path, depth = dq.popleft()
        if depth < max_depth:
            try:
                soup = BeautifulSoup(requests.get(base + path).text, "html.par
ser")
                links = soup.find_all("a")[:max_breadth]
                for link in links:
                    href = link.get("href")
                    if href not in visited:
                        visited.add(href)
                        string += ("  " * depth + f"at depth {depth}: {href}\n
")
                        return_string += (href)+' '

                        if href.startswith("http") or href.startswith("https")
:
                            dq.append([href, "", depth + 1])
                        else:
                            dq.append([base, href, depth + 1])
            except:
                pass

    filename =""
    if "https://" in url:
        filename = re.sub(r"https://","",url)
        filename = re.sub(r".com","",filename)
    elif "http://" in url:
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

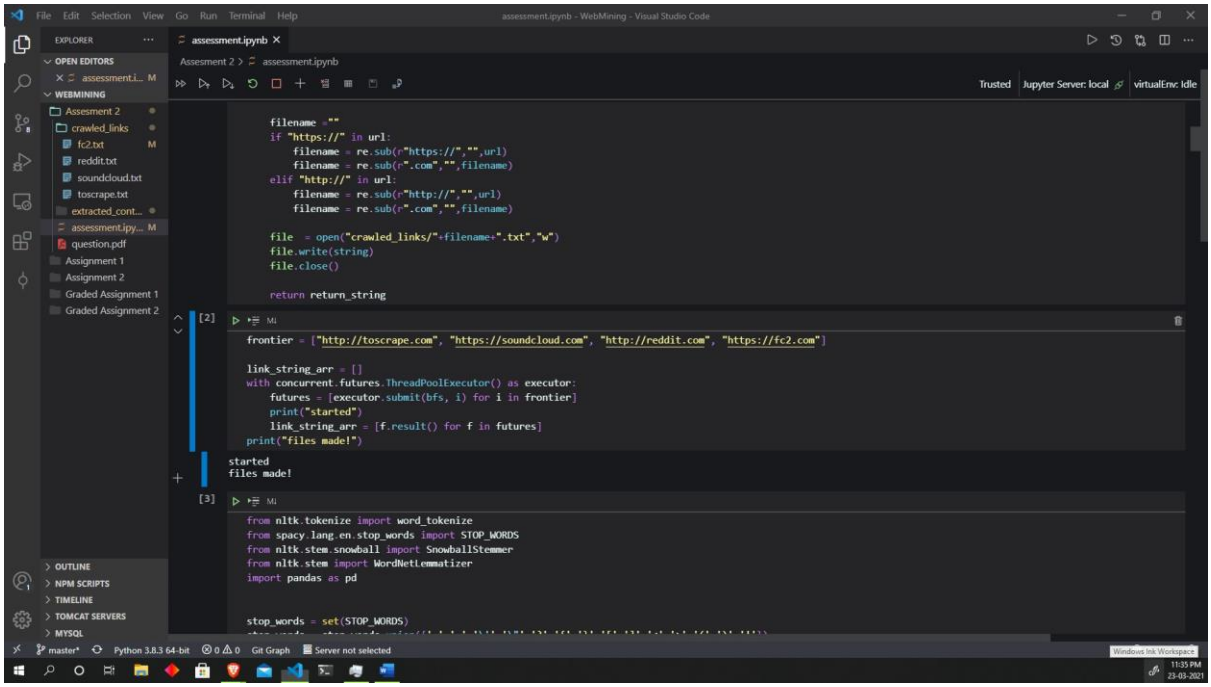
```
        filename = re.sub(r"http://","",url)
        filename = re.sub(r".com","",filename)

    file  = open("crawled_links/"+filename+".txt","w")
    file.write(string)
    file.close()

    return return_string
```

```
frontier = ["http://toscrape.com", "https://soundcloud.com", "http://reddit.co
m", "https://fc2.com"]

link_string_arr = []
with concurrent.futures.ThreadPoolExecutor() as executor:
    futures = [executor.submit(bfs, i) for i in frontier]
    print("started")
    link_string_arr = [f.result() for f in futures]
print("files made!")
```

output:



19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2



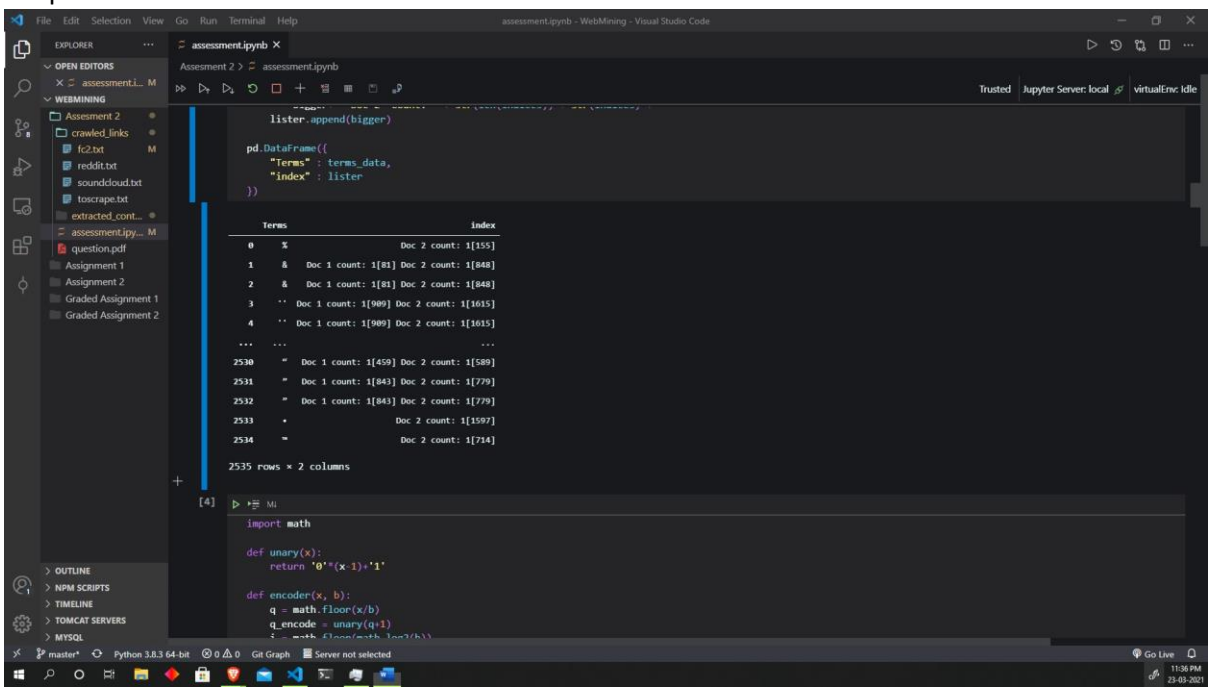
b), c)

```
from nltk.tokenize import word_tokenize
from spacy.lang.en.stop_words import STOP_WORDS
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
import pandas as pd


stop_words = set(STOP_WORDS)
stop_words = stop_words.union({'.',',','\'','\"','?','{','}','[',']','<','>','
(',')','!'})

stemmer = SnowballStemmer(language='english')
lemmatizer = WordNetLemmatizer()

base = 'https://en.wikipedia.org/wiki/'
links = ['Web_mining','Data_mining']
terms_searched = []
# assuming these are the links crawled by the crawler

for i in links:
    url = base + i
    html = request.urlopen(url).read().decode('utf8')
    data = list(set(word_tokenize(BeautifulSoup(html, 'html.parser').get_text(
))).difference(stop_words))
    # data = set(word_tokenize((BeautifulSoup(requests.urlopen(url).read().dec
ode('utf8')), 'html.parser').get_text())).difference(stop_words)
    stemmed = list(set([stemmer.stem(w) for w in data]))
```

```
    # lemmatized = list(set([lemmatizer.lemmatize(w) for w in data]))
    terms_searched.append(stemmed)

terms_data = []
for i in terms_searched:
    for j in i:
        terms_data.append(j)

terms_data.sort()

lister = []
for i in terms_data:
    bigger = ''
    if i in terms_searched[0]:
        indices = []
        for j in range(len(terms_searched[0])):
            if terms_searched[0][j] == i:
                indices.append(j)
        bigger+= 'Doc 1  count: ' + str(len(indices)) + str(indices) + ' '
    if i in terms_searched[1]:
        indices = []
        for j in range(len(terms_searched[1])):
            if terms_searched[1][j] == i:
                indices.append(j)
        bigger+= 'Doc 2  count: ' + str(len(indices)) + str(indices) + ' '
    lister.append(bigger)

pd.DataFrame({
    "Terms" : terms_data,
    "index" : lister
})
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

Output:

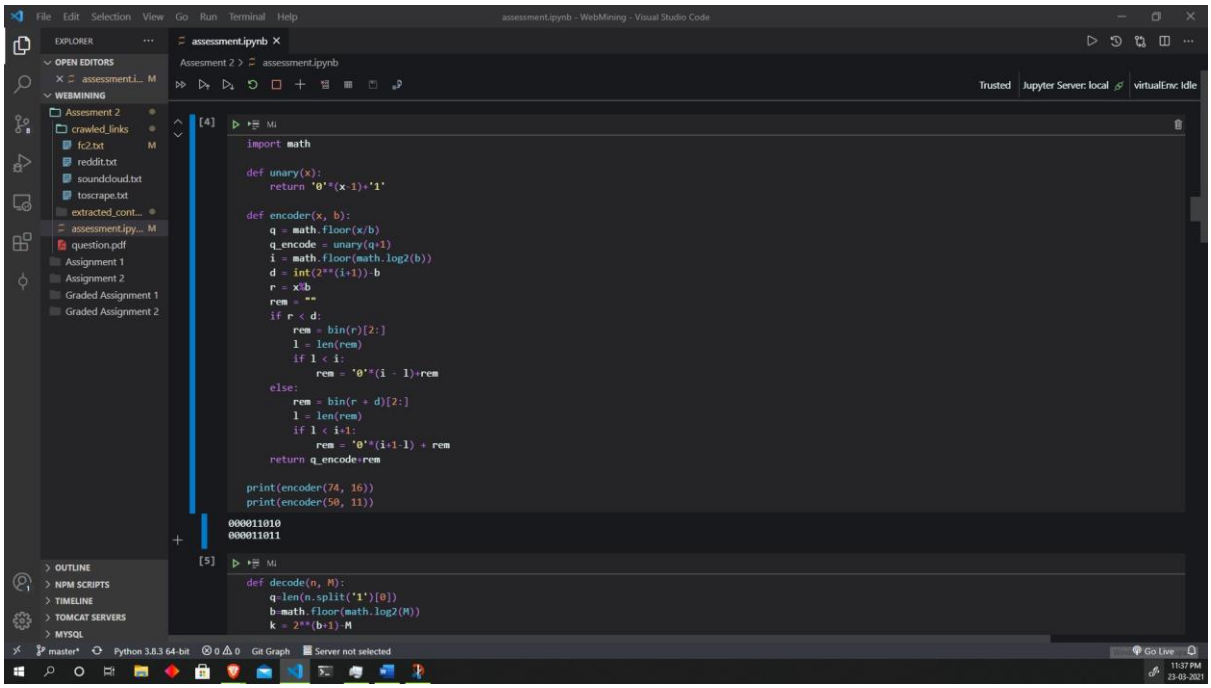


Q2

```
import math

def unary(x):
    return '0'*(x-1)+'1'

def encoder(x, b):
    q = math.floor(x/b)
    q_encode = unary(q+1)
    i = math.floor(math.log2(b))
    d = int(2**(i+1))-b
    r = x%b
    rem = ""
    if r < d:
        rem = bin(r)[2:]
        l = len(rem)
        if l < i:
            rem = '0'*(i - l)+rem
    else:
        rem = bin(r + d)[2:]
        l = len(rem)
        if l < i+1:
            rem = '0'*(i+1-l) + rem
    return q_encode+rem

print(encoder(74, 16))
print(encoder(50, 11))
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

output:

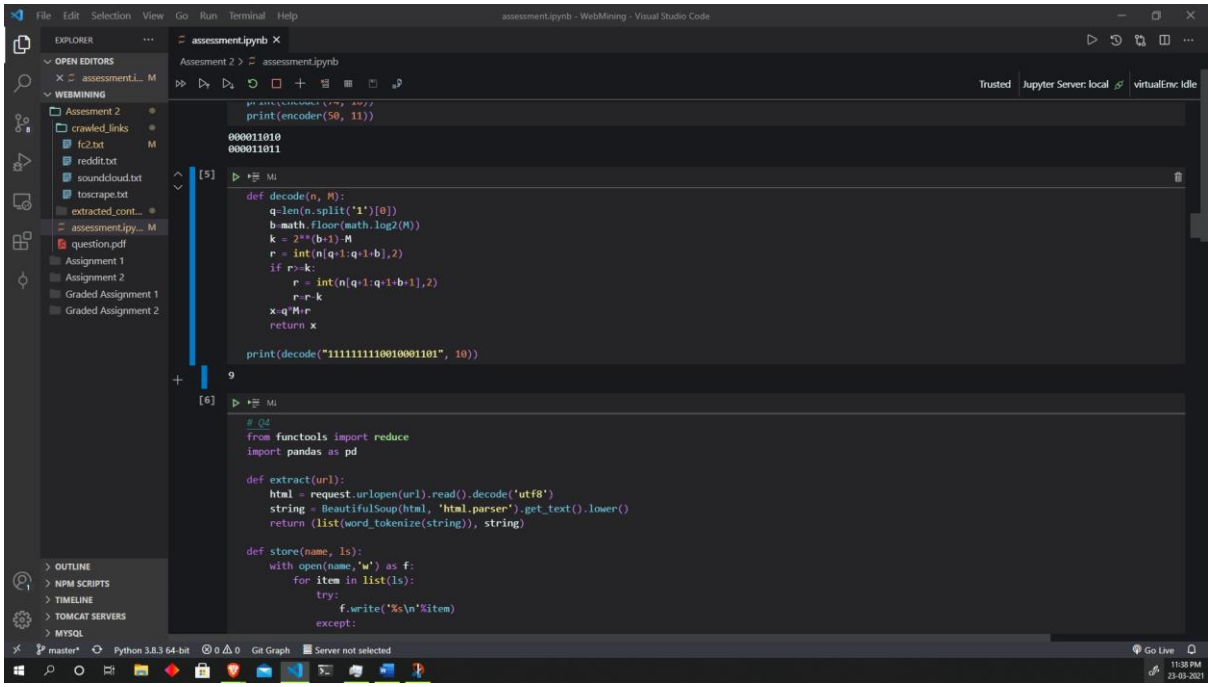


b)

```
def decode(n, M):
    q=len(n.split('1')[0])
    b=math.floor(math.log2(M))
    k = 2**(b+1)-M
    r = int(n[q+1:q+1+b],2)
    if r>=k:
        r = int(n[q+1:q+1+b+1],2)
        r=r-k
    x=q*M+r
    return x

print(decode("11111111010001101", 10))
```

output:

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

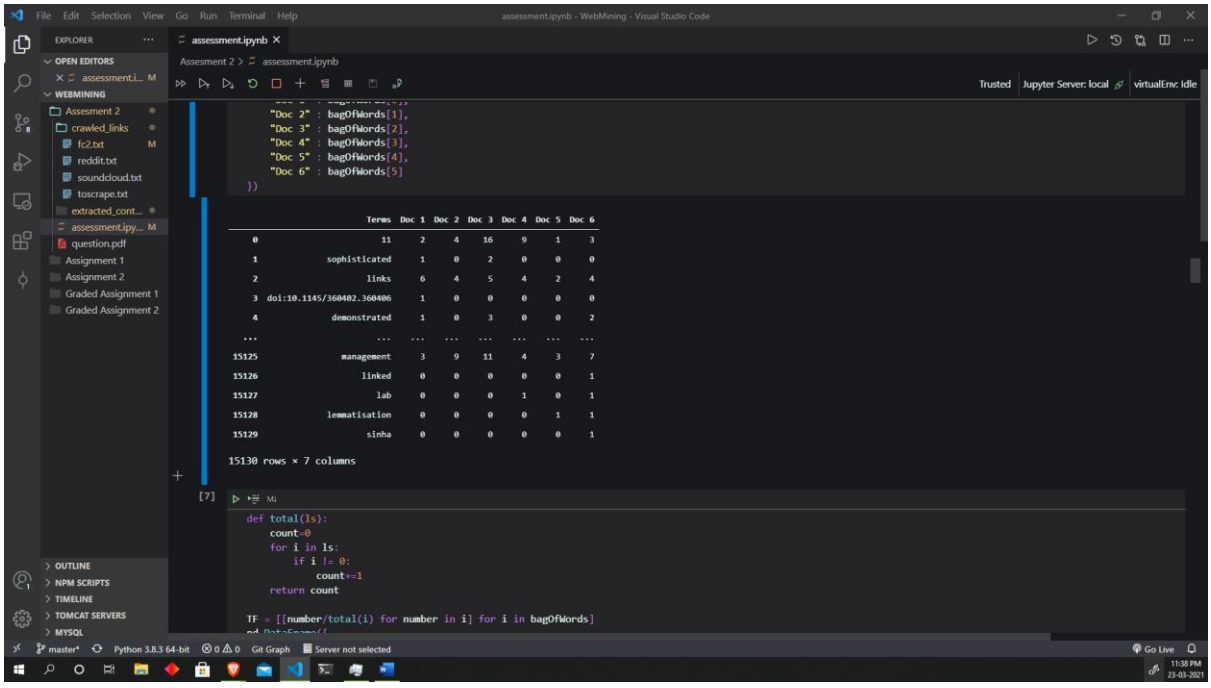


Q4

```
# Q4
from functools import reduce
import pandas as pd

def extract(url):
    html = request.urlopen(url).read().decode('utf8')
    string = BeautifulSoup(html, 'html.parser').get_text().lower()
    return (list(word_tokenize(string)), string)

def store(name, ls):
    with open(name,'w') as f:
        for item in list(ls):
            try:
                f.write('%s\n'%item)
            except:
                pass
def term_maker(ls, filenames):
    terms = []
    index = 0
    for i in ls:
        st = set(i).difference(stop_words)
        store('extracted_content/'+filenames[index]+'.txt', list(st))
        index += 1
        terms.append(list(st))
    terms = reduce(lambda z,y : z+y, terms)
    return terms
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

```
def counter(tokens, terms):
    ls = []
    for i in terms:
        ls.append(tokens.count(i))
    return ls

base = 'https://en.wikipedia.org/wiki/'
links = ['Web_mining','Data_mining','Artificial_intelligence','Machine_learnin
g','Natural_language_processing','Text_mining']
token_list = []
for i in links:
    token_list.append(extract(base+i)[0])
terms = term_maker(token_list, links)
bagOfWords = []
for i in token_list:
    bagOfWords.append(counter(i,terms))

pd.DataFrame({
    "Terms" : list(terms),
    "Doc 1" : bagOfWords[0],
    "Doc 2" : bagOfWords[1],
    "Doc 3" : bagOfWords[2],
    "Doc 4" : bagOfWords[3],
    "Doc 5" : bagOfWords[4],
    "Doc 6" : bagOfWords[5]
})
```

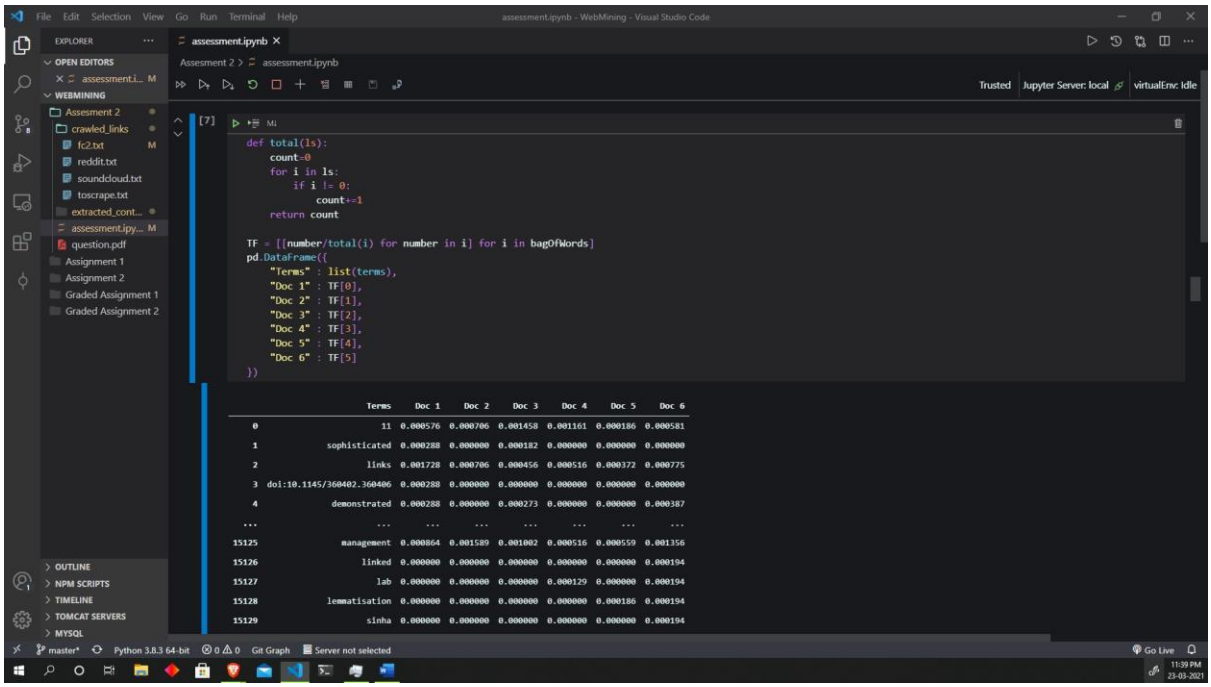



```
def total(ls):
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

```
    count=0
    for i in ls:
        if i != 0:
            count+=1
    return count

TF = [[number/total(i) for number in i] for i in bagOfWords]
pd.DataFrame({
    "Terms" : list(terms),
    "Doc 1" : TF[0],
    "Doc 2" : TF[1],
    "Doc 3" : TF[2],
    "Doc 4" : TF[3],
    "Doc 5" : TF[4],
    "Doc 6" : TF[5]
})
```

Output:

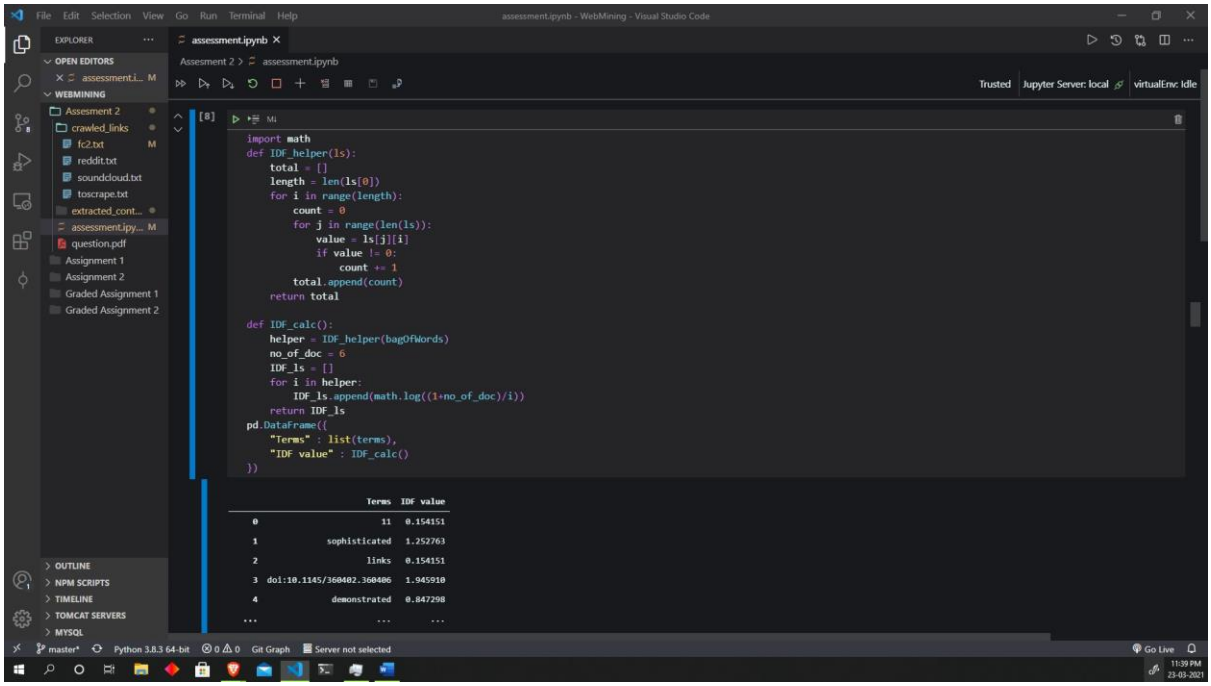


	Terms	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
0	11	0.000576	0.000706	0.001458	0.001161	0.000186	0.000581
1	sophisticated	0.000288	0.000000	0.000182	0.000000	0.000000	0.000000
2	links	0.001728	0.000706	0.000456	0.000516	0.000372	0.000775
3	doi:10.1145/360402.360406	0.000288	0.000000	0.000000	0.000000	0.000000	0.000000
4	demonstrated	0.000288	0.000000	0.000273	0.000000	0.000000	0.000387
...	...	...	...	...	...	...	...
15125	management	0.000864	0.001589	0.001002	0.000516	0.000559	0.001356
15126	linked	0.000000	0.000000	0.000000	0.000000	0.000000	0.000194
15127	lab	0.000000	0.000000	0.000000	0.000129	0.000000	0.000194
15128	lemmatisation	0.000000	0.000000	0.000000	0.000000	0.000186	0.000194
15129	sinha	0.000000	0.000000	0.000000	0.000000	0.000000	0.000194

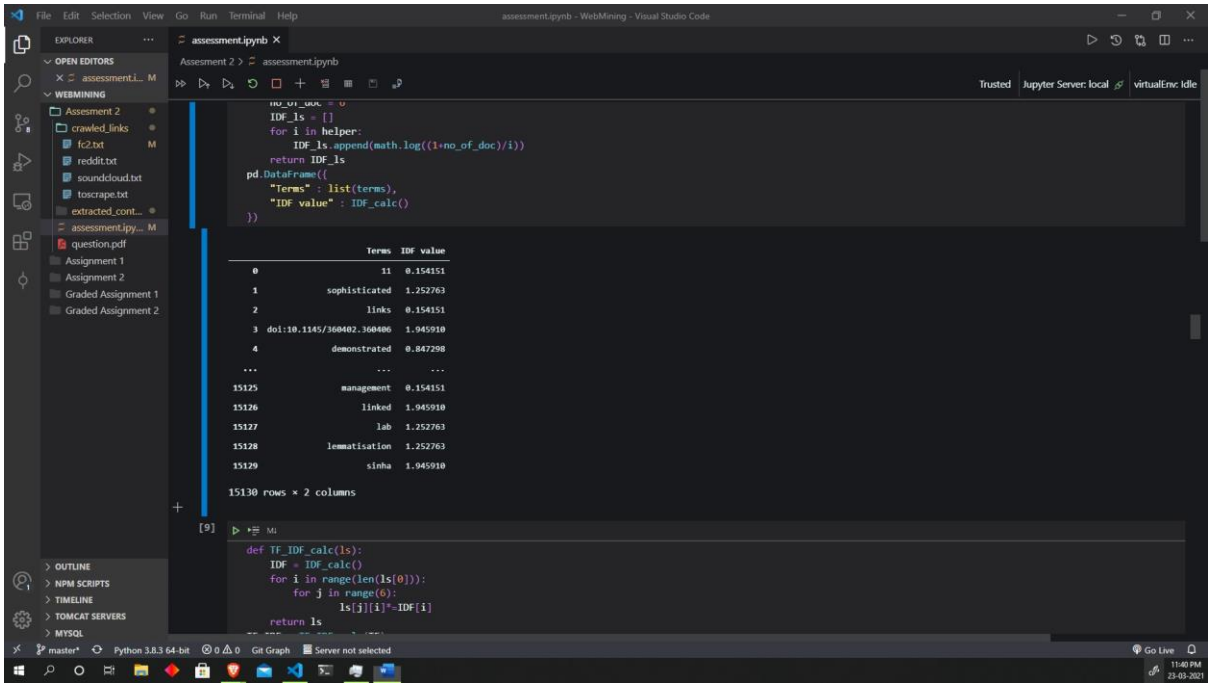
```
import math
def IDF_helper(ls):
    total = []
    length = len(ls[0])
    for i in range(length):
        count = 0
        for j in range(len(ls)):
            value = ls[j][i]
            if value != 0:
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

```
                count += 1
        total.append(count)
    return total

def IDF_calc():
    helper = IDF_helper(bagOfWords)
    no_of_doc = 6
    IDF_ls = []
    for i in helper:
        IDF_ls.append(math.log((1+no_of_doc)/i))
    return IDF_ls
pd.DataFrame({
    "Terms" : list(terms),
    "IDF value" : IDF_calc()
})
```

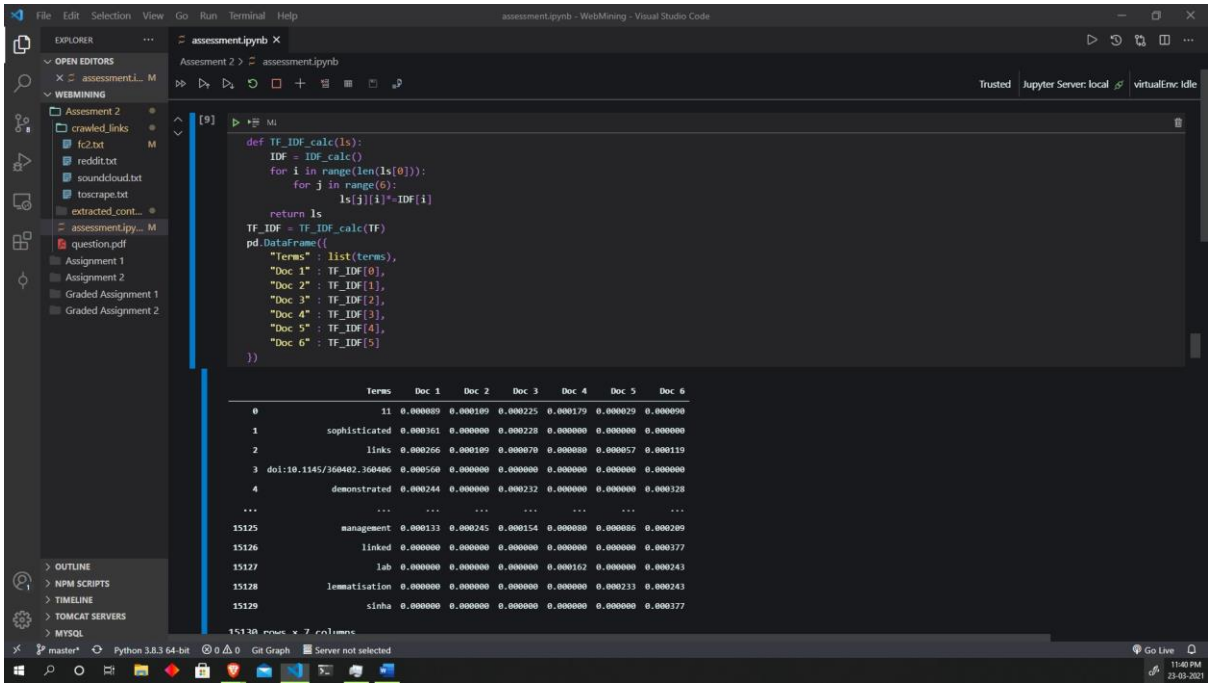


19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2



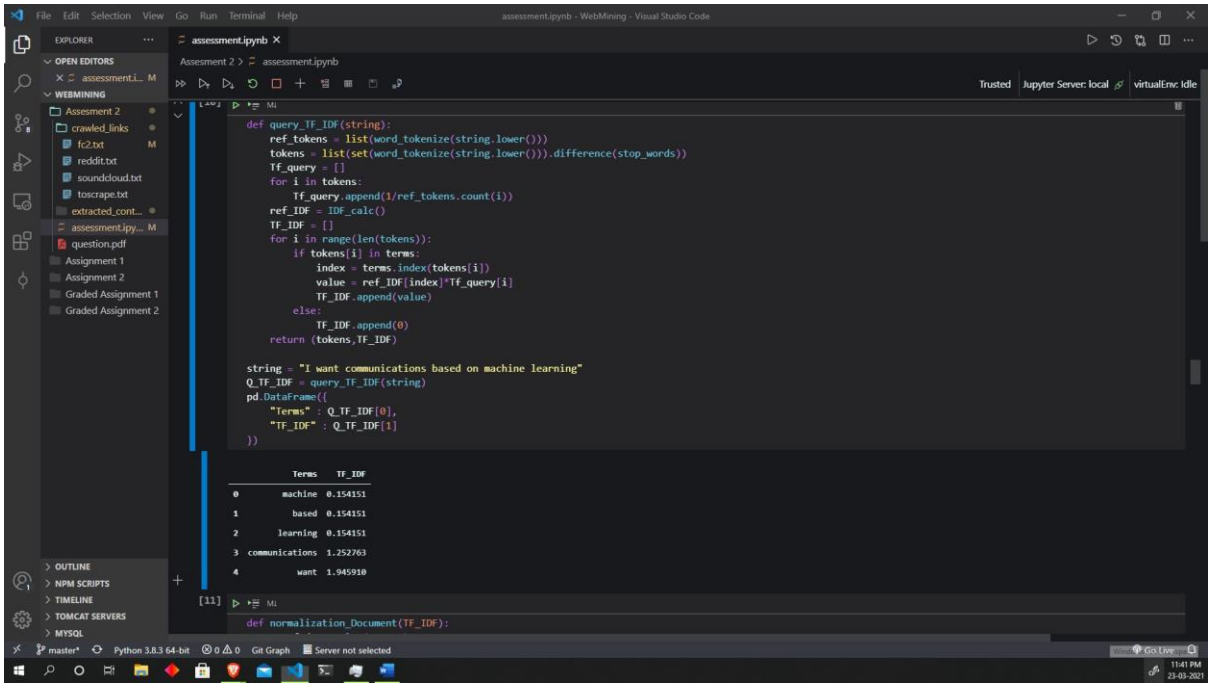

```
def TF_IDF_calc(ls):
    IDF = IDF_calc()
    for i in range(len(ls[0])):
        for j in range(6):
                ls[j][i]*=IDF[i]
    return ls
TF_IDF = TF_IDF_calc(TF)
pd.DataFrame({
    "Terms" : list(terms),
    "Doc 1" : TF_IDF[0],
    "Doc 2" : TF_IDF[1],
    "Doc 3" : TF_IDF[2],
    "Doc 4" : TF_IDF[3],
    "Doc 5" : TF_IDF[4],
    "Doc 6" : TF_IDF[5]
})
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2



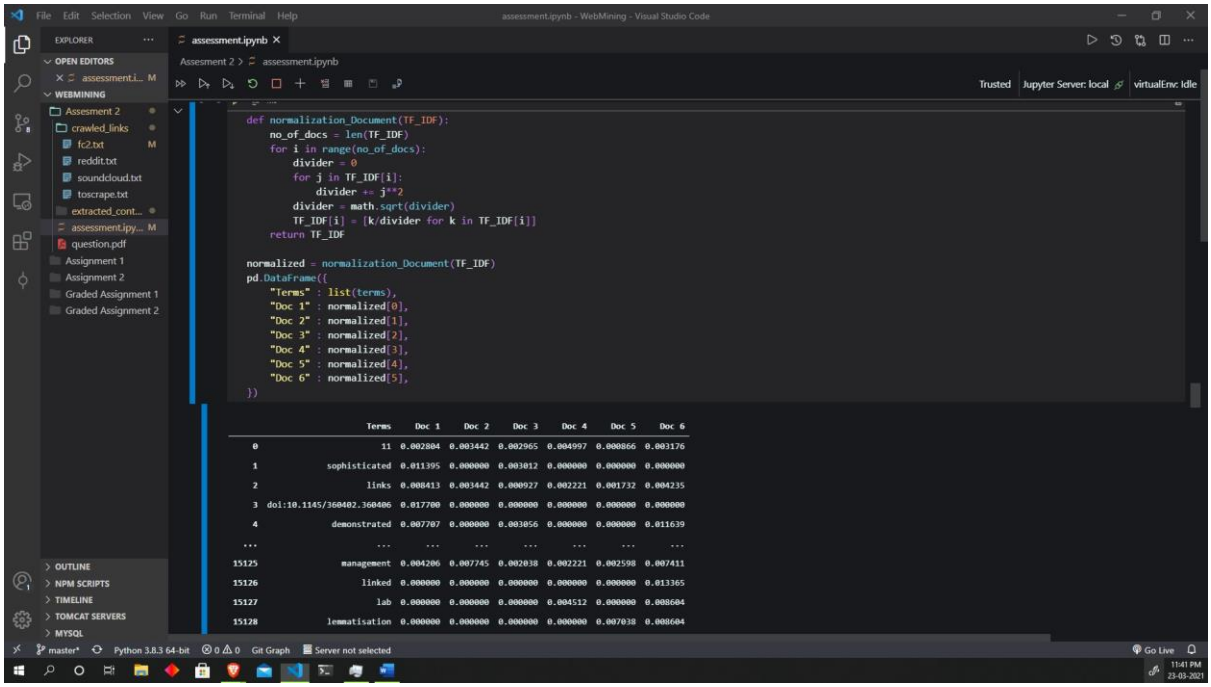

```
def query_TF_IDF(string):
    ref_tokens = list(word_tokenize(string.lower()))
    tokens = list(set(word_tokenize(string.lower())).difference(stop_words))
    Tf_query = []
    for i in tokens:
        Tf_query.append(1/ref_tokens.count(i))
    ref_IDF = IDF_calc()
    TF_IDF = []
    for i in range(len(tokens)):
        if tokens[i] in terms:
            index = terms.index(tokens[i])
            value = ref_IDF[index]*Tf_query[i]
            TF_IDF.append(value)
        else:
            TF_IDF.append(0)
    return (tokens,TF_IDF)

string = "I want communications based on machine learning"
Q_TF_IDF = query_TF_IDF(string)
pd.DataFrame({
    "Terms" : Q_TF_IDF[0],
    "TF_IDF" : Q_TF_IDF[1]
})
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2



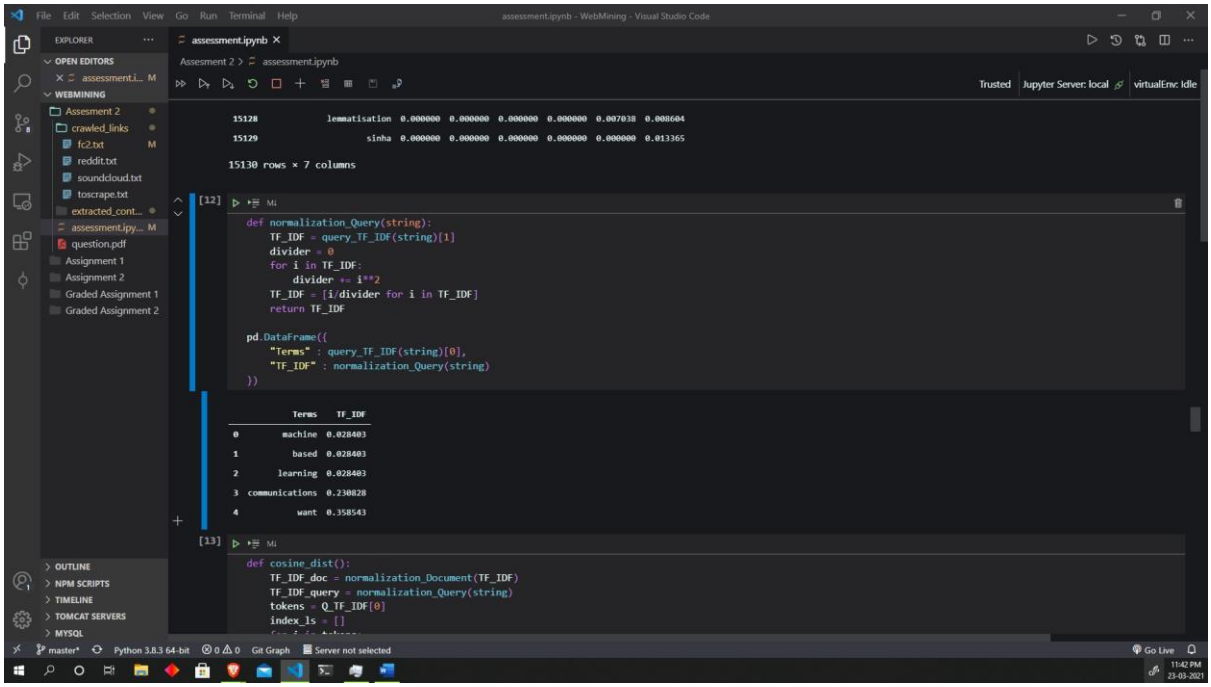

```
def normalization_Document(TF_IDF):
    no_of_docs = len(TF_IDF)
    for i in range(no_of_docs):
        divider = 0
        for j in TF_IDF[i]:
            divider += j**2
        divider = math.sqrt(divider)
        TF_IDF[i] = [k/divider for k in TF_IDF[i]]
    return TF_IDF

normalized = normalization_Document(TF_IDF)
pd.DataFrame({
    "Terms" : list(terms),
    "Doc 1" : normalized[0],
    "Doc 2" : normalized[1],
    "Doc 3" : normalized[2],
    "Doc 4" : normalized[3],
    "Doc 5" : normalized[4],
    "Doc 6" : normalized[5],
})
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2



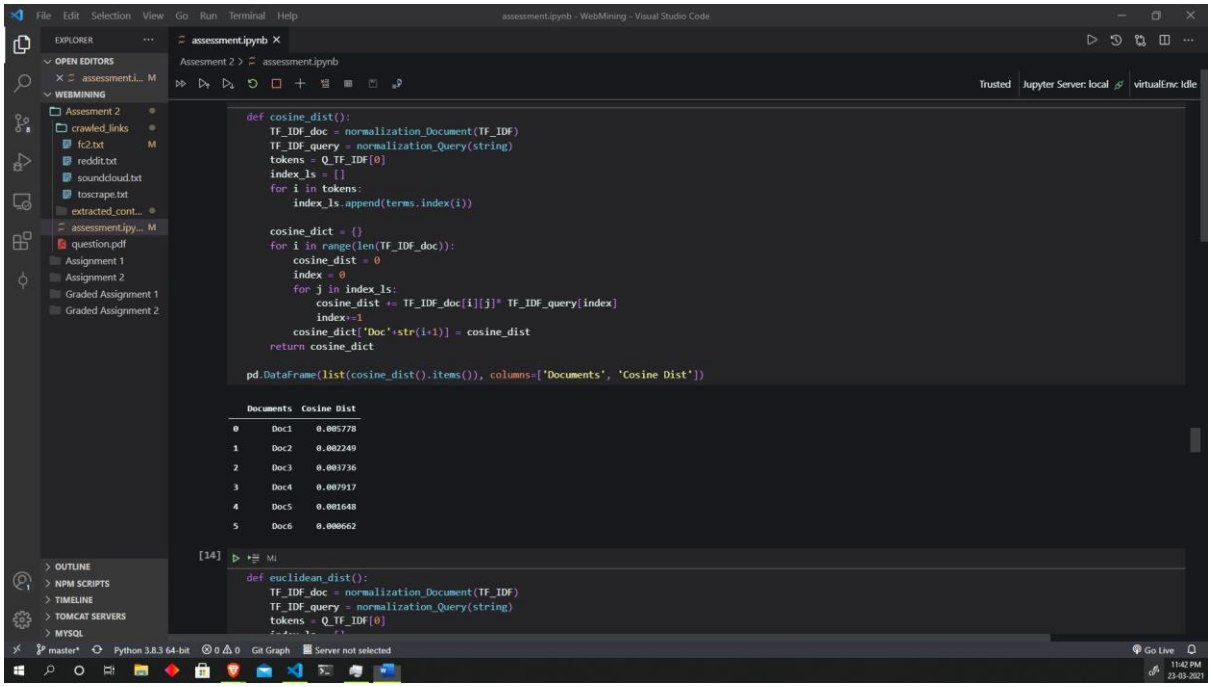

```
def normalization_Query(string):
    TF_IDF = query_TF_IDF(string)[1]
    divider = 0
    for i in TF_IDF:
        divider += i**2
    TF_IDF = [i/divider for i in TF_IDF]
    return TF_IDF

pd.DataFrame({
    "Terms" : query_TF_IDF(string)[0],
    "TF_IDF" : normalization_Query(string)
})
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2



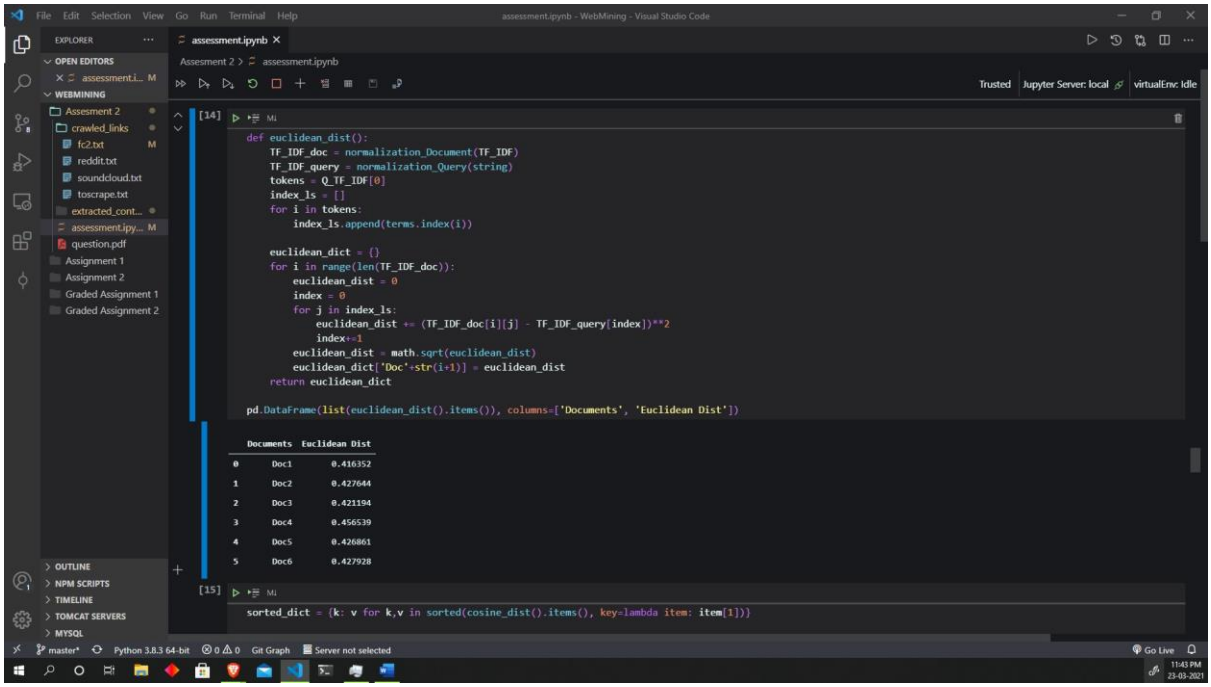

```
def cosine_dist():
    TF_IDF_doc = normalization_Document(TF_IDF)
    TF_IDF_query = normalization_Query(string)
    tokens = Q_TF_IDF[0]
    index_ls = []
    for i in tokens:
        index_ls.append(terms.index(i))

    cosine_dict = {}
    for i in range(len(TF_IDF_doc)):
        cosine_dist = 0
        index = 0
        for j in index_ls:
            cosine_dist += TF_IDF_doc[i][j]* TF_IDF_query[index]
            index+=1
        cosine_dict['Doc'+str(i+1)] = cosine_dist
    return cosine_dict

pd.DataFrame(list(cosine_dist().items()), columns=['Documents', 'Cosine Dist']
)
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2



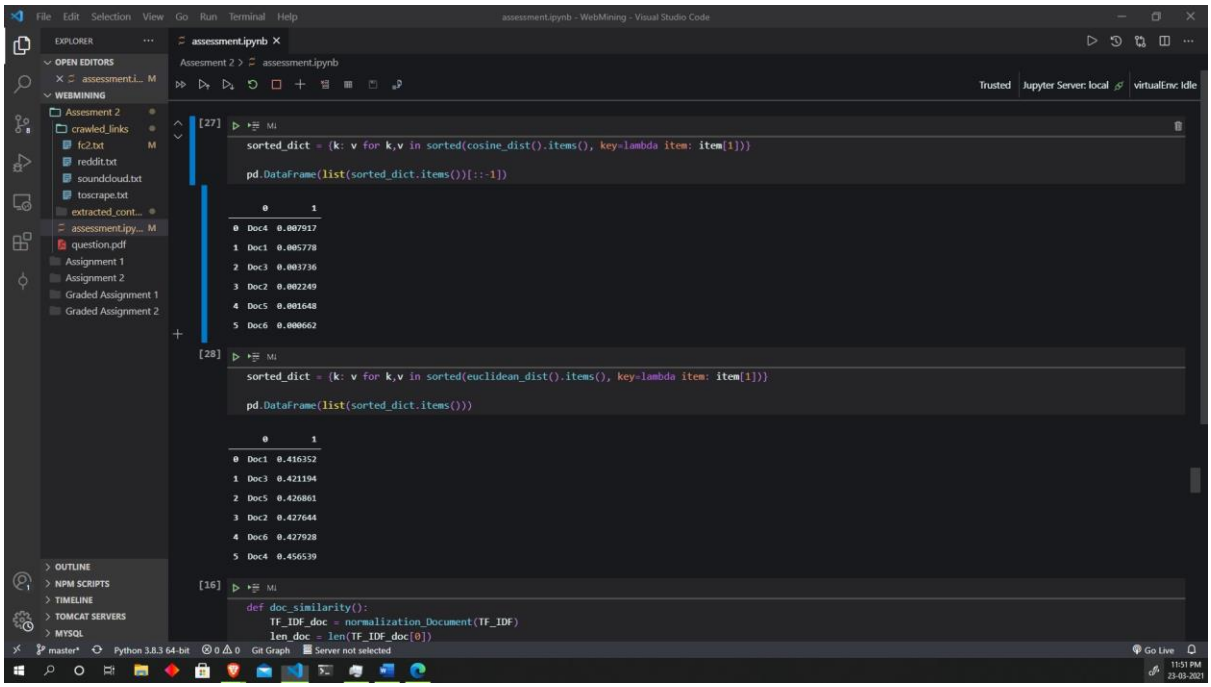

```
def euclidean_dist():
    TF_IDF_doc = normalization_Document(TF_IDF)
    TF_IDF_query = normalization_Query(string)
    tokens = Q_TF_IDF[0]
    index_ls = []
    for i in tokens:
        index_ls.append(terms.index(i))

    euclidean_dict = {}
    for i in range(len(TF_IDF_doc)):
        euclidean_dist = 0
        index = 0
        for j in index_ls:
            euclidean_dist += (TF_IDF_doc[i][j] - TF_IDF_query[index])**2
            index+=1
        euclidean_dist = math.sqrt(euclidean_dist)
        euclidean_dict['Doc'+str(i+1)] = euclidean_dist
    return euclidean_dict

pd.DataFrame(list(euclidean_dist().items()), columns=['Documents', 'Euclidean Dist'])
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2



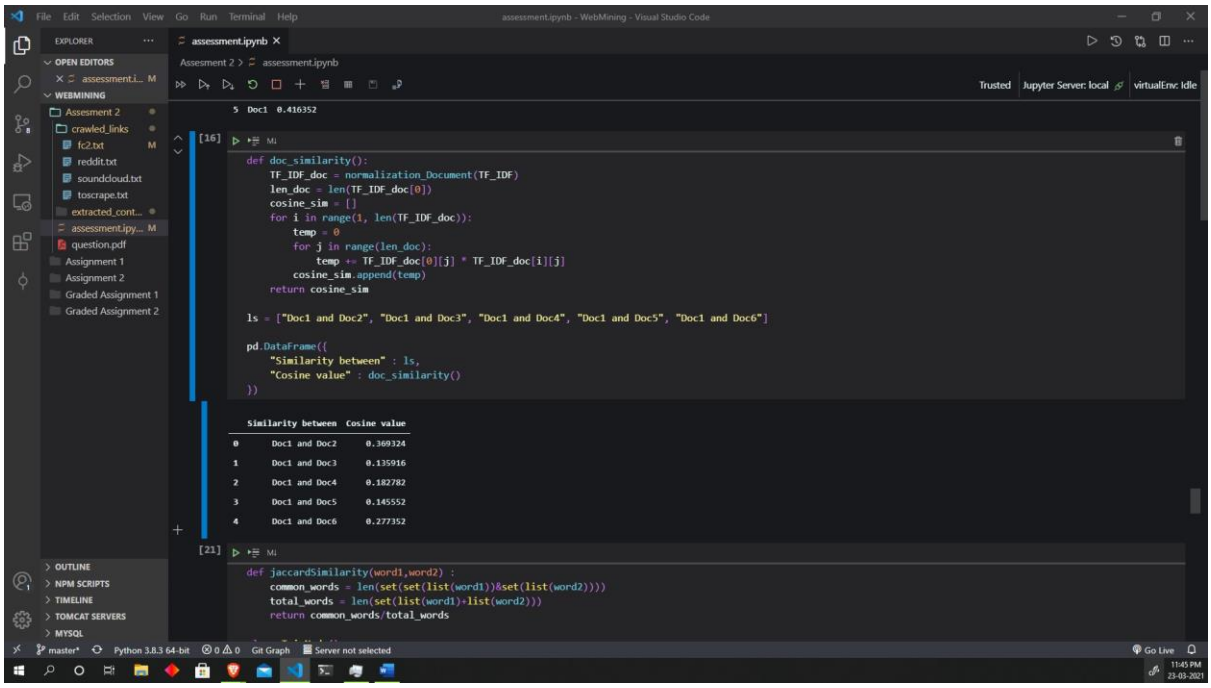
```
sorted_dict = {k: v for k,v in sorted(cosine_dist().items(), key=lambda item: item[1])}

pd.DataFrame(list(sorted_dict.items())[::-1])
```

```
sorted_dict = {k: v for k,v in sorted(euclidean_dist().items(), key=lambda item: item[1])}

pd.DataFrame(list(sorted_dict.items()))
```



19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

```
def doc_similarity():
    TF_IDF_doc = normalization_Document(TF_IDF)
    len_doc = len(TF_IDF_doc[0])
    cosine_sim = []
    for i in range(1, len(TF_IDF_doc)):
        temp = 0
        for j in range(len_doc):
            temp += TF_IDF_doc[0][j] * TF_IDF_doc[i][j]
        cosine_sim.append(temp)
    return cosine_sim

ls = ["Doc1 and Doc2", "Doc1 and Doc3", "Doc1 and Doc4", "Doc1 and Doc5", "Doc
1 and Doc6"]

pd.DataFrame({
    "Similarity between" : ls,
    "Cosine value" : doc_similarity()
})
```



	Similarity between	Cosine value
0	Doc1 and Doc2	0.369324
1	Doc1 and Doc3	0.135916
2	Doc1 and Doc4	0.182782
3	Doc1 and Doc5	0.145552
4	Doc1 and Doc6	0.277352

Q3

```
def jaccardSimilarity(word1,word2) :
    common_words = len(set(set(list(word1))&set(list(word2))))
    total_words = len(set(list(word1)+list(word2)))
    return common_words/total_words
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

```
class TrieNode():
    def __init__(self):
        self.children = {}
        self.last = False

class Trie():

    def __init__(self):
        self.root = TrieNode()
        self.word_list = []

    def formTrie(self, keys):
        for key in keys:
            self.insert(key)

    def insert(self, key):
        node = self.root

        for a in list(key):
            if not node.children.get(a):
                node.children[a] = TrieNode()

            node = node.children[a]

        node.last = True

    def suggestionsRec(self, node, word):  #recursive
        if node.last:
            self.word_list.append(word)

        for a,n in node.children.items():
            self.suggestionsRec(n, word + a)

    def PredictiveTyping(self, key):
        node = self.root
        temp_word = ''

        for a in list(key):
            if not node.children.get(a):
                break

            temp_word += a
            node = node.children[a]

        self.suggestionsRec(node, temp_word)
        return self.word_list

    def autoCorrect(self,key) :
```

```
        allwords = self.PredictiveTyping("")
        highest_jacsims = 0
        highest_jacsims_word = allwords[0]
        for word in allwords :
            temp_sim = jaccardSimilarity(word, key)
            if temp_sim > highest_jacsims :
                highest_jacsims = temp_sim
                highest_jacsims_word = word
        return highest_jacsims_word

    def printTrie(self) :
        print(str(self.root.children) + "\n")

# Driver Code
keys = ["hello", "dog", "hell", "cat", "a",
        "hel", "help", "helps", "helping"] # keys to form the trie structure.
key = "hel"


#for artificial intelligence link - trie structure

print("\n\nArtificial Intelligence Wikipedia\n\n")
raw1 = extract("https://en.wikipedia.org/wiki/Artificial_intelligence")[1]

url1_words = re.findall(r'[a-zA-z]+', raw1)

STOP_WORDS.update(['.',',', "'",'"' , "?", "[","]","(",")","{","}","<",">","!"
])

url1_nostopwords = [x for x in url1_words if x not in STOP_WORDS]

t1 = Trie()
t1.formTrie(url1_nostopwords)
# t1.printTrie()
predTemp = input()
print("predictive typing " + predTemp + " : " + str(t1.PredictiveTyping(predTe
mp)) + "\n")
autoTemp = input()
print("predicted by autocorrect : "+ str(t1.autoCorrect(autoTemp)))


print("\n\nMachine Learning Wikipedia\n\n")

raw2 = extract("https://en.wikipedia.org/wiki/Machine_learning")[1]
url2_words = re.findall(r'[a-zA-z]+', raw2)

#adding manual stop words to the STOP_WORDS set
```

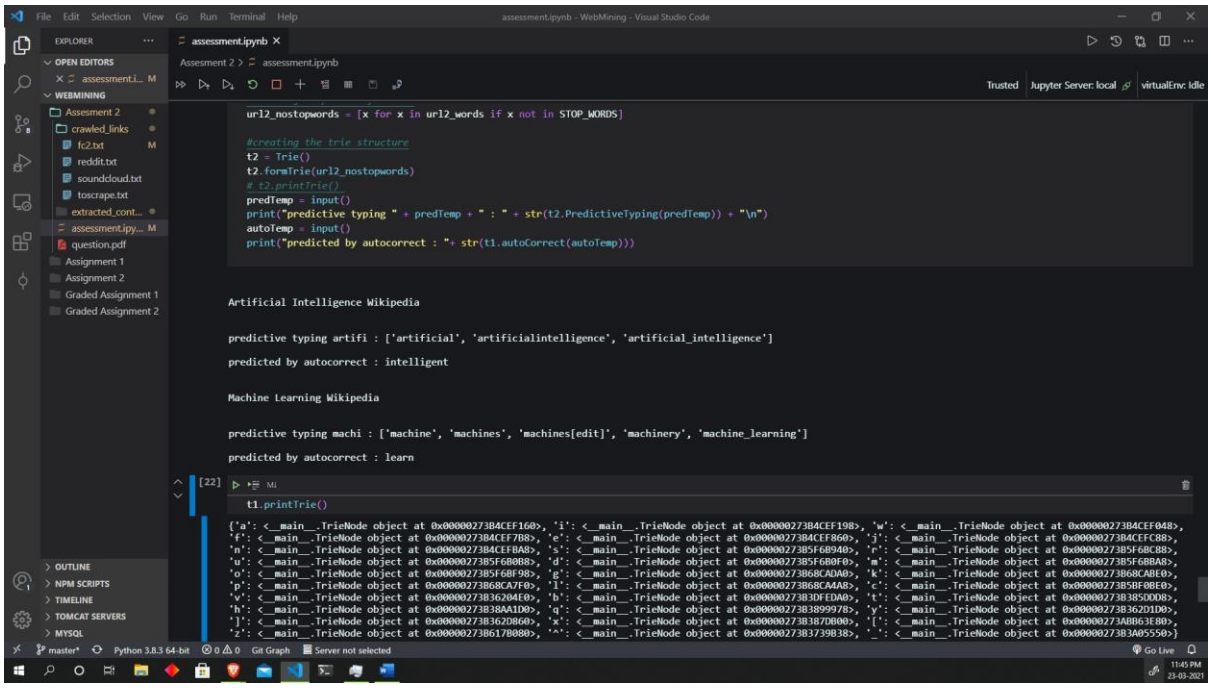
19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

```
STOP_WORDS.update(['.',',', "'",'"' , "?", "[","]","(",")","{","}","<",">","!"
])

#removing stop words from url
url2_nostopwords = [x for x in url2_words if x not in STOP_WORDS]

#creating the trie structure
t2 = Trie()
t2.formTrie(url2_nostopwords)
# t2.printTrie()
predTemp = input()
print("predictive typing " + predTemp + " : " + str(t2.PredictiveTyping(predTe
mp)) + "\n")
autoTemp = input()
print("predicted by autocorrect : "+ str(t1.autoCorrect(autoTemp)))
```

```
t1.printTrie()
```

```
t2.printTrie()
```

output:



19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2