

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.parser")

                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:
```

```

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:

```

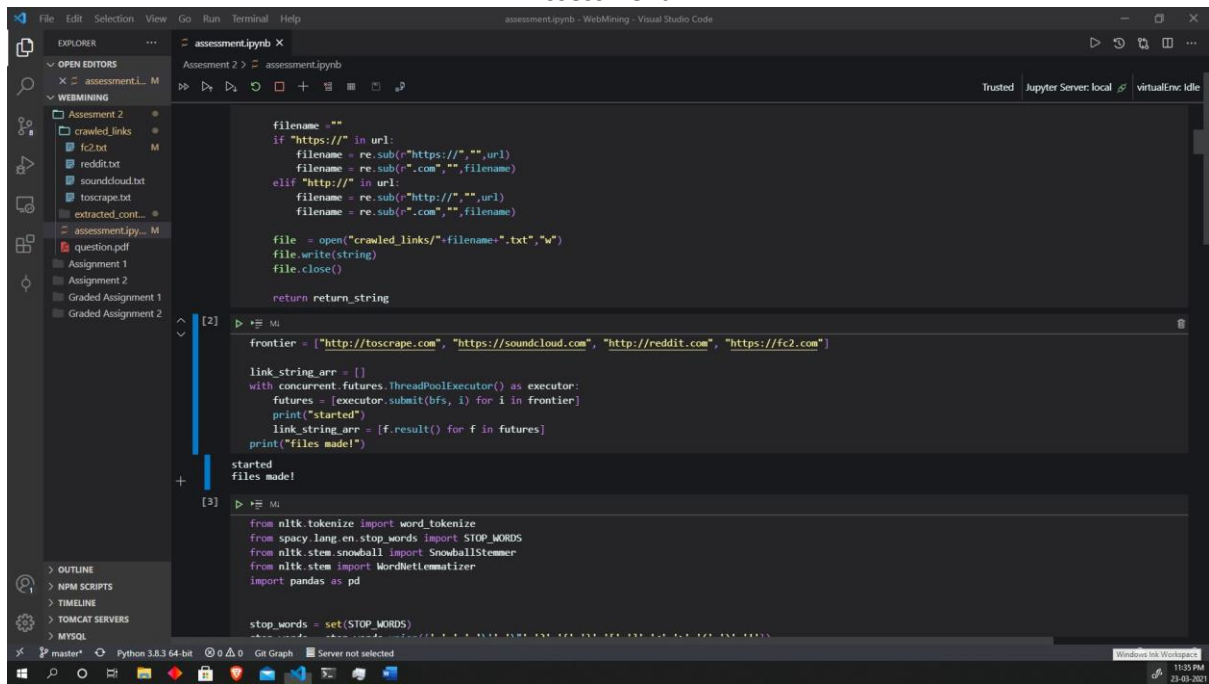
fc2.txt - WebMining - Visual Studio Code
Assessment 2 > crawled links > fc2.txt
1 at depth 0: //fc2.com/en/
2 at depth 0: https://blog.fc2.com/en/
3 at depth 0: https://video.fc2.com/en/
4 at depth 1: //fc2.com/
5 at depth 1: //help.fc2.com/
6 at depth 1: //request.fc2.com/search_wish.php?ct1=0
7 at depth 1: //blog.fc2.com/en
8 at depth 1: https://id.fc2.com/signup.php?ref-blog&switch_language=en
9 at depth 1: https://secure.id.fc2.com/done-blog&switch_language=en
10 at depth 1: https://fc2blogmanualen.blog.fc2.com/
11 at depth 1: #
12 at depth 1: https://talk.fc2.com/en/
13 at depth 1: https://blog.fc2.com/?lang=en
14 at depth 1: https://video.fc2.com/language_change.php?lang=en
15 at depth 1: https://live.fc2.com/en/Pafide-2828524
16 at depth 1: https://contents.fc2.com/language_change.php?lang=en
17 at depth 2: https://blog.fc2.com/
18 at depth 2: http://0215826.blog.fc2.com/blog-entry-11862.html
19 at depth 2: http://semenaljis.blog.fc2.com/blog-entry-18815.html
20 at depth 2: http://azukichamichan.blog48.fc2.com/blog-entry-1983.html
21 at depth 2: https://blog.fc2.com/
22 at depth 2: https://fc2blogmanualen.blog.fc2.com/blog-entry-180.html
23 at depth 2: /blog-category-0.html

soundcloud.txt
Assessment 2 > crawled links > soundcloud.txt
1 at depth 0: /
2 at depth 0: http://www.enable-javascript.com/
3 at depth 0: https://help.soundcloud.com/hc/articles/115081564388-Technical-requirements
4 at depth 0: http://google.com/chrome
5 at depth 0: http://firefox.com
6 at depth 1: #nogo
7 at depth 1: ./
8 at depth 1: /her/
9 at depth 1: /dev/
10 at depth 1: /nl/
11 at depth 1: https://help.soundcloud.com
12 at depth 1: https://community.soundcloud.com/getting-started
13 at depth 1: https://community.soundcloud.com/brow-your-career
14 at depth 1: https://community.soundcloud.com/industry
15 at depth 1: https://support.google.com/accounts/answer/1118621?hl=en
16 at depth 1: https://policies.google.com/technologies/cookies?hl=en
17 at depth 1: /chrome/
18 at depth 1: #jump-content
19 at depth 1: /en-US/
20 at depth 1: /firefox/download/thanks/
21 at depth 1: /en-US/privacy/firefox/
22 at depth 1: https://accounts.firefox.com/signupentrypoint-mozilla.org-globalnav?from
23 at depth 1: /en-US/firefox/browsers/

reddit.txt
Assessment 2 > crawled links > reddit.txt
1 at depth 0: /
2 at depth 0: https://www.reddit.com/login?dest=https%3A%2F%2Fwww.reddit.com%2F
3 at depth 0: https://www.reddit.com/register?dest=https%3A%2F%2Fwww.reddit.com%2F
4 at depth 0: /hot/
5 at depth 0: /new/
6 at depth 1: https://www.redditinc.com/policies/user-agreement
7 at depth 1: https://www.redditinc.com/policies/privacy-policy
8 at depth 1: /username?dest=https%3A%2F%2Fwww.reddit.com%2F
9 at depth 1: /password?dest=https%3A%2F%2Fwww.reddit.com%2F
10 at depth 1: /register?dest=https%3A%2F%2Fwww.reddit.com%2F
11 at depth 1: /login?dest=https%3A%2F%2Fwww.reddit.com%2F
12 at depth 1: #
13 at depth 1: https://www.reddit.com/login?dest=https%3A%2F%2Fwww.reddit.com%2F?hot%2F
14 at depth 1: https://www.reddit.com/register?dest=https%3A%2F%2Fwww.reddit.com%2F?hot%2F
15 at depth 1: https://www.reddit.com/login?dest=https%3A%2F%2Fwww.reddit.com%2F?new%2F
16 at depth 1: https://www.reddit.com/register?dest=https%3A%2F%2Fwww.reddit.com%2F?new%2F
17 at depth 2: /username?dest=https%3A%2F%2Fwww.reddit.com%2F/username?dest=https%3A%2F
18 at depth 2: /password?dest=https%3A%2F%2Fwww.reddit.com%2F/username?dest=https%3A%2F
19 at depth 2: /register?dest=https%3A%2F%2Fwww.reddit.com%2F/username?dest=https%3A%2F
20 at depth 2: /username?dest=https%3A%2F%2Fwww.reddit.com%2F/username?dest=https%3A%2F
21 at depth 2: /password?dest=https%3A%2F%2Fwww.reddit.com%2F/password?dest=https%3A%2F
22 at depth 2: /register?dest=https%3A%2F%2Fwww.reddit.com%2F/password?dest=https%3A%2F
23 at depth 2: /username?dest=https%3A%2F%2Fwww.reddit.com%2F/register?dest=https%3A%2F

toscrapet.txt
Assessment 2 > crawled links > toscrapet.txt
1 at depth 0: http://books.toscrape.com
2 at depth 0: http://quotes.toscrape.com/
3 at depth 0: http://quotes.toscrape.com
4 at depth 1: index.html
5 at depth 1: catalogue/category/books_1/index.html
6 at depth 1: catalogue/category/books/travel_2/index.html
7 at depth 1: catalogue/category/books/mystery_3/index.html
8 at depth 1: /
9 at depth 1: /login
10 at depth 1: /author/Albert-Einstein
11 at depth 1: /tag/change/page/1/
12 at depth 1: /tag/deep-thoughts/page/1/
13 at depth 2: https://www.goodreads.com/quotes
14 at depth 2: https://scrapinghub.com
15

```



```

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

[2] In:
frontier = ["http://toscraper.com", "https://soundcloud.com", "http://reddit.com", "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!")

started
files made!

[3] In:
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]))
  
```

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)

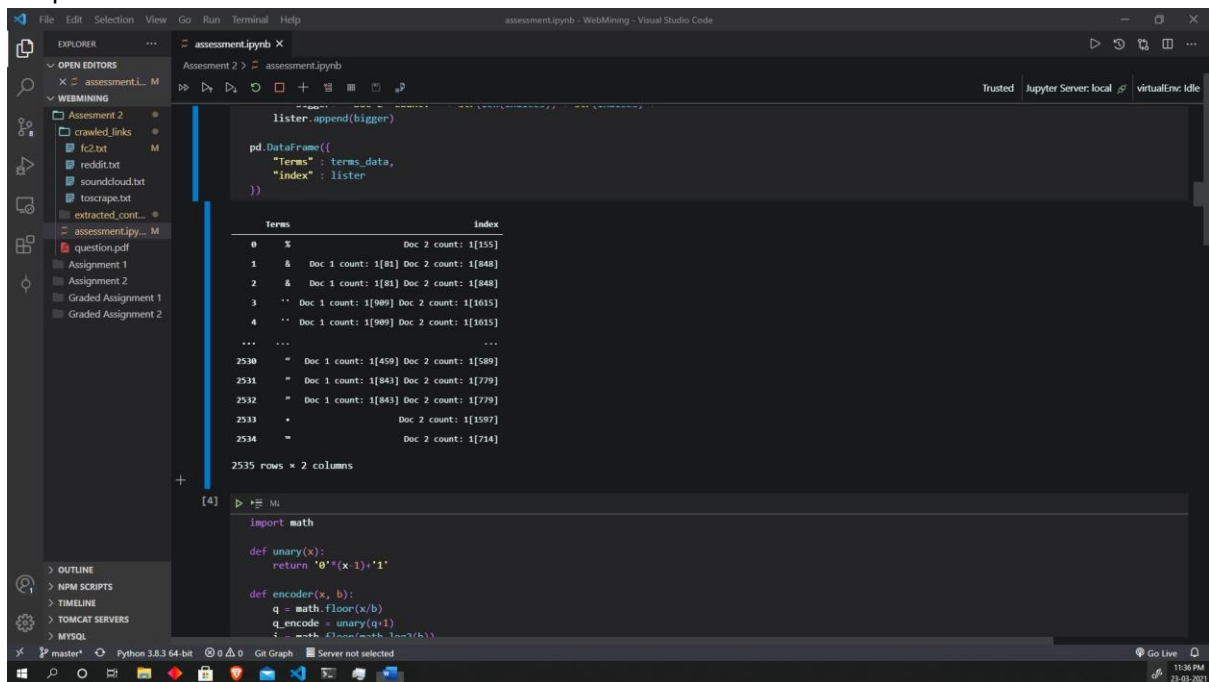
terms_data.sort()

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:



The screenshot shows a Jupyter Notebook environment in Visual Studio Code. The notebook is titled 'assessment.ipynb' and is part of a project named 'WebMining'. The code in the notebook includes a pandas DataFrame with two columns: 'Terms' and 'index'. The DataFrame contains 2535 rows and 2 columns. The output of the DataFrame is displayed as a table with columns 'Terms' and 'index'. The table shows a list of terms and their corresponding indices, along with document counts for each term. Below the table, the output is summarized as '2535 rows x 2 columns'. The code also includes a custom encoder function defined as follows:

```
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))
```

Q2

```
import math

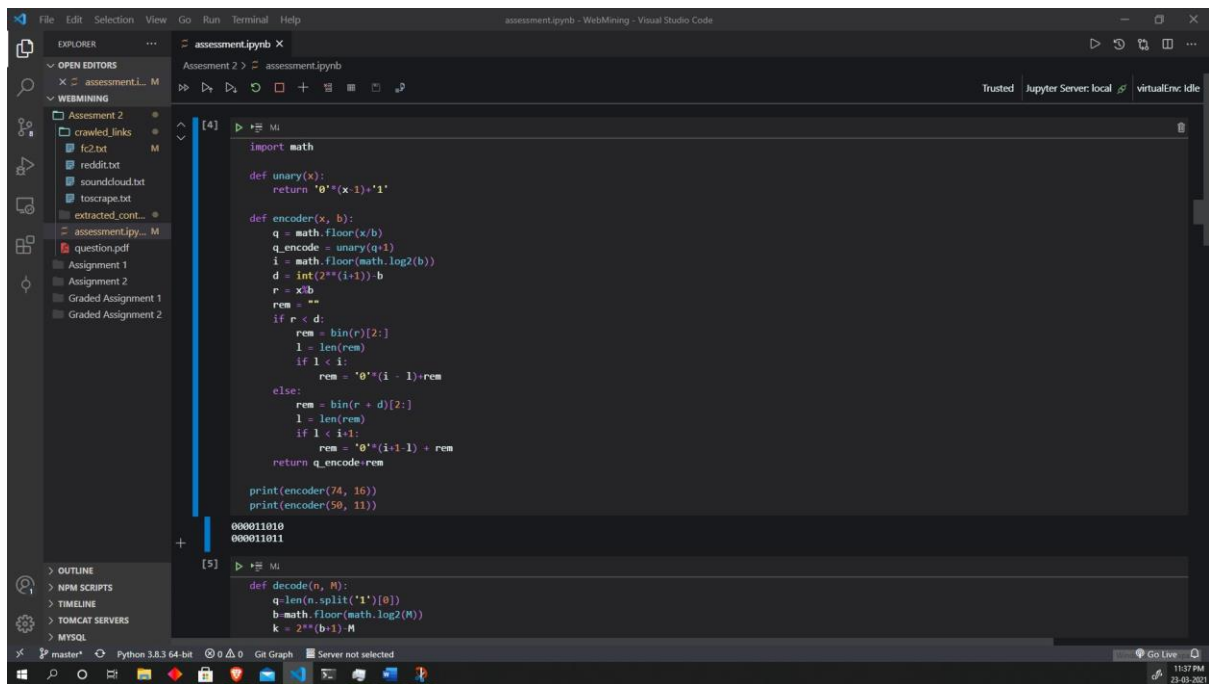
def unary(x):
    return '0'*(x-1)+'1'

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:



```
File Edit Selection View Go Run Terminal Help
assessment.ipynb - WebMining - Visual Studio Code

EXPLORER
OPEN EDITORS
assessment.ipynb
WEBMINING
Assessment 2
  crawled_links
  k2.txt
  reddit.txt
  soundcloud.txt
  toscrape.txt
  extracted_cont...
assessment.ipynb
question.pdf
Assignment 1
Assignment 2
Graded Assignment 1
Graded Assignment 2

[4]
import math

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

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

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

000011010
000011011

[5]
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("1111111110010001101", 10))
```

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("1111111110010001101", 10))
```

output:

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

```
File Edit Selection View Go Run Terminal Help
assessment.ipynb - WebMining - Visual Studio Code

EXPLORER
OPEN EDITORS
assessment.ipynb
WEBMINING
Assessment 2
crawled_links
fc2.txt
reddit.txt
soundcloud.txt
toscraps.txt
extracted_content...
question.pdf
Assignment 1
Assignment 2
Graded Assignment 1
Graded Assignment 2

assessment.ipynb
[5]
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+b],2)
    if r>=k:
        r = int(n[q-1:q+b+1],2)
        r=r-k
    x=q*M+r
    return x
print(decode("111111110010001101", 10))

[6]
# 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
```

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

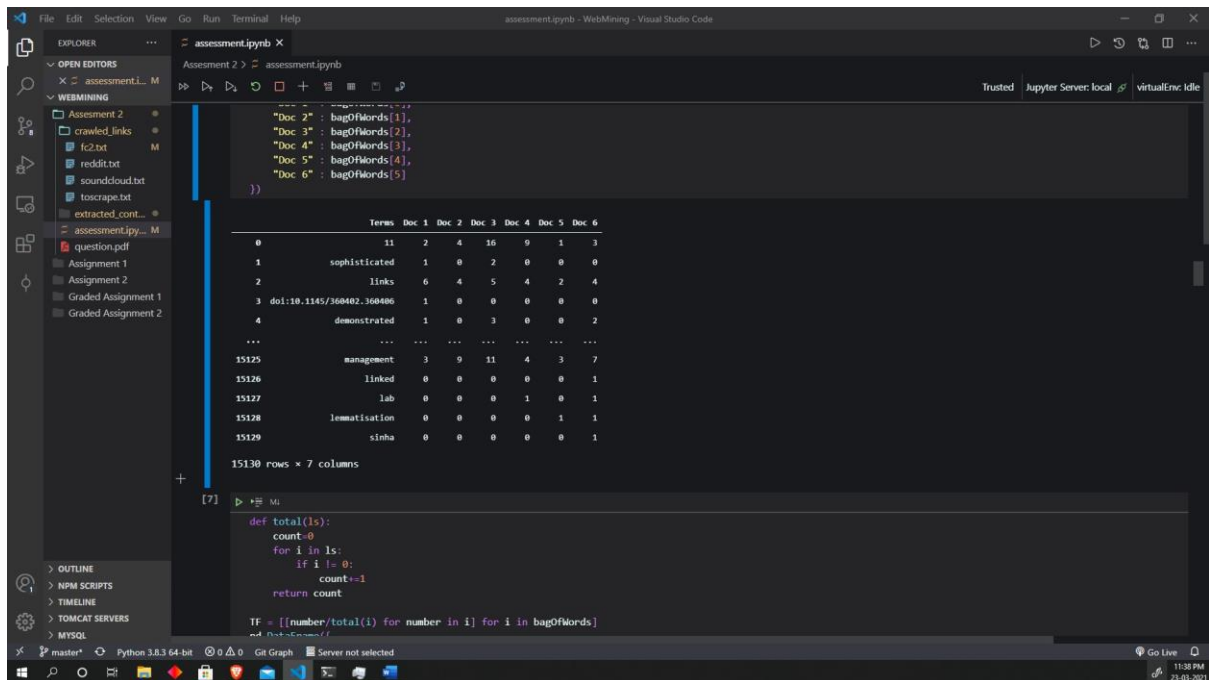
def term_maker(ls, filenames):
    terms = []
    index = 0
    for i in ls:
        st = set(i).difference(stop_words)
        store('extracted_content/'+filenames[index]+'.txt', list(st))
        index += 1
        terms.append(list(st))
    terms = reduce(lambda z,y : z+y, terms)
    return terms
```

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_learning', '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):
    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]
```

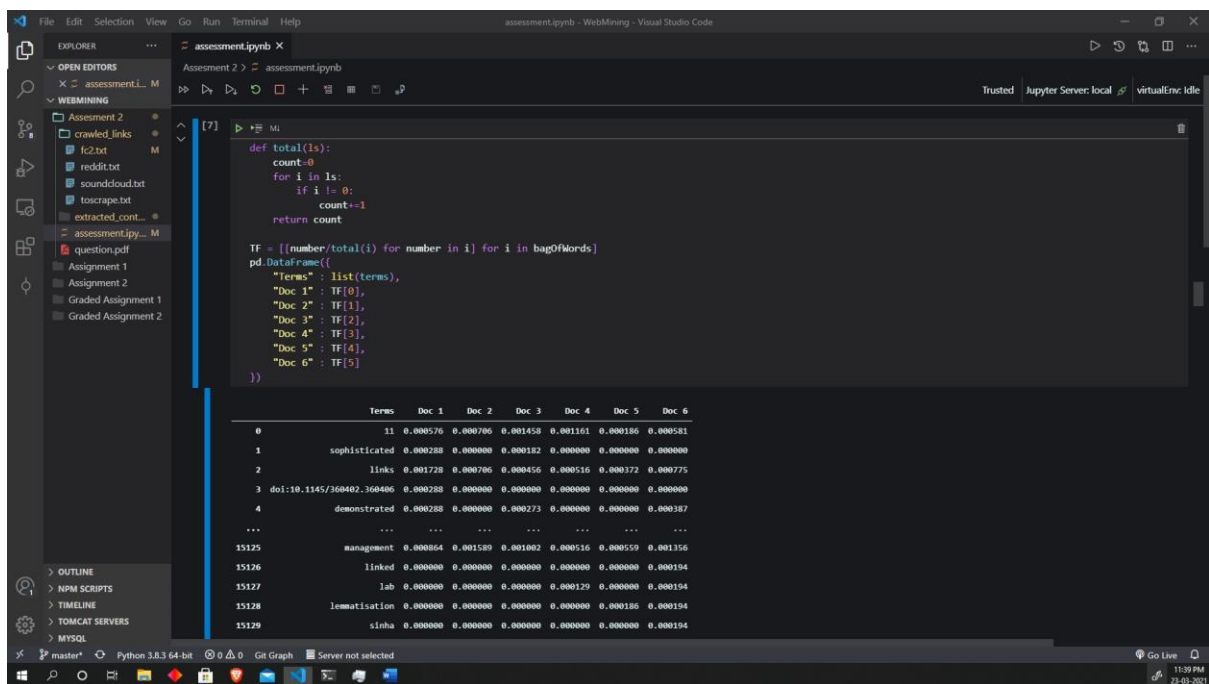
```
def total(ls):
```


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:



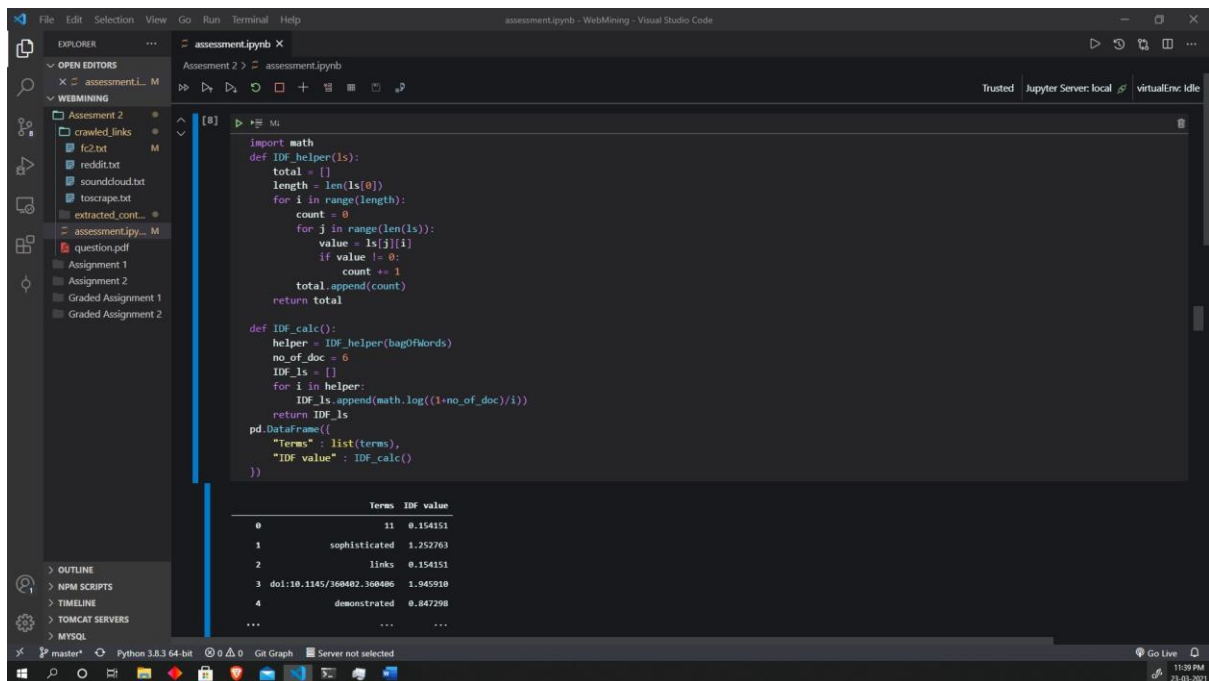
```
import math
def IDF_helper(ls):
    total = []
    length = len(ls[0])
    for i in range(length):
        count = 0
        for j in range(len(ls)):
            value = ls[j][i]
            if value != 0:
```

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()
})
```



The screenshot shows a Visual Studio Code editor window with a Jupyter Notebook open. The notebook contains Python code for calculating Inverse Document Frequency (IDF). The code defines a helper function, a main function, and uses pandas to create a DataFrame. The output of the code is displayed as a table with 5 rows and 3 columns: Terms, IDF value, and an index column.

```
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:
                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()
})
```

	Terms	IDF value
0	11	0.154151
1	sophisticated	1.252763
2	links	0.154151
3	dol:10.1145/360482.360486	1.945910
4	demonstrated	0.847298
...

```

def TF_IDF_calc(ls):
    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()
})

```

	Terms	IDF value
0	11	0.154151
1	sophisticated	1.252763
2	links	0.154151
3	dol:10.1145/360402.360406	1.945910
4	demonstrated	0.847298
...
15125	management	0.154151
15126	linked	1.945910
15127	lab	1.252763
15128	lemmatisation	1.252763
15129	sinha	1.945910

15130 rows x 2 columns

```

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

```

```

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]
})

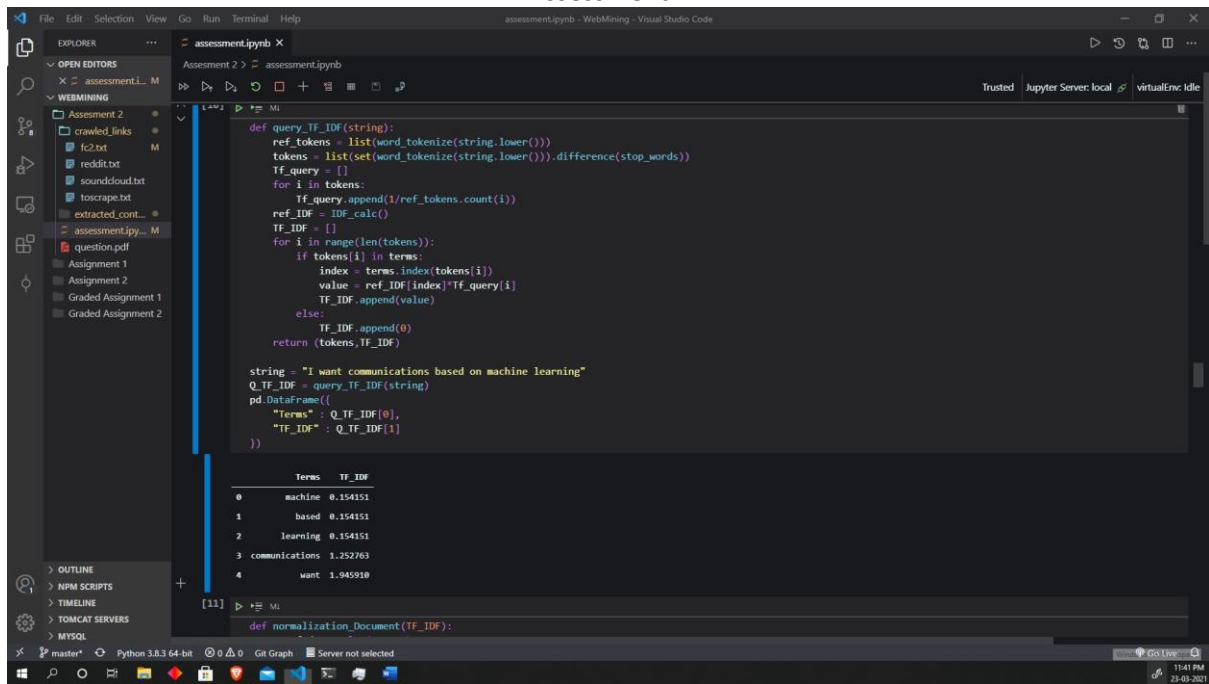
```

```
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]
})
```

	Terms	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
0	11	0.000009	0.000109	0.000225	0.000179	0.000029	0.000098
1	sophisticated	0.000361	0.000000	0.000228	0.000000	0.000000	0.000000
2	links	0.000266	0.000109	0.000070	0.000000	0.000057	0.000119
3	doi:10.1145/360402.360406	0.000560	0.000000	0.000000	0.000000	0.000000	0.000000
4	demonstrated	0.000244	0.000000	0.000232	0.000000	0.000000	0.000328
...
15125	management	0.000133	0.000245	0.000154	0.000000	0.000000	0.000209
15126	linked	0.000000	0.000000	0.000000	0.000000	0.000000	0.000377
15127	lab	0.000000	0.000000	0.000000	0.000162	0.000000	0.000243
15128	lemmatization	0.000000	0.000000	0.000000	0.000000	0.000233	0.000243
15129	sinha	0.000000	0.000000	0.000000	0.000000	0.000000	0.000377

```
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]
})
```



```
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],
})
```

The screenshot shows a Jupyter Notebook with the following code and output:

```
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],
})
```

The output is a DataFrame with the following structure:

	Terms	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6
0	11	0.002884	0.003442	0.002965	0.004997	0.000866	0.003176
1	sophisticated	0.011395	0.000000	0.003812	0.000000	0.000000	0.000000
2	links	0.008413	0.003442	0.000927	0.002221	0.001732	0.004235
3	dol:10.1145/360402.360406	0.017700	0.000000	0.000000	0.000000	0.000000	0.000000
4	demonstrated	0.007707	0.000000	0.003856	0.000000	0.000000	0.011639
...
15125	management	0.004206	0.007745	0.002038	0.002221	0.002598	0.007411
15126	linked	0.000000	0.000000	0.000000	0.000000	0.000000	0.013365
15127	lab	0.000000	0.000000	0.000000	0.004512	0.000000	0.008604
15128	lemmatization	0.000000	0.000000	0.000000	0.000000	0.007038	0.006604

```
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)
})
```

The screenshot shows a Jupyter Notebook with the following content:

```

15128      lemmatization  0.000000  0.000000  0.000000  0.000000  0.007038  0.008604
15129      sinha  0.000000  0.000000  0.000000  0.000000  0.000000  0.013365
15130 rows x 7 columns

[12] In [12]:
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)
})

      terms  TF_IDF
0  machine  0.028403
1    based  0.028403
2  learning  0.028403
3  communications  0.230828
4     want  0.358543

[13] In [13]:
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'])

```

```

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'])
)

```

The screenshot shows a Jupyter Notebook titled 'assessment.ipynb' in Visual Studio Code. The notebook is running on a Jupyter Server (local). The code defines a function 'cosine_dist()' which calculates the cosine distance between a document and a query. The function uses 'normalization_Document' and 'normalization_Query' to process the input. It then iterates over the tokens of the query and the terms of the document to calculate the cosine distance. The result is a pandas DataFrame with columns 'Documents' and 'Cosine Dist'.

```
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'])
```

	Documents	Cosine Dist
0	Doc1	0.005778
1	Doc2	0.002249
2	Doc3	0.003736
3	Doc4	0.007917
4	Doc5	0.001648
5	Doc6	0.000662

```
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

The screenshot shows a Jupyter Notebook with the following code and output:

```
[14]: 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_dist = {}
    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_dist['Doc'+str(i+1)] = euclidean_dist
    return euclidean_dist

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

	Documents	Euclidean Dist
0	Doc1	0.416352
1	Doc2	0.427644
2	Doc3	0.421194
3	Doc4	0.456539
4	Doc5	0.426861
5	Doc6	0.427928

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

```
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()))
```

The screenshot shows a Jupyter Notebook with the following code and output:

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

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

	0	1
0	Doc4	0.007917
1	Doc1	0.005778
2	Doc3	0.003736
3	Doc2	0.002289
4	Doc5	0.001648
5	Doc6	0.000662

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

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

	0	1
0	Doc1	0.416352
1	Doc3	0.421194
2	Doc5	0.426861
3	Doc2	0.427644
4	Doc6	0.427928
5	Doc4	0.456539

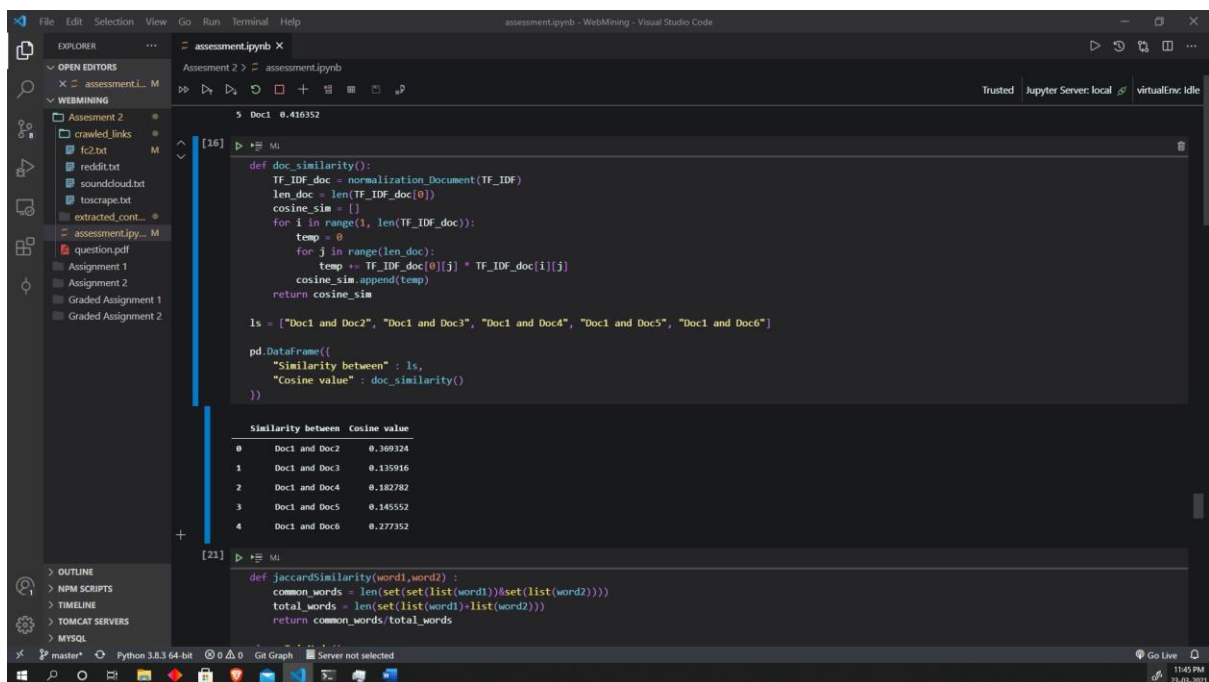
```
[16]: def doc_similarity():
    TF_IDF_doc = normalization.Document(TF_IDF)
    len_doc = len(TF_IDF_doc[0])
```

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", "Doc1 and Doc6"]

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



```
Doc1 0.416352
```

	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

```
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
```

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
```

```
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) :
```

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

```
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(predTemp)) + "\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
```

```
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(predTemp))) + "\n")
```

```
autoTemp = input()
```

```
print("predicted by autocorrect : " + str(t1.autoCorrect(autoTemp)))
```

```
t1.printTrie()
```

```
t2.printTrie()
```

output:

```

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

#creating the trie structure
t2 = Trie()
t2.formTrie(artificial_intelligence_wikipedia)
# t2.printTrie()

predTemp = input()
print("predictive typing " + predTemp + " : " + str(t2.PredictiveTyping(predTemp))) + "\n")

autoTemp = input()
print("predicted by autocorrect : " + str(t1.autoCorrect(autoTemp)))

Artificial Intelligence Wikipedia

predictive typing arti : ['artificial', 'artificialintelligence', 'artificial_intelligence']
predicted by autocorrect : intelligent

Machine Learning Wikipedia

predictive typing machi : ['machine', 'machines', 'machines[edit]', 'machinery', 'machine_learning']
predicted by autocorrect : learn

```

The screenshot shows the Visual Studio Code interface with a file explorer on the left containing files like 'Assessment 2', 'crawled_links', 'f2.txt', 'reddit.txt', 'soundcloud.txt', 'toscrapt.txt', 'extracted_cont...', 'assessment.py...', 'question.pdf', 'Assignment 1', 'Assignment 2', 'Graded Assignment 1', and 'Graded Assignment 2'. The main editor window shows the Python code being executed. The output in the terminal shows the results of the predictive typing and autocorrection for the words 'artificial' and 'machine'.

19BCE2311 Gaurav Singh
Web Mining Lab
Assessment 2

