Practical No.1

Aim: Write a program to demonstrate bitwise operation

Source Code:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
corpus=[
'this is the first document.',
'this document is second document.',
'and this is the third one.',
'is this the first document?',
 ]
vectorizer= CountVectorizer()
X=vectorizer.fit_transform(corpus)
print("fit transform is ")
print(X.toarray())
df=pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
print("the generated data frame is")
print(df)
alldata= df[(df['this']==1)&(df['first']==1)]
print("indices where 'this'and 'first'terms are present are ",alldata.index.tolist())
ordata= df[(df['this']==1)|(df['first']==1)]
print("indices where either of 'this'and 'first'terms are present are ",ordata.index.tolist())
notdata=df[(df['and']!=1)]
print("indices where 'and' term is not present ",notdata.index.tolist())
```

Output:

```
runfile('C:/Users/gauri/untitled12.py', wdir='C:/Users/gauri')
```

fit transform is

[[0 1 1 1 0 0 1 0 1]

[0 2 0 1 0 1 0 0 1]

[1 0 0 1 1 0 1 1 1]

[0 1 1 1 0 0 1 0 1]]

the generated data frame is

	and	document	first	is	one	second	the	third	this
0	0	1	1	1	0	0	1	0	1
1	0	2	0	1	0	1	0	0	1
2	1	0	0	1	1	0	1	1	1
3	0	1	1	1	0	0	1	0	1

indices where 'this'and 'first'terms are present are [0, 3]

indices where either of 'this'and 'first'terms are present are [0, 1, 2, 3]

indices where 'and' term is not present [0, 1, 3]

Practical No.2

Aim: Implement PageRank Algorithm.

Source Code:

```
import numpy as np
from scipy.sparse import csc_matrix
from fractions import Fraction
def float_format(vector,decimal):
 return np.round((vector).astype(np.float),decimals=decimal)
G=np.matrix([[1,1,0],
 [1,0,1],
 [0,1,0]])
n=len(G)
print(n)
M=csc_matrix(G,dtype=np.float)
rsums=np.array(M.sum(1))[:,0]
ri,ci=M.nonzero()
M.data/rsums[ri]
dp=Fraction(1,n)
E=np.zeros((3,3))
E[:]=dp
beta=0.85
A=beta*M+((1-beta)*E)
r=np.matrix([dp,dp,dp])
r=np.transpose(r)
previous_r=r
for it in range(1,30):
```

```
 r=A*r
 if(previous_r==r).all():
 break
 previous_r=r
 print("Final:\n",float_format(r,3))
```

Output:

```
runfile('C:/Users/ckt/prac2ir.py', wdir='C:/Users/ckt')
3
('Final:\n', array([[0.617],
 [0.617],
 [0.333]]))
('Final:\n', array([[1.127],
 [0.886],
 [0.603]]))
('Final:\n', array([[1.841],
 [1.601],
 [0.884]]))
('Final:\n', array([[3.142],
 [2.533],
 [1.577]]))
('Final:\n', array([[5.186],
 [4.373],
 [2.515]]))
('Final:\n', array([[8.729],
 [7.15 ],
 [4.321]]))
('Final:\n', array([[14.507],
```

[12.103],

[7.087]]))

('Final:\n', array([[24.303],

[20.04],

[11.972]]))

('Final:\n', array([[40.507],

[33.65],

[19.85]]))

('Final:\n', array([[67.734],

[56.004],

[33.303]]))

('Final:\n', array([[113.029],

[93.733],

[55.455]]))

('Final:\n', array([[188.859],

[156.323],

[92.784]]))

('Final:\n', array([[315.303],

[261.295],

[154.773]]))

('Final:\n', array([[526.677],

[436.133],

[258.669]]))

('Final:\n', array([[879.463],

[728.619],

[431.787]]))

('Final:\n', array([[1468.863],

[1216.556],

[721.319]]))

('Final:\n', array([[2452.943],

[2031.992],

[1204.41]]))

('Final:\n', array([[4096.662],

[3393.217],

[2011.66]]))

('Final:\n', array([[6841.474],

[5667.15],

[3359.311]]))

Practical No.3

Aim: Implement Dynamic programming algorithm for computing the edit distance between strings s1 and s2.

Source Code:

```
import numpy as np

def Levenshtein(s1, s2):
    if s1 == "":
        return len(s2)
    if s2 == "":
        return len(s1)
    if s1[-1] == s2[-1]:
        cost = 0
    else:
        cost = 1
    res = min([Levenshtein(s1[:-1], s2)+1,
               Levenshtein(s1, s2[:-1])+1,
               Levenshtein(s1[:-1], s2[:-1]) + cost])
    return res


print(Levenshtein("execution", "intention"))
```

Output:

runfile('C:/Users/gauri/editdistancepy.py', wdir='C:/Users/gauri')

5

Practical No 4

Aim: Write a program to Compute Similarity between two text documents.

Source Code:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
def cosine_similarity(x, y):

    # Ensure length of x and y are the same
    if len(x) != len(y) :
        return None

    # Compute the dot product between x and y
    dot_product = np.dot(x, y)

    # Compute the L2 norms (magnitudes) of x and y
    magnitude_x = np.sqrt(np.sum(x**2))
    magnitude_y = np.sqrt(np.sum(y**2))

    # Compute the cosine similarity
    cosine_similarity = dot_product / (magnitude_x * magnitude_y)

    return cosine_similarity
corpus = ['data science is one of the most important fields of science',
          'this is one of the best data science courses',
          'data scientists analyze data']
```

```
# Create a matrix to represent the corpus

X = CountVectorizer().fit_transform(corpus).toarray()


print(X)

cos_sim_1_2 = cosine_similarity(X[0, :], X[1, :])

cos_sim_1_3 = cosine_similarity(X[0, :], X[2, :])

cos_sim_2_3 = cosine_similarity(X[1, :], X[2, :])


print('Cosine Similarity between: ')

print('\tDocument 1 and Document 2: ', cos_sim_1_2)

print('\tDocument 1 and Document 3: ', cos_sim_1_3)

print('\tDocument 2 and Document 3: ', cos_sim_2_3)
```

Output:

```
runfile('C:/Users/gauri/cosinepython.py', wdir='C:/Users/gauri')

[[0 0 0 1 1 1 1 1 2 1 2 0 1 0]

 [0 1 1 1 0 0 1 0 1 1 1 0 1 1]

 [1 0 0 2 0 0 0 0 0 0 0 1 0 0]]

Cosine Similarity between:

        Document 1 and Document 2:  0.6885303726590962

        Document 1 and Document 3:  0.21081851067789195

        Document 2 and Document 3:  0.2721655269759087
```

Practical No. 7

Aim: Write program for pre-processing of Text document: stop word removal

Source Code:

```
from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

example_sent= "We are students of CKT college"

stop_words=set(stopwords.words('english'))

word_tokens=word_tokenize(example_sent)

filtered_sentence=[w for w in word_tokens if not w in stop_words]

filtered_sentence=[]

for w in word_tokens:

 if w not in stop_words:

 filtered_sentence.append(w)

print(word_tokens)

print(filtered_sentence)
```

Output:

```
runfile('C:/Users/CKT/prac7.py', wdir='C:/Users/CKT') ['We', 'are', 'students', 'of', 'CKT', 'college']
['We', 'students', 'CKT', 'college']
```

Practical 8

Aim: Write a program for mining Twitter to identify tweets for a specific period and identify trends and named entities.

Source Code:

```
#Import the necessary methods from tweepy library
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
#Variables that contains the user credentials to access Twitter API
consumer_key = "3yMYKK5Ben0iUaaJ0KGLqrlzk"
consumer_secret = "gIS4fQrYjpREWxi9RrtgiS4vxzPjlNTIuQmnBCizoL06nrhmNu"
access_token = "1101412887430479872-7YPZMaFXJrR3dRj4BkBHezad9wmJEI"
access_token_secret = "1a3sXc1OE892iwiEN9GXLLCB2paKkzR1VDBkyvPSbjjDn"
#This is a basic listener that just prints received tweets to stdout.
class StdOutListener(StreamListener):
 def on_data(self, data):
 print data
 return True
 def on_error(self, status):
 print status
if __name__ == '__main__':
 #This handles Twitter authetification and the connection to Twitter Streaming API
 l = StdOutListener()
 auth = OAuthHandler(consumer_key, consumer_secret)
 auth.set_access_token(access_token, access_token_secret)
 stream = Stream(auth, l)
```

```
#This line filter Twitter Streams to capture data by the keywords: 'python', 'javascript', 'ruby'

stream.filter(track=['python', 'javascript', 'ruby'])
```

Output:

{"created_at":"Tue Mar 12 09:23:42 +0000 2019","id":1105398953091035136,"id_str":"1105398953091035136","text":"RT @karen73984451: #k9hour #gorgeous girlie ready and waiting on a super duper home #Ruby #Itsallaboutthedogs #TeamZay @epsomcanine plz RT\u2026","source":"\u003ca href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for iPhone\u003c\/a\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_screen_name":null,"user":{"id":2615014568,"id_str":"2615014568","name":"Titachot","screen_name":"Titachot","location":" Chonburi, Thailand","url":null,"description":"Adopt Don't Shop ! Do not support Puppy Mills

Practical 9

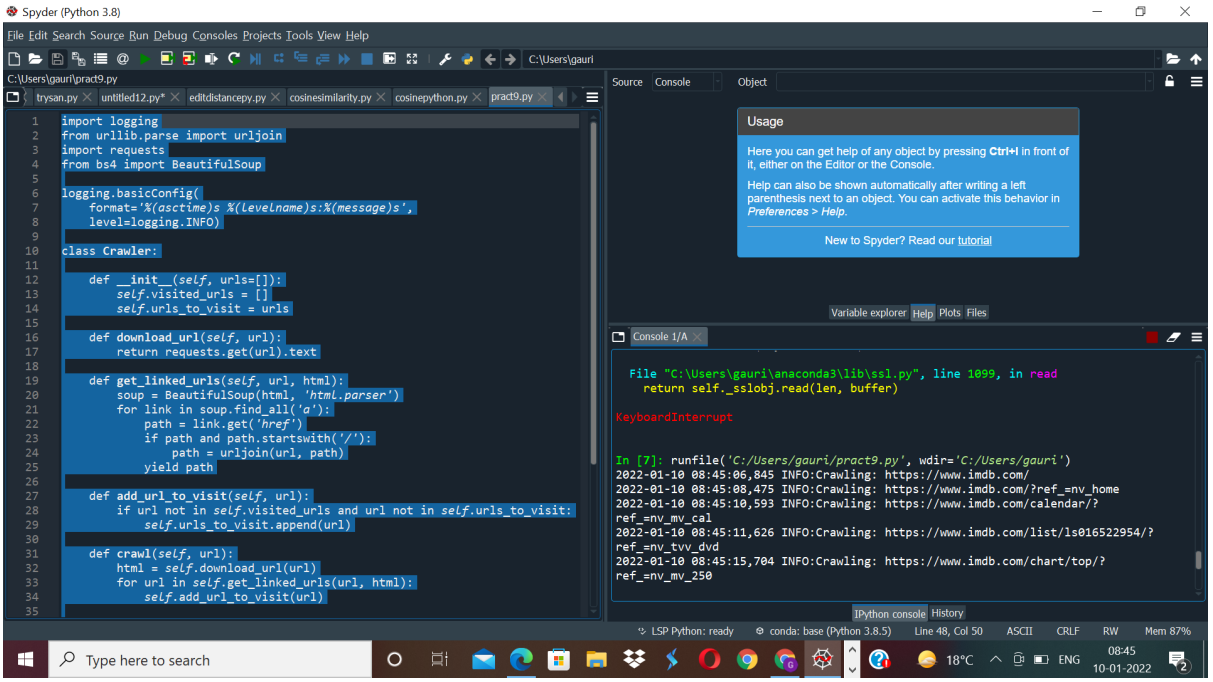
Aim: Write a program to implement simple crawler.

Source Code:

```
import logging
from urllib.parse import urljoin
import requests
from bs4 import BeautifulSoup

logging.basicConfig(
    format='%(asctime)s %(levelname)s:%(message)s',
    level=logging.INFO)

class Crawler:

    def __init__(self, urls=[]):
        self.visited_urls = []
        self.urls_to_visit = urls

    def download_url(self, url):
        return requests.get(url).text

    def get_linked_urls(self, url, html):
        soup = BeautifulSoup(html, 'html.parser')
        for link in soup.find_all('a'):
            path = link.get('href')
            if path and path.startswith('/'):
```

```
            path = urljoin(url, path)
        yield path

    def add_url_to_visit(self, url):
        if url not in self.visited_urls and url not in self.urls_to_visit:
            self.urls_to_visit.append(url)

    def crawl(self, url):
        html = self.download_url(url)
        for url in self.get_linked_urls(url, html):
            self.add_url_to_visit(url)

    def run(self):
        while self.urls_to_visit:
            url = self.urls_to_visit.pop(0)
            logging.info(f'Crawling: {url}')
            try:
                self.crawl(url)
            except Exception:
                logging.exception(f'Failed to crawl: {url}')
            finally:
                self.visited_urls.append(url)

if __name__ == '__main__':
    Crawler(urls=['https://www.imdb.com/']).run()
```

Output:


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\gauri\pract9.py
trysan.py
untitled12.py*
editdistancepy.py
cosinesimilarity.py
cosinepython.py
pract9.py
import logging
from urllib.parse import urljoin
import requests
from bs4 import BeautifulSoup
logging.basicConfig(
format='%(asctime)s %(levelname)s:%(message)s',
level=logging.INFO)
class Crawler:
def __init__(self, urls=[]):
self.visited_urls = []
self.urls_to_visit = urls
def download_url(self, url):
return requests.get(url).text
def get_linked_urls(self, url, html):
soup = BeautifulSoup(html, 'html.parser')
for link in soup.find_all('a'):
path = link.get('href')
if path and path.startswith('/'):
path = urljoin(url, path)
yield path
def add_url_to_visit(self, url):
if url not in self.visited_urls and url not in self.urls_to_visit:
self.urls_to_visit.append(url)
def crawl(self, url):
html = self.download_url(url)
for url in self.get_linked_urls(url, html):
self.add_url_to_visit(url)
Source
Console
Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
New to Spyder? Read our tutorial
Variable explorer
Help
Plots
Files
Console 1/A
File "C:\Users\gauri\anaconda3\lib\ssl.py", line 1099, in read
return self._sslobj.read(len, buffer)
KeyboardInterrupt
In [7]: runfile('C:/Users/gauri/pract9.py', wdir='C:/Users/gauri')
2022-01-10 08:45:06,845 INFO:Crawling: https://www.imdb.com/
2022-01-10 08:45:08,475 INFO:Crawling: https://www.imdb.com/?ref_=nv_home
2022-01-10 08:45:10,593 INFO:Crawling: https://www.imdb.com/calendar/?ref_=nv_mv_cal
2022-01-10 08:45:11,626 INFO:Crawling: https://www.imdb.com/list/ls016522954/?ref_=nv_tvv_dvd
2022-01-10 08:45:15,704 INFO:Crawling: https://www.imdb.com/chart/top/?ref_=nv_mv_250
IPython console
History
LSP Python: ready
conda: base (Python 3.8.5)
Line 48, Col 50
ASCII
CRLF
RW
Mem 87%
Type here to search
18°C
ENG
08:45
10-01-2022

Practical 10

Aim: Write a program to parse XML text, generate Web graph and compute topic specific page rank.

movies.xml:

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
  <type>War, Thriller</type>
  <format>DVD</format>
  <year>2003</year>
  <rating>PG</rating>
  <stars>10</stars>
  <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
  <type>Anime, Science Fiction</type>
  <format>DVD</format>
  <year>1989</year>
  <rating>R</rating>
  <stars>8</stars>
  <description>A schientific fiction</description>
</movie>
  <movie title="Trigun">
  <type>Anime, Action</type>
  <format>DVD</format>
  <episodes>4</episodes>
  <rating>PG</rating>
  <stars>10</stars>
  <description>Vash the Stampede!</description>
```

```
</movie>
<movie title="Ishtar">
  <type>Comedy</type>
  <format>VHS</format>
  <rating>PG</rating>
  <stars>2</stars>
  <description>Viewable boredom</description>
</movie>
</collection>
```

Source Code:

```
import networkx as nx
import matplotlib.pyplot as plt
from xml.dom.minidom import parse
import xml.dom.minidom
# Open xml document using minidom parser
DOMTree=xml.dom.minidom.parse("movies.xml")
collection=DOMTree.documentElement
if collection.hasAttribute("shelf"):
 print "Root element: %s" % collection.getAttribute("shelf")

# get all the movies in the collection
movies = collection.getElementsByTagName("movie")
#print detail of each movie.
for movie in movies:
 print"*****Movie*****"
 if movie.hasAttribute("title"):
 print"Title: %s" %movie.getAttribute("title")
```

```
type = movie.getElementsByTagName('type')[0]
print "Type: %s" % type.childNodes[0].data
format= movie.getElementsByTagName('format')[0]
print "format: %s" % format.childNodes[0].data
rating= movie.getElementsByTagName('rating')[0]
print "Rating: %s" % rating.childNodes[0].data
description=movie.getElementsByTagName('description')[0]
print"description: %s" % description.childNodes[0].data

def GenerateGraph():
G=nx.Graph()

# adding just one node:
G.add_node("a")

# adding a list of edges:
G.add_edges_from([("a","b"),("b","c"), ("c","d"), ("d","a"),("a","c")])

nx.draw(G)
plt.savefig("simple_path.png") # save as png
plt.show() # display

print("Nodes of graph: ")
print(G.nodes())
print("Edges of graph: ")
print(G.edges())
```

GenerateGraph()

Output

runfile('C:/Users/ckt/.spyder/prct 10 .py', wdir='C:/Users/ckt/.spyder')

Root element: New Arrivals

*****Movie*****

Title: Enemy Behind

Type: War, Thriller

format: DVD

Rating: PG

description: Talk about a US-Japan war

*****Movie*****

Title: Transformers

Type: Anime, Science Fiction

format: DVD

Rating: R

description: A schientific fiction

*****Movie*****

Title: Trigun

Type: Anime, Action

format: DVD

Rating: PG

description: Vash the Stampede!

*****Movie*****

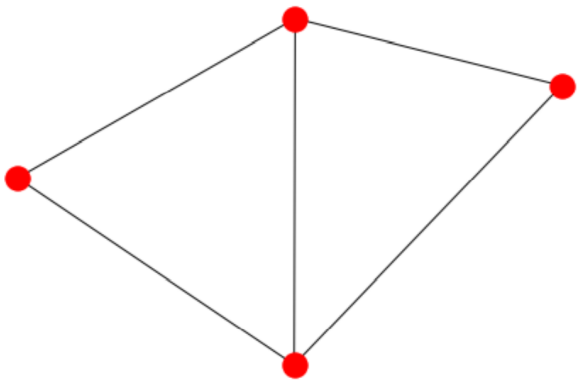
Title: Ishtar

Type: Comedy

format: VHS

Rating: PG

description: Viewable boredom



Nodes of graph:

['a', 'c', 'b', 'd']

Edges of graph:

[('a', 'c'), ('a', 'b'), ('a', 'd'), ('c', 'b'), ('c', 'd')