:[1] In

```
import nltk #need for dealing with text
import os # need for looping through folders
import string
import numpy as np
import copy
import pandas as pd
import pickle
import re
import math # need for computing TF-IDF score
```

:[2] In

```
pip install num2words
```

```
Requirement already satisfied: num2words in d:\users\d7me_\anaconda3\lib\sit
(e-packages (0.5.10
Requirement already satisfied: docopt>=0.6.2 in d:\users\d7me_\anaconda3\lib
(\site-packages (from num2words) (0.6.2
.Note: you may need to restart the kernel to use updated packages
```

:[3] In

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import Counter
from num2words import num2words
```

:[4] In

```
title = "Used"
os.chdir("C:\\Users\\D7me_\\mini_newsgroups\\mini_newsgroups\\comp.graphics")
paths = []
for (dirpath, dirnames, filenames) in os.walk(str(os.getcwd())+'/'+title+'/'):
    for i in filenames:
        paths.append(str(dirpath)+str("\\")+i)
print(dirpath)
```

```
/C:\Users\D7me_\mini_newsgroups\mini_newsgroups\comp.graphics/Used
```

:[ ] In

```

```

:[5] In

```
myfile = open(paths[0])
txt = myfile.read()
print(txt)
myfile.close()
```

```
Path: cantaloupe.srv.cs.cmu.edu!crabapple.srv.cs.cmu.edu!fs7.ece.cmu.edu!eur
opa.eng.gtefsd.com!gatech!asuvax!cs.utexas.edu!zaphod.mps.ohio-state.edu!sai
miri.primate.wisc.edu!usenet.coe.montana.edu!news.u.washington.edu!uw-beave
r!cs.ubc.ca!unixg.ubc.ca!kakwa.ucs.ualberta.ca!ersys!joth
(From: joth@ersys.edmonton.ab.ca (Joe Tham
Newsgroups: comp.graphics
?Subject: Where can I find SIPP
<Message-ID: <yFXJ2B2w165w@ersys.edmonton.ab.ca
Date: Mon, 05 Apr 93 14:58:21 MDT
Organization: Edmonton Remote Systems #2, Edmonton, AB, Canada
Lines: 11

 I recently got a file describing a library of rendering routines
 called SIPP (SImple Polygon Processor).  Could anyone tell me where I can
?FTP the source code and which is the newest version around
 Also, I've never used Renderman so I was wondering if Renderman
 is like SIPP?  ie. a library of rendering routines which one uses to make
...a program that creates the image

Thanks,  Joe Tham

--
 Joe Tham              joth@ersys.edmonton.ab.ca
```

:[6] In

```
myfile = open(paths[1])
txt = myfile.read()
print(txt)
myfile.close()
```

```
Xref: cantaloupe.srv.cs.cmu.edu alt.3d:2141 comp.graphics:37921
Path: cantaloupe.srv.cs.cmu.edu!crabapple.srv.cs.cmu.edu!fs7.ece.cmu.edu!eur
opa.eng.gtefsd.com!gatech!swrinde!zaphod.mps.ohio-state.edu!usc!elroy.jpl.na
sa.gov!ames!olivea!uunet!mcsun!fuug!kiae!relcom!newsserv
(From: alex@talus.msk.su (Alex Kolesov
Newsgroups: alt.3d,comp.graphics
!Subject: Help on RenderMan language wanted
<Message-ID: <9304051103.AA01274@talus.msk.su
Date: 5 Apr 93 11:00:50 GMT
Sender: news-service@newcom.kiae.su
Reply-To: alex@talus.msk.su
Organization: unknown
Lines: 17

! Hello everybody

If you are using PIXAR'S RenderMan 3D scene description language for creatin
 .g 3D worlds, please, help me

I'm using RenderMan library on my NeXT but there is no documentation about N
 eXTSTEP version of RenderMan available. I can create very complicated scenes
 ,and render them using surface shaders
.but I can not bring them to life by applying shadows and reflections

As far as I understand I have to define environmental and shadows maps to pr
.oduce reflections and shadows, but I do not know how to use them

.Any advises or simple RIB or C examples will be appreciated
...Thanks in advance

---
.Alex Kolesov                               Moscow, Russia
Talus Imaging & Communications Corporation
                 (e-mail: <alex@talus.msk.su>            (NeXT mail accepted
   .
```

:[7] In

```
def remove_stop_words(data):
    stop_words = stopwords.words('english')
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if w not in stop_words:
            new_text = new_text + " " + w
    return np.char.strip(new_text)

def remove_punctuation(data):
    symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
    for i in range(len(symbols)):
        data = np.char.replace(data, symbols[i], ' ')
        data = np.char.replace(data, "  ", " ")
    data = np.char.replace(data, ',', '')
    return data


def convert_lower_case(data):
    return np.char.lower(data)


def stemming(data):
    stemmer= PorterStemmer()

    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        new_text = new_text + " " + stemmer.stem(w)
    return np.char.strip(new_text)


def convert_numbers(data):
    data = np.char.replace(data, "0", " zero ")
    data = np.char.replace(data, "1", " one ")
    data = np.char.replace(data, "2", " two ")
    data = np.char.replace(data, "3", " three ")
    data = np.char.replace(data, "4", " four ")
    data = np.char.replace(data, "5", " five ")
    data = np.char.replace(data, "6", " six ")
    data = np.char.replace(data, "7", " seven ")
    data = np.char.replace(data, "8", " eight ")
    data = np.char.replace(data, "9", " nine ")
    return data


def remove_header(data):
    try:
        ind = data.index('\n\n')
        data = data[ind:]
    except:
        print("No Header")
    return data


def remove_apostrophe(data):
    return np.char.replace(data, "'", "")

def remove_single_characters(data):
```

```
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if len(w) > 1:
            new_text = new_text + " " + w
    return np.char.strip(new_text)
```

:[8] In

```
def preprocess(data, query):
    if not query:
        data = remove_header(data)
        data = convert_lower_case(data)
        data = convert_numbers(data)
        data = remove_punctuation(data)
        data = remove_stop_words(data)
        data = remove_apostrophe(data)
        data = remove_single_characters(data)
        data = stemming(data)
    return data
```

:[9] In

```
doc = 0
postings = pd.DataFrame()

for path in paths:
    file = open(path, 'r', encoding='cp1250')
    text = file.read().strip()
    file.close()
    preprocessed_text = preprocess(text, False)

    #Genrate matrex posting list
    if doc%100 == 0:
        print(doc)
    tokens = word_tokenize(str(preprocessed_text))
    for token in tokens:
        if token in postings:
            p = postings[token][0]
            p.add(doc)
            postings[token][0] = p
        else:
            postings.insert(value=[{doc}], loc=0, column=token)
    doc += 1


postings.to_pickle(title + "_unigram_postings")
```

```
             0
```

:[10] In

```
def preprocess(data):
    data = remove_header(data)
    data = convert_lower_case(data)
    data = convert_numbers(data)
    data = remove_punctuation(data)
    data = remove_stop_words(data)
    data = remove_apostrophe(data)
    data = remove_single_characters(data)
    data = stemming(data)
    return data
```

:[11] In

```
processed_text = []
for i in range(len(filenames)):
    file = open(dirpath+'/'+ filenames[i], 'r', encoding='cp1250', errors='ignore')
    text = file.read().strip()
    file.close()

    processed_text.append(word_tokenize(str(preprocess(text))))
    print(processed_text)
```

```
recent', 'got', 'file', 'describ', 'librari', 'render', 'routin', 'cal']]
l', 'sipp', 'simpl', 'polygon', 'processor', 'could', 'anyon', 'tell', 'f
tp', 'sourc', 'code', 'newest', 'version', 'around', 'also', 've', 'neve
r', 'use', 'renderman', 'wonder', 'renderman', 'like', 'sipp', 'ie', 'lib
rari', 'render', 'routin', 'one', 'use', 'make', 'program', 'creat', 'ima
g', 'thank', 'joe', 'tham', 'joe', 'tham', 'joth', 'ersi', 'edmonton', 'a
[['b', 'ca
recent', 'got', 'file', 'describ', 'librari', 'render', 'routin', 'cal']]
l', 'sipp', 'simpl', 'polygon', 'processor', 'could', 'anyon', 'tell', 'f
tp', 'sourc', 'code', 'newest', 'version', 'around', 'also', 've', 'neve
r', 'use', 'renderman', 'wonder', 'renderman', 'like', 'sipp', 'ie', 'lib
rari', 'render', 'routin', 'one', 'use', 'make', 'program', 'creat', 'ima
g', 'thank', 'joe', 'tham', 'joe', 'tham', 'joth', 'ersi', 'edmonton', 'a
 b', 'ca'], ['hello', 'everybodi', 'use', 'pixar', 'renderman', 'three',
'scene', 'descript', 'languag', 'creat', 'three', 'world', 'pleas', 'hel
p', 'use', 'renderman', 'librari', 'next', 'document', 'nextstep', 'versi
 on', 'renderman', 'avail', 'creat', 'complic', 'scene', 'render', 'use',
 'surfac', 'shader', 'bring', 'life', 'appli', 'shadow', 'reflect', 'far',
'understand', 'defin', 'environment', 'shadow', 'map', 'produc', 'reflec
t', 'shadow', 'know', 'use', 'advis', 'simpl', 'rib', 'exampl', 'apprec
i', 'thank', 'advanc', 'alex', 'kolesov', 'moscow', 'russia', 'talu', 'im
ag', 'commun', 'corpor', 'mail', 'alex', 'talu', 'msk', 'su', 'next', 'ma
[['il', 'accept
recent', 'got', 'file', 'describ', 'librari', 'render', 'routin', 'cal']]
l', 'sipp', 'simpl', 'polygon', 'processor', 'could', 'anyon', 'tell', 'f
tp', 'sourc', 'code', 'newest', 'version', 'around', 'also', 've', 'neve
r', 'use', 'renderman', 'wonder', 'renderman', 'like', 'sipp', 'ie', 'lib
rari', 'render', 'routin', 'one', 'use', 'make', 'program', 'creat', 'ima
g', 'thank', 'joe', 'tham', 'joe', 'tham', 'joth', 'ersi', 'edmonton', 'a
 b', 'ca'], ['hello', 'everybodi', 'use', 'pixar', 'renderman', 'three',
'scene', 'descript', 'languag', 'creat', 'three', 'world', 'pleas', 'hel
p', 'use', 'renderman', 'librari', 'next', 'document', 'nextstep', 'versi
 on', 'renderman', 'avail', 'creat', 'complic', 'scene', 'render', 'use',
 'surfac', 'shader', 'bring', 'life', 'appli', 'shadow', 'reflect', 'far',
'understand', 'defin', 'environment', 'shadow', 'map', 'produc', 'reflec
t', 'shadow', 'know', 'use', 'advis', 'simpl', 'rib', 'exampl', 'apprec
i', 'thank', 'advanc', 'alex', 'kolesov', 'moscow', 'russia', 'talu', 'im
ag', 'commun', 'corpor', 'mail', 'alex', 'talu', 'msk', 'su', 'next', 'ma
 il', 'accept'], ['anybodi', 'know', 'good', 'two', 'graphic', 'packag',
'avail', 'ibm', 'rs', 'six', 'zero', 'zero', 'zero', 'aix', 'look', 'some
th', 'like', 'dec', 'gk', 'hewlett', 'packard', 'starbas', 'reason', 'goo
 d', 'support', 'differ', 'output', 'devic', 'like', 'plotter', 'termin',
'etc', 'tri', 'also', 'xgk', 'one', 'one', 'distribut', 'ibm', 'implemen
t', 'phig', 'work', 'requir', 'output', 'devic', 'window', 'salesman', 'i
bm', 'familiar', 'graphic', 'expect', 'good', 'solut', 'ari', 'ari', 'suu
tari', 'ari', 'carel', 'fi', 'carelcomp', 'oy', 'lappeenranta', 'finlan
[['d
recent', 'got', 'file', 'describ', 'librari', 'render', 'routin', 'cal']]
l', 'sipp', 'simpl', 'polygon', 'processor', 'could', 'anyon', 'tell', 'f
```

```
tp', 'sourc', 'code', 'newest', 'version', 'around', 'also', 've', 'neve
r', 'use', 'renderman', 'wonder', 'renderman', 'like', 'sipp', 'ie', 'lib
rari', 'render', 'routin', 'one', 'use', 'make', 'program', 'creat', 'ima
g', 'thank', 'joe', 'tham', 'joe', 'tham', 'joth', 'ersi', 'edmonton', 'a
 b', 'ca'], ['hello', 'everybodi', 'use', 'pixar', 'renderman', 'three',
'scene', 'descript', 'languag', 'creat', 'three', 'world', 'pleas', 'hel
p', 'use', 'renderman', 'librari', 'next', 'document', 'nextstep', 'versi
 on', 'renderman', 'avail', 'creat', 'complic', 'scene', 'render', 'use',
 'surfac', 'shader', 'bring', 'life', 'appli', 'shadow', 'reflect', 'far',
'understand', 'defin', 'environment', 'shadow', 'map', 'produc', 'reflec
t', 'shadow', 'know', 'use', 'advis', 'simpl', 'rib', 'exampl', 'apprec
i', 'thank', 'advanc', 'alex', 'kolesov', 'moscow', 'russia', 'talu', 'im
ag', 'commun', 'corpor', 'mail', 'alex', 'talu', 'msk', 'su', 'next', 'ma
 il', 'accept'], ['anybodi', 'know', 'good', 'two', 'graphic', 'packag',
'avail', 'ibm', 'rs', 'six', 'zero', 'zero', 'zero', 'aix', 'look', 'some
th', 'like', 'dec', 'gk', 'hewlett', 'packard', 'starbas', 'reason', 'goo
 d', 'support', 'differ', 'output', 'devic', 'like', 'plotter', 'termin',
'etc', 'tri', 'also', 'xgk', 'one', 'one', 'distribut', 'ibm', 'implemen
t', 'phig', 'work', 'requir', 'output', 'devic', 'window', 'salesman', 'i
bm', 'familiar', 'graphic', 'expect', 'good', 'solut', 'ari', 'ari', 'suu
tari', 'ari', 'carel', 'fi', 'carelcomp', 'oy', 'lappeenranta', 'finlan
d'], ['requir', 'bgi', 'driver', 'super', 'vga', 'display', 'super', 'xvg
a', 'display', 'anyon', 'know', 'could', 'obtain', 'relev', 'driver', 'ft
[['p', 'site', 'regard', 'simon', 'crow
```

:[12] In

```
DF = {}
N = len(processed_text)
for i in range(N):
    tokens = processed_text[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}
for i in DF:
    DF[i] = len(DF[i])
```

:[13] In

```
total_voca=len(DF)
print(total_voca)
```

```
144
```

:[33] In

```
def doc_freq(word):
    c = 0
    try:
        c = DF[word]
    except:
        pass
    return c

word = "path"

print("word:", word, "-->frequency", doc_freq(word))
```

```
word: path -->frequency 0
```

:[34] In

```
doc = 0
tf_idf = {}
for i in range(N):
    tokens = processed_text[i]
    counter = Counter(tokens + processed_text[i])

    words_count = len(tokens + processed_text[i])

    for token in np.unique(tokens):
        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))
        tf_idf[doc, token] = tf*idf

    doc += 1

tf_idf
```

Out[34]:

```
                ,ab'): 0.018325814637483104' ,0)}
                ,also'): 0.010216512475319815' ,0)
                ,anyon'): 0.010216512475319815' ,0)
                ,around'): 0.018325814637483104' ,0)
                ,ca'): 0.018325814637483104' ,0)
                ,call'): 0.018325814637483104' ,0)
                ,code'): 0.018325814637483104' ,0)
                ,could'): 0.010216512475319815' ,0)
                ,creat'): 0.010216512475319815' ,0)
                ,describ'): 0.018325814637483104' ,0)
                ,edmonton'): 0.018325814637483104' ,0)
                ,ersi'): 0.018325814637483104' ,0)
                ,file'): 0.018325814637483104' ,0)
                ,ftp'): 0.010216512475319815' ,0)
                ,got'): 0.018325814637483104' ,0)
                ,ie'): 0.018325814637483104' ,0)
                ,imag'): 0.010216512475319815' ,0)
                ,joe'): 0.03665162927496621' ,0)
```

:[35] In

```
tf_idf
```

Out[35]:

```
,ab'): 0.018325814637483104' ,0)}
,also'): 0.010216512475319815' ,0)
,anyon'): 0.010216512475319815' ,0)
,around'): 0.018325814637483104' ,0)
,ca'): 0.018325814637483104' ,0)
,call'): 0.018325814637483104' ,0)
,code'): 0.018325814637483104' ,0)
,could'): 0.010216512475319815' ,0)
,creat'): 0.010216512475319815' ,0)
,describ'): 0.018325814637483104' ,0)
,edmonton'): 0.018325814637483104' ,0)
,ersi'): 0.018325814637483104' ,0)
,file'): 0.018325814637483104' ,0)
,ftp'): 0.010216512475319815' ,0)
,got'): 0.018325814637483104' ,0)
,ie'): 0.018325814637483104' ,0)
,imag'): 0.010216512475319815' ,0)
,ioe'): 0.03665162927496621' ,0)
```

:[36] In

```
tf_idf[(0,'also')]
```

Out[36]:

```
0.010216512475319815
```

:[37] In

```
DF
```

Out[37]:

```
,recent': 1'}
,got': 1'
,file': 1'
,describ': 1'
,librari': 2'
,render': 2'
,routin': 1'
,call': 1'
,sipp': 1'
,simpl': 2'
,polygon': 1'
,processor': 1'
,could': 2'
,anyon': 2'
,tell': 1'
,ftp': 2'
,sourc': 1'
.code': 1'
```

:[45] In

```
def matching_score(k, query):
    preprocessed_query = preprocess(query)
    tokens = word_tokenize(str(preprocessed_query))

    print("Matching Score")
    print("\nQuery:", query)
    print("")
    print(tokens)

    query_weights = {}

    for key in tf_idf:

        if key[1] in tokens:
            try:
                query_weights[key[0]] += tf_idf[key]
            except:
                query_weights[key[0]] = tf_idf[key]


    query_weights = sorted(query_weights.items(), key=lambda x: x[1],reverse=True)

    print("")

    l = []

    for i in query_weights[:k]:
        l.append(i[0])

    print(l)

matching_score(2,"I recently got a file describing a library")
```

```
No Header
Matching Score

Query: I recently got a file describing a library

['recent', 'got', 'file', 'describ', 'librari']

[1 ,0]
```

:[47] In

```
title = "comp.graphics"
os.chdir(r'C:\Users\D7me_\mini_newsgroups\mini_newsgroups\comp.graphics')
paths = []
for (dirpath, dirnames, filenames) in os.walk(str(os.getcwd())+'/'+title+'/'):
    for i in filenames:
        paths.append(str(dirpath)+str("\\")+i)

processed_text = []
for i in range(len(filenames)):
    file = open(dirpath+'/'+ filenames[i], 'r', encoding='cp1250', errors='ignore')
    text = file.read().strip()
    file.close()
    processed_text.append(word_tokenize(str(preprocess(text))))


DF = {}
N = len(processed_text)

for i in range(N):
    tokens = processed_text[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}

for i in DF:
    DF[i] = len(DF[i])


doc = 0
tf_idf = {}
for i in range(N):
    tokens = processed_text[i]
    counter = Counter(tokens + processed_text[i])
    words_count = len(tokens + processed_text[i])

    for token in np.unique(tokens):
        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))
        tf_idf[doc, token] = tf*idf

    doc += 1

tf_idf
```

Out[47]:

```
,ab'): 0.018325814637483104' ,0)}
,also'): 0.010216512475319815' ,0)
,anyon'): 0.010216512475319815' ,0)
,around'): 0.018325814637483104' ,0)
,ca'): 0.018325814637483104' ,0)
,call'): 0.018325814637483104' ,0)
,code'): 0.018325814637483104' ,0)
,could'): 0.010216512475319815' ,0)
,creat'): 0.010216512475319815' ,0)
```

```
,describ'): 0.018325814637483104' ,0)
,edmonton'): 0.018325814637483104' ,0)
,ersi'): 0.018325814637483104' ,0)
,file'): 0.018325814637483104' ,0)
,ftp'): 0.010216512475319815' ,0)
,got'): 0.018325814637483104' ,0)
,ie'): 0.018325814637483104' ,0)
,imag'): 0.010216512475319815' ,0)
```

:[ ] In