In [1]:

```python
"""
Created on Mon Nov  7 18:05:34 2022

@author:Brandon Botzer - btb5103



1.Use the following codes to load the assignment12.txt
    which contains file names. How many file names are in it? (10 points)

file = open("Assignment_12.txt" , 'r')

text1 = file.read()

file.close()



2.Identify the pattern of the file names,  and find out how many file names
match the pattern. (20 points)


3.Find out file names who don't match with the pattern you designed.(20 points)


4.Use the following codes to read the text from
    "arxiv_annotate1_13_1.txtDownload arxiv_annotate1_13_1.txt"
in

file = open("arxiv_annotate1_13_1.txt", 'r')

text = file.read()

file.close()

Normalize the words and find out their counts.
"""

#imports
import os
import nltk
import re
import pandas as pd
```

In [2]:

```python
#1.Use the following codes to load the assignment12.txt
#    which contains file names. How many file names are in it? (10 points)

#set the read path
readpath = "J:\DSDegree\PennState\DAAN_862\Week 12\Homework"
#change the directory
os.chdir(readpath)

file_name = "Assignment_12.txt"

#open the file
file = open(file_name , 'r')
#grab the text as a string
text1 = file.read()
#close the file
file.close()

#use regex to split text1 by any number of blank spaces
file_list = re.split('\s+', text1)

#number of filenames in text1
num_filenames = len(file_list)

print("\nSplitting the files by spaces, there are " + str(num_filenames) +
      " files in " + file_name)
```

```
Splitting the files by spaces, there are 90 files in Assignment_12.txt
```

In [3]:

```python
#2.Identify the pattern of the file names,  and find out how many file names
#match the pattern. (20 points)

#The commented section here works as well as the function below
"""
#decleare the pattern
pattern = r'[a-zA-Z]+_[a-zA-Z]+[0-9]+_[0-9]+_[0-9]+\.[a-zA-Z]+'
#build the regex object with the pattern
regex = re.compile(pattern, flags = re.IGNORECASE)
#Get the list of cases that match the pattern
reg_list = regex.findall(text1)
#find the length of the list of  matching cases
reg_len = len(reg_list)

print("Using Regex, there are " + str(reg_len) + " files in " + file_name)
print("This was due to typos in some of the file names.")
"""

#I tried to use the regex-generator for the pattern
#but it would not work correctly.  I used it as a start and then
#modified it.

#Pattern from regex-generator shown here:
    #r"^[a-zA-Z]+_[a-zA-Z]+([0-9]+(_[0-9]+)+)\.[a-zA-Z]+$"

def use_regex(input_text):
    #compile the regex object
    regex = re.compile(r"[a-zA-Z]+_[a-zA-Z]+[0-9]+_[0-9]+_[0-9]+\.[a-zA-Z]+",
                       re.IGNORECASE)
    #find all cases that match the pattern and return them
    return regex.findall(input_text)

#Get the list of cases that match the pattern
reg_list = use_regex(text1)
#find the length of the list of  matching cases
reg_len = len(reg_list)

print("\nUsing Regex, there are " + str(reg_len) + " files in " + file_name)
print("This was due to typos in some of the file names.")
```

```
Using Regex, there are 84 files in Assignment_12.txt
This was due to typos in some of the file names.
```

In [4]:

```python
#3.Find out file names who don't match with the pattern you designed.(20 points)

#Use sets to effectivly look for the missing pieces

#Bring the lists into sets
set_a = set(file_list)
set_b = set(reg_list)

#Full set - partial set = missing files  (similar to an outer join)
#put it back into a list so all of my lists of files are var type list
typo_list = list(set_a - set_b)

print("\nThe files which were spelled incorrectly are: \n")
print(typo_list)
```

```
The files which were spelled incorrectly are:

['jdm_annotat#e8_177_2.txt', 'jdm_ann^otate3_120_1.txt', 'plos_annotat*e1_6_2.txt', 'plos_anno%tate5_1375_3.tx
t', 'jdm_anno&tate6_32_2.txt', 'plos_annot@ate7_1233_2.txt']
```

In [5]:

```python
#4.Use the following codes to read the text from
#    "arxiv_annotate1_13_1.txt"

#open and read in the file
file_name2 = "arxiv_annotate1_13_1.txt"
file = open(file_name2, 'r')
text2 = file.read()
file.close()


#Normalize the words and find out their counts.


#First we'll remove special characters
#Taken from "Text Analytics with Python, 2nd ed."
def remove_special_characters(text, remove_digits = False):
    pattern = r'[^a-zA-Z0-9\s]' if not remove_digits else r'[^a-zA-Z\s]'
    text = re.sub(pattern, '', text)
    return text

text2 = remove_special_characters(text2)
```

In [6]:

```python
#break the large string up into a list of words
#use regex to split text1 by any number of blank spaces
words = re.split('\s+', text2)
#Try a nltk word tokenizer for possible different results
words_token = nltk.word_tokenize(text2)
#Try a split() function for possible differenet results
words_split = text2.split(' ')

#I will go with the word_tokenizer as it removed the empty string
```

In [7]:

```python
#find some word counts
#number of words in the list
num_words = len(words_token)
#unique words
unique_words = list(set(words_token))
#number of unique words
num_unique_words = len(unique_words)

print("\nFrom the Arxiv file, there are by the NLTK word_tokenizer:")
print("  " + str(num_words) + " words.")
print("  " + str(num_unique_words) + " unique words.\n")
```

```
From the Arxiv file, there are by the NLTK word_tokenizer:
  844 words.
  333 unique words.
```

In [8]:

```python
#we can also do this with a FreqDist
from nltk.probability import FreqDist
from nltk.tokenize import word_tokenize


#find the unique words from the word list
dist = FreqDist(words_token)
print("The FreqDist function likewise reports "
      + str(len(dist)) + " unique words.")


print("\n\nSome of these words may come from the same root stem/word.")
print("We'll look at both root stems and root words here.\n")
```

```
The FreqDist function likewise reports 333 unique words.


Some of these words may come from the same root stem/word.
We'll look at both root stems and root words here.
```

In [9]:

```python
#Root Stemming
#build the Porter Stemmer object
porter = nltk.PorterStemmer()
#Get the root stems for each word
word_stems = [porter.stem(word) for word in words_token]

#Get the distribution of the stem'd words
stem_dist = FreqDist(word_stems)


#Download for functionallity if needed
nltk.download('omw-1.4')

#Lemmatization
#Build the WordNetLemmatizer object
WNLemma = nltk.WordNetLemmatizer()
#Get the Lemmatization of the words
word_lemma = [WNLemma.lemmatize(word) for word in words_token]
#lemma distribution
lemma_dist = FreqDist(word_lemma)
```

```
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\btb51\anaconda3\lib\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [13]:

```python
#display the counts
#make a data frame of all of the different word grabs and counts

#dist
#stem_dist
#lemma_dist

#Get tuples of the distributions
tup_words = tuple(dist.items())
tup_stem = tuple(stem_dist.items())
tup_lemma = tuple(lemma_dist.items())

#create data frames and join them together...
#there is probably a better way to do this using a pivot but here we are.

twdf = pd.DataFrame(data = tup_words, columns = ["words", "word_count"])
tsdf = pd.DataFrame(data = tup_stem, columns = ["stems", "stem_count"])

#join the tuple stem data frame to the tuple word data frame on the left
#this keeps all values and will just join the frames together
df = twdf.join(tsdf, how='left', sort = True)

tldf = pd.DataFrame(data = tup_lemma, columns = ["lemmas", "lemma_count"])

#join the lemma data frame to the word/stem data frame on the left
#this keeps all values and will just join the frames together
df = df.join(tldf, how='left', sort = True)

print("\nThe word, stem, and lemma distribution counts:\n")
print("You'll notice that eventually the stem and lemma columns will " +
      "not match up with the words column as the tokenizers interpret " +
      "differently.  In the case of lemma, it missed the word 'a'.  " +
      "It also had trouble with strange breakes such as 'eg.' which " +
      "in the text file was presented as 'e g' and thus taken as two words.\n")

print("\nI attempted to make use of Dipanjan's sequence of text to compare " +
      "normalization results but there seems to be an issue with a " +
      "dependancy I was unable to resolve." )

#print all of the words, counts, stems, stem_counts, lemmas, and lemma_counts
print(df.to_string())
```

```
The word, stem, and lemma distribution counts:

You'll notice that eventually the stem and lemma columns will not match up with the words column as the tokeniz
ers interpret differently.  In the case of lemma, it missed the word 'a'.  It also had trouble with strange bre
akes such as 'eg.' which in the text file was presented as 'e g' and thus taken as two words.


I attempted to make use of Dipanjan's sequence of text to compare normalization results but there seems to be a
n issue with a dependancy I was unable to resolve.
```

	words	word_count	stems	stem_count	lemmas	lemma_count
0	abstract	1	abstract	1.0	abstract	1.0
1	MISC	20	misc	20.0	MISC	20.0
2	although	1	although	1.0	although	1.0
3	the	44	the	44.0	the	44.0
4	internet	15	internet	15.0	internet	15.0
5	as	28	as	28.0	a	47.0
6	level	6	level	6.0	level	6.0
7	topology	8	topolog	11.0	topology	11.0
8	has	4	ha	4.0	ha	4.0
9	been	1	been	1.0	been	1.0
10	extensively	1	extens	1.0	extensively	1.0
11	studied	1	studi	2.0	studied	1.0
12	over	2	over	2.0	over	2.0
13	past	1	past	1.0	past	1.0
14	few	2	few	2.0	few	2.0
15	years	1	year	1.0	year	1.0
16	little	1	littl	1.0	little	1.0
17	is	8	is	8.0	is	8.0
18	known	1	known	1.0	known	1.0
19	about	1	about	1.0	about	1.0
20	details	1	detail	1.0	detail	1.0
21	of	34	of	34.0	of	34.0
22	taxonomy	6	taxonomi	6.0	taxonomy	6.0
23	an	7	an	7.0	an	7.0
24	node	1	node	1.0	node	1.0
25	can	4	can	4.0	can	4.0
26	represent	1	repres	3.0	represent	1.0
27	a	19	a	19.0	wide	2.0
28	wide	2	wide	2.0	variety	1.0
29	variety	1	varieti	1.0	organization	2.0
30	organizations	2	organ	2.0	e	2.0
31	e	2	e	2.0	g	2.0
32	g	2	g	2.0	large	3.0
33	large	3	larg	3.0	isp	2.0
34	isp	2	isp	4.0	or	3.0
35	or	3	or	3.0	small	4.0
36	small	4	small	4.0	private	2.0
37	private	2	privat	2.0	business	3.0
38	business	3	busi	3.0	university	4.0
39	university	3	univers	4.0	with	9.0
40	with	9	with	9.0	vastly	1.0
41	vastly	1	vastli	1.0	different	12.0
42	different	12	differ	15.0	network	10.0
43	network	8	network	10.0	characteristic	2.0
44	characteristics	2	characterist	2.0	external	1.0
45	external	1	extern	1.0	connectivity	2.0
46	connectivity	2	connect	3.0	pattern	3.0
47	patterns	3	pattern	3.0	growth	4.0
48	growth	4	growth	4.0	tendency	1.0
49	tendencies	1	tendenc	1.0	and	24.0
50	and	24	and	24.0	other	6.0
51	other	6	other	6.0	property	2.0
52	properties	2	properti	2.0	that	12.0
53	that	12	that	12.0	we	20.0
54	we	20	we	20.0	hardly	1.0
55	hardly	1	hardli	1.0	neglect	1.0
56	neglect	1	neglect	1.0	while	1.0
57	while	1	while	1.0	working	1.0
58	working	1	work	3.0	on	9.0
59	on	9	on	9.0	veracious	1.0
60	veracious	1	veraci	1.0	representation	1.0
61	representations	1	represent	1.0	in	19.0
62	in	19	in	19.0	simulation	1.0
63	simulation	1	simul	1.0	environment	2.0
64	environments	2	environ	2.0	AIMX	3.0
65	AIMX	3	aimx	3.0	this	5.0
66	this	5	thi	5.0	paper	1.0
67	paper	1	paper	1.0	introduce	2.0
68	introduce	2	introduc	3.0	radically	2.0
69	radically	2	radic	2.0	new	3.0
70	new	3	new	3.0	approach	3.0
71	approach	3	approach	3.0	based	3.0
72	based	3	base	3.0	machine	3.0
73	machine	3	machin	3.0	learning	3.0
74	learning	3	learn	3.0	technique	2.0
75	techniques	1	techniqu	2.0	to	18.0
76	to	18	to	18.0	map	1.0
77	map	1	map	2.0	all	1.0
78	all	1	all	1.0	as	12.0
79	ases	12	ase	12.0	into	2.0
80	into	2	into	2.0	natural	1.0
81	natural	1	natur	1.0	OWNX	12.0
82	OWNX	12	ownx	12.0	successfully	1.0
83	successfully	1	success	1.0	classify	4.0
84	classify	4	classifi	5.0	NUMBER	15.0
85	NUMBER	15	number	16.0	percent	3.0
86	percent	3	percent	3.0	expected	1.0
87	expected	1	expect	2.0	accuracy	2.0
88	accuracy	2	accuraci	2.0	release	1.0
89	release	1	releas	1.0	community	1.0
90	community	1	commun	1.0	dataset	2.0
91	dataset	2	dataset	2.0	augmented	1.0
92	augmented	1	augment	3.0	information	2.0
93	information	2	inform	2.0	set	3.0
94	set	3	set	3.0	attribute	1.0
95	attributes	1	attribut	1.0	used	2.0
96	used	2	use	5.0	believe	1.0
97	believe	1	believ	1.0	will	3.0
98	will	3	will	3.0	serve	1.0
99	serve	1	serv	1.0	invaluable	1.0
100	invaluable	1	invalu	1.0	addition	1.0
101	addition	1	addit	1.0	further	2.0
102	further	2	further	2.0	understanding	3.0
103	understanding	3	understand	3.0	structure	3.0
104	structure	3	structur	4.0	evolution	5.0
105	evolution	5	evolut	5.0	introduction	1.0
106	introduction	1	introduct	1.0	rapid	1.0
107	rapid	1	rapid	1.0	expansion	1.0
108	expansion	1	expans	1.0	last	1.0
109	last	1	last	1.0	two	1.0
110	two	1	two	1.0	decade	1.0
111	decades	1	decad	1.0	produced	1.0
112	produced	1	produc	1.0	scale	1.0
113	scale	1	scale	1.0	system	2.0
114	system	2	system	2.0	thousand	2.0
115	thousands	2	thousand	2.0	diverse	1.0
116	diverse	1	divers	2.0	independently	1.0
117	independently	1	independ	1.0	managed	1.0
118	managed	1	manag	1.0	collectively	1.0
119	networks	2	collect	1.0	provide	1.0
120	collectively	1	provid	5.0	global	1.0
121	provide	1	global	2.0	across	1.0
122	global	1	across	1.0	spectrum	1.0
123	across	1	spectrum	1.0	geopolitical	1.0
124	spectrum	1	geopolit	1.0	from	6.0
125	geopolitical	1	from	6.0	number	1.0
126	from	6	routabl	1.0	globally	1.0
127	number	1	identifi	4.0	routable	1.0
128	globally	1	increas	1.0	identifier	1.0
129	routable	1	less	1.0	increased	1.0
130	identifiers	1	than	2.0	le	1.0
131	increased	1	more	1.0	than	2.0
132	less	1	exert	1.0	more	1.0
133	than	2	signific	1.0	exerting	1.0
134	more	1	pressur	1.0	significant	1.0
135	exerting	1	interdomain	1.0	pressure	1.0
136	significant	1	rout	2.0	interdomain	1.0
137	pressure	1	well	2.0	routing	2.0
138	interdomain	1	function	1.0	well	2.0
139	routing	2	part	1.0	functional	1.0
140	well	2	impress	1.0	structural	1.0
141	functional	1	result	4.0	part	1.0
142	structural	1	heterogen	1.0	impressive	1.0
143	parts	1	highli	1.0	resulted	1.0
144	impressive	1	complex	1.0	heterogenous	1.0
145	resulted	1	challeng	1.0	highly	1.0
146	heterogenous	1	accur	3.0	complex	1.0
147	highly	1	realist	2.0	challenge	1.0
148	complex	1	model	5.0	accurate	2.0
149	challenges	1	infrastructur	2.0	realistic	2.0
150	accurate	2	particular	1.0	modeling	4.0
151	realistic	2	intermix	1.0	infrastructure	2.0
152	modeling	4	own	1.0	particular	1.0
153	infrastructure	1	oper	1.0	intermix	1.0
154	particular	1	by	4.0	owned	1.0
155	intermix	1	mani	2.0	operated	1.0
156	owned	1	backbon	1.0	by	4.0
157	operated	1	region	1.0	many	2.0
158	by	4	access	1.0	backbone	1.0
159	many	2	compani	4.0	provider	4.0
160	backbone	1	statist	2.0	regional	1.0
161	providers	4	faith	1.0	access	1.0
162	regional	1	character	2.0	company	4.0
163	access	1	type	11.0	statistical	1.0
164	universities	1	critic	1.0	faithfully	1.0
165	companies	2	path	1.0	characterizes	1.0
166	statistical	1	toward	1.0	type	11.0
167	faithfully	1	for	7.0	critical	1.0
168	characterizes	1	it	3.0	path	1.0
169	types	10	knowledg	1.0	toward	1.0
170	critical	1	mandatori	1.0	for	7.0
171	path	1	synthet	1.0	it	3.0
172	toward	1	construct	2.0	knowledge	1.0
173	for	7	measur	1.0	mandatory	1.0
174	its	3	intra	1.0	augmenting	1.0
175	knowledge	1	inter	1.0	synthetically	1.0
176	mandatory	1	router	4.0	constructed	1.0
177	augmenting	1	exampl	3.0	measured	1.0
178	synthetically	1	dual	2.0	intra	1.0
179	constructed	1	home	3.0	inter	1.0
180	measured	1	be	1.0	router	4.0
181	topologies	3	drastic	1.0	example	3.0
182	intra	1	like	1.0	expect	1.0
183	inter	1	contain	1.0	dual	2.0
184	router	3	dozen	1.0	homed	2.0
185	example	3	intern	1.0	be	1.0
186	expect	1	host	1.0	drastically	1.0
187	dual	2	element	1.0	likely	1.0
188	homed	2	switch	1.0	contain	1.0
189	be	1	server	1.0	dozen	1.0
190	drastically	1	firewal	1.0	internal	1.0
191	company	2	hand	2.0	host	1.0
192	likely	1	most	1.0	element	1.0
193	contain	1	probabl	1.0	switch	1.0
194	dozens	1	have	1.0	server	1.0
195	internal	1	singl	1.0	firewall	1.0
196	routers	1	simpl	1.0	hand	2.0
197	hosts	1	sinc	2.0	most	1.0
198	elements	1	there	1.0	probably	1.0
199	switches	1	such	1.0	have	1.0
200	servers	1	among	1.0	single	1.0
201	firewalls	1	not	3.0	simple	1.0
202	hand	2	appropri	1.0	since	2.0
203	most	1	if	1.0	there	1.0
204	probably	1	compos	1.0	such	1.0
205	have	1	moreov	1.0	diversity	1.0
206	single	1	annot	1.0	among	1.0
207	simple	1	their	1.0	not	3.0
208	since	2	prerequisit	1.0	accurately	1.0
209	there	1	exhibit	1.0	augment	1.0
210	such	1	servic	1.0	appropriate	1.0
211	diversity	1	grow	2.0	if	1.0
212	among	1	attract	1.0	characterize	1.0
213	not	3	custom	1.0	composing	1.0
214	accurately	1	engag	1.0	moreover	1.0
215	augment	1	agreement	1.0	annotating	1.0
216	appropriate	1	through	1.0	their	1.0
217	if	1	one	1.0	prerequisite	1.0
218	characterize	1	do	1.0	exhibit	1.0
219	composing	1	significantli	1.0	service	1.0
220	moreover	1	time	1.0	grow	2.0
221	annotating	1	thu	1.0	attracting	1.0
222	their	1	categor	1.0	customer	1.0
223	prerequisite	1	necessari	2.0	engaging	1.0
224	exhibit	1	develop	2.0	agreement	1.0
225	service	1	also	1.0	isps	2.0
226	grow	2	ip	2.0	connect	1.0
227	attracting	1	address	2.0	through	1.0
228	customers	1	user	2.0	one	1.0
229	engaging	1	traffic	1.0	do	1.0
230	agreements	1	analysi	1.0	significantly	1.0
231	isps	2	often	1.0	time	1.0
232	connect	1	requir	1.0	thus	1.0
233	through	1	distinguish	1.0	categorizing	1.0
234	one	1	between	3.0	necessary	2.0
235	do	1	packet	1.0	identify	2.0
236	significantly	1	come	1.0	develop	2.0
237	time	1	given	1.0	model	1.0
238	thus	1	possibl	1.0	also	1.0
239	categorizing	1	realiz	1.0	mapping	1.0
240	necessary	2	goal	1.0	ip	2.0
241	identify	2	check	1.0	address	2.0
242	develop	2	origin	1.0	user	2.0
243	models	1	prefix	1.0	traffic	1.0
244	also	1	which	1.0	analysis	1.0
245	mapping	1	lie	1.0	study	1.0
246	ip	2	algorithm	3.0	often	1.0
247	addresses	1	empir	1.0	required	1.0
248	users	2	observ	1.0	distinguish	1.0
249	traffic	1	data	2.0	between	3.0
250	analysis	1	registri	1.0	packet	1.0
251	studies	1	irr	1.0	come	1.0
252	often	1	citat	2.0	home	1.0
253	required	1	routeview	1.0	given	1.0
254	distinguish	1	intrins	1.0	possible	1.0
255	between	3	then	1.0	realize	1.0
256	packets	1	employ	1.0	goal	1.0
257	come	1	novel	1.0	checking	1.0
258	home	1	build	1.0	originates	1.0
259	given	1	classif	4.0	prefix	1.0
260	possible	1	exploit	1.0	which	1.0
261	realize	1	these	1.0	lie	1.0
262	goal	1	six	1.0	work	2.0
263	checking	1	class	2.0	construct	1.0
264	type	1	reflect	1.0	representative	2.0
265	originates	1	deriv	1.0	algorithm	3.0
266	prefix	1	macroscop	1.0	empirically	1.0
267	which	1	valid	3.0	observed	1.0
268	address	1	our	7.0	difference	3.0
269	lies	1	sampl	1.0	use	2.0
270	work	2	manual	1.0	data	2.0
271	construct	1	demonstr	1.0	registry	1.0
272	representative	2	achiev	1.0	irr	1.0
273	algorithm	3	high	1.0	CITATION	2.0
274	empirically	1	examin	1.0	routeviews	1.0
275	observed	1	were	1.0	intrinsic	1.0
276	differences	3	correct	1.0	then	1.0
277	use	2	final	1.0	employ	1.0
278	data	2	make	1.0	novel	1.0
279	registries	1	publicli	1.0	build	1.0
280	irr	1	avail	1.0	classification	4.0
281	CITATION	2	promot	1.0	exploit	1.0
282	routeviews	1	research	1.0	these	1.0
283	intrinsic	1	s	1.0	six	1.0
284	then	1	section	6.0	class	2.0
285	employ	1	start	1.0	reflect	1.0
286	novel	1	brief	1.0	derive	1.0
287	technique	1	discuss	1.0	macroscopic	1.0
288	build	1	relat	1.0	statistic	1.0
289	classification	3	describ	1.0	validate	2.0
290	exploits	1	specifi	1.0	our	7.0
291	these	1	experi	1.0	result	3.0
292	six	1	them	1.0	using	1.0
293	classes	2	conclud	1.0	sample	1.0
294	reflect	1	NaN	NaN	manually	1.0
295	infrastructures	1	NaN	NaN	identified	1.0
296	derive	1	NaN	NaN	validation	1.0
297	macroscopic	1	NaN	NaN	demonstrates	1.0
298	statistics	1	NaN	NaN	achieves	1.0
299	validate	2	NaN	NaN	high	1.0
300	our	7	NaN	NaN	examined	1.0
301	results	3	NaN	NaN	were	1.0
302	using	1	NaN	NaN	correct	1.0
303	sample	1	NaN	NaN	finally	1.0
304	manually	1	NaN	NaN	make	1.0
305	identified	1	NaN	NaN	classifier	1.0
306	validation	1	NaN	NaN	publicly	1.0
307	demonstrates	1	NaN	NaN	available	1.0
308	achieves	1	NaN	NaN	promote	1.0
309	high	1	NaN	NaN	research	1.0
310	examined	1	NaN	NaN	s	1.0
311	classifications	1	NaN	NaN	section	6.0
312	were	1	NaN	NaN	start	1.0
313	correct	1	NaN	NaN	brief	1.0
314	finally	1	NaN	NaN	discussion	1.0
315	make	1	NaN	NaN	related	1.0
316	classifier	1	NaN	NaN	describes	1.0
317	publicly	1	NaN	NaN	specify	1.0
318	available	1	NaN	NaN	experiment	1.0
319	promote	1	NaN	NaN	introduces	1.0
320	research	1	NaN	NaN	them	1.0
321	s	1	NaN	NaN	conclude	1.0
322	section	6	NaN	NaN	NaN	NaN
323	start	1	NaN	NaN	NaN	NaN
324	brief	1	NaN	NaN	NaN	NaN
325	discussion	1	NaN	NaN	NaN	NaN
326	related	1	NaN	NaN	NaN	NaN
327	describes	1	NaN	NaN	NaN	NaN
328	specify	1	NaN	NaN	NaN	NaN
329	experiments	1	NaN	NaN	NaN	NaN
330	introduces	1	NaN	NaN	NaN	NaN
331	them	1	NaN	NaN	NaN	NaN
332	conclude	1	NaN	NaN	NaN	NaN

In [11]:

```python
#^^^^ Here ends the Homework Submission ^^^^
```

In [12]:

```python
#Contined here is the non-workable Dipanjan sequence I had dependancy issues with.
"""

#Here I now bring in Dipanjan's sequence of text normalization
#IT is having issues with spaCy and using fileno for file imports

#Strip HTML
import re
from bs4 import BeautifulSoup

def strip_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    [s.extract() for s in soup(['iframe', 'script'])]
    stripped_text = soup.get_text()
    stripped_text = re.sub(r'[\r|\n|\r\n]+', '\n', stripped_text)
    return stripped_text


#Removing Accented Characters
import unicodedata

def remove_accented_chars(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text

#Expanding Contractions
from contractions import contractions_dict
import re

def expand_contractions(text, contraction_mapping=contractions_dict):

    contractions_pattern = re.compile('({})'.format('|'.join(contraction_mapping.keys())),
                                      flags=re.IGNORECASE|re.DOTALL)
    def expand_match(contraction):
        match = contraction.group(0)
        first_char = match[0]
        expanded_contraction = contraction_mapping.get(match)\
                                if contraction_mapping.get(match)\
                                else contraction_mapping.get(match.lower())
        expanded_contraction = first_char+expanded_contraction[1:]
        return expanded_contraction

    expanded_text = contractions_pattern.sub(expand_match, text)
    expanded_text = re.sub("'", "", expanded_text)
    return expanded_text

#Remvoing Special Characters
#You can find this above

#Case Conversion
#Use the .lower()

#Text Correction:

    #Correcting Repeating Characters
from nltk.corpus import wordnet
def remove_repeated_characters(tokens):
    repeat_pattern = re.compile(r'(\w*)(\w)\2(\w*)')
    match_substitution = r'\1\2\3'
    def replace(old_word):
        if wordnet.synsets(old_word):
            return old_word
        new_word = repeat_pattern.sub(match_substitution, old_word)
        return replace(new_word) if new_word != old_word else new_word

    correct_tokens = [replace(word) for word in tokens]
    return correct_tokens




    #Correcting Spellings
import re, collections

#make the vocabulary
def tokens(text):
    """
    #Get all words from the corpus
"""
    return re.findall('[a-z]+', text.lower())

WORDS = words
WORD_COUNTS = collections.Counter(WORDS)

#Define set of words that are one, two, or threee away from out input
def edits0(word):
    """
    #Return all strings that are zero edits away
    #from the input word (i.e., the word itself).
"""
    return {word}



def edits1(word):
    """
    #Return all strings that are one edit away
    #from the input word.
"""
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    def splits(word):
        """
        #Return a list of all possible (first, rest) pairs
        #that the input word is made of.
"""
        return [(word[:i], word[i:])
                for i in range(len(word)+1)]

    pairs      = splits(word)
    deletes    = [a+b[1:]           for (a, b) in pairs if b]
    transposes = [a+b[1]+b[0]+b[2:] for (a, b) in pairs if len(b) > 1]
    replaces   = [a+c+b[1:]         for (a, b) in pairs for c in alphabet if b]
    inserts    = [a+c+b             for (a, b) in pairs for c in alphabet]
    return set(deletes + transposes + replaces + inserts)


def edits2(word):
    """
    #Return all strings that are two edits away
    #from the input word.
"""
    return {e2 for e1 in edits1(word) for e2 in edits1(e1)}

#Define function to return a subset of words from our candidate set of words
#obtained from the edit functions based on if they occur in the vocabuary
#of the tokenfunction
def known(words):
    """
    #Return the subset of words that are actually
    #in our WORD_COUNTS dictionary.
"""
    return {w for w in words if w in WORD_COUNTS}

#Select the correct word from a number of candidates
def correct(word):
    """
    #Get the best correct spelling for the input word
"""
    # Priority is for edit distance 0, then 1, then 2
    # else defaults to the input word itself.
    candidates = (known(edits0(word)) or
                  known(edits1(word)) or
                  known(edits2(word)) or
                  [word])
    return max(candidates, key=WORD_COUNTS.get)

#We now go back and check case and ensure a correct match
#and show the correct text
def correct_match(match):
    """
    #Spell-correct word in match,
    #and preserve proper upper/lower/title case.
"""

    word = match.group()
    def case_of(text):
        """
        #Return the case-function appropriate
        #for text: upper, lower, title, or just str.:
"""
        return (str.upper if text.isupper() else
                str.lower if text.islower() else
                str.title if text.istitle() else
                str)
    return case_of(word)(correct(word.lower()))


def correct_text_generic(text):
    """
    #Correct all the words within a text,
    #returning the corrected text.
"""
    return re.sub('[a-zA-Z]+', correct_match, text)



    #word correction for spelling can also be done with the textblob library
#from textblob import Word
#w = Word('flaot')
#w.spellcheck()


#Stemming Words
def simple_stemmer(text):
    ps = nltk.porter.PorterStemmer()
    text = ' '.join([ps.stem(word) for word in text.split()])
    return text


#Lemmatization The book's version is having issues so I'll make my own
#import spacy
# use spacy.load('en') if you have downloaded the language model en directly after install spacy
#nlp = spacy.load('en_core', parse=True, tag=True, entity=True)
#text = 'My system keeps crashing his crashed yesterday, ours crashes daily'

#def lemmatize_text(text):
#    text = nlp(text)
#    text = ' '.join([word.lemma_ if word.lemma_ != '-PRON-' else word.text for word in text])
#    return text


def lemmatize_text_botzer(text):
    WNLemma = nltk.WordNetLemmatizer()

    #tokenize text
    words_token = nltk.word_tokenize(text)
    #lemmatize
    word_lemma = [WNLemma.lemmatize(word) for word in words_token]

    text = ' '.join(word_lemma)
    return text



#Remmove Stopwords
from nltk.tokenize.toktok import ToktokTokenizer
tokenizer = ToktokTokenizer()
stopword_list = nltk.corpus.stopwords.words('english')
def remove_stopwords(text, is_lower_case=False, stopwords=stopword_list):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopwords]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopwords]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text



#All of the Dipanjan funcitons build into one normalizer
def normalize_corpus(corpus, html_stripping=True, contraction_expansion=True,
                     accented_char_removal=True, text_lower_case=True,
                     text_stemmer=True, special_char_removal=True,
                     stopword_removal=True, remove_digits=True):

    normalized_corpus = []
    # normalize each document in the corpus
    for doc in corpus:
        # strip HTML
        if html_stripping:
            doc = strip_html_tags(doc)
        # remove accented characters
        if accented_char_removal:
            doc = remove_accented_chars(doc)
        # expand contractions
        if contraction_expansion:
            doc = expand_contractions(doc)
        # lowercase the text
        if text_lower_case:
            doc = doc.lower()
        # remove extra newlines
        doc = re.sub(r'[\r|\n|\r\n]+', ' ',doc)
        # Stem text
        if text_stemmer:
            doc = simple_stemmer(doc)
        # remove special characters and\or digits
        if special_char_removal:
            # insert spaces between special characters to isolate them
            special_char_pattern = re.compile(r'([{.(-)!}])')
            doc = special_char_pattern.sub(" \\1 ", doc)
            doc = remove_special_characters(doc, remove_digits=remove_digits)
        # remove extra whitespace
        doc = re.sub(' +', ' ', doc)
        # remove stopwords
        if stopword_removal:
            doc = remove_stopwords(doc, is_lower_case=text_lower_case)

        normalized_corpus.append(doc)

    return normalized_corpus




#and this doesn't work as intended...
test = normalize_corpus(text2)


"""
```

```
Out[12]: '\n    return re.sub(\'[a-zA-Z]+\', correct_match, text)\n\n\n\n    #word correction for spelling can also be d
one with the textblob library\n#from textblob import Word\n#w = Word(\'flaot\')\n#w.spellcheck()\n\n\n#Stemming
Words\ndef simple_stemmer(text):\n    ps = nltk.porter.PorterStemmer()\n    text = \' \'.join([ps.stem(word) fo
r word in text.split()])\n    return text\n\n\n#Lemmatization The book\'s version is having issues so I\'ll mak
e my own\n#import spacy\n# use spacy.load(\'en\') if you have downloaded the language model en directly after i
nstall spacy\n#nlp = spacy.load(\'en_core\', parse=True, tag=True, entity=True)\n#text = \'My system keeps cras
hing his crashed yesterday, ours crashes daily\'\n\n#def lemmatize_text(text):\n#    text = nlp(text)\n#    tex
t = \' \'.join([word.lemma_ if word.lemma_ != \'-PRON-\' else word.text for word in text])\n#    return text\n
\n\ndef lemmatize_text_botzer(text):\n    WNLemma = nltk.WordNetLemmatizer()\n    \n    #tokenize text\n    wor
ds_token = nltk.word_tokenize(text)\n    #lemmatize\n    word_lemma = [WNLemma.lemmatize(word) for word in word
s_token]\n    \n    text = \' \'.join(word_lemma)\n    return text\n    \n\n\n#Remmove Stopwords\nfrom nltk.tok
enize.toktok import ToktokTokenizer\ntokenizer = ToktokTokenizer()\nstopword_list = nltk.corpus.stopwords.words
(\'english\')\ndef remove_stopwords(text, is_lower_case=False, stopwords=stopword_list):\n    tokens = tokenize
r.tokenize(text)\n    tokens = [token.strip() for token in tokens]\n    if is_lower_case:\n        filtered_tok
ens = [token for token in tokens if token not in stopwords]\n    else:\n        filtered_tokens = [token for to
ken in tokens if token.lower() not in stopwords]\n    filtered_text = \' \'.join(filtered_tokens)    \n    retu
rn filtered_text\n\n\n\n#All of the Dipanjan funcitons build into one normalizer\ndef normalize_corpus(corpus,
html_stripping=True, contraction_expansion=True,\n                     accented_char_removal=True, text_lower_c
ase=True, \n                     text_stemmer=True, special_char_removal=True, \n                     stopword_
removal=True, remove_digits=True):\n    \n    normalized_corpus = []\n    # normalize each document in the corp
us\n    for doc in corpus:\n        # strip HTML\n        if html_stripping:\n            doc = strip_html_tags
(doc)\n        # remove accented characters\n        if accented_char_removal:\n            doc = remove_accent
ed_chars(doc)\n        # expand contractions    \n        if contraction_expansion:\n            doc = expand_c
ontractions(doc)\n        # lowercase the text    \n        if text_lower_case:\n            doc = doc.lower()
\n        # remove extra newlines\n        doc = re.sub(r\'[\r|\n|\r\n]+\', \' \',doc)\n        # Stem text\n
      if text_stemmer:\n            doc = simple_stemmer(doc)\n        # remove special characters and\\or digi
ts    \n        if special_char_removal:\n            # insert spaces between special characters to isolate the
m    \n            special_char_pattern = re.compile(r\'([{.(-)!}])\')\n            doc = special_char_pattern.
sub(" \\1 ", doc)\n            doc = remove_special_characters(doc, remove_digits=remove_digits)  \n        # r
emove extra whitespace\n        doc = re.sub(\' +\', \' \', doc)\n        # remove stopwords\n        if stopwo
rd_removal:\n            doc = remove_stopwords(doc, is_lower_case=text_lower_case)\n            \n        norm
alized_corpus.append(doc)\n        \n    return normalized_corpus\n\n\n\n\n#and this doesn\'t work as intende
d...\ntest = normalize_corpus(text2)\n\n\n'
```