```
[22] import nltk
     from nltk.corpus import stopwords
     import csv
     from nltk.tag import pos_tag # for proper noun
     from nltk.tokenize import word_tokenize, sent_tokenize
     import pandas as pd
     import math
     from nltk.stem import PorterStemmer
```

```
[54] filename="/008.txt"
     f = open((filename), "r")
     text=f.read() #append each line in the file to a list
     f.close()
```

```
[55]
     nltk.download('punkt')
     nltk.download('averaged_perceptron_tagger')
     nltk.download('stopwords')
```

```
     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Package punkt is already up-to-date!
     [nltk_data] Downloading package averaged_perceptron_tagger to
     [nltk_data]     /root/nltk_data...
     [nltk_data]   Package averaged_perceptron_tagger is already up-to-
     [nltk_data]       date!
     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Package stopwords is already up-to-date!
     True
```

```
[56] sent_tokens = nltk.sent_tokenize(text)
     word_tokens = nltk.word_tokenize(text)
     word_tokens_lower=[word.lower() for word in word_tokens]
     stopWords = list(set(stopwords.words("english")))
     word_tokens_refined=[x for x in word_tokens_lower if x not in stopWords]
     print(len(word_tokens_refined))
```

```
     310
```

```
[57] stem = []
     ps = PorterStemmer()
     for w in word_tokens_refined:
         stem.append(ps.stem(w))
     word_tokens_refined=stem
```

```
[58] # ......................part 1 (cue phrases).................
     QPhrases=["incidentally", "example", "anyway", "furthermore","according"
               "first", "second", "then", "now", "thus", "moreover", "therefore", "hence", "lastly", "finally", "summary"]

     cue_phrases={}
     for sentence in sent_tokens:
         cue_phrases[sentence] = 0
         word_tokens = nltk.word_tokenize(sentence)
         for word in word_tokens:
             if word.lower() in QPhrases:
                 cue_phrases[sentence] += 1
     maximum_frequency = max(cue_phrases.values())
     for k in cue_phrases.keys():
         try:
             cue_phrases[k] = cue_phrases[k] / maximum_frequency
             cue_phrases[k]=round(cue_phrases[k],3)
         except ZeroDivisionError:
             x=0
     print(cue_phrases.values())
```

```
     dict_values([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
[59] # ......................part2 (numerical data)...................
     numeric_data={}
     for sentence in sent_tokens:
         numeric_data[sentence] = 0
         word_tokens = nltk.word_tokenize(sentence)
         for k in word_tokens:
             if k.isdigit():
                 numeric_data[sentence] += 1
     maximum_frequency = max(numeric_data.values())
     for k in numeric_data.keys():
         try:
             numeric_data[k] = (numeric_data[k]/maximum_frequency)
             numeric_data[k] = round(numeric_data[k], 3)
         except ZeroDivisionError:
             x=0
```

```
print(numeric_data.values())
```

```
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0])
```

[60]
```
#.....................part3(sentence length).........................
sent_len_score={}
for sentence in sent_tokens:
    sent_len_score[sentence] = 0
    word_tokens = nltk.word_tokenize(sentence)
    if len(word_tokens) in range(0,10):
        sent_len_score[sentence]=1-0.05*(10-len(word_tokens))
    elif len(word_tokens) in range(7,20):
        sent_len_score[sentence]=1
    else:
        sent_len_score[sentence]=1-(0.05)*(len(word_tokens)-20)
for k in sent_len_score.keys():
    sent_len_score[k]=round(sent_len_score[k],4)
print(sent_len_score.values())
```

```
dict_values([1, 0.8, 0.85, 1, 1, 1, 0.65, 0.95, 0.85, 1, 0.6, 0.2, 0.95, 1, 0.65, 0.4, 0.35, 1.0, 1, 0.7, 1, 1])
```

[61]
```
#.....................part4(sentence position).........................
sentence_position={}
d=1
no_of_sent=len(sent_tokens)
for i in range(no_of_sent):
    a=1/d
    b=1/(no_of_sent-d+1)
    sentence_position[sent_tokens[d-1]]=max(a,b)
    d=d+1
for k in sentence_position.keys():
    sentence_position[k]=round(sentence_position[k],3)
print(sentence_position.values())
```

```
dict_values([1.0, 0.5, 0.333, 0.25, 0.2, 0.167, 0.143, 0.125, 0.111, 0.1, 0.091, 0.091, 0.1, 0.111, 0.125, 0.143, 0.167, 0.2, 0.25, 0.333, 0.5, 1.0])
```

[62]
```
#Create a frequency table to compute the frequency of each word.
freqTable = {}
for word in word_tokens_refined:
    if word in freqTable:
        freqTable[word] += 1
    else:
        freqTable[word] = 1
for k in freqTable.keys():
    freqTable[k]= math.log10(1+freqTable[k])
#Compute word frequnecy score of each sentence
word_frequency={}
for sentence in sent_tokens:
    word_frequency[sentence]=0
    e=nltk.word_tokenize(sentence)
    f=[]
    for word in e:
        f.append(ps.stem(word))
    for word,freq in freqTable.items():
        if word in f:
            word_frequency[sentence]+=freq
maximum=max(word_frequency.values())
for key in word_frequency.keys():
    try:
        word_frequency[key]=word_frequency[key]/maximum
        word_frequency[key]=round(word_frequency[key],3)
    except ZeroDivisionError:
        x=0
print(word_frequency.values())
```

```
dict_values([0.52, 0.969, 0.658, 0.415, 0.282, 0.703, 0.821, 0.571, 0.544, 0.571, 0.751, 0.836, 0.415, 0.528, 0.585, 0.765, 1.0, 0.735, 0.466, 0.708, 0.628, 0
```

[63]
```
#.....................part 6 (upper cases)..............................
upper_case={}
for sentence in sent_tokens:
    upper_case[sentence] = 0
    word_tokens = nltk.word_tokenize(sentence)
    for k in word_tokens:
        if k.isupper():
            upper_case[sentence] += 1
maximum_frequency = max(upper_case.values())
for k in upper_case.keys():
    try:
        upper_case[k] = (upper_case[k]/maximum_frequency)
        upper_case[k] = round(upper_case[k], 3)
    except ZeroDivisionError:
        x=0
print(upper_case.values())
```

```
dict_values([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0])
```

[64]
```
#..................... part7 (number of proper noun).................
proper_noun={}
```

```
    for sentence in sent_tokens:
        tagged_sent = pos_tag(sentence.split())
        propernouns = [word for word, pos in tagged_sent if pos == 'NNP']
        proper_noun[sentence]=len(propernouns)
    maximum_frequency = max(proper_noun.values())
    for k in proper_noun.keys():
        try:
            proper_noun[k] = (proper_noun[k]/maximum_frequency)
            proper_noun[k] = round(proper_noun[k], 3)
        except ZeroDivisionError:
            x=0
    print(proper_noun.values())
```

```
dict_values([0.333, 0.167, 0.333, 0.333, 0.0, 0.167, 0.333, 0.333, 0.5, 0.333, 1.0, 0.167, 0.0, 0.333, 0.667, 0.167, 0.333, 0.167, 0.333, 0.167, 1.0, 0.167])
```

[65]
```
#.............................. part 8 (word matches with heading) ..................
head_match={}
heading=sent_tokens[0]
for sentence in sent_tokens:
    head_match[sentence]=0
    word_tokens = nltk.word_tokenize(sentence)
    for k in word_tokens:
        if k not in stopWords:
            k = ps.stem(k)
            if k in ps.stem(heading):
                head_match[sentence] += 1
maximum_frequency = max(head_match.values())
for k in head_match.keys():
    try:
        head_match[k] = (head_match[k]/maximum_frequency)
        head_match[k] = round(head_match[k], 3)
    except ZeroDivisionError:
        x=0
print(head_match.values())
```

```
dict_values([1.0, 0.625, 0.375, 0.25, 0.125, 0.375, 0.25, 0.25, 0.25, 0.375, 0.125, 0.25, 0.125, 0.25, 0.125, 0.25, 0.5, 0.125, 0.125, 0.25, 0.125, 0.125])
```

```
total_score={}
for k in cue_phrases.keys():
    total_score[k]=cue_phrases[k]+numeric_data[k]+sent_len_score[k]+sentence_position[k]+word_frequency[k]+upper_case[k]+proper_noun[k]+head_match[k]
print(total_score.values())
```

```
dict_values([3.853, 3.061, 2.5490000000000004, 2.248, 1.607, 2.412, 2.197, 2.229, 2.255, 2.379, 2.567, 1.544, 2.59, 2.222, 2.152, 2.7249999999999996, 3.350000
```

[67]
```
sumValues = 0
for sentence in total_score:
    sumValues += total_score[sentence]
average = sumValues / len(total_score)
print(average)
# Storing sentences into our summary.
summary = ''
for sentence in sent_tokens:
    if (sentence in total_score) and (total_score[sentence] > (1.2*average)):
        summary += " " + sentence
print(summary)
```

```
2.6115909090909093
 Coronavirus: Belgium facing 'tsunami' of new infections. According to the Belgian health institute Sciensano, Belgium has recorded an average of 7,876 new da
```

[67]