Instructions

Download Python.py file and store it in a folder

Import given excel and txt files and store in the same folder as Python.py:

- 1. https://drive.google.com/file/d/1VStRARpVjfh218ic38XGo1pacN-vsvb5/view?usp=sharing (Name as cik_list.xlsx)
- 2. https://drive.google.com/file/d/1DrVXid5X2HZuJuw35MTpCaz17yz9RRag/view?usp=sharing (Name as constraining_dictionary.xlsx)
- 3. https://drive.google.com/file/d/19usmVV-cM0KxENwh9EzjU1qmqMCaVA_W/view?usp=sharing (Name as uncertainty_dictionary.xlsx)
- 4. https://drive.google.com/file/d/14eQ-YJDmX-vRzjY21eLR2EO-RJuZdYvO/view (Name as LoughranMcDonald_MasterDictionary_2018.xlsx)
- 5 https://drive.google.com/file/d/0B4niqV00F3mseWZrUk1YMGxpVzQ/view (Name as StopWords_Generic.txt)
- $6. \ \underline{https://drive.google.com/file/d/0B4niqV00F3msV1h6N2RhLTNBZG8/view} \ (Name\ as\ StopWords_Auditor.txt)$
- 7. https://drive.google.com/file/d/0B4niqV00F3msT18yTE42VWdLdVE/view (Name as StopWords_Currencies.txt)
- 8. https://drive.google.com/file/d/0B4niqV00F3msVGc4NldrajhQbDg/view (Name as StopWords_DatesandNumbers.txt)
- 9. https://drive.google.com/file/d/0B4niqV00F3msSktONVhfaElXeEk/view (Name as StopWords_GenericLong.txt)
- 10. https://drive.google.com/file/d/0B4niqV00F3msTXRiSmUxRmZWUFE/view (Name as StopWords_Geographic.txt)
- $11. \ \underline{https://drive.google.com/file/d/0B4niqV00F3msYlZxTm5QaEQ1dTQ/view} \ (Name\ as\ StopWords_Names.txt)$

Also install pandas, urllib, requests, urllib.request, re, nltk and nltk.data libraries into system using pip install command.

Run the Python.py file

Data is imported Make Sure that cik_list.xlsx and Python.py are in same folder import pandas as pd df = pd.read excel ('cik list.xlsx') Complete the url of SECFNAME df['SECFNAME url']='https://www.sec.gov/Archives/'+df['SECFNAME'] Import necessary ibararies import urllib, requests, urllib.request Extract URL text and store it in list_data In [4]: list=[] list data=[] for i in df['SECFNAME url']: hdr = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90 req= requests.get(i, headers=hdr) content=req.content.decode() list.append(req.content.decode()) list data.append(content) print(req) <Response [200]> Tokenize text into sentences import nltk.data import nltk nltk.download('punkt') sentence list=[] for i in list data: sentence_list.append(nltk.tokenize.sent_tokenize(i)) [nltk data] Downloading package punkt to [nltk data] C:\Users\vipin\AppData\Roaming\nltk data... [nltk_data] Package punkt is already up-to-date! Remove noise form the text lke non-alapabetical characters and tokenize into words import re tokenizedsent list=[] for i in sentence list: word list=[] for y in i: y=y.strip() processed_text = y.lower() processed_text = re.sub('[^a-zA-Z]', ' ', processed_text) processed_text = re.sub(r'\s+', ' ', processed text) processed_text = re.sub('<[^<]+>', '', processed_text) all_words = nltk.sent_tokenize(processed_text) word list+=([nltk.word tokenize(sent) for sent in all words]) tokenizedsent_list.append(word_list) Import Master dictionary and Stop words Make sure all the filles are in same folder as Python.py master = pd.read excel ('LoughranMcDonald MasterDictionary 2018.xlsx') shortgeneric=open('StopWords_Generic.txt','r').readlines() longgeneric=open('StopWords_GenericLong.txt','r').readlines() currencies=open('StopWords_Currencies.txt','r').readlines() auditor=open('StopWords_Auditor.txt','r').readlines() dataandnumbers=open('StopWords_DatesandNumbers.txt','r').readlines() geographic=open('StopWords_Geographic.txt','r').readlines() names=open('StopWords Names.txt','r').readlines() Join all the stopwords to form a stopwords list stopwords = [] for i in shortgeneric, longgeneric, currencies, auditor, dataandnumbers, geographic, names: for y in i: y =y.partition('|')[0] y=y.strip() y=y.lower() stopwords.append(y) Creating list of positive, negative and complex wors form Master **Dictionary** positive=[] for i in range(len(master['Word'])): if master['Positive'][i]!=0: positive.append(master['Word'][i]) negative=[] for i in range(len(master['Word'])): if master['Negative'][i]!=0: negative.append(master['Word'][i]) complex_words=[] for i in range(len(master['Word'])): if master['Syllables'][i]>2: complex words.append(master['Word'][i].lower()) Importing constraining_dictionary.xlsx and uncertainty_dictionary.xlsx' as lists Make Sure Both the files are in same folder as Python.py constraining=pd.read excel ('constraining dictionary.xlsx')['Word'].to list() uncertainly=pd.read_excel ('uncertainty_dictionary.xlsx')['Word'].to_list() Since positive, negative, constarining and uncertainty are mutually excluxive, a combined dictionary sentiment_dic is made sentiment_dict={} s = -1for i in negative, positive, uncertainly, constraining: for element in i: sentiment dict[element.lower()]=s A counter funtion is created that return all the words in a list with their count def counter(slist): dict={} for y in slist: if y not in dict.keys(): dict[y]=1else: dict[y] +=1return dict All the word in a document are joined with their count to form a bag of words and stores in dictsentence dictsentence=[] for i in range(len(tokenizedsent_list)): dummy=[] for y in range(len(tokenizedsent list[i])): dummy+=(tokenizedsent list[i][y]) dictsentence.append(counter(dummy)) Stopwords are removed and the processed dictionaries are stored in procesed_sent procesed sent=[] for i in dictsentence: dummy=i.copy() for y in i.keys(): if y in stopwords: **del** dummy[y] procesed sent.append(dummy) Each word in the list of dictionary is checked in our sentiment_dict and complex_words list. The count of positive, negative, constraint and uncertaint words for a document are increased accordingly score sentiment=[] for i in processed_sent: positive_count=0 negative count=0 complex_score=0 constraining_count=0 uncertainty count=0 for y in i.keys(): if y in sentiment dict.keys(): if(sentiment_dict[y] ==-1): negative_count+=i[y] elif(sentiment dict[y]==1): positive count+=i[y] elif(sentiment_dict[y]==3): uncertainty_count+=i[y] elif(sentiment_dict[y]==5): constraining_count+=i[y] if y in complex_words: complex_score+=i[y] s**+=**1 score_sentiment.append([positive_count, negative_count, constraining_count, uncertainty_count, complex_score]) print(s,[positive_count,negative_count,constraining_count,uncertainty_count,complex_score]) 1 [921, 2828, 1488, 398, 35144] 2 [536, 1396, 1047, 264, 22111] 3 [2, 8, 5, 5, 147] 4 [341, 1355, 717, 277, 17646] 5 [2, 7, 4, 6, 174] 6 [42, 387, 92, 143, 3424] 7 [116, 565, 272, 180, 5689] 8 [59, 347, 106, 167, 3613] 9 [2, 5, 3, 4, 164] 10 [379, 1518, 502, 459, 18229] 11 [2, 5, 3, 5, 167] 12 [66, 368, 83, 163, 3397] 13 [2, 5, 3, 3, 147] 14 [129, 515, 151, 243, 5250] 15 [247, 1201, 336, 550, 13433] 16 [134, 641, 169, 270, 5676] 17 [101, 625, 157, 248, 5176] 18 [3, 3, 3, 2, 133] 19 [144, 855, 195, 336, 7381] 20 [297, 1329, 523, 320, 15839] 21 [28, 62, 15, 8, 1988] 22 [160, 1120, 652, 163, 11506] 23 [58, 227, 74, 127, 4689] 24 [29, 138, 63, 74, 2775] 25 [55, 120, 67, 118, 3348] 26 [99, 253, 112, 174, 4762] 27 [181, 889, 374, 110, 12621] 28 [254, 740, 271, 383, 12676] 29 [350, 1763, 1305, 207, 21585] 30 [64, 129, 66, 102, 3483] 31 [164, 404, 167, 126, 7188] 32 [273, 702, 300, 440, 12719] 33 [102, 204, 94, 34, 4738] 34 [49, 185, 85, 74, 2750] 35 [176, 463, 277, 263, 8837] 36 [25, 54, 34, 14, 782] 37 [31, 120, 41, 49, 1909] 38 [38, 157, 41, 54, 2107] 39 [59, 269, 114, 80, 3502] 40 [368, 1212, 1075, 367, 20897] 41 [33, 110, 37, 45, 1966] 42 [41, 165, 44, 55, 2255] 43 [36, 121, 42, 50, 2121] 44 [26, 101, 32, 40, 2028] 45 [27, 101, 35, 39, 1918] 46 [459, 1122, 464, 178, 18937] 47 [19, 65, 24, 24, 1386] 48 [56, 239, 80, 31, 3778] 49 [47, 231, 55, 34, 3092] 50 [112, 404, 121, 146, 6384] 51 [2, 10, 5, 2, 165] 52 [15, 46, 27, 34, 1238] 53 [19, 136, 19, 31, 1552] 54 [116, 825, 274, 179, 13100] 55 [663, 2746, 794, 391, 51695] 56 [78, 404, 105, 116, 5864] 57 [218, 999, 379, 212, 15758] 58 [136, 614, 174, 168, 9278] 59 [5, 83, 30, 12, 1398] 60 [315, 975, 364, 341, 22412] 61 [86, 390, 94, 99, 6329] 62 [184, 4405, 283, 295, 13475] 63 [174, 4109, 248, 223, 10890] 64 [635, 3040, 678, 592, 72222] 65 [793, 3142, 650, 626, 80464] 66 [24, 81, 26, 40, 1755] 67 [311, 813, 360, 251, 11555] 68 [21, 22, 17, 48, 1306] 69 [64, 145, 58, 67, 3028] 70 [19, 26, 21, 50, 1366] 71 [179, 288, 143, 206, 7633] 72 [21, 28, 20, 37, 1145] 73 [70, 172, 79, 59, 3161] 74 [67, 189, 87, 71, 2933] 75 [2, 2, 5, 2, 116] 76 [234, 438, 186, 242, 9240] 77 [255, 438, 182, 255, 9242] 78 [564, 659, 514, 368, 19110] 79 [110, 165, 64, 96, 3826] 80 [185, 236, 97, 158, 6393] 81 [200, 304, 113, 218, 7268] 82 [248, 406, 113, 199, 6862] 83 [1126, 2074, 1043, 892, 40952] [1126, 2085, 1045, 895, 41197] 85 [201, 436, 173, 243, 8336] 86 [202, 444, 174, 241, 8400] 87 [142, 352, 101, 177, 5860] 88 [146, 434, 87, 181, 4968] 89 [14, 25, 17, 13, 896] 90 [15, 31, 21, 17, 1067] [175, 384, 107, 155, 6322] [20, 67, 27, 23, 1206] 93 [50, 175, 68, 34, 2877] 94 [33, 82, 32, 30, 1730] 95 [170, 315, 98, 154, 5309] 96 [18, 62, 26, 32, 1114] 97 [28, 187, 43, 66, 2016] 98 [30, 166, 43, 72, 1990] 99 [312, 978, 409, 295, 14461] 100 [17, 127, 37, 31, 1119] 101 [27, 140, 36, 48, 1398] 102 [28, 161, 36, 53, 1428] 103 [649, 2896, 1498, 603, 32519] 104 [26, 115, 44, 55, 1324] 105 [27, 138, 45, 92, 1734] 106 [33, 211, 77, 131, 2437] 107 [181, 468, 233, 312, 7229] 108 [17, 112, 40, 51, 1219] 109 [76, 358, 117, 82, 5021] 110 [63, 260, 89, 102, 2867] 111 [24, 132, 44, 51, 1594] 112 [29, 180, 56, 61, 2121] 113 [2, 14, 3, 3, 168] 114 [724, 2074, 802, 513, 25929] 115 [63, 287, 114, 92, 3160] 116 [106, 354, 162, 147, 4029] 117 [132, 850, 284, 194, 15809] 118 [821, 3104, 1052, 436, 64213] 119 [80, 418, 114, 132, 6641] 120 [224, 1017, 391, 236, 17746] 121 [144, 637, 183, 195, 11158] 122 [5, 82, 30, 12, 1404] 123 [357, 1818, 443, 400, 38157] 124 [87, 400, 98, 103, 7025] 125 [189, 4605, 307, 315, 14035] 126 [183, 4360, 274, 246, 11515] 127 [9, 12, 7, 11, 676] 128 [10, 12, 7, 13, 644] 129 [167, 427, 169, 128, 7041] 130 [31, 164, 57, 19, 2036] [31, 62, 27, 23, 1064] 132 [32, 108, 41, 34, 1433] 133 [170, 339, 138, 143, 6513] 134 [17, 65, 27, 39, 1190] 135 [55, 227, 75, 66, 3266] 136 [33, 152, 30, 50, 1692] 137 [79, 239, 37, 35, 2370] 138 [384, 997, 392, 126, 15783] 139 [14, 49, 9, 18, 726] 140 [22, 16, 14, 17, 992] 141 [21, 70, 12, 28, 830] 142 [59, 252, 87, 42, 3522] 143 [655, 1658, 1243, 221, 26512] 144 [15, 31, 10, 5, 502] 145 [20, 16, 16, 20, 930] 146 [32, 32, 10, 14, 629] [25, 68, 12, 15, 668] [176, 4484, 146, 135, 6069] 148 149 [2, 80, 3, 3, 135] 150 [23, 983, 25, 51, 1307] 151 [177, 4487, 145, 135, 6038] 152 [32, 1721, 39, 61, 1720] The total count of words and sentecnces for each document are calculated total_data=[] for i in range(len(procesed sent)): for y in procesed_sent[i].keys(): count+=procesed_sent[i][y] total_data.append([count,len(tokenizedsent_list[i])]) The list of respective output columns are made positive_score=[] negative_score=[] polarity_score=[] average_sentence_length=[] percentage_of_complex_words=[] fog_index=[] complex_word_count=[] word_count=[] uncertainty_score=[] constraining_score=[] positive_word_proportion=[] negative_word_proportion=[] uncertainty_word_proportion=[] constraining_word_proportion=[] constraining_words_whole_report=[] count_cwwr=0 for cik in range(len(procesed_sent)): cik_p=score_sentiment[cik][0] cik_n=score_sentiment[cik][1] cik_c=score_sentiment[cik][2] cik_u=score_sentiment[cik][3] cik_cwc=score_sentiment[cik][4] cik_tw=total_data[cik][0] cik_ts=total_data[cik][1] positive_score.append(cik_p) negative_score.append(cik_n) constraining_score.append(cik_c) count_cwwr+=cik_c uncertainty_score.append(cik_u) complex_word_count.append(cik_cwc) word_count.append(cik_tw) average_sentence_length.append(cik_tw/cik_ts) percentage_of_complex_words.append(cik_cwc/cik_tw) fog_index.append(0.4*(cik_tw/cik_ts+cik_cwc/cik_tw)) positive_word_proportion.append(cik_p/cik_tw) negative_word_proportion.append(cik_n/cik_tw) constraining_word_proportion.append(cik_c/cik_tw) uncertainty_word_proportion.append(cik_u/cik_tw) polarity_score=(cik_p-cik_n)/(cik_p+cik_n+0.000001) **if**(cik==0): constraining words whole report.append(count cwwr) constraining_words_whole_report.append('') The list are exported as Output_Score.xlsx data = {'CIK':df['CIK'].to list(), 'CONAME':df['CONAME'].to_list(), 'FYRMO':df['FYRMO'].to_list(), 'FDATE':df['FDATE'].to_list(), 'FORM':df['FORM'].to_list(), 'SECFNAME':df['SECFNAME'].to_list(), 'positive_score':positive_score, 'negative_score':negative_score, 'polarity_score':polarity_score, 'average_sentence_length':average_sentence_length, 'percentage_of_complex_words':percentage_of_complex_words, 'fog index':fog index, 'complex word count':complex word count, 'word count':word count, 'uncertainty_score':uncertainty_score, 'constraining_score':constraining score, 'positive_word_proportion':positive_word_proportion, 'negative_word_proportion':negative_word_proportion, 'uncertainty word proportion':uncertainty word proportion, 'constraining_word_proportion':constraining_word_proportion, 'constraining_words_whole_report':constraining_words_whole_report} file= pd.DataFrame(data, columns = ['CIK','CONAME','FYRMO','FDATE','FORM','SECFNAME','positive score','negative 'polarity_score','average_sentence_length','percentage_of_complex_words','i 'word_count','uncertainty_score','constraining_score','positive_word_proport 'uncertainty word proportion','constraining word proportion','constraining vfile.to excel ('Output Score.xlsx', index = False)