UE19CS332- AIW and IR

Assignment 3

Nikhil Adyapak PES2UG19CS257

Team 4, D Section

Team Members –

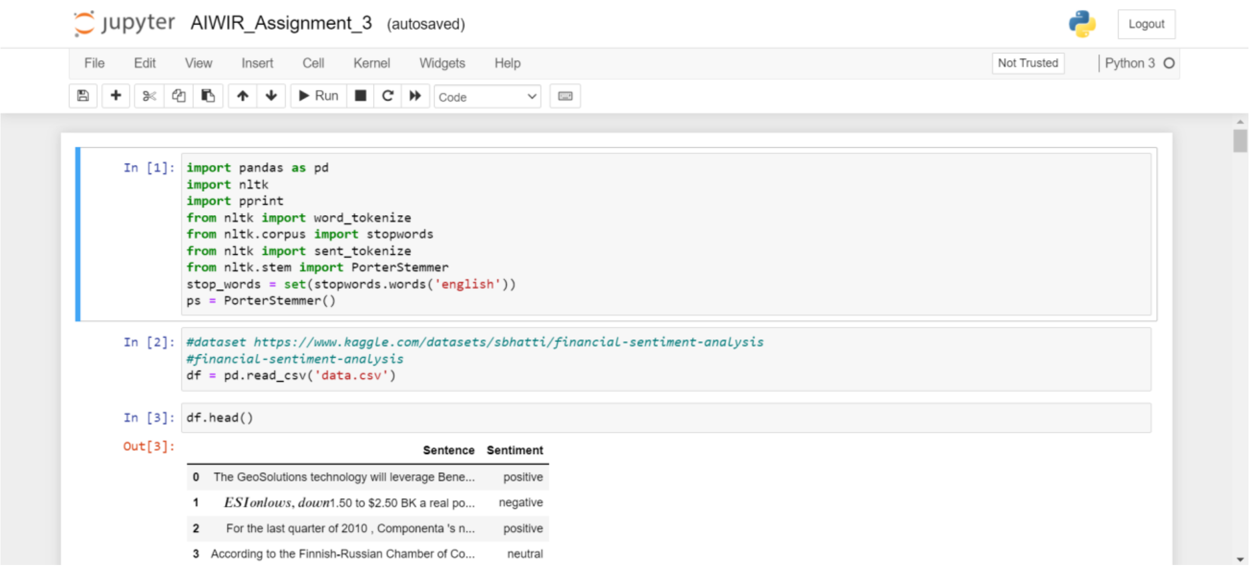
1.	Manoj Kumar	PES2UG19CS222
2.	Nidhi Gupta	PES2UG19CS256
3.	Nikhil Adyapak	PES2UG19CS257

Dataset used –

Financial Sentiment Analysis

https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis

Importing Libraries and checking the Dataset


jupyter AIWIR_Assignment_3 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

```
In [1]: import pandas as pd
        import nltk
        import pprint
        from nltk import word_tokenize
        from nltk.corpus import stopwords
        from nltk import sent_tokenize
        from nltk.stem import PorterStemmer
        stop_words = set(stopwords.words('english'))
        ps = PorterStemmer()
```

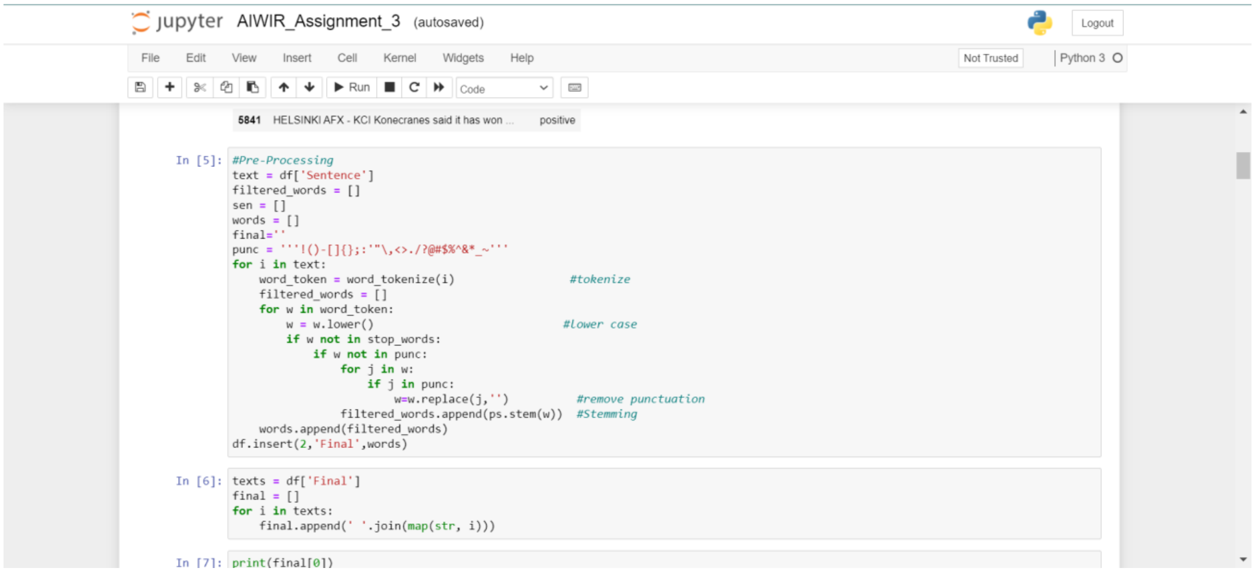
```
In [2]: #dataset https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis
        #financial-sentiment-analysis
        df = pd.read_csv('data.csv')
```

```
In [3]: df.head()
```

Out[3]:

	Sentence	Sentiment
0	The GeoSolutions technology will leverage Bene...	positive
1	*ESIonlows, down*1.50 to $2.50 BK a real po...	negative
2	For the last quarter of 2010 , Componenta 's n...	positive
3	According to the Finnish-Russian Chamber of Co...	neutral

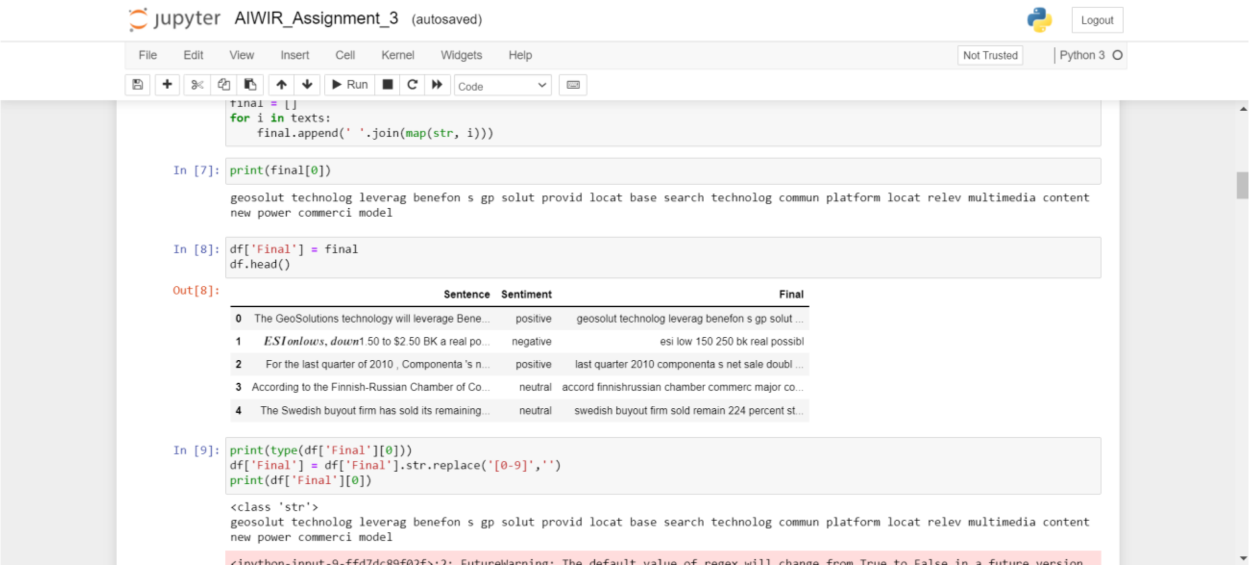
Pre-Processing the “Sentence” column in the Dataset


jupyter AIWIR_Assignment_3 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

5841	HELSINKI AFX - KCI Konecranes said it has won ...	positive

```
In [5]: #Pre-Processing
        text = df['Sentence']
        filtered_words = []
        sen = []
        words = []
        final=''
        punc = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
        for i in text:
            word_token = word_tokenize(i)                 #tokenize
            filtered_words = []
            for w in word_token:
                w = w.lower()                             #lower case
                if w not in stop_words:
                    if w not in punc:
                        for j in w:
                            if j in punc:
                                w=w.replace(j,'')          #remove punctuation
                        filtered_words.append(ps.stem(w))  #Stemming
            words.append(filtered_words)
        df.insert(2,'Final',words)

In [6]: texts = df['Final']
        final = []
        for i in texts:
            final.append(' '.join(map(str, i)))

In [7]: print(final[0])
```

The column “Final” contains the Pre-Processed sentences.


jupyter AIWIR_Assignment_3 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

```
        final = []
        for i in texts:
            final.append(' '.join(map(str, i)))

In [7]: print(final[0])

        geosolut technolog leverag benefon s gp solut provid locat base search technolog commun platform locat relev multimedia content
        new power commerci model

In [8]: df['Final'] = final
        df.head()

Out[8]:
```

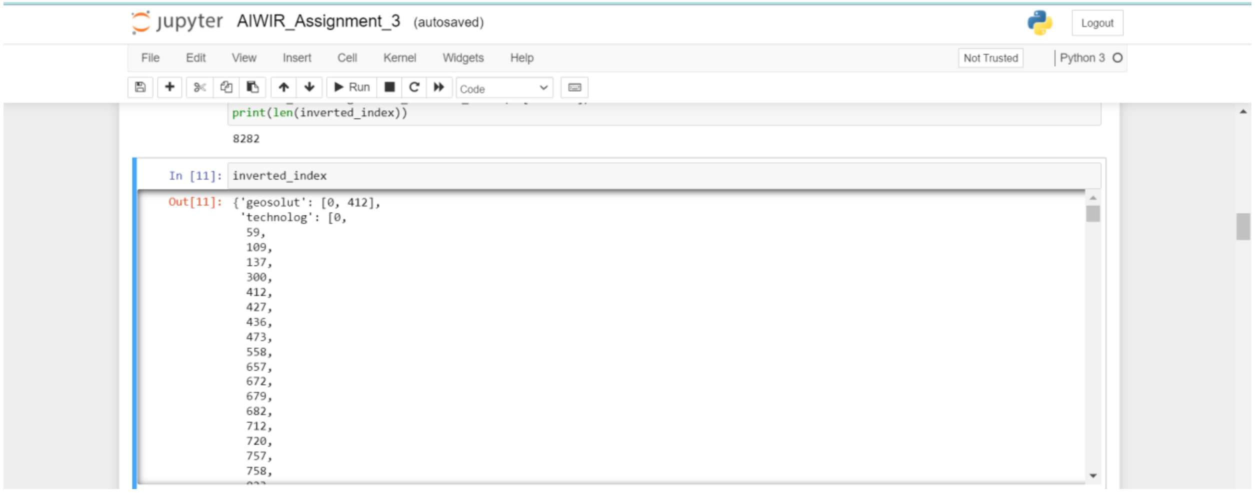
	Sentence	Sentiment	Final
0	The GeoSolutions technology will leverage Bene...	positive	geosolut technolog leverag benefon s gp solut ...
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative	esi low 150 250 bk real possibl
2	For the last quarter of 2010 , Componenta 's n...	positive	last quarter 2010 componenta s net sale doubl ...
3	According to the Finnish-Russian Chamber of Co...	neutral	accord finnishrussian chamber commerc major co...
4	The Swedish buyout firm has sold its remaining...	neutral	swedish buyout firm sold remain 224 percent st...

```
In [9]: print(type(df['Final'][0]))
        df['Final'] = df['Final'].str.replace('[0-9]','')
        print(df['Final'][0])

        <class 'str'>
        geosolut technolog leverag benefon s gp solut provid locat base search technolog commun platform locat relev multimedia content
        new power commerci model

        <ipython-input-9-ffd7dc89f02f>:2: FutureWarning: The default value of regex will change from True to False in a future version
```

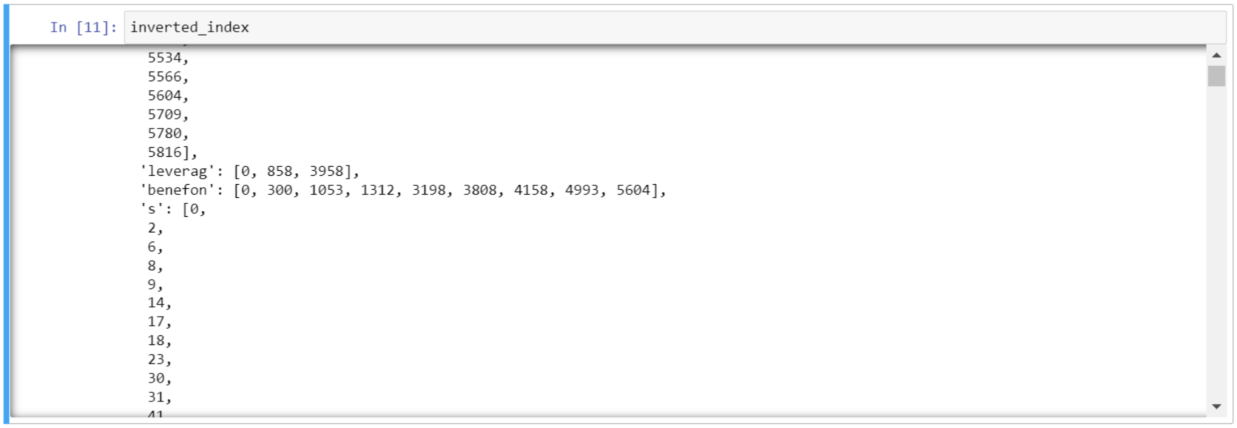
Generating the Inverted Index

```
In [10]: #inverted index
         def generate_inverted_index(data: list):
             inv_idx_dict = {}

             for index, doc_text in enumerate(data):
                 for word in doc_text.split():
                     if word not in inv_idx_dict.keys():
                         inv_idx_dict[word] = [index]
                     elif index not in inv_idx_dict[word]:
                         inv_idx_dict[word].append(index)
             return inv_idx_dict
         inverted_index = generate_inverted_index(df['Final'])
         print(len(inverted_index))

         8282
```

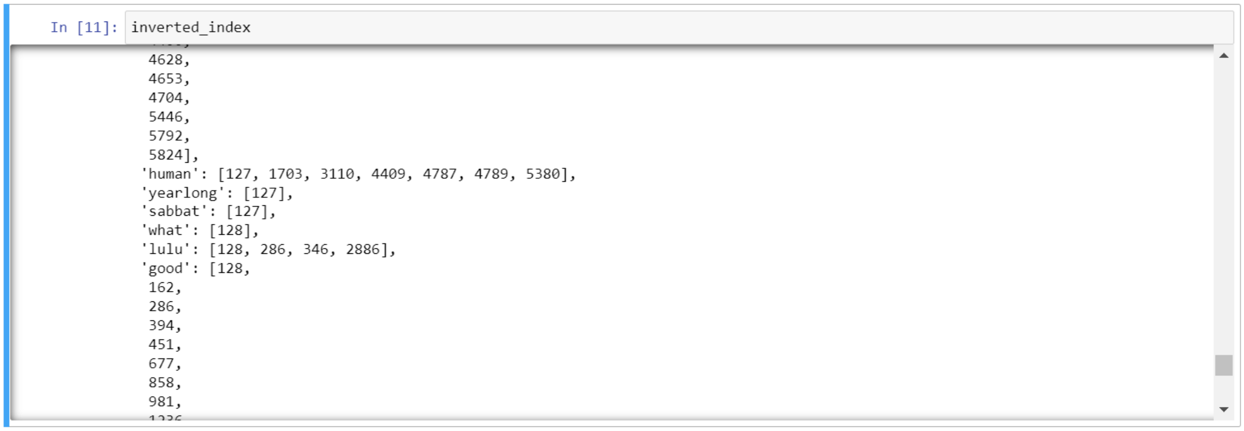
Inverted Index



```
print(len(inverted_index))

8282

In [11]: inverted_index

Out[11]: {'geosolut': [0, 412],
          'technolog': [0,
           59,
           109,
           137,
           300,
           412,
           427,
           436,
           473,
           558,
           657,
           672,
           679,
           682,
           712,
           720,
           757,
           758,
```



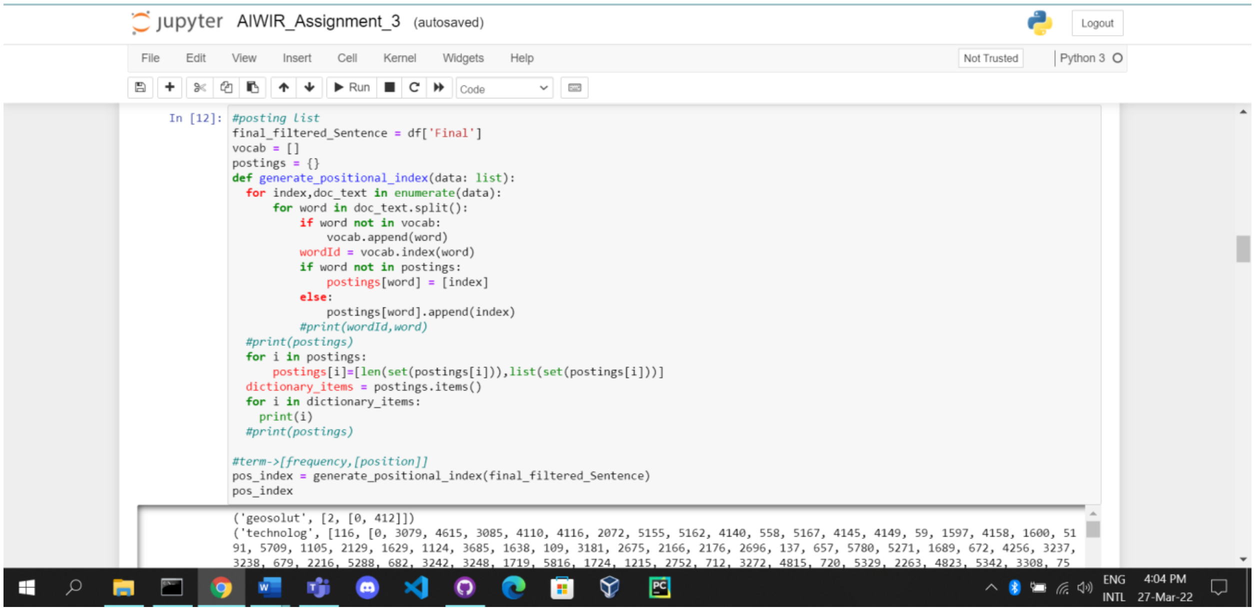
```
In [11]: inverted_index

           5534,
           5566,
           5604,
           5709,
           5780,
           5816],
          'leverag': [0, 858, 3958],
          'benefon': [0, 300, 1053, 1312, 3198, 3808, 4158, 4993, 5604],
          's': [0,
           2,
           6,
           8,
           9,
           14,
           17,
           18,
           23,
           30,
           31,
```



```
In [11]: inverted_index
```

```
        4628,
        4653,
        4704,
        5446,
        5792,
        5824],
       'human': [127, 1703, 3110, 4409, 4787, 4789, 5380],
       'yearlong': [127],
       'sabbat': [127],
       'what': [128],
       'lulu': [128, 286, 346, 2886],
       'good': [128,
        162,
        286,
        394,
        451,
        677,
        858,
        981,
```

Generating the Posting List

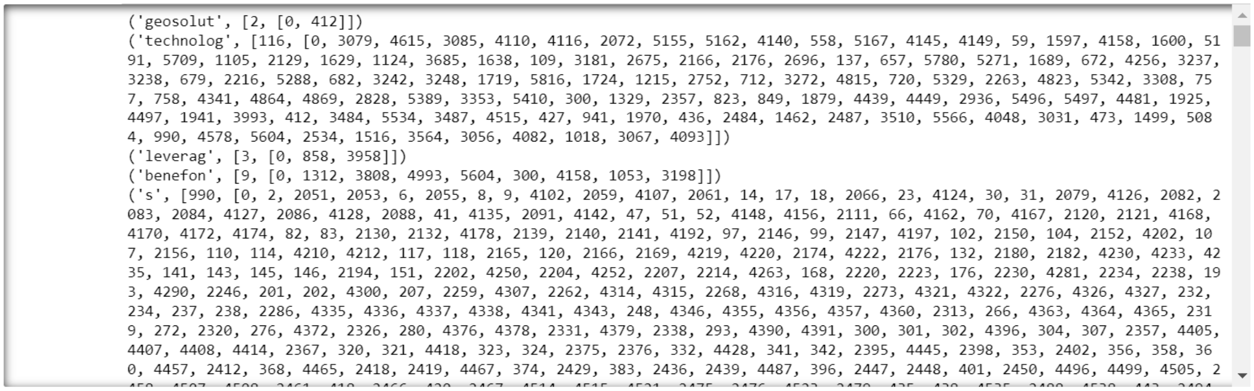


jupyter AIWIR_Assignment_3 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code
ENG
INTL
4:04 PM
27-Mar-22

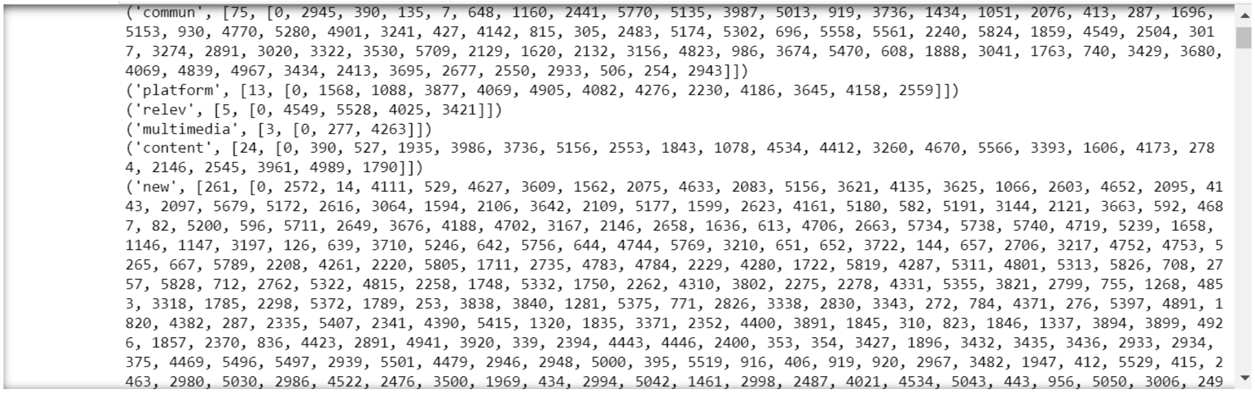
```
In [12]: #posting list
         final_filtered_Sentence = df['Final']
         vocab = []
         postings = {}
         def generate_positional_index(data: list):
           for index,doc_text in enumerate(data):
               for word in doc_text.split():
                   if word not in vocab:
                       vocab.append(word)
                   wordId = vocab.index(word)
                   if word not in postings:
                       postings[word] = [index]
                   else:
                       postings[word].append(index)
                   #print(wordId,word)
           #print(postings)
           for i in postings:
               postings[i]=[len(set(postings[i])),list(set(postings[i]))]
           dictionary_items = postings.items()
           for i in dictionary_items:
             print(i)
           #print(postings)

         #term->[frequency,[position]]
         pos_index = generate_positional_index(final_filtered_Sentence)
         pos_index
```

```
('geosolut', [2, [0, 412]])
('technolog', [116, [0, 3079, 4615, 3085, 4110, 4116, 2072, 5155, 5162, 4140, 558, 5167, 4145, 4149, 59, 1597, 4158, 1600, 51
91, 5709, 1105, 2129, 1629, 1124, 3685, 1638, 109, 3181, 2675, 2166, 2176, 2696, 137, 657, 5780, 5271, 1689, 672, 4256, 3237,
```

Posting List



```
('geosolut', [2, [0, 412]])
('technolog', [116, [0, 3079, 4615, 3085, 4110, 4116, 2072, 5155, 5162, 4140, 558, 5167, 4145, 4149, 59, 1597, 4158, 1600, 51
91, 5709, 1105, 2129, 1629, 1124, 3685, 1638, 109, 3181, 2675, 2166, 2176, 2696, 137, 657, 5780, 5271, 1689, 672, 4256, 3237,
3238, 679, 2216, 5288, 682, 3242, 3248, 1719, 5816, 1724, 1215, 2752, 712, 3272, 4815, 720, 5329, 2263, 4823, 5342, 3308, 75
7, 758, 4341, 4864, 4869, 2828, 5389, 3353, 5410, 300, 1329, 2357, 823, 849, 1879, 4439, 4449, 2936, 5496, 5497, 4481, 1925,
4497, 1941, 3993, 412, 3484, 5534, 3487, 4515, 427, 941, 1970, 436, 2484, 1462, 2487, 3510, 5566, 4048, 3031, 473, 1499, 508
4, 990, 4578, 5604, 2534, 1516, 3564, 3056, 4082, 1018, 3067, 4093]])
('leverag', [3, [0, 858, 3958]])
('benefon', [9, [0, 1312, 3808, 4993, 5604, 300, 4158, 1053, 3198]])
('s', [990, [0, 2, 2051, 2053, 6, 2055, 8, 9, 4102, 2059, 4107, 2061, 14, 17, 18, 2066, 23, 4124, 30, 31, 2079, 4126, 2082, 2
083, 2084, 4127, 2086, 4128, 2088, 41, 4135, 2091, 4142, 47, 51, 52, 4148, 4156, 2111, 66, 4162, 70, 4167, 2120, 2121, 4168,
4170, 4172, 4174, 82, 83, 2130, 2132, 4178, 2139, 2140, 2141, 4192, 97, 2146, 99, 2147, 4197, 102, 2150, 104, 2152, 4202, 10
7, 2156, 110, 114, 4210, 4212, 117, 118, 2165, 120, 2166, 2169, 4219, 4220, 2174, 4222, 2176, 132, 2180, 2182, 4230, 4233, 42
35, 141, 143, 145, 146, 2194, 151, 2202, 4250, 2204, 4252, 2207, 2214, 4263, 168, 2220, 2223, 176, 2230, 4281, 2234, 2238, 19
3, 4290, 2246, 201, 202, 4300, 207, 2259, 4307, 2262, 4314, 4315, 2268, 4316, 4319, 2273, 4321, 4322, 2276, 4326, 4327, 232,
234, 237, 238, 2286, 4335, 4336, 4337, 4338, 4341, 4343, 248, 4346, 4355, 4356, 4357, 4360, 2313, 266, 4363, 4364, 4365, 231
9, 272, 2320, 276, 4372, 2326, 280, 4376, 4378, 2331, 4379, 2338, 293, 4390, 4391, 300, 301, 302, 4396, 304, 307, 2357, 4405,
4407, 4408, 4414, 2367, 320, 321, 4418, 323, 324, 2375, 2376, 332, 4428, 341, 342, 2395, 4445, 2398, 353, 2402, 356, 358, 36
0, 4457, 2412, 368, 4465, 2418, 2419, 4467, 374, 2429, 383, 2436, 2439, 4487, 396, 2447, 2448, 401, 2450, 4496, 4499, 4505, 2
```



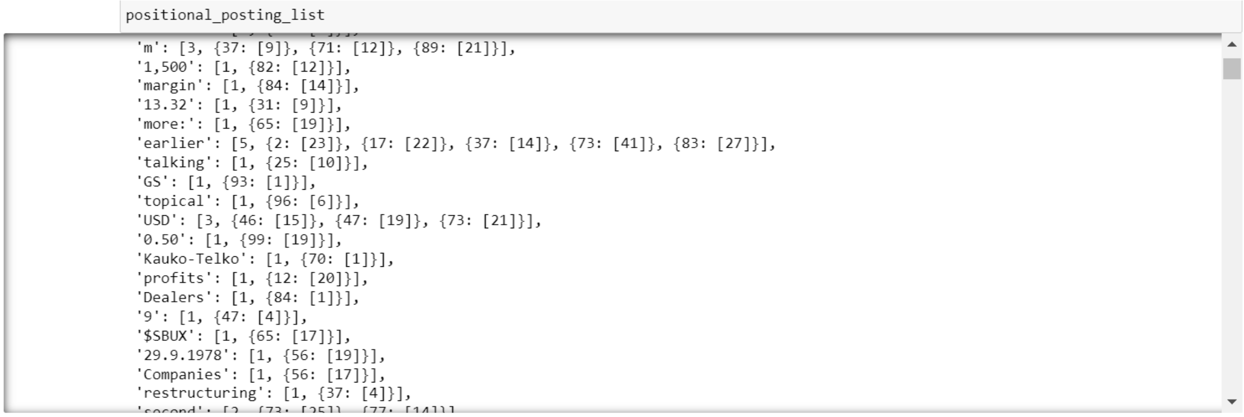
```
('commun', [75, [0, 2945, 390, 135, 7, 648, 1160, 2441, 5770, 5135, 3987, 5013, 919, 3736, 1434, 1051, 2076, 413, 287, 1696,
5153, 930, 4770, 5280, 4901, 3241, 427, 4142, 815, 305, 2483, 5174, 5302, 696, 5558, 5561, 2240, 5824, 1859, 4549, 2504, 301
7, 3274, 2891, 3020, 3322, 3530, 5709, 2129, 1620, 2132, 3156, 4823, 986, 3674, 5470, 608, 1888, 3041, 1763, 740, 3429, 3680,
4069, 4839, 4967, 3434, 2413, 3695, 2677, 2550, 2933, 506, 254, 2943]])
('platform', [13, [0, 1568, 1088, 3877, 4069, 4905, 4082, 4276, 2230, 4186, 3645, 4158, 2559]])
('relev', [5, [0, 4549, 5528, 4025, 3421]])
('multimedia', [3, [0, 277, 4263]])
('content', [24, [0, 390, 527, 1935, 3986, 3736, 5156, 2553, 1843, 1078, 4534, 4412, 3260, 4670, 5566, 3393, 1606, 4173, 278
4, 2146, 2545, 3961, 4989, 1790]])
('new', [261, [0, 2572, 14, 4111, 529, 4627, 3609, 1562, 2075, 4633, 2083, 5156, 3621, 4135, 3625, 1066, 2603, 4652, 2095, 41
43, 2097, 5679, 5172, 2616, 3064, 1594, 2106, 3642, 2109, 5177, 1599, 2623, 4161, 5180, 582, 5191, 3144, 2121, 3663, 592, 468
7, 82, 5200, 596, 5711, 2649, 3676, 4188, 4702, 3167, 2146, 2658, 1636, 613, 4706, 2663, 5734, 5738, 5740, 4719, 5239, 1658,
1146, 1147, 3197, 126, 639, 3710, 5246, 642, 5756, 644, 4744, 5769, 3210, 651, 652, 3722, 144, 657, 2706, 3217, 4752, 4753, 5
265, 667, 5789, 2208, 4261, 2220, 5805, 1711, 2735, 4783, 4784, 2229, 4280, 1722, 5819, 4287, 5311, 4801, 5313, 5826, 708, 27
57, 5828, 712, 2762, 5322, 4815, 2258, 1748, 5332, 1750, 2262, 4310, 3802, 2275, 2278, 4331, 5355, 3821, 2799, 755, 1268, 485
3, 3318, 1785, 2298, 5372, 1789, 253, 3838, 3840, 1281, 5375, 771, 2826, 3338, 2830, 3343, 272, 784, 4371, 276, 5397, 4891, 1
820, 4382, 287, 2335, 5407, 2341, 4390, 5415, 1320, 1835, 3371, 2352, 4400, 3891, 1845, 310, 823, 1846, 1337, 3894, 3899, 492
6, 1857, 2370, 836, 4423, 2891, 4941, 3920, 339, 2394, 4443, 4446, 2400, 353, 354, 3427, 1896, 3432, 3435, 3436, 2933, 2934,
375, 4469, 5496, 5497, 2939, 5501, 4479, 2946, 2948, 5000, 395, 5519, 916, 406, 919, 920, 2967, 3482, 1947, 412, 5529, 415, 2
463, 2980, 5030, 2986, 4522, 2476, 3500, 1969, 434, 2994, 5042, 1461, 2998, 2487, 4021, 4534, 5043, 443, 956, 5050, 3006, 249
```

Generating the Positional Index

In [17]:

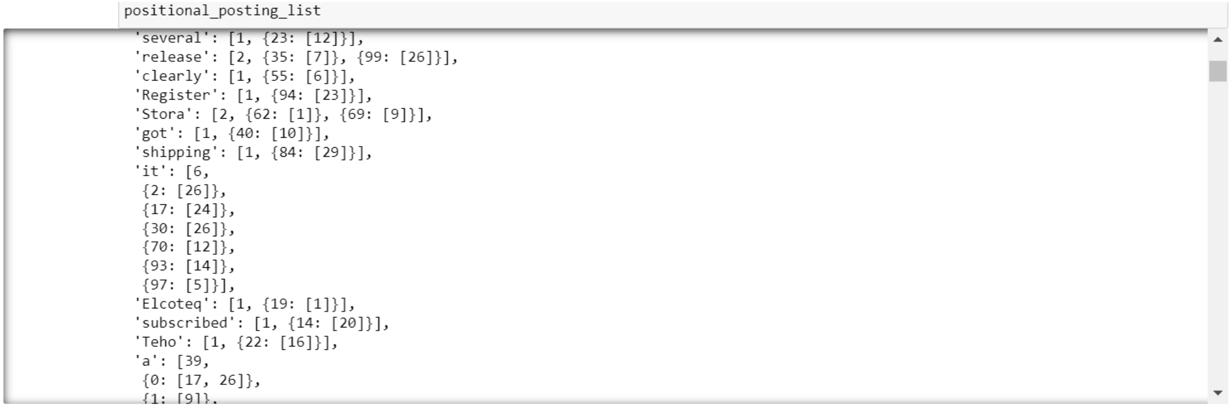
```python
text = df['Sentence'].head(100)
distinct_words = []
#Positional Posting List
for words in text:
    distinct_words.extend(words.split(' '))

tokens = list(set(distinct_words))
len(tokens)

#Positional Index
positional_posting_list = {}
for word in tokens:
    lst = []
    positional_posting_list[word] = [distinct_words.count(word)]
    for i in range(0, len(text)):
        j = text[i].split()
        dic = {}
        dic[i] = []
        if (word in j):
            list_words = j
            dic[i].extend([word_pos+1 for word_pos in range(len(list_words)) if list_words[word_pos]==word ])
            lst.append(dic)
    positional_posting_list[word].extend(lst)

positional_posting_list
```

Positional Index

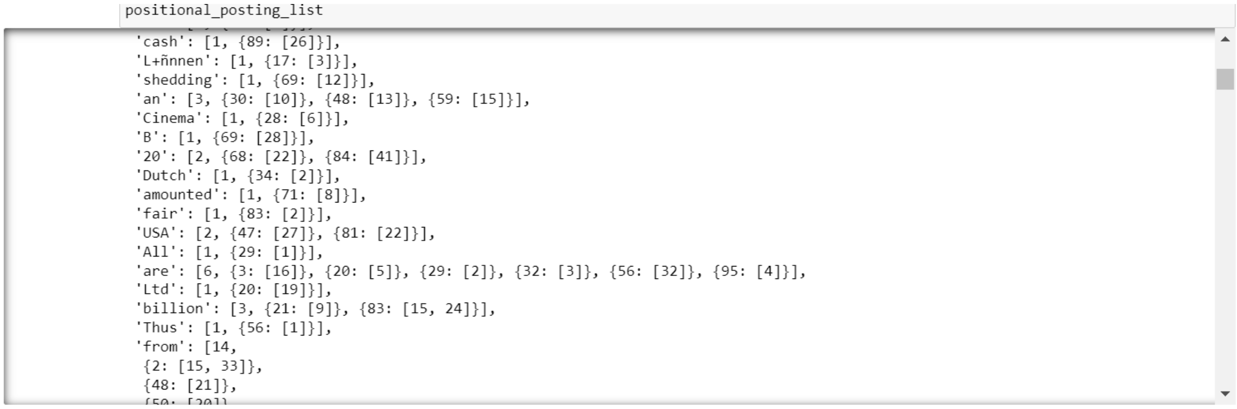


```
positional_posting_list
```

```
 'm': [3, {37: [9]}, {71: [12]}, {89: [21]}],
 '1,500': [1, {82: [12]}],
 'margin': [1, {84: [14]}],
 '13.32': [1, {31: [9]}],
 'more:': [1, {65: [19]}],
 'earlier': [5, {2: [23]}, {17: [22]}, {37: [14]}, {73: [41]}, {83: [27]}],
 'talking': [1, {25: [10]}],
 'GS': [1, {93: [1]}],
 'topical': [1, {96: [6]}],
 'USD': [3, {46: [15]}, {47: [19]}, {73: [21]}],
 '0.50': [1, {99: [19]}],
 'Kauko-Telko': [1, {70: [1]}],
 'profits': [1, {12: [20]}],
 'Dealers': [1, {84: [1]}],
 '9': [1, {47: [4]}],
 '$SBUX': [1, {65: [17]}],
 '29.9.1978': [1, {56: [19]}],
 'Companies': [1, {56: [17]}],
 'restructuring': [1, {37: [4]}],
 'second': [2, {73: [25]}, {77: [14]}],
```



```
positional_posting_list
 'several': [1, {23: [12]}],
 'release': [2, {35: [7]}, {99: [26]}],
 'clearly': [1, {55: [6]}],
 'Register': [1, {94: [23]}],
 'Stora': [2, {62: [1]}, {69: [9]}],
 'got': [1, {40: [10]}],
 'shipping': [1, {84: [29]}],
 'it': [6,
  {2: [26]},
  {17: [24]},
  {30: [26]},
  {70: [12]},
  {93: [14]},
  {97: [5]}],
 'Elcoteq': [1, {19: [1]}],
 'subscribed': [1, {14: [20]}],
 'Teho': [1, {22: [16]}],
 'a': [39,
  {0: [17, 26]},
  {1: [9]},
```



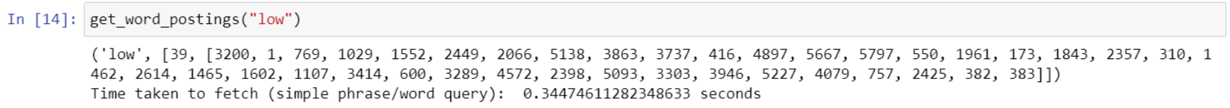
```
positional_posting_list
 'cash': [1, {89: [26]}],
 'L+ñnnen': [1, {17: [3]}],
 'shedding': [1, {69: [12]}],
 'an': [3, {30: [10]}, {48: [13]}, {59: [15]}],
 'Cinema': [1, {28: [6]}],
 'B': [1, {69: [28]}],
 '20': [2, {68: [22]}, {84: [41]}],
 'Dutch': [1, {34: [2]}],
 'amounted': [1, {71: [8]}],
 'fair': [1, {83: [2]}],
 'USA': [2, {47: [27]}, {81: [22]}],
 'All': [1, {29: [1]}],
 'are': [6, {3: [16]}, {20: [5]}, {29: [2]}, {32: [3]}, {56: [32]}, {95: [4]}],
 'Ltd': [1, {20: [19]}],
 'billion': [3, {21: [9]}, {83: [15, 24]}],
 'Thus': [1, {56: [1]}],
 'from': [14,
  {2: [15, 33]},
  {48: [21]},
  {50: [20]}
```

Simple phrase/word query

In [13]:

```
#simple phrase/word query
import time
def get_word_postings(word):
    flag = False
    start=time.time()
    dictionary_items = postings.items()
    for i in dictionary_items:
        if(i[0] == word):
            flag = True
            print(i)
            break
        else:
            time.sleep(0.0000000001)
            continue
    end=time.time()
    time_taken=end-start       #Time
    if flag:
        print("Time taken to fetch (simple phrase/word query): ",time_taken,"seconds")
    else:
        print("Could not find the word:",word)
```

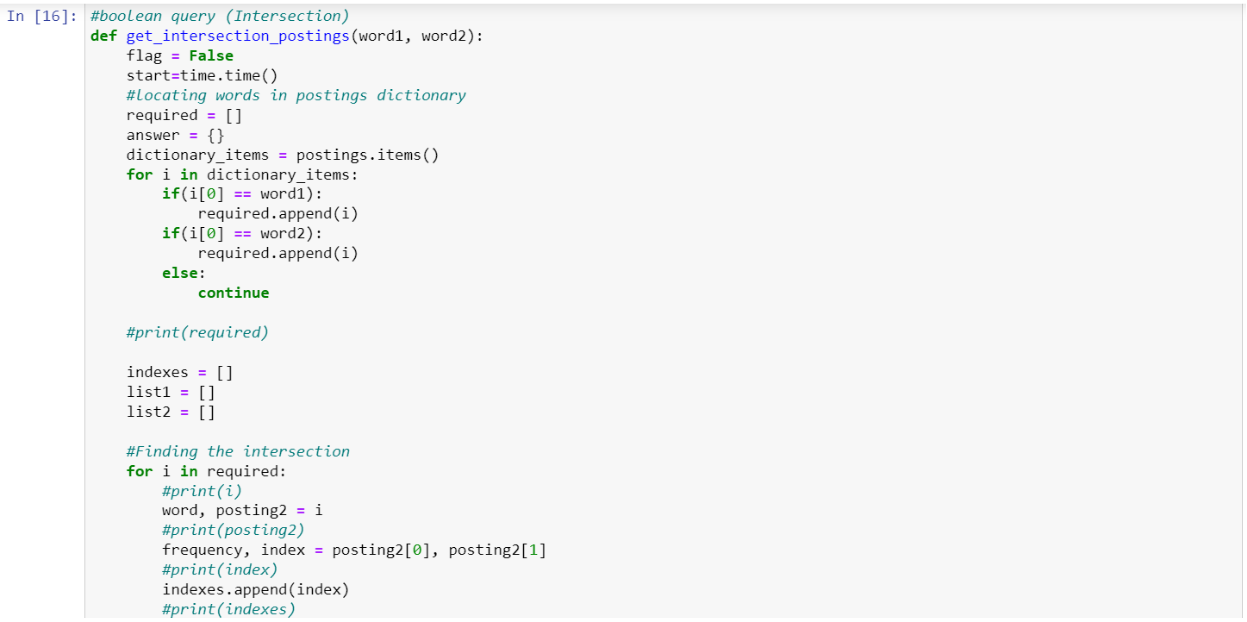
An Example by searching the word “low”



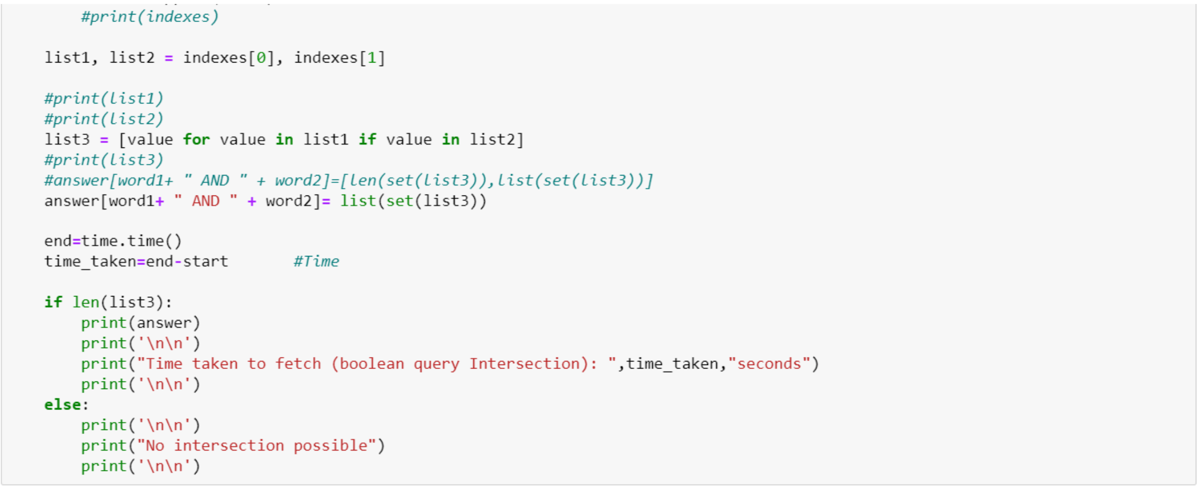
```
In [14]: get_word_postings("low")

         ('low', [39, [3200, 1, 769, 1029, 1552, 2449, 2066, 5138, 3863, 3737, 416, 4897, 5667, 5797, 550, 1961, 173, 1843, 2357, 310, 1
         462, 2614, 1465, 1602, 1107, 3414, 600, 3289, 4572, 2398, 5093, 3303, 3946, 5227, 4079, 757, 2425, 382, 383]])
         Time taken to fetch (simple phrase/word query):  0.34474611282348633 seconds
```

The posting list as well as the Query Response Time is displayed as seen above.

Boolean Query for Intersection



```
In [16]: #boolean query (Intersection)
         def get_intersection_postings(word1, word2):
             flag = False
             start=time.time()
             #locating words in postings dictionary
             required = []
             answer = {}
             dictionary_items = postings.items()
             for i in dictionary_items:
                 if(i[0] == word1):
                     required.append(i)
                 if(i[0] == word2):
                     required.append(i)
                 else:
                     continue

             #print(required)

             indexes = []
             list1 = []
             list2 = []

             #Finding the intersection
             for i in required:
                 #print(i)
                 word, posting2 = i
                 #print(posting2)
                 frequency, index = posting2[0], posting2[1]
                 #print(index)
                 indexes.append(index)
                 #print(indexes)
```

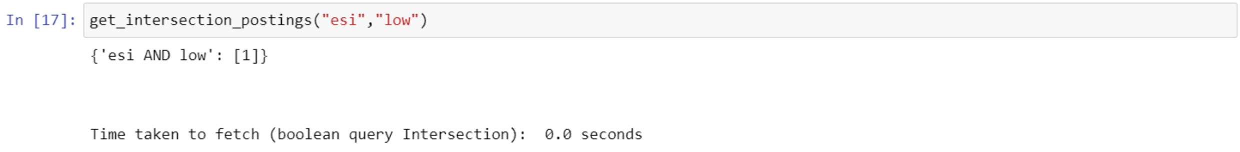


```
        #print(indexes)

    list1, list2 = indexes[0], indexes[1]

    #print(list1)
    #print(list2)
    list3 = [value for value in list1 if value in list2]
    #print(list3)
    #answer[word1+ " AND " + word2]=[len(set(list3)),list(set(list3))]
    answer[word1+ " AND " + word2]= list(set(list3))

    end=time.time()
    time_taken=end-start       #Time

    if len(list3):
        print(answer)
        print('\n\n')
        print("Time taken to fetch (boolean query Intersection): ",time_taken,"seconds")
        print('\n\n')
    else:
        print('\n\n')
        print("No intersection possible")
        print('\n\n')
```

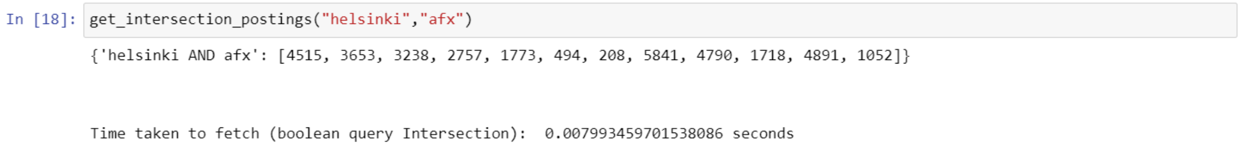
Examples for Boolean Query (Intersection) and Query Response Time

1.esi AND low

2. Helsinki and afx

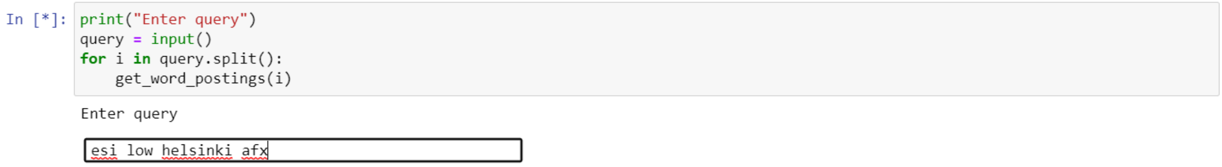


```
In [17]: get_intersection_postings("esi","low")

         {'esi AND low': [1]}


         Time taken to fetch (boolean query Intersection):  0.0 seconds
```



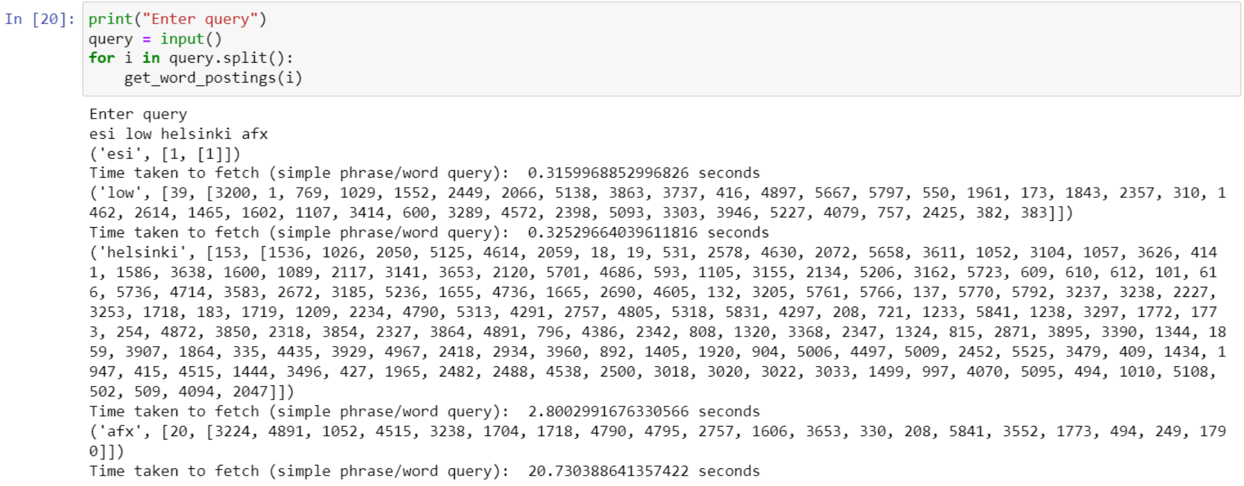
```
In [18]: get_intersection_postings("helsinki","afx")

         {'helsinki AND afx': [4515, 3653, 3238, 2757, 1773, 494, 208, 5841, 4790, 1718, 4891, 1052]}


         Time taken to fetch (boolean query Intersection):  0.007993459701538086 seconds
```

Search by user Entering a Query String "esi low Helsinki afx"

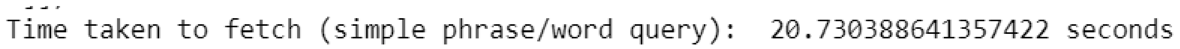


```
In [*]: print("Enter query")
        query = input()
        for i in query.split():
            get_word_postings(i)

        Enter query

        esi low helsinki afx
```

Output for the Simple search of Query String "esi low Helsinki afx" showing the posting list as well as the Query Response Time for each word and overall time at the end.



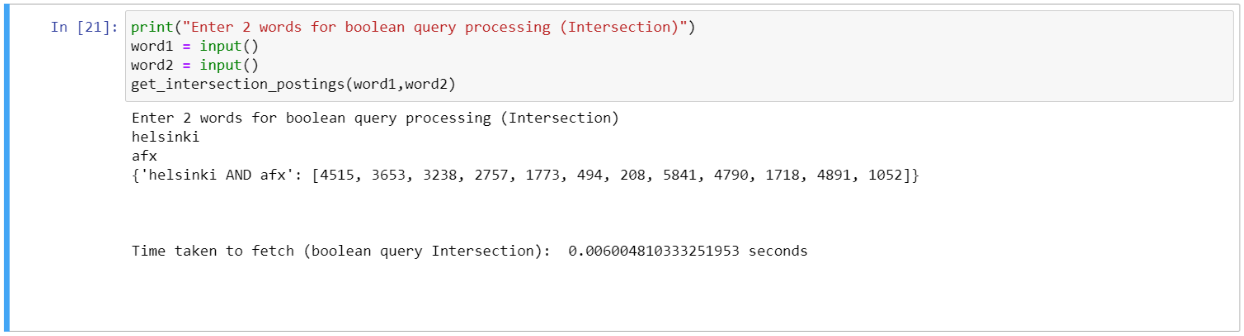
```
In [20]: print("Enter query")
         query = input()
         for i in query.split():
             get_word_postings(i)

         Enter query
         esi low helsinki afx
         ('esi', [1, [1]])
         Time taken to fetch (simple phrase/word query):  0.3159968852996826 seconds
         ('low', [39, [3200, 1, 769, 1029, 1552, 2449, 2066, 5138, 3863, 3737, 416, 4897, 5667, 5797, 550, 1961, 173, 1843, 2357, 310, 1
         462, 2614, 1465, 1602, 1107, 3414, 600, 3289, 4572, 2398, 5093, 3303, 3946, 5227, 4079, 757, 2425, 382, 383]])
         Time taken to fetch (simple phrase/word query):  0.32529664039611816 seconds
         ('helsinki', [153, [1536, 1026, 2050, 5125, 4614, 2059, 18, 19, 531, 2578, 4630, 2072, 5658, 3611, 1052, 3104, 1057, 3626, 414
         1, 1586, 3638, 1600, 1089, 2117, 3141, 3653, 2120, 5701, 4686, 593, 1105, 3155, 2134, 5206, 3162, 5723, 609, 610, 612, 101, 61
         6, 5736, 4714, 3583, 2672, 3185, 5236, 1655, 4736, 1665, 2690, 4605, 132, 3205, 5761, 5766, 137, 5770, 5792, 3237, 3238, 2227,
         3253, 1718, 183, 1719, 1209, 2234, 4790, 5313, 4291, 2757, 4805, 5318, 5831, 4297, 208, 721, 1233, 5841, 1238, 3297, 1772, 177
         3, 254, 4872, 3850, 2318, 3854, 2327, 3864, 4891, 796, 4386, 2342, 808, 1320, 3368, 2347, 1324, 815, 2871, 3895, 3390, 1344, 18
         59, 3907, 1864, 335, 4435, 3929, 4967, 2418, 2934, 3960, 892, 1405, 1920, 904, 5006, 4497, 5009, 2452, 5525, 3479, 409, 1434, 1
         947, 415, 4515, 1444, 3496, 427, 1965, 2482, 2488, 4538, 2500, 3018, 3020, 3022, 3033, 1499, 997, 4070, 5095, 494, 1010, 5108,
         502, 509, 4094, 2047]])
         Time taken to fetch (simple phrase/word query):  2.8002991676330566 seconds
         ('afx', [20, [3224, 4891, 1052, 4515, 3238, 1704, 1718, 4790, 4795, 2757, 1606, 3653, 330, 208, 5841, 3552, 1773, 494, 249, 179
         0]])
         Time taken to fetch (simple phrase/word query):  20.730388641357422 seconds
```



```
Time taken to fetch (simple phrase/word query):  20.730388641357422 seconds
```

The query response time has been made to have a time delay of 1 nanosecond between each iteration.

Taking User inputs, performing Boolean Query Search (Intersection)



```
In [21]: print("Enter 2 words for boolean query processing (Intersection)")
         word1 = input()
         word2 = input()
         get_intersection_postings(word1,word2)

         Enter 2 words for boolean query processing (Intersection)
         helsinki
         afx
         {'helsinki AND afx': [4515, 3653, 3238, 2757, 1773, 494, 208, 5841, 4790, 1718, 4891, 1052]}



         Time taken to fetch (boolean query Intersection):  0.006004810333251953 seconds
```

Boolean Query for Union

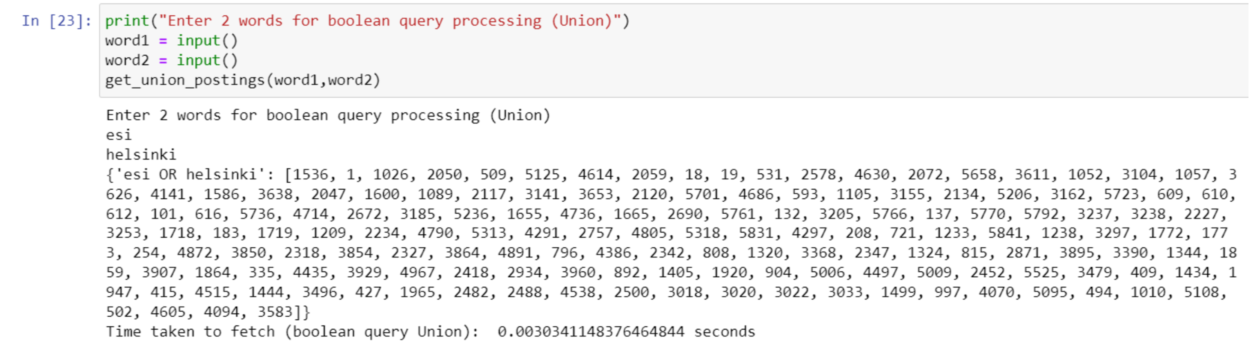
In [22]:
```
#boolean query (Union)
def get_union_postings(word1, word2):
    flag = False
    start=time.time()
    #Locating words in postings dictionary
    required = []
    answer = {}
    dictionary_items = postings.items()
    for i in dictionary_items:
      if(i[0] == word1):
          required.append(i)
      if(i[0] == word2):
          required.append(i)
      else:
        continue
    #print(required)

    indexes = []
    list1 = []
    list2 = []
    #Finding the union
    for i in required:
      #print(i)
      word, posting2 = i
      #print(posting2)
```

```
      #print(posting2)
      frequency, index = posting2[0], posting2[1]
      #print(index)
      indexes.append(index)
    #print(indexes)

    list1, list2 = indexes[0], indexes[1]

    #print(list1)
    #print(list2)
    list3 = list1 + list2
    #print(list3)
    #answer[word1+ " OR " + word2]=[len(set(list3)),list(set(list3))]
    answer[word1+ " OR " + word2]= list(set(list3))
    end=time.time()
    time_taken=end-start       #Time
    if len(list3):
      print(answer)
      print("Time taken to fetch (boolean query Union): ",time_taken,"seconds")
    else:
      print("No Union possible")
```

Taking User inputs, performing Boolean Query Search (Union)

1. Esi OR Helsinki

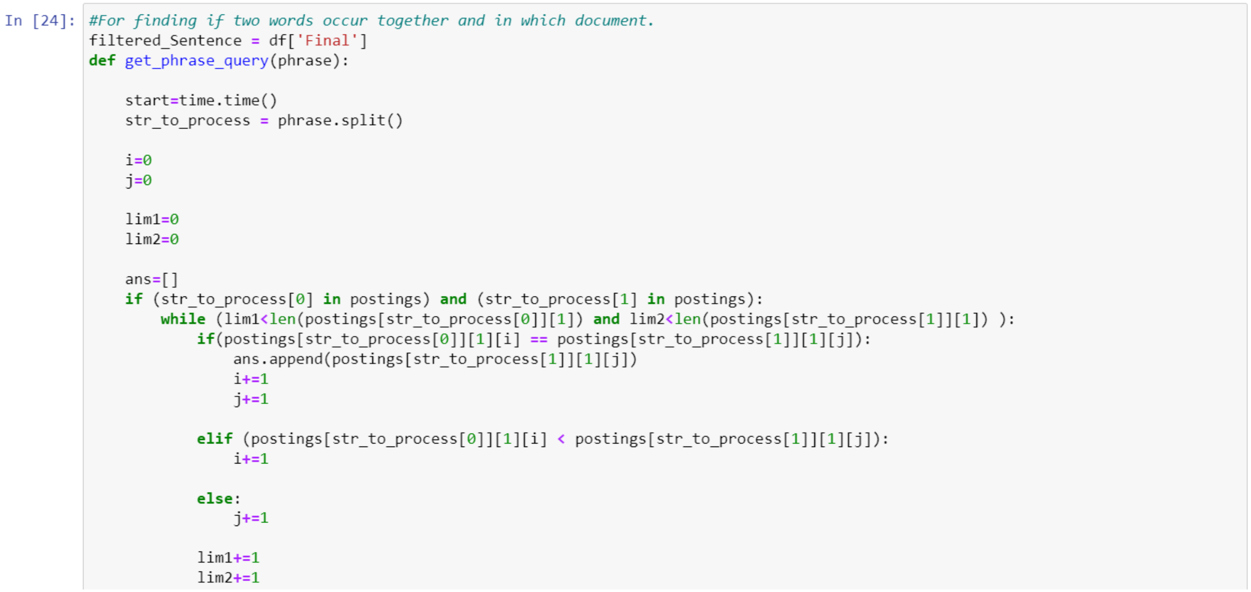
and returning the union and the Query Response Time



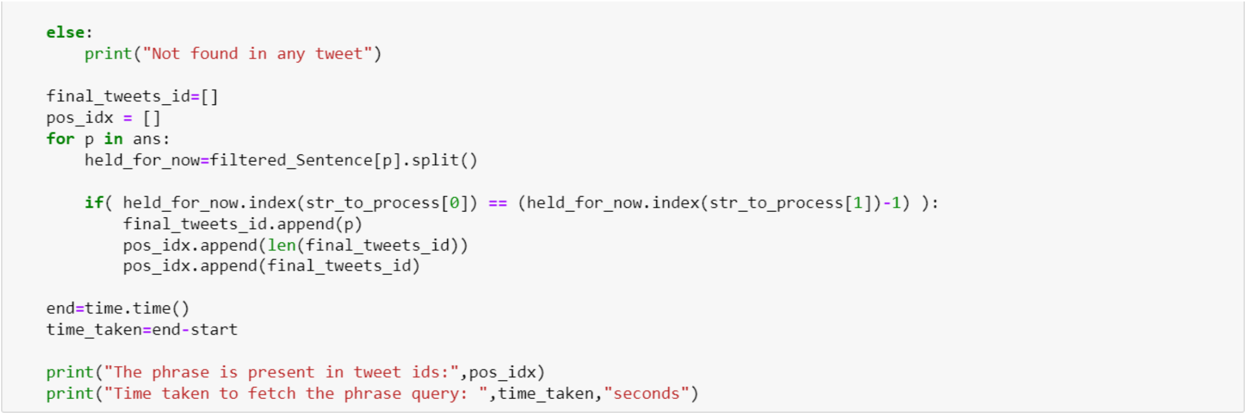
```
In [23]: print("Enter 2 words for boolean query processing (Union)")
         word1 = input()
         word2 = input()
         get_union_postings(word1,word2)
```

```
Enter 2 words for boolean query processing (Union)
esi
helsinki
{'esi OR helsinki': [1536, 1, 1026, 2050, 509, 5125, 4614, 2059, 18, 19, 531, 2578, 4630, 2072, 5658, 3611, 1052, 3104, 1057, 3
626, 4141, 1586, 3638, 2047, 1600, 1089, 2117, 3141, 3653, 2120, 5701, 4686, 593, 1105, 3155, 2134, 5206, 3162, 5723, 609, 610,
612, 101, 616, 5736, 4714, 2672, 3185, 5236, 1655, 4736, 1665, 2690, 5761, 132, 3205, 5766, 137, 5770, 5792, 3237, 3238, 2227,
3253, 1718, 183, 1719, 1209, 2234, 4790, 5313, 4291, 2757, 4805, 5318, 5831, 4297, 208, 721, 1233, 5841, 1238, 3297, 1772, 177
3, 254, 4872, 3850, 2318, 3854, 2327, 3864, 4891, 796, 4386, 2342, 808, 1320, 3368, 2347, 1324, 815, 2871, 3895, 3390, 1344, 18
59, 3907, 1864, 335, 4435, 3929, 4967, 2418, 2934, 3960, 892, 1405, 1920, 904, 5006, 4497, 5009, 2452, 5525, 3479, 409, 1434, 1
947, 415, 4515, 1444, 3496, 427, 1965, 2482, 2488, 4538, 2500, 3018, 3020, 3022, 3033, 1499, 997, 4070, 5095, 494, 1010, 5108,
502, 4605, 4094, 3583]}
Time taken to fetch (boolean query Union):  0.0030341148376464844 seconds
```

Implementing Phrase Query

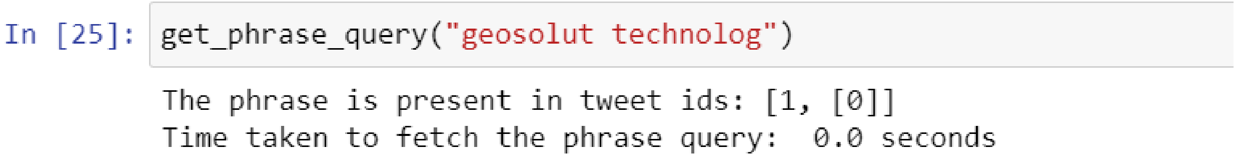


```
In [24]: #For finding if two words occur together and in which document.
         filtered_Sentence = df['Final']
         def get_phrase_query(phrase):

             start=time.time()
             str_to_process = phrase.split()

             i=0
             j=0

             lim1=0
             lim2=0

             ans=[]
             if (str_to_process[0] in postings) and (str_to_process[1] in postings):
                 while (lim1<len(postings[str_to_process[0]][1]) and lim2<len(postings[str_to_process[1]][1]) ):
                     if(postings[str_to_process[0]][1][i] == postings[str_to_process[1]][1][j]):
                         ans.append(postings[str_to_process[1]][1][j])
                         i+=1
                         j+=1

                     elif (postings[str_to_process[0]][1][i] < postings[str_to_process[1]][1][j]):
                         i+=1

                     else:
                         j+=1

                     lim1+=1
                     lim2+=1
```



```
    else:
        print("Not found in any tweet")

    final_tweets_id=[]
    pos_idx = []
    for p in ans:
        held_for_now=filtered_Sentence[p].split()

        if( held_for_now.index(str_to_process[0]) == (held_for_now.index(str_to_process[1])-1) ):
            final_tweets_id.append(p)
            pos_idx.append(len(final_tweets_id))
            pos_idx.append(final_tweets_id)

    end=time.time()
    time_taken=end-start

    print("The phrase is present in tweet ids:",pos_idx)
    print("Time taken to fetch the phrase query: ",time_taken,"seconds")
```

Taking an example to demonstrate Phrase Query



```
In [25]: get_phrase_query("geosolut technolog")

         The phrase is present in tweet ids: [1, [0]]
         Time taken to fetch the phrase query:  0.0 seconds
```

Dataset link –

Financial Sentiment Analysis

https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis

Github Link to the Assignment-

https://github.com/NikhilAdyapak/AIWIR_Assignment_Team4