

# UE19CS332- AIW and IR

## Assignment 3

Nikhil Adyapak PES2UG19CS257

Team 4, D Section

### Team Members –

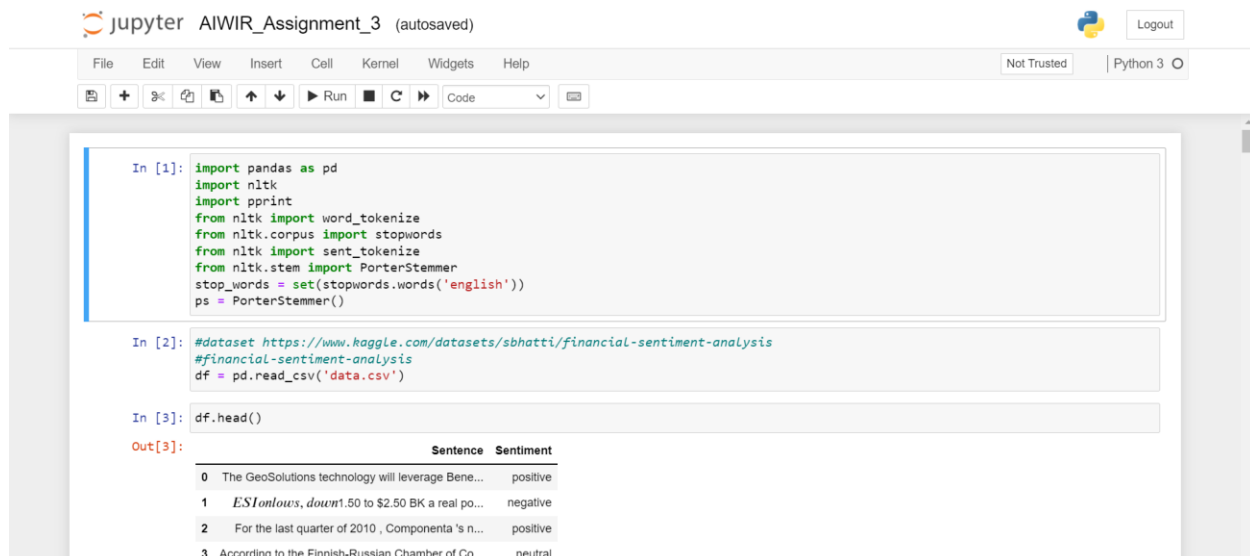
1. Manoj Kumar PES2UG19CS222
2. Nidhi Gupta PES2UG19CS256
3. Nikhil Adyapak PES2UG19CS257

### Dataset used –

#### Financial Sentiment Analysis

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

#### Importing Libraries and checking the Dataset



The screenshot shows a Jupyter Notebook titled "AIWIR\_Assignment\_3 (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook contains three input cells and one output cell.

```
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()

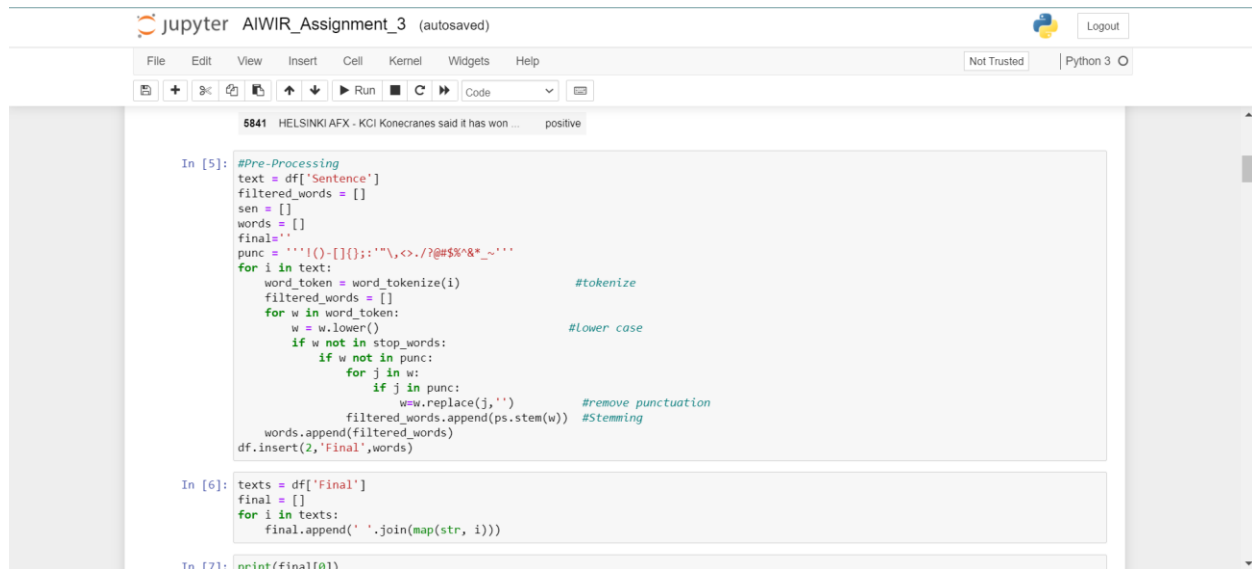
In [2]: #dataset https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis
#financial-sentiment-analysis
df = pd.read_csv('data.csv')

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

The output of the third cell is a table showing the first four rows of the dataset:

	Sentence	Sentiment
0	The GeoSolutions technology will leverage Bene...	positive
1	ESI on lows, 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



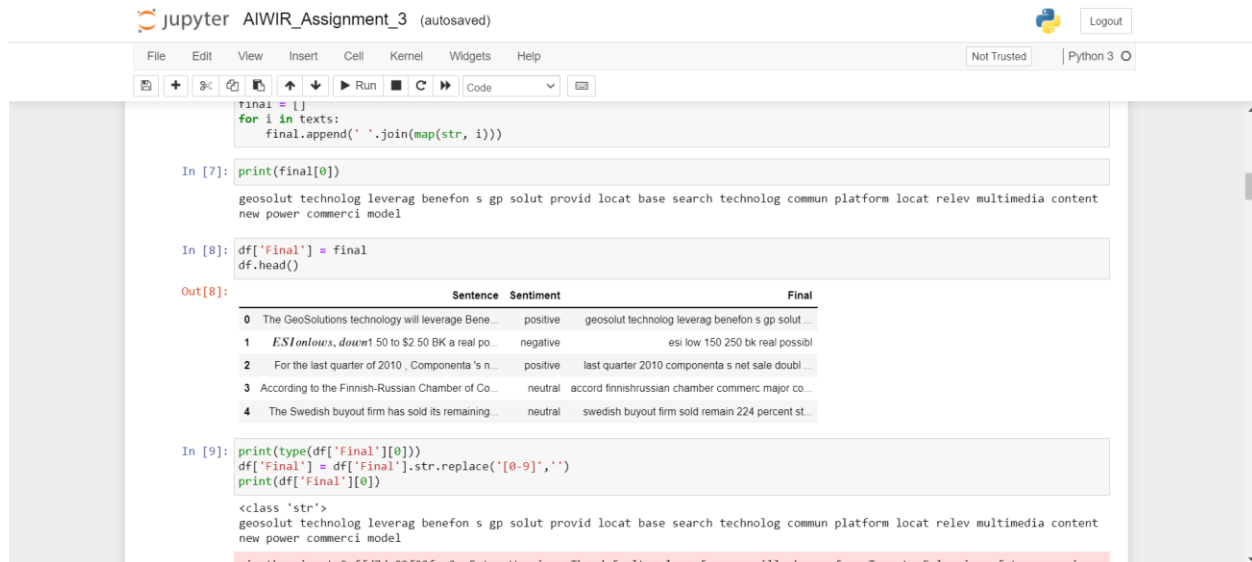
The screenshot shows a Jupyter Notebook titled "AIWIR\_Assignment\_3 (autosaved)". The code in the first cell (In [5]) performs the following steps:

- Extracts the 'Sentence' column from a DataFrame into a list 'text'.
- Initializes empty lists for 'filtered\_words', 'sen', 'words', and 'final'.
- Defines a punctuation string 'punc' containing various punctuation marks.
- Iterates over each sentence in 'text':
  - Tokenizes the sentence into 'word\_token'.
  - Converts tokens to lowercase ('w = w.lower()').
  - Filters out words in 'stop\_words'.
  - Removes punctuation from words ('w = w.replace(j, '')').
  - Stems the words using 'ps.stem(w)' and appends them to 'filtered\_words'.
- Appends the processed words to the 'words' list and inserts a new entry into the 'final' list.

The second cell (In [6]) extracts the 'Final' column into 'texts' and joins the words back into sentences.

The third cell (In [7]) prints the first element of the 'final' list.

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



The screenshot shows the continuation of the Jupyter Notebook. The code in the second cell (In [6]) is repeated. The third cell (In [7]) prints the first element of the 'final' list, showing a pre-processed sentence.

The fourth cell (In [8]) updates the 'Final' column in the DataFrame with the processed sentences and displays the first few rows of the DataFrame.

	Sentence	Sentiment	Final
0	The GeoSolutions technology will leverage Bene...	positive	geosolut technolog leverag benefon s gp solut...
1	ESI onlows, 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...

The fifth cell (In [9]) prints the type of the first element in the 'Final' column, which is a string, and then prints the first element of the 'Final' column.

## Generating the Inverted Index

```

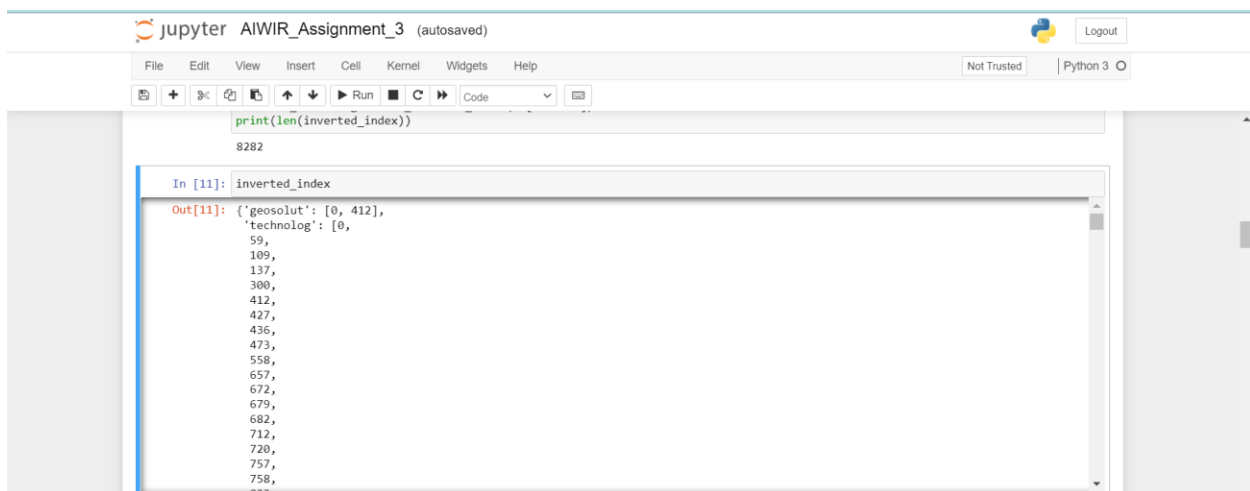
In [10]: #inverted index
def generate_inverted_index(data: list):
    inv_idx_dict = {}

    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



Jupyter AIWIR\_Assignment\_3 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted Python 3

```

print(len(inverted_index))

```

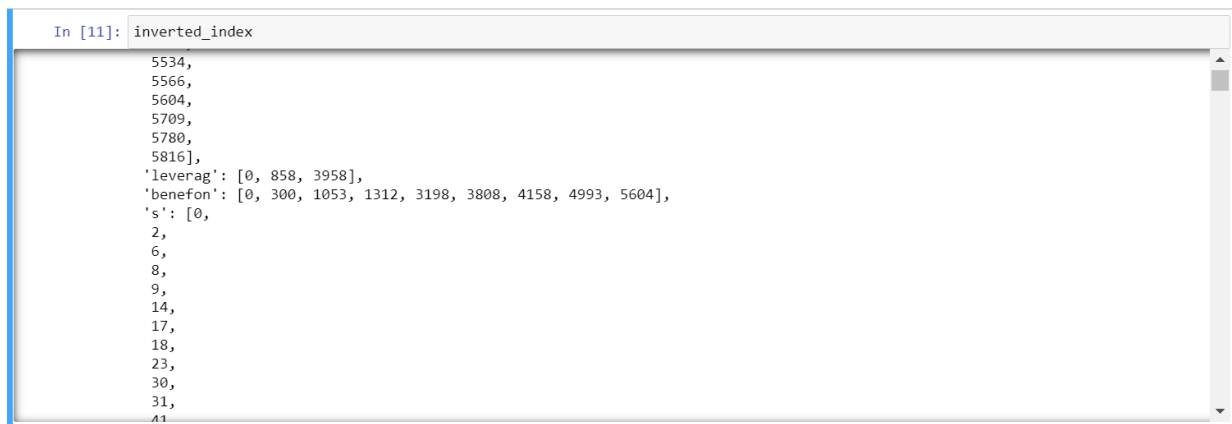
8282

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,
                        8282]}

```



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,
      41]

```

```
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,
1336
```

## Generating the Posting List

```
jupyter AIWIR_Assignment_3 (autosaved) Logout
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3
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,
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
456, 4567, 4568, 2461, 4569, 2466, 4570, 2467, 4571, 4572, 2476, 4577, 4578, 2479, 4580, 2483, 4584, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 2680, 2681, 2682, 2683, 2684, 2685, 2686, 2687, 2688, 2689, 2690, 2691, 2692, 2693, 2694, 2695, 2696, 2697, 2698, 2699, 2700, 2701, 2702, 2703, 2704, 2705, 2706, 2707, 2708, 2709, 2710, 2711, 2712, 2713, 2714, 2715, 2716, 2717, 2718, 2719, 2720, 2721, 2722, 2723, 2724, 2725, 2726, 2727, 2728, 2729, 2730, 2731, 2732, 2733, 2734, 2735, 2736, 2737, 2738, 2739, 2740, 2741, 2742, 2743, 2744, 2745, 2746, 2747, 2748, 2749, 2750, 2751, 2752, 2753, 2754, 2755, 2756, 2757, 2758, 2759, 2760, 2761, 2762, 2763, 2764, 2765, 2766, 2767, 2768, 2769, 2770, 2771, 2772, 2773, 2774, 2775, 2776, 2777, 2778, 2779, 2780, 2781, 2782, 2783, 2784, 2785, 2786, 2787, 2788, 2789, 2790, 2791, 2792, 2793, 2794, 2795, 2796, 2797, 2798, 2799, 2800, 2801, 2802, 2803, 2804, 2805, 2806, 2807, 2808, 2809, 2810, 2811, 2812, 2813, 2814, 2815, 2816, 2817, 2818, 2819, 2820, 2821, 2822, 2823, 2824, 2825, 2826, 2827, 2828, 2829, 2830, 2831, 2832, 2833, 2834, 2835, 2836, 2837, 2838, 2839, 2840, 2841, 2842, 2843, 2844, 2845, 2846, 2847, 2848, 2849, 2850, 2851, 2852, 2853, 2854, 2855, 2856, 2857, 2858, 2859, 2860, 2861, 2862, 2863, 2864, 2865, 2866, 2867, 2868, 2869, 2870, 2871, 2872, 2873, 2874, 2875, 2876, 2877, 2878, 2879, 2880, 2881, 2882, 2883, 2884, 2885, 2886, 2887, 2888, 2889, 2890, 2891, 2892, 2893, 2894, 2895, 2896, 2897, 2898, 2899, 2900, 2901, 2902, 2903, 2904, 2905, 2906, 2907, 2908, 2909, 2910, 2911, 2912, 2913, 2914, 2915, 2916, 2917, 2918, 2919, 2920, 2921, 2922, 2923, 2924, 2925, 2926, 2927, 2928, 2929, 2930, 2931, 2932, 2933, 2934, 2935, 2936, 2937, 2938, 2939, 2940, 2941, 2942, 2943, 2944, 2945, 2946, 2947, 2948, 2949, 2950, 2951, 2952, 2953, 2954, 2955, 2956, 2957, 2958, 2959, 2960, 2961, 2962, 2963, 2964, 2965, 2966, 2967, 2968, 2969, 2970, 2971, 2972, 2973, 2974, 2975, 2976, 2977, 2978, 2979, 2980, 2981, 2982, 2983, 2984, 2985, 2986, 2987, 2988, 2989, 2990, 2991, 2992, 2993, 2994, 2995, 2996, 2997, 2998, 2999, 3000, 3001, 3002, 3003, 3004, 3005, 3006, 3007, 3008, 3009, 3010, 3011, 3012, 3013, 3014, 3015, 3016, 3017, 3018, 3019, 3020, 3021, 3022, 3023, 3024, 3025, 3026, 3027, 3028, 3029, 3030, 3031, 3032, 3033, 3034, 3035, 3036, 3037, 3038, 3039, 3040, 3041, 3042, 3043, 3044, 3045, 3046, 3047, 3048, 3049, 3050, 3051, 3052, 3053, 3054, 3055, 3056, 3057, 3058, 3059, 3060, 3061, 3062, 3063, 3064, 3065, 3066, 3067, 3068, 3069, 3070, 3071, 3072, 3073, 3074, 3075, 3076, 3077, 3078, 3079, 3080, 3081, 3082, 3083, 3084, 3085, 3086, 3087, 3088, 3089, 3090, 3091, 3092, 3093, 3094, 3095, 3096, 3097, 3098, 3099, 3100, 3101, 3102, 3103, 3104, 3105, 3106, 3107, 3108, 3109, 3110, 3111, 3112, 3113, 3114, 3115, 3116, 3117, 3118, 3119, 3120, 3121, 3122, 3123, 3124, 3125, 3126, 3127, 3128, 3129, 3130, 3131, 3132, 3133, 3134, 3135, 3136, 3137, 3138, 3139, 3140, 3141, 3142, 3143, 3144, 3145, 3146, 3147, 3148, 3149, 3150, 3151, 3152, 3153, 3154, 3155, 3156, 3157, 3158, 3159, 3160, 3161, 3162, 3163, 3164, 3165, 3166, 3167, 3168, 3169, 3170, 3171, 3172, 3173, 3174, 3175, 3176, 3177, 3178, 3179, 3180, 3181, 3182, 3183, 3184, 3185, 3186, 3187, 3188, 3189, 3190, 3191, 3192, 3193, 3194, 3195, 3196, 3197, 3198, 3199, 3200, 3201, 3202, 3203, 3204, 3205, 3206, 3207, 3208, 3209, 3210, 3211, 3212, 3213, 3214, 3215, 3216, 3217, 3218, 3219, 3220, 3221, 3222, 3223, 3224, 3225, 3226, 3227, 3228, 3229, 3230, 3231, 3232, 3233, 3234, 3235, 3236, 3237, 3238, 3239, 3240, 3241, 3242, 3243, 3244, 3245, 3246, 3247, 3248, 3249, 3250, 3251, 3252, 3253, 3254, 3255, 3256, 3257, 3258, 3259, 3260, 3261, 3262, 3263, 3264, 3265, 3266, 3267, 3268, 3269, 3270, 3271, 3272, 3273, 3274, 3275, 3276, 3277, 3278, 3279, 3280, 3281, 3282, 3283, 3284, 3285, 3286, 3287, 3288, 3289, 3290, 3291, 3292, 3293, 3294, 3295, 3296, 3297, 3298, 3299, 3300, 3301, 3302, 3303, 3304, 3305, 3306, 3307, 3308, 3309, 3310, 3311, 3312, 3313, 3314, 3315, 3316, 3317, 3318, 3319, 3320, 3321, 3322, 3323, 3324, 3325, 3326, 3327, 3328, 3329, 3330, 3331, 3332, 3333, 3334, 3335, 3336, 3337, 3338, 3339, 3340, 3341, 3342, 3343, 3344, 3345, 3346, 3347, 3348, 3349, 3350, 3351, 3352, 3353, 3354, 3355, 3356, 3357, 3358, 3359, 3360, 3361, 3362, 3363, 3364, 3365, 3366, 3367, 3368, 3369, 3370, 3371, 3372, 3373, 3374, 3375, 3376, 3377, 3378, 3379, 3380, 3381, 3382, 3383, 3384, 3385, 3386, 3387, 3388, 3389, 3390, 3391, 3392, 3393, 3394, 3395, 3396, 3397, 3398, 3399, 3400, 3401, 3402, 3403, 3404, 3405, 3406, 3407, 3408, 3409, 3410, 3411, 3412, 3413, 3414, 3415, 3416, 3417, 3418, 3419, 3420, 3421, 3422, 3423, 3424, 3425, 3426, 3427, 3428, 3429, 3430, 3431, 3432, 3433, 3434, 3435, 3436, 3437, 3438, 3439, 3440, 3441, 3442, 3443, 3444, 3445, 3446, 3447, 3448, 3449, 3450, 3451, 3452, 3453, 3454, 3455, 3456, 3457, 3458, 3459, 3460, 3461, 3462, 3463, 3464, 3465, 3466, 3467, 3468, 3469, 3470, 3471, 3472, 3473, 3474, 3475, 3476, 3477, 3478, 3479, 3480, 3481, 3482, 3483, 3484, 3485, 3486, 3487, 3488, 3489, 3490, 3491, 3492, 3493, 3494, 3495, 3496, 3497, 3498, 3499, 3500, 3501, 3502, 3503, 3504, 3505, 3506, 3507, 3508, 3509, 3510, 3511, 3512, 3513, 3514, 3515, 3516, 3517, 3518, 3519, 3520, 3521, 3522, 3523, 3524, 3525, 3526, 3527, 3528, 3529, 3530, 3531, 3532, 3533, 3534, 3535, 3536, 3537, 3538, 3539, 3540, 3541, 3542, 3543, 3544, 3545, 3546, 3547, 3548, 3549, 3550, 3551, 3552, 3553, 3554, 3555, 3556, 3557, 3558, 3559, 3560, 3561, 3562, 3563, 3564, 3565, 3566, 3567, 3568, 3569, 3570, 3571, 3572, 3573, 3574, 3575, 3576, 3577, 3578, 3579, 3580, 3581, 3582, 3583, 3584, 3585, 3586, 3587, 3588, 3589, 3590, 3591, 3592, 3593, 3594, 3595, 3596, 3597, 3598, 3599, 3600, 3601, 3602, 3603, 3604, 3605, 3606, 3607, 3608, 3609, 3610, 3611, 3612, 3613, 3614, 3615, 3616, 3617, 3618, 3619, 3620, 3621, 3622, 3623, 3624, 3625, 3626, 3627, 3628, 3629, 3630, 3631, 3632, 3633, 3634, 3635, 3636, 3637, 3638, 3639, 3640, 3641, 3642, 3643, 3644, 3645, 3646, 3647, 3648, 3649, 3650, 3651, 3652, 3653, 3654, 3655, 3656, 3657, 3658, 3659, 3660, 3661, 3662, 3663, 3664, 3665, 3666, 3667, 3668, 3669, 3670, 3671, 3672, 3673, 3674, 3675, 3676, 3677, 3678, 3679, 3680, 3681, 3682, 3683, 3684, 3685, 3686, 3687, 3688, 3689, 3690, 3691, 3692, 3693, 3694, 3695, 3696, 3697, 3698, 3699, 3700, 3701, 3702, 3703, 3704, 3705, 3706, 3707, 3708, 3709, 3710, 3711, 3712, 3713, 3714, 3715, 3716, 3717, 3718, 3719, 3720, 3721, 3722, 3723, 3724, 3725, 3726, 3727, 3728, 3729, 3730, 3731, 3732, 3733, 3734, 3735, 3736, 3737, 3738, 3739, 3740, 3741, 3742, 3743, 3744, 3745, 3746, 3747, 3748, 3749, 3750, 3751, 3752, 3753, 3754, 3755, 3756, 3757, 3758, 3759, 3760, 3761, 3762, 3763, 3764, 3765, 3766, 3767, 3768, 3769, 3770, 3771, 3772, 3773, 3774, 3775, 3776, 3777, 3778, 3779, 3780, 3781, 3782, 3783, 3784, 3785, 3786, 3787, 3788, 3789, 3790, 3791, 3792, 3793, 3794, 3795, 3796, 3797, 3798, 3799, 3800, 3801, 3802, 3803, 3804, 3805, 3806, 3807, 3808, 3809, 3810, 3811, 3812, 3813, 3814, 3815, 3816, 3817, 3818, 3819, 3820, 3821, 3822, 3823, 3824, 3825, 3826, 3827, 3828, 3829, 3830, 3831, 3832, 3833, 3834, 3835, 3836, 3837, 3838, 3839, 3840, 3841, 3842, 3843, 3844, 3845, 3846, 3847, 3848, 3849, 3850, 3851, 3852, 3853, 3854, 3855, 3856, 3857, 3858, 3859, 3860, 3861, 3862, 3863, 3864, 3865, 3866, 3867, 3868, 3869, 3870, 3871, 3872, 3873, 3874, 3875, 3876, 3877, 3878, 3879, 3880, 3881, 3882, 3883, 3884, 3885, 3886, 3887, 3888, 3889, 3890, 3891, 3892, 3893, 3894, 3895, 3896, 3897, 3898, 3899, 3900, 3901, 3902, 3903, 3904, 3905, 3906, 3907, 3908, 3909, 3910, 3911, 3912, 3913, 3914, 3915, 3916, 3917, 3918, 3919, 3920, 3921, 3922, 3923, 3924, 3925, 3926, 3927, 3928, 3929, 3930, 3931, 3932, 3933, 3934, 3935, 3936, 3937, 3938, 3939, 3940, 3941, 3942, 3943, 3944, 3945, 3946, 3947, 3948, 3949, 3950, 3951, 3952, 3953, 3954, 3955, 3956, 3957, 3958, 3959, 3960, 3961, 3962, 3963, 3964, 3965, 3966, 3967, 3968, 3969, 3970, 3971, 3972, 3973, 3974, 3975, 3976, 3977, 3978, 3979, 3980, 3981, 3982, 3983, 3984, 3985, 3986, 3987, 3988, 3989, 3990, 3991, 3992, 3993, 3994, 3995, 3996, 3997, 3998, 3999, 4000, 4001, 4002, 4003, 4004, 4005, 4006, 4007, 4008, 4009, 4010, 4011, 4012, 4013, 4014, 4015, 4016, 4017, 4018, 4019, 4020, 4021, 4022, 4023, 4024, 4025, 4026, 4027, 4028, 4029, 4030, 4031, 4032, 4033, 4034, 4035, 4036, 4037, 4038, 4039, 4040, 4041, 4042, 4043, 4044, 4045, 4046, 4047, 4048, 4049, 4050, 4051, 4052, 4053, 4054, 4055, 4056, 4057, 4058, 4059, 4060, 4061, 4062, 4063, 4064, 4065, 4066, 4067, 4068, 4069, 4070, 4071, 4072, 4073, 4074, 4075, 4076, 4077, 4078, 4079, 4080, 4081, 4082, 4083, 4084, 4085, 4086, 4087, 4088, 4089, 4090, 4091, 4092, 4093, 4094, 4095, 4096, 4097, 4098, 4099, 4100, 4101, 4102, 4103, 4104, 41
```

```
(('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]: 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+ñinnen': [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(0.000000001)
            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

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](https://github.com/NikhilAdyapak/AIWIR_Assignment_Team4)