Importing Libraries

In [1]:

```
import nltk
import numpy as np
import pandas as pd
```

In [2]:

```
nltk.download("punkt")
nltk.download("stopwords")
nltk.download("wordnet")
nltk.download('omw-1.4')
from nltk.tokenize import word_tokenize
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Initialising Preprocessing Functions

In [3]:

```
stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.SnowballStemmer('english')
lemmatizer = nltk.stem.WordNetLemmatizer()
```

Loading Data

In [4]:

```
df_train = pd.read_csv("../data/disaster/train.csv")
df_test = pd.read_csv("../data/disaster/test.csv")
df_train.drop(columns=["target"], inplace=True)
df = pd.concat([df_train, df_test])
```

In [5]:

```
print(df.shape)
df.head()
```

```
(10876, 4)
```

Out[5]:

	id	keyword	location	text
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...

In [6]:

```
corpus = df["text"].values
```

Week 1 - 21/01/2022

Tokenizing Corpus

In [7]:

```
def tokenize_corpus(tweets):
    return list(map(word_tokenize, tweets))
```

In [8]:

```
tokenized_corpus = tokenize_corpus(corpus)
```

In [9]:

```
for i in range(3):
    print(f"Tweet {i}: {tokenized_corpus[i]}")
```

```
Tweet 0: ['Our', 'Deeds', 'are', 'the', 'Reason', 'of', 'this', '#', 'earthq
uake', 'May', 'ALLAH', 'Forgive', 'us', 'all']
Tweet 1: ['Forest', 'fire', 'near', 'La', 'Ronge', 'Sask', '.', 'Canada']
Tweet 2: ['All', 'residents', 'asked', 'to', "'shelter", 'in', 'place', "'",
'are', 'being', 'notified', 'by', 'officers', '.', 'No', 'other', 'evacuatio
n', 'or', 'shelter', 'in', 'place', 'orders', 'are', 'expected']
```

Removing Stopwords

In [10]:

```
def remove_stopwords(tokenized_tweets):
    _function = lambda tweet: [word for word in tweet if word not in stopwords]
    return list(map(_function, tokenized_tweets))
```

In [11]:

```
tokenized_nostopw_corpus = remove_stopwords(tokenized_corpus)
```

In [12]:

```
for i in range(3):
    print(f"Tweet {i}: {tokenized_nostopw_corpus[i]}")
```

```
Tweet 0: ['Our', 'Deeds', 'Reason', '#', 'earthquake', 'May', 'ALLAH', 'Forg
ive', 'us']
Tweet 1: ['Forest', 'fire', 'near', 'La', 'Ronge', 'Sask', '.', 'Canada']
Tweet 2: ['All', 'residents', 'asked', "'shelter", 'place', "'", 'notified',
'officers', '.', 'No', 'evacuation', 'shelter', 'place', 'orders', 'expecte
d']
```

Creating Inverted Index

In [13]:

```
def create_inverted_index(tokenized_tweets):
    inverted_index = dict()
    for document_idx in range(len(tokenized_tweets)):
        for word in tokenized_tweets[document_idx]:
            if word not in inverted_index:
                inverted_index[word] = list()
            inverted_index[word].append(document_idx)

    for key in inverted_index:
        inverted_index[key] = list(set(inverted_index[key]))

    return inverted_index
```

In [14]:

```
inverted_index = create_inverted_index(tokenized_nostopw_corpus)
```

In [15]:

```
for key in list(inverted_index.keys())[:3]:
    print(f"Word: {key}\nTweet Indices: {inverted_index[key]}\n")
```

```
Word: Our
Tweet Indices: [0, 9095, 7816, 1673, 1550, 8080, 3345, 8979, 4630, 3099, 297
6, 3618, 3235, 2855, 6567, 3369, 2220, 4659, 3124, 4024, 4281, 3786, 10829,
1371, 9570, 3172, 10603, 1645, 4209, 7157, 4987, 2431]

Word: Deeds
Tweet Indices: [0, 4985]

Word: Reason
Tweet Indices: [0, 8610, 304, 305, 317, 319]
```

Week 2 - 28/01/2022

Case Folding

In [16]:

```
def case_folding(tokenized_tweets):
    _function = lambda tweet: [word.lower() for word in tweet]
    return list(map(_function, tokenized_tweets))
```

In [17]:

```
tokenized_nostopw_case_corpus = case_folding(tokenized_nostopw_corpus)
```

In [18]:

```
for i in range(3):
    print(f"Tweet {i}: {tokenized_nostopw_case_corpus[i]}")
```

```
Tweet 0: ['our', 'deeds', 'reason', '#', 'earthquake', 'may', 'allah', 'forg
ive', 'us']
Tweet 1: ['forest', 'fire', 'near', 'la', 'ronge', 'sask', '.', 'canada']
Tweet 2: ['all', 'residents', 'asked', "'shelter", 'place', "'", 'notified',
'officers', '.', 'no', 'evacuation', 'shelter', 'place', 'orders', 'expecte
d']
```

Lemmatizing Words

In [19]:

```
def lemmatize_words(tokenized_tweets):
    _function = lambda tweet: [lemmatizer.lemmatize(word) for word in tweet]
    return list(map(_function, tokenized_tweets))
```

In [20]:

```
tokenized_nostopw_case_lemm_corpus = lemmatize_words(tokenized_nostopw_case_corpus)
```

In [21]:

```
for i in range(3):
    print(f"Tweet {i}: {tokenized_nostopw_case_lemm_corpus[i]}")
```

```
Tweet 0: ['our', 'deed', 'reason', '#', 'earthquake', 'may', 'allah', 'forgi
ve', 'u']
Tweet 1: ['forest', 'fire', 'near', 'la', 'ronge', 'sask', '.', 'canada']
Tweet 2: ['all', 'resident', 'asked', "'shelter", 'place', "'", 'notified',
'officer', '.', 'no', 'evacuation', 'shelter', 'place', 'order', 'expected']
```

Stemming

In [22]:

```
def stem_words(tokenized_tweets):
    _function = lambda tweet: [stemmer.stem(word) for word in tweet]
    return list(map(_function, tokenized_tweets))
```

In [23]:

```
tokenized_nostopw_case_stem_corpus = lemmatize_words(tokenized_nostopw_case_corpus)
```

In [24]:

```
for i in range(3):
    print(f"Tweet {i}: {tokenized_nostopw_case_stem_corpus[i]}")
```

```
Tweet 0: ['our', 'deed', 'reason', '#', 'earthquake', 'may', 'allah', 'forgi
ve', 'u']
Tweet 1: ['forest', 'fire', 'near', 'la', 'ronge', 'sask', '.', 'canada']
Tweet 2: ['all', 'resident', 'asked', "'shelter", 'place', "'", 'notified',
'officer', '.', 'no', 'evacuation', 'shelter', 'place', 'order', 'expected']
```

Created new Inverted Index

In [25]:

```
inverted_index_processed = create_inverted_index(tokenized_nostopw_case_lemm_corpus) # toke
```

In [26]:

```
for key in list(inverted_index_processed.keys())[:3]:
    print(f"Word: {key}\nTweet Indices: {inverted_index_processed[key]}\n")
```

```
Word: our
Tweet Indices: [0, 9095, 7816, 1673, 1550, 8080, 3345, 8979, 4630, 3099, 297
6, 3618, 3235, 2855, 6567, 3369, 2220, 4659, 3124, 4024, 4281, 3786, 10829,
1371, 9570, 3172, 10603, 1645, 4209, 7157, 4987, 2431]

Word: deed
Tweet Indices: [0, 4985]

Word: reason
Tweet Indices: [0, 1920, 10496, 4997, 5000, 5001, 9737, 9739, 781, 8593, 537
2, 6166, 7961, 10526, 8224, 8610, 10670, 304, 305, 7218, 6453, 9911, 2747, 6
459, 4669, 317, 319, 2112, 7740, 8001, 8260, 2252, 6991, 6232, 6242, 746, 48
43, 4333, 6898, 4084, 763, 3452, 894, 4991]
```

Week 3 - 04/02/2022

Boolean Query

In [27]:

```
def parse_query(infix_tokens):

    precedence = {}
    precedence['NOT'] = 3
    precedence['AND'] = 2
    precedence['OR'] = 1
    precedence['('] = 0
    precedence[')'] = 0

    output = []
    operator_stack = []

    for token in infix_tokens:
        if (token == '('):
            operator_stack.append(token)

        elif (token == ')'):
            operator = operator_stack.pop()
            while operator != '(':
                output.append(operator)
                operator = operator_stack.pop()

        elif (token in precedence):
            if (operator_stack):
                current_operator = operator_stack[-1]
                while (operator_stack and precedence[current_operator] > precedence[token]):
                    output.append(operator_stack.pop())
                    if (operator_stack):
                        current_operator = operator_stack[-1]
            operator_stack.append(token) # add token to stack
        else:
            output.append(token.lower())

    while (operator_stack):
        output.append(operator_stack.pop())

    return output
```

In [28]:

```
def boolean_query(query, inverted_index):
    query = query.strip()
    query_tokens = query.split()
    boolean_query = parse_query(query_tokens)

    result_stack = list()
    for idx, token in enumerate(boolean_query):
        if token not in ["AND", "NOT", "OR"]:
            result = set(inverted_index[token])
        else:
            if token in ['AND', 'OR']:
                right_operand = result_stack.pop()
                left_operand = result_stack.pop()

                if token == 'AND':
                    operation = set.intersection
                else:
                    operation = set.union

                result = operation(left_operand, right_operand)

            else:
                operand = result_stack.pop()
                complement_document_ids = inverted_index[boolean_query[idx-1]]
                result = list()
                for word in inverted_index:
                    result.extend([_id for _id in inverted_index[word] if _id not in comple
                result = set(result)

        result_stack.append(result)

    return result_stack.pop()
```

In [29]:

```
document_ids = boolean_query("our AND deed", inverted_index_processed)
print(f"Document IDs: {document_ids}")
```

```
Document IDs: {0}
```

In [30]:

```
document_ids = boolean_query("our AND deed OR india", inverted_index_processed)
print(f"Document IDs: {document_ids}")
```

```
Document IDs: {0, 4480, 2435, 4483, 2696, 2827, 6795, 7438, 3987, 9503, 950
4, 6564, 1710, 1712, 2484, 4534, 1600, 2373, 2376, 2377, 5450, 10443, 8652,
8654, 2386, 2390, 471, 2392, 2391, 9302, 6627, 8677, 6631, 9579, 4468, 6776,
10493}
```

Positional Index

In [31]:

```
def construct_positional_posting_list(tokenized_corpus):
    positional_index = dict()
    for tweet_id, tweet in enumerate(tokenized_corpus):
        for token_id, token in enumerate(tweet):
            if token not in positional_index:
                positional_index[token] = dict()
            if tweet_id not in positional_index[token]:
                positional_index[token][tweet_id] = list()
            positional_index[token][tweet_id].append(token_id)

    for token in positional_index:
        for tweet_id in positional_index[token]:
            positional_index[token][tweet_id] = sorted(positional_index[token][tweet_id])
        items = list(positional_index[token].items())
        items.sort(key=lambda x: x[0])
        for k, v in items:
            positional_index[token][k] = v

    return positional_index
```

In [32]:

```
positional_index = construct_positional_posting_list(tokenized_nostopw_case_lemm_corpus)
```

In [33]:

```
for key in list(positional_index.keys())[:3]:
    print(f"Word: {key}\nTweet & Token Indices: {positional_index[key]}\n")
```

```
Word: our
Tweet & Token Indices: {0: [0], 1371: [6], 1550: [0], 1645: [5], 1673: [4],
2220: [8], 2431: [4], 2855: [11], 2976: [7], 3099: [12], 3124: [12], 3172:
[0], 3235: [0], 3345: [0], 3369: [0], 3618: [5], 3786: [0], 4024: [0], 4209:
[0], 4281: [8], 4630: [0], 4659: [6], 4987: [8], 6567: [10], 7157: [4], 781
6: [16, 22], 8080: [0], 8979: [8], 9095: [0], 9570: [0], 10603: [14], 10829:
[0]}

Word: deed
Tweet & Token Indices: {0: [1], 4985: [9]}

Word: reason
Tweet & Token Indices: {0: [2], 304: [19], 305: [19], 317: [19], 319: [19],
746: [4], 763: [3], 781: [7], 894: [8], 1920: [0], 2112: [10], 2252: [3], 27
47: [12], 3452: [0], 4084: [7], 4333: [7], 4669: [2], 4843: [8], 4991: [1],
4997: [1], 5000: [1], 5001: [1], 5372: [13], 6166: [0], 6232: [1], 6242:
[3], 6453: [9], 6459: [2], 6898: [11], 6991: [0], 7218: [9], 7740: [5], 796
1: [1], 8001: [2], 8224: [7], 8260: [6], 8593: [9], 8610: [16], 9737: [1], 9
739: [1], 9911: [9], 10496: [7], 10526: [3], 10670: [2]}
```

Phrase Query with Postional Index

In [34]:

```
def positional_intersect(p1, p2, K):
    answer = list()
    i = 0
    j = 0
    while i < len(p1) and j < len(p2):
        document_id_p1 = list(p1.keys())[i]
        document_id_p2 = list(p2.keys())[j]

        if document_id_p1 == document_id_p2:
            l = list()
            pp1 = p1[document_id_p1]
            pp2 = p2[document_id_p2]

            k = 0
            while k < len(pp1):
                m = 0
                while m < len(pp2):
                    distance = pp2[m] - pp1[k]
                    if distance == K:
                        l.append(m)
                    m += 1

                for ps in l:
                    distance = (pp2[ps] - pp1[k])
                    if distance != K:
                        l.remove(ps)

                for ps in l:
                    answer.append((document_id_p1, pp1[k], pp2[ps]))

                k += 1

            i += 1
            j += 1

        elif document_id_p1 < document_id_p2:
            i += 1
        else:
            j += 1

    return answer
```

In [35]:

```
to = {
    1: [7,18,33,72,86,231],
    2: [1,17,74,222,255],
    4: [8,16,190,429,433],
    5: [363,367],
    7: [13,23,191]
}

be = {
    1: [17,25],
    4: [17,191,291,430,434],
    5: [14,19,101]
}
```

In [36]:

```
positional_intersect(to, be, 1)
```

Out[36]:

```
[(4, 16, 17), (4, 190, 191), (4, 429, 430), (4, 433, 434)]
```

In [37]:

```
def search_positional_index(query):
    query = query.split()
    words = [query[i] for i in range(0, len(query), 2)]
    k = [int(query[i][1:]) for i in range(1, len(query), 2)]

    document_list = list()
    for i in range(0, len(words)-1):
        word1, word2 = words[i:i+2]
        p1 = positional_index[word1]
        p2 = positional_index[word2]
        result = positional_intersect(p1, p2, k[i])
        document_list.extend(result)

    return document_list
```

In [38]:

```
search_positional_index("to /1 be")
```

Out[38]:

```
[(4921, 4, 5),
 (4933, 4, 5),
 (5099, 3, 4),
 (7788, 6, 7),
 (8614, 4, 5),
 (9695, 4, 5),
 (9706, 4, 5),
 (9707, 4, 5),
 (9712, 4, 5),
 (9806, 8, 9)]
```