Assignment 1- FIT5196 Data Wrangling

Task 2: Text Pre-Processing

Name: Akshaya Kumar Chandrasekaran

ID : 31021301

In this section, an excel file with multiple sheets related to COVID-19 is given.

Contents begin at different places in each sheet.

Given:

An excel document with mutiple sheets(each sheet is a day's tweet)

Asked:

- To generate top 100 Unigrams per day
- To generate top 100 bi-grams per day
- To create an Vocab text after removing the context dependent stopwords, context independent stop words, words that are less than 3 in length.
  - Context Dependent Stop words Criteria : Words that occur in more than 60 days
  - Rare token : Words that occur in less than 5 days

2 Methodology to solve the problem

2.1 Importing packages:

Importing the allowed packages

In [1]:
```
from nltk.probability import FreqDist
import langid
from nltk.tokenize import RegexpTokenizer
import pandas as pd
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import MWETokenizer
from sklearn.feature_extraction.text import CountVectorizer
from nltk.probability import *
from itertools import chain
from nltk.util import ngrams
nltk.download('wordnet')
stemmer = PorterStemmer()
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\SAIC\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

2.2 Reading the files

The given excel file is read in python and is being stored. To complete this task, pandas package is used.

In [2]:
```
#reading the given excel file
given_excel = pd.ExcelFile(r'31021301.xlsx')
```

2.3 Importing context dependent stop words

In [3]:
```
#Opening the stop words text file in read only mode
file_stop = open("stopwords_en.txt","r")

#reading each line and storing it as a list.
stop_words_list = file_stop.read().splitlines()

#Closing the file
file_stop.close()

#Converting the list to set and then back to list to avoid duplicates
stop_words = list(set(stop_words_list))
```

2.4 Defining and declaring regex tokenizer to tokenize the words

In [4]:
```
#Declaring the regular expression to tokenize the words
tokenizer = RegexpTokenizer(r"[a-zA-Z]+(?:[-'][a-zA-Z]+)?")
```

2.5 Defining user defined functions:

In this section, few user defined functions are defined to create top unigrams per day, top bigrams per day.

In [5]:
```
'''
generate_unitoken_perday() is an user defined function that takes in a list as argument.
This function returns two lists,
unigrams_english_only --> Consists of only english tweets
token_day_unigram --> list of tokenized words per day
'''

#User defined function
def generate_unitoken_perday(only_text):

    #Declaring variables to store the values as list
    unigrams_english_only = []
    token_day_unigram = []

    #Iterating throught the range of tweets
    for i in range(len(only_text)):

        #Checking if the type is only string and if the tweet language is in english
        if((type(only_text[i]) == str) and langid.classify(only_text[i])[0]=='en'):

            #Encoding and decoding is done for emoji handling
            temp = only_text[i].encode('ascii','ignore').decode('unicode-escape')
            only_text[i] = temp.encode('utf-16', 'surrogatepass').decode('utf-16')

            #appending it to a list if the above conditions are met
            unigrams_english_only.append(only_text[i])

    #From the list got above, for each sentense applying the tokeniser defined above to extract only
    #required and relevant words.
    for j in range(len(unigrams_english_only)):

        #Applying on each sentense and extracting a list of words
        unigram_tokens_words = tokenizer.tokenize(unigrams_english_only[j].lower())

        #for the length of the list of words got above
        for k in range(len(unigram_tokens_words)):

            #checking if the length of the list is greater than 3 and
            #if the words are not in stop words
            if(len(unigram_tokens_words[k]) >=3 and (unigram_tokens_words[k] not in stop_words)):

                #Converting it to lower and then appending it to perday list
                token_day_unigram.append(unigram_tokens_words[k])

    #Returing values to function call
    return unigrams_english_only,token_day_unigram
```

In [6]:
```
'''
generate_bitoken_perday() is an user defined function that takes in only english tweets as inputs
and from that, apply tokenization and extract only the required and needed words. No removal of stop words or
words of length less than 3 is done as per the sample output given.

This method returns a list of words after tokenisation
'''

#user defined function
def generate_bitoken_perday(eng_only_uni):

    #Declaring variable to store the values as list
    token_day_bigram = []

    #Iterating throught the range of english tweets
    for j in range(len(eng_only_uni)):

        #Applying on each sentense and extracting a list of words
        unigram_tokens_words = tokenizer.tokenize(eng_only_uni[j])

        #for the length of the list of words got above
        for k in range(len(unigram_tokens_words)):

            #Converting it to lower and then appending it to perday list
            token_day_bigram.append(unigram_tokens_words[k].lower())

    #Returing values to function call
    return token_day_bigram
```

In [7]:
```
'''
generate_unigram() is a user defined function that takes in the stemmed vocabulary and the current date as input.

Based on the inputs, it will generate top 100 unigrams for the day and then write it to the text file.
'''

#User defined function
def generate_unigram(vocab_stemmed,curr_date):

    #using the function FreqDist to find each vocab's frequency
    unigram_curr_day = FreqDist(vocab_stemmed)

    #initialising a dict to store the values in key value pair
    uni_dic = {}

    #Based on the frequency from the above FreqDist fucntion, sorting it in reverse order
    #to get the top 100 unigrams for the day
    sorted_unigram_curr_day = sorted(unigram_curr_day.items(),key=lambda x:x[1],reverse=True)

    #openinng the file
    uni_word_file = open('31021301_100uni.txt','a')

    #openinng the file
    #Checking if the length of the list is greater than 100
    check_count = len(sorted_unigram_curr_day)


    #openinng the file
    #Checking if the length of the list is greater than 100
    #if the length is greater than 100, print only the top 100 words
    if(check_count >100):
        uni_dic[curr_date] = sorted_unigram_curr_day[0:100]

        #Writing each word and its corresponding frequency into the file
        for word, count in uni_dic.items():
            uni_word_file.write('%s:%s\n' % (word, count))

    #IF the length of the list is less than 100, print entire list
    else:
        uni_dic[curr_date] = sorted_unigram_curr_day

        #Writing each word and its corresponding frequency into the file
        for word, count in uni_dic.items():
            uni_word_file.write('%s:%s\n' % (word, count))

    #closing the file
    uni_word_file.close()
```

In [8]:
```
'''
generate_bigram() is a user defined function that takes in the vocabulary and the current date as input.

Based on the inputs, it will generate top 100 bigrams for the day and then write it to the text file.
'''
#User defined function
def generate_bigram(vocab_stemmed,curr_date):

    #Using the inbuilt package and collocation funtion to create most frequent 100 bigrams for the day
    bigram_measures = nltk.collocations.BigramAssocMeasures()

    #passing the token words
    bigram_finder = nltk.collocations.BigramCollocationFinder.from_words(vocab_stemmed)

    #Finding the top 100 bigrams
    bigram_dict = list(bigram_finder.ngram_fd.most_common(100))

    #a varibale to store is as key value pair
    bi_dic ={}

    #openinng the file
    bi_word_file = open('31021301_100bi.txt','a')


    #Checking if the length of the list is greater than 100
    check_count = len(bigram_dict)

    #if the length is greater than 100, print only the top 100 words
    if( check_count>100):

        bi_dic[curr_date] = bigram_dict[0:100]

        #Writing each word and its corresponding frequency into the file
        for word, count in bi_dic.items():
            bi_word_file.write('%s:%s\n' % (word, count))

    #IF the length of the list is less than 100, print entire list
    else:
        bi_dic[curr_date] = bigram_dict

        #Writing each word and its corresponding frequency into the file
        for word, count in bi_dic.items():
            bi_word_file.write('%s:%s\n' % (word, count))

    #File close
    bi_word_file.close()
```

In [9]:
```
'''
generate_raw is an user defined function that takes in list of all english tweets as input and tokenize it
and returns a list of tokenized words.
'''

#user defined function
def generate_raw_tokens(sep_tweets):

    #Empty list to append and store the tokenized words
    token_words = []

    #Iterating through each sentense(each tweet)
    for i in range(len(sep_tweets)):

        #Tokenizing it with the regular expression provided
        unigram_tokens_words = tokenizer.tokenize(sep_tweets[i])

        #For the length of each token
        for j in range(len(unigram_tokens_words)):

            #Appending it to the token_words in lower case
            token_words.append(unigram_tokens_words[j].lower())

    #returning the list
    return token_words
```

In [10]:
```
'''
generate_top200_bigrams is an user defined function that takes in two arguments
token_words --> A list of tokenized words
day_uni_words --> A dictionary with key as date and value as tokens in the corresponding date

This function will generate top 200 bi-grams.
BigramAssocMeasures --> From this method, pointwise mutual information (pmi) will give us
the top n bigrams given inside the argument

BigramCollocationFinder --> Find the bigrams from the words passed in arguments

MWETokenizer is applied to retokenize the tokens

This function returns the list of unique tokens
'''

#User defined function
def generate_top200_bigrams(token_words,day_uni_words):

    #For finding the top PMI
    bigram_measures = nltk.collocations.BigramAssocMeasures()

    #Collocations on the done on the token words
    bigram_finder = nltk.collocations.BigramCollocationFinder.from_words(token_words)

    #top 200 bigrams are found
    bigram_200 = bigram_finder.nbest(bigram_measures.pmi, 200)

    #Tokenises the tokenised text
    mwetokenizer = MWETokenizer(bigram_200)

    #creates a dictionary (key value pair) with
    #key as the date and value as the individual tokens in the date
    colloc_patents =  dict((pid, mwetokenizer.tokenize(patent)) for pid,patent in day_uni_words.items())

    all_words_colloc = list(chain.from_iterable(colloc_patents.values()))

    #Converting to set and then back to list to avoid duplicate tokens
    colloc_voc = list(set(all_words_colloc))

    #returning the list of unique tokens
    return colloc_voc
```

In [11]:
```
'''
find_context_dependent_words is an user defined function that takes in one list as its argument
convert to set to remove duplicates and then back to list for iteration purpose.

If the words count is greater than 60 or if the words count is less than 5, then words are appended to the list

This list containing all context dependent words is then returned
'''

#User defined function
def find_context_dependent_words(temp_list):

    #to avoid duplicates
    unique_words = list(set(temp_list))

    #Creating a new list to store the context dependent words
    context_dependent = []

    #Iterating through the list to compare and add the context dependent words
    for i in range(len(unique_words)):

        #if the count of the words are greater than 60 or less than 5(for rare tokens)
        if(temp_list.count(unique_words[i]) > 60 or temp_list.count(unique_words[i]) < 5):

            #appending it to the list
            context_dependent.append(unique_words[i])

    #Returning the list of context dependent words
    return context_dependent
```

In [12]:
```
'''
remove_unwanted_words is a user defined function that takes in three lists as arguments
colloc_voc --> list of collocated wordS( actual tokens)
context_dependent--> context dependent stop words
stop_words--> context independent stop words

All three has been converted to set to that set operations can be performed.

Since we have to remove the context dependent words from the colloc_voc, just a minus would perform the
action much faster. Like wise, operations has been performed and

1. Context dependent
2. Context independent
3. words of length less than 3 has been removed.
'''

#user defined function
def remove_unwanted_words(colloc_voc,context_dependent,stop_words):

    #converting to set
    set_coll = set(colloc_voc)

    #converting to set
    set_cont = set(context_dependent)

    #converting to set
    set_stop = set(stop_words)

    #Subtracting collocated from the context dependent.
    #what is there in collocated set and not in context dependent set
    temp_1 = set_coll - set_cont

    #there in (set_coll - set_cont) and not in stop words
    temp_2 = temp_1 - set_stop

    #Conveting to back to list for iteration purpose
    temp_2 = list(temp_2)

    #if the length of the tokens are greater than or equal to 3, include them
    proper_tokens = [i for i in temp_2 if len(i)>=3]

    #return the proper tokens generated after
    return proper_tokens
```

In [13]:
```
'''
generate_vocab is an user defined function that generates the vocabulary text file. Takes in the stemmed words
as input
'''

#user defined function
def generate_vocab(vocab_stemmed):

    #initialising count to 0 for indexing purpose
    count=0

    #opening the file
    file = open('31021301_vocab.txt','w',encoding = 'utf-8')

    #For the range of length of the text
    for i in vocab_stemmed:

        #write te word with it's count
        file.write(i + ':'+ str(count) + '\n')

        #increamenting the count
        count+=1

    #closing the file
    file.close()
```

In [25]:
```
'''
generate_countVec is a user defined function that takes in a dictionary as input.

For each sheet, set of tokens is taken and then updated in the vocab dict.
tokens are appended to the vocab list to perform the count operation to get the context dependent
stop words.
Making sure than _ is not there in the text(this is for bigrams seperated with an _)

Once done, context independent stop words, context dependent stop words, words in length less than 3 and bigrams are
removed and then stemmed if it is not a bigram, else appending it directly.

This function will return the dictionary with key as its date and value as its corresponding tokens reqd.
'''

#user defined function
def generate_countVec(day_uni_words):

    #dummy variable to extract only the tokens
    vocab_dict = {}

    #For the range of the lenght of keys
    for i in range(len(day_uni_words)):

        #set to avoid duplicates
        unique_tokens = list(day_uni_words[given_excel.sheet_names[i]])

        #with the sheet name as key and the unique tokens as value
        #updating it to the dictionary
        vocab_dict.update({given_excel.sheet_names[i]:unique_tokens})

    #Extracting onlt the values and storing it in list
    vocab_list = [list(value) for value in vocab_dict.values()]

    #to create a single list from list of lists
    vocab_list = sum(vocab_list,[])

    #ingoring the _(bigram words) and adding only uni tokens
    vocab_list = [a for a in vocab_list if "_" not in a]

    #counting frequency of words to get to know the context dependent words
    freq_vocab_list = FreqDist(vocab_list)

    #appending the context dependent words to the variable
    context_dep_vec = [key for key,value in freq_vocab_list.items() if value > 60 or value > 5]

    #Converting it to set and then back to list to avoid duplicates
    context_dep_vec = list(set(context_dep_vec))

    #Keeiping only the required tokens (i,e) removing dependent words, independent words,
    #words less than length 2, stemming the words
    for i in day_uni_words.keys():
        day_uni_words[i] = [x for x in day_uni_words[i] if x not in context_dep_vec]
        day_uni_words[i] = [x for x in day_uni_words[i] if len(x) >= 3]
        day_uni_words[i] = [x for x in day_uni_words[i] if x not in stop_words]
        day_uni_words[i] = [stemmer.stem(x) if "_" not in x else x for x in day_uni_words[i]]

    #returning the dictionary
    return day_uni_words
```

In [15]:
```
'''
generate_vecdoc is an user defined function that takes in 3 arguments
day_uni_words_final --> Final dictionary received from the above function
given_excel --> Excel sheet to iterate through names of the sheet
vocab_stemmed --> vocab stemmed to check if the word is present in the vocab file
'''

#user defined function
def generate_vecdoc(day_uni_words_final,given_excel,vocab_stemmed):

    #Initialising the count vectorizer
    vectorizer = CountVectorizer(analyzer = "word")

    #Performing fit and transformation of the vectorizer on the tokens in the dictionary file
    data_features = vectorizer.fit_transform([' '.join(value) for value in day_uni_words_final.values()])

    #Getting the feature names
    vocab = vectorizer.get_feature_names()

    #opening the document
    count_vec_file = open('31021301_countVec.txt','a')

        #for the range of length of the excel sheets
        for i in range(len(given_excel.sheet_names)):

            #append the sheet name
            count_vec_file.write(given_excel.sheet_names[i] + ",")

            #Zipping the vocab and count
            for k, l in zip(vocab, data_features.toarray()[i]):

                #if the count of the current word is greater than 0
                if(l>0):

                    #if the word is present in the vocabulary
                    if(k in vocab_stemmed):

                        #append it to the file
                        count_vec_file.write(str(vocab_stemmed.index(k))+":"+str(l)+",")

            #writing a new line
            count_vec_file.write("\n")

    #closing the file
    count_vec_file.close()
```

2.6 Function call and uni/bi gram file creation

- In this section, the above defined functions are called and the word documents are generated for top 100 unigram and bigram.
- Also for creating the vocab text, words per day is stored.

In [16]:
```
#Initialsing variables to store values in a key value pair
date_dict_words ={}
all_words = []
day_uni_words = {}

#For the range of length of the sheets
for i in range(len(given_excel.sheet_names)):

    #Reading the excel file in each sheet per the iteration
    per_sheet = pd.read_excel(r'31021301.xlsx',sheet_name=given_excel.sheet_names[i])

    #Removing all empty rows
    per_sheet = per_sheet.dropna(0,how="all")

    #Removing all empty column
    per_sheet = per_sheet.dropna(1,how="all")

    #Ignore the 0th index since 0th index has got the column name
    per_sheet = per_sheet.iloc[1:]

    #Manually declaring the column name to avoid inconsistency in the data frame column name
    per_sheet.columns=['text','id','created_at']

    #reset the index
    per_sheet = per_sheet.reset_index(drop = True)

    #Index name is set to null
    per_sheet.iloc[0].index.name = ''

    #Storing the contents of the column text into a list for iterating purpose
    only_text = list(per_sheet['text'])

    #Calling the function to generate unigram tokens per day
    tokens_gen = generate_unitoken_perday(only_text)

    #Storing the values of only english tweets
    all_words.append(tokens_gen[0])

    #Storing the values of tokenised words per day
    ind_uni_words = tokens_gen[1]

    #Converting to set and then to list to remove duplicates
    unique_per_day = list(set(ind_uni_words))

    #Adding every values to a key value pair so as to generate Vocab text and count vector
    date_dict_words[given_excel.sheet_names[i]] = unique_per_day

    #For bigram generation
    eng_only_uni = tokens_gen[0]

    #Function call to retrive tokens
    ind_bi_words = generate_bitoken_perday(eng_only_uni)
    day_uni_words[given_excel.sheet_names[i]] = ind_bi_words

    #Stemming is done on the unigram words
    vocab_stemmed = []
    for a in ind_uni_words:
            vocab_stemmed.append(stemmer.stem(a))

    #Function call to generate top 100 unigrams for the day
    generate_bigram(ind_bi_words,given_excel.sheet_names[i])

    #Function call to generate top 100 bigrams for the day
    generate_unigram(vocab_stemmed,given_excel.sheet_names[i])
```

```
<ipython-input-5-bc890be913b5>:22: DeprecationWarning: invalid escape sequence '\ '
  temp = only_text[i].encode('ascii','ignore').decode('unicode-escape')
<ipython-input-5-bc890be913b5>:22: DeprecationWarning: invalid escape sequence '\_'
  temp = only_text[i].encode('ascii','ignore').decode('unicode-escape')
<ipython-input-5-bc890be913b5>:22: DeprecationWarning: invalid escape sequence '\l'
  temp = only_text[i].encode('ascii','ignore').decode('unicode-escape')
<ipython-input-5-bc890be913b5>:22: DeprecationWarning: invalid escape sequence '\#'
  temp = only_text[i].encode('ascii','ignore').decode('unicode-escape')
```

2.7 Generating Vocab Text

- In this section generation of Vocabulary text file is done. The process for generating the vocab file is as follows:
  - Since every day's token is stored as a list from the above 2.6 section (all_words), merging the list of list to a single list which contains all english tweets alone.
  - Tokenization is done on the tweets and seperate tokens are extracted based on the given regular expression.
  - For finding top 200 bigrams, BigramCollocationFinder and PMI measure is used.
  - MWETokenizer is applied on each day's token to extract the collocated words.
  - Context dependent stop words are found based on the given criteria and removed.

```
+ If the words has been repeated for more than 60 days.

+ Rare tokens : If the words has been repeated for less than 5 days.
```

- Context independent stop words are imported from the text file provided and removed.
- Stemming is done using porter stemmer.
- Generated Vocab list and written it to a file.

In [17]:
```
#Converting the list of list to a single list
sep_tweets = [val for sublist in all_words for val in sublist]
```

In [18]:
```
#Function call generate_raw_tokens(sep_tweets) with the list created above to
#get the tokenised words
token_words = generate_raw_tokens(sep_tweets)
```

In [19]:
```
#Function call generate_top200_bigrams(token_words,day_uni_words)
#to get the collocated words
colloc_voc = generate_top200_bigrams(token_words,day_uni_words)
print(len(colloc_voc))
```

```
202318
```

In [20]:
```
#From dictionary of the bi_grams (since no constraints are applied)
#extracting all the values and storing each value as a single list
temp_list = [item for sublist in date_dict_words.values() for item in sublist]

#Function call to get the context dependent text words
context_dependent = find_context_dependent_words(temp_list)
```

```
201328
```

In [21]:
```
#function call to remove all the unwanted words and
#to get the proper tokens
proper_tokens = remove_unwanted_words(colloc_voc,context_dependent,stop_words)
```

In [22]:
```
#Porter stemmer is used to stem the video.

vocab_stemmed = []

#for all proper tokens
for i in proper_tokens:

    #if the token contains _ append bigrams
    if '_' in i:
        vocab_stemmed.append(i)

    #If there are uni words, stem and append
    else:
        vocab_stemmed.append(stemmer.stem(i))

#Creating a set of unique stems
vocab_stemmed = set(vocab_stemmed)

#converting it back to list and sorting it
vocab_stemmed = sorted(list(vocab_stemmed))

#Function call to generate bigrams
generate_vocab(vocab_stemmed)
```

2.8 Generate Count Vector

- In this section, count vector has been generated.
- To achieve this task, a dictionary with key as "day" and value as it's "tokens" are used.
- Process of again finding context dependent stop words, words of length less than 3, stemming, context independent words.
- With the use of package "CountVectorizer", count vectors are created.

In [23]:
```
#Function call to generate vector vocabulary and this function will return
#a dictionary with proper tokens required to create a count vector document
day_uni_words_final = generate_countVec(day_uni_words)
```

In [24]:
```
#Function call to generate count vector document
generate_vecdoc(day_uni_words_final,given_excel,vocab_stemmed)
```

2.9 References:

1. Solutions from Lab 4 Tasks
2. Solutions from Lab 5 Tasks
3. Python Documentation, Iter tools. Retrived from https://docs.python.org/2/library/itertools.html (https://docs.python.org/2/library/itertools.html)
4. jonrsharpe , Appending list of lists to a single list.Retrived from, https://stackoverflow.com/questions/33541947/what-does-the-built-in-function-sum-do-with-sumlist/33542054 (https://stackoverflow.com/questions/33541947/what-does-the-built-in-function-sum-do-with-sumlist/33542054)