First name: Rashmi Rao
last name: Raghavendra
NJIT UCID: RR683
email address: rr683@njit.edu
Option Number: 4

Text Mining System

Contents

Introduction

What is Text Mining?

Text mining is an artificial intelligence (AI) technology that uses natural language processing (NLP) to transform the free text in documents and databases into normalized, structured data suitable for analysis

Why Text Mining?

Text mining enables to analyze massive amounts of information quickly.

It can be used to make large quantities of unstructured data accessible and useful by extracting useful information and knowledge hidden in text content and revealing patterns, trends and insight in large amounts of information

Quoting Example:

If there are thousands of documents and if each document consists of thousands of pages, then the most efficient way to understand the document is by going through the keywords. This is when Text Mining comes into picture. Text Mining helps to extract the keywords from the documents without the hassle of going through each and every word.

Requirements

1. The coding is done on a Spyder platform, Spyder must be pre-installed or can also be done using Jupyter Notebook
2. CSV is required, input files are in CSV formats.
3. Standard Core NLP installed (Package: Stanza)
4. NLTK installed with packages (stopwords, WordNetLemmatizer, word_tokenize)
5. Apriori Package installed

Dataset: Reuters-21578 dataset

The data is extracted from –

http://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection

The Reuters-21578 collection is distributed in 22 files. Each of the first 21 files (reut2-000.sgm through reut2-020.sgm) contain 1000 documents, while the last (reut2-021.sgm) contains 578 documents. The files are in SGML format. There are several tags in each of the .sgm file. In this project, we have mainly focused on the contents inside the <BODY/> tag. These contents inside the body tag help us extract keywords mainly because of the description of the documents.

Tools

Stanford's Core NLP – **Stanza**

http://nlp.stanford.edu/software/corenlp.shtml

Stanza is a Python natural language analysis package. It contains tools, which can be used in a pipeline, to convert a string containing human language text into lists of sentences and words, to generate base forms of those words, their parts of speech and morphological features, to give a syntactic structure dependency parse, and to recognize named entities.

Stanza is built with highly accurate neural network components that also enable efficient training and evaluation with your own annotated data. The modules are built on top of the PyTorch library. You will get much faster performance if you run this system on a GPU-enabled machine.

To summarize, Stanza features:

- Native Python implementation requiring minimal efforts to set up
- Full neural network pipeline for robust text analytics, including tokenization, multi-word token (MWT) expansion, lemmatization, part-of-speech (POS) and morphological features tagging, dependency parsing, and named entity recognition
- Pretrained neural models supporting 66 (human) languages
- A stable, officially maintained Python interface to CoreNLP

Natural language Toolkit (NLTK)

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

NLTK Features

- NLTK consists of the most common algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition.
- NLTK helps the computer to analysis, preprocess, and understand the written text.

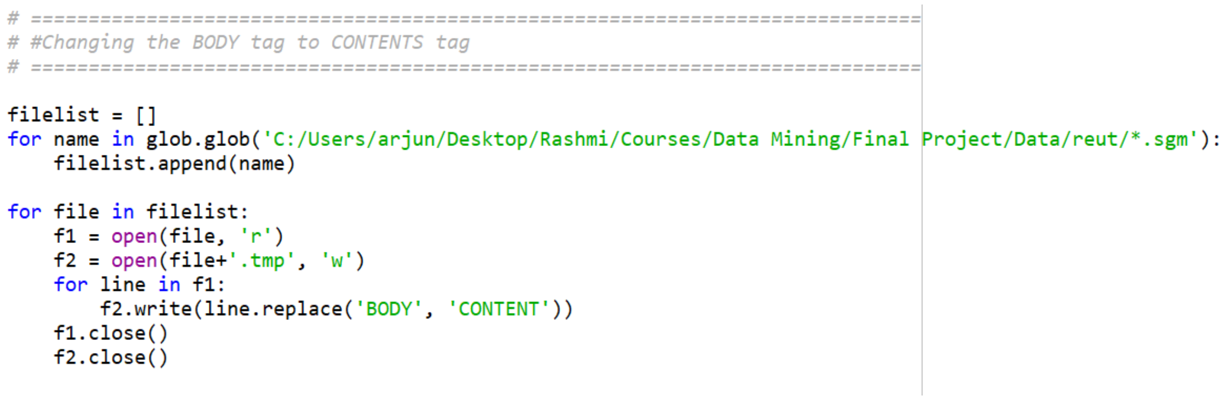
Text Mining Implementation

The implementation is done in several stages.

Stage 1: Changing the tags

In this stage, we make use of Glob library to extract the paths of all the .sgm files within the folder.

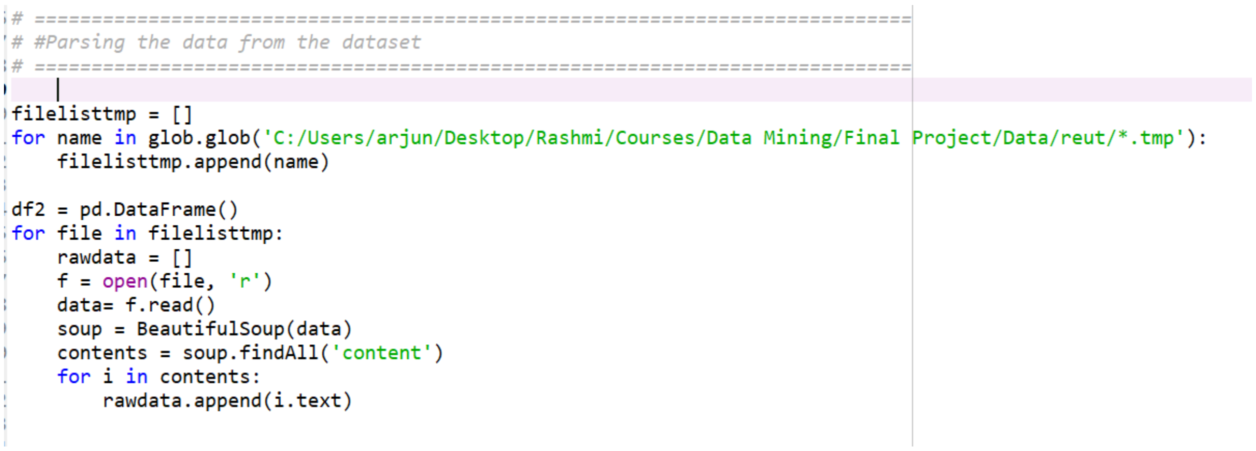
Here we change the <BODY> tag to <CONTENTS> tag because the extraction is not possible if the contents are present under body tag. We change the tag and create a new .tmp files for all the .sgm files.



```
# =============================================================================
# #Changing the BODY tag to CONTENTS tag
# =============================================================================

filelist = []
for name in glob.glob('C:/Users/arjun/Desktop/Rashmi/Courses/Data Mining/Final Project/Data/reut/*.sgm'):
    filelist.append(name)

for file in filelist:
    f1 = open(file, 'r')
    f2 = open(file+'.tmp', 'w')
    for line in f1:
        f2.write(line.replace('BODY', 'CONTENT'))
    f1.close()
    f2.close()
```

Stage 2: Parsing the data from the temp files

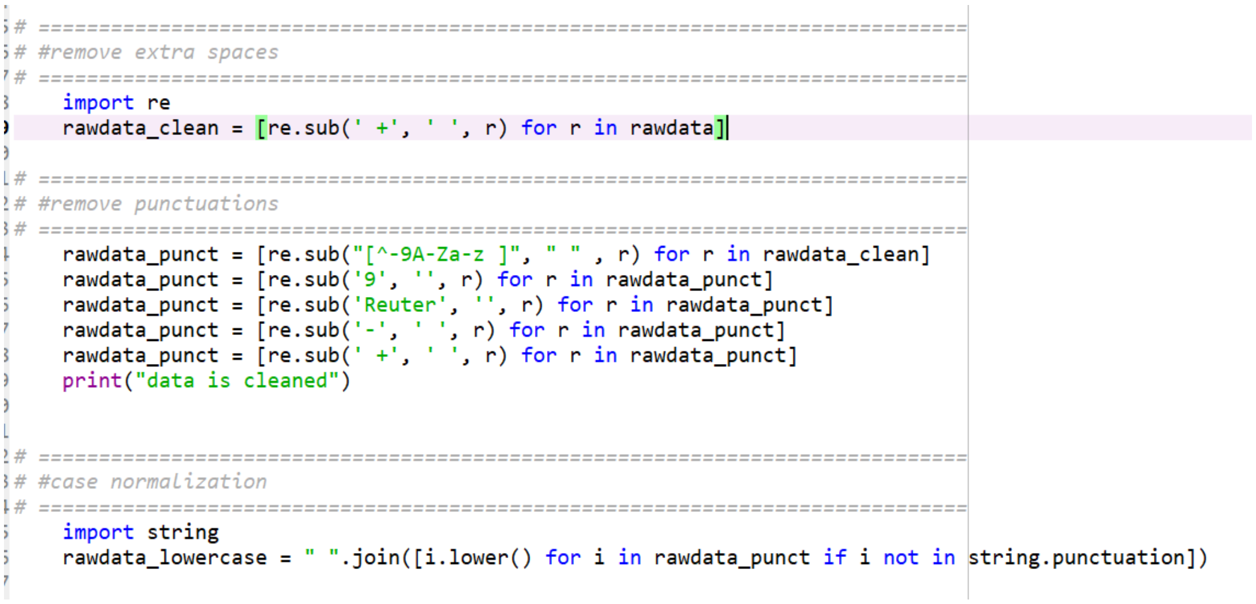
In this stage, we extract the contents which is present within the contents tag from the .tmp files.



```
# ==========================================================================
# #Parsing the data from the dataset
# ==========================================================================
    
filelisttmp = []
for name in glob.glob('C:/Users/arjun/Desktop/Rashmi/Courses/Data Mining/Final Project/Data/reut/*.tmp'):
    filelisttmp.append(name)

df2 = pd.DataFrame()
for file in filelisttmp:
    rawdata = []
    f = open(file, 'r')
    data= f.read()
    soup = BeautifulSoup(data)
    contents = soup.findAll('content')
    for i in contents:
        rawdata.append(i.text)
```

Stage 3: Data Cleansing

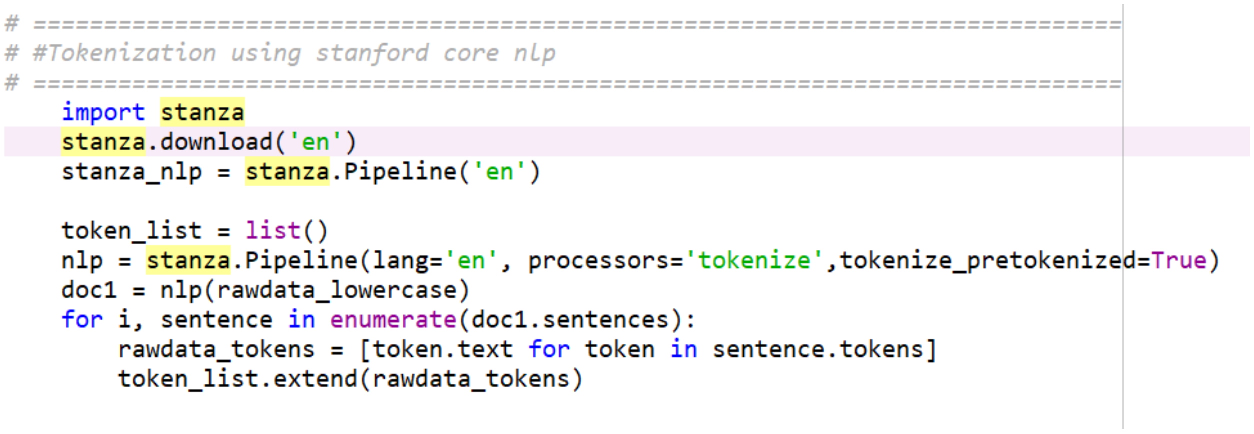
- get rid of unnecessary spaces between the words
- remove punctuations, special characters and numbers
- convert all the strings to lowercase



```
# ==========================================================================
# #remove extra spaces
# ==========================================================================
    import re
    rawdata_clean = [re.sub(' +', ' ', r) for r in rawdata]

# ==========================================================================
# #remove punctuations
# ==========================================================================
    rawdata_punct = [re.sub("[^-9A-Za-z ]", " " , r) for r in rawdata_clean]
    rawdata_punct = [re.sub('9', '', r) for r in rawdata_punct]
    rawdata_punct = [re.sub('Reuter', '', r) for r in rawdata_punct]
    rawdata_punct = [re.sub('-', ' ', r) for r in rawdata_punct]
    rawdata_punct = [re.sub(' +', ' ', r) for r in rawdata_punct]
    print("data is cleaned")


# ==========================================================================
# #case normalization
# ==========================================================================
    import string
    rawdata_lowercase = " ".join([i.lower() for i in rawdata_punct if i not in string.punctuation])
```

Stage 4: Tokenization

Tokenization is the process of turning text into tokens. Here, we use Stanford's core NLP tokenizer for tokenization. For instance, the sentence "Marie was born in Paris" would be tokenized as the list "Marie", "was", "born", "in", "Paris", ".". CoreNLP splits texts into tokens with an elaborate collection of rules.



```
# ==============================================================================
# #Tokenization using stanford core nlp
# ==============================================================================
    import stanza
    stanza.download('en')
    stanza_nlp = stanza.Pipeline('en')

    token_list = list()
    nlp = stanza.Pipeline(lang='en', processors='tokenize',tokenize_pretokenized=True)
    doc1 = nlp(rawdata_lowercase)
    for i, sentence in enumerate(doc1.sentences):
        rawdata_tokens = [token.text for token in sentence.tokens]
        token_list.extend(rawdata_tokens)
```

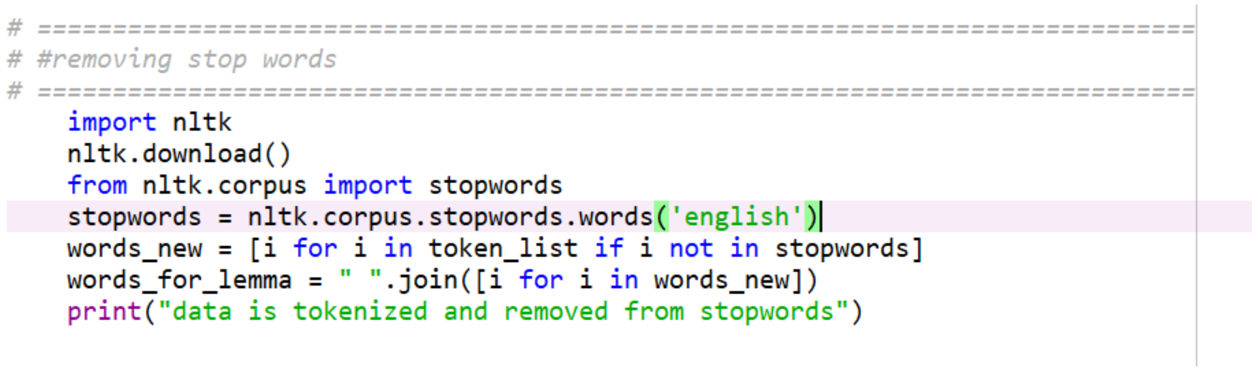
Stage 5: Removing Stop Words

The process of converting data to something a computer can understand is referred to as pre-processing. One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

Stop Words: A stop word is a commonly used word (such as "the", "a", "an", "in") that does not add any valuable information to the sentences.

We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK (Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages.

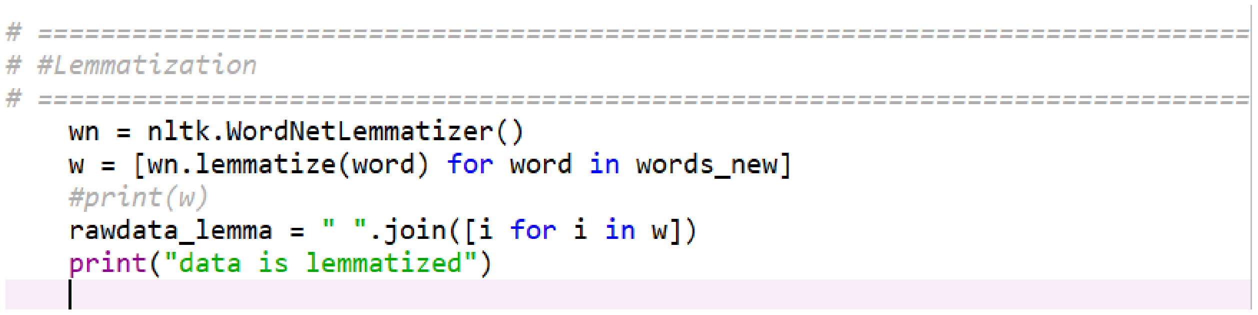
In this stage, we download NLTKs English stopwords package. We check if our dataset contains these stopwords. If yes, then we remove them from the dataset.



```
# ==============================================================================
# #removing stop words
# ==============================================================================
    import nltk
    nltk.download()
    from nltk.corpus import stopwords
    stopwords = nltk.corpus.stopwords.words('english')
    words_new = [i for i in token_list if i not in stopwords]
    words_for_lemma = " ".join([i for i in words_new])
    print("data is tokenized and removed from stopwords")
```

Stage 6: Lemmatization

Lemmatization maps a word to its lemma (dictionary form). For instance, the word “was” is mapped to the word “be”



```
# ==============================================================================
# #Lemmatization
# ==============================================================================
    wn = nltk.WordNetLemmatizer()
    w = [wn.lemmatize(word) for word in words_new]
    #print(w)
    rawdata_lemma = " ".join([i for i in w])
    print("data is lemmatized")
```

Stage 7: TF-IDF

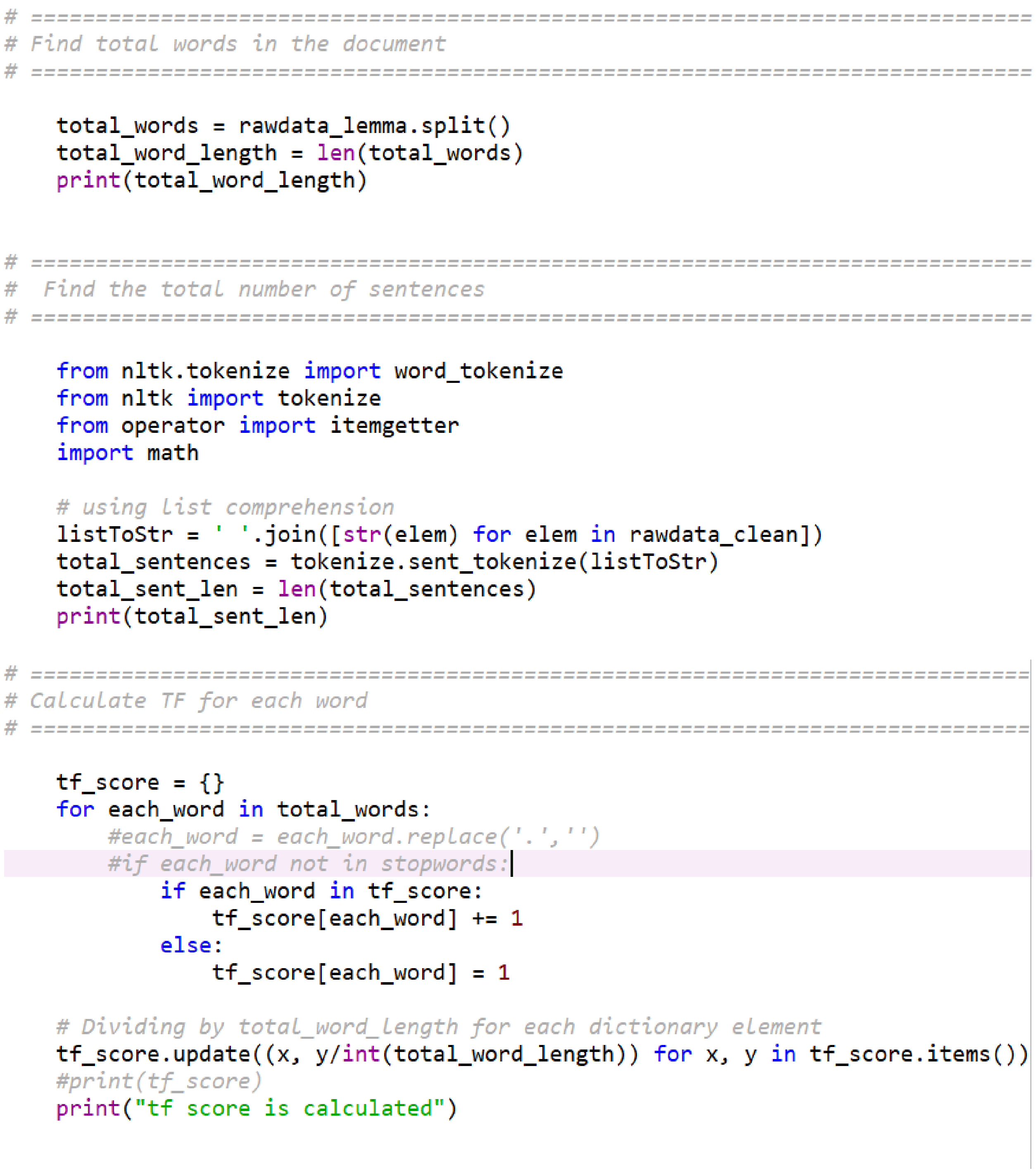
Term Frequency (TF) – How frequently a term occurs in a text. It is measured as the number of times a term t appears in the text / Total number of words in the document

Inverse Document Frequency (IDF) – How important a word is in a document. It is measured as log (total number of sentences / Number of sentences with term t)

TF-IDF – Words’ importance is measure by this score. It is measured as TF * IDF

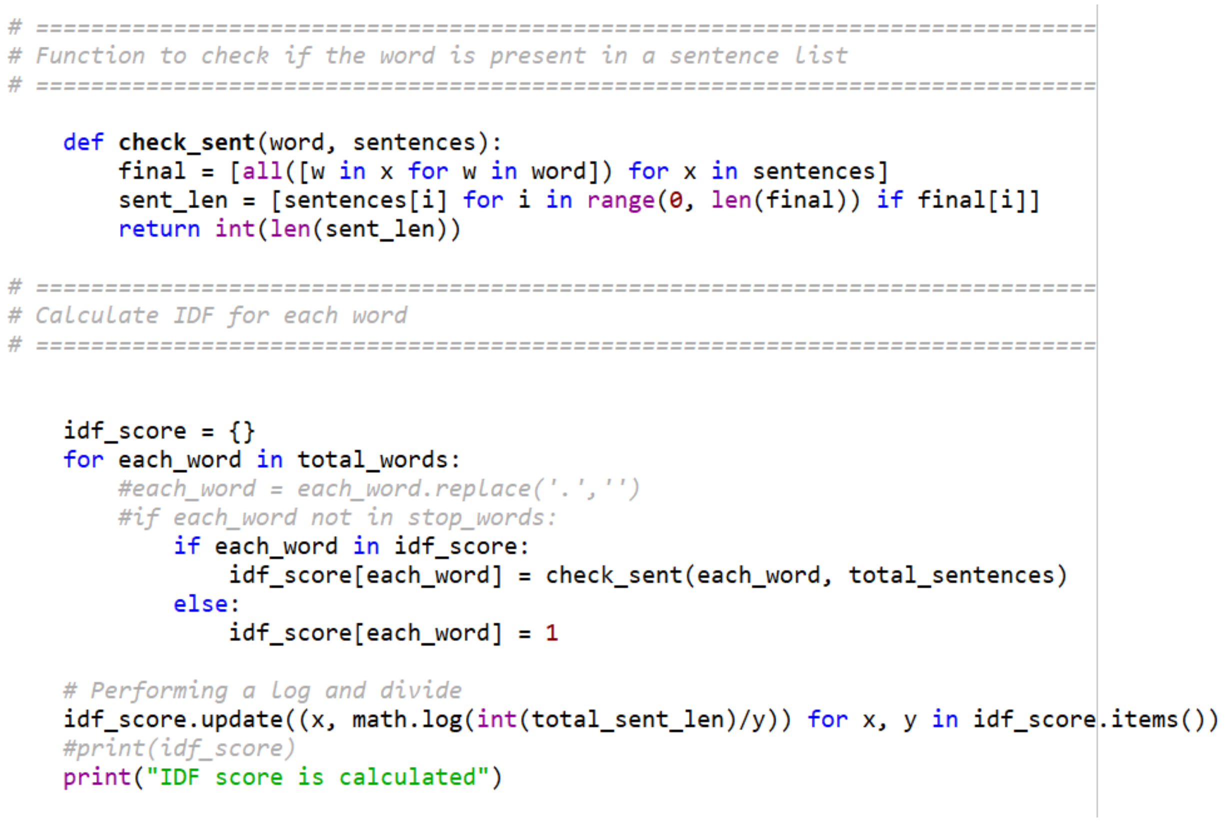
In this stage, we calculate total number of words and sentences to find Term Frequency

TF: We will begin by calculating the word count for each non-stop words and finally divide each element by the result of total word length



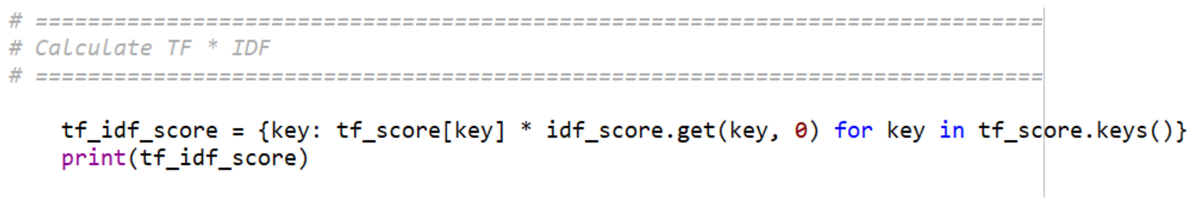
```
# =============================================================================
# Find total words in the document
# =============================================================================

    total_words = rawdata_lemma.split()
    total_word_length = len(total_words)
    print(total_word_length)


# =============================================================================
#  Find the total number of sentences
# =============================================================================

    from nltk.tokenize import word_tokenize
    from nltk import tokenize
    from operator import itemgetter
    import math

    # using list comprehension
    listToStr = ' '.join([str(elem) for elem in rawdata_clean])
    total_sentences = tokenize.sent_tokenize(listToStr)
    total_sent_len = len(total_sentences)
    print(total_sent_len)

# =============================================================================
# Calculate TF for each word
# =============================================================================

    tf_score = {}
    for each_word in total_words:
        #each_word = each_word.replace('.','')
        #if each_word not in stopwords:
            if each_word in tf_score:
                tf_score[each_word] += 1
            else:
                tf_score[each_word] = 1

    # Dividing by total_word_length for each dictionary element
    tf_score.update((x, y/int(total_word_length)) for x, y in tf_score.items())
    #print(tf_score)
    print("tf score is calculated")
```

IDF: We wrote a function – (check_sent) to iterate the non-stop word and store the result for Inverse Document Frequency



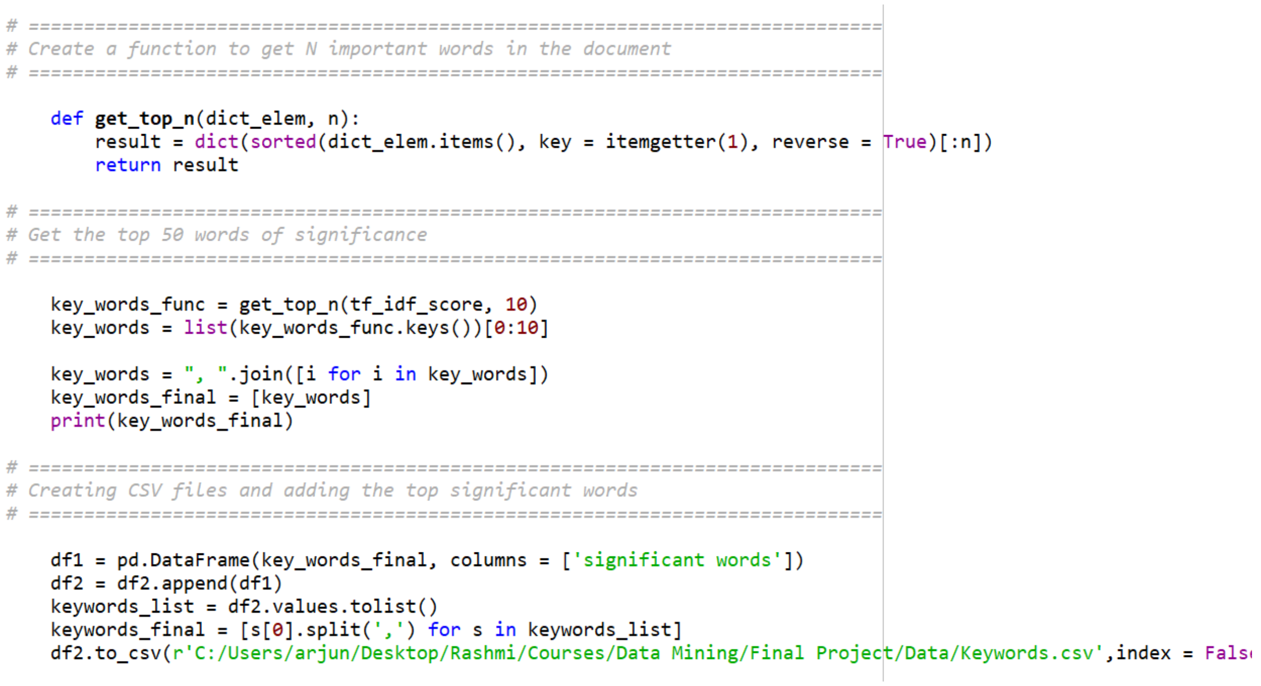
```
# ============================================================================
# Function to check if the word is present in a sentence list
# ============================================================================

    def check_sent(word, sentences):
        final = [all([w in x for w in word]) for x in sentences]
        sent_len = [sentences[i] for i in range(0, len(final)) if final[i]]
        return int(len(sent_len))

# ============================================================================
# Calculate IDF for each word
# ============================================================================


    idf_score = {}
    for each_word in total_words:
        #each_word = each_word.replace('.','')
        #if each_word not in stop_words:
            if each_word in idf_score:
                idf_score[each_word] = check_sent(each_word, total_sentences)
            else:
                idf_score[each_word] = 1

    # Performing a log and divide
    idf_score.update((x, math.log(int(total_sent_len)/y)) for x, y in idf_score.items())
    #print(idf_score)
    print("IDF score is calculated")
```

Calculate *TF*IDF*



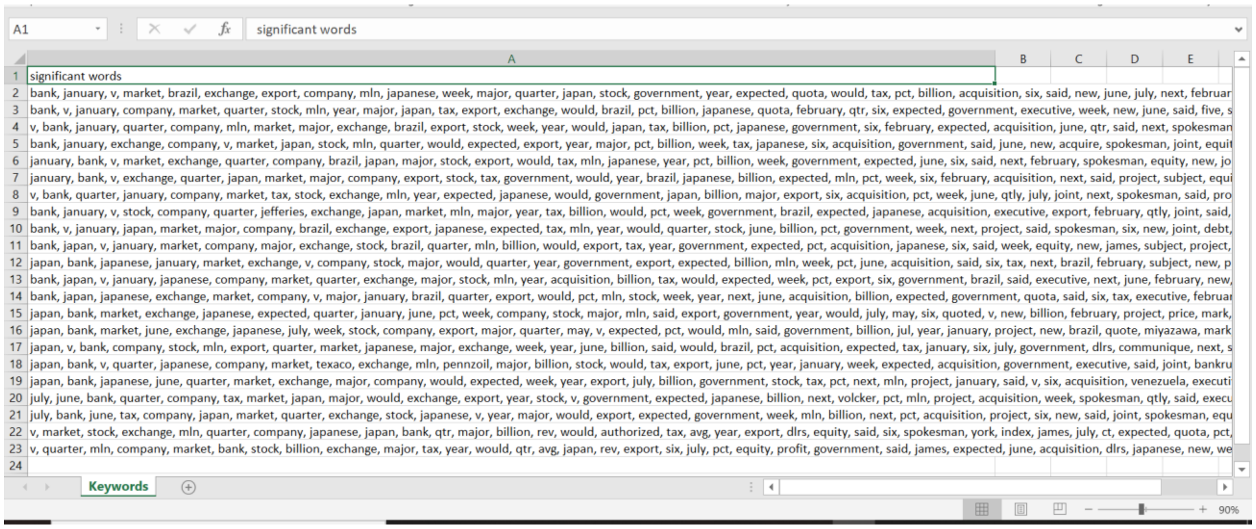
```
# ============================================================================
# Calculate TF * IDF
# ============================================================================

    tf_idf_score = {key: tf_score[key] * idf_score.get(key, 0) for key in tf_score.keys()}
    print(tf_idf_score)
```

Create a function to get N important words in the document and extract 50 keywords (most significant words) from each of the document. We add these words to csv files.



```
# ============================================================================
# Create a function to get N important words in the document
# ============================================================================

    def get_top_n(dict_elem, n):
        result = dict(sorted(dict_elem.items(), key = itemgetter(1), reverse = True)[:n])
        return result

# ============================================================================
# Get the top 50 words of significance
# ============================================================================

    key_words_func = get_top_n(tf_idf_score, 10)
    key_words = list(key_words_func.keys())[0:10]

    key_words = ", ".join([i for i in key_words])
    key_words_final = [key_words]
    print(key_words_final)

# ============================================================================
# Creating CSV files and adding the top significant words
# ============================================================================

    df1 = pd.DataFrame(key_words_final, columns = ['significant words'])
    df2 = df2.append(df1)
    keywords_list = df2.values.tolist()
    keywords_final = [s[0].split(',') for s in keywords_list]
    df2.to_csv(r'C:/Users/arjun/Desktop/Rashmi/Courses/Data Mining/Final Project/Data/Keywords.csv',index = Fals
```

Keywords.csv File: It consists of 50 keywords representing 22 .sgm files respectively.



A1 | significant words

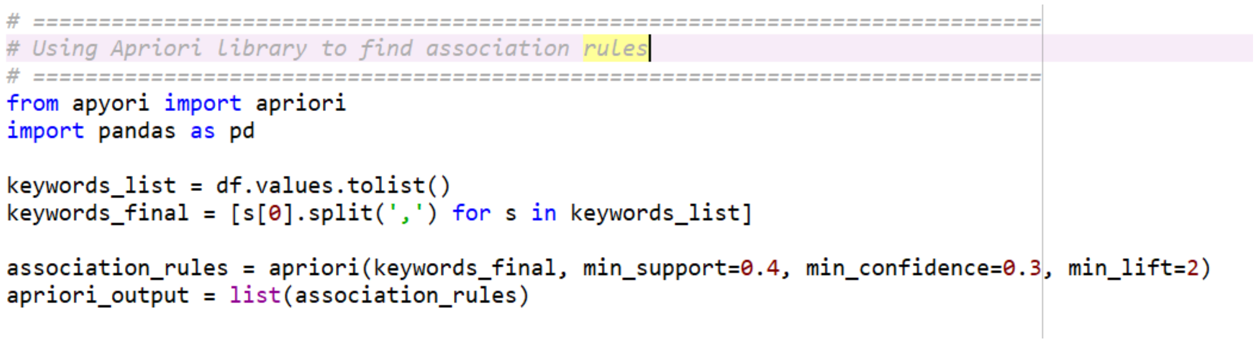
	A	B	C	D	E
1	significant words				
2	bank, january, v, market, brazil, exchange, export, company, mln, japanese, week, major, quarter, japan, stock, government, year, expected, quota, would, tax, pct, billion, acquisition, six, said, new, june, july, next, februar				
3	bank, v, january, company, market, quarter, stock, mln, year, major, japan, tax, export, exchange, would, brazil, pct, billion, japanese, quota, february, qtr, six, expected, government, executive, week, new, june, said, five, s				
4	v, bank, january, quarter, company, mln, market, major, exchange, brazil, export, stock, week, year, would, japan, tax, billion, pct, japanese, government, six, february, expected, acquisition, june, qtr, said, next, spokesman				
5	bank, january, exchange, company, v, market, japan, stock, mln, quarter, would, expected, export, year, major, pct, billion, week, tax, japanese, six, acquisition, government, said, june, new, acquire, spokesman, joint, equit				
6	january, bank, v, market, exchange, quarter, company, brazil, japan, major, stock, export, would, tax, mln, japanese, year, pct, billion, week, government, expected, june, six, said, next, february, spokesman, equity, new, jo				
7	january, bank, v, exchange, quarter, japan, market, major, company, export, stock, tax, government, would, year, brazil, japanese, billion, expected, mln, pct, week, six, february, acquisition, next, said, project, subject, equi				
8	v, bank, quarter, january, company, market, tax, stock, exchange, mln, year, expected, japanese, would, government, japan, billion, major, export, six, acquisition, pct, week, june, qtly, july, joint, next, spokesman, said, pro				
9	bank, january, v, stock, company, quarter, jefferies, exchange, japan, market, mln, major, year, tax, billion, would, pct, week, government, brazil, expected, japanese, acquisition, executive, export, february, qtly, joint, said,				
10	bank, v, january, japan, market, major, company, brazil, exchange, export, japanese, expected, tax, mln, year, would, quarter, stock, june, billion, pct, government, week, next, project, said, spokesman, six, new, joint, debt,				
11	bank, japan, v, january, market, company, major, exchange, stock, brazil, quarter, mln, billion, would, export, tax, year, government, expected, pct, acquisition, japanese, six, said, week, equity, new, james, subject, project,				
12	japan, bank, japanese, january, market, exchange, v, company, stock, major, would, quarter, year, government, export, expected, billion, mln, week, pct, june, acquisition, said, six, tax, next, brazil, february, subject, new, p				
13	bank, japan, v, january, japanese, company, market, quarter, exchange, major, stock, mln, year, acquisition, billion, tax, would, expected, week, pct, export, six, government, brazil, said, executive, next, june, february, new,				
14	bank, japan, japanese, exchange, market, company, v, major, january, brazil, quarter, export, would, pct, mln, stock, week, year, next, june, acquisition, billion, expected, government, quota, said, six, tax, executive, februar				
15	japan, bank, market, exchange, japanese, expected, quarter, january, june, pct, week, company, stock, major, mln, said, export, government, year, would, july, may, six, quoted, v, new, billion, february, project, price, mark,				
16	japan, bank, market, june, exchange, japanese, july, week, stock, company, export, major, quarter, may, v, expected, pct, would, mln, said, government, billion, jul, year, january, project, new, brazil, quote, miyazawa, mark				
17	japan, v, bank, company, stock, mln, export, quarter, market, japanese, major, exchange, week, year, june, billion, said, would, brazil, pct, acquisition, expected, tax, january, six, july, government, dlrs, communique, next, s				
18	japan, bank, v, quarter, japanese, company, market, texaco, exchange, mln, pennzoil, major, billion, stock, would, tax, export, june, pct, year, january, week, expected, acquisition, government, executive, said, joint, bankru				
19	japan, bank, japanese, june, quarter, market, exchange, major, company, would, expected, week, year, export, july, billion, government, stock, tax, pct, next, mln, project, january, said, v, six, acquisition, venezuela, executi				
20	july, june, bank, quarter, company, tax, market, japan, major, would, exchange, export, year, stock, v, government, expected, japanese, billion, next, volcker, pct, mln, project, acquisition, week, spokesman, qtly, said, execu				
21	july, bank, june, tax, company, japan, market, quarter, exchange, stock, japanese, v, year, major, would, export, expected, government, week, mln, billion, next, pct, acquisition, project, six, new, said, joint, spokesman, equ				
22	v, market, stock, exchange, mln, quarter, company, japanese, japan, bank, qtr, major, billion, rev, would, authorized, tax, avg, year, export, dlrs, equity, said, six, spokesman, york, index, james, july, ct, expected, quota, pct,				
23	v, quarter, mln, company, market, bank, stock, billion, exchange, major, tax, year, would, qtr, avg, japan, rev, export, six, july, pct, equity, profit, government, said, james, expected, june, acquisition, dlrs, japanese, new, we				
24					

Stage 8: Association rules between the keywords which represent each of these documents

Here we use two methods

a) Apriori Package from apyori
b) Apriori algorithm developed from scratch

Method A:



```
# ==============================================================================
# Using Apriori library to find association rules
# ==============================================================================
from apyori import apriori
import pandas as pd

keywords_list = df.values.tolist()
keywords_final = [s[0].split(',') for s in keywords_list]

association_rules = apriori(keywords_final, min_support=0.4, min_confidence=0.3, min_lift=2)
apriori_output = list(association_rules)
```

Method A Output 1:

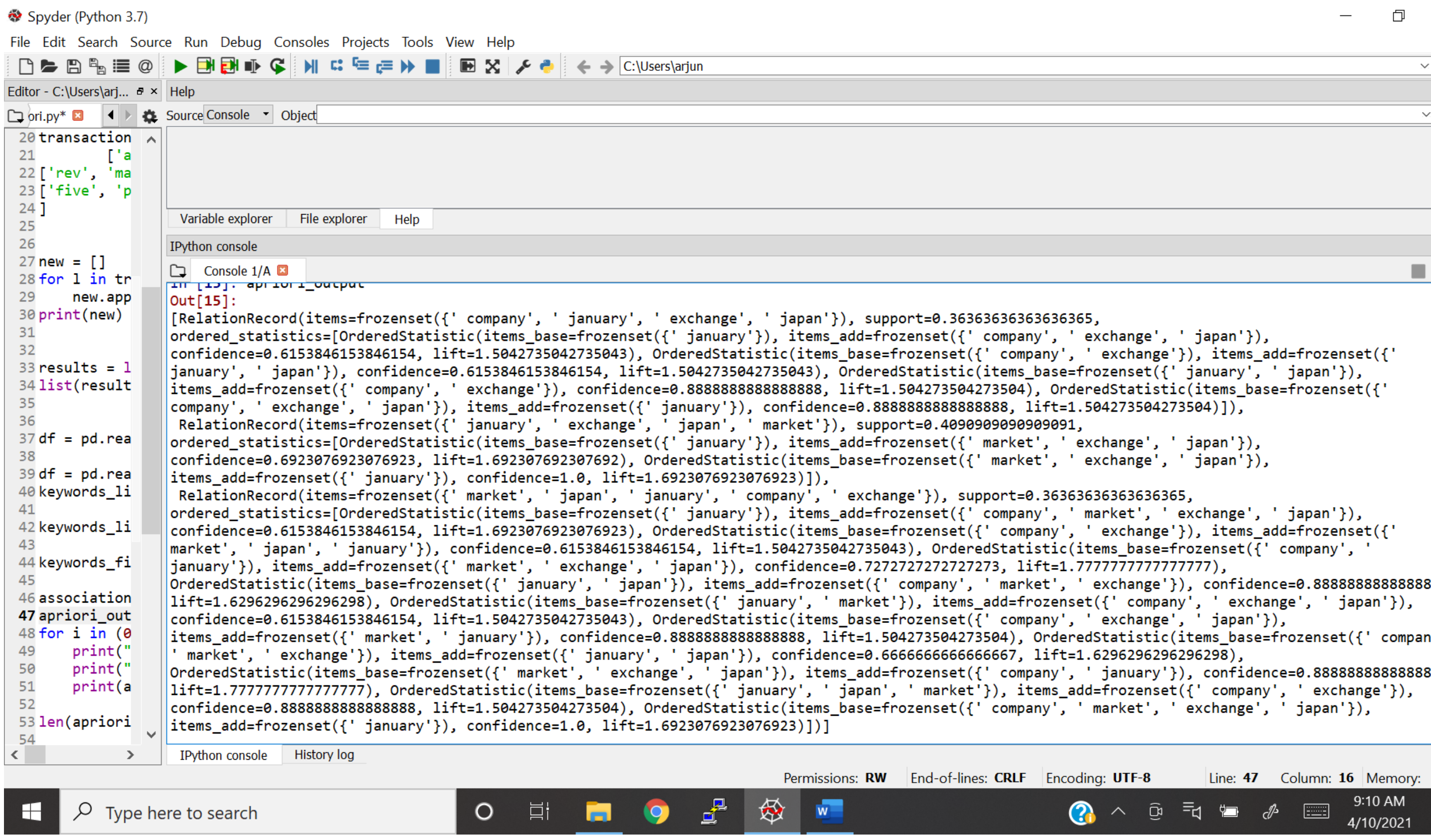
My laptop kept heating up and generated memory error for using 50 keywords for each document. Hence, I used eight key words to represent each document. Below is the output foe sending the 8 keywords of all the documents: Generating association rules between 50 documents represented by eight keywords each. I have also filtered it for various support and confidence levels.

The below output is generated for min_support = 0.4, min_confidence = 0.3, min_lift=2

```
output for : 0


RelationRecord(items=frozenset({' january', 'bank', ' company'}), support=0.45454545454545453,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'bank'}), items_add=frozenset({' january', ' company'}), confidence=1.0, lift=2.0),
OrderedStatistic(items_base=frozenset({' january', ' company'}), items_add=frozenset({'bank'}), confidence=0.9090909090909091, lift=2.0)])
output for : 1


RelationRecord(items=frozenset({' january', ' exchange', 'bank', ' company'}), support=0.4090909090909091,
ordered_statistics=[OrderedStatistic(items_base=frozenset({' january', ' company'}), items_add=frozenset({' exchange', 'bank'}),
confidence=0.8181818181818182, lift=2.0), OrderedStatistic(items_base=frozenset({' exchange', 'bank'}), items_add=frozenset({' january', ' company'}),
confidence=1.0, lift=2.0)])
output for : 2


RelationRecord(items=frozenset({' market', ' january', 'bank', ' company'}), support=0.45454545454545453,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'bank'}), items_add=frozenset({' market', ' january', ' company'}), confidence=1.0,
lift=2.0), OrderedStatistic(items_base=frozenset({' january', ' company'}), items_add=frozenset({' market', 'bank'}), confidence=0.9090909090909091,
lift=2.0), OrderedStatistic(items_base=frozenset({' market', 'bank'}), items_add=frozenset({' january', ' company'}), confidence=1.0, lift=2.0),
OrderedStatistic(items_base=frozenset({' market', ' january', ' company'}), items_add=frozenset({'bank'}), confidence=0.9090909090909091, lift=2.0)])
output for : 3

output for : 3


RelationRecord(items=frozenset({' market', ' company', ' january', ' exchange', 'bank'}), support=0.4090909090909091,
ordered_statistics=[OrderedStatistic(items_base=frozenset({' january', ' company'}), items_add=frozenset({' market', ' exchange', 'bank'}),
confidence=0.8181818181818182, lift=2.0), OrderedStatistic(items_base=frozenset({' exchange', 'bank'}), items_add=frozenset({' market', ' january', '
company'}), confidence=1.0, lift=2.0), OrderedStatistic(items_base=frozenset({' market', ' january', ' company'}), items_add=frozenset({' exchange',
'bank'}), confidence=0.8181818181818182, lift=2.0), OrderedStatistic(items_base=frozenset({' market', ' exchange', 'bank'}), items_add=frozenset({'
january', ' company'}), confidence=1.0, lift=2.0)])
```

Method A Output 2:

The below output is generated for min_support=0.3, min_confidence=0.1, min_lift=1.5



```
transaction
      ['a
['rev', 'ma
['five', 'p
]


new = []
for l in tr
    new.app
print(new)


results = l
list(result


df = pd.rea

df = pd.rea
keywords_li

keywords_li

keywords_fi

association
apriori_out
for i in (0
    print("
    print("
    print(a

len(apriori
```

```
Out[15]:
[RelationRecord(items=frozenset({' company', ' january', ' exchange', ' japan'}), support=0.36363636363636365,
ordered_statistics=[OrderedStatistic(items_base=frozenset({' january'}), items_add=frozenset({' company', ' exchange', ' japan'}),
confidence=0.6153846153846154, lift=1.5042735042735043), OrderedStatistic(items_base=frozenset({' company', ' exchange'}), items_add=frozenset({'
january', ' japan'}), confidence=0.6153846153846154, lift=1.5042735042735043), OrderedStatistic(items_base=frozenset({' january', ' japan'}),
items_add=frozenset({' company', ' exchange'}), confidence=0.8888888888888888, lift=1.504273504273504), OrderedStatistic(items_base=frozenset({'
company', ' exchange', ' japan'}), items_add=frozenset({' january'}), confidence=0.8888888888888888, lift=1.504273504273504)]),
 RelationRecord(items=frozenset({' january', ' exchange', ' japan', ' market'}), support=0.4090909090909091,
ordered_statistics=[OrderedStatistic(items_base=frozenset({' january'}), items_add=frozenset({' market', ' exchange', ' japan'}),
confidence=0.6923076923076923, lift=1.692307692307692), OrderedStatistic(items_base=frozenset({' market', ' exchange', ' japan'}),
items_add=frozenset({' january'}), confidence=1.0, lift=1.6923076923076923)]),
 RelationRecord(items=frozenset({' market', ' japan', ' january', ' company', ' exchange'}), support=0.36363636363636365,
ordered_statistics=[OrderedStatistic(items_base=frozenset({' january'}), items_add=frozenset({' company', ' market', ' exchange', ' japan'}),
confidence=0.6153846153846154, lift=1.6923076923076923), OrderedStatistic(items_base=frozenset({' company', ' exchange'}), items_add=frozenset({'
market', ' japan', ' january'}), confidence=0.6153846153846154, lift=1.5042735042735043), OrderedStatistic(items_base=frozenset({' company', '
january'}), items_add=frozenset({' market', ' exchange', ' japan'}), confidence=0.7272727272727273, lift=1.7777777777777777),
OrderedStatistic(items_base=frozenset({' january', ' japan'}), items_add=frozenset({' company', ' market', ' exchange'}), confidence=0.8888888888888
lift=1.6296296296296298), OrderedStatistic(items_base=frozenset({' january', ' market'}), items_add=frozenset({' company', ' exchange', ' japan'}),
confidence=0.6153846153846154, lift=1.5042735042735043), OrderedStatistic(items_base=frozenset({' company', ' exchange', ' japan'}),
items_add=frozenset({' market', ' january'}), confidence=0.8888888888888888, lift=1.504273504273504), OrderedStatistic(items_base=frozenset({' compan
' market', ' exchange'}), items_add=frozenset({' january', ' japan'}), confidence=0.6666666666666667, lift=1.6296296296296298),
OrderedStatistic(items_base=frozenset({' market', ' exchange', ' japan'}), items_add=frozenset({' company', ' january'}), confidence=0.8888888888888
lift=1.7777777777777777), OrderedStatistic(items_base=frozenset({' january', ' japan', ' market'}), items_add=frozenset({' company', ' exchange'}),
confidence=0.8888888888888888, lift=1.504273504273504), OrderedStatistic(items_base=frozenset({' company', ' market', ' exchange', ' japan'}),
items_add=frozenset({' january'}), confidence=1.0, lift=1.6923076923076923)])]
```

From the above output, we can clearly observe that the most frequent items are (company, January, exchange, japan, market, company).

We can further notice that association rules with all combinations which are greater than the minimum values are also listed down.

Example:

- January → exchange, japan, company
- Company, exchange → January, japan
- January → exchange, japan, market
- January → exchange, japan, market, company
- Company, exchange → January, japan, market
- Company, January → exchange, japan, market
- market, January → exchange, japan, company

There are lot more such combinations listed in the above output

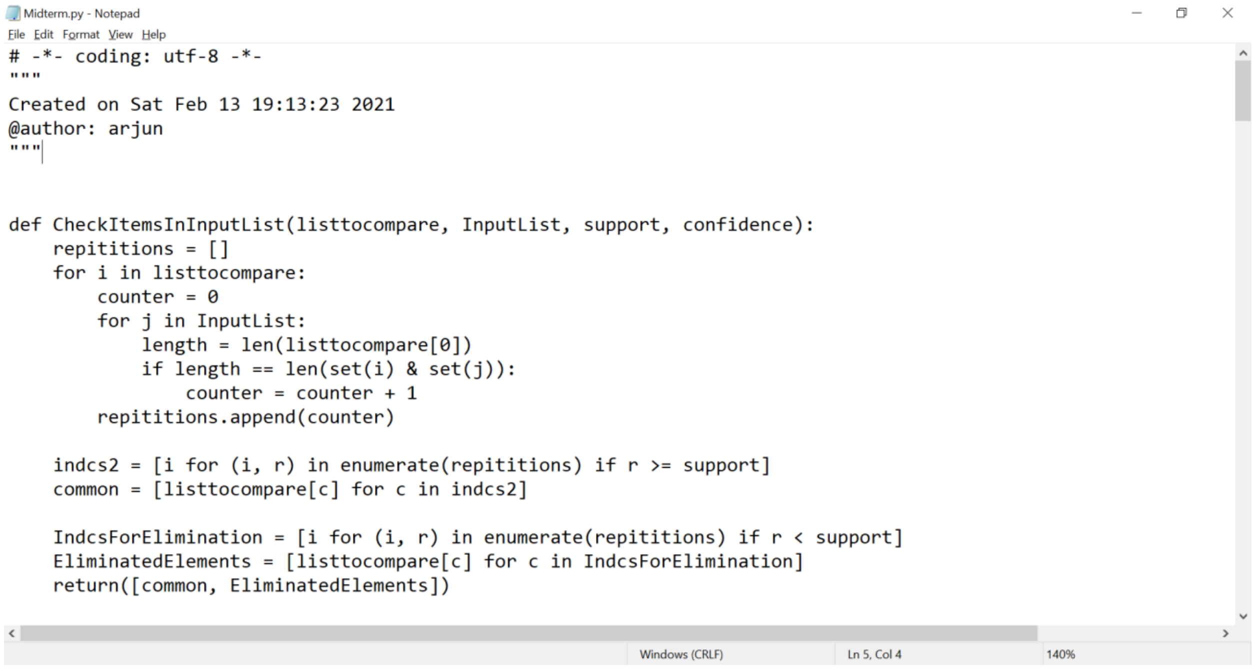
Method B: Apriori Algorithm Implementation

I have implemented the Apriori algorithm in 3 stages

- Stage 1: To check if the keywords are present in csv file and generate a frequency distribution for the keywords in the transaction database
- Stage 2: Combine the keywords that are generated in stage 1
- Stage 3: Check if the generated combinations is not a superset of eliminated non-frequent items.
- All the above stages are repeated until there can be no further new combinations.

Method B Stage 1 Details:

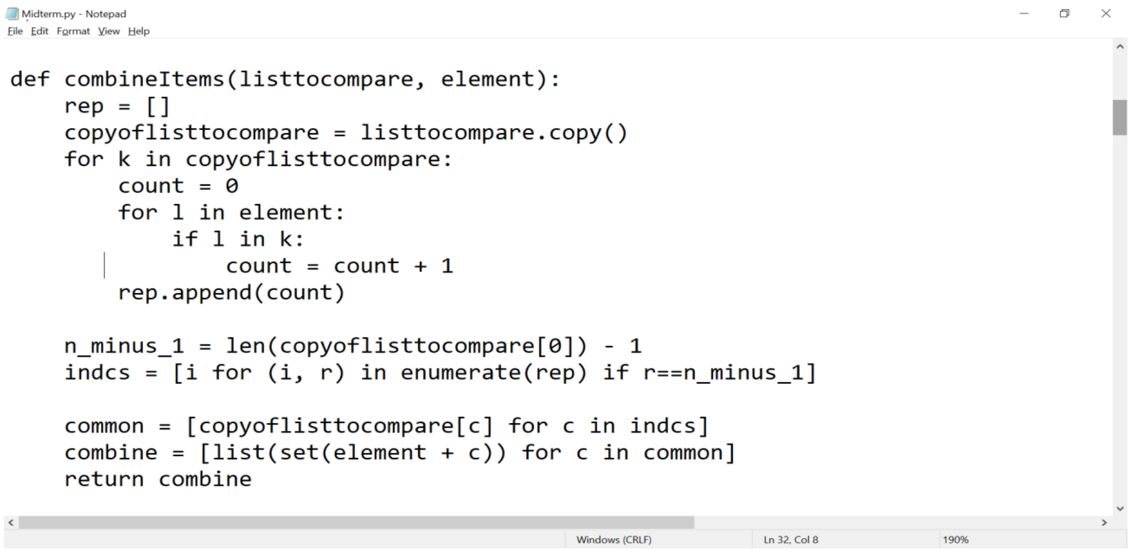
For stage 1 implementation I have written a function called CheckItemsInInputList.



```
# -*- coding: utf-8 -*-
"""
Created on Sat Feb 13 19:13:23 2021
@author: arjun
"""


def CheckItemsInInputList(listtocompare, InputList, support, confidence):
    repititions = []
    for i in listtocompare:
        counter = 0
        for j in InputList:
            length = len(listtocompare[0])
            if length == len(set(i) & set(j)):
                counter = counter + 1
        repititions.append(counter)

    indcs2 = [i for (i, r) in enumerate(repititions) if r >= support]
    common = [listtocompare[c] for c in indcs2]

    IndcsForElimination = [i for (i, r) in enumerate(repititions) if r < support]
    EliminatedElements = [listtocompare[c] for c in IndcsForElimination]
    return([common, EliminatedElements])
```

The above shown picture checks for the keywords in Input Transaction list and if the keywords frequency is more than the user provided support value then those items will be sent further for next set of combinations. If the Items frequency in the transaction dataset is less than the user provided support value, then those items will not be further combined in the next function. Those items will be stored in a list to check for non-frequent supersets.

Method B Stage 2 Details:

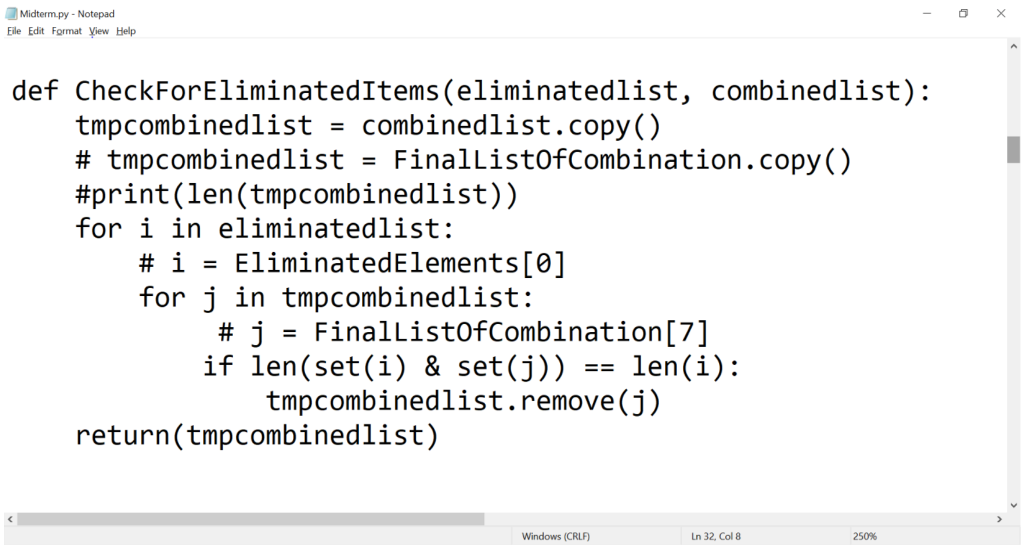


```
def combineItems(listtocompare, element):
    rep = []
    copyoflisttocompare = listtocompare.copy()
    for k in copyoflisttocompare:
        count = 0
        for l in element:
            if l in k:
                count = count + 1
        rep.append(count)

    n_minus_1 = len(copyoflisttocompare[0]) - 1
    indcs = [i for (i, r) in enumerate(rep) if r==n_minus_1]

    common = [copyoflisttocompare[c] for c in indcs]
    combine = [list(set(element + c)) for c in common]
    return combine
```

For Stage 2 I have written a function called combineItems.

This function combines the keywords from the output of the previous function (CheckItemsInInputList). Here while combining the items, we make sure that there are enough items in common to make a further combination. For example, if we are combining 2-frequent Itemset, like (A, B) and (B, C) there must be at least one item in common to combine. Here, B is common in both the itemsets and hence we can combine to form (A, B, C).

Method B Stage 3 Details:



```
def CheckForEliminatedItems(eliminatedlist, combinedlist):
    tmpcombinedlist = combinedlist.copy()
    # tmpcombinedlist = FinalListOfCombination.copy()
    #print(len(tmpcombinedlist))
    for i in eliminatedlist:
        # i = EliminatedElements[0]
        for j in tmpcombinedlist:
             # j = FinalListOfCombination[7]
            if len(set(i) & set(j)) == len(i):
                tmpcombinedlist.remove(j)
    return(tmpcombinedlist)
```

In stage 3, we make sure that the combined items are not a super set of non-frequent itemset. There by eliminating all the non-frequent supersets, we only pass the items that contains the frequent itemset.

Repeat all the stages until there are no more combinations to be formed. The final combination will be the frequent Itemset.

Method B Stage 4 Details

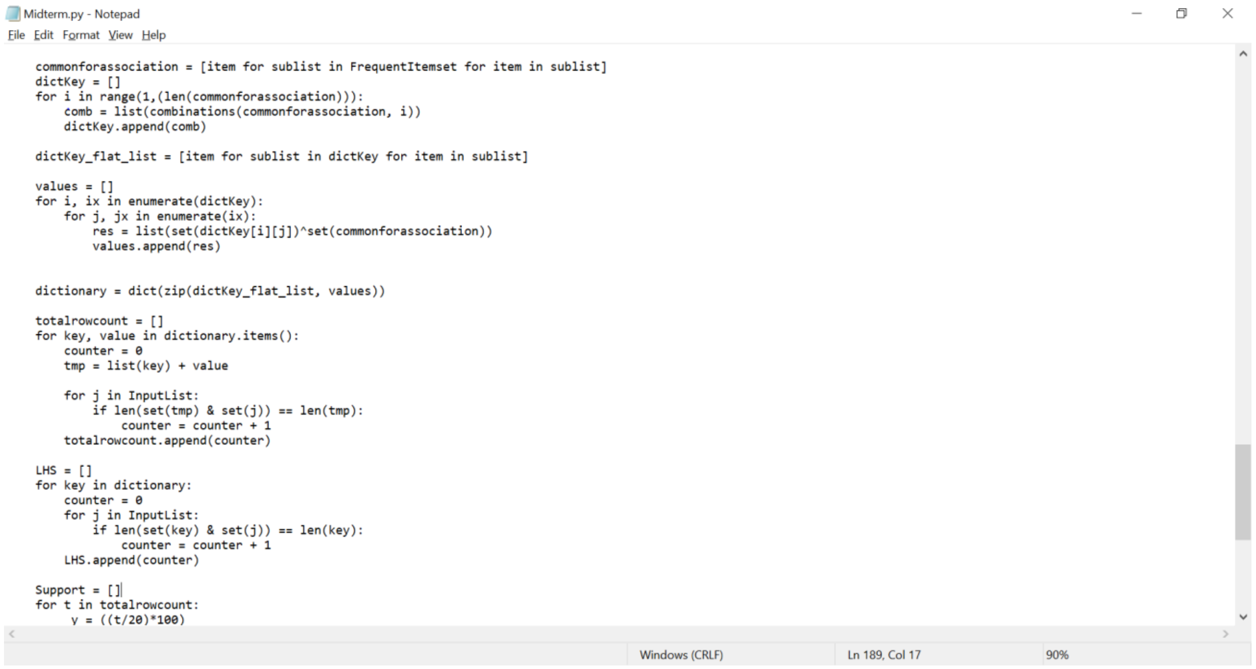
Then the frequent Itemset checks for association rules. Calculate the support and confidence for all the relationships.

Eg: A→ B

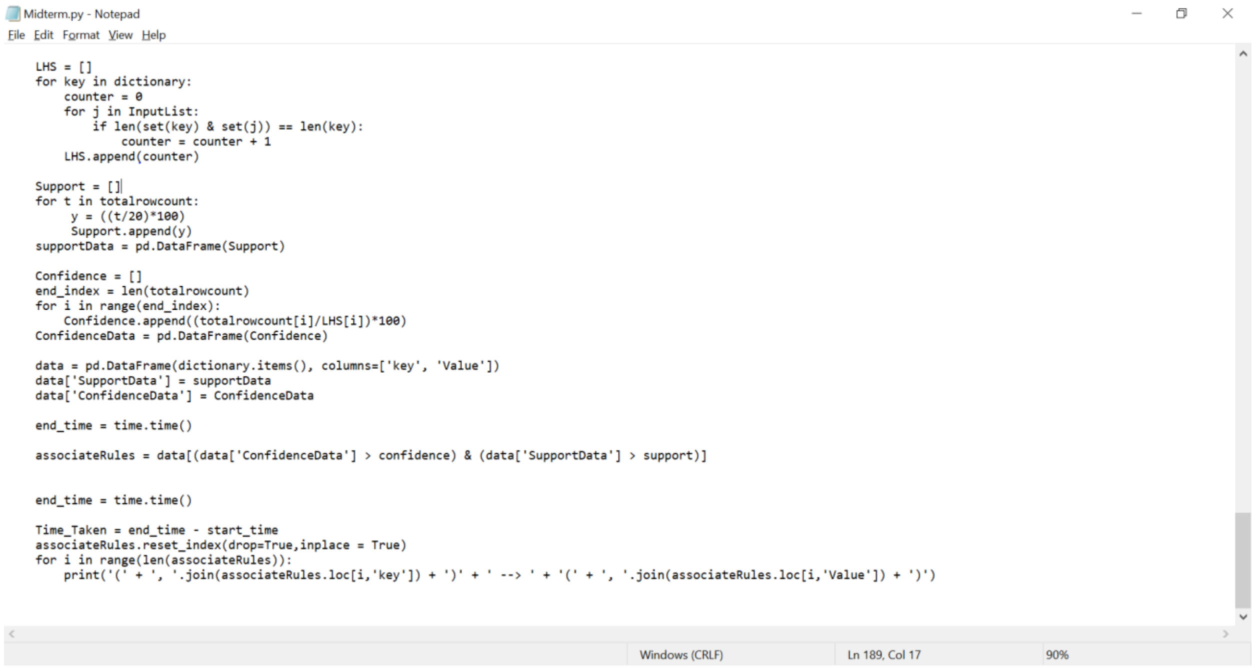
Support = The number of transactions in which both A and B are involved by total number of transactions

Confidence = Support of A and B by Support of A

If the support and confidence values of an association rule is greater than the user input support and confidence values then those verify the associative rule and are called the associative rules of the transaction.



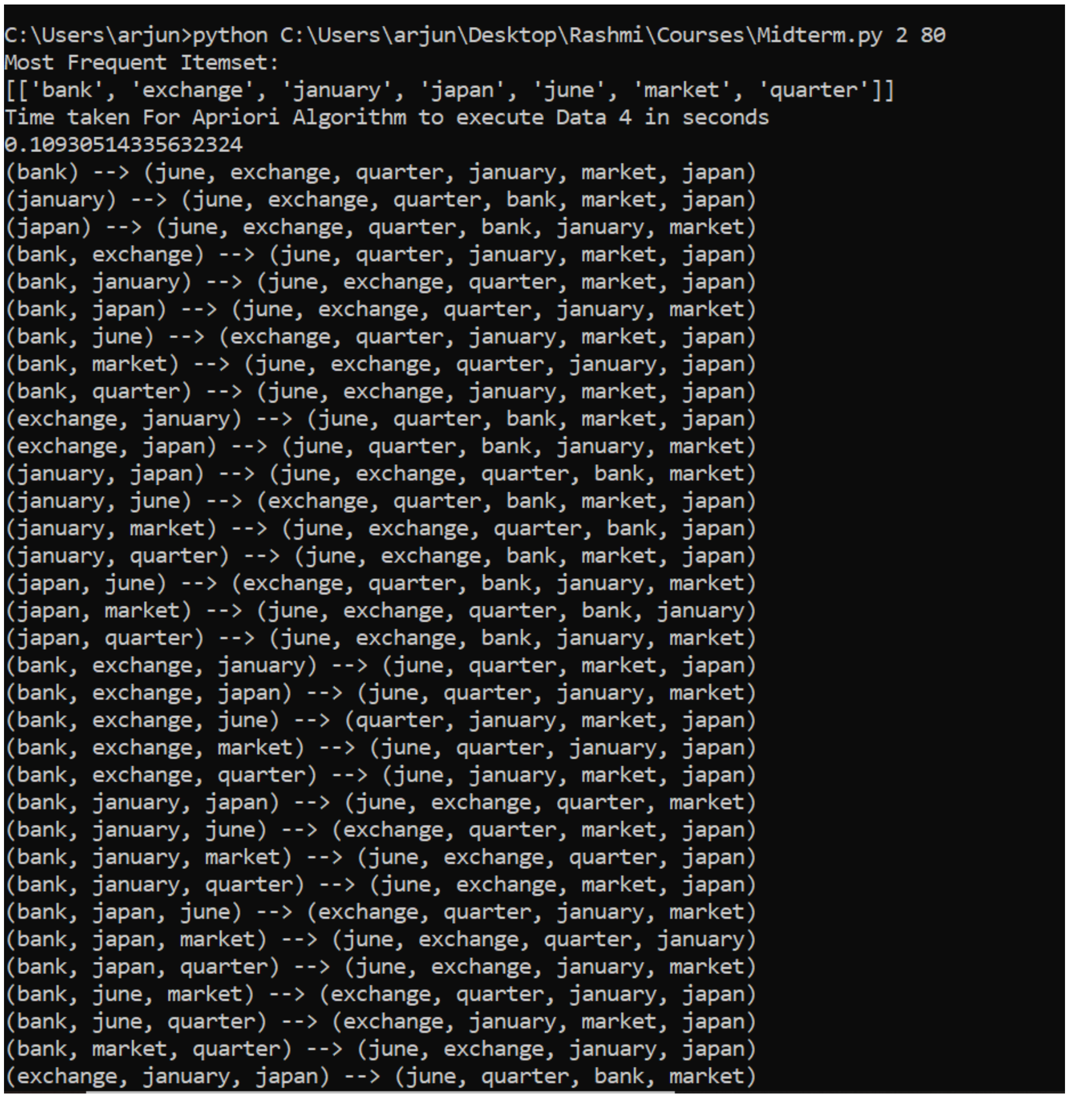
```python
commonforassociation = [item for sublist in FrequentItemset for item in sublist]
dictKey = []
for i in range(1,(len(commonforassociation))):
    comb = list(combinations(commonforassociation, i))
    dictKey.append(comb)

dictKey_flat_list = [item for sublist in dictKey for item in sublist]

values = []
for i, ix in enumerate(dictKey):
    for j, jx in enumerate(ix):
        res = list(set(dictKey[i][j])^set(commonforassociation))
        values.append(res)


dictionary = dict(zip(dictKey_flat_list, values))

totalrowcount = []
for key, value in dictionary.items():
    counter = 0
    tmp = list(key) + value

    for j in InputList:
        if len(set(tmp) & set(j)) == len(tmp):
            counter = counter + 1
    totalrowcount.append(counter)

LHS = []
for key in dictionary:
    counter = 0
    for j in InputList:
        if len(set(key) & set(j)) == len(key):
            counter = counter + 1
    LHS.append(counter)

Support = []
for t in totalrowcount:
     y = ((t/20)*100)
```



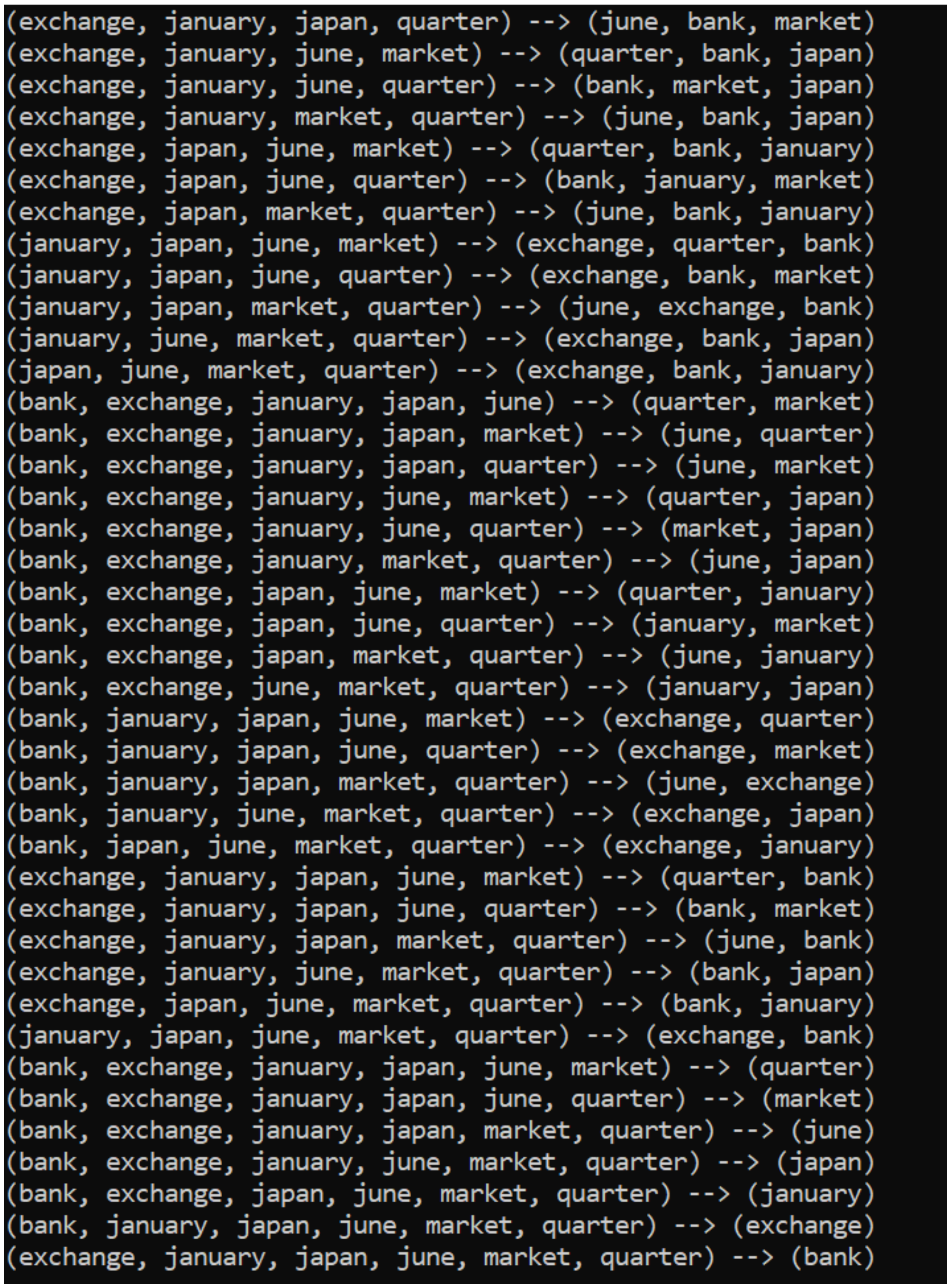
```python
LHS = []
for key in dictionary:
    counter = 0
    for j in InputList:
        if len(set(key) & set(j)) == len(key):
            counter = counter + 1
    LHS.append(counter)

Support = []
for t in totalrowcount:
     y = ((t/20)*100)
     Support.append(y)
supportData = pd.DataFrame(Support)

Confidence = []
end_index = len(totalrowcount)
for i in range(end_index):
    Confidence.append((totalrowcount[i]/LHS[i])*100)
ConfidenceData = pd.DataFrame(Confidence)

data = pd.DataFrame(dictionary.items(), columns=['key', 'Value'])
data['SupportData'] = supportData
data['ConfidenceData'] = ConfidenceData

end_time = time.time()

associateRules = data[(data['ConfidenceData'] > confidence) & (data['SupportData'] > support)]


end_time = time.time()

Time_Taken = end_time - start_time
associateRules.reset_index(drop=True,inplace = True)
for i in range(len(associateRules)):
    print('(' + ', '.join(associateRules.loc[i,'key']) + ')' + ' --> ' + '(' + ', '.join(associateRules.loc[i,'Value']) + ')')
```

Method B Output:

My laptop kept heating up and generated memory error for using 50 keywords for each document. Hence, I used eight key words to represent each document. Below is the output: generating association rules between 50 documents represented by eight words each.



```
C:\Users\arjun>python C:\Users\arjun\Desktop\Rashmi\Courses\Midterm.py 2 80
Most Frequent Itemset:
[['bank', 'exchange', 'january', 'japan', 'june', 'market', 'quarter']]
Time taken For Apriori Algorithm to execute Data 4 in seconds
0.10930514335632324
(bank) --> (june, exchange, quarter, january, market, japan)
(january) --> (june, exchange, quarter, bank, market, japan)
(japan) --> (june, exchange, quarter, bank, january, market)
(bank, exchange) --> (june, quarter, january, market, japan)
(bank, january) --> (june, exchange, quarter, market, japan)
(bank, japan) --> (june, exchange, quarter, january, market)
(bank, june) --> (exchange, quarter, january, market, japan)
(bank, market) --> (june, exchange, quarter, january, japan)
(bank, quarter) --> (june, exchange, january, market, japan)
(exchange, january) --> (june, quarter, bank, market, japan)
(exchange, japan) --> (june, quarter, bank, january, market)
(january, japan) --> (june, exchange, quarter, bank, market)
(january, june) --> (exchange, quarter, bank, market, japan)
(january, market) --> (june, exchange, quarter, bank, japan)
(january, quarter) --> (june, exchange, bank, market, japan)
(japan, june) --> (exchange, quarter, bank, january, market)
(japan, market) --> (june, exchange, quarter, bank, january)
(japan, quarter) --> (june, exchange, bank, january, market)
(bank, exchange, january) --> (june, quarter, market, japan)
(bank, exchange, japan) --> (june, quarter, january, market)
(bank, exchange, june) --> (quarter, january, market, japan)
(bank, exchange, market) --> (june, quarter, january, japan)
(bank, exchange, quarter) --> (june, january, market, japan)
(bank, january, japan) --> (june, exchange, quarter, market)
(bank, january, june) --> (exchange, quarter, market, japan)
(bank, january, market) --> (june, exchange, quarter, japan)
(bank, january, quarter) --> (june, exchange, market, japan)
(bank, japan, june) --> (exchange, quarter, january, market)
(bank, japan, market) --> (june, exchange, quarter, january)
(bank, japan, quarter) --> (june, exchange, january, market)
(bank, june, market) --> (exchange, quarter, january, japan)
(bank, june, quarter) --> (exchange, january, market, japan)
(bank, market, quarter) --> (june, exchange, january, japan)
(exchange, january, japan) --> (june, quarter, bank, market)
```


```
(exchange, january, japan, quarter) --> (june, bank, market)
(exchange, january, june, market) --> (quarter, bank, japan)
(exchange, january, june, quarter) --> (bank, market, japan)
(exchange, january, market, quarter) --> (june, bank, japan)
(exchange, japan, june, market) --> (quarter, bank, january)
(exchange, japan, june, quarter) --> (bank, january, market)
(exchange, japan, market, quarter) --> (june, bank, january)
(january, japan, june, market) --> (exchange, quarter, bank)
(january, japan, june, quarter) --> (exchange, bank, market)
(january, japan, market, quarter) --> (june, exchange, bank)
(january, june, market, quarter) --> (exchange, bank, japan)
(japan, june, market, quarter) --> (exchange, bank, january)
(bank, exchange, january, japan, june) --> (quarter, market)
(bank, exchange, january, japan, market) --> (june, quarter)
(bank, exchange, january, japan, quarter) --> (june, market)
(bank, exchange, january, june, market) --> (quarter, japan)
(bank, exchange, january, june, quarter) --> (market, japan)
(bank, exchange, january, market, quarter) --> (june, japan)
(bank, exchange, japan, june, market) --> (quarter, january)
(bank, exchange, japan, june, quarter) --> (january, market)
(bank, exchange, japan, market, quarter) --> (june, january)
(bank, exchange, june, market, quarter) --> (january, japan)
(bank, january, japan, june, market) --> (exchange, quarter)
(bank, january, japan, june, quarter) --> (exchange, market)
(bank, january, japan, market, quarter) --> (june, exchange)
(bank, january, june, market, quarter) --> (exchange, japan)
(bank, japan, june, market, quarter) --> (exchange, january)
(exchange, january, japan, june, market) --> (quarter, bank)
(exchange, january, japan, june, quarter) --> (bank, market)
(exchange, january, japan, market, quarter) --> (june, bank)
(exchange, january, june, market, quarter) --> (bank, japan)
(exchange, japan, june, market, quarter) --> (bank, january)
(january, japan, june, market, quarter) --> (exchange, bank)
(bank, exchange, january, japan, june, market) --> (quarter)
(bank, exchange, january, japan, june, quarter) --> (market)
(bank, exchange, january, japan, market, quarter) --> (june)
(bank, exchange, january, june, market, quarter) --> (japan)
(bank, exchange, japan, june, market, quarter) --> (january)
(bank, january, japan, june, market, quarter) --> (exchange)
(exchange, january, japan, june, market, quarter) --> (bank)
```