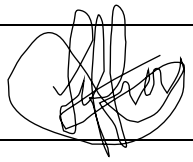
Specialist Diploma in Applied Artificial Intelligence
Individual Coursework 1 / AAI-5-C3389C-A

C3389C AY2021 Sem3 Cohort: 1
Individual Coursework 1 (CW1) Submission

Personal Details	
Name	**WONG QI YUAN, JEFFREY**
Student No.	**20053371**

Compliance Statement	
Plagiarism I declare that this report is my original work. I understand that if I am suspected of plagiarism, my enrolment in the programme may be terminated.	✓
Retention of Backup Copy I declare that I have a back-up electronic copy of this report for immediate submission.	✓
Signature	
IMPORTANT: Non-compliance to these clauses will result in unconditional rejection of your submission	

Guideline on Academic Dishonesty and Plagiarism

REPUBLIC POLYTECHNIC

School of Infocomm (SOI)

Guideline on Academic Dishonesty and Plagiarism

SOI is committed that all SOI students who submit work for assessment receives due credit for their work. This means that it is unfair and wrong for students to submit work for assessment that dishonestly represents the work of others as their own. This is called plagiarism and it represents a common form of academic dishonesty given as there is intent to deceive the assessor.

Plagiarism is broadly defined as knowingly presenting another person's ideas, findings, or work as one's own. This also means copying or reproducing the work without due acknowledgement of the source. Work submitted for assessment may be regarded as plagiarised where significant portions of the work has been reproduced from the work of another student(s), since this exceeds the boundaries of collaborative learning.

Collaborative learning can be defined as any constructive educational and intellectual activity that aims to facilitate optimal learning outcomes through interaction between students, including

- researching, writing and/or presenting joint work;
- discussion of general themes and concepts;
- interpretation of assessment criteria;
- informal study / discussion groups; and
- strengthening and development of academic writing skills through peer assistance.

i) Within individual assignments (those not explicitly labelled group assignments), discussion with other students is legitimate but joint writing of solutions or viewing another student's solution is not.
ii) Within group assignments, joint writing of solutions is legitimate and expected, but the work must bear all and only the names of the actual contributors. Parts of group assignments labelled as individual can be legitimately shown to and discussed in detail with other group members, but must be written by the individual alone.

Within the discipline of IT, the following activities are frequently associated with plagiarism and are therefore treated as evidence of academic dishonesty except when carried out legitimately within a declared group project:

i) making files associated with assessed work available to others, by any means (making such files available is always beyond the boundaries of collaborative learning);
ii) attempting to view files owned by another student without permission, even when those files have been made accessible to others;
iii) encouraging other students to carry out operations which have the effect of making files accessible;
iv) using another student's computer while that student is temporarily absent;
v) taking other students' work from a printer;

vi) using any quantity of material from one or more web sites or other published sources without acknowledgement and attempting to pass it off as one's own work, is also plagiarism with intent to deceive.

Other forms of academic dishonesty include:

vii) recycling (the submission for assessment of one's own work, or of work which is substantially the same, which has previously been counted towards the satisfactory completion of another unit of study, and credited towards a polytechnic diploma, and where the assessor has not been informed);
viii) fabrication of data;
ix) the engagement of another person to complete an assessment or examination in place of the student, whether for payment or otherwise;
x) communication, whether by speaking or some other means, to other students during an examination;
xi) bringing into an examination and concealing forbidden material such as text books, notes, calculators, mobile phones or computers;
xii) attempts to read other students' work during an examination; and
xiii) writing an examination or test paper, or consulting with another person about the examination or test, outside the confines of the examination room without permission.

Students submitting work for assessment in SOI will be required to sign a declaration stating that, except where specifically acknowledged, the work contained in the assignment / project is their own work, has not been copied from other sources and has not been previously submitted for award or assessment.

Where there is a question about a student's contribution, then in order to arrive at the final mark/grade for the assessment, students may be:

i) asked to identify those portions of the work contributed by them personally and required to demonstrate their knowledge of the relevant material by answering oral questions or
ii) asked to undertake supplementary work, either written or in the laboratory.

SOI is opposed to and will not tolerate Plagiarism. It is the responsibility of all students to:

- ensure that they do not commit or collude with another person to commit Plagiarism;
- report possible instances of Plagiarism; and
- comply with this guideline on Academic Dishonesty and Plagiarism.

(Adapted from :
http://sydney.edu.au/engineering/it/current_students/undergrad/policies/academic_honesty.shtml)

ASSIGNMENT / PROJECT – INDIVIDUAL / GROUP ASSESSMENT

Module : Recommender Systems (C3389C)

Assignment or Project Number and / or Name : Individual Coursework 1/ AAI-5-C3389C-A

Assessor or Supervisor Name : Andy Lee & Ricky Teo

DECLARATION

I declare that I have read and understood SOI's *Guideline on Academic Dishonesty and Plagiarism*. I understand that failure to comply with the *Guideline on Academic Dishonesty and Plagiarism* can lead to severe penalties.

Student ID: 20053371

Student Name: Wong Qi Yuan, Jeffrey

Signed: 

Signed Date: 12/7/2021

Appendix II

Section A [20 marks]

Based on your lesson on Term Frequency – Inverse Document Frequency (TF-IDF) (Lesson 2 – Content-based Recommender Systems), you are required to apply your theoretical understanding to perform a TF-IDF exercise using Python programming with pandas.

Requirements

1) Corpus selection (2 marks)
    i. any open source volume, e.g. from Project Gutenberg
    ii. create five documents, e.g. from consecutive chapters in your selected volume, with each document having at least 1500 words
2) Reading of the documents created in Step 1 (3 marks)
3) Pre-processing (4 marks)
    i. Punctuation removal
    ii. Stop-words removal
4) TF calculation (3 marks)
5) IDF calculation (3 marks)
6) TF-IDF calculation (2 marks)
7) Identification of most relevant/important keywords in the corpus, e.g. by sorting, ranking, etc. using relevant pandas functions (3 marks)

Notes:

- All steps above besides Step 1 should be done using Python code
- You are allowed to use open-source libraries as appropriate, but more credit will be given for self-written functions to accomplish the tasks where possible, which serves to demonstrate your understanding of the algorithms and coding proficiency
- You will be graded based on the correctness and quality of your code, i.e. it should be error-free and clearly explained through your documentation (Section B), code comments and good coding practices (e.g. meaningful variable names, and clear, logical program structure)
- More credit will be given for a stand-alone Python program, i.e. a program that can be run without requiring jupyter notebook or Google Colab.

Answer for Question 1

```
import pandas as pd
import numpy as np
from nltk.stem import WordNetLemmatizer

################################ Define Functions ########################
def removePunctuation(docs):
   # define a set of punctuations
   punctuation = "[!#$%&'()*+,-./:;<=>?@—[\]^_`{|}“~]"

   # remove all the punctuations present in a list of documents by using the built-in function "replace
()"
   # store it into a new variable
   rpunct = docs.replace(punctuation,"", regex = True).replace('"', "", regex = True)
   # return a list of documents with all punctuations removed
   return rpunct


def removeNumbers(rPunctDocs):
   # define a set of numerical digits
   numbers = "[0123456789]"

   # remove all the numerical digits present in a list of documents by using the built-in function
"replace ()"
   # store it into a new variable
   rnumbers = rPunctDocs.replace(numbers, "", regex = True)
   # return a list of documents with all numerical digits removed
   return rnumbers


def tokenization(rnumberDocs):

   # use the built-in function - "split ()" to split a phrase, sentences, paragraph or the entire
documents into an individual words
   # store it into a new variable
   tokenizedText = rnumberDocs.str.split(" ", expand = False)
   # return a list of documents with tokenized texts
   return tokenizedText
```

```
def removeStopwords(tokenizedDocs):
   # define a set of stopwords
   stopwords = ['i','the','me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
                'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',
                'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she',
                "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
                'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who',
                'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are',
                'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do',
                'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
                'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against',
                'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below',
                'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again',
                'further', 'then', 'may', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all',
                'any', 'both', 'each', 'few', 'more', 'whose','most', 'other', 'some', 'such', 'no', 'nor',
                'not', 'only', 'else', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will',
                'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're',
                've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
                "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
                'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't",
                'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
                "wouldn't"]

   # go through each set of documents with tokenized texts
   for j in range(num_of_docs):
      # go through each word in the list of stopwords
      for stopword in stopwords:
         # go through each tokenized text in each set of documents
         for element in tokenizedDocs[j]:
            # if each tokenized text in each set of documents represent the word in the list of stopwords,
            if element == stopword:
               # remove the respective tokenized text from its respective document by using the built-in
function "remove ()"
               tokenizedDocs[j].remove(element)
   # return a list of documents with stopwords removed.
   return tokenizedDocs
```

```
def lemmatizingWords(tokenizedDocs):
    # create a class object for WordNetLemmatizer()
    # and store it into a new variable
    wnl = WordNetLemmatizer()
    # Go through each set of documents with tokenized texts
    for j in range(num_of_docs):
        # go through each tokenized text for each set of documents
        for index in range(len(tokenizedDocs[j])):
            # perform lemmatization process for each tokenized text for each set of documents
            # store it into the same variable
            tokenizedDocs[j][index] = wnl.lemmatize(tokenizedDocs[j][index])
    # return a list of documents with lemmatized texts
    return tokenizedDocs

def compute_TermFreq(wordsDict, bag_of_words):
    # initialize an empty dict
    termFreqDict = dict()

    # count the number of terms present in the bags of words
    count_bows = len(bag_of_words)

    # iterate a list of words dictionary with value counts using the built-in function "items()"
    for wordOccurrence, val_count in wordsDict.items():
        # compute the term frequency for the respective word occurrence by counting the number of
words appear in a document
        # divided by the total number of terms in the bags of words
        # store the computed term frequency into dict variable
        termFreqDict[wordOccurrence] = val_count/ float(count_bows)
    # return the computed term-frequency
    return termFreqDict


def compute_InverseDocFreq(list_of_docs):
    # create a key-value pairs dict by using the built in function "dict.fromkeys()"
    # initialize all the respective key terms with zero values
    # store it into a new variable
    inverseDocFreqDict = dict.fromkeys(list_of_docs[0].keys(), 0)


    # go through each list of documents
    for docs in list_of_docs:
        # go through each key-value pairs dictionary from each list of documents
        for wordOccurrence, val_count in docs.items():
            # if the value count is greater than zero,
            if val_count > 0:
                # increment the respective key terms by 1 and
                # store the computed value count with its respective key terms into the dict variable
                inverseDocFreqDict[wordOccurrence] = inverseDocFreqDict[wordOccurrence] + 1
```

```
    # go through each of the key-value pairs dictionary as computed above
    for wordOccurrence, val_count in inverseDocFreqDict.items():
        # compute the inverse-doc frequency for each respective key terms using IDF-smoothing
technique
        inverseDocFreqDict[wordOccurrence] = (np.log((1+len(list_of_docs))/ (1+float(val_count)))) + 1
    # return the computed idf values
    return inverseDocFreqDict


def compute_TFIDF(tf, idf):
    # initialize an empty dict
    tf_idfDict = {}

    # go through each key-value pairs dict from the list of computed tf values.
    for wordOccurrence, val in tf.items():
        # compute the tf-idf scores by multiplying the TF-value for the respective word term with the
        # IDF-value for the same word terms
        # store the computed tf-idf scores with its respective key-term into the dict
        tf_idfDict[wordOccurrence] = val * idf[wordOccurrence]
    # return the tf-idf scores with its all the respective key-terms
    return tf_idfDict


################################### main ###################################
# display and prompt the user to upload the 5 documents
num_of_docs = 5
documents = {}
doc_names = ['Doc 1', 'Doc 2', 'Doc 3', 'Doc 4', 'Doc 5']

print("Welcome to TFIDF Calculation")
print("*******************************************")
print("Instructions to User:")
print("1) You are required to prepare and upload 5 documents with consecutive chapters.")
for i in range(num_of_docs):
    filepath = input("Enter the filepath of Document {}: ".format(i+1)) # prompt the user for filepath
    text = open(filepath, "r", encoding = "utf-8") # open the content of the file
    documents[doc_names[i]] = text.read().replace('\n', ' ').lower() # read and store the content from
respective file into dict func
    text.close() # close the file after each cycle of reading the text file

# store all the context from respective docs into the Pandas Series
docs = pd.Series(documents)
```

```
########################## Preprocessing Steps##########################

# call the function to remove punctuations
# and store it into a new variable
rPunctDocs = removePunctuation(docs)

# call the function to remove numerical digits
# and store it into a new variable
rnumberDocs = removeNumbers(rPunctDocs)

# call the function to perform tokenization process and
# store the returned tokenized texts into a new variable
tokenizedDocs = tokenization(rnumberDocs)

# call the function to remove stopwords
# store the returned tokenized texts with stopwords removed into a new same variable
tokenizedDocs = removeStopwords(tokenizedDocs)

# go through each list of documents with tokenized texts
for j in range(num_of_docs):
   # using the built-in function "filter ()" to remove all the empty list from each respective documents
   # store it into a same variable
   tokenizedDocs[j] = list(filter(None, tokenizedDocs[j]))

# call the function to perform lemmatization process
# store the returned lemmatized texts into a new variable
lemmatizedDocs = lemmatizingWords(tokenizedDocs)

# create  a copy of the pre-processed text documents
# store it into a new variable
bag_of_words = lemmatizedDocs.copy()
```

```
########################## Counting Words Occurrence ###################
# extract the unique and non-duplicate feature names
feature_names = list(set().union(bag_of_words[0], bag_of_words[1], bag_of_words[2],
bag_of_words[3], bag_of_words[4]))

# initialize an empty dict with zero values for respective terms or words for respective docs
num_of_words = [dict.fromkeys(feature_names, 0) for i in range(num_of_docs)]

# count the number of unique words occurs in the respective documents
for j in range(num_of_docs):
   numOfWords = num_of_words[j]
   for word in bag_of_words.iloc[j]:
      numOfWords[word] = numOfWords[word] + 1
   num_of_words[j] = numOfWords

# store the respective words count from respective documents into a DataFrame Object
num_of_words_df = pd.DataFrame(num_of_words, index = doc_names)
num_of_words_df_T = pd.DataFrame(num_of_words_df, index = doc_names).T # transpose the
dataframe
wordDictIDF = num_of_words_df.copy() # create another copy for IDF computation

# display the respective words count for the respective documents
print("\nNumber of Words Occurrence for the Respective Documents: ")
print("*********************************************************")
print(num_of_words_df_T)


############################## TF Computation ############################

# in the list comprehrension, call the function to compute the term-frequency by going through
# each document with a set of bags of words and a list of word dictionary with value counts
# and store it into a new variable
termFreq_res = [compute_TermFreq(num_of_words_df.iloc[i], bag_of_words.iloc[i]) for i in
range(num_of_docs)]

# store the resultant term-frequency into a new dataframe
termFreq_df = pd.DataFrame(termFreq_res, index = doc_names)
# transpose the dataframe for the resultant term-frequency
termFreq_df_T = pd.DataFrame(termFreq_res, index = doc_names).T

# display the transpose term frequency results
print("\nTerm Frequency (TF) for the Respective Keywords Across Documents: ")
print("*****************************************************************")
print(termFreq_df_T)
```

```
############################## IDF Computation ############################

# create a list of documents
list_of_docs = [wordDictIDF.iloc[i] for i in range(num_of_docs)]

# call the function to compute IDF
idf_res = compute_InverseDocFreq(list_of_docs)

# convert the resultant idf from a dictionary into a Pandas Series
idf_series = pd.Series(idf_res)

# display the idf results
print("\nInverse Documents Frequency (IDF) for the Respective Keywords: ")
print("*****************************************************************")
print(idf_series)

############################## TF*IDF Computation ######################

# initialize an empty list
tfidf_res = list()
# go through each list of documents
for j in range(num_of_docs):
   # call the function to compute TFIDF for each documents
   # append the return TFIDF key- value pairs for each document
   # and store it into an empty list
   tfidf_res.append(compute_TFIDF(termFreq_df.iloc[j], idf_res))

# convert the tfidf scores into a DataFrame
tfidf_df = pd.DataFrame(tfidf_res, index = doc_names)
# transpose the dataframe
tfidf_df_T = pd.DataFrame(tfidf_res, index = doc_names).T

# display the transpose tfidf results with its TF-IDF scores for each respective keys across the
documents
print("\nTerm Frequency - Inverse Documents Frequency (TF*IDF) for the Respective Documents: ")
print("*********************************************************************")
print(tfidf_df_T)
```

```
######################### Ranking & Identification ####################
# sum up the tf-idf scores for each keywords across the documents and sort it in descending order
# store the summed tf-idf scores into a DataFrame
ranking = pd.DataFrame(tfidf_df.sum().sort_values(ascending = False), columns = ['Overall TFIDF
Score'])


# display the summed tfidf scores for each keywords
print("\nSummed TFIDF Scores for Respective Keywords")
print("*********************************************")
print(ranking)

# identification of most relevant/ important keywords in the corpus
# extract the index name for the top 100 important keywords
top_100 = ranking.head(100).index
# set the top 100 important keywords into a list
impt_keywords = top_100.tolist()

# display the top 100 important keywords
print("\nTop 100 relevant & important keywords in the corpus")
print("************************************************************")
print(impt_keywords)
```

Section B [10 marks]

Write a report to explain your work in Section A using the following template:

Name: WONG QI YUAN, JEFFREY	**Student ID: 20053371**
Corpus title:	ANOMALIES and CURIOSITIES of MEDICINE
Corpus source:	https://www.gutenberg.org/files/747/747-h/747-h.htm
Document description	Chapter 1: Genetic Anomalies Chapter 2: Prenatal Anomalies Chapter 3: Obstetric Anomalies Chapter 4: Prolificity Chapter 5: Major Terata
Pre-processing steps	**<Define Function to Remove Punctuations:>** Get a list of unprocessed documents as the parameter input. Step 1: Define a set of punctuations. Step 2: Remove all the punctuations present in a list of documents by using the built-in function - 'replace ()' and store it into a new variable - 'rpunct'. Step 3: Return a list of documents with all punctuations removed - 'rpunct'. **<Calling Function to Remove Punctuations:>** Input a list of unprocessed documents as the function call parameter. Step 1: Call the function to remove the punctuations and store the return values into a new variable – 'rPunctDocs'. **<Define Function to Remove Numerical Digits: >** Get a list of documents with punctuation removed as the parameter input. Step 1: Define a set of numerical digits. Step 2: Remove all the numerical digits present in a list of documents by using the built-in function – 'replace ()' and store it into a new variable – 'rnumbers'. Step 3: Return a list of documents with all numerical digits removed – 'rnumbers'.

	<Calling Function to Remove Numerical Digits:> Input a list of documents with punctuation removed as the function call parameter. Step 1: Call the function to remove the numerical digits and store the return values into a new variable – 'rNumberDocs'.
	<Define Function to Perform Tokenization: > Get a list of documents with both punctuations and numerical digits removed as the parameter input. Step 1: Use the built-in function – 'split ()' to split a phrase, sentence, paragraph or the entire documents into an individual term and store it into a new variable – 'tokenizedText'. Step 2: Return a list of documents with tokenized texts – 'tokenized Text'. **<Calling Function to Perform Tokenization: >** Input a list of documents with both punctuations and numerical digits removed as the function call parameter. Step 1: Call the function to perform tokenization process and store the returned tokenized texts into a new variable – 'tokenizedDocs'.
	<Define Function to Remove Stopwords: > Get a list of documents with tokenized texts and with both punctuations and numerical digits removed as the parameter input. Step 1: Define a set of stopwords. Step 2: Go through each set of documents with tokenized texts. Step 2.1: Go through each word in the list of stopwords. Step 2.2: Go through each tokenized text in each set of documents. Step 2.3: If each tokenized text in each set of documents represent the word in the list of stopwords, remove the respective tokenized text from its respective document by using the built-in function – 'remove ()'. Step 3: Return a list of documents with stopwords removed – 'tokenizedDocs'.

	<Calling Function to Remove Stopwords: > Input a list of documents with tokenized texts and with both punctuations and numerical digits removed as the function call parameter. Step 1: Call the function to remove stopwords and store the returned tokenized texts with stopwords removed into the same variable – 'tokenizedDocs'. --- **<Remove the Empty List in the Tokenized Text>** Step 1 – Go through each list of documents with tokenized texts. Step 2 – Use the built-in function – 'filter ()' to remove the empty list from each respective document and store each respective document into the same variable – 'tokenizedDocs' as a list. --- **<Define Function to Perform Lemmatization: >** Get a list of documents with tokenized texts and with punctuations, numerical digits, and stopwords removed as the parameter input. Step 1: Define a NTLK library as "from nltk.stem" import WordNetLemmatizer". Step 2: Create a class object for WordNetLemmatizer () and store it into a new variable – 'wnl'. Step 3: Go through each set of documents with tokenized texts. Step 3.1: Go through each tokenized text for each set of documents. Step 3.2: Use pre-defined class object – 'wnl' and the built-in function – 'lemmatize' to perform lemmatization process for tokenized text for each set of documents and store it into the same variable – 'tokenizedDocs'. Step 4: Return a list of documents with lemmatized texts. **<Calling Function to Perform Lemmatization: >** Input a list of documents with tokenized texts and with all punctuations, numerical digits, and stopwords removed as the function call parameter. Step 1: Call the function to perform lemmatization and store the returned lemmatized texts into the new variable – 'lemmatizedDocs'.

	<After the end of pre-processing steps: > Step 1: Create a copy of the pre-processed text documents and store it into a new variable – 'bag_of_words'.
TF calculation	**<<Counting Words Occurrence: >>** Step 1: Create a new set with distinct words from bags of words using the built-in functions - 'union () and set ()' and store it as a list into a new variable – 'feature_names'. Step 2: In the list comprehension, initialize the key-value pairs dictionary with all the respective feature words with zero values for each document and store it as a list into a new variable – 'num_of_words'. Step 3: Go through each set of pre-processed documents by counting the number of unique words occurrence and store it into the same dictionary variable – 'num_of_words'. Step 4: Store the number of words occurrence into a new DataFrame variable – 'num_of_words_df'. Step 5: Transpose the 'num_of_words_df' dataframe and store it into a new variable – 'num_of_words_df_T'. Step 6: Display the transpose "num_of_words_df" as a DataFrame. **<<Computing Term-Frequency (TF): >>** **<Define Function to Compute Term-Frequency (TF): >** Get a list of bags of words for the current document and a list of words dictionary with value counts (key-value pairs) for the current document as the parameter's inputs. Step 1: Initialize "termFreqDict" variable with an empty dictionary. Step 2: Count the total number of terms present in the bags of words and store it into a new variable – 'count_bows'. Step 3: Iterate the key-value pairs (a list of words dictionary with value counts) using the built-in function – 'items ()': Step 3.1: Compute the term frequency for the respective word occurrence by counting the number of words appear in the current document divided by the total number of terms in the bags of words for the current document and store the computed term-frequency value for the respective word into the variable as declared in Step 1. Step 4: Return the computed term-frequency values.

	<Calling Function to Compute Term-Frequency (TF): > Input a list of bags of words for the current document and a list of words dictionary with value counts (key-value pairs) for the current document as the function call parameter. Step 1: In the list comprehension, call the function to compute the term-frequency by going through each document with a set of bags of words and a set of words dictionary with value counts and store the returned computation as a list into a new variable – 'termFreq_res'. **<Store TF into a DataFrame: >** Step 1: Store the "termFreq_res" into a new DataFrame variable – "termFreq_df". Step 2: Transpose the "termFreq_res" DataFrame and store it into a new DataFrame variable – "termFreq_df_T". **<Display Term-Frequency (TF) values for all respective documents: >** Step 1: Display the transpose term-frequency result as a DataFrame.
IDF calculation	**<Define Function to Compute Inverse-Doc Frequency (IDF): >** Get a list of documents with counting word occurrence as the parameter input. Step 1: Create a key-value pairs dictionary by using the built-in function – 'dict.fromkeys ()', initialize all the respective key terms with zero values and store it as a dictionary into a new variable – 'inverseDocFreqDict'. Step 2: Go through each list of documents. Step 2.1: Go through each key-value pairs dictionary for each list of documents. Step 2.2: If the value is greater than zero, increment the respective key term by 1 and store the computed value count with its respective key term into the dictionary variable as declared in Step 1. The whole process in Step 2 is iterated across the documents. Step 3: Go through each key-value pairs for 'inverseDocFreqDict' as computed in Step 2.2. Step 3.1: Compute the Inverse-Document Frequency (IDF) for each respective key term using IDF-Smoothing Technique. The whole process in Step 3 is iterated across the key-value pairs. Step 4: Return the computed IDF values.

	<Calling Function to Compute IDF: > Input a list of documents with counting words occurrence into the function call parameter. Step 1: Call the function to compute the Inverse-Document Frequency (IDF) and store it into a new variable – 'idf_res'. **<Display the IDF Result:>** Step 1: Convert the "idf_res" variable from a dictionary into a Pandas Series and store it into a new variable – 'idf_series'. Step 2: Display the "idf_series" table as a Series.
Keyword Identification	**<Define Function to Compute TF-IDF: >** Get a list of key-value pairs of the computed TF values with all respective key terms for each document and a list of key-value pairs of the computed IDF values with all respective key terms. Step 1: Initialize a "tf_idfDict" variable with an empty dictionary. Step 2: Go through each key-value pairs from a list of computed TF values. Step 2.1: Compute the TF-IDF scores by multiplying the TF-value for the respective term with the IDF-value for the same respective term and store the computed TF-IDF scores with its respective key term into the dictionary variable as declared in Step 1. The whole process is iterated for every other key-value pairs. Step 3: Return a set of TF-IDF scores with its all the respective key terms. **<Calling Function to Compute TF-IDF: >** Input a list of key-value pairs of the computed TF values with respective key terms for each document and a list of key-value pairs of the computed IDF values with respective key terms into the function call parameter. Step 1: Initialize a "tfidf_res" variable with an empty list. Step 2: Go through each list of documents. Step 2.1: Call the function to compute the TF-IDF for each document and append the returned computed TF-IDF values with its all the respective key terms into the new variable as declared in Step 1. The whole process is iterated for each list of documents. **<Store TF-IDF into DataFrame: >** Step 1: Convert the "tfidf_res" variable into a Padas DataFrame and store it into a new variable – "tfidf_df". Step 2: Transpose the "tfidf_df" variable created in Step 1 and store it into a new variable – "tfidf_df_T".

	<Display the TF-IDF Score for Respective Keywords Across Documents: > Step 1: Display the transpose DataFrame – "tfidf_df_T" with its TF-IDF scores for each respective keywords across the documents. **<Ranking of Keywords: >** Step 1: Using the "tfidf_df" variable, sum up the TF-IDF scores for each keyword across the documents and sort it in descending order. Then, store the summed TF-IDF scores into a DataFrame into a new variable – "ranking". **<Keywords Identification: >** Step1: Extract the index name for the top 100 important keywords from "ranking" variable DataFrame and store it into new variable – "top_100". Step 2: Set the "top_100" variable into a list and store it into a new variable – "impt_keywords". Step 3: Display the top 100 relevant & important keywords using "impt_keywords" variable.
Conclusion	**< TF-IDF managed to work well in surfacing important keywords in the corpus>** The corpus I selected was about the rare and extraordinary cases and abnormality in all branches of medicine and surgery – a book written in December 1996 by George M. Gould and Walter Lytle Pyle, surfaced the following top 100 relevant and important keywords. ['child', 'case', 'woman', 'year', 'one', 'birth', 'month', 'two', 'time', 'mother', 'instance', 'pregnancy', 'day', 'twin', 'fetus', 'menstruation', 'born', 'speaks', 'report', 'age', 'first', 'delivered', 'uterus', 'girl', 'found', 'three', 'female', 'died', 'well', 'death', 'living', 'old', 'record', 'mention', 'wife', 'give', 'hour', 'lived', 'pregnant', 'many', 'labor', 'gave', 'second', 'head', 'male', 'four', 'six', 'pound', 'also', 'several', 'inch', 'delivery', 'made', 'pain', 'say', 'married', 'man', 'left', 'body', 'abdomen', 'similar', 'five', 'husband', 'operation', 'account', 'boy', 'became', 'infant', 'following', 'history', 'another', 'ten', 'breast', 'without', 'great', 'much', 'example', 'placenta', 'father', 'cesarean', 'long', 'healthy', 'extrauterine', 'reported', 'penis', 'third', 'menstrual', 'last', 'discharge', 'cite', 'relates', 'seen', 'quite', 'week', 'bore', 'period', 'fact', 'describes', 'given', 'medical']

	Based on the above relevant and important keywords leads to the following conclusion: - the medical anatomy of a foetus, a female reproductive system and a male reproductive system - the biological and stages of the development of the foetus - the body weight of a baby at its birth - the women psychological experiences of physiological childbirth - the psychological and physiological experiences of menstruation - the physical, growth and psychologically development in girls and boys - Planning to have a child after marriage - Importance of a married man playing its role in accompanying his wife during and after the labouring process - Human development process from its birth to death

Section C [10 marks]

This will be a one-to-one interview during Lesson 9, for a duration of eight minutes.

- In the first half, you will present on your work in Sections A and B above.
- In the second half, the interviewer will ask you a series of questions.
- The objective is to test your level of understanding of the fundamental concepts and your ability to apply them. This includes proficiency of the relevant Python programming skills related to this module.
- More details about the interview will be announced nearer the date.

~~~ END OF COURSEWORK 1 (CW1) ~~~