Introduction to Statistical Methods for Data Science

Information Retrieval

Lecturer: Ade Shonola

Submitted By: Mariam Khalid

Student ID: 10076576

Coursework Type: Individual Assignment

Module Code: 7071CEM

Faculty of Engineering, Environment and Computing

Introduction

The exponential growth of technology and data has lead scientists to work on storing and retrieving required information in secure and fastest ways possible. Growing data also increasing textual information stored on web which can be accessed by anyone. Whether you are a student, engineer, scholar, scientist or a doctor, you go to internet and retrieve information on research publications, medical records, books, universities, biographies and much more.

The information is growing on each second pass. Which information is stored in what way and how can a person lookup a particular information was an important question in the past. Therefore, scientist worked on storing the information and developed traditional database systems which stored information and information was accessed in certain ways. But, with the passage of time, information scaled exponentially and those conventional systems failed. Researcher then, came up with information retrieval which is a technique to store information on web in optimal way and retrieve a particular and correct information in fastest possible way.

Christopher D. Manning defines information retrieval in his book **[1]** as IR is finding information which has a unstructured type based on the information need (query). IR is not restricted to unstructured only and goes beyond any definition. The field covers users in extracting and filtering information from the collection of documents, classification of documents and showing the results in ranked manner.

Over the passage of time, Information retrieval researchers and developers have created multiple models for storing and indexing the information and retrieving it. From Boolean retrieval to probabilistic retrieval, language based model, indexing of information, compression and score based retrieval everything got developed over time which is helping everyone in the fast pace environment we are now a days living in.

The general model for information retrieval has following components:

- Text operations
- Indexing
- Searching
- Ranking

The above list is not restricted to those only. The base model is shown in the following figure.

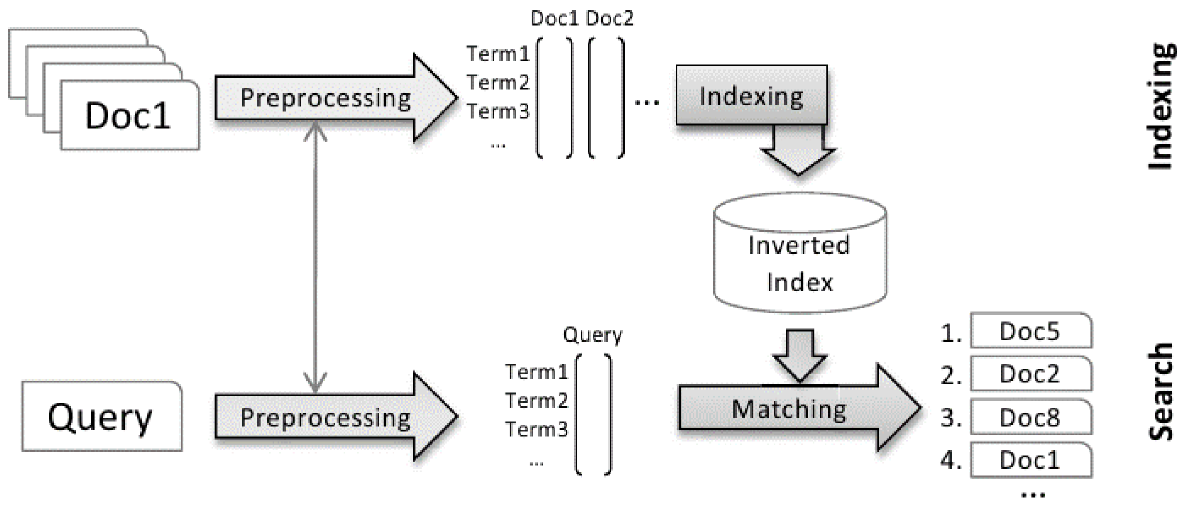


Figure 1: Base Information Retrieval Model. Reprinted from [2]

In this study, we are going to build the process from scratch to understand the how information retrieval work and implement the learnings from our course. As the above figure is showing, user enter the query for which required information is required. The information goes from indexer and systems shows the output the relevant information in a ranked fashion.

The first part of search engine is to have information indexed. We are required to crawl information from the web.

Crawler

Crawlers are the automated programs which goes to web pages and extract information and update the search engine's information. The mainstream search engine used these crawlers to extract information from every web page there is iteratively to update their indexed information so that when a new query related to those pages comes, the engine would be able to show the credible result.

In this study, we are going to crawl information of professors and supervisors from different fields and from different universities of United Kingdom for the students who wants to search for available supervisors for post-graduate study. We have the liberty to select universities and supervisors. We have selected three different universities and three different fields for students who wants to apply for post-graduate study. We will be crawling information from respective university websites and create our own search engine.

Universities and Faculty Names – (Requirement one)

List of universities **[5] [6] [7]** we have selected are as follow:

- Oxford Internet Institute – University of Oxford
- Coventry University
- Kingston University

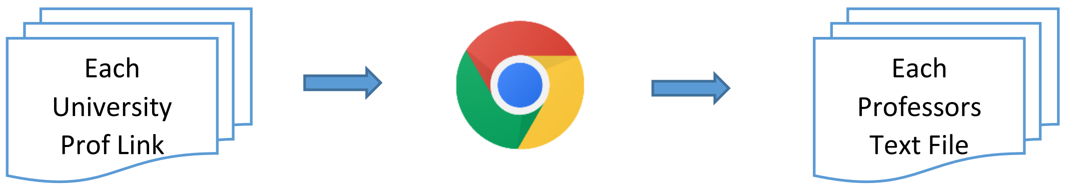
Selected faculties are as follow:

- Data Science
- Arts and Humanities
- Business, Finance and Logistics

Reason for selecting above universities is that each faculty staff members has a profile page hosted by university official website which includes biography of staff member, publications, current projects and research professor has done previously. This lead to have more textual information related to professors which will be helpful in our indexed information and our search engine will correctly identify the field and shows relevant professor against the search of student.

Reason for selecting three different fields is that we wanted to distinguish between how indexed store different information and present the result upon querying the words. We also wanted to rank the results so it was better to select distinct fields.

For Crawler we have used **Python** for implementing and crawling information for professors. We first manually identified professors from each university and field. Noted their profile links and as each university has a separate website structure so we had to define three different functions for crawling from each university. **Selenium** was used to crawling. Selenium is a library[1] which helps users to automate the interaction on the web browser. For our purpose we used Chrome browser for crawling.

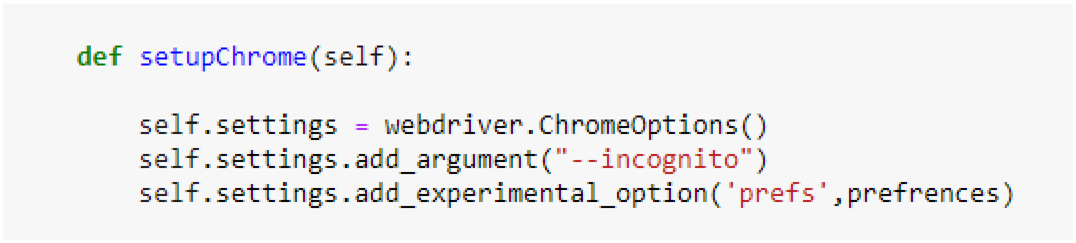


Each professor's is passed to a function, which then, automatically go that page and interact with the web page and goes to each link for research, publications and active projects and extract textual information and as output store the text file with text information for each professor.

[1] https://pypi.org/project/selenium/

For crawler application, we require chrome web drivers that can be downloaded from **[3].** The following steps were followed by the crawler application:

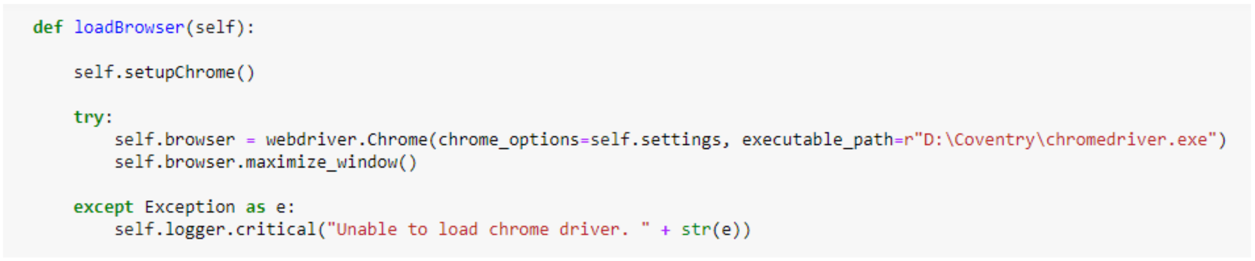
1. Downloading the Chrome driver
2. Setting up chrome



```python
def setupChrome(self):

    self.settings = webdriver.ChromeOptions()
    self.settings.add_argument("--incognito")
    self.settings.add_experimental_option('prefs',prefrences)
```

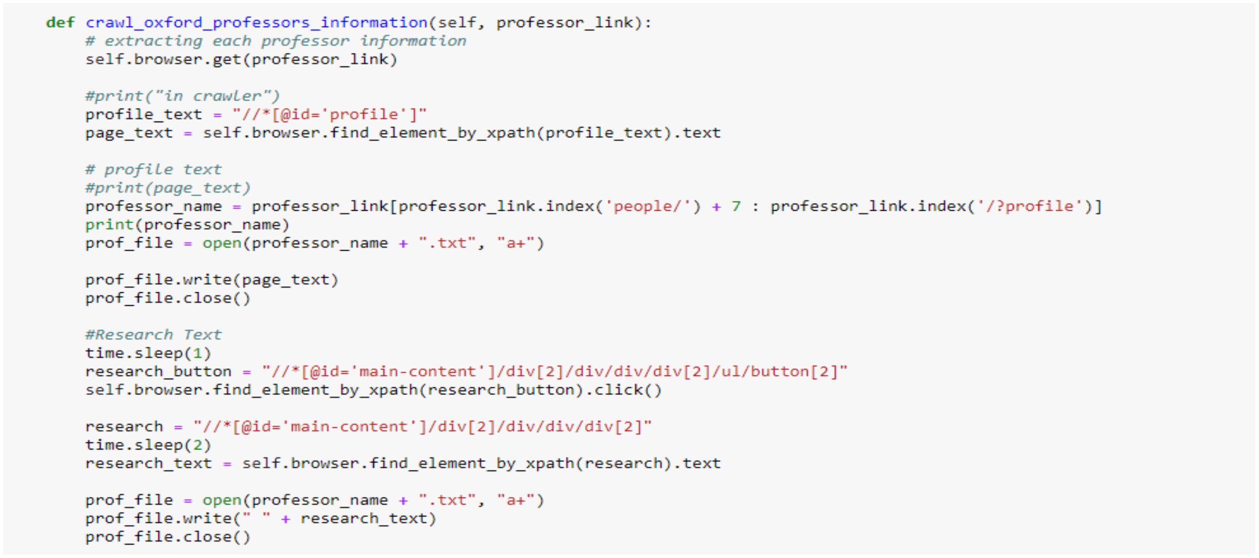
The above function setup the webdriver option to chrome and explicitly set the opening of page to private window option to handle cookies and permissions. Preferences are set for downloading option in which we set the folder where we want to store the output files.

3. Loading the Browser



```python
def loadBrowser(self):

    self.setupChrome()

    try:
        self.browser = webdriver.Chrome(chrome_options=self.settings, executable_path=r"D:\Coventry\chromedriver.exe")
        self.browser.maximize_window()

    except Exception as e:
        self.logger.critical("Unable to load chrome driver. " + str(e))
```

4. Call one university function by passing the professors links in arguments



```python
def crawl_oxford_professors_information(self, professor_link):
    # extracting each professor information
    self.browser.get(professor_link)

    #print("in crawler")
    profile_text = "//*[@id='profile']"
    page_text = self.browser.find_element_by_xpath(profile_text).text

    # profile text
    #print(page_text)
    professor_name = professor_link[professor_link.index('people/') + 7 : professor_link.index('/?profile')]
    print(professor_name)
    prof_file = open(professor_name + ".txt", "a+")

    prof_file.write(page_text)
    prof_file.close()

    #Research Text
    time.sleep(1)
    research_button = "//*[@id='main-content']/div[2]/div/div/div[2]/ul/button[2]"
    self.browser.find_element_by_xpath(research_button).click()

    research = "//*[@id='main-content']/div[2]/div/div/div[2]"
    time.sleep(2)
    research_text = self.browser.find_element_by_xpath(research).text

    prof_file = open(professor_name + ".txt", "a+")
    prof_file.write(" " + research_text)
    prof_file.close()
```

Similarly each university website function was called with their professor's links and crawler application went to those links and extracted particular information. We passed website **DOM** objects to locate and iterate with webpage.

Indexer (Requirement Two)

Indexer is a data structure which stores information in a hash like structure. Against each word as key it stores the occurrence of word in all documents as values. The search engine then can easily retrieve information quickly from the indexer and shows the result against each query. All the tokenized words are stored as keys and the value of that key contains the documents where it occurred and at which place so that the engine can easily identity which pages to retrieve for each word of query.

For this assignment, we created an index over the information we crawled for professors.

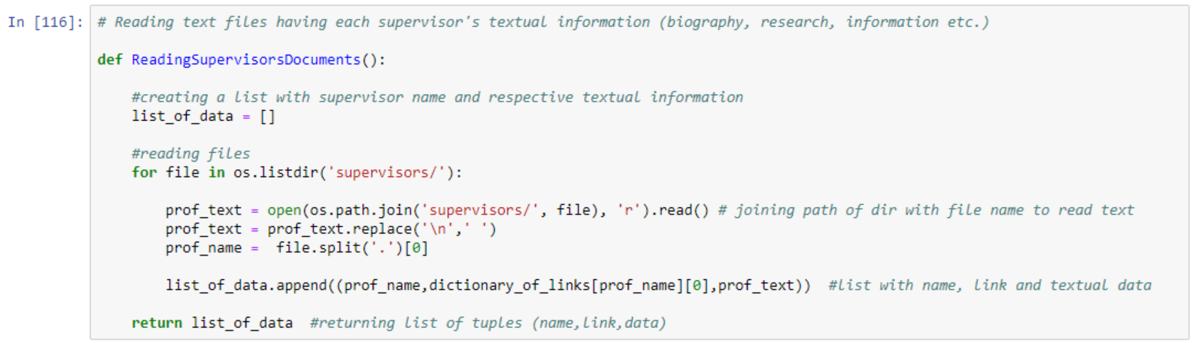
We have created two indexes a simple inverted index and a full inverted index. A simple inverted index is the one which stores word as key and value as its respective documents where it occurred. A full inverted index is the one in which words are stored as keys and values of those keys as documents where that key occurred and at which place they occurred.

Steps that we followed for creating those indexes are as follow:

1. Loading the required libraries

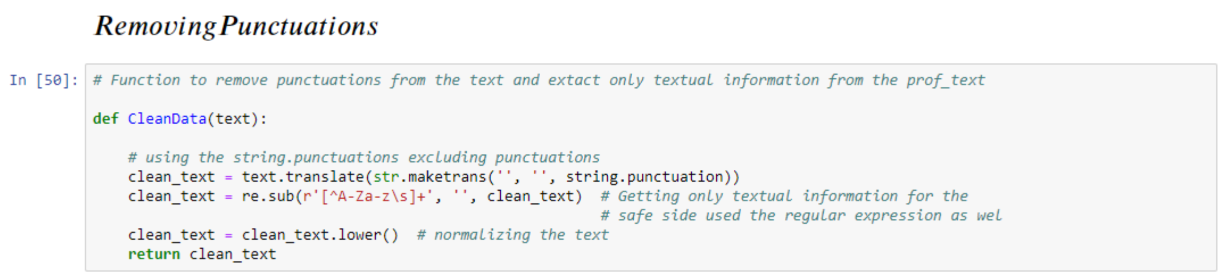
Libraries include the **nltk** as well from which we are extracting the stop words for our pre-processing function.

2. Creating a dictionary of professors with their respective profile link, their field and their university
3. Reading the supervisors documents



```
In [116]: # Reading text files having each supervisor's textual information (biography, research, information etc.)

          def ReadingSupervisorsDocuments():

              #creating a list with supervisor name and respective textual information
              list_of_data = []

              #reading files
              for file in os.listdir('supervisors/'):

                  prof_text = open(os.path.join('supervisors/', file), 'r').read() # joining path of dir with file name to read text
                  prof_text = prof_text.replace('\n',' ')
                  prof_name =  file.split('.')[0]

                  list_of_data.append((prof_name,dictionary_of_links[prof_name][0],prof_text))  #list with name, link and textual data

              return list_of_data  #returning list of tuples (name,link,data)
```

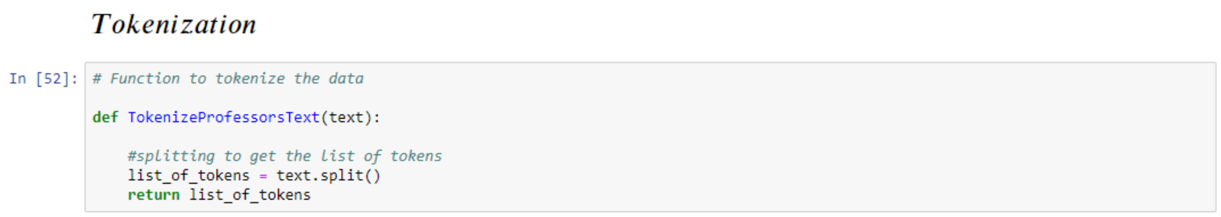
4. As the textual information included punctuations and digits and years. For removing and cleaning the text we wrote a separate function.



RemovingPunctuations

```
In [50]: # Function to remove punctuations from the text and extact only textual information from the prof_text

         def CleanData(text):

             # using the string.punctuations excluding punctuations
             clean_text = text.translate(str.maketrans('', '', string.punctuation))
             clean_text = re.sub(r'[^A-Za-z\s]+', '', clean_text)  # Getting only textual information for the
                                                                   # safe side used the regular expression as wel
             clean_text = clean_text.lower()  # normalizing the text
             return clean_text
```

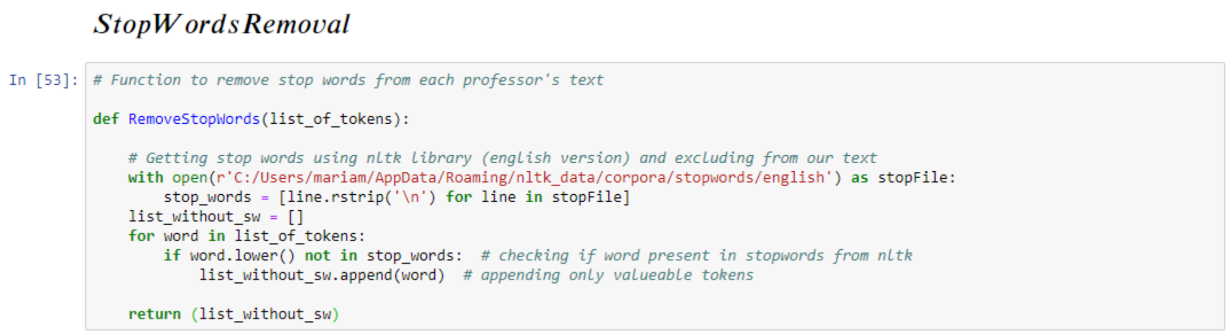
5. Tokenization is the process of creating tokens out of documents by splitting the text on space. We passed the textual information to a written function and it returned us the tokens.



Tokenization

```
In [52]: # Function to tokenize the data

         def TokenizeProfessorsText(text):

             #splitting to get the list of tokens
             list_of_tokens = text.split()
             return list_of_tokens
```

6. Avery text has stop words and that does not increase or decrease the content value. So for their removal we wrote a separate function. In that function we used the downloaded stop words file from nltk and filtered our tokens for each professor.



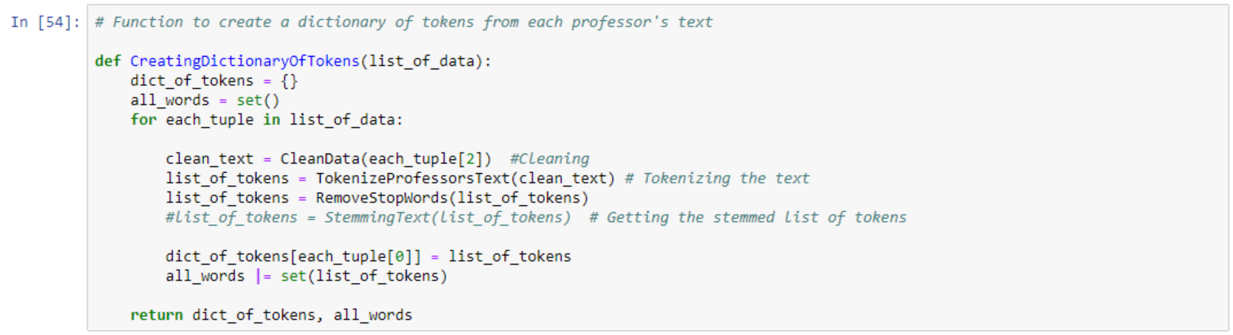
StopWordsRemoval

```
In [53]: # Function to remove stop words from each professor's text

         def RemoveStopWords(list_of_tokens):

             # Getting stop words using nltk library (english version) and excluding from our text
             with open(r'C:/Users/mariam/AppData/Roaming/nltk_data/corpora/stopwords/english') as stopFile:
                 stop_words = [line.rstrip('\n') for line in stopFile]
             list_without_sw = []
             for word in list_of_tokens:
                 if word.lower() not in stop_words:  # checking if word present in stopwords from nltk
                     list_without_sw.append(word)  # appending only valueable tokens

             return (list_without_sw)
```

7. Creating the dictionary of tokens against each professor's text and passing it through the pre-processing steps.
   a. Cleaning
   b. Tokenization

c. Removing stop words

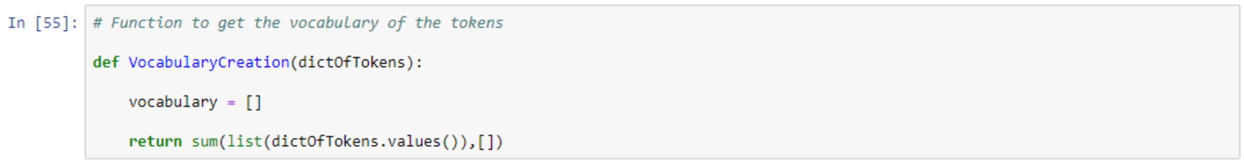
$CreatingDictionaryOfTokens$



```
In [54]: # Function to create a dictionary of tokens from each professor's text

         def CreatingDictionaryOfTokens(list_of_data):
             dict_of_tokens = {}
             all_words = set()
             for each_tuple in list_of_data:

                 clean_text = CleanData(each_tuple[2])  #Cleaning
                 list_of_tokens = TokenizeProfessorsText(clean_text) # Tokenizing the text
                 list_of_tokens = RemoveStopWords(list_of_tokens)
                 #list_of_tokens = StemmingText(list_of_tokens)  # Getting the stemmed list of tokens

                 dict_of_tokens[each_tuple[0]] = list_of_tokens
                 all_words |= set(list_of_tokens)

             return dict_of_tokens, all_words
```

8. Generating the vocabulary of all tokens that appeared in professor’s texts.

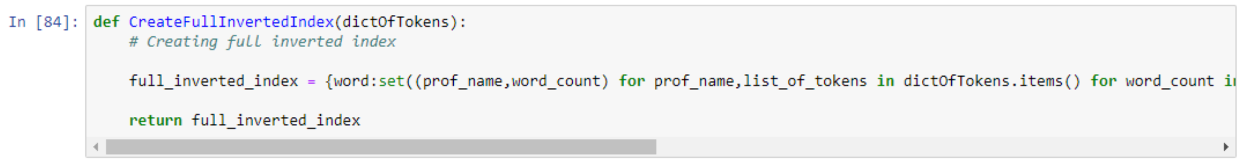
$GenerateVocabularyForTFidf$



```
In [55]: # Function to get the vocabulary of the tokens

         def VocabularyCreation(dictOfTokens):

             vocabulary = []

             return sum(list(dictOfTokens.values()),[])
```

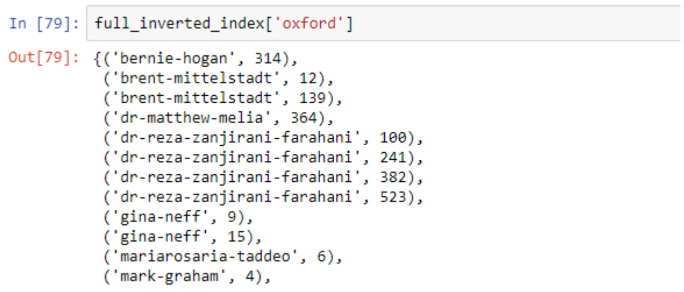
9. Creation of Full inverted index

$Full - Inverted - Index$



```
In [84]: def CreateFullInvertedIndex(dictOfTokens):
             # Creating full inverted index

             full_inverted_index = {word:set((prof_name,word_count) for prof_name,list_of_tokens in dictOfTokens.items() for word_count i

             return full_inverted_index
```

After calling each function above, we setup everything to create our full inverted index. Which is expected to show the professor’s files and the respective index of location where a particular word occurred given a word. The sample output of our full inverted index is as follow:

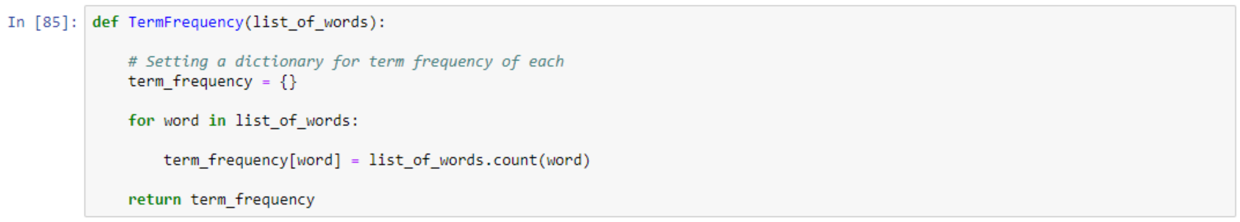


```
In [79]: full_inverted_index['oxford']

Out[79]: {('bernie-hogan', 314),
          ('brent-mittelstadt', 12),
          ('brent-mittelstadt', 139),
          ('dr-matthew-melia', 364),
          ('dr-reza-zanjirani-farahani', 100),
          ('dr-reza-zanjirani-farahani', 241),
          ('dr-reza-zanjirani-farahani', 382),
          ('dr-reza-zanjirani-farahani', 523),
          ('gina-neff', 9),
          ('gina-neff', 15),
          ('mariarosaria-taddeo', 6),
          ('mark-graham', 4),
```

10. Calculating the Term Frequency **[4]** for ranking purpose. Term frequency is the count of term in the document divided by total number of words in document as shown in equation below:

$$tf(t,d) = \frac{count\ of\ t\ in\ d}{number\ of\ words\ in\ d}$$

$$Term - Frequency$$



```
In [85]: def TermFrequency(list_of_words):

             # Setting a dictionary for term frequency of each
             term_frequency = {}

             for word in list_of_words:

                 term_frequency[word] = list_of_words.count(word)

             return term_frequency
```

11. Creating the inverted index and calculate the idf scores. IDF **[7]** is measure of informativeness of token t. It gives the relative weightage to each term. Which is defined as:

$$idf(t) = \log(\frac{N}{df + 1})$$

N: Number of corpus

t: term (token)

d: document (list of tokens)

df: document frequency

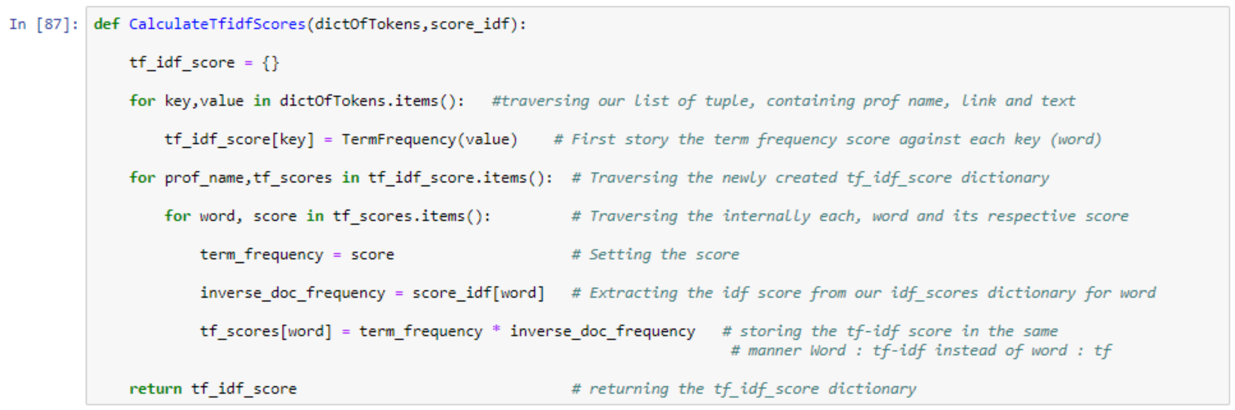
$$df\ = occuracne\ of\ t\ in\ documents$$

Finally, when we multiply the tf with idf we get the tf-idf score on the basis of which we will be ranking our search results.

$$tf - idf(t,d) = tf(t,d) * \log(N/(df + 1)$$

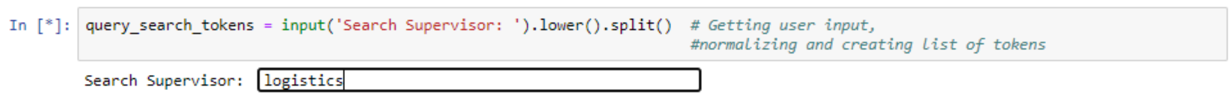
Added add one smoothing in case we miss the word in any document.

Calculating – Tfidf Scores


```
In [87]: def CalculateTfidfScores(dictOfTokens,score_idf):

             tf_idf_score = {}

             for key,value in dictOfTokens.items():   #traversing our list of tuple, containing prof name, link and text

                 tf_idf_score[key] = TermFrequency(value)    # First story the term frequency score against each key (word)

             for prof_name,tf_scores in tf_idf_score.items():  # Traversing the newly created tf_idf_score dictionary

                 for word, score in tf_scores.items():        # Traversing the internally each, word and its respective score

                     term_frequency = score                   # Setting the score

                     inverse_doc_frequency = score_idf[word]  # Extracting the idf score from our idf_scores dictionary for word

                     tf_scores[word] = term_frequency * inverse_doc_frequency   # storing the tf-idf score in the same
                                                                                 # manner Word : tf-idf instead of word : tf

             return tf_idf_score                              # returning the tf_idf_score dictionary
```

12. Query Component (Requirement three)

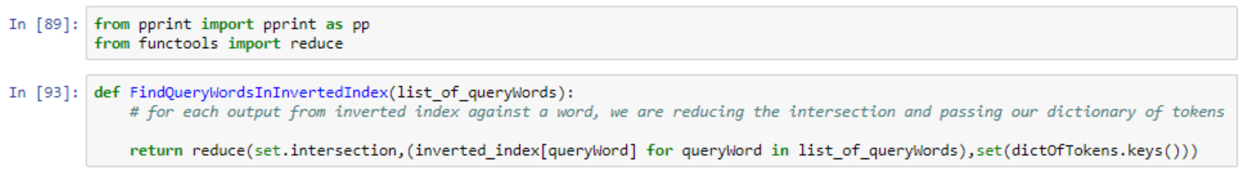
Getting – User – Input

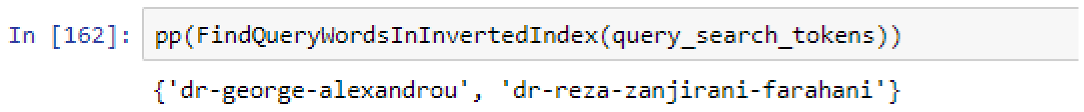

```
In [*]: query_search_tokens = input('Search Supervisor: ').lower().split()  # Getting user input,
                                                                            #normalizing and creating list of tokens

        Search Supervisor: logistics
```

User enters the query which get gets normalized and tokenized for pass to index and get result against it.

13. Basic search from the inverted index directly without ranking using the td-idf score is as follow

BasicSearch

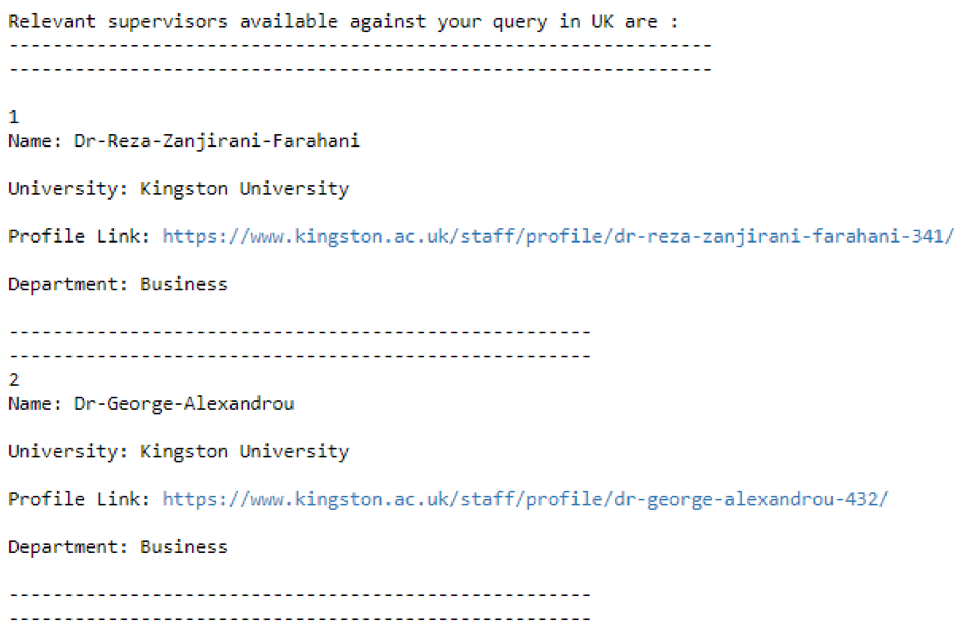

```
In [89]: from pprint import pprint as pp
         from functools import reduce

In [93]: def FindQueryWordsInInvertedIndex(list_of_queryWords):
             # for each output from inverted index against a word, we are reducing the intersection and passing our dictionary of tokens

             return reduce(set.intersection,(inverted_index[queryWord] for queryWord in list_of_queryWords),set(dictOfTokens.keys()))
```


```
In [162]: pp(FindQueryWordsInInvertedIndex(query_search_tokens))

          {'dr-george-alexandrou', 'dr-reza-zanjirani-farahani'}
```

User entered the word logistics, our inverted index has shown two professors names which had their experience in logistics and supply chain accurately. This is the inverted index result but we further took our study to ranking as well and implemented tf-idf relevance as shown above. The ranking code can be reviewed in the attached code in appendix as it is a bigger code block and to maintain is indentation we cannot paste it here.

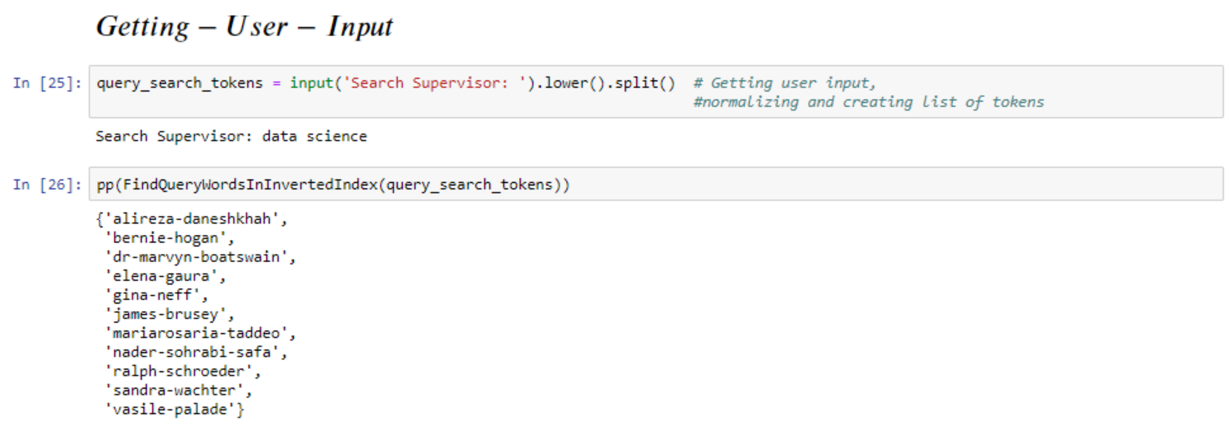
Ranked Results (Requirement Four)

For the same query ‘logistics’ our ranking implementation has shown results as follow:



```
Relevant supervisors available against your query in UK are :
---------------------------------------------------------------
---------------------------------------------------------------

1
Name: Dr-Reza-Zanjirani-Farahani

University: Kingston University

Profile Link: https://www.kingston.ac.uk/staff/profile/dr-reza-zanjirani-farahani-341/

Department: Business

----------------------------------------------------
----------------------------------------------------
2
Name: Dr-George-Alexandrou

University: Kingston University

Profile Link: https://www.kingston.ac.uk/staff/profile/dr-george-alexandrou-432/

Department: Business

----------------------------------------------------
----------------------------------------------------
```

Sample Output 2 Query = ‘Data science’



Getting – User – Input

```
In [25]: query_search_tokens = input('Search Supervisor: ').lower().split()  # Getting user input,
                                                                             #normalizing and creating list of tokens

         Search Supervisor: data science

In [26]: pp(FindQueryWordsInInvertedIndex(query_search_tokens))

         {'alireza-daneshkhah',
          'bernie-hogan',
          'dr-marvyn-boatswain',
          'elena-gaura',
          'gina-neff',
          'james-brusey',
          'mariarosaria-taddeo',
          'nader-sohrabi-safa',
          'ralph-schroeder',
          'sandra-wachter',
          'vasile-palade'}
```

Simple inverted result can be seen in the above results. Ranked results for query ‘data science’ is as follow (over the next page). Our search engine has accurately ranked all the professors and as per the requirement we have provided the profile link of professor so that student can directly go their page and ask for meeting time and discuss their research proposal with them and apply for positions.

```
Relevant supervisors available against your query in UK are:
----------------------------------------------------------------
----------------------------------------------------------------

1
Name: Brent-Mittelstadt

University: OII-University of Oxford

Profile Link: https://www.oii.ox.ac.uk/people/brent-mittelstadt/?profile

Department: Data Science

tf-idf Score: 9.994601148099505

-----------------------------------------------------
-----------------------------------------------------
2
Name: Gina-Neff

University: OII-University of Oxford

Profile Link: https://www.oii.ox.ac.uk/people/gina-neff/?profile

Department: Data Science

tf-idf Score: 8.026371639393075

-----------------------------------------------------
-----------------------------------------------------
3
Name: Alireza-Daneshkhah

University: Coventry University

Profile Link: https://pureportal.coventry.ac.uk/en/persons/alireza-daneshk
hah

Department: Data Science

tf-idf Score: 7.116543037270774

-----------------------------------------------------
-----------------------------------------------------
4
Name: Ralph-Schroeder

University: OII-University of Oxford

Profile Link: https://www.oii.ox.ac.uk/people/ralph-schroeder/?profile
```

```
Department: Data Science

tf-idf Score: 6.886718478935268

-----------------------------------------------------
-----------------------------------------------------
5
Name: Sandra-Wachter

University: OII-University of Oxford

Profile Link: https://www.oii.ox.ac.uk/people/sandra-wachter/?profile

Department: Data Science

tf-idf Score: 4.6327216234416

-----------------------------------------------------
-----------------------------------------------------
6
Name: Vasile-Palade

University: Coventry University

Profile Link: https://pureportal.coventry.ac.uk/en/persons/vasile-palade

Department: Data Science

tf-idf Score: 4.338513331076667

-----------------------------------------------------
-----------------------------------------------------
7
Name: Bernie-Hogan

University: OII-University of Oxford

Profile Link: https://www.oii.ox.ac.uk/people/bernie-hogan/?profile

Department: Data Science

tf-idf Score: 3.9975328248724504

-----------------------------------------------------
-----------------------------------------------------
8
Name: Mark-Graham

University: OII-University of Oxford

Profile Link: https://www.oii.ox.ac.uk/people/mark-graham/?profile

Department: Data Science
```

```
tf-idf Score: 3.695261203509037

-----------------------------------------------------
-----------------------------------------------------
9
Name: James-Brusey

University: Coventry University

Profile Link: https://pureportal.coventry.ac.uk/en/persons/james-brusey

Department: Data Science

tf-idf Score: 3.296357757788301

-----------------------------------------------------
-----------------------------------------------------
10
Name: Mariarosaria-Taddeo

University: OII-University of Oxford

Profile Link: https://www.oii.ox.ac.uk/people/mariarosaria-taddeo/?profile

Department: Data Science

tf-idf Score: 2.1952817772541087

-----------------------------------------------------
-----------------------------------------------------
11
Name: Dr-Marvyn-Boatswain

University: Kingston University

Profile Link: https://www.kingston.ac.uk/staff/profile/dr-marvyn-boatswain
-126/

Department: Business

tf-idf Score: 2.0393327732404694

-----------------------------------------------------
-----------------------------------------------------
12
Name: Elena-Gaura

University: Coventry University

Profile Link: https://pureportal.coventry.ac.uk/en/persons/elena-gaura

Department: Data Science
```

```
tf-idf Score: 1.5319892819486682

-----------------------------------------------------
-----------------------------------------------------
13
Name: Ye-Liu

University: Coventry University

Profile Link: https://pureportal.coventry.ac.uk/en/persons/ye-liu

Department: Data Science

tf-idf Score: 1.5059350588472984

-----------------------------------------------------
-----------------------------------------------------
14
Name: Nader-Sohrabi-Safa

University: Coventry University

Profile Link: https://pureportal.coventry.ac.uk/en/persons/nader-sohrabi-s
afa

Department: Data Science

tf-idf Score: 1.372724824019593

-----------------------------------------------------
-----------------------------------------------------
15
Name: Dr-Mariacutea-Menciacutea

University: Kingston University

Profile Link: https://www.kingston.ac.uk/staff/profile/dr-mariacutea-menci
acutea-629/

Department: Arts

tf-idf Score: 0.9306502443451371

-----------------------------------------------------
-----------------------------------------------------
```

References

[1] Cambridge, U.P., 2009. Introduction to information retrieval.

[2] Alshari, E.M., 2015. Semantic arabic information retrieval framework. arXiv preprint arXiv:1512.03165.

[3] "Downloads - ChromeDriver - WebDriver for Chrome". (2020). [online], available from https://chromedriver.chromium.org/downloads [Accessed 26 July 2020]

[4] Scott, W. (2020). "TF-IDF for Document Ranking from scratch in python on real world dataset". [Online] available from https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089 [Accessed on 26 July 2020]

[5] Find Profiles — Coventry University. (2020). [online], available from https://pureportal.coventry.ac.uk/en/persons/ [Accessed 27 July 2020]

[6] OII | People — Oxford Internet Institute. (2020). [online], available from https://www.oii.ox.ac.uk/people/ [Accessed 27 July 2020]

[7] Robertson, S. (2004). "Understanding inverse document frequency": on theoretical arguments for IDF. *Journal of documentation*.

Appendices

Github Repository Link: https://github.com/mariamkhalid341/InvertedIndex

Crawler code: https://github.com/mariamkhalid341/InvertedIndex/blob/master/Crawler.ipynb

Inverted Index Code:
https://github.com/mariamkhalid341/InvertedIndex/blob/master/InvertedIndex.ipynb

Selenium (Python) Link: https://pypi.org/project/selenium/

Chrome Driver Link: https://chromedriver.chromium.org/downloads (download as per your chrome version)

Stop words download: `import nltk` and download using = `nltk.download('stopwords')`