INFORMATION RETRIVAL (7071CEM)

COURSE WORK

INFORMATION RETRIVAL (7071CEM)

12135637

1. ABSTRACT

This report mainly focuses on the core concepts of information retrieval namely building a search engine and building a document clustering model. For web crawling, scrapy and Nltk library were majorly used to do the crawling and the preprocessing for task one and NumPy, pandas and Sklearn libraries were also utilized in task two. The dataset used for clustering was taken from Kaggle and the website used for crawling was Coventry university's SEFA research webpage.

For an interactive search we have also utilized flask framework to build up an UI for the user and styled it with the help of HTML and CSS.

Keywords – NumPy, Pandas, Sklearn, Scrapy, Web crawling, Flask, HTML, CSS, Clustering, Nltk.

TABLE OF CONTENTS

2. INTRODUCTION

This report is written as a part of information retrieval module. This report contains all the necessary steps involved on building a smart web crawler and use the outputs to run a search engine. The task 2 involves on building a K-means clustering model for document clustering.

The text editors used to achieve the results were google collab and VScode, google collab was mainly used in task two and for task 1 VScode was uses to build an app and host the web search engine

One of the key factors of this course work is to gain deep knowledge on what goes inside a search engine and a clustering model. The steps involved to do so are:

1) Use Scrapy to crawl over the website
2) Store the results and preprocess the crawled data
3) Perform inverse indexing
4) Perform query search
5) Host the final search engine using flask framework
6) Finally create a clustering model for document classification

TASK - I

3. Understanding Scrapy

Scrapy is a Python-based web crawling platform that is freeware. It was originally intended for web scraping, but it may also be used to collect data via APIs or as a general-purpose web crawler. Upon using scrapy for web crawling it was found the the newer versions by default respect the robots.txt file therefor we don't have to worry about our crawler being very aggressive.

We start by generating an initial request to crawl the URL and specify an callback function to be called when the response is downloaded. In the call back function, we parse the webpage and return the items and objects and then with the call back function we can pass the page contents using selectors. Finally, the items returned from the spider is typically stored in a data base in our case as an csv file. ("Spiders — Scrapy 2.6.2 documentation", 2022)

3.1 Building the Spider

To build an efficient spider, we need to know what we are trying to achieve by crawling the URL. In this case we will require the project title, authors name, authors page link and date of publication. Upon inspecting all the required elements from the websites HTML and can build a class for our spider to loop through the page and get required elements needed by doing this we can save a lot of time by crawling the reverent elements rather than crawling everything.

3.2 Pagination

Building a crawler and retrieving elements from a single page is straight forward, we also face the challenge of crawling every single page related to the website therefore we do pagination. we construct an IF statement when the page is not empty the crawler will continue to crawl on the other hand when the page is empty the crawler will stop crawling the save the retrieved outputs.

3.3 Storing the Retrieved Data

The retrieved data is store in a csv file in the local system which is easy to access the data for our searching, this data needs to undergo preprocessing for our search engine to work the next section of the report covers the steps involved in preprocessing the retried data.

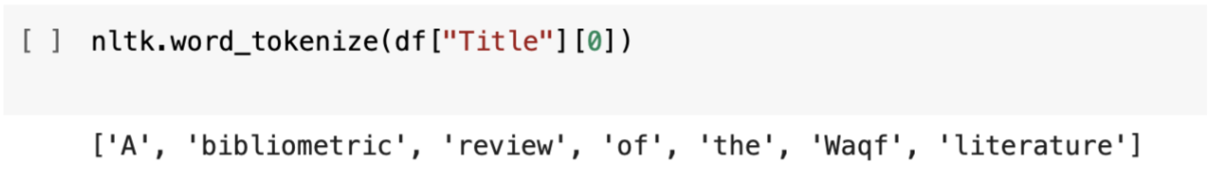
4. Data Preprocessing

One of the major components on building an effective search engine is to preprocess the data and there are several methods and techniques needed to be deployed on the crawled data. The library used to do the preprocessing is Nltk.

NOTE: preprocessing should be done for both the crawled data and the users query input.

4.1 Tokenizing data

The term tokenizing means splitting a phrase, sentence, or a paragraph. the entire text is converted into smaller pieces or units. Each of these smaller units are called tokens. This was achieved by using Nltk tokenizer package.

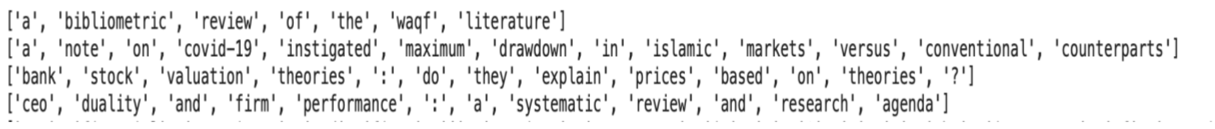


```
[ ]  nltk.word_tokenize(df["Title"][0])

     ['A', 'bibliometric', 'review', 'of', 'the', 'Waqf', 'literature']
```

Figure 1Tokenizing data

4.2 Lower casing

One of the important steps involved in the preprocessing step is converting the tokenized words into lower case this is very vital because when we remove stop words or while doing stemming the packages requires the words to be lower there, we convert into lower case.

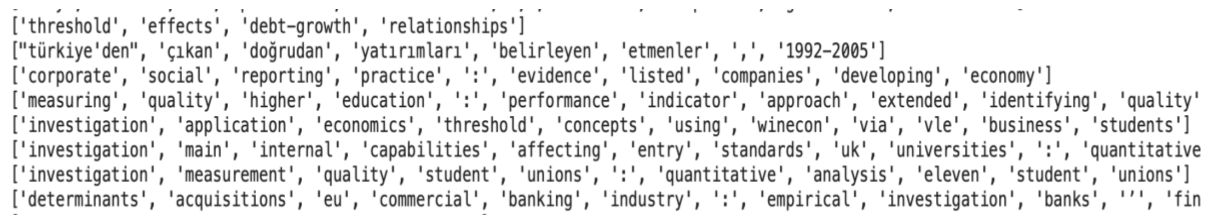


```
['a', 'bibliometric', 'review', 'of', 'the', 'waqf', 'literature']
['a', 'note', 'on', 'covid-19', 'instigated', 'maximum', 'drawdown', 'in', 'islamic', 'markets', 'versus', 'conventional', 'counterparts']
['bank', 'stock', 'valuation', 'theories', ':', 'do', 'they', 'explain', 'prices', 'based', 'on', 'theories', '?']
['ceo', 'duality', 'and', 'firm', 'performance', ':', 'a', 'systematic', 'review', 'and', 'research', 'agenda']
```

Figure 2 Lower casing

4.3 Removing Stop Words

In this step we intend to remove the stop words and punctuation like (a, is, for,", >) from the data this will make our life much easier while searching or while creating a search engine. Removing these words will help us for a better search and quick results.

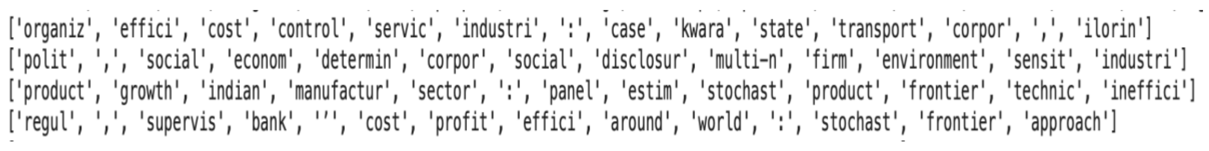


```
['threshold', 'effects', 'debt-growth', 'relationships']
["türkiye'den", 'çıkan', 'doğrudan', 'yatırımları', 'belirleyen', 'etmenler', ',', '1992-2005']
['corporate', 'social', 'reporting', 'practice', ':', 'evidence', 'listed', 'companies', 'developing', 'economy']
['measuring', 'quality', 'higher', 'education', ':', 'performance', 'indicator', 'approach', 'extended', 'identifying', 'quality'
['investigation', 'application', 'economics', 'threshold', 'concepts', 'using', 'winecon', 'via', 'vle', 'business', 'students']
['investigation', 'main', 'internal', 'capabilities', 'affecting', 'entry', 'standards', 'uk', 'universities', ':', 'quantitative
['investigation', 'measurement', 'quality', 'student', 'unions', ':', 'quantitative', 'analysis', 'eleven', 'student', 'unions']
['determinants', 'acquisitions', 'eu', 'commercial', 'banking', 'industry', ':', 'empirical', 'investigation', 'banks', '’', 'fin
```

Figure 3 Removing stop words

4.4 Stemming

Improving the search and the search engine has been the main motivation behind the preprocessing, stemming is one major part of preprocessing. Stemming is the process of reducing the inflection to their root meaning this will give us the singular form of the word by still preserving the meaning of the word. After preprocessing the data, we intend to access the document much quicker which bring us to the next process of building a search engine.



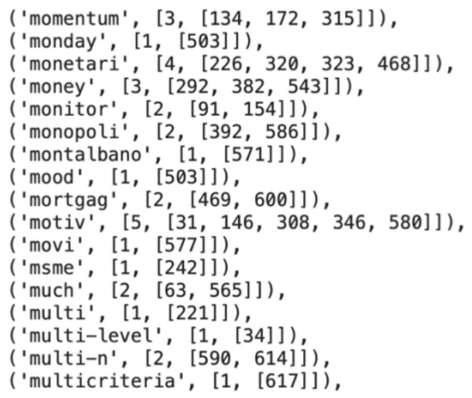
```
['organiz', 'effici', 'cost', 'control', 'servic', 'industri', ':', 'case', 'kwara', 'state', 'transport', 'corpor', ',', 'ilorin']
['polit', ',', 'social', 'econom', 'determin', 'corpor', 'social', 'disclosur', 'multi-n', 'firm', 'environment', 'sensit', 'industri']
['product', 'growth', 'indian', 'manufactur', 'sector', ':', 'panel', 'estim', 'stochast', 'product', 'frontier', 'technic', 'ineffici']
['regul', ',', 'supervis', 'bank', '’', 'cost', 'profit', 'effici', 'around', 'world', ':', 'stochast', 'frontier', 'approach']
```

Figure 4 Stemming the document

5. Inverse Indexing

The process of inverse indexing is pretty straight forward but a bit tricky to implement on a actual data. Our indexer will loop through each word and takes a note of number of times the word has occurred which is the document frequency which will be the first part of our list

The second part our list consists of all the places the document has occurred this will make our search much faster and easier.



```
('momentum', [3, [134, 172, 315]]),
('monday', [1, [503]]),
('monetari', [4, [226, 320, 323, 468]]),
('money', [3, [292, 382, 543]]),
('monitor', [2, [91, 154]]),
('monopoli', [2, [392, 586]]),
('montalbano', [1, [571]]),
('mood', [1, [503]]),
('mortgag', [2, [469, 600]]),
('motiv', [5, [31, 146, 308, 346, 580]]),
('movi', [1, [577]]),
('msme', [1, [242]]),
('much', [2, [63, 565]]),
('multi', [1, [221]]),
('multi-level', [1, [34]]),
('multi-n', [2, [590, 614]]),
('multicriteria', [1, [617]]),
```

Figure 5 Inverse indexing and sorting

5.1 Sorting

After all the preprocessing the steps and indexing the documents will need to sort it into an order this will again help in improving the efficiency of the search like how the words are sorted in a dictionary.
This concludes our crawler which now is fully prepared to crawl through any given link.

6. Querying

After building and crawler we need to query any text and get the relevant outputs from our query for building a query processer we follow the same preprocessing methos which are followed for building our crawler, therefor when the query is give the query is preprocessed, sorted and inverse indexed, with these two documents we can do an intersection to find the relevant search document. The practical outputs will be displayed on the VIVA video.

7. Building Search Interface

Although we have successfully built a search engine which can give us optimal results. We also intend to build an interactive search UI will will give us outputs which are like google scholar. The steps involved to build an UI are shown in the upcoming sections.

7.2 Creating a Python Script

The contents of our python script are our spider class which crawls the web page upon called. Therefor this python document only contains our spider. After crawling the data is store in VS code as a csv file.

7.3 Query Processing Python Script

This scrip only contains the preprocessing steps which are discussed above therefor when the app is called the scrip automatically preprocess the data and inverse index and sort the document.

7.4 Creating a Flask App

With the help of flask, we can host our search engine, the flask app needs to be routed to the home and the results needs to be store in the app and the query part needs to be in the app. We use jinja template in flask to run the python code inside the html which we have created.

7.5 Creating and HTML and CSS File

Creating a HTML file which has an input dialogue box and a button. Upon clicking the button out app will compare the crawled data with the query data and give out the relevant results in the next page.

We use CSS to style the page like google scholar and order the result outputs for the end user to be accessible. The template was taken from bootstrap and implemented on our html file.

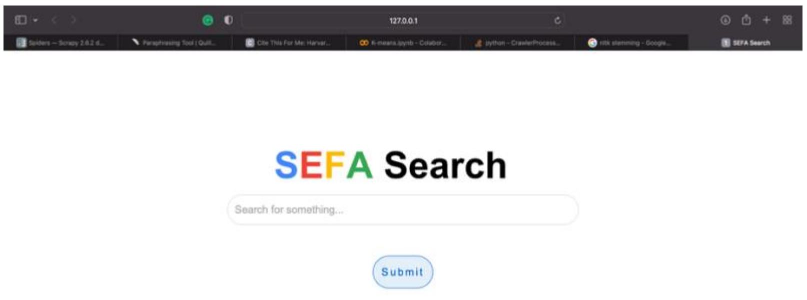
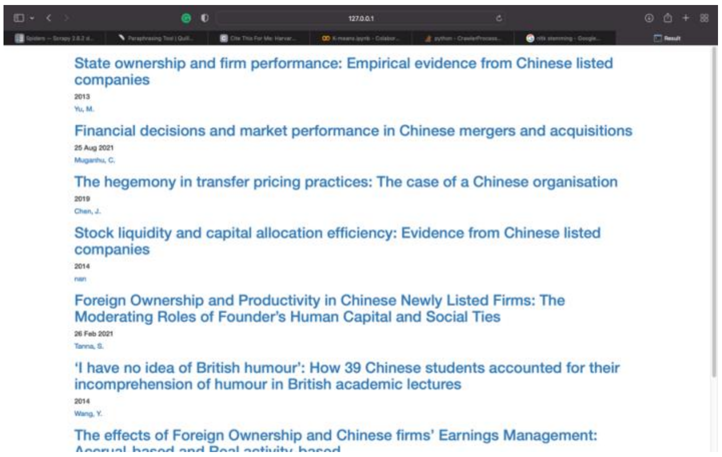


Figure 6 Search Interface



State ownership and firm performance: Empirical evidence from Chinese listed companies
2013
Yu, M.

Financial decisions and market performance in Chinese mergers and acquisitions
25 Aug 2021
Muganhu, C.

The hegemony in transfer pricing practices: The case of a Chinese organisation
2019
Chen, J.

Stock liquidity and capital allocation efficiency: Evidence from Chinese listed companies
2014
nan

Foreign Ownership and Productivity in Chinese Newly Listed Firms: The Moderating Roles of Founder's Human Capital and Social Ties
26 Feb 2021
Tanna, S.

'I have no idea of British humour': How 39 Chinese students accounted for their incomprehension of humour in British academic lectures
2014
Wang, Y.

The effects of Foreign Ownership and Chinese firms' Earnings Management: Accrual-based and Real activity-based

Figure 7 Search result interface

Sub Conclusion:

We have successfully built a search engine which can effectively search the given query and the user interface was bult for more interactive search. Although our search engine can give good results it lacks ranking the outputs based on relevancy.

TASK - II

8. K- Mean clustering

K – means clustering is one of the simplest and the fastest way of unsupervised learning Unsupervised algorithms often make inferences from datasets using just input vectors and without reference to known, or labelled, outcomes. k-means clustering is a vector quantization approach derived from signal processing that seeks to splits observations into k clusters, with each observation belonging to the cluster with the nearest mean, which serves as the cluster's prototype. A cluster can be represented with collection of several data points aggravated together with certain similarities. We also calculate the inertia of each cluster which is done by calculating the sum of distance of all points inside the cluster from the center of the cluster, the values will we discussed further in the report

Defining the k value can be sometimes challenging we can use elbow method to determine the number of clusters. Each data point is assigned to a cluster by minimizing the in-cluster sum of squares. To put it another way, the K-means algorithm determines k centroids and then assigns every data point to the closest cluster while keeping the centroids as tiny as feasible. To analyze the learning data, the K-means method in data mining begins with a first set of randomly picked centroids, which serve as the starting points for each cluster, and then performs iterative (repetitive) computations to optimize the centroids' locations. ("Understanding K-means Clustering in Machine Learning", 2022)

It stops creating and optimizing clusters when:

- Because the clustering was effective, the centroids have stabilized — there has been no change in their values.
- The maximum number of iterations has been reached.

In this report we are going to implement k means clustering for different cluster count and compare the accuracy and check the cluster words.

8.2 Dataset

In this section we are going to discuss about the dataset. Exploring datasets and understanding is one of the key elements on running a successful machine learning algorithm, the dataset consists of 5 columns and 592 rows although the dataset is big, we have shortlisted sports, politics, and wellbeing categories

Column name	Data Type	Description
Category	string	The column consists of which category the news article belongs
headlines	string	This column has the headlines of the news
Short description	string	A short description of the news article
keywords	string	Few key words describing the news article

Table1: Dataset

The dataset has a lots of string objects. Working with string objects can be tricky therefor we need to vectorize them before implementing it to our model

8.3 Implementation

We begin my importing necessary libraries into our environment. We have use google collab to implement our clustering and the libraries used are taken from Sklearn for unsupervised machine learning, matplotlib for plotting, pandas and NumPy for working with the data.

8.4 Data preprocessing

like other machine learning algorithms like regression or clustering we need to do some data preprocessing in our case we need to vectorize the feature that is in simple terms we need to convert the string into separate words and do indexing as well.

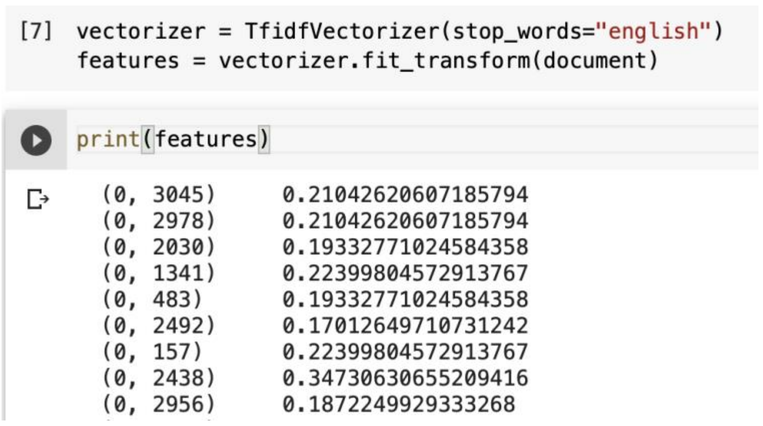
8.4.1 TFidfVectorizer

The work of the TF-IDF is to convert the words into number thus is done by finding the document frequency and the place where the word has accrued, we also do TF-IDF to do ranking this can be vital when building a search engine or a search engine or doing clustering. the formula for TF-IDF is

$$tf - idf(t,d) = tf(t,d) * idf(t)$$

$$idf(t) = \log\left[\frac{n}{df(t)}\right] + 1$$

The outputs after applying TF-IDF looks something like this we also take oyt stop words which are unnecessary words



```
[7] vectorizer = TfidfVectorizer(stop_words="english")
    features = vectorizer.fit_transform(document)
```

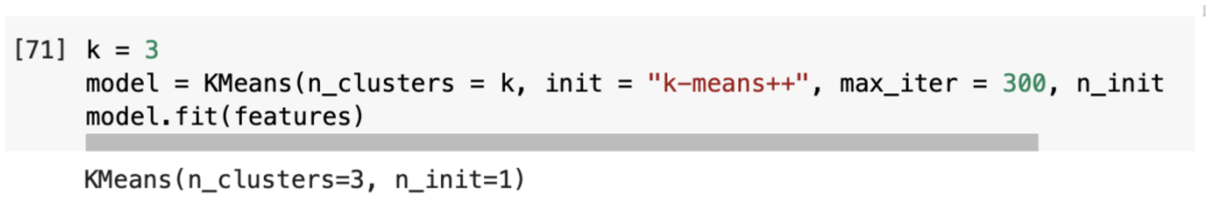
```
print(features)
```

```
(0, 3045)     0.21042620607185794
(0, 2978)     0.21042620607185794
(0, 2030)     0.19332771024584358
(0, 1341)     0.22399804572913767
(0, 483)      0.19332771024584358
(0, 2492)     0.17012649710731242
(0, 157)      0.22399804572913767
(0, 2438)     0.34730630655209416
(0, 2956)     0.1872249929333268
```

Figure 8 TF-IDF Output

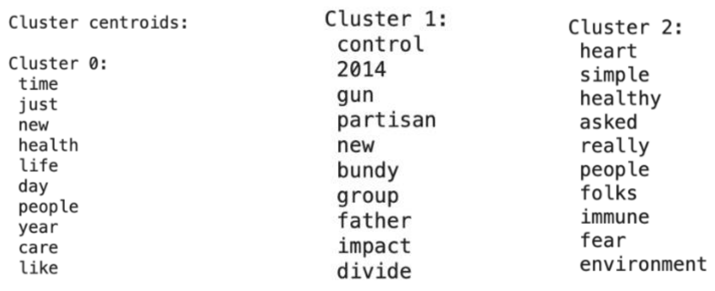
8.5 Clustering

We first create a model with 3 clusters(k=3) and check the outputs for the created clusters. We also use k-means++ for plotting the random points



```
[71] k = 3
     model = KMeans(n_clusters = k, init = "k-means++", max_iter = 300, n_init
     model.fit(features)

     KMeans(n_clusters=3, n_init=1)
```

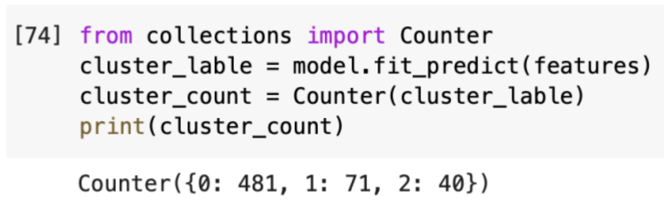
Upon fitting the model, we create the 3 different clusters separately and put them into csv file. Some frequent words in the 3 clusters are:



```
Cluster centroids:          Cluster 1:          Cluster 2:
                             control             heart
Cluster 0:                   2014                simple
 time                        gun                 healthy
 just                        partisan            asked
 new                         new                 really
 health                      bundy               people
 life                        group               folks
 day                         father              immune
 people                      impact              fear
 year                        divide              environment
 care
 like
```

Figure 9 Predominant word in 3 different clusters

We can also count the number of words in a certain cluster by a simple code which will give us better understanding of the cluster content refer the image below



```
[74] from collections import Counter
     cluster_lable = model.fit_predict(features)
     cluster_count = Counter(cluster_lable)
     print(cluster_count)

     Counter({0: 481, 1: 71, 2: 40})
```

Figure 10 Cluster count for 3 clusters

The result summary of the created 3 cluster model is represented in the below table

K- Means			
K	3		
Cluster	0	1	2
Cluster count	481	71	40
Results			
Accuracy		0.25	
Adjusted Rand- Index		0.119	
V-measure		0.121	
Completeness		0.132	
Homogeneity		0.112	
Inertia		574.4710	

Table 2 Result summary table for 3 clusters

As we can see for the created three clusters, we the find the accuracy very low and the cluster inertia is very high this is a clear indication that creating 3 clusters in not a good idea.

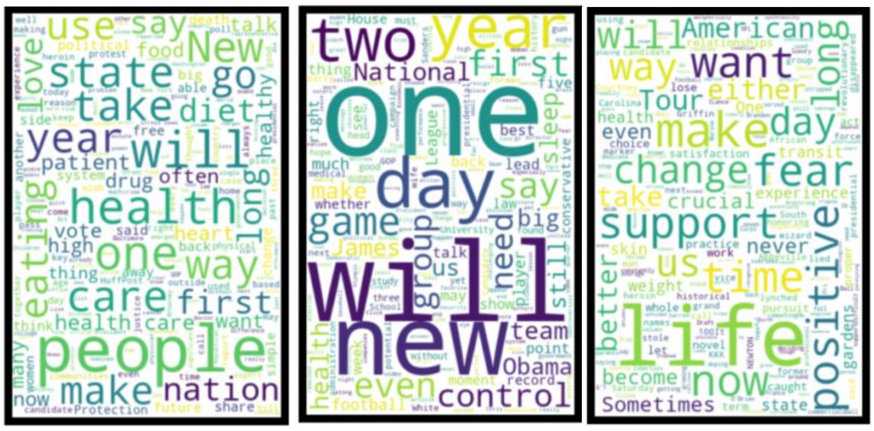
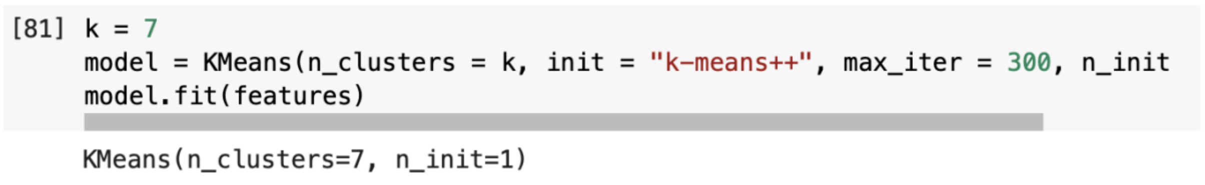


Figure 11 Word Cloud for 3 Clusters

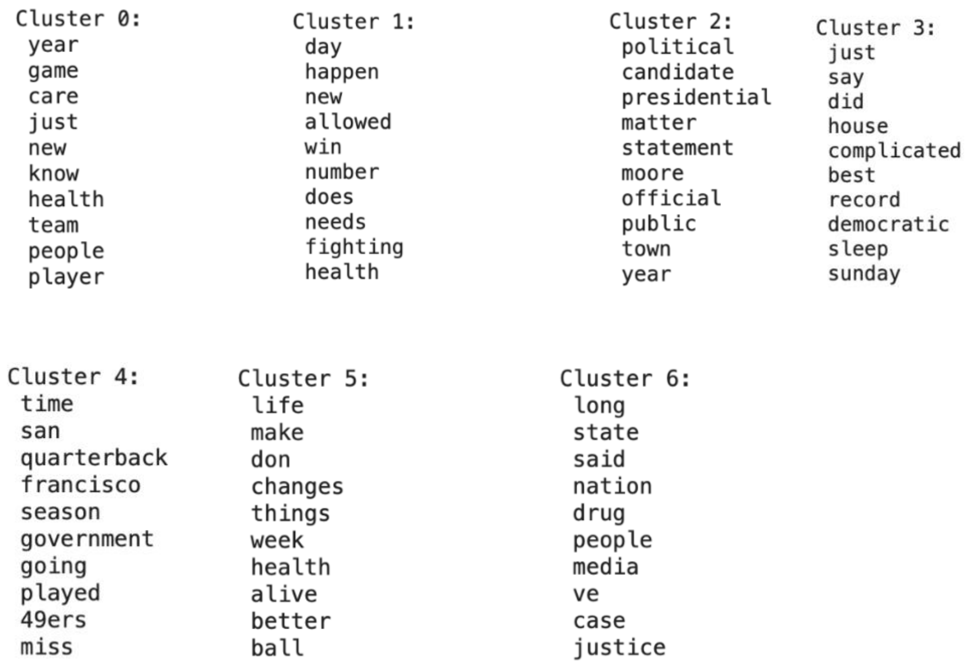
For our second trail we create a model with 7 clusters and check weather increasing cluster can improve the model.



```
[81] k = 7
     model = KMeans(n_clusters = k, init = "k-means++", max_iter = 300, n_init
     model.fit(features)

     KMeans(n_clusters=7, n_init=1)
```

Figure 12 A 7 cluster model

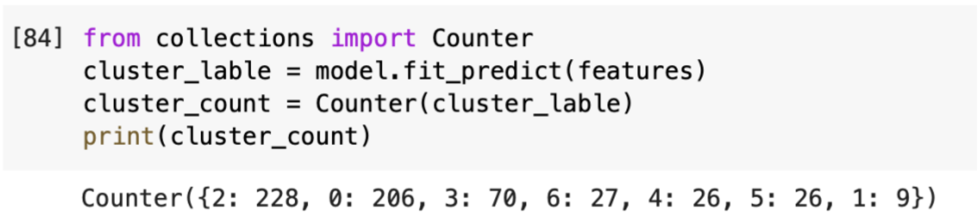
We do the similar process for the 7-cluster model we find the frequent words.



```
Cluster 0:          Cluster 1:        Cluster 2:       Cluster 3:
 year                day               political        just
 game                happen            candidate        say
 care                new               presidential     did
 just                allowed           matter           house
 new                 win               statement        complicated
 know                number            moore            best
 health              does              official         record
 team                needs             public           democratic
 people              fighting          town             sleep
 player              health            year             sunday

Cluster 4:          Cluster 5:        Cluster 6:
 time                life              long
 san                 make              state
 quarterback         don               said
 francisco           changes           nation
 season              things            drug
 government          week              people
 going               health            media
 played              alive             ve
 49ers               better            case
 miss                ball              justice
```

Figure 13 Frequent words in 7 clusters

Finding the cluster count for each cluster



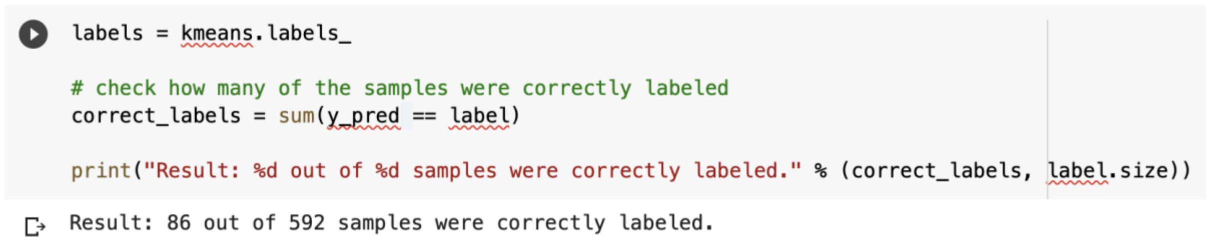
```
[84] from collections import Counter
     cluster_lable = model.fit_predict(features)
     cluster_count = Counter(cluster_lable)
     print(cluster_count)

     Counter({2: 228, 0: 206, 3: 70, 6: 27, 4: 26, 5: 26, 1: 9})
```

Figure 14 Image showing the cluster count

The result summary for the 7-cluster model

K- Means							
K	7						
Cluster	0	1	2	3	4	5	6
Cluster count	206	9	228	70	26	26	27
Results							
Accuracy	0.15						
Adjusted Rand- Index	0.312						
V-measure	0.273						
Completeness	0.268						
Homogeneity	0.278						
Inertia	574.4710						

Table3 Result summary for 7 cluster model



```
labels = kmeans.labels_

# check how many of the samples were correctly labeled
correct_labels = sum(y_pred == label)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, label.size))

Result: 86 out of 592 samples were correctly labeled.
```

Figure 15 Image cluster checking

It is quite evident that the 7-cluster model is not good enough as the inertia is very high, and the accuracy is very low, and the result show us that only 86 were correctly labeled out of 592

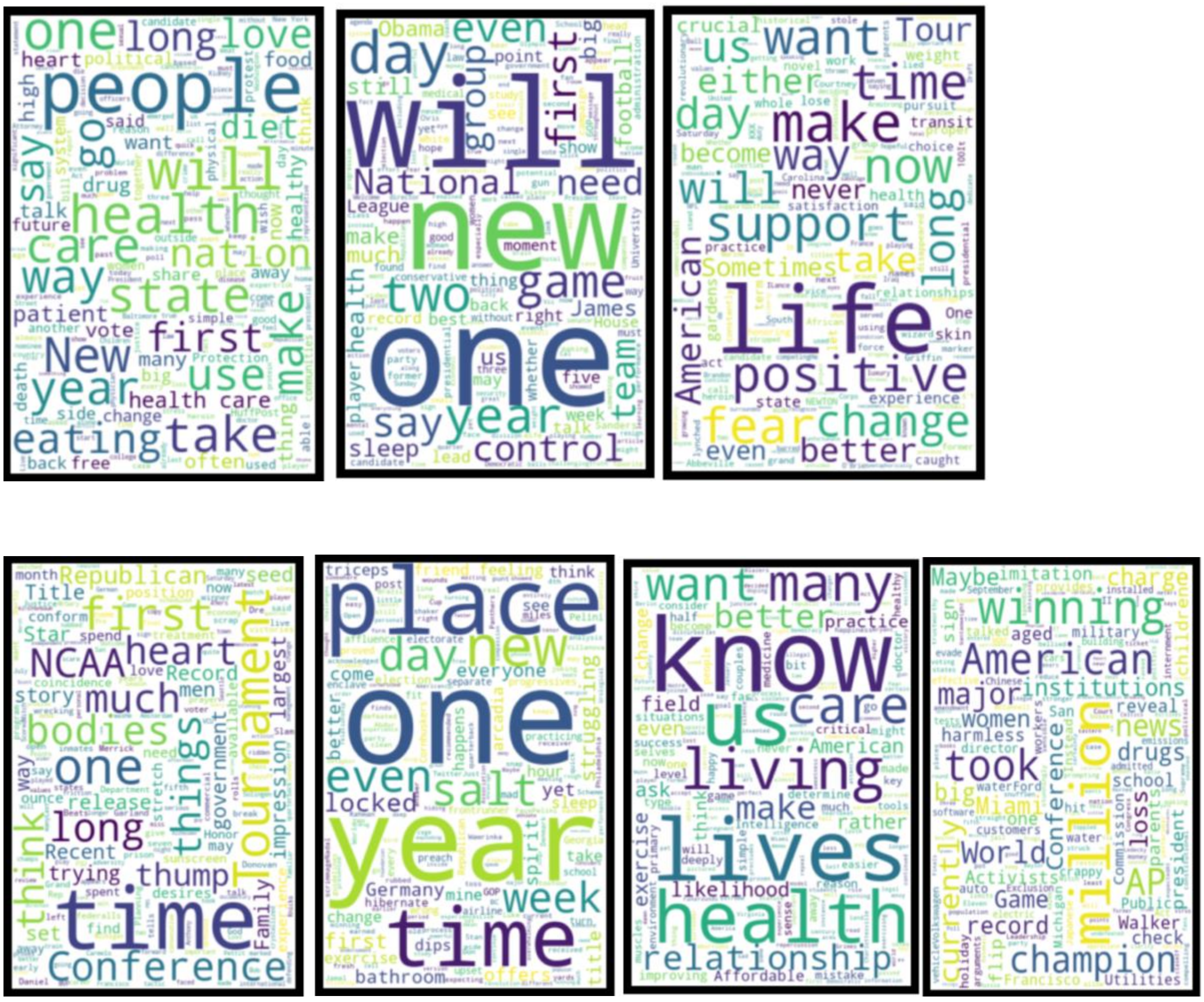

one long love people health care nation state will diet way make New year use first eating take
day even will first new one year National need game two control say sleep team football
us want Tour time either day make way now will support life positive fear change better American long
Republican first NCAA heart much bodies one long time Conference tournament things think
place one day new year salt time week even locked spirit
want many know us care living lives health relationship better
winning American major institutions took news World million champion AP record

Figure 16 Image Word cloud for 7 cluster model

Elbow method

For determining the current number of clusters can be very tricky sometime. To tackle the problem of finding the right cluster count for the dataset we do the elbow method, the k means model is run multiple times and when there is no change in the cluster or reduce wcss value an elbow is created we use that point as the reference.

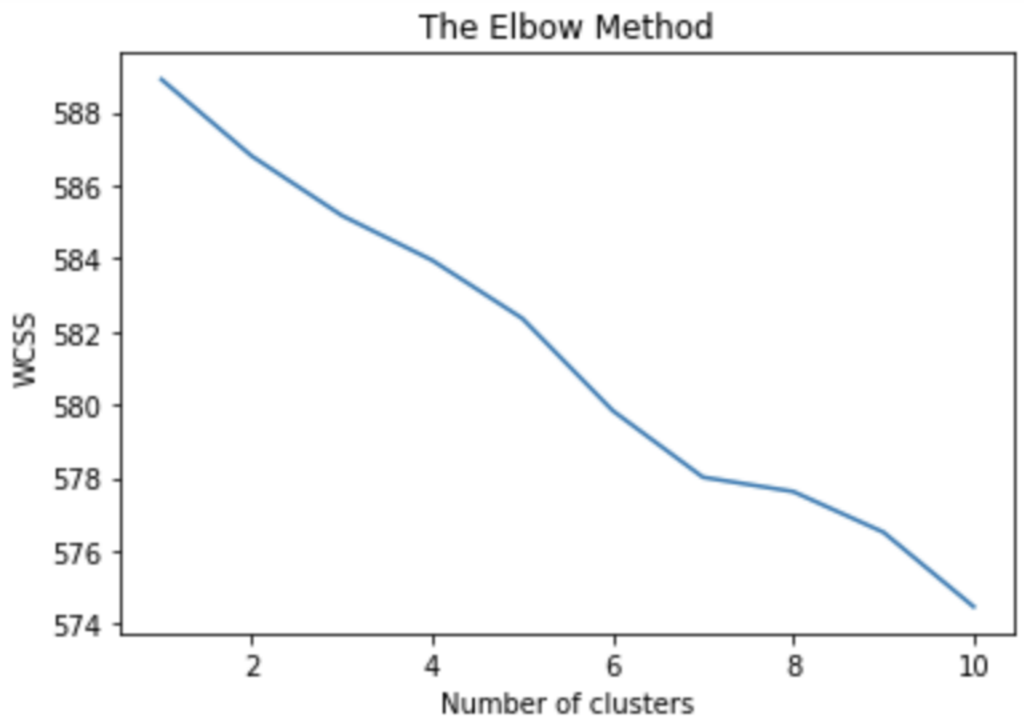
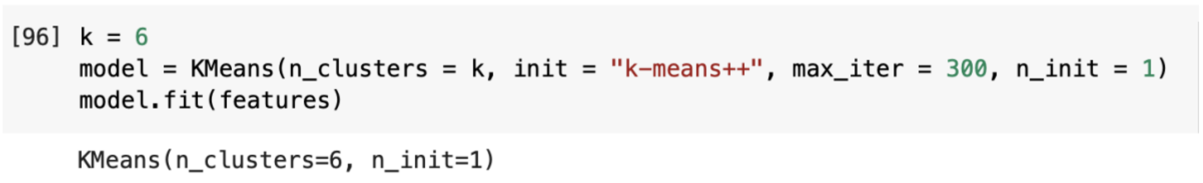


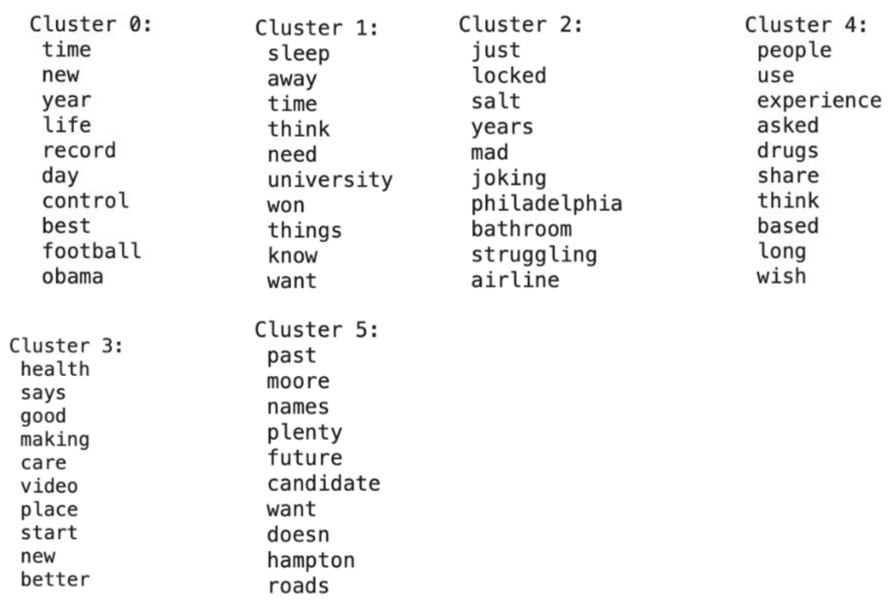
Figure 17 elbow plot

As we can see the elbow plot, we have an elbow in the 6th cluster. Now that we have found the right k value we can create a model with 6 clusters and compare the results.



```
[96] k = 6
     model = KMeans(n_clusters = k, init = "k-means++", max_iter = 300, n_init = 1)
     model.fit(features)

     KMeans(n_clusters=6, n_init=1)
```

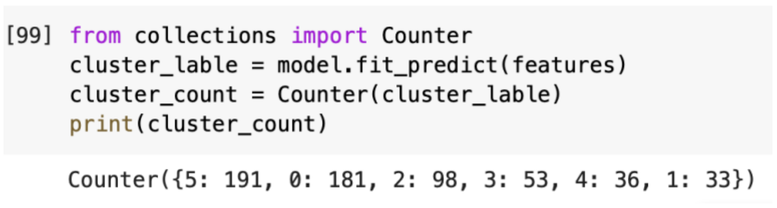
Figure 18 Model with 6 clusters



```
Cluster 0:
 time
 new
 year
 life
 record
 day
 control
 best
 football
 obama

Cluster 1:
 sleep
 away
 time
 think
 need
 university
 won
 things
 know
 want

Cluster 2:
 just
 locked
 salt
 years
 mad
 joking
 philadelphia
 bathroom
 struggling
 airline

Cluster 4:
 people
 use
 experience
 asked
 drugs
 share
 think
 based
 long
 wish

Cluster 3:
 health
 says
 good
 making
 care
 video
 place
 start
 new
 better

Cluster 5:
 past
 moore
 names
 plenty
 future
 candidate
 want
 doesn
 hampton
 roads
```

Figure 19 Image common words in the cluster

We can now find the distribution in each cluster



```
[99] from collections import Counter
     cluster_lable = model.fit_predict(features)
     cluster_count = Counter(cluster_lable)
     print(cluster_count)

     Counter({5: 191, 0: 181, 2: 98, 3: 53, 4: 36, 1: 33})
```

Figure 20 cluster count

As we can see there is some equal distribution among the clusters now to check the results of the model

K- Means						
K	6					
Cluster	0	1	2	3	4	5
Cluster count	181	33	98	53	36	191
Results						
Accuracy	0.65					
Adjusted Rand- Index	0.143					
V-measure	0.194					
Completeness	0.211					

Homogeneity	0.180
Inertia	487.512

Table 4 model summary

Word cloud for the model are shown below

Figure 21: 6 cluster word cloud

9. Discussion

In the task we have successfully built a search engine and host it on a static website although the process is straight forward, there are several details that need to be noted and errors needed to be tackled to get the required output. The search engine can be further improved my inducing ranking inro the search algorithm for which we can use TF-IDF to do the ranking, this can drastically increase the search outputs. An interactive UI has been implemented for the user to have a pleasing experience.

On the second part of the report, we have performed clustering for 3 different K values and compared the results it was found that model with 6 clusters was better than the other two even though the inertia as high and accuracy was 65%, we can say out of the 3 models the third model is the better model. The problem face while implementing clustering was most of the points were sitting on a certain cluster this problem roots to the beginning of the clustering process where we randomly place points and cluster the data near the points. Unfortunately, this problem was not handled in the report

10. Conclusion

In Conclusion we have built a search engine which is capable of crawling, storing the data and indexing the mentioned website and document clustering was implemented and clusters were created spreading politics, sports, and wellbeing.

11. Bibliography

Spiders — Scrapy 2.6.2 documentation. (2022). Retrieved 28 July 2022, from https://docs.scrapy.org/en/latest/topics/spiders.html

Understanding K-means Clustering in Machine Learning. (2022). Retrieved 28 July 2022, from https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

12 Appendix

12.1 Appendix A – Creating an APP on Flask

```
import pandas as pd
from crypt import methods
from flask import Flask, render_template , request
from spider import run_spider
from query_processing import inverted_Title, processText, search_index, file_path


app = Flask(__name__)

@app.route('/')
@app.route('/home')
def home():
    run_spider()
    return render_template("index.html")
@app.route("/spider", methods = ['POST','GET'])
def result():
    output = request.form.to_dict()
    name = output['name']
    title_index = inverted_Title(file_path)

    q = processText(name)
    search_results = search_index(q, title_index)
    print(search_results)
    exit()

    return render_template('result.html', search_results=search_results)

if __name__ == "__main__":
    app.run(debug=True,port=5001)
```

12.2 Appendix B – Creating a Crawler file

```
import scrapy
from scrapy.crawler import CrawlerRunner
from crochet import setup, wait_for
setup()

class PurePortalSpider(scrapy.Spider):
  name = "SEFAAuthors"

  start_urls = [
    "https://pureportal.coventry.ac.uk/en/organisations/school-of-economics-
finance-and-accounting/publications/"
  ]

  custom_settings = {
      "FEEDS": {
          "authors.csv": {
              "format": "csv",
              "overwrite": True
          }
      }
  }



  def parse(self, response):
    page = response
    li = page.css('li.list-result-item')
    title = li.css('h3.title span::text').get()
    for item in page.css('li.list-result-item'):
      title = item.css('h3.title span::text').get()
      authors = item.css('a.person span::text').getall()
      date = item.css('span.date::text').get()
      pub_link = item.css('h3.title a.link::attr(href)').extract()
      author_profile = item.css('a.person::attr(href)').getall()
      yield {
          "Title":title,
          "Authors":authors,
          'Date': date,
          'Pureportal': author_profile,
          'Pub Link': pub_link
      }


    next_page = response.css('a.nextLink::attr(href)').extract()

    if next_page is not None or len(next_page) > 0:
      next_page = response.urljoin(next_page[0])
      #print(next_page)

      yield scrapy.Request(next_page, callback=self.parse)



@wait_for(100)
def run_spider():
  my_spider = CrawlerRunner()
  s = my_spider.crawl(PurePortalSpider)
  return s

run_spider()
```

12.3 Appendix C – Creating a pre processing File

```
import pandas as pd
import numpy as np
import re
import string
import sys
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
stopword = stopwords.words('english')
import collections
from nltk.stem.porter import PorterStemmer


file_path = "/Users/roshandhanashekeran/Desktop/flask/authors.csv"

def inverted_Title(file_path):
  df2= pd.read_csv(file_path)
  df2=df2["Title"]

  inverted_title={}

  doc_id = 0
  for x in df2:
    #print('review', doc_id)
    token = word_tokenize(x)
    token = [w.translate(str.maketrans("","" , string.punctuation)) for w in token]
    token = [w for w in token if w]
    token_lower = [w.lower() for w in token]
    removing_stopwords = [w for w in token_lower if w not in stopword]
    stemmed_title = [stemmer.stem(w) for w in removing_stopwords]

    for a in stemmed_title:
      val = inverted_title.get(a)
      if val == None:
        l = [1,[doc_id]]
        inverted_title[a] = l
      else:
        l =  inverted_title[a]
        if doc_id not in l[1]:
          l[1].append(doc_id)
          l[0] += 1

      #print(doc_id)
    doc_id+=1

      #print(doc_id)
  ordered_inverted_index = collections.OrderedDict(sorted(inverted_title.items()))
  return ordered_inverted_index

#inverted = inverted_Title(file_path)


def processText(text):
  token = word_tokenize(text)
  token = [w.translate(str.maketrans("","" , string.punctuation)) for w in token]
  token = [w for w in token if w]
  token_lower = [w.lower() for w in token]
  stemmed_title = [stemmer.stem(w) for w in token_lower]
  return stemmed_title

#q = input("Enter the Query: ")
#q = q.lower()
#q = processText(q)


def search_index(stemmed_title, index):
    q = stemmed_title

    all_pl=[]
    for s in q:
        pl = index.get(s)
        if pl is not None:
            print(s)
            print(pl)
            all_pl.append(pl[1])

    results_set = set.intersection(*map(set,all_pl))
    results_set = list(results_set)
    df = pd.read_csv(file_path)
    results = df.iloc[results_set].T.to_dict('dict')
    return results


#the_set = set.intersection(*map(set,all_pl))
#print(the_set)
```

12.4 Appendix D – Creating the HTML file

```
<html>

    <head>
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
        <link rel = "stylesheet" type = "text/css" href="{{url_for('static',
filename = 'index.css')}}">
        <title>SEFA Search</title>
    </head>

        <body>
            <img src="{{url_for('static', filename= 'SEFA Search.png')}}" alt="SEFA
Search">

            <form class = "grid" action = "/spider" method = "POST">
                    <!-- <label for = "Name"> What The Hell are You Looking For
:</label> -->
                    <input type = "text" name = "name" id = "name"
placeholder="Search for something..." required>

                <center> <input type="submit" class = "file_submit"> </center>

            </form>
        <br/><br/><br/><br/><br/>



        </body>

</html>
```

12.5 Appendix E – Creating a CSS File

```
html,body{
    /* margin: 10px; */
    padding: 7% 10% 0 10%;
    list-style: none;
    box-sizing: border-box;
    font-family: Arial,sans-serif;
    letter-spacing: 3px;
}

center{
    padding-top: 15%;
    padding-top: 2%;
    align-content: flex-start;
}

a{
    cursor: pointer;
}

h2{
    color : #0066cb;
    padding-top: 2%;
}

/* element */

.let_space{
    letter-spacing: 2px;
}

img{
    padding-left: 25%;
    width: 70%;
}

.c_grid{
    display: grid;
    justify-items: center;
    align-items: center;
    grid-gap: 2vmax;
}

.grid{
    display: grid;
    grid-gap: 2vmax;
}



.box2{
    grid-template-columns: 1fr 1fr;
}

.box3{
    grid-template-columns: 1fr 1fr 1fr;
}

/* element */

.table_cont{
    overflow-x: scroll;
    width: 70vw;
}

table {
    border-collapse: collapse;
    letter-spacing: 3px;
    /* width: 100%; */
}

td, th {
    border: 1px solid #dddddd;
    text-align: center;
    padding: 1vmax;
    width: fit-content;
}

form{
    margin: 2vmax 10vmax;
    /* padding-top: 15%; */
   /* border: 1px solid #dddddd; */
}

label{
    font-size: 1.5vmax;
}

input{
    padding: 2%;
    font-size: 1.3vmax;
    border: 1px solid #dddddd;
    outline: none;
    border-radius: 30px;
    border-color: #d9d4d4;
}

select{
    padding:1vmax;
    font-size: 1.3vmax;
    border: 1px solid #dddddd;
    outline: none;
    letter-spacing: 3px;
}

option{
    font-size: 1.5vmax;
    letter-spacing: 3px;
}

.file_submit{
  margin-top: 1vmax;
  border: 1px solid #0066cb;
  outline: none;
  background-color: #e3eff9;
  color: 0066cb;
  padding: 1vmax;
  cursor: pointer;
  transition: 0.3s;
  letter-spacing: 3px;
  padding: 15px;
}

.file_submit:hover{
  background-color: white;
  color: #0066cb
}
```

12.6 Appendix F – creating results page HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <title>Result</title>
</head>
<body>

    <div class="container">
        {% for row in search_results%}
        <div class="result-container card">
            <h2><a target="_blank" href="{{ search_results[row]['Pub Link']}}">{{
search_results[row]['Title']}}</a></h2>
            <h5>{{ search_results[row]['Date']}}<h5>
            <h5><a target="_blank" href= "{{ search_results[row]['Pureportal']}}"
>{{ search_results[row]['Authors']}}
            </a></h5>
            <!--<h5> <a target="_blank" href= "{{
search_results[row]['Pureportal']}}" >Author profile</a> </h5> !-->
            <!--<h5> <a target="_blank">{{ search_results[row]['Pub Link']}}
</a></h5>-->
        </div>
        {% endfor %}
    </div>

</body>
</html>
```

12.7 Appendix G – creating a k- means clustering model

```
import pandas as pd
import numpy as np
import sklearn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud , STOPWORDS, ImageColorGenerator
from PIL import Image
import seaborn as sns
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from nltk.tokenize import RegexpTokenizer
from nltk.stem.snowball import SnowballStemmer
# %matplotlib inline
```

```
import pandas as pd
df=pd.read_csv("/content/NewsCategorizer.csv",error_bad_lines=False)
```

```
categories = ['POLITICS', 'SPORTS', 'WELLNESS']
#df['category'].isin(c1)
df =df[df['category'].isin(categories)]
df = df.groupby('category').sample(200)
```

```
print(df)

#print(res)
df.head()

df = df.drop_duplicates("short_description")
df['category'].unique()

df.info()

document = df["short_description"].values.astype("U")

vectorizer = TfidfVectorizer(stop_words="english")
features = vectorizer.fit_transform(document)

print(features)

k = 6
model = KMeans(n_clusters = k, init = "k-means++", max_iter = 300, n_init = 1)
model.fit(features)

df["cluster"] = model.labels_

#df.tail()



clusters = df.groupby('cluster')

for cluster in clusters.groups:
    f = open('cluster'+str(cluster)+ '.csv', 'w') # create csv file
    data = clusters.get_group(cluster)[['category','short_description']] # get title and
overview columns
    f.write(data.to_csv(index_label='id')) # set index to id
    f.close()

print("Cluster centroids: \n")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()

for i in range(k):
    print("Cluster %d:" % i)
    for j in order_centroids[i, :10]: #print out 10 feature terms of each cluster
        print (' %s' % terms[j])
    print('------------')

from collections import Counter
cluster_lable = model.fit_predict(features)
cluster_count = Counter(cluster_lable)
print(cluster_count)

model.inertia_

labels = kmeans.labels_

# check how many of the samples were correctly labeled
correct_labels = sum(y_pred == label)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels,
label.size))
```

```
# Commented out IPython magic to ensure Python compatibility.
from sklearn import metrics
y_pred = KMeans(n_clusters=6).fit_predict(features)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(label,y_pred))
print("Completeness: %0.3f" % metrics.completeness_score(label,y_pred))
print("V-measure: %0.3f" % metrics.v_measure_score(label, y_pred))
print("Adjusted Rand-Index: %.3f"
#      % metrics.adjusted_rand_score(label, y_pred))
#print("Silhouette Coefficient: %0.3f"
     #% metrics.silhouette_score(label, y_pred, sample_size=1000))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(label.size)))

from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
  kmeans = KMeans(n_clusters=i,init='k-
means++',max_iter=100,n_init=10,random_state=0)
  kmeans.fit(features)
  wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.savefig('elbow.png')
plt.show()

# cluster0
# word cloud
```

```
cluster0 = pd.read_csv("/content/cluster0.csv", error_bad_lines=False,usecols
=["short_description"])
cluster1 = pd.read_csv("/content/cluster1.csv", error_bad_lines=False,usecols
=["short_description"])
cluster2 = pd.read_csv("/content/cluster2.csv", error_bad_lines=False,usecols
=["short_description"])
cluster3 = pd.read_csv("/content/cluster3.csv", error_bad_lines=False,usecols
=["short_description"])
cluster4 = pd.read_csv("/content/cluster4.csv", error_bad_lines=False,usecols
=["short_description"])
cluster5 = pd.read_csv("/content/cluster5.csv", error_bad_lines=False,usecols
=["short_description"])
#cluster6 = pd.read_csv("/content/cluster6.csv", error_bad_lines=False,usecols
=["short_description"])

#text = open("/content/cluster0.csv", mode="r")
stopwords = STOPWORDS

wc = WordCloud(
   background_color = "white",
   stopwords = stopwords,
   height= 600,
   width = 400
)

wc.generate("".join(cluster0["short_description"]))
plt.figure(figsize=(20,5),facecolor = 'k')
plt.imshow(wc , interpolation = "bilinear")
plt.axis("off")
plt.tight_layout(pad= 0)
plt.show()

wc.generate("".join(cluster1["short_description"]))
plt.figure(figsize=(20,5),facecolor = 'k')
plt.imshow(wc , interpolation = "bilinear")
plt.axis("off")
plt.tight_layout(pad= 0)
plt.show()

wc.generate("".join(cluster2["short_description"]))
plt.figure(figsize=(20,5),facecolor = 'k')
plt.imshow(wc , interpolation = "bilinear")
plt.axis("off")
plt.tight_layout(pad= 0)
plt.show()

wc.generate("".join(cluster3["short_description"]))
plt.figure(figsize=(20,5),facecolor = 'k')
plt.imshow(wc , interpolation = "bilinear")
plt.axis("off")
plt.tight_layout(pad= 0)
plt.show()

wc.generate("".join(cluster4["short_description"]))
plt.figure(figsize=(20,5),facecolor = 'k')
plt.imshow(wc , interpolation = "bilinear")
plt.axis("off")
plt.tight_layout(pad= 0)
plt.show()
```

```
wc.generate("".join(cluster5["short_description"]))
plt.figure(figsize=(20,5),facecolor = 'k')
plt.imshow(wc , interpolation = "bilinear")
plt.axis("off")
plt.tight_layout(pad= 0)
plt.show()

wc.generate("".join(cluster6["short_description"]))
plt.figure(figsize=(20,5),facecolor = 'k')
plt.imshow(wc , interpolation = "bilinear")
plt.axis("off")
plt.tight_layout(pad= 0)
plt.show()

import matplotlib.pyplot as plt
u_labels = np.unique(cluster_lable)




filtered_label0 = df[cluster_lable == 0]
filtered_label1 = df[cluster_lable == 1]
filtered_label2 = df[cluster_lable == 2]
filtered_label3 = df[cluster_lable == 3]
filtered_label4 = df[cluster_lable == 4]
filtered_label5 = df[cluster_lable == 5]
#filtered_label6 = df[cluster_lable == 6]


plt.scatter(filtered_label0.iloc[:,0] , filtered_label0.iloc[:,1] , color = 'gold')
plt.scatter(filtered_label1.iloc[:,0] , filtered_label1.iloc[:,1] , color = 'red')
plt.scatter(filtered_label2.iloc[:,0] , filtered_label2.iloc[:,1] , color = 'pink')
plt.scatter(filtered_label3.iloc[:,0] , filtered_label3.iloc[:,1] , color = 'purple')
plt.scatter(filtered_label4.iloc[:,0], filtered_label4.iloc[:,1], color = 'green')
plt.scatter(filtered_label5.iloc[:,0] , filtered_label5.iloc[:,1] , color = 'grey')
#plt.scatter(filtered_label6.iloc[:,0].values , filtered_label6.iloc[:,1].values , color =
'orange')
#plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = "red")
plt.yticks(filtered_label5.iloc[:,1], labels=" ")

plt.show()

label = df["cluster"]
print(label)





while True:
  User_input = []
  User_input =input("Search :")
  y = vectorizer.transform([User_input])
  prediction = model.predict(y)
  print(prediction)

from collections import Counter
cluster_lable = model.fit_predict(features)
cluster_count = Counter(cluster_lable)
print(cluster_count)
```

```
cluster0 = cluster0["short_description"]

df.to_csv(r'clustor0.txt', header=None, index=None, sep='\t', mode='a')

cluster0 = (cluster0["short_description"].to_string())
#print(df.to_string())

df = pd.read_csv("/content/uci-news-aggregator.csv",error_bad_lines=False,usecols
=["TITLE"])
#df2 = df["TITLE"]
#df2.head()
df

df.info()

df[df['headline'].duplicated(keep=False)].sort_values('headline').head(8)



import missingno as msno
msno.bar(df, color=(0.1, 0.4, 0.5)) # There are null value in "keywords" column.

punc = ['.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}',"%"]
stop_words = text.ENGLISH_STOP_WORDS.union(punc)
desc = df['short_description'].values
vectorizer = TfidfVectorizer(stop_words = stop_words)
X = vectorizer.fit_transform(desc)
```

```
word_features = vectorizer.get_feature_names()
print(len(word_features))
print(word_features[5000:5100])

stemmer = SnowballStemmer('english')
tokenizer = RegexpTokenizer(r'[a-zA-Z\']+')

def tokenize(text):
   return [stemmer.stem(word) for word in tokenizer.tokenize(text.lower())]

vectorizer2 = TfidfVectorizer(stop_words = stop_words, tokenizer = tokenize)
X2 = vectorizer2.fit_transform(desc)
word_features2 = vectorizer2.get_feature_names()
print(len(word_features2))
print(word_features2[:50])

vectorizer3 = TfidfVectorizer(stop_words = stop_words, tokenizer = tokenize,
max_features = 1000)
X3 = vectorizer3.fit_transform(desc)
words = vectorizer3.get_feature_names()

from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
   kmeans = KMeans(n_clusters=i,init='k-
means++',max_iter=300,n_init=10,random_state=0)
   kmeans.fit(X3)
   wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.savefig('elbow.png')
plt.show()

print(words[250:300])

kmeans = KMeans(n_clusters = 5,init= 'k-means++',n_init = 50, max_iter=1000)
# n_init(number of iterations for clsutering) n_jobs(number of cpu cores to use)
count = kmeans.fit(X3)
# We look at  the clusters generated by k-means.
common_words = kmeans.cluster_centers_.argsort()[:,-1:-26:-1]
for num, centroid in enumerate(common_words):
    print(str(num) + ' : ' + ', '.join(words[word] for word in centroid))

from collections import Counter
cluster_lable = kmeans.fit_predict(X3)
cluster_count = Counter(cluster_lable)
print(cluster_count)

count.inertia_

label = count.labels_

#from sklearn.feature_extraction.text import TfidfVectorizer
#vect = TfidfVectorizer()
#X = vect.fit_transform(desc)
#print(X.todense())
```

```
while True:
  User_input = []
  User_input =input("Search :")
  y = vectorizer3.transform([User_input])
  prediction = kmeans.predict(y)
  print(prediction)

#Getting unique labels
import matplotlib.pyplot as plt
u_labels = np.unique(cluster_lable)
#X3 = np.array(X3)
#plotting the results:
#for i in u_labels:

#    plt.scatter(df[cluster_lable == i , 0] , df[cluster_lable == i , 1] , cluster_lable = i)
#filter rows of original data

filtered_label0 = df[cluster_lable == 0]
filtered_label1 = df[cluster_lable == 1]
filtered_label2 = df[cluster_lable == 2]
filtered_label3 = df[cluster_lable == 3]
filtered_label4 = df[cluster_lable == 4]
filtered_label5 = df[cluster_lable == 5]
#filtered_label6 = df[cluster_lable == 6]

#Plotting the results
plt.scatter(filtered_label0.iloc[:,0] , filtered_label0.iloc[:,1] , color = 'gold')
plt.scatter(filtered_label1.iloc[:,0] , filtered_label1.iloc[:,1] , color = 'red')
plt.scatter(filtered_label2.iloc[:,0] , filtered_label2.iloc[:,1] , color = 'pink')
plt.scatter(filtered_label3.iloc[:,0] , filtered_label3.iloc[:,1] , color = 'purple')
plt.scatter(filtered_label4.iloc[:,0], filtered_label4.iloc[:,1], color = 'green')
plt.scatter(filtered_label5.iloc[:,0] , filtered_label5.iloc[:,1] , color = 'grey')
#plt.scatter(filtered_label6.iloc[:,0].values , filtered_label6.iloc[:,1].values , color =
'orange')
#plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = "red")
plt.yticks(filtered_label5.iloc[:,1], labels=" ")
plt.show()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import numpy as np
show_label = np.unique(label)

for i in show_label:
  plt.scatter(df[label == i,0], df[label == i, 1], label = i)
plt.legend()
plt.show()

import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords

sw = stopwords.words('english')
print(sw)

nltk.download("punkt")
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
```

```
ps = PorterStemmer()
filtered_docs = []
for doc in df2:
    tokens = word_tokenize(doc)
    tmp = ""
    for w in tokens:
        if w not in sw:
            tmp += ps.stem(w) + " "
    filtered_docs.append(tmp)

print(filtered_docs)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(filtered_docs)
print(X.todense())
```