INFORMATION RETRIEVAL
Assignment 2

Group Members :
1. Abhishek Nigam (MT22003)
2. Dhananjay Bagul (MT22028)
3. Wilson Radadia (MT22091)

❖ **Question 1:**

➔ **Necessary Libraries:**

- **math**: provides standard mathematical constants and functions..
- **pandas:** It offers capabilities for cleaning, combining, and reshaping data as well as data structures for effectively storing and handling huge datasets.
- **np**: used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.
- **NLTK**: The Natural Language Toolkit (nltk) is used for performing tasks such as: Tokenization, Part-of-speech tagging, Sentiment Analysis,Stemming and Lemmatization etc.
- **re :** Provides operations of detecting patterns with the help of regular expressions.
- **json :** Using json we can easily convert between Python objects and JSON data, making it easier to exchange data between applications written in different programming languages. The json module provides a method for encoding Python objects into JSON format: json.dumps() used to return a string representation of a JSON-encoded object.
- **string :** For working with text data present in dataset files.
- **numpy :** It offers functions for mathematical operations like linear algebra, Fourier transforms, and random number generation, as well as efficient numerical operations on multidimensional arrays and matrices.
- **joblib :** provides a channel for lightning.
- **BeautifulSoup :** It is used for web scraping and parsing HTML and XML documents, providing a convenient and flexible way to extract and manipulate data from web pages.
- **operator :** It is used for providing mathematical operations.

➔ **Data-Preprocessing:**

- Import the Dataset from the google drive.
- In Preprocessing,convert the files into lowercase.
- Tokenize the data in the next steps into small small words.
- Further remove the stopwords from the tokenized files.
- Remove the punctuation after the stopwords.
- Remove the blank spaces from the files after the punctuation.

➔ After preprocessing, printing first 5 files for result:

```
[ ] #printing 5 files before removing blank space tokens
    first5 = ['cranfield0001', 'cranfield0002', 'cranfield0003', 'cranfield0004', 'cranfield0005']

    for filename in first5:
        with open(filename, 'r') as file:
            contents = file.read()
            print(contents)
```

```
experimental investigation aerodynamics wing slipstream  experimental study wing propeller slipstream made order determine spanwise distribution lift increase due slipstream different
simple shear flow past flat plate incompressible fluid small viscosity  study highspeed viscous flow past twodimensional body usually necessary consider curved shock wave emitting nose
boundary layer simple shear flow past flat plate  boundarylayer equations presented steady incompressible flow pressure gradient
approximate solutions incompressible laminar boundary layer equations plate shear flow  twodimensional steady boundarylayer problem flat plate shear flow incompressible fluid considere
onedimensional transient heat conduction doublelayer slab subjected linear heat input small time internal  analytic solutions presented transient heat conduction composite slabs expose
```

➢ Question 1.1: TF-IDF Matrix

- Created a dictionary in word_freq the count of that word in a specific file is stored and there is another dictionary in which the count of that word in all the files is stored which will be used for TF.

```
simplicitys 1
sake 1
denoted 1
619 1
v184 1
12671270 1
refined 1
latitude 2
season 1
daytonight 1
245 1
huggettdj 1
19713 1
581 1
hits 1
comments 1
flapped 1
880 1
mccarthy 1
jf 1
halfman 1
scaled 1
617 1
2834 1
airdensity 1
profileieits 1
heightto 1
subperigee 1
220 1
829 1
donnellh 1
r479 1
```

- After that, unique words are stored in a dictionary unique_words.
- After that a posting list is created with a dictionary named posting_list in which the word and in which file it is present is stored in it.

```
sake {'cranfield0414', 'cranfield0202'}
1955226 {'cranfield0414'}
5th {'cranfield1036', 'cranfield0414', 'cranfield1330', 'cranfield0268', 'cranfield1378'}
latitude {'cranfield0619', 'cranfield0621'}
refined {'cranfield1138', 'cranfield0497', 'cranfield1259', 'cranfield0983', 'cranfield122
v184 {'cranfield0619'}
619 {'cranfield0619'}
daytonight {'cranfield0619'}
12671270 {'cranfield0619'}
season {'cranfield0619'}
comments {'cranfield1086', 'cranfield0245', 'cranfield0756', 'cranfield0046', 'cranfield0
flapped {'cranfield0245'}
245 {'cranfield0371', 'cranfield0245'}
19713 {'cranfield0245'}
hits {'cranfield0245'}
581 {'cranfield0245', 'cranfield0581'}
huggettdj {'cranfield0245'}
mccarthy {'cranfield0880'}
halfman {'cranfield0880'}
880 {'cranfield0880'}
jf {'cranfield0880'}
scaled {'cranfield0880'}
subperigee {'cranfield0617'}
617 {'cranfield0617'}
airdensity {'cranfield0617'}
2834 {'cranfield0617'}
heightto {'cranfield0617'}
profileieits {'cranfield0617'}
220 {'cranfield1001', 'cranfield0617', 'cranfield0220', 'cranfield0621'}
r479 {'cranfield0829'}
829 {'cranfield0829'}
ascribed {'cranfield0829'}
checks {'cranfield0829'}
experimentalfailure {'cranfield0829'}
items {'cranfield0829'}
donnellh {'cranfield0829'}
```

● Then idf value is calculated according to the given formula: log(total number of documents/document frequency(term)+1) for all the terms.

```
sake 8.866248611111173
1955226 9.451211111832329
5th 7.866248611111173
latitude 8.866248611111173
refined 7.643856189774724
v184 9.451211111832329
619 9.451211111832329
daytonight 9.451211111832329
12671270 9.451211111832329
season 9.451211111832329
comments 7.643856189774724
flapped 9.451211111832329
245 8.866248611111173
19713 9.451211111832329
hits 9.451211111832329
581 8.866248611111173
huggettdj 9.451211111832329
mccarthy 9.451211111832329
halfman 9.451211111832329
880 9.451211111832329
jf 9.451211111832329
scaled 9.451211111832329
subperigee 9.451211111832329
617 9.451211111832329
airdensity 9.451211111832329
2834 9.451211111832329
heightto 9.451211111832329
profileieits 9.451211111832329
220 8.129283016944967
r479 9.451211111832329
829 9.451211111832329
ascribed 9.451211111832329
checks 9.451211111832329
experimentalfailure 9.451211111832329
items 9.451211111832329
donnellh 9.451211111832329
```

(i). Creating a matrix of size no. of documents x vocab size:

- **For Binary Weighting Scheme (0,1):**

  - TF-IDF values:

```
latent 9.451211111832329
heat 2.631032149417141
evaporation 8.866248611111173
water 6.54432051622381
results 1.227209437634224
obtained 1.7787857698608331
mach 1.8587540745642483
number 1.541318028062287
possibility 5.36374827058199
extending 6.203283598388744
vapour 9.451211111832329
screen 8.866248611111173
technique 4.497014801445454
transonic 4.473931188332412
subsonic 3.643856189774725
speeds 3.147430363655226
also 2.1797480839279544
considered 2.6963236096688603
results 1.227209437634224
obtained 1.7787857698608331
mach 1.8587540745642483
number 1.541318028062287
085 8.451211111832329
included 3.927649155775316
text 0.004127885622676564
doc 0.004127885622676564
```

  - Filling the tf-idf values for each term in the vocabulary in the matrix:

```
               canard  exists  recompression  hicks2  261  704  primaryshock  \
cranfield0045       0       0              0       0    0    0             0
cranfield0255       0       0              0       0    0    0             0
cranfield1387       0       0              0       0    0    0             0
cranfield0691       0       0              0       0    0    0             0
cranfield1123       0       0              0       0    0    0             0
...               ...     ...            ...     ...  ...  ...           ...
cranfield0252       0       0              0       0    0    0             0
cranfield1124       0       0              0       0    0    0             0
cranfield0239       0       0              0       0    0    0             0
cranfield1311       0       0              0       0    0    0             0
cranfield0466       0       0              0       0    0    0             0

               1379  observing  multilayered  ...  holding  windtunnel  \
cranfield0045     0          0             0  ...        0           0
cranfield0255     0          0             0  ...        0           0
cranfield1387     0          0             0  ...        0           0
cranfield0691     0          0             0  ...        0           0
cranfield1123     0          0             0  ...        0           0
...             ...        ...           ...  ...      ...         ...
cranfield0252     0          0             0  ...        0           0
cranfield1124     0          0             0  ...        0           0
cranfield0239     0          0             0  ...        0           0
cranfield1311     0          0             0  ...        0           0
cranfield0466     0          0             0  ...        0           0
```

- **For Raw Count Weighting Scheme (f(t,d)):**
  - TF-IDF values:

```
pressure 1.4825443186371206
investigated 3.4970148014454536
compared 2.851298269645201
theoretical 2.6830267870554025
estimates 5.807354922057604
nominal 6.36374827058199
20 4.36374827058199
shown 2.2913397750539395
adverse 5.866248611111173
high 2.927649155775316
numbers 2.5443205162238103
may 2.526398608226548
alleviated 8.866248611111173
liquids 7.866248611111173
lower 4.385121921374557
latent 9.451211111832329
heat 2.631032149417141
evaporation 8.866248611111173
water 6.54432051622381
obtained 3.5575715397216663
possibility 5.36374827058199
extending 6.203283598388744
transonic 4.473931188332412
subsonic 3.643856189774725
speeds 3.147430363655226
also 2.1797480839279544
```

  - Filling the tf-idf values for each term in the vocabulary in the matrix:

	canard	exists	recompression	hicks2	261	704	primaryshock	\
cranfield0045	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cranfield0255	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cranfield1387	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cranfield0691	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cranfield1123	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
cranfield0252	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cranfield1124	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cranfield0239	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cranfield1311	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
cranfield0466	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	1379	observing	multilayered	...	holding	windtunnel	\
cranfield0045	0.0	0.0	0.0	...	0.0	0.0	
cranfield0255	0.0	0.0	0.0	...	0.0	0.0	
cranfield1387	0.0	0.0	0.0	...	0.0	0.0	
cranfield0691	0.0	0.0	0.0	...	0.0	0.0	
cranfield1123	0.0	0.0	0.0	...	0.0	0.0	
...	...	...	...	...	...	...	
cranfield0252	0.0	0.0	0.0	...	0.0	0.0	
cranfield1124	0.0	0.0	0.0	...	0.0	0.0	
cranfield0239	0.0	0.0	0.0	...	0.0	0.0	
cranfield1311	0.0	0.0	0.0	...	0.0	0.0	
cranfield0466	0.0	0.0	0.0	...	0.0	0.0	

- **For Term frequency Weighting Scheme (f(t,d)/Pf(t‘, d)):**
  - TF_IDF values:

```
theoretical 0.4471711311759004
estimates 0.9678924870096006
nominal 1.060624711763665
20 0.7272913784303316
shown 0.3818899625089899
adverse 0.9777081018518621
high 0.48794152596255264
numbers 0.424053419370635
may 0.4210664347044247
alleviated 1.477708101851862
liquids 1.3110414351851953
lower 0.7308536535624262
latent 1.5752018519720548
heat 0.4385053582361902
evaporation 1.477708101851862
water 1.0907200860373016
obtained 0.592928589953611
possibility 0.8939580450969983
extending 1.0338805997314573
transonic 0.745655198055402
subsonic 0.6073093649624541
speeds 0.524571727275871
also 0.36329134732132573
```

○Filling the tf-idf values for each term in the vocabulary in the matrix:

```
               canard  exists  recompression  hicks2  261  704  primaryshock  \
cranfield0045     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0255     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1387     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0691     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1123     0.0     0.0            0.0     0.0  0.0  0.0           0.0
...               ...     ...            ...     ...  ...  ...           ...
cranfield0252     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1124     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0239     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1311     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0466     0.0     0.0            0.0     0.0  0.0  0.0           0.0

               1379  observing  multilayered  ...  holding  windtunnel  \
cranfield0045   0.0        0.0           0.0  ...      0.0         0.0
cranfield0255   0.0        0.0           0.0  ...      0.0         0.0
cranfield1387   0.0        0.0           0.0  ...      0.0         0.0
cranfield0691   0.0        0.0           0.0  ...      0.0         0.0
cranfield1123   0.0        0.0           0.0  ...      0.0         0.0
...             ...        ...           ...  ...      ...         ...
cranfield0252   0.0        0.0           0.0  ...      0.0         0.0
cranfield1124   0.0        0.0           0.0  ...      0.0         0.0
cranfield0239   0.0        0.0           0.0  ...      0.0         0.0
cranfield1311   0.0        0.0           0.0  ...      0.0         0.0
cranfield0466   0.0        0.0           0.0  ...      0.0         0.0
```

- **For Log Normalization Weighting Scheme (log(1+f(t,d))):**

○TF_IDF values:

```
pressure 1.0276214145184852
investigated 2.423945949998313
compared 1.9763693565400218
theoretical 1.859732452814261
estimates 4.025351690735149
nominal 4.4110141715471345
20 3.0247198104272437
shown 1.5882357047834974
adverse 4.0661736852554045
high 2.0292917579943643
numbers 1.7635885922613586
may 1.7511660722628017
alleviated 6.145615226935241
liquids 5.452468046375295
lower 3.039534896212384
latent 6.551080335043404
heat 1.823692516331064
evaporation 6.145615226935241
water 4.53617731450114
obtained 1.9541959056770755
possibility 3.7178669909871886
extending 4.2997885364369095
transonic 3.101092789211817
subsonic 2.5257286443082556
speeds 2.1816324825763833
also 1.5108862387056046
```

○ Filling the tf-idf values for each term in the vocabulary in the matrix:

```
               canard  exists  recompression  hicks2  261  704  primaryshock  \
cranfield0045     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0255     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1387     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0691     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1123     0.0     0.0            0.0     0.0  0.0  0.0           0.0
...               ...     ...            ...     ...  ...  ...           ...
cranfield0252     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1124     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0239     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1311     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0466     0.0     0.0            0.0     0.0  0.0  0.0           0.0

               1379  observing  multilayered  ...  holding  windtunnel  \
cranfield0045   0.0        0.0           0.0  ...      0.0         0.0
cranfield0255   0.0        0.0           0.0  ...      0.0         0.0
cranfield1387   0.0        0.0           0.0  ...      0.0         0.0
cranfield0691   0.0        0.0           0.0  ...      0.0         0.0
cranfield1123   0.0        0.0           0.0  ...      0.0         0.0
...             ...        ...           ...  ...      ...         ...
cranfield0252   0.0        0.0           0.0  ...      0.0         0.0
cranfield1124   0.0        0.0           0.0  ...      0.0         0.0
cranfield0239   0.0        0.0           0.0  ...      0.0         0.0
cranfield1311   0.0        0.0           0.0  ...      0.0         0.0
cranfield0466   0.0        0.0           0.0  ...      0.0         0.0
```

- **For Double Normalization Weighting Scheme (0.5+0.5*(f(t,d)/ max(f(t‘,d))):**

  - TF_IDF values:

```
pressure 0.8648175192049871
investigated 2.0399253008431812
compared 1.6632573239597008
theoretical 1.5650989591156517
estimates 3.387623704533602
nominal 3.7121864911728277
20 2.5455198245061608
shown 1.3366148687814647
adverse 3.421978356481518
high 1.7077953408689344
numbers 1.4841869677972228
may 1.4737325214654866
alleviated 5.171978356481518
liquids 4.588645023148184
lower 2.5579877874684915
latent 5.513206481902192
heat 1.5347687538266657
evaporation 5.171978356481518
water 3.817520301130556
obtained 1.185857179907222
possibility 3.1288531578394942
extending 3.618582099060101
transonic 2.6097931931939073
subsonic 2.1255827773685896
speeds 1.8360010454655487
also 1.27151971562464
```

  - Filling the tf-idf values for each term in the vocabulary in the matrix:

```
                canard  exists  recompression  hicks2  261  704  primaryshock  \
cranfield0045     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0255     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1387     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0691     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1123     0.0     0.0            0.0     0.0  0.0  0.0           0.0
...               ...     ...            ...     ...  ...  ...           ...
cranfield0252     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1124     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0239     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield1311     0.0     0.0            0.0     0.0  0.0  0.0           0.0
cranfield0466     0.0     0.0            0.0     0.0  0.0  0.0           0.0

                1379  observing  multilayered  ...  holding  windtunnel  \
cranfield0045    0.0        0.0           0.0  ...      0.0         0.0
cranfield0255    0.0        0.0           0.0  ...      0.0         0.0
cranfield1387    0.0        0.0           0.0  ...      0.0         0.0
cranfield0691    0.0        0.0           0.0  ...      0.0         0.0
cranfield1123    0.0        0.0           0.0  ...      0.0         0.0
...              ...        ...           ...  ...      ...         ...
cranfield0252    0.0        0.0           0.0  ...      0.0         0.0
cranfield1124    0.0        0.0           0.0  ...      0.0         0.0
cranfield0239    0.0        0.0           0.0  ...      0.0         0.0
cranfield1311    0.0        0.0           0.0  ...      0.0         0.0
cranfield0466    0.0        0.0           0.0  ...      0.0         0.0
```

(ii). Constructing the query vector of size vocab:

- **Binary Weighting Scheme:**
  - TF-IDF Score:

```
cranfield0045     0.000000
cranfield0255     0.000000
cranfield1387     0.000000
cranfield0691     0.000000
cranfield1123    13.564686
                    ...
cranfield0252    13.564686
cranfield1124     0.000000
cranfield0239    13.564686
cranfield1311     0.000000
cranfield0466     0.000000
```

  - Top 5 documents:

```
cranfield0798
cranfield1008
cranfield1134
cranfield1268
cranfield0668
```

- **Raw Count Weighting Scheme:**

- TF-IDF Score:

```
cranfield0045     0.000000
cranfield0255     0.000000
cranfield1387     0.000000
cranfield0691     0.000000
cranfield1123    13.564686
                    ...
cranfield0252    27.129373
cranfield1124     0.000000
cranfield0239    27.129373
cranfield1311     0.000000
cranfield0466     0.000000
```

- Top 5 documents:

```
cranfield0536
cranfield0484
cranfield0274
cranfield0595
cranfield0252
```

- **Term Frequency Weighting Scheme:**
  - TF-IDF score:

```
cranfield0045    0.00000
cranfield0255    0.00000
cranfield1387    0.00000
cranfield0691    0.00000
cranfield1123    0.85096
                   ...
cranfield0252    0.85096
cranfield1124    0.00000
cranfield0239    0.63822
cranfield1311    0.00000
cranfield0466    0.00000
```

  - Top 5 documents:

```
cranfield0536
cranfield0203
cranfield0291
cranfield1395
cranfield0896
```

- **Logarithmic normalization Weighting Scheme:**
  - TF-IDF score:

```
cranfield0045     0.000000
cranfield0255     0.000000
cranfield1387     0.000000
cranfield0691     0.000000
cranfield1123     6.517194
                    ...
cranfield0252    10.329509
cranfield1124     0.000000
cranfield0239    10.329509
cranfield1311     0.000000
cranfield0466     0.000000
```

- ○Top 5 document:

```
cranfield0536
cranfield0484
cranfield0274
cranfield0595
cranfield0252
```

- **Double normalization Weighting Scheme:**

  - ○TF-IDF score:

```
cranfield0045     0.000000
cranfield0255     0.000000
cranfield1387     0.000000
cranfield0691     0.000000
cranfield1123     9.402324
                    ...
cranfield0252    14.902331
cranfield1124     0.000000
cranfield0239    14.902331
cranfield1311     0.000000
cranfield0466     0.000000
```

  - ○Top 5 document:

```
cranfield0536
cranfield0484
cranfield0274
cranfield0595
cranfield0252
```

(iii)Pros and Cons of weighting schemes:

- Binary Weighting Scheme:
  - ○Pros:
    - ➔ This weighting scheme is simple and easy to implement.

- ➔ It can be effective for short documents where the presence or absence of a term is more important than its frequency.
  - ○Cons:
    - ➔ This weighting scheme does not take into consideration the frequency of terms due to which it can lead to poor results.
- ● Raw Count Weighting Scheme:
  - ○Pros:
    - ➔ This weighting scheme is also simple and easy to implement.
    - ➔ This weighting scheme takes into consideration the frequency of terms, which can be helpful for many types of documents and queries.
  - ○Cons:
    - ➔ This weighting scheme can be bias by the length of documents, where longer documents tend to have higher term frequencies.
- ● Term Frequency Weighting Scheme:
  - ○Pros:
    - ➔ This weighting scheme takes into consideration the frequency of terms, but also provides a way to normalize the term frequency based on the length of the document. This can be helpful to address different problems.
  - ○Cons:
    - ➔ This weighting scheme can be more complex to implement and it may not work well for all types of documents and queries.
- ● Log normalization Weighting Scheme:
  - ○Pros:
    - ➔ This weighting scheme can reduce the impact of very frequent terms while still taking into consideration the frequency of other terms.
    - ➔ It is a common weighting scheme that is often used in information retrieval.
  - ○Cons:
    - ➔ This weighting scheme can be more complex to implement than binary weighting.
- ● Double normalization Weighting Scheme:
  - ○Pros:
    - ➔ This weighting scheme takes into consideration the frequency of terms, which can be helpful for many types of documents and queries.
  - ○Cons:
    - ➔ This weighting scheme can be more complex to implement than other weighting schemes, and it may not work well for short documents and queries.

➢ **Question 1.2 : Jaccard Coefficient :**

Basically the jaccard coefficient is used to measure the similarity between 2 sets. It is calculated by dividing the size of the intersection of the 2 sets with the size of the union of the 2 same sets.

● First we are performing an intersection on two sets using the function " A_and_B" passing sets as a list.

● After performing the intersection function, return "A_and_B" ,using this we can get the size of the intersection of the two sets.

● Similarly we are performing an union on two sets using the function "A_or_B" passing sets as a list.

● After performing the union function, return "A_or_B" ,using this we can get the size of the union of the two sets.

● Dividing the size of the intersection of 2 sets by the size of the union of 2 sets. Which we obtained from above steps to get the Jaccard coefficient.

● Formula for calculating the Jaccard coefficient is: **$J(A, B) = |A \cap B| / |A \cup B|$.**

● For each document we perform the above steps to calculate jaccard's coefficient.

```
[ ]  #intersection function
     def fun_intersection(q1,d1):
       return len(set(q1) & set(d1))
```

```
[ ]  #union function
     def fun_union(q1,d1):
       return len(set(q1)|set(d1))
```

```
[ ]  #calculation of jaccard coefficient
     def fun_calculate_jaccard(q1, d1):
       coefficient1_text = data_after_preprocessing(q1)
       coefficient1_data = data_after_preprocessing(d1)
       count1 = fun_intersection(coefficient1_text, coefficient1_data)
       count2 = fun_union(coefficient1_text,coefficient1_data)
       final_calculation = count1 / count2
       return final_calculation
```

●Returning those document which have the top 10 documents ranked by Jaccard coefficient are:

```
[ ] #Enter the query
    Query = input("Enter input query: ")
    print("The top 10 relevant documents are:\n")
    fun_Jaccard_coefficient_query(Query,jaccard_files)
```

```
Enter input query: flow of the theory
The top 10 relevant documents are:

cranfield1014
cranfield0313
cranfield1266
cranfield0920
cranfield0224
cranfield0026
cranfield0418
cranfield1124
cranfield0242
cranfield0445
```

❖ **Question 2:**

➔ **Necessary Libraries:**

- **pandas:** It offers capabilities for cleaning, combining, and reshaping data as well as data structures for effectively storing and handling huge datasets.
- **ntlk:** The Natural Language Toolkit (nltk) is used for performing tasks such as: Tokenization, Part-of-speech tagging, Sentiment Analysis,Stemming and Lemmatization etc.
- **re:** Provides operations of detecting patterns with the help of regular expressions.
- **string:** For working with text data present in dataset files.
- **np:** used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.
- **sklearn:** provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.
- **scipy:** provides more utility functions for optimization, stats and signal processing.

➔ **Data-Preprocessing:**

- Read the csv format dataset from the google drive by importing the drive.
- In Preprocessing,convert the files into lowercase.

- Tokenize the data in the next steps into small small words.
- Further remove the stopwords from the tokenized files.
- Remove the punctuation after the stopwords.
- Remove the blank spaces from the files after the punctuation.

➔ **TF-IDF weighting** :

- Converting list of words back to a string for Tf-idf Vectorizer.
- Generating the TF-IDF matrix using
  "Tfidf_matrix = tfidf_transformer.fit_transform(term_doc_matrix).toarray()"
- Computing ICF values using
  "icf_values = np.log(num_docs / np.count_nonzero(tfidf_matrix, axis=0))" .
- Further Converting tfidf_matrix and icf_values to sparse matrices.

```
(0, 19606)    0.10889205198430775
(0, 19573)    0.008007005010754938
(0, 19441)    2.5116048084301688
(0, 19355)    0.43127312668158546
(0, 19230)    0.5607384552093957
(0, 19109)    0.03217036938024452
(0, 19105)    0.09122074155620755
(0, 19024)    0.06681431475818761
(0, 18976)    0.20449198889429906
(0, 18778)    0.35359487977417275
(0, 18700)    0.09989043993963045
(0, 18641)    0.027270847768020513
(0, 18624)    0.020321011806398313
(0, 18330)    0.39967044232612003
(0, 18308)    0.2880634540053271
(0, 18040)    0.2442351105941548
(0, 17840)    0.03020974571860851
(0, 17523)    0.12169570325070507
(0, 17510)    0.18105378649676682
(0, 17483)    0.14378230336963452
(0, 17135)    0.4573280121882653
(0, 16993)    0.10384549838485468
(0, 16808)    0.060552227058777976
(0, 15977)    0.1497012419931611
(0, 15871)    0.08718029915616202
```

➔ **Next Split the dataset into training and testing sets :**

"X_train, X_test, y_train, y_test = train_test_split(tf_icf_matrix, data['Category'], test_size=0.3, random_state=42) ".

“train_test_split” function splits a dataset into two sets - a training set and a testing set.

“tf_icf_matrix “, which is the feature matrix data['Category'], which is the target variable. The test_size parameter is set to 0.3, which means that 30% of the data will be allocated to the testing set, and the remaining 70% will be allocated to the training set.

➔ **Training the Naive Bayes classifier with TF-ICF :**

○ Spliting training data and test data into 70:30 ratio and getting the Accuracy, Confusion matrix, precision, recall and f1 score.

```
accuracy = accuracy_score(y_test, predict_y)

# Accuracy score
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9731543624161074
```

```
# Confusion matrix
confusion_matrix = confusion_matrix(y_test, predict_y)
print(confusion_matrix)
```

```
[[ 87   0   2   0   2]
 [  1  82   2   0   2]
 [  1   1  72   0   0]
 [  0   0   0 118   0]
 [  0   1   0   0  76]]
```

```
precision = precision_score(y_test, predict_y, average='macro')
recall = recall_score(y_test, predict_y, average='macro')
f1 = f1_score(y_test, predict_y, average='macro')


print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 score: {f1:.4f}')
```

```
Precision: 0.9702
Recall: 0.9717
F1 score: 0.9708
```

➔ **Testing the Naive Bayes classifier with TF-ICF :**

○ Spliting training data and test data into 50:50 ratio and getting the Accuracy, Confusion matrix, precision, recall and f1 score.

```
accuracy2 = accuracy_score(y_test2, predict_y2)

# Accuracy score
print('Accuracy:', accuracy2)
```

```
Accuracy: 0.9624161073825503
```

```
# Precision

precision2 = precision_score(y_test2,predict_y2,average='macro')
print('Precision: %f' % precision)

# recall: tp / (tp + fn)

recall = recall_score(y_test2, predict_y2,average='macro')
print('Recall: %f' % recall)

# f1: 2 tp / (2 tp + fp + fn)

f1 = f1_score(y_test2, predict_y2,average='macro')
print('F1 score: %f' % f1)
```

```
Precision: 0.970217
Recall: 0.961962
F1 score: 0.961254
```

```
# Confusion matrix
confusion_matrix2 = metrics.confusion_matrix(y_test2, predict_y2)
print(confusion_matrix2)
```

```
[[162   0   3   1   2]
 [  2 147   3   1   5]
 [  3   2 123   1   1]
 [  0   0   0 160   0]
 [  1   1   2   0 125]]
```

Comparison Among different Splits				
Splits	Accuracy	Precision	Recall	F1-score
70:30	0.970917225950783	0.9717560217560217	0.9691219398977251	0.970230855958358
80:20	0.959731543624161	0.9625114604424949	0.9580153494681231	0.9597864452798662
60:40	0.9580536912751678	0.9602853980551822	0.9557449464752084	0.9572293239000491
50:50	0.9651006711409396	0.968422190708948	0.9629493749447338	0.965126842151743

➔ **Improving the classifier :**

○ Spliting training data and test data into 70:50 ratio using Ngram and getting the Accuracy, Confusion matrix, precision, recall and f1 score.

Comparison Among different Splits				
Splits	Accuracy	Precision	Recall	F1-score
70:30	0.970917225950783	0.9717560217560217	0.9691219398977251	0.970230855958358
80:20	0.959731543624161	0.9625114604424949	0.9580153494681231	0.9597864452798662
60:40	0.9580536912751678	0.9602853980551822	0.9557449464752084	0.9572293239000491
50:50	0.9651006711409396	0.968422190708948	0.9629493749447338	0.965126842151743
N-gram	0.9574944071588367	0.9594070541129364	0.9550239885463148	0.9565723795072

```python
accuracy4 = accuracy_score(y_test4, predict_y4)

# Accuracy score
print('Accuracy:', accuracy4)
```

```
Accuracy: 0.959731543624161
```

```python
[ ] # Precision

precision4 = precision_score(y_test4,predict_y4,average='macro')
print('Precision: %f' % precision4)

# recall: tp / (tp + fn)

recall4 = recall_score(y_test4, predict_y4,average='macro')
print('Recall: %f' % recall4)

# f1: 2 tp / (2 tp + fp + fn)

f1 = f1_score(y_test4, predict_y4,average='macro')
print('F1 score: %f' % f1)
```

```
Precision: 0.961155
Recall: 0.957556
F1 score: 0.958878
```

➔ **Conclusion :**

We have checked the accuracy, confusion matrix, precision, recall and f1-score by splitting the data into different - 2 split i.e- 70:30, 60:40, 50:50, 20:80 and found that we are getting the best results at 60:40 split.

❖ <u>Question 3:</u>

➔ **Necessary Libraries:**

- **math**: provides standard mathematical constants and functions.
- **pd**: used for the data analysis and dataframe of the packages.
- **np**: used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.
- **NLTK**: The Natural Language Toolkit (nltk) is used for performing tasks such as: Tokenization, Part-of-speech tagging, Sentiment Analysis,Stemming and Lemmatization etc.
- **os**: os is used for importing dataset folders from directory.
- **plt**: used to create 2D graphs and plots by using python scripts.

○ Initially load the dataset into a pandas dataframe named as “df_initial.

+ Code + Text

```
[ ] #read the dataset
    df_initial=pd.read_csv('/content/drive/MyDrive/Mtech/Sem2/IR/My_assignment/Assignment2/IR-assignment-2-data (2).txt',sep=" ",header=None)
```

```
[ ] #printing the first 5 entries
    df_initial.head()
```

	0	1	2	3	4	5	6	7	8	9	...	129	130	131	132	133	134	135	136	137	138
0	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:2	129:9	130:124	131:4678	132:54	133:74	134:0	135:0	136:0	NaN
1	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:0	129:8	130:122	131:508	132:131	133:136	134:0	135:0	136:0	NaN
2	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:2	129:8	130:115	131:508	132:51	133:70	134:0	135:0	136:0	NaN
3	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:82	129:17	130:122	131:508	132:83	133:107	134:0	135:10	136:13.35	NaN
4	1	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:11	129:8	130:121	131:508	132:103	133:120	134:0	135:0	136:0	NaN

5 rows × 139 columns

○ Further pick the qid:4 from the dataset named df_initial and store it in a new dataframe named as “df_updated”.

df_updated = df_initial[df_initial[1] ==’qid: 4’]

○ Sort the dataset in decreasing order in terms of relevance.

○ Created a file named as “Filecreated.txt” that rearranges the query-url pairs in order of the maximum DCG (discounted cumulative gain).

(i). Maximum DCG based query-url reaarrangment.

```
[ ] #creating the file
    created_data={}
    for iterator in range(0,len(df_updated.index)):
        check1=df_updated.at[iterator,1]
        if ("qid:4"==check1):
            created_data[iterator]=df_updated.at[iterator,0]
```

```
def queryurlfile(df, created_data):
    newdataframe=df.drop((df.index[len(created_data):]))
    #newdataframe.to_csv("fileobtained.csv")
    np.savetxt('Filecreated.txt',newdataframe.values,delimiter=" ",fmt='%s')

queryurlfile(df_updated,created_data)
tupledocid=created_data.items()
tupledocid=list(tupledocid)
created_data=sorted(created_data.items(),key=lambda pairs:(pairs[1],pairs[0]),reverse=True)
```

○ Now in order to calculate nDCG we need DCG and mDCG. So I have calculated both along with the total number of files that could be made.

```
Maximum Discounted Cumulative Gain (MDCG) is: 20.989750804831445

Discounted Cumulative Gain (DCG) is: 12.550247459532576

Total files: 19893497375938370599826047614905329896936840170566570588205180312704857992695193482412686565431050240000000000000000000000
```

○ Further computed the nDCG at 50 and for the whole dataset.

(ii) nDCG Calculation

```
# nDCG at postion 50
top_50=df_updated.iloc[:50]
top_50_sorted = top_50.sort_values(by=[0], ascending=False, ignore_index=True)
top50_NDCG=0
top50_DCG5=0
for i in range(top_50_sorted.shape[0]):
  relevance = top_50_sorted.iloc[i,0]
  if(i==0):
    top50_NDCG = top_50_sorted.iloc[i,0]
  else:
    top50_NDCG = top50_NDCG + relevance/math.log(i+1, 2)

#calculating the NDCG and DCG for top 50 items.
for i in range(top_50.shape[0]):
  relevance = top_50.iloc[i,0]
  if(i==0):
    top50_DCG5 = top_50.iloc[i,0]
  else:
    top50_DCG5 = top50_DCG5 + relevance / math.log(i+1, 2)

top_normalizedDCG=top50_DCG5/top50_NDCG
print(f'Normalized Discounted Cumulative Gain (nDCG) at position 50 is: {top_normalizedDCG} \n')
```

```
Normalized Discounted Cumulative Gain (nDCG) at position 50 is: 0.5253808413557646
```

```
[ ] # nDCG for whole dataset
    nDCG_Whole = DCG/maxDCG

    print(f'Normalized Discounted Cumulative Gain (nDCG) for whole dataset is: {nDCG_Whole} \n')
```

```
Normalized Discounted Cumulative Gain (nDCG) for whole dataset is: 0.5979226516897831
```

○ Performing the operation on the 75th feature of the dataset and plotting the precision and recall graph.

```
Text(0.5, 1.0, 'Precison-Recall Curve at qid:4')
```