NLP Based Search Engine

Developed By:

1. Loai Bitawi
2. Mohannad Hindi
3. Rabee Qasem

Table of Contents

[**Abstract** 3](#_Toc64235428)

[**1.** **Introduction** 3](#_Toc64235429)

[**1.1.** **Background** 3](#_Toc64235430)

[**1.1.1.** **Pre-processing** 5](#_Toc64235431)

[**1.1.2.** **Check eligibility layer** 5](#_Toc64235432)

[**1.1.3.** **Processing phase** 6](#_Toc64235433)

[**1.1.4.** **Ranking result:** 7](#_Toc64235434)

[**2.** **Problem Statement** 7](#_Toc64235435)

[**3.** **System analysis** 8](#_Toc64235436)

[**3.1.** **System Requirement** 8](#_Toc64235437)

[**3.2.** **Functional requirement** 8](#_Toc64235438)

[**3.3.** **System architecture** 9](#_Toc64235439)

[**3.4.** **Development Methodology** 10](#_Toc64235440)

[**3.5.** **Use case diagram.** 11](#_Toc64235441)

[**4.** **System design** 12](#_Toc64235442)

[**4.1.** **Class Diagram** 12](#_Toc64235443)

[**4.2.** **Interface Design** 12](#_Toc64235444)

[**4.3.** **Database Design** 13](#_Toc64235445)

[**5.** **Implementation** 16](#_Toc64235446)

[**5.1.** **Employed Techniques** 16](#_Toc64235447)

[**5.2.** **Back-End Side** 16](#_Toc64235448)

[**5.3.** **Data Preprocessing** 17](#_Toc64235449)

[**5.4.** **Front-End Development** 34](#_Toc64235450)

[**5.5.** **Code Documentation** 40](#_Toc64235451)

[**5.5.1.** **Data Parser** 40](#_Toc64235452)

[**5.5.2.** **Positional Indexer** 42](#_Toc64235453)

[**5.5.3.** **Check Search (Stemming, URL)** 43](#_Toc64235454)

[**5.5.4.** **Tokenizer** 45](#_Toc64235455)

[**5.5.5.** **Main Window** 46](#_Toc64235456)

[**5.5.6.** **Boolean Function** 63](#_Toc64235457)

[**5.5.7.** **Zone Based Matching** 66](#_Toc64235458)

[**5.5.8.** **Cosine Similarity** 69](#_Toc64235459)

[**5.5.9.** **Semantic Based Matching** 72](#_Toc64235460)

[**5.5.10.** **Proximity Matching** 74](#_Toc64235461)

[**5.5.11.** **Regular Expression** 78](#_Toc64235462)

[**5.5.12.** **Exact Match** 79](#_Toc64235463)

[**6.** **Conclusions and Future Work** 81](#_Toc64235464)

# **Abstract**

NLP is a branch of Data science which received a lot of interest by the researchers in the last decade in many fields due to its magnificent impact of analyzing, understanding and deriving information from the text data in a smart and efficient manner [1]. Moreover, the increasing demand on efficient and convenient information retrieval systems for organizations made NLP pipelines the solution to this kind of problems.

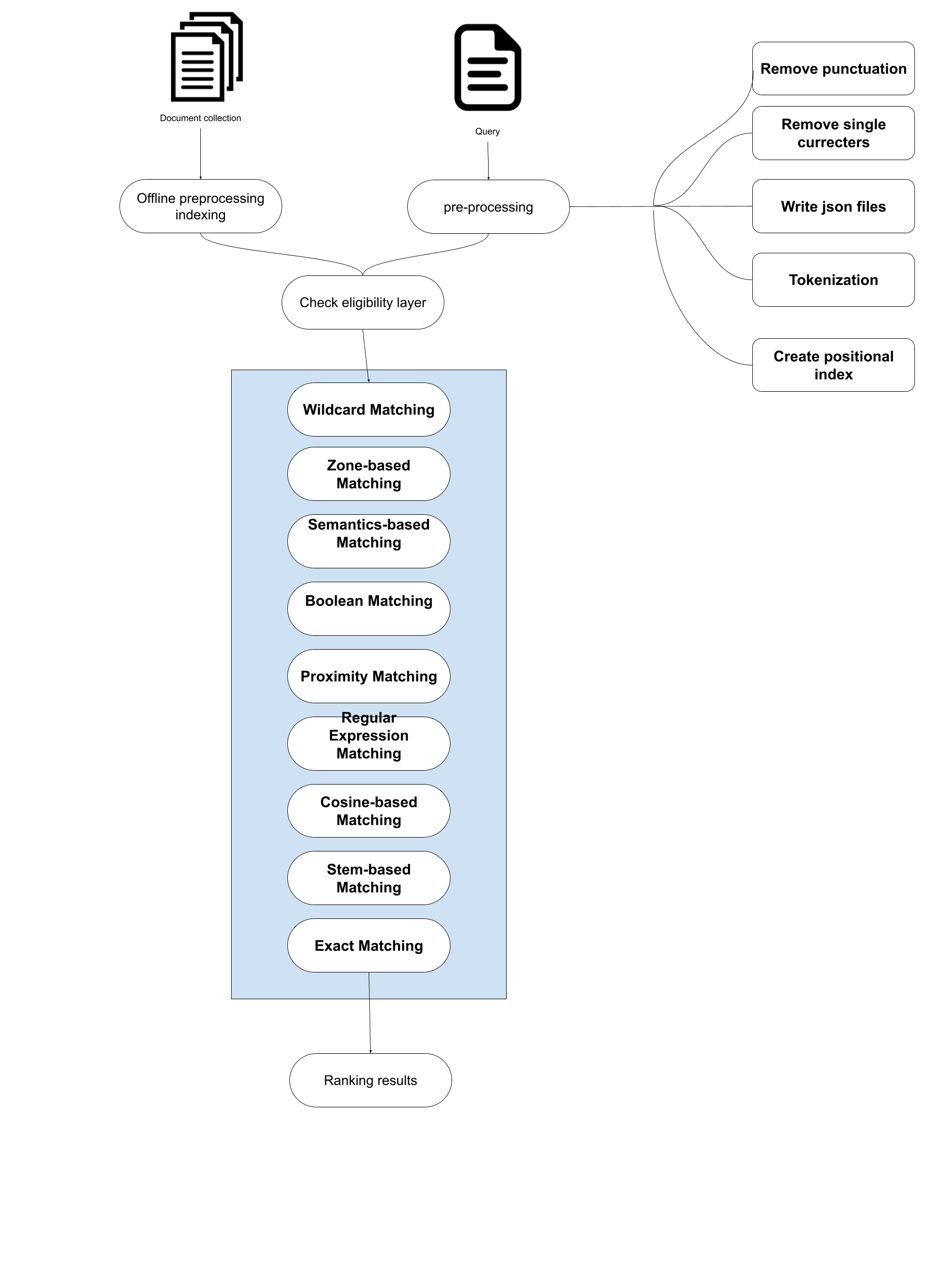
This project implements some of the NLP techniques that increases the retrieval system performance, and adapts with various user query needs. The techniques implemented are Boolean queries, wildcard matching, zone-based matching, semantic-based matching, regular expressions, proximity queries, stem-based matching, exact matching, and stop words removal. The project also implements interactions between different techniques according to eligibility rules that ensure the best use of the system without any faults. The system follows lean processes design. That is; the process is done in one way that makes errors the minimum.

# **Introduction**

## **Background**

In this project, we will discuss in brief the pipeline that has been used in our system, and the flow of the process in each phase of the pipeline. The dataset used in the project is the iaprtc12 dataset which is a tourism-related dataset where each document represented as XML file and each file contains 5 tags: (DOCNO, TITLE, DESCRIPTION, NOTES, LOCATION, DATE, IMAGE). The corpus in our system is the textual data in each tag. Moreover, each document is associated with an image that represents the data in the document's tag.

The pipeline is illustrated in the following figure, containing the 4 phases of the pipeline:



### **Pre-processing**

This step is divided into 5 phases:

* Document transformation: In order to get a faster way of handling and searching the document, we converted the whole dataset into a JSON file, which is one way to represent a NoSQL database that will ensure better and faster performance of the system. We converted each document as a key value into our json file and the tags and text are represented as values for this key and for petter indexing.
* Remove punctuation from the text for better tokenization performance.
* Remove single letters such as “i, a... etc.”.
* Tokenize the whole text for each tag and assign the list into key tag as a value for it.
* Write json files for each Data preparation phase in all scenarios (with and without stop words, with and without stemming...etc.) and save it as an offline repository for later use.
* Positional index creation: creating a positional index which shows each word of the corpus in which document is mentioned and on which index of the text is presented and we discussed in detail in the Database design section.

### **Check eligibility layer**

In our system we integrated a lot of NLP information retrieval techniques; Boolean Query, Exact Matching, Stop words Removal, Zone-based Matching, Cosine Similarity, Semantic-based Matching, Regular Expression, and Wildcard Matching. Some of these techniques have interaction with each other. For example, zone-based matching and Boolean query can interact with each other since zone matching treats each zone as a single query and Boolean query can be applied on zone level. On the other hand, Boolean queries can’t interact with regular expressions since Boolean requires the existence of the two tokens if there is an “and” between them, and regular expression searches for a pattern, which might match more than one token. Thus, we need a layer which controls the process of selecting what function can or can’t interact with each other.

### **Processing phase**

This process is concerned with processing the data based on the flag that received from the Check eligibility layer and it have the 10 techniques that we use to process the data which are as following

* Boolean matching: is the process of retrieving the document that contain both or either of set of words in the same document. It also includes the negation of words.
* Exact matching: is the process of retrieving the document that contain the exact expression given by the user
* Zone based matching: The process of defining specific part of the document like the title or notes and the search function will only search on these sections and retrieve word that exist in it
* Cosine similarity: we used the cosine similarity to measures the similarity between two documents by converting these two documents into vectors and the angle between two vectors determine if these documents are similar or not [5]
* Semantic based matching: We integrated in the system the most widely used lexical databases which they are Wordnet and YAGO to handle if the user wants to retrieve document that have a semantic meaning in of the query [6]
* Proximity matching: is the process of searching the documents that contain two or more phrases which are separated by certain number or words from each other
* Regular expressions: is a sequence of characters that define a search pattern in our system [3]
* Stop word removal: Stop words are words like (we, out, you...etc.) which are filtered out before processing of corpus
* Wild card matching: the system handles leading, non-constrained wildcard queries
* Stem-Based matching: using 3 predefined stemmers (Porter, Snowball, and Lancaster), and a customized stemmer.

### **Ranking result:**

Since queries given by the user might include different tokens, or different query types, it leads to using different techniques to retrieve the results. The best way to do it is by splitting the tokens by the type of the queries needed. For example, text like “Plaza hotel and large building with great view” is treated as following:

1. Standard matching for the tokens “Plaza, large, building, with, great, view
2. Boolean matching for “hotel and large”

In such cases, the system collects documents for each type, then combining the results. The combination of these results might take one of two forms; combining the two lists, or finding the intersection between them. The choice depends on the query type. However, for the first type, there must be a way to rank the documents based on their relatedness. Thus, documents should be ranked. In this project, the ranking of documents is based on the frequency of the document. If the document appears in the results of all tokens, the document will appear on the top.

# **Problem Statement**

When a user has a lot of files that are archived in their databases and he/she would like to retrieve a set of documents that contains specific information, the user has to search for all the documents to get it.

Our system will solve this issue by implementing an NLP technique to help the user retrieve the information of a set of documents based on specific query or functionality and retrieve some similar document in case the query doesn’t match any exact documents.

# **System analysis**

## **System Requirement**

Minimum Hardware Requirements

* RAM: 8 GB or more
* Hard disk space: 1 GB (minimum) free space available

Additional Software Requirements

* Windows 7 or more

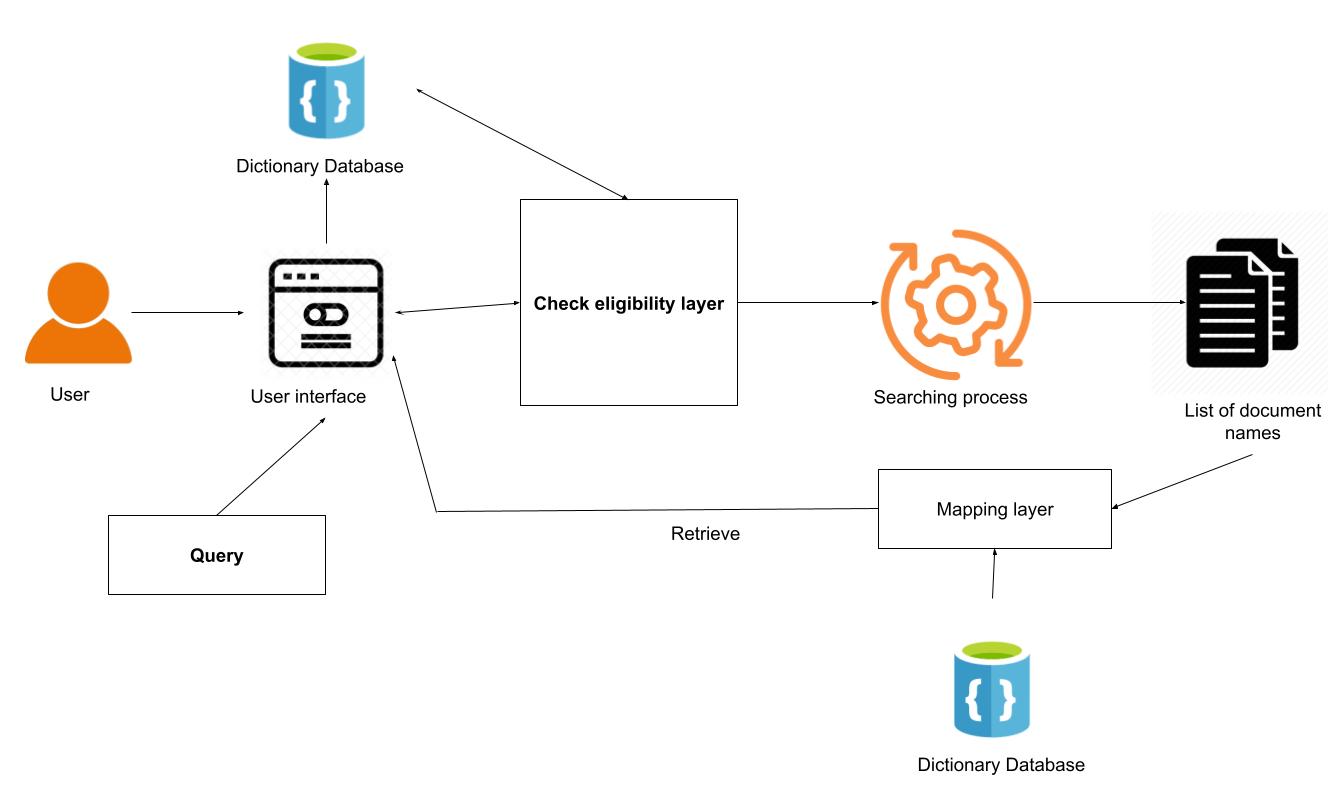
## **Functional requirement**

|  |  |
| --- | --- |
| Functional requirement No. | Functional requirement description |
| 1 | The user should be able to enter any length of text |
| 2 | The user should be able to enter a URL |
| 3 | The user should be able to enter a regular expression |
| 4 | The user should be able to enter a Boolean query |
| 5 | The user should be able to decide if he/she want to retrieve the similar document (cosine TF-IDF) |
| 6 | The user should be able to decide if he/she want to retrieve exact match document |
| 7 | The user should be able to decide if the document will take into consideration the stop words and stemming |
| 8 | The user should be able to decide if he/she want to retrieve the document with the extrinsic resources |
| 9 | The user should be able to perform wildcard queries |
| 10 | The user should be able to perform proximity matching queries |
| 11 | The user should be able to view the list of results and document content within the interface |

## **System architecture**

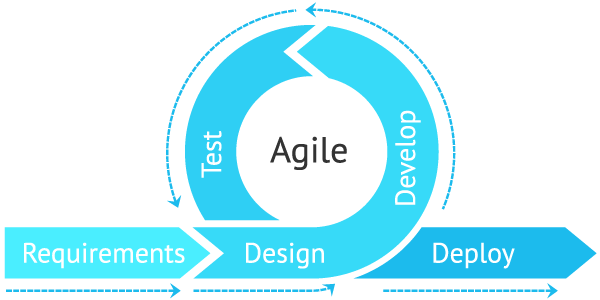
Our system works as following:

1. The user enters a query using our interface
2. transfer what checkboxes are selected into the Flag layer
3. Decide what JSON file that the system needs to handle the process into our Flag layer
4. Then the check eligibility layer maps what functions are we using to transfer it to process layer
5. The process layer uses our JSON file to implement the functions and get a list of the file name that meets user requirements
6. The list of filenames then transferred into the mapping layer which retrieves the file content and views it on our interface

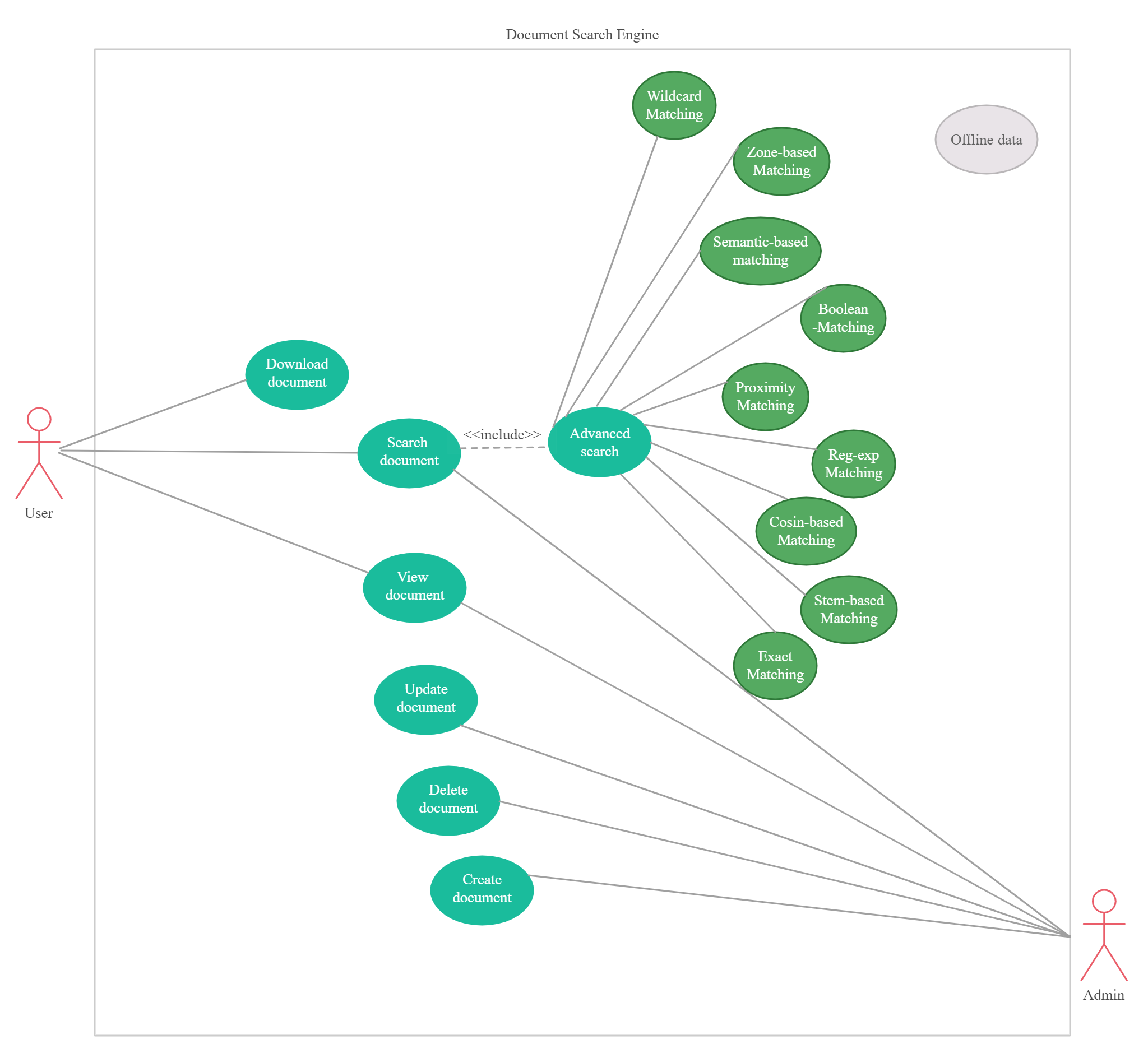


## **Development Methodology**

The development methodology that we are following in our system is ‘Agile Methodology’, we started gathering requirements to have a full understanding of the system. This system helps us to complete our prototype with the least time and the required results. Although we are administrators of the application, we engage the user and product owner in the system developed through the feedback that they give us.



## **Use case diagram.**



# **System design**

## **Class Diagram**



## **Interface Design**

Our interface design is user friendly and easy to use by the user since the user is able to choose what functionality want to implement by only checking the intended check box and we make sure when the user chooses a certain functionality all other functions that don’t work with this certain feature are turned off to minimize the user error and mistakes.

When the user enters a query and our system retrieve the documents, we added a feature to make the user able to view the documents and the photos within these documents in the same place of our interface and that help the user to explore the result within the place rather than open the document outside our software.

## **Database Design**

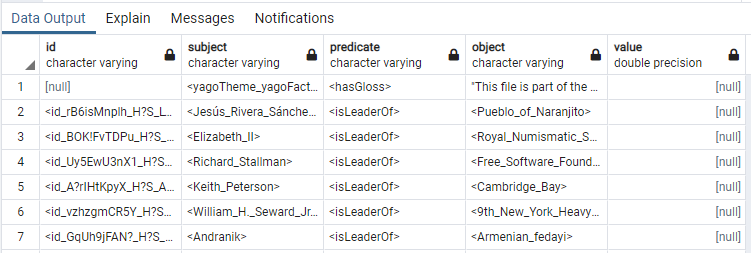
In our system, we decided to choose NoSQL Database in order to make our system performance faster in searching and retrieving the data so we converted all the documents into one single JSON file each file name represent the key for the JSON file and each key has a dictionary value which contain the tags as a key and the token of the list as a value of key This method reduced the retrieving time from 1 hour to 3 seconds

"112.eng": {  
 "DOCNO": [  
 "annotations",  
 "00",  
 "112.eng"  
 ],  
 "TITLE": [  
 "Excursion",  
 "with",  
 "the",  
 "godchildren"  
 ],  
 "DESCRIPTION": [  
 "About",  
 "20",  
 "kids",  
 "in",  
 "traditional",  
 "clothing",  
 "and",  
 "hats",  
 "waiting",  
 "on",  
 "stairs",  
 "house",  
 "and",  
 "green",  
 "wall",  
 "with",  
 "gate",  
 "in",  
 "the",  
 "background",  
 "sign",  
 "saying",  
 "that",  
 "plants",  
 "can't",  
 "be",  
 "picked",  
 "up",  
 "on",  
 "the",  
 "right"  
 ],  
 "NOTES": [  
 "None"  
 ],  
 "LOCATION": [  
 "Quilotoa",  
 "Ecuador"  
 ],  
 "DATE": [  
 "April",  
 "2002"  
 ],  
 "IMAGE": [  
 "images",  
 "00",  
 "112.jpg"  
 ],  
 "THUMBNAIL": [  
 "thumbnails",  
 "00",  
 "112.jpg"  
 ]  
 },

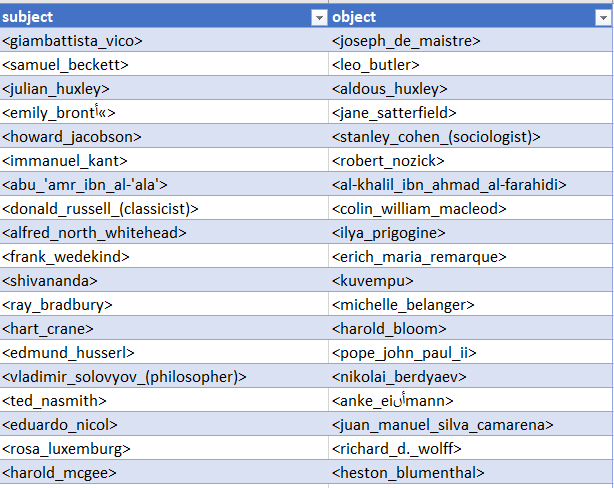
On the other hand, some aspects of the system like Boolean matching, exact match, and proximity matching needed some phrase search techniques so we decided to go with Positional index which enable faster phrase search performance. The idea of positional indexing that we assigned each word in our documents as a key value of the json file and then we add the document that a certain phrase is mention in it and which position this phrase are in the same document the json file is look like this:

"excursion": [  
 7,  
 {  
 "0": {  
 "TITLE": [  
 0  
 ]  
 },  
 "8255": {  
 "TITLE": [  
 4  
 ]  
 },  
 "8391": {  
 "NOTES": [  
 12  
 ]  
 },  
 "8422": {  
 "TITLE": [  
 5  
 ]  
 },  
 "8563": {  
 "TITLE": [  
 0  
 ]  
 },  
 "12866": {  
 "TITLE": [  
 0  
 ]  
 },  
 "13799": {  
 "TITLE": [  
 4  
 ]  
 }  
 }  
 ],

About the semantic resources unlike Wordnet which have a build in function from NLTK that can be used easily YAGO on the other had we had to download the TSV file from YAGO website and save it into Postgres database and the shape of the data look like this



And postgres give you the ability to connect directly to python file and retrieve data frame the database but the problem with this method is that it takes time to get the data, so in order enhance the performance we had to save as csv file and using as offline repository and that increase the performance.



# **Implementation**

In this chapter, in-depth analysis of technical development for each feature in the system, including code snippets and output.

## **Employed Techniques**

As any full stack development projects, it consists of two main sides: Front-End, and Back-End.

The next subsections will discuss the two sides in detail.

## **Back-End Side**

This side contains all the background development of the system including:

1. Data Preprocessing
2. Functions Development
3. Eligibility Rules
4. Results Ranking
5. Result representation

In this context, some doubts about system complexity would rise on the surface, and how the system would handle such interactions between functions. Moreover, the system must be as adaptive as possible with the user. This means that the system should predict user misuse of the functions, syntax problems and others. However, the backend development of this project uses agile development, which allows the developer to deal with features and develop the interactions easily and efficiently.

The development of this phase using python, with dependencies on the following libraries:

* Pandas and NumPy
* Json library
* NLTK
* Re Library
* Collections Library
* Beautiful Soup
* Requests
* TQDM
* XML Element Tree
* Math Library
* OS library

The next sections will discuss in detail all five steps of backend development.

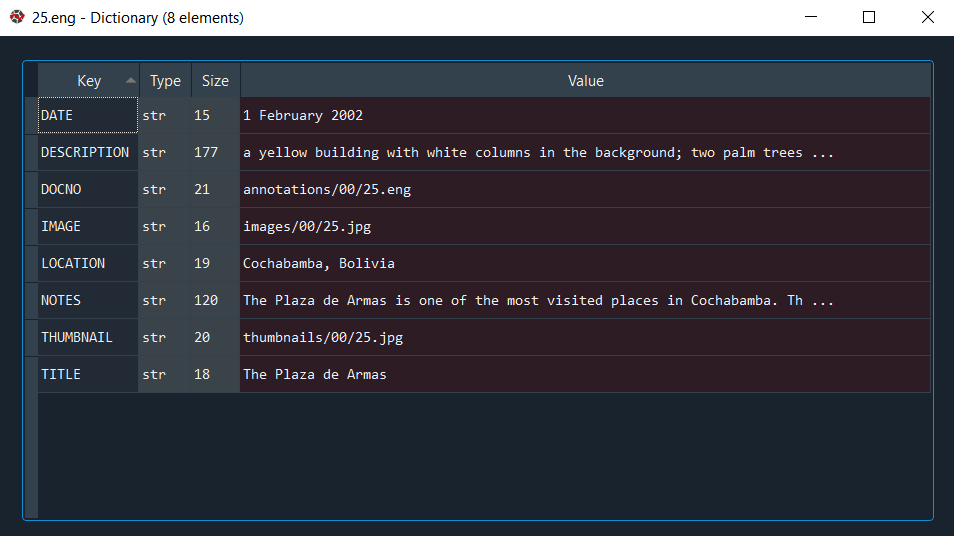
## **Data Preprocessing**

This is the first step of the pipeline, where the dataset is prepared to be utilized in the system. At first, the source of the data contains 20,000 XML documents that contain semi-structured textual data, with a file that contains 20,000 JPG images each related to a document. If the system handles the XML files directly, the processing time will be very high. To solve this issue, all documents must be parsed and combined in one source. One might think of data frames as a solution to combine files. Although it's efficient and easy to deal with, its performance is low. Thus, the project utilizes dictionaries saved as JSON files. JSON performance is much higher than data frames and it is easy to use as well.

To parse the data from XML, we used XML Element Tree library that finds the nodes and values in the XML document and parses the data to a dictionary. The following code represents this step:

|  |
| --- |
| main\_path='D:/Data science Master/NLP/Project/annotations\_complete\_eng/' entries = os.listdir(main\_path) path=[] Cols=['DOCNO','TITLE','DESCRIPTION','NOTES','LOCATION','DATE','IMAGE','THUMBNAIL'] dataset=pd.DataFrame(data=None, columns=Cols) indexed={} subdic={} for folder in entries:  sub=os.listdir(main\_path+folder+'/')  for file in sub:  filename = main\_path+folder+'/'+file   with open(filename, 'r') as content:  root = et.parse(content).getroot()  path.append(filename)  subdic={}  for col in Cols:  value = root.find(col).text  #if value is not None: value=value.lower()  subdic[col]=value  indexed[file]=subdic |

The dictionary “Indexed” has the shape of Key: Value, where keys are the document names (ex: 25.eng) and the value is a dictionary of dictionaries that contains each field of the data and its value. The following figure shows an example of a parsed XML file:



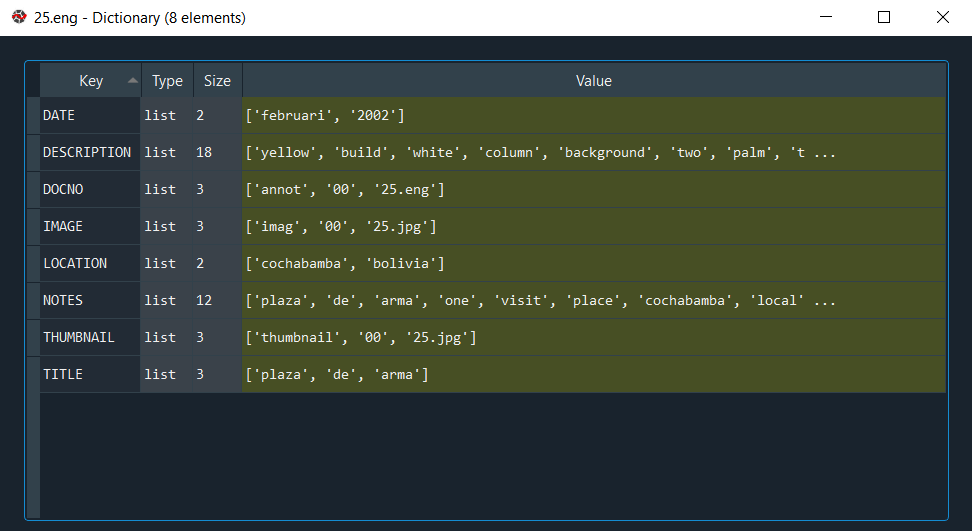
After this step, data should be tokenized, stop words removed, stemmed, and their indexes should be structured. These processes consume much time due to data size. Thus, it is done in the background and offline files of the output of these processes are saved on the hard drive. Moreover, to manage the interactions with features, any effect on the data structure due to this interaction should be done offline. Thus, the data should be tokenized with and without stop words, then stemmed with and without stop words, and finally constructing indexes for each of the output files. The following code snippet defines a class that is used for different tokenization scenarios (text tokenization, single file tokenization, unique tokens, and dataset tokenization):

#toknize single string  
 #the function receive a string text  
 def tokenizec(self,text):  
 regx="\w\w\*[.|:|?|'||...|!|-|()]?\w\w\*"  
 tk=re.findall(regx,text)  
 tk=[item.lower() for item in tk]  
 return tk  
  
 #this function return set list of tokens (remove dublicates tokenz)  
 #the function recive a string text  
 def unitoknizer(self,text):  
 ListOfTokens=self.tokenizec(text)  
 SetTkn=set(ListOfTokens)  
 return SetTkn  
   
 #this functino return list of token for a single file for example data['3843.eng']  
 def singlefiletk(self,single\_file):   
 listt=[]  
 for tag in single\_file.keys():  
 listt.append(single\_file[tag])  
 string=' '.join(map(str, listt))  
 listt=self.unitoknizer(string)  
 #listt=self.tokenizec(string)  
 return listt  
  
 #used to tokenize the whole document, input is a dictionary  
 def datatoken(self,datafile):  
 list1={}  
 for fn in datafile.keys():  
 tags=[]  
 tokens=[]  
 for tag in datafile[fn]:  
 tokens.append(self.tokenizec(str(datafile[fn][tag])))  
 tags.append(tag)  
 list1[fn]=dict(zip(tags,tokens))  
   
 return list1

The next step is to stem the data. The system uses 4 different stemmers; Porter stemmer, Snowball Stemmer, Lancaster Stemmer, and customized stemmer. The first 3 tokenizers are predefined in the NLTK library and can be used easily by calling the function and passing the tokenized text to it. The customized tokenizer uses rule-based stemming and lemmatization of the most common cases. The code of this stemmer can be found in “custom\_stemmer.py”, and it is added in section 4.2. The following code represents the process of tokenizing and stemming the data and saving the results into JSON files:

tk=NlpTokenizer()  
tokens= tk.datatoken(indexed)  
with open('tokenized.json', 'w') as json\_file:  
 json.dump(tokens, json\_file,indent=4)  
  
sr=searcher()  
ps\_stemmed=sr.Porter\_Stemmer(indexed)  
with open('ps\_stemmed.json', 'w') as json\_file:  
 json.dump(ps\_stemmed, json\_file,indent=4)  
  
sb\_stemmed=sr.Snowball\_Stemmer(indexed)  
with open('sb\_stemmed.json', 'w') as json\_file:  
 json.dump(sb\_stemmed, json\_file,indent=4)  
  
lc\_stemmed=sr.Lancaster\_Stemmer(indexed)  
with open('lc\_stemmed.json', 'w') as json\_file:  
 json.dump(lc\_stemmed, json\_file,indent=4)  
  
custom\_stemmed=sr.Customized\_Stemmer(indexed)  
with open('custom\_stemmed.json', 'w') as json\_file:  
 json.dump(custom\_stemmed, json\_file,indent=4)

The output of this step is 10 JSON files. The following figure shows an example of an output from porter stemmed data with removal of stop words:

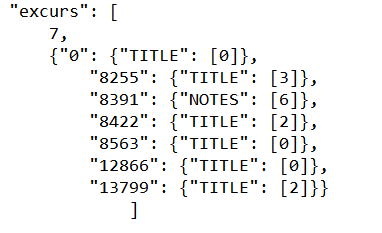


Nevertheless, to increase the performance of search through the documents, indexing of word positions in the documents should be constructed offline. The code of positional indexing constructs a dictionary with unique tokens as keys, and the value is a list that contains a frequency of a word, and a dictionary that contains all documents that includes the word, and for each document in what field it was mentioned, and at what position in the field it was mentioned.

The code that develops these indexes is in “positional\_indexer.py”. The following code snippet uses the positional indexing class to construct the indexes:

stem\_no\_stop=['without\_stopword\_tokenized.json', 'without\_stopword\_ps\_stemmed.json', 'without\_stopword\_sb\_stemmed.json',  
 'without\_stopword\_lc\_stemmed.json', 'without\_stopword\_custom\_stemmed.json']  
  
stem\_with\_stop=['tokenized.json', 'ps\_stemmed.json', 'sb\_stemmed.json', 'lc\_stemmed.json', 'custom\_stemmed.json']  
pos=pos\_index()  
for file in tqdm(stem\_no\_stop):  
 data=json.loads(open("stemmers/"+file).read())  
 file\_map, index=pos.pos\_indexer(data, 0)  
 with open('pos\_index/pos\_'+file, 'w') as json\_file:  
 json.dump(index, json\_file,indent=4)  
  
for file in tqdm(stem\_with\_stop):  
 data=json.loads(open("stemmers/"+file).read())  
 file\_map, index=pos.pos\_indexer(data, 0)  
 with open('pos\_index/pos\_'+file, 'w') as json\_file:  
 json.dump(index, json\_file,indent=4)  
   
with open('pos\_index/file\_map.json', 'w') as json\_file:  
 json.dump(file\_map, json\_file,indent=4)

The following figure illustrates an example of the output:



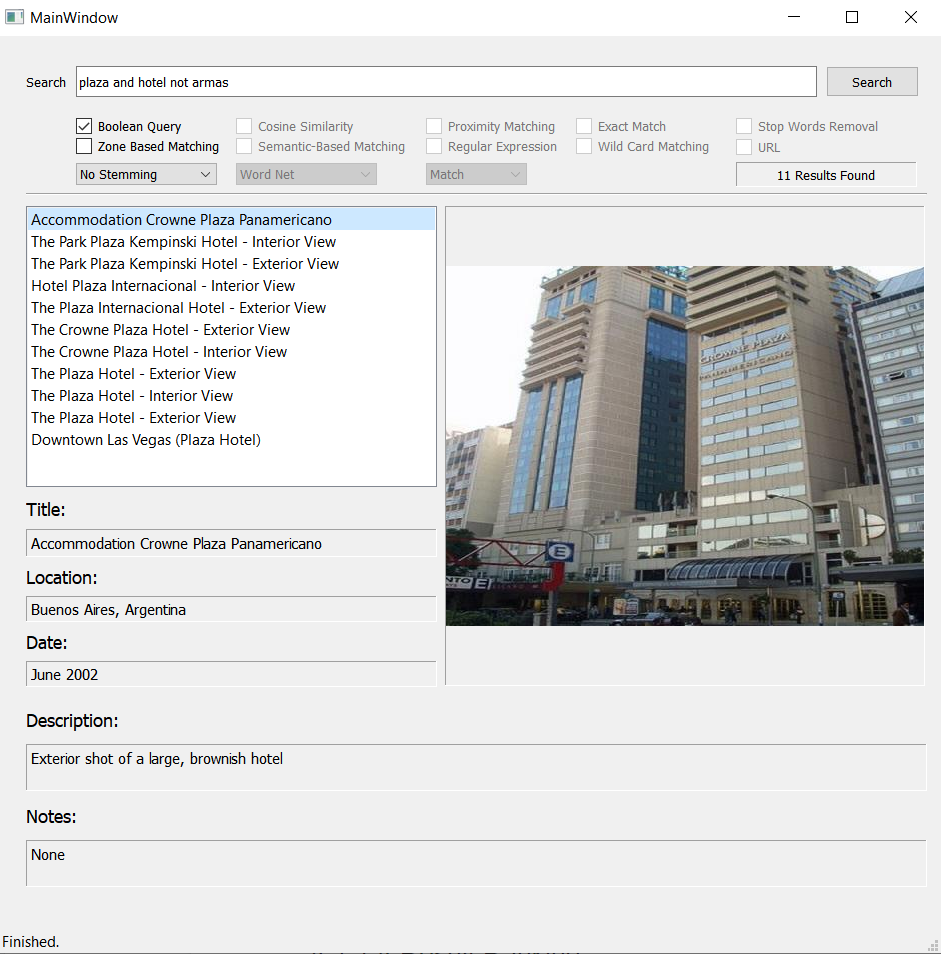
In the above figure, the token “excurs” is mentioned 7 times in the documents, and it was mentioned in the title of document number 8255 at position 3.

Finally, the same steps of tokenization and stemming is done on the query side upon customer request in the main GUI, and for the URL data, the data is parsed and cleaned using Beautiful Soup library. The function “if\_url” in the searcher class that is implemented in “Check\_search.py” parses the data and returns the text of the URL.

After the data preprocessing phase, each of the functions mentioned in the requirements is implemented, and the interaction between these functions as well. The functions are:

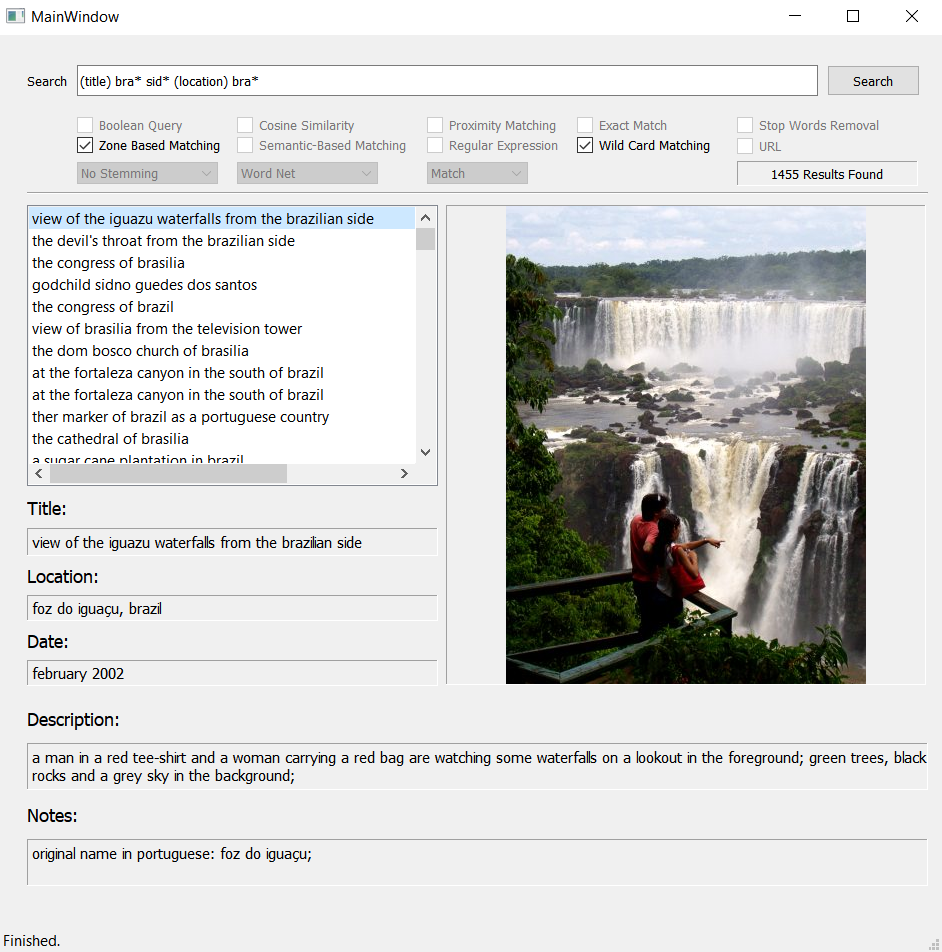
1. Boolean Matching:   
   This function searches for the words (and, or, not) in the given tokens, splits the text according to them, then constructs a one hot vector for each token where the hot ones represent the occurrence of the token in the document. The vector size is the number of documents, and then logical and, or, and not is performed on the vector, and then retrieving the list of documents. The file “boolean\_query.py” represents the code used for this feature. The function returns the set of documents that matches the query. It can handle multiple operators in one query. The function also checks for the operators in the text; if there are missing operators, it fills them with “and” operators. Moreover, it checks for a single-term query and handles the matching with another vector of all hot ones to perform the logical operation.

The following figure represents the output of a Boolean query:



1. Zone Based Matching:

This function searches for query tokens in a specific zone/zones. It can search in all document zones except for the date zone. The function checks for the syntax “(zone) text”. If the user didn’t use the syntax, it displays a warning message with directions on how to use it. It can also deal with repeated zones. For example: (title) plaza (title) hotel. It searches for each token in its specific zone, then combines all results and sorts them according to their frequency in the result (if a document is retrieved for the two tokens it appears on the top). The function can also have multiple sub-functions that are performed on the zone level; it can handle cosine similarity, wild cards, semantic expansion of the query, proximity matching, exact matching, stemming, and stop words removal. The following figure shows an example of using wildcard on the title zone:



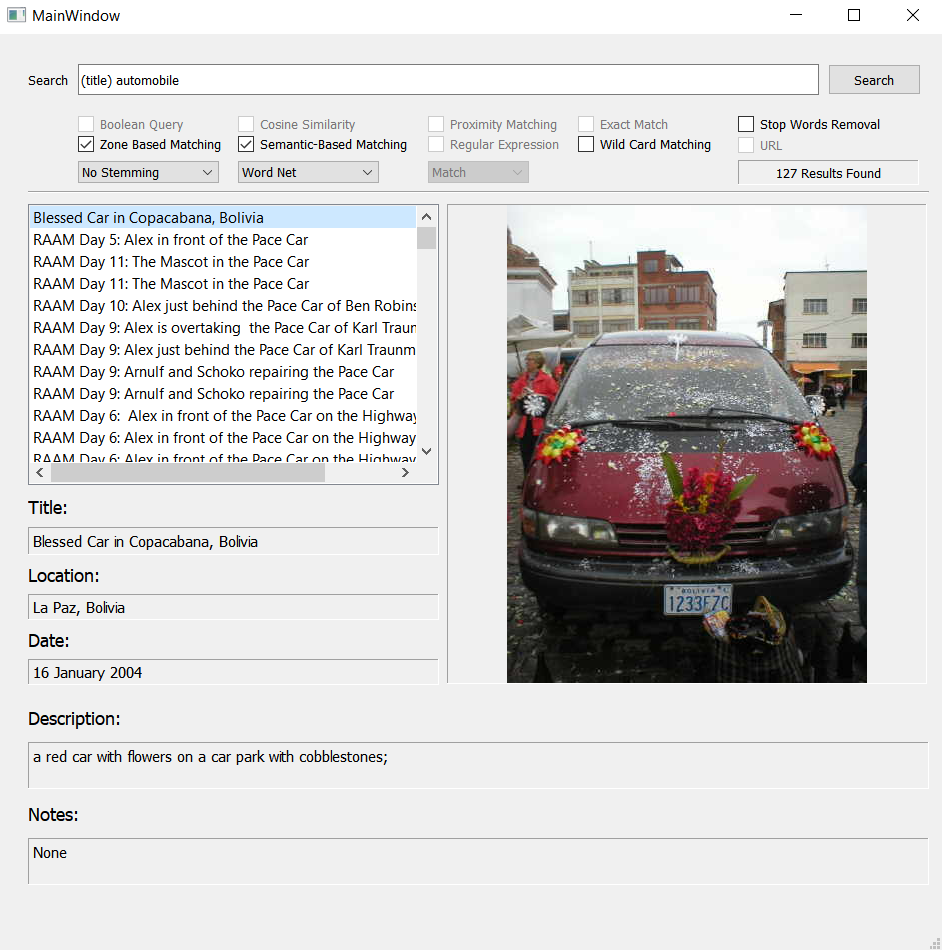
1. Cosine Similarity:

This function generates TFIDF vectors for the whole dataset, then uses these vectors to create the vector for the query, then finding the cosine distance between the query and each of the documents, then ranking the results according to the similarity value. All documents with similarity score <0.01 are excluded. The function can be used with zone-based function, stop words removal, stemming, and URL queries. The code that represents the function can be found in “cosine.py”. The following figure is an example of using cosine function on the description zone and using porter stemmer:



1. Semantic-Based Matching:

This function uses extrinsic semantic resources (Word Net and Yago5) to find the synonymous terms of each token, then expanding the query with these synonyms and searching for matching. The function uses NLTK word net corpus for wordnet, and an offline CSV file of YAGO knowledge for Yago. the function can be used with zone matching, stop words removal, stemming, and URL queries. the semantic function can be found in “semantic.py”, and the following figure represents the output on a zone for the word automobile:

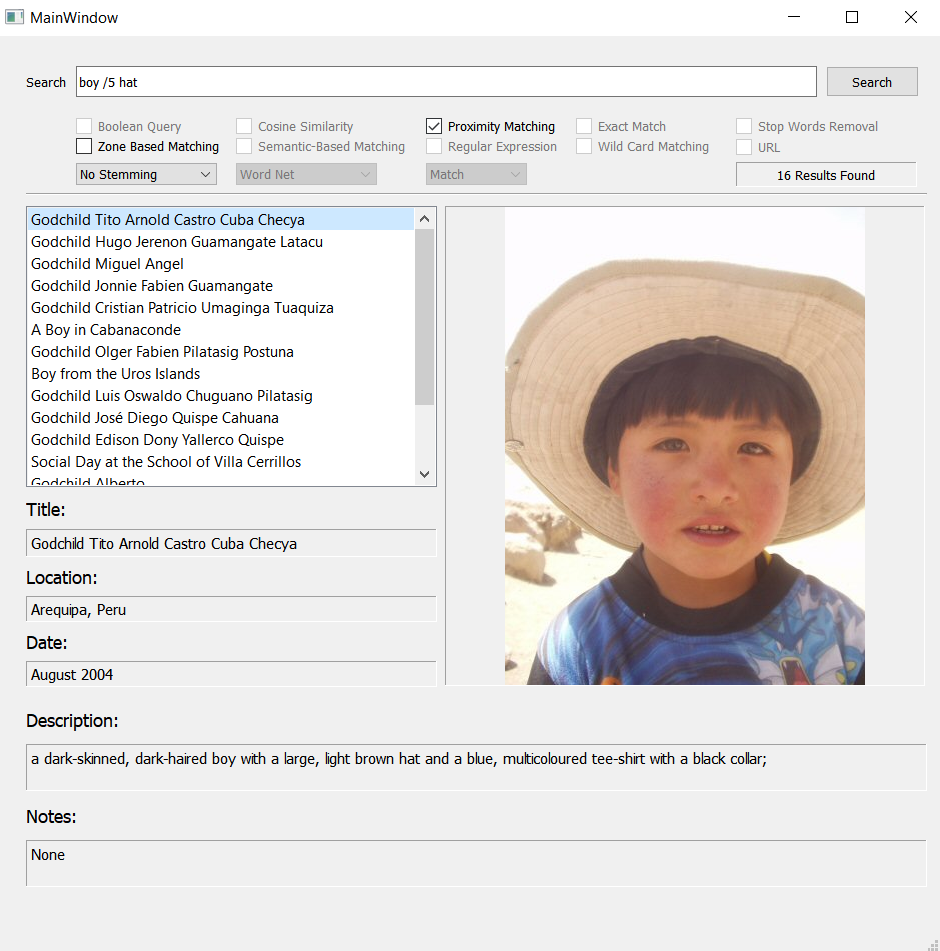


As seen in the figure, the word automobile doesn’t exist in the titles, but other words like car exist, which is a synonym for automobile.

1. Proximity Matching:

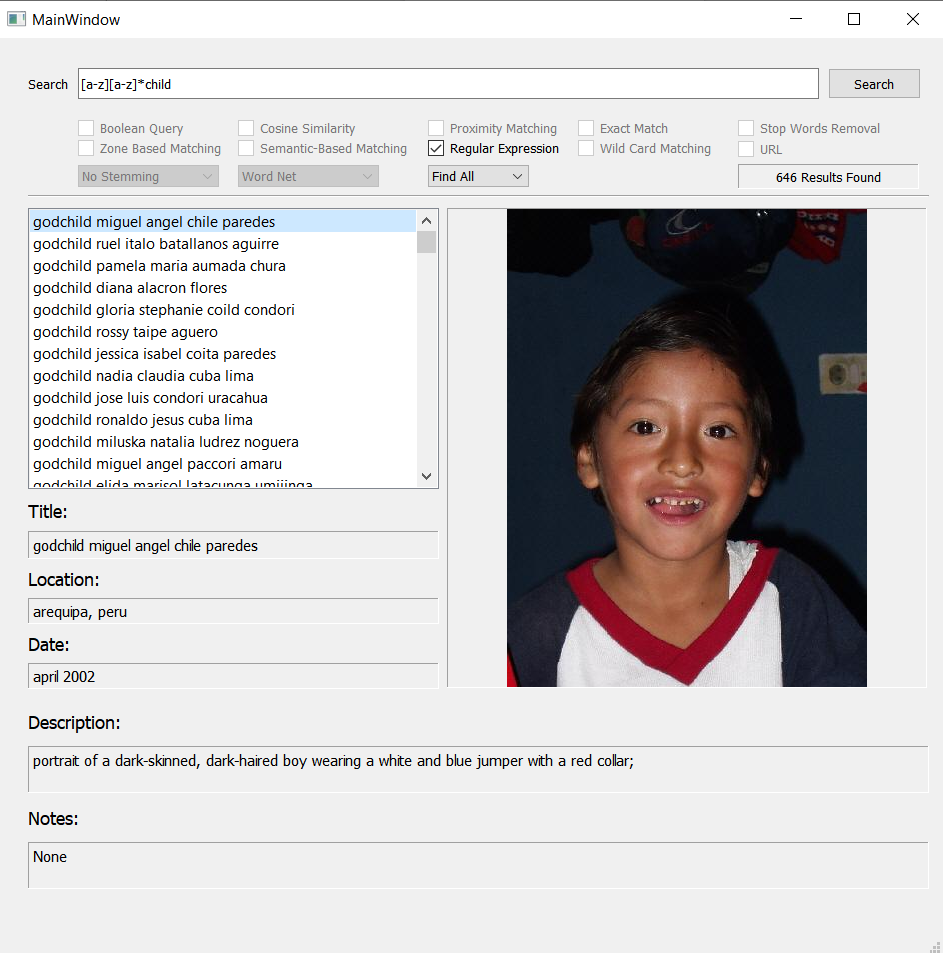
This function performs a proximity query that searches for two terms within k words between them. It takes a query of a form text /k text, it checks for any preceding or trailing /k (/5 text1 /3 text2 /4) and removes them (text1 /3 text2), then it splits the text into text1, number, text 2 and finds the documents that satisfy the query. For example: suppose the query (plaza de armas /5 hotel /2 Spain). The code will search for documents that have plaza, de, and armas, then all documents that have hotel, then the same for Spain, then all documents that have armas /5 hotel, and same for hotel /2 Spain, then find the documents that are matched between them all. The result is then ranked by the maximum space found between the two words (distance between armas and hotel is 5, then 4 ... etc.).

The function can be used with zone-based matching, and stemming only. It can hold multiple /k’s and can handle repeated /k’s with the same values. It can also check for syntax and provides hints on the correct syntax. The file “proximity.py” holds the function code. The following figure represents an illustration of the function output:



1. Regular Expressions:

This function performs the three regular expressions implemented in python (match, search, find all). It takes the regular expression as a query and uses the predefined functions to find the output. This function uses the original data combined without any tokenization or stemming. This function can’t be used with any other functions. The file “reg\_ex.py” represents the code. The following figure represents the output of the function:



1. Exact Match:

The exact match uses regular expressions without any changes on the query for the text in the double quotations except for case folding. The other text is tokenized, stemmed (optional), stop words removal (optional), and matched using positional indexes. Then the results are combined and ranked based on document frequency. The function can’t be used with any other functions.

The file “exact\_match.py” holds the code of this function. The following snippet represents the function:

def exact\_match\_fun(query,flags,zone=None):  
 docs={}  
 tk = NlpTokenizer()  
 x = (re.findall(r'"(.\*?)"', query.lower()))  
 input\_match = x[0]  
 input\_token = tk.tokenizec(input\_match)  
 tokenized = tk.tokenizec(query.lower())  
 tokens = [item for item in tokenized if item not in input\_token]  
 final\_tokens=[]  
 if flags['stop\_words'][0] == 1:   
 Stop words=stopwords.words('english')  
 for t in tokens:  
 if t not in Stop words:  
 final\_tokens.append(t)  
 if flags['stemmer'][0] == 'Porter Stemmer':   
 final\_tokens=searcher().Porter\_Stemmer(final\_tokens)  
 elif flags['stemmer'][0] == 'Snowball Stemmer':   
 final\_tokens=searcher().Snowball\_Stemmer(final\_tokens)  
 elif flags['stemmer'][0] == 'Lancaster Stemmer':  
 final\_tokens=searcher().Lancaster\_Stemmer(final\_tokens)  
 elif flags['stemmer'][0] == 'Customized Stemmer':  
 final\_tokens=searcher().Customized\_Stemmer(final\_tokens)   
 if zone != None:  
 files\_exact =regular\_exp\_matcher().reg\_ex('Find All',input\_match,tags=[zone.upper()])  
 freq\_t, files\_tokens= matcher(final\_tokens,flags,tags=[zone.upper()])  
 docs.update(files\_exact)  
 docs.update(files\_tokens)   
 else:  
 files\_exact =regular\_exp\_matcher().reg\_ex('Find All',input\_match)  
 freq\_t, files\_tokens= matcher(final\_tokens,flags)  
 docs.update(files\_exact)  
 docs.update(files\_tokens)   
 f1=list(files\_exact.keys())  
 freq=Counter(f1) freq=pd.DataFrame(zip(freq.keys(),freq.values()),columns=['docs','freq']).sort\_values(by='freq',ascending=False,ignore\_index=True)  
 freq=freq.append(freq\_t,ignore\_index=True) freq=freq.groupby(by='docs')['freq'].sum().reset\_index().sort\_values(by='freq',ascending=False,ignore\_index=True)  
 return freq, docs

The following figure represents the output:



1. Wild Card Queries:

This function searches for the wild card “\*” within the query tokens, then finds all the documents that includes the expression by replacing the \* with \w\* and use regular expressions to match the wildcard tokens. Other tokens are matched using positional indexes, then the results are combined and ranked based on the document frequency. The function can be used with zone-based matching only. The code snippet for this function is within the “main.py” file, and represented in the following snippet:

elif self.wild\_card\_flag.isChecked() and self.zone\_flag.isChecked()==False:  
 qr=self.query\_tokens  
 wilds=re.findall("\w\*\\*\w\*",qr)   
 text=re.findall("\\*?\w\w\*[.|:|?|'||...|!|-|()]?\w\w\*\\*?",qr)  
 tokens=[item for item in text if item not in wilds]  
 file\_names=[]  
 docs={}  
 freq=pd.DataFrame(data=None,columns=['docs,freq'])  
 for tk in wilds:  
 exp=tk.replace('\*','\w\*')  
 files =regular\_exp\_matcher().reg\_ex('Find All',exp)  
 file\_names.extend(list(files.keys()))  
 docs.update(files)  
 freq\_files=Counter(file\_names)  
 freq=freq.append(pd.DataFrame(zip(freq\_files.keys(),freq\_files.values()),columns=['docs','freq'],ignore\_index=True).sort\_values(by='freq',ascending=False,ignore\_index=True))  
   
 self.stemmer(tokens)   
 f2,freq\_2=matcher(self.stemmed\_query,self.flags)  
 freq=freq.append(freq\_2,ignore\_index=True)  
 freq=freq.groupby(by='docs')['freq'].sum().reset\_index().sort\_values(by='freq',ascending=False,ignore\_index=True)  
 docs.update(f2)   
 self.files=docs  
 self.result\_lister(self.files,freq)

1. Stop Words Removal:

This function uses the NLTK stop words corpus to exclude all tokens that are considered stop words. The function is implemented within the codes of zone-based matching, cosine similarity, semantic-based search and URL queries. The code snippet differs in the terms used in each function, but the lines are the same as the following:

if self.flags['stop\_words'][0]==1:  
 Stop words=stopwords.words('english')  
 self.stemmed\_query=[tk for tk in self.stemmed\_query if tk not in Stop words]

1. URL Queries:

This function uses Beautiful Soup library to parse a given URL in the query, then tokenize the parsed text and match it according to the input rules. The function can be used

With cosine similarity, semantic matching, stop words removal, and stemming.

The following code snippet is used to parse the url:

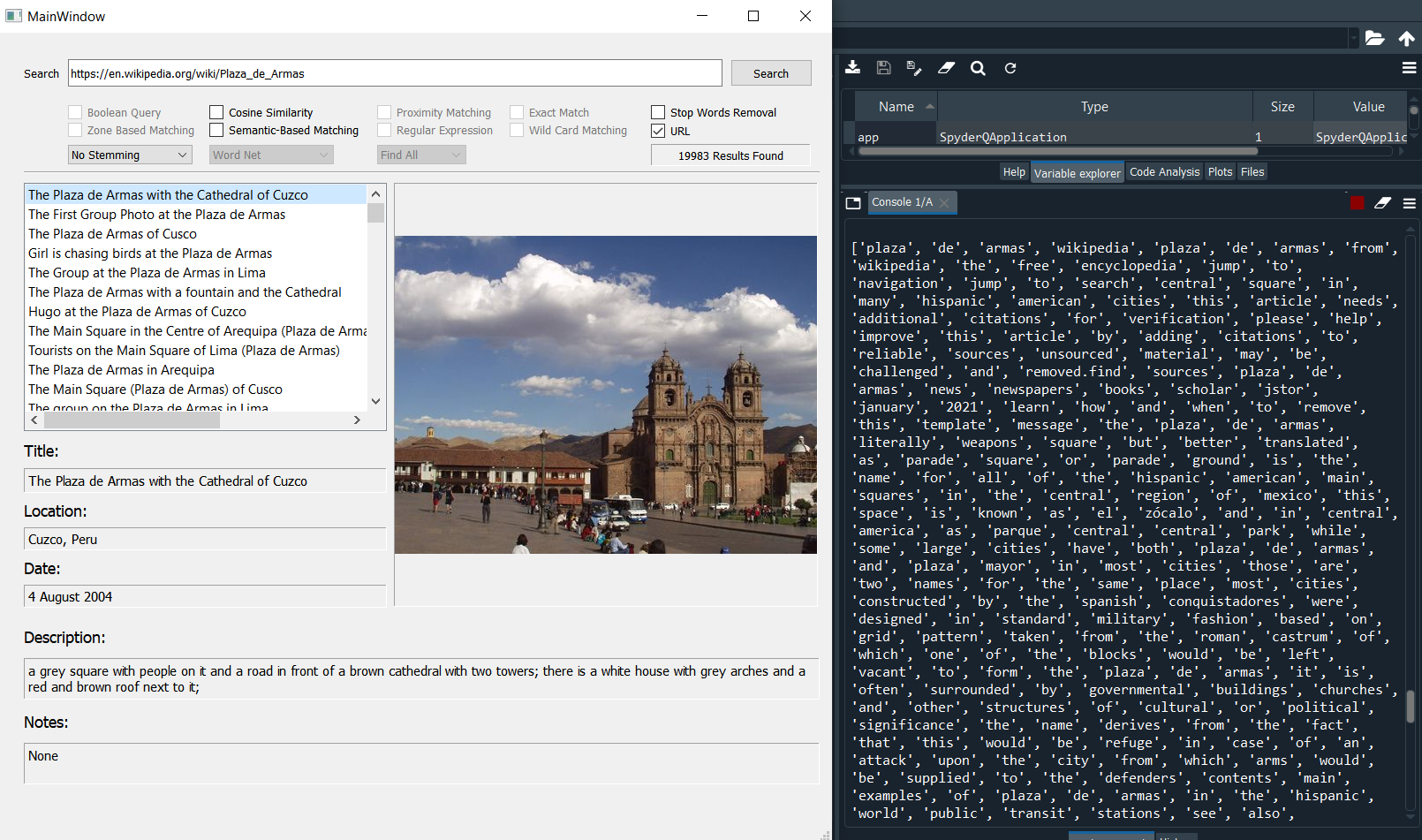
def if\_url(self,text):  
 page = requests.get(text)  
 soup = BeautifulSoup(page.content,'html.parser')  
 return soup.get\_text()

The following snippet is used to tokenize the parsed url:

if self.url\_flag.isChecked():  
 sr=searcher()  
 self.query\_tokens = sr.if\_url(self.query.text())  
 print(NlpTokenizer().tokenizec(self.query\_tokens))

Then, the query is treated as a normal query.

The following figure represents the output:



At the left side, the results of matching site tokens with documents, and on the right side, a sample of the tokens of the URL is shown in the consol.

## **Front-End Development**

All the functions developed in the back-end provide results that must be shown to the user, which is the front-end side. This part is responsible for developing a GUI that is convenient for the user in terms of use, functionality, responsiveness, and result showing. This project utilized PYQT5 library for developing the interface. This library has multiple pros among other GUI libraries:

1. It consists of many useful widgets.
2. Easy to handle layouts.
3. Ability to use qt designer for layout design instead of writing commands
4. Ability to modify styles of screens, add animations, ...etc.
5. Easy to handle signals and time slots.

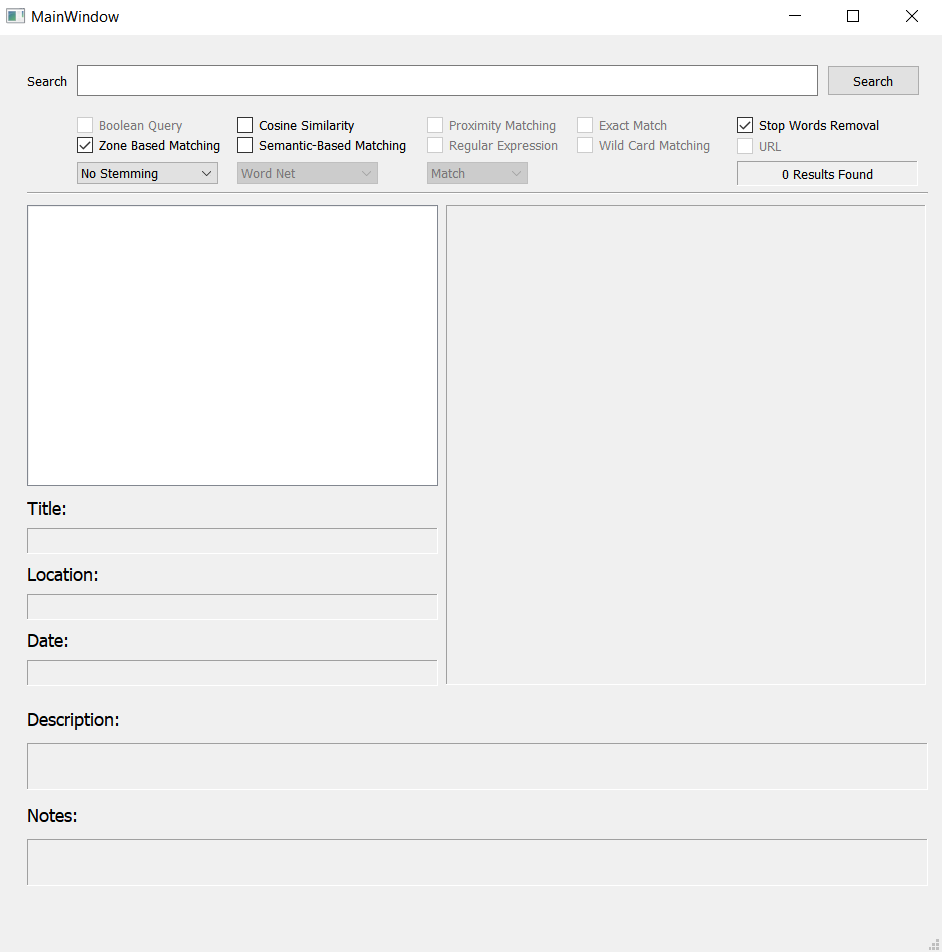
The interface is shown in the following figure:



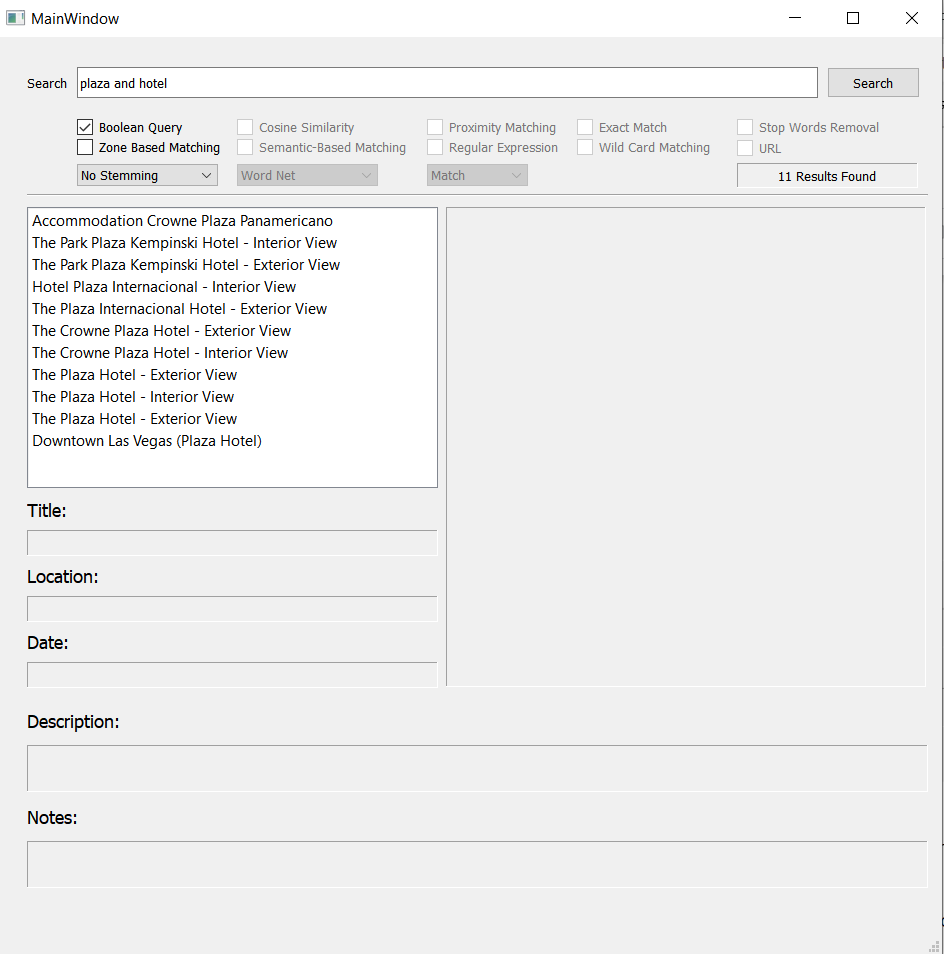
The GUI has many pros:

1. Responsive to user interactions:

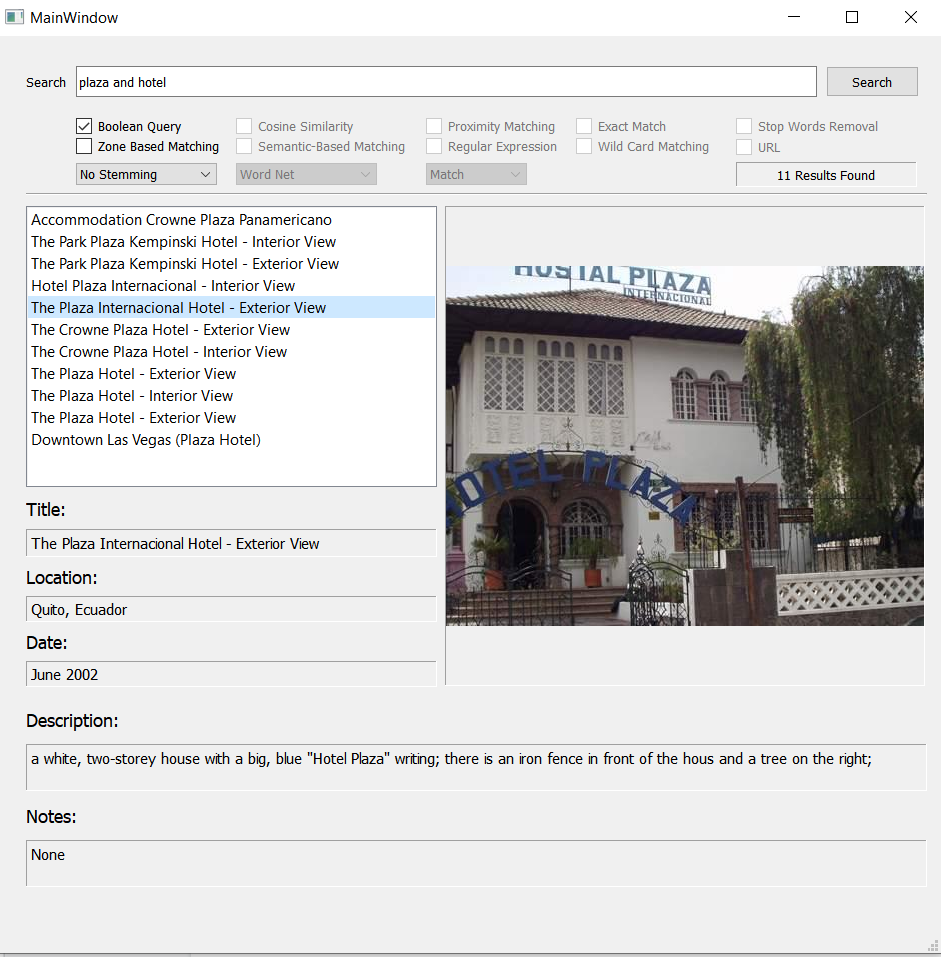
It adapts with user preferences to ensure correct use of the interface. It enables and disables buttons according to eligibility rules to ensure logical combination of the functions. The following figure shows how some functions are disabled when using zone-based matching and stop words removal:



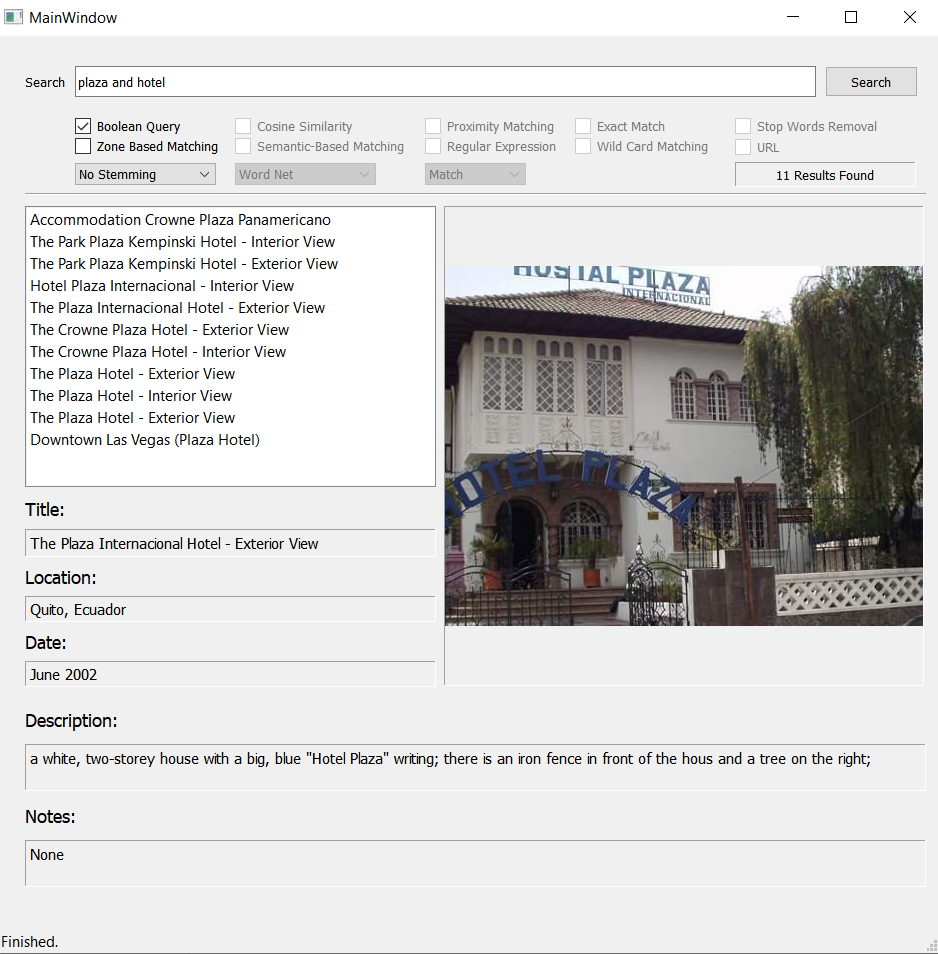
1. It shows the number of results found:



1. Displays the content of each item in the results just by a single click, and the interface retrieves all the related info about the record:



1. Sorts the values from the most related result to the least one (shown in the back-end functions).
2. Provides status when searching is completed in the status bar at the bottom of the interface:



## **Code Documentation**

### **Data Parser**

import pandas as pd  
import xml.etree.ElementTree as et  
import os  
import tqdm  
import json  
from check\_search import searcher  
from tokenizec import NlpTokenizer  
from positional\_indexer import pos\_index  
from tqdm import tqdm  
  
main\_path='D:/Data science Master/NLP/Project/annotations\_complete\_eng/'  
entries = os.listdir(main\_path)  
path=[]  
Cols=['DOCNO','TITLE','DESCRIPTION','NOTES','LOCATION','DATE','IMAGE','THUMBNAIL']  
dataset=pd.DataFrame(data=None, columns=Cols)  
indexed={}  
subdic={}  
for folder in entries:  
 sub=os.listdir(main\_path+folder+'/')  
 for file in sub:  
 filename = main\_path+folder+'/'+file   
 with open(filename, 'r') as content:  
 root = et.parse(content).getroot()  
 path.append(filename)  
 subdic={}  
 for col in Cols:  
 value = root.find(col).text  
 subdic[col]=value  
 indexed[file]=subdic  
  
with open('dataset.json', 'w') as json\_file:  
 json.dump(indexed, json\_file,indent=4)  
  
#--------------------------------------------------------------#  
#this code for stemming dataset and save it offline  
  
tk=NlpTokenizer()  
tokens= tk.datatoken(indexed)  
with open('tokenized.json', 'w') as json\_file:  
 json.dump(tokens, json\_file,indent=4)  
  
sr=searcher()  
ps\_stemmed=sr.Porter\_Stemmer(indexed)  
with open('ps\_stemmed.json', 'w') as json\_file:  
 json.dump(ps\_stemmed, json\_file,indent=4)  
  
sb\_stemmed=sr.Snowball\_Stemmer(indexed)  
with open('sb\_stemmed.json', 'w') as json\_file:  
 json.dump(sb\_stemmed, json\_file,indent=4)  
  
lc\_stemmed=sr.Lancaster\_Stemmer(indexed)  
with open('lc\_stemmed.json', 'w') as json\_file:  
 json.dump(lc\_stemmed, json\_file,indent=4)  
  
custom\_stemmed=sr.Customized\_Stemmer(indexed)  
with open('custom\_stemmed.json', 'w') as json\_file:  
 json.dump(custom\_stemmed, json\_file,indent=4)  
  
#----------------------------------------------------#  
#positional indexers  
  
stem\_no\_stop=['without\_stopword\_tokenized.json', 'without\_stopword\_ps\_stemmed.json', 'without\_stopword\_sb\_stemmed.json',  
 'without\_stopword\_lc\_stemmed.json', 'without\_stopword\_custom\_stemmed.json']  
  
stem\_with\_stop=['tokenized.json', 'ps\_stemmed.json', 'sb\_stemmed.json', 'lc\_stemmed.json', 'custom\_stemmed.json']  
  
pos=pos\_index()  
  
for file in tqdm(stem\_no\_stop):  
 data=json.loads(open("stemmers/"+file).read())  
 file\_map, index=pos.pos\_indexer(data, 0)  
 with open('pos\_index/pos\_'+file, 'w') as json\_file:  
 json.dump(index, json\_file,indent=4)  
  
for file in tqdm(stem\_with\_stop):  
 data=json.loads(open("stemmers/"+file).read())  
 file\_map, index=pos.pos\_indexer(data, 0)  
 with open('pos\_index/pos\_'+file, 'w') as json\_file:  
 json.dump(index, json\_file,indent=4)  
   
with open('pos\_index/file\_map.json', 'w') as json\_file:  
 json.dump(file\_map, json\_file,indent=4)

### **Positional Indexer**

import nltk  
from nltk.corpus import stopwords  
import re  
import os  
import numpy as np  
import json  
from tokenizec import NlpTokenizer  
Stopwords = set(stopwords.words('english'))  
class pos\_index ():  
 def pos\_indexer(self,data, flag):   
 fileno = 0  
 pos\_index = {}   
 file\_map = {}   
 for key in data.keys():  
 for tag in data[key]:  
 if tag in ['TITLE','DESCRIPTION','NOTES','LOCATION']:  
 final\_token\_list = [word.lower() for word in data[key][tag]] ## lower   
 if flag == 1:   
 final\_token\_list = [word for word in final\_token\_list if word not in Stopwords]  
 for pos, term in enumerate(final\_token\_list):   
 if term in pos\_index:   
 pos\_index[term][0] = pos\_index[term][0] + 1  
 if fileno in pos\_index[term][1]:  
 if tag in pos\_index[term][1][fileno]:  
 pos\_index[term][1][fileno][tag].append(pos)   
 else:  
 pos\_index[term][1][fileno].update({tag:[pos]})  
 else:  
 pos\_index[term][1].update({fileno:{tag:[pos]}})   
 else:   
 pos\_index[term] = []   
 pos\_index[term].append(1)   
 pos\_index[term].append({})   
 pos\_index[term][1][fileno]={tag:[pos]}  
 file\_map[fileno] = list(data.keys())[fileno]  
 fileno += 1  
 return file\_map, pos\_index

### **Check Search (Stemming, URL)**

import requests   
import nltk   
from bs4 import BeautifulSoup   
import sys  
import re  
from nltk.stem import PorterStemmer  
from nltk.stem import LancasterStemmer  
from nltk.stem import SnowballStemmer  
from tokenizec import NlpTokenizer  
from custom\_stemmer import customized\_stemmer  
  
from nltk.corpus import wordnet  
  
stem\_words = []  
  
class searcher():   
   
 def check\_text(self,text):  
 regex = re.compile(  
 r'^(?:http|ftp)s?://' # http:// or https://  
 r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+(?:[A-Z]{2,6}\.?|[A-Z0-9-]{2,}\.?)|' #domain...  
 r'localhost|' #localhost...  
 r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})' # ...or ip  
 r'(?::\d+)?' # optional port  
 r'(?:/?|[/?]\S+)$', re.IGNORECASE)  
   
 flag = re.match(regex,text.strip()) is not None # True  
 return flag  
 def if\_url(self,text):  
 page = requests.get(text)  
 soup = BeautifulSoup(page.content,'html.parser')  
 return soup.get\_text()  
 def if\_text(self,text):  
 tokens = self.token\_fun(text)  
 return tokens  
 def token\_fun(self,words):  
 tokens = nltk.word\_tokenize(words)  
 return tokens  
 def Porter\_Stemmer(self,data):  
 porter = PorterStemmer()  
 tk=NlpTokenizer()  
 if type(data) == dict:   
 data\_tokens=tk.datatoken(data)  
 stem\_data={}  
 for file in data\_tokens.keys():  
 stems=[]  
 tags=[]  
 for tag in data\_tokens[file]:  
 stemmed=[]  
 for token in data\_tokens[file][tag]:  
 stemmed.append(porter.stem(token))  
 stems.append(stemmed)  
 tags.append(tag)  
   
 stem\_data[file]=dict(zip(tags,stems))  
 return stem\_data  
 else:  
 stem\_words=[]  
 for w in data:   
 stems = porter.stem(w)   
 stem\_words.append(stems)   
 return stem\_words   
 def Snowball\_Stemmer(self,data):  
 snowball= SnowballStemmer(language='english')  
 tk=NlpTokenizer()  
 if type(data) == dict:   
 data\_tokens=tk.datatoken(data)  
 stem\_data={}  
 for file in data\_tokens.keys():  
 stems=[]  
 tags=[]  
 for tag in data\_tokens[file]:  
 stemmed=[]  
 for token in data\_tokens[file][tag]:  
 stemmed.append(snowball.stem(token))  
 stems.append(stemmed)  
 tags.append(tag)  
   
 stem\_data[file]=dict(zip(tags,stems))  
 return stem\_data  
 else:  
 stem\_words=[]  
 for w in data:   
 stems = snowball.stem(w)   
 stem\_words.append(stems)   
 return stem\_words  
 def Lancaster\_Stemmer(self,data):  
 lancaster=LancasterStemmer()  
 tk=NlpTokenizer()  
 if type(data) == dict:   
 data\_tokens=tk.datatoken(data)  
 stem\_data={}  
 for file in data\_tokens.keys():  
 stems=[]  
 tags=[]  
 for tag in data\_tokens[file]:  
 stemmed=[]  
 for token in data\_tokens[file][tag]:  
 stemmed.append(lancaster.stem(token))  
 stems.append(stemmed)  
 tags.append(tag)  
   
 stem\_data[file]=dict(zip(tags,stems))  
 return stem\_data  
 else:  
 stem\_words=[]  
 for w in data:   
 stems = lancaster.stem(w)   
 stem\_words.append(stems)   
 return stem\_words  
 def Customized\_Stemmer(self, data):  
 my\_stem = customized\_stemmer()  
 stem\_data = my\_stem.my\_stemmer(data)  
 return stem\_data

### **Tokenizer**

import re  
import json  
class NlpTokenizer:  
 def tokenizec(self,text):  
 regx="\w\w\*[.|:|?|'||...|!|-|()]?\w\w\*"  
 tk=re.findall(regx,text)  
 tk=[item.lower() for item in tk]  
 return tk  
 def unitoknizer(self,text):  
 ListOfTokens=self.tokenizec(text)  
 SetTkn=set(ListOfTokens)  
 return SetTkn  
 def datatoken(self,datafile):  
 list1={}  
 for fn in datafile.keys():  
 tags=[]  
 tokens=[]  
 for tag in datafile[fn]:  
 tokens.append(self.tokenizec(str(datafile[fn][tag])))  
 tags.append(tag)  
 list1[fn]=dict(zip(tags,tokens))  
   
 return list1

### **Main Window**

from PyQt5.QtWidgets import QApplication  
from PyQt5 import QtCore, QtGui, QtWidgets  
import sys  
import json  
from reg\_ex import regular\_exp\_matcher  
from check\_search import searcher  
from tokenizec import NlpTokenizer  
import numpy as np  
import pandas as pd  
from zone\_match import zone\_matching  
import re  
from matcher import matcher  
from exact\_match import exact\_match\_fun  
from collections import Counter  
from boolean\_query import boolean  
from nltk.corpus import stopwords  
from semantic import semantic  
from cosine import cosine\_sim  
from proximity import prox  
import time  
data = json.loads(open("dataset.json").read())  
class Ui\_MainWindow(object):  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName("MainWindow")  
 MainWindow.setEnabled(True)  
 MainWindow.resize(945, 917)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Minimum)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())  
 MainWindow.setSizePolicy(sizePolicy)  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName("centralwidget")  
 self.label = QtWidgets.QLabel(self.centralwidget)  
 self.label.setGeometry(QtCore.QRect(30, 30, 51, 31))  
 self.label.setObjectName("label")  
 self.reg\_ex\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.reg\_ex\_flag.setGeometry(QtCore.QRect(430, 100, 141, 20))  
 self.reg\_ex\_flag.setObjectName("reg\_ex\_flag")  
 self.bool\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.bool\_flag.setGeometry(QtCore.QRect(80, 80, 141, 20))  
 self.bool\_flag.setObjectName("bool\_flag")  
 self.exct\_match\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.exct\_match\_flag.setGeometry(QtCore.QRect(580, 80, 141, 20))  
 self.exct\_match\_flag.setObjectName("exct\_match\_flag")  
 self.cos\_sim\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.cos\_sim\_flag.setGeometry(QtCore.QRect(240, 80, 141, 20))  
 self.cos\_sim\_flag.setObjectName("cos\_sim\_flag")  
 self.stemmers\_list = QtWidgets.QComboBox(self.centralwidget)  
 self.stemmers\_list.setGeometry(QtCore.QRect(80, 127, 141, 22))  
 self.stemmers\_list.setObjectName("stemmers\_list")  
 self.stemmers\_list.addItem("")  
 self.stemmers\_list.addItem("")  
 self.stemmers\_list.addItem("")  
 self.stemmers\_list.addItem("")  
 self.stemmers\_list.addItem("")  
 self.prox\_mtch\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.prox\_mtch\_flag.setGeometry(QtCore.QRect(430, 80, 141, 20))  
 self.prox\_mtch\_flag.setObjectName("prox\_mtch\_flag")  
 self.wild\_card\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.wild\_card\_flag.setGeometry(QtCore.QRect(580, 100, 141, 20))  
 self.wild\_card\_flag.setObjectName("wild\_card\_flag")  
 self.zone\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.zone\_flag.setGeometry(QtCore.QRect(80, 100, 151, 20))  
 self.zone\_flag.setObjectName("zone\_flag")  
 self.sem\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.sem\_flag.setGeometry(QtCore.QRect(240, 100, 181, 20))  
 self.sem\_flag.setObjectName("sem\_flag")  
 self.sem\_ont = QtWidgets.QComboBox(self.centralwidget)  
 self.sem\_ont.setGeometry(QtCore.QRect(240, 127, 141, 22))  
 self.sem\_ont.setObjectName("sem\_ont")  
 self.sem\_ont.addItem("")  
 self.sem\_ont.addItem("")  
 self.sem\_ont.addItem("")  
 self.stp\_wrd\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.stp\_wrd\_flag.setGeometry(QtCore.QRect(740, 80, 181, 20))  
 self.stp\_wrd\_flag.setObjectName("stp\_wrd\_flag")  
 self.reg\_ex\_list = QtWidgets.QComboBox(self.centralwidget)  
 self.reg\_ex\_list.setGeometry(QtCore.QRect(430, 127, 101, 22))  
 self.reg\_ex\_list.setObjectName("reg\_ex\_list")  
 self.reg\_ex\_list.addItem("")  
 self.reg\_ex\_list.addItem("")  
 self.reg\_ex\_list.addItem("")  
 self.search = QtWidgets.QPushButton(self.centralwidget)  
 self.search.setGeometry(QtCore.QRect(830, 30, 93, 31))  
 self.search.setObjectName("search")  
 self.line = QtWidgets.QFrame(self.centralwidget)  
 self.line.setGeometry(QtCore.QRect(30, 150, 901, 16))  
 self.line.setFrameShape(QtWidgets.QFrame.HLine)  
 self.line.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line.setObjectName("line")  
 self.query = QtWidgets.QLineEdit(self.centralwidget)  
 self.query.setGeometry(QtCore.QRect(80, 30, 741, 31))  
 self.query.setObjectName("query")  
 self.gridLayoutWidget\_2 = QtWidgets.QWidget(self.centralwidget)  
 self.gridLayoutWidget\_2.setGeometry(QtCore.QRect(30, 460, 411, 191))  
 self.gridLayoutWidget\_2.setObjectName("gridLayoutWidget\_2")  
 self.gridLayout\_6 = QtWidgets.QGridLayout(self.gridLayoutWidget\_2)  
 self.gridLayout\_6.setContentsMargins(0, 0, 0, 0)  
 self.gridLayout\_6.setObjectName("gridLayout\_6")  
 self.verticalLayout\_3 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_3.setObjectName("verticalLayout\_3")  
 self.label\_3 = QtWidgets.QLabel(self.gridLayoutWidget\_2)  
 self.label\_3.setMinimumSize(QtCore.QSize(0, 0))  
 font = QtGui.QFont()  
 font.setPointSize(11)  
 font.setBold(False)  
 font.setWeight(50)  
 self.label\_3.setFont(font)  
 self.label\_3.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.label\_3.setFrameShape(QtWidgets.QFrame.NoFrame)  
 self.label\_3.setFrameShadow(QtWidgets.QFrame.Plain)  
 self.label\_3.setMidLineWidth(0)  
 self.label\_3.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)  
 self.label\_3.setObjectName("label\_3")  
 self.verticalLayout\_3.addWidget(self.label\_3)  
 self.title = QtWidgets.QLabel(self.gridLayoutWidget\_2)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.title.sizePolicy().hasHeightForWidth())  
 self.title.setSizePolicy(sizePolicy)  
 self.title.setMinimumSize(QtCore.QSize(0, 0))  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 font.setBold(False)  
 font.setWeight(50)  
 self.title.setFont(font)  
 self.title.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.title.setFrameShape(QtWidgets.QFrame.Panel)  
 self.title.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.title.setMidLineWidth(0)  
 self.title.setText("")  
 self.title.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)  
 self.title.setWordWrap(True)  
 self.title.setTextInteractionFlags(QtCore.Qt.LinksAccessibleByMouse|QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)  
 self.title.setObjectName("title")  
 self.verticalLayout\_3.addWidget(self.title)  
 self.label\_10 = QtWidgets.QLabel(self.gridLayoutWidget\_2)  
 self.label\_10.setMinimumSize(QtCore.QSize(0, 0))  
 font = QtGui.QFont()  
 font.setPointSize(11)  
 font.setBold(False)  
 font.setWeight(50)  
 self.label\_10.setFont(font)  
 self.label\_10.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.label\_10.setFrameShape(QtWidgets.QFrame.NoFrame)  
 self.label\_10.setFrameShadow(QtWidgets.QFrame.Plain)  
 self.label\_10.setMidLineWidth(0)  
 self.label\_10.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)  
 self.label\_10.setObjectName("label\_10")  
 self.verticalLayout\_3.addWidget(self.label\_10)  
 self.location = QtWidgets.QLabel(self.gridLayoutWidget\_2)  
 self.location.setMinimumSize(QtCore.QSize(0, 0))  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 font.setBold(False)  
 font.setWeight(50)  
 self.location.setFont(font)  
 self.location.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.location.setFrameShape(QtWidgets.QFrame.Panel)  
 self.location.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.location.setMidLineWidth(0)  
 self.location.setText("")  
 self.location.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)  
 self.location.setWordWrap(True)  
 self.location.setTextInteractionFlags(QtCore.Qt.LinksAccessibleByMouse|QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)  
 self.location.setObjectName("location")  
 self.verticalLayout\_3.addWidget(self.location)  
 self.label\_12 = QtWidgets.QLabel(self.gridLayoutWidget\_2)  
 self.label\_12.setMinimumSize(QtCore.QSize(0, 0))  
 font = QtGui.QFont()  
 font.setPointSize(11)  
 font.setBold(False)  
 font.setWeight(50)  
 self.label\_12.setFont(font)  
 self.label\_12.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.label\_12.setFrameShape(QtWidgets.QFrame.NoFrame)  
 self.label\_12.setFrameShadow(QtWidgets.QFrame.Plain)  
 self.label\_12.setMidLineWidth(0)  
 self.label\_12.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)  
 self.label\_12.setObjectName("label\_12")  
 self.verticalLayout\_3.addWidget(self.label\_12)  
 self.date = QtWidgets.QLabel(self.gridLayoutWidget\_2)  
 self.date.setMinimumSize(QtCore.QSize(0, 0))  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 font.setBold(False)  
 font.setWeight(50)  
 self.date.setFont(font)  
 self.date.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.date.setFrameShape(QtWidgets.QFrame.Panel)  
 self.date.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.date.setMidLineWidth(0)  
 self.date.setText("")  
 self.date.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)  
 self.date.setWordWrap(True)  
 self.date.setTextInteractionFlags(QtCore.Qt.LinksAccessibleByMouse|QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)  
 self.date.setObjectName("date")  
 self.verticalLayout\_3.addWidget(self.date)  
 self.gridLayout\_6.addLayout(self.verticalLayout\_3, 0, 0, 1, 1)  
 self.gridLayoutWidget = QtWidgets.QWidget(self.centralwidget)  
 self.gridLayoutWidget.setGeometry(QtCore.QRect(30, 170, 411, 281))  
 self.gridLayoutWidget.setObjectName("gridLayoutWidget")  
 self.gridLayout\_5 = QtWidgets.QGridLayout(self.gridLayoutWidget)  
 self.gridLayout\_5.setContentsMargins(0, 0, 0, 0)  
 self.gridLayout\_5.setObjectName("gridLayout\_5")  
 self.listWidget = QtWidgets.QListWidget(self.gridLayoutWidget)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.listWidget.sizePolicy().hasHeightForWidth())  
 self.listWidget.setSizePolicy(sizePolicy)  
 self.listWidget.setEditTriggers(QtWidgets.QAbstractItemView.EditKeyPressed)  
 self.listWidget.setObjectName("listWidget")  
 self.gridLayout\_5.addWidget(self.listWidget, 0, 0, 1, 1)  
 self.layoutWidget = QtWidgets.QWidget(self.centralwidget)  
 self.layoutWidget.setGeometry(QtCore.QRect(30, 660, 901, 191))  
 self.layoutWidget.setObjectName("layoutWidget")  
 self.verticalLayout\_4 = QtWidgets.QVBoxLayout(self.layoutWidget)  
 self.verticalLayout\_4.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstraint)  
 self.verticalLayout\_4.setContentsMargins(0, 0, 0, 0)  
 self.verticalLayout\_4.setSpacing(1)  
 self.verticalLayout\_4.setObjectName("verticalLayout\_4")  
 self.label\_6 = QtWidgets.QLabel(self.layoutWidget)  
 font = QtGui.QFont()  
 font.setPointSize(11)  
 font.setBold(False)  
 font.setWeight(50)  
 self.label\_6.setFont(font)  
 self.label\_6.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.label\_6.setFrameShape(QtWidgets.QFrame.NoFrame)  
 self.label\_6.setFrameShadow(QtWidgets.QFrame.Plain)  
 self.label\_6.setMidLineWidth(0)  
 self.label\_6.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)  
 self.label\_6.setObjectName("label\_6")  
 self.verticalLayout\_4.addWidget(self.label\_6)  
 self.description = QtWidgets.QLabel(self.layoutWidget)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.description.sizePolicy().hasHeightForWidth())  
 self.description.setSizePolicy(sizePolicy)  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 font.setBold(False)  
 font.setWeight(50)  
 self.description.setFont(font)  
 self.description.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.description.setFrameShape(QtWidgets.QFrame.Panel)  
 self.description.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.description.setMidLineWidth(0)  
 self.description.setText("")  
 self.description.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)  
 self.description.setWordWrap(True)  
 self.description.setTextInteractionFlags(QtCore.Qt.LinksAccessibleByMouse|QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)  
 self.description.setObjectName("description")  
 self.verticalLayout\_4.addWidget(self.description)  
 self.label\_7 = QtWidgets.QLabel(self.layoutWidget)  
 font = QtGui.QFont()  
 font.setPointSize(11)  
 font.setBold(False)  
 font.setWeight(50)  
 self.label\_7.setFont(font)  
 self.label\_7.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.label\_7.setFrameShape(QtWidgets.QFrame.NoFrame)  
 self.label\_7.setFrameShadow(QtWidgets.QFrame.Plain)  
 self.label\_7.setMidLineWidth(0)  
 self.label\_7.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignVCenter)  
 self.label\_7.setObjectName("label\_7")  
 self.verticalLayout\_4.addWidget(self.label\_7)  
 self.notes = QtWidgets.QLabel(self.layoutWidget)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.notes.sizePolicy().hasHeightForWidth())  
 self.notes.setSizePolicy(sizePolicy)  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 font.setBold(False)  
 font.setWeight(50)  
 self.notes.setFont(font)  
 self.notes.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.notes.setFrameShape(QtWidgets.QFrame.Panel)  
 self.notes.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.notes.setMidLineWidth(0)  
 self.notes.setText("")  
 self.notes.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)  
 self.notes.setWordWrap(True)  
 self.notes.setTextInteractionFlags(QtCore.Qt.LinksAccessibleByMouse|QtCore.Qt.TextSelectableByKeyboard|QtCore.Qt.TextSelectableByMouse)  
 self.notes.setObjectName("notes")  
 self.verticalLayout\_4.addWidget(self.notes)  
 self.label\_2 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_2.setGeometry(QtCore.QRect(449, 170, 480, 480))  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.label\_2.sizePolicy().hasHeightForWidth())  
 self.label\_2.setSizePolicy(sizePolicy)  
 self.label\_2.setMinimumSize(QtCore.QSize(210, 210))  
 self.label\_2.setFrameShape(QtWidgets.QFrame.Panel)  
 self.label\_2.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.label\_2.setText("")  
 self.label\_2.setScaledContents(False)  
 self.label\_2.setAlignment(QtCore.Qt.AlignCenter)  
 self.label\_2.setObjectName("label\_2")  
 self.results\_lable = QtWidgets.QLabel(self.centralwidget)  
 self.results\_lable.setGeometry(QtCore.QRect(740, 126, 181, 25))  
 self.results\_lable.setFrameShape(QtWidgets.QFrame.Panel)  
 self.results\_lable.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.results\_lable.setAlignment(QtCore.Qt.AlignCenter)  
 self.results\_lable.setObjectName("results\_lable")  
 self.url\_flag = QtWidgets.QCheckBox(self.centralwidget)  
 self.url\_flag.setGeometry(QtCore.QRect(740, 100, 83, 22))  
 self.url\_flag.setObjectName("url\_flag")  
 MainWindow.setCentralWidget(self.centralwidget)  
 self.menubar = QtWidgets.QMenuBar(MainWindow)  
 self.menubar.setGeometry(QtCore.QRect(0, 0, 945, 25))  
 self.menubar.setObjectName("menubar")  
 MainWindow.setMenuBar(self.menubar)  
 self.statusbar = QtWidgets.QStatusBar(MainWindow)  
 self.statusbar.setObjectName("statusbar")  
 MainWindow.setStatusBar(self.statusbar)   
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
 self.reg\_ex\_list.setEnabled(False)   
 self.reg\_ex\_flag.clicked.connect(self.flag\_check)  
 self.sem\_flag.clicked.connect(self.flag\_check)  
 self.exct\_match\_flag.clicked.connect(self.flag\_check)  
 self.bool\_flag.clicked.connect(self.flag\_check)  
 self.prox\_mtch\_flag.clicked.connect(self.flag\_check)  
 self.zone\_flag.clicked.connect(self.flag\_check)  
 self.sem\_ont.setEnabled(False)  
 self.stp\_wrd\_flag.clicked.connect(self.flag\_check)  
 self.search.clicked.connect(self.action)  
 self.sem\_ont.currentIndexChanged.connect(self.flag\_check)  
 self.listWidget.itemClicked.connect(self.get\_info)  
 self.wild\_card\_flag.clicked.connect(self.flag\_check)  
 self.cos\_sim\_flag.clicked.connect(self.flag\_check)  
 self.url\_flag.clicked.connect(self.flag\_check)  
 self.data=data #assign default dataset is the dataset without any stemming   
 self.stemmed\_data={} # assign empty dictionary to stemmed data  
 self.stemmed\_query=[] # assign empty list to stemmed query  
 self.flags=pd.DataFrame(data=np.zeros([1,12]),columns=['boolean','zone\_based','cosine','wildcard','stemming','stemmer','proximity','semantic','ontology','exact\_match','regex','stop\_words'])  
 def get\_info(self,item):  
 index=self.listWidget.currentRow()  
 if len(self.freq):  
 target\_doc = self.freq['docs'][index]  
 else:  
 target\_doc = list(self.files.keys())[index]  
 self.title.setText(self.files[target\_doc]['TITLE'])  
 self.location.setText(self.files[target\_doc]['LOCATION'])  
 self.date.setText(self.files[target\_doc]['DATE'])  
 self.description.setText(self.files[target\_doc]['DESCRIPTION'])  
 self.notes.setText(str(self.files[target\_doc]['NOTES']))  
 self.label\_2.setPixmap(QtGui.QPixmap(self.files[target\_doc]['IMAGE']))  
 def stemmer(self,query\_tokens):  
 self.flags['stemmer']=self.stemmers\_list.currentText()  
 if self.stemmers\_list.currentText()=='Porter Stemmer':  
 if self.flags['stop\_words'][0] == 0:  
 self.stemmed\_data=json.loads(open("stemmers/ps\_stemmed.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_ps\_stemmed.json").read())  
 else:  
 self.stemmed\_data=json.loads(open("stemmers/without\_stopword\_ps\_stemmed.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_without\_stopword\_ps\_stemmed.json").read())  
 sr=searcher()  
 self.stemmed\_query=sr.Porter\_Stemmer(query\_tokens)  
   
 elif self.stemmers\_list.currentText()=='Snowball Stemmer':  
 if self.flags['stop\_words'][0] == 0:  
 self.stemmed\_data=json.loads(open("stemmers/sb\_stemmed.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_sb\_stemmed.json").read())  
 else:  
 self.stemmed\_data=json.loads(open("stemmers/without\_stopword\_sb\_stemmed.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_without\_stopword\_sb\_stemmed.json").read())  
 sr=searcher()  
 self.stemmed\_query=sr.Snowball\_Stemmer(query\_tokens)  
   
 elif self.stemmers\_list.currentText()=='Lancaster Stemmer':  
 if self.flags['stop\_words'][0] == 0:  
 self.stemmed\_data=json.loads(open("stemmers/lc\_stemmed.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_lc\_stemmed.json").read())  
 else:  
 self.stemmed\_data=json.loads(open("stemmers/without\_stopword\_lc\_stemmed.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_without\_stopword\_lc\_stemmed.json").read())  
 sr=searcher()  
 self.stemmed\_query=sr.Lancaster\_Stemmer(query\_tokens)  
   
 elif self.stemmers\_list.currentText()=='Customized Stemmer':  
 if self.flags['stop\_words'][0] == 0:  
 self.stemmed\_data=json.loads(open("stemmers/custom\_stemmed.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_custom\_stemmed.json").read())  
 else:  
 self.stemmed\_data=json.loads(open("stemmers/without\_stopword\_custom\_stemmed.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_without\_stopword\_custom\_stemmed.json").read())  
 sr=searcher()  
 self.stemmed\_query=sr.Customized\_Stemmer(query\_tokens)  
 else:  
 if self.flags['stop\_words'][0] == 0:  
 self.stemmed\_data=json.loads(open("stemmers/tokenized.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_tokenized.json").read())  
 else:  
 self.stemmed\_data=json.loads(open("stemmers/without\_stopword\_tokenized.json").read())  
 self.pos\_index=json.loads(open("pos\_index/pos\_without\_stopword\_tokenized.json").read())  
 self.stemmed\_query=query\_tokens  
   
 return self.stemmed\_query  
  
 def flag\_update(self):  
 'boolean','zone\_based','cosine','wildcard','stemming','stemmer','proximity','semantic','ontology','exact\_match','regex','stop\_words'  
 update=[self.bool\_flag.isEnabled() and self.bool\_flag.isChecked(),  
 self.zone\_flag.isEnabled() and self.zone\_flag.isChecked(),  
 self.cos\_sim\_flag.isEnabled() and self.cos\_sim\_flag.isChecked(),  
 self.wild\_card\_flag.isEnabled() and self.wild\_card\_flag.isChecked(),  
 self.stemmers\_list.isEnabled(),  
 self.stemmers\_list.currentText(),  
 self.prox\_mtch\_flag.isEnabled() and self.prox\_mtch\_flag.isChecked(),  
 self.sem\_flag.isEnabled() and self.sem\_flag.isChecked(),  
 self.sem\_ont.currentText(),  
 self.exct\_match\_flag.isEnabled() and self.exct\_match\_flag.isChecked(),  
 self.reg\_ex\_flag.isEnabled() and self.reg\_ex\_flag.isChecked(),  
 self.stp\_wrd\_flag.isEnabled() and self.stp\_wrd\_flag.isChecked(),  
 ]  
 self.flags.loc[0]=update  
 def flag\_check(self):  
 self.bool\_flag.setEnabled(True)  
 self.zone\_flag.setEnabled(True)  
 self.wild\_card\_flag.setEnabled(True)  
 self.cos\_sim\_flag.setEnabled(True)  
 self.exct\_match\_flag.setEnabled(True)  
 self.reg\_ex\_flag.setEnabled(True)  
 self.prox\_mtch\_flag.setEnabled(True)  
 self.stp\_wrd\_flag.setEnabled(True)  
 self.sem\_flag.setEnabled(True)  
 self.stemmers\_list.setEnabled(True)  
 self.reg\_ex\_list.setEnabled(False)  
 self.sem\_ont.setEnabled(False)  
 self.url\_flag.setEnabled(True)  
   
 if self.bool\_flag.isChecked():  
 self.wild\_card\_flag.setEnabled(False)  
 self.reg\_ex\_flag.setEnabled(False)  
 self.prox\_mtch\_flag.setEnabled(False)  
 self.stp\_wrd\_flag.setEnabled(False)  
 self.cos\_sim\_flag.setEnabled(False)  
 self.reg\_ex\_list.setEnabled(False)  
 self.sem\_flag.setEnabled(False)  
 self.exct\_match\_flag.setEnabled(False)  
 self.url\_flag.setEnabled(False)  
 self.flag\_update()  
 self.flags['boolean'][0]=True  
   
 if self.reg\_ex\_flag.isChecked():  
 self.bool\_flag.setEnabled(False)  
 self.wild\_card\_flag.setEnabled(False)  
 self.exct\_match\_flag.setEnabled(False)  
 self.prox\_mtch\_flag.setEnabled(False)  
 self.stp\_wrd\_flag.setEnabled(False)  
 self.zone\_flag.setEnabled(False)  
 self.cos\_sim\_flag.setEnabled(False)  
 self.sem\_flag.setEnabled(False)  
 self.reg\_ex\_list.setEnabled(True)  
 self.stemmers\_list.setEnabled(False)  
 self.url\_flag.setEnabled(False)  
 self.flag\_update()  
 self.flags['regex'][0]=True  
  
   
 if self.exct\_match\_flag.isChecked():  
 self.bool\_flag.setEnabled(False)  
 self.zone\_flag.setEnabled(False)  
 self.wild\_card\_flag.setEnabled(False)  
 self.cos\_sim\_flag.setEnabled(False)  
 self.reg\_ex\_flag.setEnabled(False)  
 self.prox\_mtch\_flag.setEnabled(False)  
 self.stp\_wrd\_flag.setEnabled(False)  
 self.sem\_flag.setEnabled(False)  
 self.stemmers\_list.setEnabled(False)  
 self.reg\_ex\_list.setEnabled(False)  
 self.sem\_ont.setEnabled(False)  
 self.url\_flag.setEnabled(False)  
 self.flag\_update()  
 self.flags['exact\_match'][0]=True  
 if self.sem\_flag.isChecked():  
 self.sem\_ont.setEnabled(True)  
 self.bool\_flag.setEnabled(False)  
 self.exct\_match\_flag.setEnabled(False)  
 self.cos\_sim\_flag.setEnabled(False)  
 self.reg\_ex\_flag.setEnabled(False)  
 self.prox\_mtch\_flag.setEnabled(False)  
 self.stp\_wrd\_flag.setEnabled(True)  
 self.sem\_flag.setEnabled(True)  
 self.stemmers\_list.setEnabled(True)  
 self.reg\_ex\_list.setEnabled(False)  
 self.flag\_update()  
 self.flags['semantic'][0]=True   
 if self.prox\_mtch\_flag.isChecked():  
 self.bool\_flag.setEnabled(False)  
 self.cos\_sim\_flag.setEnabled(False)  
 self.exct\_match\_flag.setEnabled(False)  
 self.stp\_wrd\_flag.setEnabled(False)  
 self.reg\_ex\_flag.setEnabled(False)  
 self.sem\_flag.setEnabled(False)  
 self.reg\_ex\_list.setEnabled(False)  
 self.wild\_card\_flag.setEnabled(False)  
 self.url\_flag.setEnabled(False)  
 self.flag\_update()  
 self.flags['proximity'][0]=True  
 if self.wild\_card\_flag.isChecked():  
 self.bool\_flag.setEnabled(False)  
 self.cos\_sim\_flag.setEnabled(False)  
 self.exct\_match\_flag.setEnabled(False)  
 self.reg\_ex\_flag.setEnabled(False)  
 self.sem\_flag.setEnabled(False)  
 self.prox\_mtch\_flag.setEnabled(False)  
 self.reg\_ex\_list.setEnabled(False)  
 self.stp\_wrd\_flag.setEnabled(False)  
 self.stemmers\_list.setEnabled(False)  
 self.url\_flag.setEnabled(False)  
 self.flag\_update()  
 self.flags['wildcard'][0]=True  
 if self.stp\_wrd\_flag.isChecked():  
 self.reg\_ex\_flag.setEnabled(False)  
 self.exct\_match\_flag.setEnabled(False)  
 self.prox\_mtch\_flag.setEnabled(False)  
 self.wild\_card\_flag.setEnabled(False)  
 self.bool\_flag.setEnabled(False)  
 self.flag\_update()  
 self.flags['stop\_words'][0]=True  
 if self.zone\_flag.isChecked():  
 self.reg\_ex\_flag.setEnabled(False)  
 self.url\_flag.setEnabled(False)  
 self.flag\_update()  
 self.flags['zone\_based'][0]=True   
 if self.cos\_sim\_flag.isChecked():  
 self.bool\_flag.setEnabled(False)  
 self.wild\_card\_flag.setEnabled(False)  
 self.exct\_match\_flag.setEnabled(False)  
 self.reg\_ex\_flag.setEnabled(False)  
 self.prox\_mtch\_flag.setEnabled(False)  
 self.sem\_flag.setEnabled(False)  
 self.stp\_wrd\_flag.setEnabled(True)  
 self.stemmers\_list.setEnabled(True)  
 self.reg\_ex\_list.setEnabled(False)  
 self.flag\_update()  
 self.flags['cosine'][0]=True  
 if self.url\_flag.isChecked():  
 self.bool\_flag.setEnabled(False)  
 self.zone\_flag.setEnabled(False)  
 self.wild\_card\_flag.setEnabled(False)  
 self.exct\_match\_flag.setEnabled(False)  
 self.reg\_ex\_flag.setEnabled(False)  
 self.prox\_mtch\_flag.setEnabled(False)  
 self.reg\_ex\_list.setEnabled(False)  
 self.flag\_update()  
 def result\_lister(self,files,freq=''):  
 self.freq=freq  
 self.results\_lable.setText(str(len(files))+' Results Found')  
 self.listWidget.clear()  
 titles=[]  
 if len(freq) > 0 :  
 for file in freq['docs']:  
 titles.append(files[file]['TITLE'])  
 else:  
 for file in files:  
 titles.append(files[file]['TITLE'])  
 if len(titles)==0:   
 self.listWidget.addItem('No Results')  
 self.title.clear()  
 self.location.clear()  
 self.date.clear()  
 self.description.clear()  
 self.notes.clear()  
 self.label\_2.clear()  
 else:  
 self.listWidget.addItems(titles)  
 QApplication.restoreOverrideCursor()  
 self.statusbar.showMessage('Finished.',10000)  
 def action(self):  
 tk=NlpTokenizer()  
 QApplication.setOverrideCursor(QtCore.Qt.WaitCursor)   
 if self.url\_flag.isChecked():  
 sr=searcher()  
 self.query\_tokens = sr.if\_url(self.query.text())  
 print(NlpTokenizer().tokenizec(self.query\_tokens))  
 else:   
 self.query\_tokens = self.query.text()  
 if self.zone\_flag.isChecked() and self.zone\_flag.isEnabled():  
 self.flags['zone\_based']=True  
 self.files,query\_tokens,freq=zone\_matching(self.query\_tokens,self.flags)  
 if len(query\_tokens.keys()):  
 stemmed\_queries={}  
 files2={}  
 freq\_2=pd.DataFrame(data=None, columns=['docs','freq'])  
 for key in query\_tokens.keys():  
 stemmed\_queries[key]=self.stemmer(query\_tokens[key])   
 fr\_2,f2= matcher(stemmed\_queries[key],self.flags,tags=[key])  
 files2.update(f2)  
 freq\_2=freq\_2.append(fr\_2,ignore\_index=True)  
 self.files.update(files2)  
 freq=freq.append(freq\_2)  
 freq=freq.groupby(by='docs')['freq'].sum().reset\_index().sort\_values(by='freq',ascending=False,ignore\_index=True)   
 self.result\_lister(self.files,freq)  
 elif self.exct\_match\_flag.isChecked() and self.zone\_flag.isEnabled()==False:  
 freq, self.files=exact\_match\_fun(self.query\_tokens,self.flags)  
 self.result\_lister(self.files,freq)  
 elif self.reg\_ex\_flag.isChecked() and self.zone\_flag.isEnabled()==False:  
 exp=self.query\_tokens  
 func=self.reg\_ex\_list.currentText()  
 self.files =regular\_exp\_matcher().reg\_ex(func,exp)  
 self.result\_lister(self.files)   
 elif self.wild\_card\_flag.isChecked() and self.zone\_flag.isChecked()==False:  
 qr=self.query\_tokens  
 wilds=re.findall("\w\*\\*\w\*",qr)   
 text=re.findall("\\*?\w\w\*[.|:|?|'||...|!|-|()]?\w\w\*\\*?",qr)  
 tokens=[item for item in text if item not in wilds]  
 file\_names=[]  
 docs={}  
 freq=pd.DataFrame(data=None,columns=['docs,freq'])  
 for tk in wilds:  
 exp=tk.replace('\*','\w\*')  
 files =regular\_exp\_matcher().reg\_ex('Find All',exp)  
 file\_names.extend(list(files.keys()))  
 docs.update(files)  
 freq\_files=Counter(file\_names)  
 freq=freq.append(pd.DataFrame(zip(freq\_files.keys(),freq\_files.values()),columns=['docs','freq'],ignore\_index=True).sort\_values(by='freq',ascending=False,ignore\_index=True))  
 self.stemmer(tokens)   
 f2,freq\_2=matcher(self.stemmed\_query,self.flags)  
 freq=freq.append(freq\_2,ignore\_index=True)  
 freq=freq.groupby(by='docs')['freq'].sum().reset\_index().sort\_values(by='freq',ascending=False,ignore\_index=True)  
 docs.update(f2)   
 self.files=docs  
 self.result\_lister(self.files,freq)  
 elif self.bool\_flag.isChecked() and self.zone\_flag.isChecked()==False:  
 self.files=boolean().bool\_match(tk.tokenizec(self.query\_tokens), self.flags)  
 self.result\_lister(self.files)  
 elif self.sem\_ont.isEnabled() and self.zone\_flag.isChecked()==False:   
 qr=self.query\_tokens  
 self.files,freq= semantic().semantic\_search(qr,self.flags)  
 self.result\_lister(self.files,freq)  
 elif self.cos\_sim\_flag.isChecked() and self.zone\_flag.isChecked()==False:  
 qr=self.query\_tokens   
 self.files= cosine\_sim().cosine(self.flags,qr)   
 self.result\_lister(self.files)   
 elif self.prox\_mtch\_flag.isChecked() and self.zone\_flag.isChecked()==False:  
 qr=self.query\_tokens   
 self.files,freq=prox().proximity(qr,self.flags)  
 self.result\_lister(self.files,freq)  
 else: # other checks for flags can be done here  
 query\_tokens=tk.tokenizec(self.query\_tokens)   
 self.stemmer(query\_tokens)  
 if self.flags['stop\_words'][0]==1:  
 Stopwords=stopwords.words('english')  
 self.stemmed\_query=[tk for tk in self.stemmed\_query if tk not in Stopwords]  
 freq, self.files= matcher(self.stemmed\_query,self.flags)  
 self.result\_lister(self.files,freq)  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))  
 self.label.setText(\_translate("MainWindow", "Search"))  
 self.reg\_ex\_flag.setText(\_translate("MainWindow", "Regular Expression"))  
 self.bool\_flag.setText(\_translate("MainWindow", "Boolean Query"))  
 self.exct\_match\_flag.setText(\_translate("MainWindow", "Exact Match"))  
 self.cos\_sim\_flag.setText(\_translate("MainWindow", "Cosine Similarity"))  
 self.stemmers\_list.setItemText(0, \_translate("MainWindow", "No Stemming"))  
 self.stemmers\_list.setItemText(1, \_translate("MainWindow", "Porter Stemmer"))  
 self.stemmers\_list.setItemText(2, \_translate("MainWindow", "Snowball Stemmer"))  
 self.stemmers\_list.setItemText(3, \_translate("MainWindow", "Lancaster Stemmer"))  
 self.stemmers\_list.setItemText(4, \_translate("MainWindow", "Customized Stemmer"))  
 self.prox\_mtch\_flag.setText(\_translate("MainWindow", "Proximity Matching"))  
 self.wild\_card\_flag.setText(\_translate("MainWindow", "Wild Card Matching"))  
 self.zone\_flag.setText(\_translate("MainWindow", "Zone Based Matching"))  
 self.sem\_flag.setText(\_translate("MainWindow", "Semantic-Based Matching"))  
 self.sem\_ont.setItemText(0, \_translate("MainWindow", "Word Net"))  
 self.sem\_ont.setItemText(1, \_translate("MainWindow", "Yago5"))  
 self.stp\_wrd\_flag.setText(\_translate("MainWindow", "Stop Words Removal"))  
 self.reg\_ex\_list.setItemText(0, \_translate("MainWindow", "Match"))  
 self.reg\_ex\_list.setItemText(1, \_translate("MainWindow", "Search"))  
 self.reg\_ex\_list.setItemText(2, \_translate("MainWindow", "Find All"))  
 self.search.setText(\_translate("MainWindow", "Search"))  
 self.label\_3.setText(\_translate("MainWindow", "Title:"))  
 self.label\_6.setText(\_translate("MainWindow", "Description:"))  
 self.label\_7.setText(\_translate("MainWindow", "Notes:"))  
 self.label\_10.setText(\_translate("MainWindow", "Location:"))  
 self.label\_12.setText(\_translate("MainWindow", "Date:"))  
 self.results\_lable.setText(\_translate("MainWindow", "0 Results Found"))  
 self.url\_flag.setText(\_translate("MainWindow", "URL"))  
if \_\_name\_\_ == "\_\_main\_\_":  
 import sys  
 app = QtWidgets.QApplication(sys.argv)  
 MainWindow = QtWidgets.QMainWindow()  
 ui = Ui\_MainWindow()  
 ui.setupUi(MainWindow)  
 MainWindow.show()  
 sys.exit(app.exec\_())

### **Boolean Function**

from nltk.corpus import stopwords  
import numpy as np  
import json  
import pandas as pd  
from PyQt5 import QtWidgets  
class boolean ():  
 def loader(self,flags):  
 stem\_no\_stop={'No Stemming': "without\_stopword\_tokenized.json", 'Porter Stemmer': 'without\_stopword\_ps\_stemmed.json','Snowball Stemmer':'without\_stopword\_sb\_stemmed.json',  
 'Lancaster Stemmer':'without\_stopword\_lc\_stemmed.json','Customized Stemmer':'without\_stopword\_custom\_stemmed.json'}  
  
 stem\_with\_stop={'No Stemming': "tokenized.json", 'Porter Stemmer': 'ps\_stemmed.json','Snowball Stemmer':'sb\_stemmed.json',  
 'Lancaster Stemmer':'lc\_stemmed.json','Customized Stemmer':'custom\_stemmed.json'}  
 if flags['stop\_words'][0] == True:  
 self.data = json.loads(open('stemmers/'+stem\_no\_stop[flags['stemmer'][0]]).read())  
 self.pos\_index=json.loads(open('pos\_index/'+'pos\_'+stem\_no\_stop[flags['stemmer'][0]]).read())  
 elif flags['stop\_words'][0] == False:  
 self.data = json.loads(open('stemmers/'+stem\_with\_stop[flags['stemmer'][0]]).read())  
 self.pos\_index=json.loads(open('pos\_index/'+'pos\_'+stem\_with\_stop[flags['stemmer'][0]]).read())  
 self.file\_map=json.loads(open('pos\_index/'+'file\_map.json').read())  
   
 def bool\_match(self,text,flags,tag=None):  
 self.loader(flags)   
 operator\_words = []  
 cnt = 1  
 query\_words = []  
 not\_found=[]  
 for i,word in enumerate(text):  
 if word.lower() != "and" and word.lower() != "or" and word.lower() != "not":  
 query\_words.append(word.lower())  
 if len(operator\_words) != len(query\_words)-1:  
 operator\_words.append("and")  
 else:  
 if text[i+1].lower()=='not' or text[i+1]==text[i] or text[i+1] in ['and','or','not']:  
 pass  
 else:  
 operator\_words.append(word.lower())   
 if len(query\_words)==1:  
 operator\_words.append('and')  
 print(len(query\_words))  
 print(len(operator\_words))  
 total\_files = len(self.file\_map) ## need to stor as file  
 one\_hot\_vector = []  
 one\_hot\_vector\_of\_all\_words = []  
 for word in (query\_words):  
 if word.lower() in set(self.pos\_index.keys()):  
 one\_hot\_vector = [0] \* total\_files  
   
 if tag != None:  
 foundlist=[]  
 for file in self.pos\_index[word][1].keys():  
 if tag in self.pos\_index[word][1][file].keys():  
 foundlist.append(file)  
 else:  
 foundlist = list(self.pos\_index[word][1].keys())  
   
 for i,doc in enumerate(foundlist):  
 one\_hot\_vector[int(doc)] = 1  
 one\_hot\_vector\_of\_all\_words.append(one\_hot\_vector)  
 else:  
 not\_found.append(word)  
 vector = [0] \* total\_files  
 one\_hot\_vector\_of\_all\_words.append(vector)  
 print(word,' not found')   
 if len(query\_words)==1:  
 vector = [1] \* total\_files  
 one\_hot\_vector\_of\_all\_words.append(vector)  
 print(len(one\_hot\_vector\_of\_all\_words))  
 for word in operator\_words:  
 copy\_list1 = one\_hot\_vector\_of\_all\_words[0]  
 copy\_list2 = one\_hot\_vector\_of\_all\_words[1]  
 if word == "and" :  
 cells = [w1 & w2 for (w1,w2) in zip(copy\_list1,copy\_list2)]  
 one\_hot\_vector\_of\_all\_words.remove(copy\_list1)  
 one\_hot\_vector\_of\_all\_words.remove(copy\_list2)  
 one\_hot\_vector\_of\_all\_words.insert(0, cells)   
 elif word == "or":  
 cells = [w1 | w2 for (w1,w2) in zip(copy\_list1,copy\_list2)]  
 one\_hot\_vector\_of\_all\_words.remove(copy\_list1)  
 one\_hot\_vector\_of\_all\_words.remove(copy\_list2)  
 one\_hot\_vector\_of\_all\_words.insert(0, cells)  
 elif word == "not":  
 cells = [not w1 for w1 in copy\_list2]  
 cells = [int(b == True) for b in cells]  
 one\_hot\_vector\_of\_all\_words.remove(copy\_list2)  
 one\_hot\_vector\_of\_all\_words.remove(copy\_list1)  
 cells = [w1 & w2 for (w1,w2) in zip(copy\_list1,cells)]  
 one\_hot\_vector\_of\_all\_words.insert(0, cells)  
 files\_results = []   
 list\_doc = {}  
 one\_hot\_list = one\_hot\_vector\_of\_all\_words[0]  
 cnt = 0  
 for index in one\_hot\_list:  
 if index == 1:  
 files\_results.append(self.file\_map[str(cnt)])  
 list\_doc[self.file\_map[str(cnt)]]= self.data[self.file\_map[str(cnt)]]  
 cnt = cnt+1  
 results={}   
 data = json.loads(open("dataset.json").read())  
 for file in files\_results:  
 results[file]=data[file]  
 return results

### **Zone Based Matching**

import re  
from tokenizec import NlpTokenizer  
from nltk.corpus import stopwords  
import pandas as pd  
import numpy as np  
from boolean\_query import boolean  
from reg\_ex import regular\_exp\_matcher  
import json  
from exact\_match import exact\_match\_fun  
from collections import Counter  
from semantic import semantic  
from cosine import cosine\_sim  
from proximity import prox  
from PyQt5 import QtWidgets  
def zone\_matching(text,flags):  
 fields=['title','location','date','description','notes']  
 zones=re.findall('\((.\*?)\)',text,re.IGNORECASE)  
 texts={}  
 docs={}  
 file\_names=[]  
 wilds={}  
 freq=pd.DataFrame(data=None,columns=['docs,freq'])  
 tk=NlpTokenizer()  
 if len(zones)==0:  
 msg= QtWidgets.QMessageBox()  
 msg.setWindowTitle('Notification')  
 msg.setText('Syntax Error.\nMake sure the syntax is:\n(zone) Text')  
 msg.setIcon(QtWidgets.QMessageBox.Warning)  
 x=msg.exec\_()  
 return docs, texts, freq.reset\_index()  
 s\_index=0  
 indexes=[]  
 for zone in zones:  
 i=text.index(zone,s\_index)  
 indexes.append(i)  
 s\_index+=i+len(zone)  
 for i in reversed(range(len(zones))):  
 if zones[i].lower() in fields:  
 if i==len(zones)-1:  
 temp = text[indexes[i]+len(zones[i])+1:]  
 else:  
 temp=text[indexes[i]+len(zones[i])+1:indexes[i+1]-1]  
 if flags['wildcard'][0] == 1:  
 wilds[zones[i].upper()]=re.findall("\w\*\\*\w\*",temp)   
 txt=re.findall("\\*?\w\w\*[.|:|?|'||...|!|-|()]?\w\w\*\\*?",temp)   
 if zones[i].upper() in texts.keys():  
 texts[zones[i].upper()].extend([item for item in txt if item not in wilds[zones[i].upper()]])  
 else:  
 texts[zones[i].upper()]=[item for item in txt if item not in wilds[zones[i].upper()]]  
 for zone in wilds.keys():  
 for tk in wilds[zone]:  
 exp="\\b"+tk.replace('\*','\w\*')  
 files={}  
 files =regular\_exp\_matcher().reg\_ex('Find All',exp,tags=[zone])  
 file\_names.extend(list(files.keys()))  
 docs.update(files)  
 elif flags['boolean'][0] == 1:   
 files=boolean().bool\_match(tk.tokenizec(temp), flags,tag=zones[i].upper())  
 file\_names.extend(list(files.keys()))  
 docs.update(files)  
   
 elif flags['exact\_match'][0]==1:  
 freq\_exct,files=exact\_match\_fun(temp,flags,zone=zones[i].upper())  
 file\_names.extend(list(files.keys()))  
 docs.update(files)  
 freq.append(freq\_exct,ignore\_index=True)  
 elif flags['semantic'][0]==1:  
 files, freq\_sem=semantic().semantic\_search(temp,flags,zone=zones[i].upper())  
 file\_names.extend(list(files.keys()))  
 docs.update(files)  
 freq.append(freq\_sem,ignore\_index=True)  
 elif flags['cosine'][0] ==1:  
 docs=cosine\_sim().cosine(flags,temp,zones=[zones[i].upper()])   
 elif flags['proximity'][0] ==1:  
 docs,freq\_prox=prox().proximity(temp,flags,zones=[zones[i].upper()])  
 freq.append(freq\_prox,ignore\_index=True)  
 else:  
 texts[zones[i].upper()]=tk.tokenizec(temp)   
 if flags['stop\_words'][0] == 1:   
 Stopwords=stopwords.words('english')  
 for key in texts.keys():  
 tokens=[]  
 for token in texts[key]:  
 if token not in Stopwords:  
 tokens.append(token)  
 texts[key]=tokens  
 else:  
 msg= QtWidgets.QMessageBox()  
 msg.setWindowTitle('Notification')  
 msg.setText('Syntax Error.\nMake sure the syntax is:\n(zone) Text')  
 msg.setIcon(QtWidgets.QMessageBox.Warning)  
 x=msg.exec\_()  
 freq\_files=Counter(file\_names)  
 freq=freq.append(pd.DataFrame(zip(freq\_files.keys(),freq\_files.values()),columns=['docs','freq']).sort\_values(by='freq',ascending=False))  
 freq=freq.groupby(by='docs')['freq'].sum().reset\_index().sort\_values(by='freq',ascending=False,ignore\_index=True)  
 t2={}  
 for zone in texts.keys():  
 if len(texts[zone]):  
 t2[zone]=texts[zone]   
 print(t2)  
 return docs, texts, freq.reset\_index()

### **Cosine Similarity**

from nltk.corpus import stopwords  
from collections import Counter  
from tokenizec import NlpTokenizer  
from tqdm import tqdm  
import numpy as np  
import math  
import json  
from check\_search import searcher  
import pandas as pd  
class cosine\_sim:  
 def doc\_freq(self,word):  
 c = 0  
 try:  
 c = self.DF[word]  
 except:  
 pass  
 return c  
 def cosine(self,flags,query, zones=None):  
 stem\_no\_stop={'No Stemming': "without\_stopword\_tokenized.json", 'Porter Stemmer': 'without\_stopword\_ps\_stemmed.json','Snowball Stemmer':'without\_stopword\_sb\_stemmed.json',  
 'Lancaster Stemmer':'without\_stopword\_lc\_stemmed.json','Customized Stemmer':'without\_stopword\_custom\_stemmed.json'}  
 stem\_with\_stop={'No Stemming': "tokenized.json", 'Porter Stemmer': 'ps\_stemmed.json','Snowball Stemmer':'sb\_stemmed.json',  
 'Lancaster Stemmer':'lc\_stemmed.json','Customized Stemmer':'custom\_stemmed.json'}  
 if flags['stop\_words'][0] == True:  
 self.data = json.loads(open('stemmers/'+stem\_no\_stop[flags['stemmer'][0]]).read())  
 elif flags['stop\_words'][0] == False:  
 self.data = json.loads(open('stemmers/'+stem\_with\_stop[flags['stemmer'][0]]).read())  
 self.file\_map=json.loads(open('pos\_index/'+'file\_map.json').read())   
 self.data1={}  
 if zones == None:   
 zones=['TITLE','DESCRIPTION','NOTES','LOCATION']  
 for d in self.data.keys():  
 elist=[]  
 for tag in self.data[d].keys():  
 if tag in zones:  
 for word\_list in self.data[d][tag]:  
 #we should add stopwords if the selected file is note containg stopwords  
 if word\_list != 'None':  
 elist.append(word\_list)  
 self.data1[d]=elist  
 self.N = len (self.data1.keys())  
 fn=[]  
 for k in self.data1.keys():  
 fn.append(self.data1[k])  
 self.DF = {}  
 for i in range(self.N):  
 tokens = fn[i]  
 for w in tokens:  
 try:  
 self.DF[w].add(i)  
 except:  
 self.DF[w] = {i}  
 for i in self.DF:  
 self.DF[i] = len(self.DF[i])  
 self.total\_vocab\_size = len(self.DF)  
 self.total\_vocab = [x for x in self.DF]  
 doc = 0  
 tf\_idf = {}  
 for i in tqdm(range(self.N)):  
 tokens = fn[i]  
 counter = Counter(tokens)  
 words\_count = len(tokens)  
 for token in np.unique(tokens):  
 tf=counter[token]  
 df = self.doc\_freq(token)  
 idf = np.log((self.N+1)/(df+1))  
 tf\_idf[doc, token] = tf\*idf  
 doc += 1  
 self.D = np.zeros((self.N, self.total\_vocab\_size))  
 for i in tqdm(tf\_idf):  
 try:  
 ind = self.total\_vocab.index(i[1])  
 self.D[i[0]][ind] = tf\_idf[i]  
 except:  
 pass  
 Q = self.cosine\_similarity(20000,query,flags)  
 return Q  
 def gen\_vector(self,tokens):  
 Q = np.zeros((len(self.total\_vocab)))  
 counter = Counter(tokens)  
 words\_count = len(tokens)  
 query\_weights = {}  
 for token in tqdm(np.unique(tokens)):  
 tf = counter[token]/words\_count  
 df = self.doc\_freq(token)  
 idf = math.log((self.N+1)/(df+1))  
 try:  
 ind = self.total\_vocab.index(token)  
 Q[ind] = tf\*idf  
 except:  
 pass  
 return Q  
 def cosine\_sim(self,a, b):  
 cos\_sim = np.dot(a, b)/(np.linalg.norm(a)\*np.linalg.norm(b))  
 return cos\_sim  
 def cosine\_similarity(self,k, query,flags):  
 tk=NlpTokenizer()  
 token = tk.tokenizec(query)  
 if flags['stop\_words'][0]==1:  
 Stopwords=stopwords.words('english')  
 token=[item for item in token if item not in Stopwords]  
 if flags['stemmer'][0] == 'Porter Stemmer':   
 token=searcher().Porter\_Stemmer(token)  
 elif flags['stemmer'][0] == 'Snowball Stemmer':   
 token=searcher().Snowball\_Stemmer(token)  
 elif flags['stemmer'][0] == 'Lancaster Stemmer':  
 token=searcher().Lancaster\_Stemmer(token)  
 elif flags['stemmer'][0] == 'Customized Stemmer':  
 token=searcher().Customized\_Stemmer(token)   
 d\_cosines = []  
 query\_vector = self.gen\_vector(token)  
 for d in tqdm(self.D):  
 result=self.cosine\_sim(query\_vector, d)  
 d\_cosines.append(result)  
 out = np.array(d\_cosines).argsort()[-k:][::-1]  
 dictt={}  
 for index,d in enumerate(self.data1.keys()):  
 dictt[index]=d  
 fn\_list=[]  
 fn\_cosine=[]  
 for i in out:  
 fn\_list.append(dictt[i])  
 fn\_cosine.append(d\_cosines[i])  
 df=pd.DataFrame(list(zip(fn\_list,fn\_cosine)),columns=['file','cosine'])  
 df=df[df['cosine']>0.01]['file']  
 org\_files=json.loads(open('dataset.json').read())  
 docs={}  
 for key in df:  
 docs[key]=org\_files[key]  
 return docs

### **Semantic Based Matching**

from tokenizec import NlpTokenizer  
from nltk.corpus import wordnet as wn  
import pandas as pd  
from tqdm import tqdm  
import numpy as np  
from nltk.corpus import stopwords  
from check\_search import searcher  
from matcher import matcher  
from boolean\_query import boolean  
from collections import Counter  
import json  
import re  
class semantic:  
 def semantic\_search(self,text,flags,zone=None):  
 if flags['ontology'][0]=='Word Net':  
 print('wordnet')  
 docs, freq=self.wordnet(text, flags,zone)  
 #wordnet  
 elif flags['ontology'][0]=='Yago5':  
 print('yago')  
 docs, freq=self.yago(text,flags,zone)  
 #Yago5  
 return docs, freq  
 def generate\_ngrams(self,s, n):  
 s = s.lower()  
 s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s)  
 tokens = [token for token in s.split(" ") if token != ""]  
 ngrams = zip(\*[tokens[i:] for i in range(n)])  
 return [" ".join(ngram) for ngram in ngrams]  
 def wordnet(self,text, flags,zone):  
 tc=NlpTokenizer()  
 tkn=tc.unitoknizer(text)  
 sem=[]  
 for tk in tkn:  
 token=wn.synsets(tk)  
 sem.append(tk)  
 for tkk in range(len(token)):  
 for index in range(len(token[tkk].lemmas())):  
 sem.append(token[tkk].lemmas()[index].name().lower())  
 sem=list(set(sem))  
 if flags['stop\_words'][0]==1:  
 Stopwords=stopwords.words('english')  
 sem=[item for item in sem if item not in Stopwords]  
 if flags['stemmer'][0] == 'Porter Stemmer':   
 sem=searcher().Porter\_Stemmer(sem)  
 elif flags['stemmer'][0] == 'Snowball Stemmer':   
 sem=searcher().Snowball\_Stemmer(sem)  
 elif flags['stemmer'][0] == 'Lancaster Stemmer':  
 sem=searcher().Lancaster\_Stemmer(sem)  
 elif flags['stemmer'][0] == 'Customized Stemmer':  
 sem=searcher().Customized\_Stemmer(sem)  
 if zone == None:  
 freq,docs=matcher(sem, flags)  
 elif zone != None:   
 freq,docs=matcher(sem, flags, tags=[zone.upper()])  
 return docs, freq  
 def yago(self,text, flags, zone=None):  
 print('yago')  
 results = []  
 ln = len(text.split())  
 for x in range(ln):  
 result = self.generate\_ngrams(text,ln- x)  
 results.append(result)  
 token=results  
 yago=[text]  
 for tk in tqdm(token):  
 for t in tqdm(tk):  
 y=self.doyago(t)  
 yago.extend(y)  
 file\_names=[]  
 docs={}  
 if zone == None:  
 freq, docs=matcher(yago, flags)  
 return docs, freq  
 elif zone != None:   
 freq, docs=matcher(yago,flags,tags=[zone.upper()])  
 return docs, freq  
 def doyago(self,text):  
 newtext='<'+text.replace(' ','\_')+'>'  
 newtext=newtext.strip()  
 newtext=newtext.lower()  
 path='yago\_df.csv'  
  
 df=pd.read\_csv(path)  
 data=df['subject']==newtext  
 data2=df[data]  
 sem\_list=list(data2['object'])  
 yago\_sem=[]  
 #yago\_sem=[text]  
 for lt in range(len(sem\_list)):  
 text=sem\_list[lt].replace('\_',' ')  
 text=text.replace('<','')  
 text=text.replace('>','')  
 text=text.replace('-','')  
 text=text.strip()  
 yago\_sem.append(text)  
 return yago\_sem

### **Proximity Matching**

import json  
from tokenizec import NlpTokenizer  
import re  
from check\_search import searcher  
from tokenizec import NlpTokenizer  
import pandas as pd  
import numpy as np   
from PyQt5 import QtWidgets  
class prox:  
 def get\_fn(self,text1,text2,spacing):  
 text1=text1.strip()  
 text2=text2.strip()  
 token1=NlpTokenizer().tokenizec(text1)  
 token2=NlpTokenizer().tokenizec(text2)  
 f1=[]  
 f1\_s=pd.DataFrame(data=None,columns=['docs','space'])  
 for i in range(len(token1)-1):  
 docs=self.finder(token1[i],token1[i+1],1)  
 if len(f1)==0:  
 f1.extend(docs['docs'])  
 else:  
 d1=set(f1)  
 d2=set(docs['docs'])  
 f1=list(d1.intersection(d2))  
 f1\_s=docs[docs['docs'].isin(f1)]  
 f2=[]   
 f2\_s=pd.DataFrame(data=None,columns=['docs','space'])  
 for i in range(len(token2)-1):  
 docs=self.finder(token2[i],token2[i+1],1)  
 if len(f2)==0:  
 f2.extend(docs['docs'])  
 else:  
 d1=set(f2)  
 d2=set(docs['docs'])  
 f2=list(d1.intersection(d2))   
 f2\_s=docs[docs['docs'].isin(f2)]  
 f3=[]  
 f3\_s=pd.DataFrame(data=None,columns=['docs','space'])  
 docs=self.finder(token1[-1],token2[0],spacing)  
 f3=docs['docs']  
 f3\_s=docs[docs['docs'].isin(f3)]  
 freq=pd.DataFrame(data=None,columns=['docs','space'])  
 if len(token1)>1 and len(token2)>1:  
 d1=set(f1)  
 d2=set(f2)  
 d3=set(f3)  
 f4=list(d1.intersection(d2,d3))  
 freq=freq.append(f1\_s,ignore\_index=True)  
 freq=freq.append([f2\_s,f3\_s],ignore\_index=True)  
 freq=freq[freq['docs'].isin(f4)]  
 freq=freq.groupby('docs').max('space').reset\_index().sort\_values('space',ascending=False)   
 elif len(token1)==1 and len(token2)>1:  
 d2=set(f2)  
 d3=set(f3)  
 f4=list(d2.intersection(d3))  
 freq=freq.append(f2\_s,ignore\_index=True)  
 freq=freq.append(f3\_s,ignore\_index=True)  
 freq=freq[freq['docs'].isin(f4)]  
 freq=freq.groupby('docs').max('space').reset\_index().sort\_values('space',ascending=False)  
 elif len(token1)>1 and len(token2)==1:  
 d1=set(f1)  
 d3=set(f3)  
 f4=list(d1.intersection(d3))  
 freq=freq.append(f1\_s,ignore\_index=True)  
 freq=freq.append(f3\_s,ignore\_index=True)  
 freq=freq[freq['docs'].isin(f4)]  
 freq=freq.groupby('docs').max('space').reset\_index().sort\_values('space',ascending=False)  
 elif len(token1)==1 and len(token2)==1:  
 f4=f3  
 freq=freq.append(f3\_s,ignore\_index=True)  
 freq=freq[freq['docs'].isin(f4)]  
 freq=freq.groupby('docs').max('space').reset\_index().sort\_values('space',ascending=False)  
 return f4,freq.reset\_index()  
 def finder(self,token1, token2,spacing):   
 file\_list=[]   
 spaces=[]  
 for fn in self.data.keys():  
 for tags in self.zones:  
 values=self.data[fn][tags]  
 if token1 in values and token2 in values:  
 for key in self.file\_map.keys():  
 if fn ==self.file\_map[key]:  
 k=key  
 break  
 idx1=self.pos\_index[token1][1][k][tags]  
 idx2=self.pos\_index[token2][1][k][tags]  
 for i in idx1:  
 for j in idx2:  
 if j-i <= spacing and j-i > 0:  
 file\_list.append(fn)  
 spaces.append(j-i)  
 result=pd.DataFrame(zip(file\_list,spaces),columns=['docs','space'])  
 result=result.groupby(by='docs').max('space').reset\_index()  
 return result  
 def proximity(self,query,flags, zones=None):  
 if zones==None:  
 self.zones=['TITLE','LOCATION','DESCRIPTION','NOTES']  
 else: self.zones=zones  
 stem\_with\_stop={'No Stemming': "tokenized.json", 'Porter Stemmer': 'ps\_stemmed.json','Snowball Stemmer':'sb\_stemmed.json',  
 'Lancaster Stemmer':'lc\_stemmed.json','Customized Stemmer':'custom\_stemmed.json'}  
 self.data = json.loads(open('stemmers/'+stem\_with\_stop[flags['stemmer'][0]]).read())  
 self.pos\_index=json.loads(open('pos\_index/'+'pos\_'+stem\_with\_stop[flags['stemmer'][0]]).read())  
 self.file\_map=json.loads(open('pos\_index/'+'file\_map.json').read())  
 query\_tokens = re.findall('(/[0-9][0-9]\*)',query,re.IGNORECASE)  
 if len(query\_tokens):  
 if query.index(query\_tokens[0])==0:  
 query=query[query.index(query\_tokens[0])+len(query\_tokens[0]):]  
 query\_tokens = re.findall('(/[0-9])',query,re.IGNORECASE)  
 if len(query\_tokens):   
 if query[-len(query\_tokens[-1]):]==query\_tokens[-1]:  
 query=query[:-len(query\_tokens[-1])]  
 query\_tokens = re.findall('(/[0-9])',query,re.IGNORECASE)  
 if len(query\_tokens):   
 if flags['stemmer'][0] == 'Porter Stemmer':   
 query\_tokens=searcher().Porter\_Stemmer(query\_tokens)  
 elif flags['stemmer'][0] == 'Snowball Stemmer':   
 query\_tokens=searcher().Snowball\_Stemmer(query\_tokens)  
 elif flags['stemmer'][0] == 'Lancaster Stemmer':  
 query\_tokens=searcher().Lancaster\_Stemmer(query\_tokens)  
 elif flags['stemmer'][0] == 'Customized Stemmer':  
 query\_tokens=searcher().Customized\_Stemmer(query\_tokens)   
 indexes=[]  
 s\_index=0  
 for tk in query\_tokens:  
 index=query.index(tk,s\_index)  
 indexes.append(index)  
 s\_index+=index+len(tk)  
 doc\_list=[]   
 freqs=[]  
 if len(query\_tokens)==1:  
 temp = query[:query.index(query\_tokens[0])]  
 temp2 = query[ query.index(query\_tokens[0])+len(query\_tokens[0])+1:]  
 number=int(query\_tokens[0].replace('/',''))  
 fn, freq=self.get\_fn(temp,temp2,number)  
 if len(fn):  
 doc\_list.append(fn)  
 freqs.append(freq)  
 else:  
 for i in range(len(query\_tokens)):  
 if i==0 :  
 temp = query[:indexes[i]].strip()  
 temp2 = query[indexes[i]+len(query\_tokens[i]):indexes[i+1]].strip()  
 number=int(query\_tokens[i].replace('/',''))  
 fn,freq=self.get\_fn(temp,temp2,number)  
 if len(fn):  
 doc\_list.append(fn)  
 freqs.append(freq)   
 elif i==len(query\_tokens)-1:  
 temp = query[indexes[i-1]+len(query\_tokens[i-1]): indexes[i]].strip()  
 temp2= query[indexes[i]+len(query\_tokens[i]):].strip()  
 number=int(query\_tokens[i].replace('/',''))  
 fn,freq=self.get\_fn(temp,temp2,number)  
 if len(fn):  
 doc\_list.append(fn)  
 freqs.append(freq)  
 else:  
 temp = query[indexes[i-1]+len(query\_tokens[i-1]):indexes[i]].strip()  
 temp2 = query[indexes[i]+len(query\_tokens[i]):indexes[i+1]].strip()  
 number=int(query\_tokens[i].replace('/',''))  
 fn,freq=self.get\_fn(temp,temp2,number)  
 if len(fn):  
 doc\_list.append(fn)  
 freqs.append(freq)  
 docs=[]  
 for l in doc\_list:  
 if len(docs)==0:  
 docs.extend(l)  
 else:  
 f1=set(docs)  
 f2=set(l)  
 docs=list(f1.intersection(f2))  
 freq=freq.groupby('docs').max('space').reset\_index().sort\_values('space',ascending=False)  
 freq=freq[freq['docs'].isin(docs)].reset\_index()  
 org\_files=json.loads(open('dataset.json').read())  
 result={}  
 for key in docs:  
 result[key]=org\_files[key]  
 else:   
 msg= QtWidgets.QMessageBox()  
 msg.setWindowTitle('Notification')  
 msg.setText('Syntax Error.\n Make sure the syntax is:\nText1 /K / Text2')  
 msg.setIcon(QtWidgets.QMessageBox.Warning)  
 x=msg.exec\_()  
 result={}  
 return result, freq

### **Regular Expression**

import pandas as pd  
import re  
import json  
from tqdm import tqdm  
class regular\_exp\_matcher():  
 def reg\_ex (self, func\_type,reg\_ex, data=None, tags=None):  
 if tags==None:  
 tags=['DOCNO', 'TITLE', 'DESCRIPTION', 'NOTES', 'LOCATION', 'DATE', 'IMAGE', 'THUMBNAIL']  
 if data==None:  
 data = json.loads(open("dataset\_lower.json").read())  
 docs={}  
 flag=0  
 while flag<2:   
 if func\_type=='Match' :  
 for file in data.keys():  
 for tag in tags:  
 if re.match(reg\_ex,str(data[file][tag]),re.IGNORECASE) is not None:   
 docs[file]=data[file]  
 break  
 elif func\_type=='Search' :  
 for file in data.keys():  
 for tag in tags:  
 if re.search(reg\_ex,str(data[file][tag]),re.IGNORECASE) is not None:   
 docs[file]=data[file]  
 break  
 elif func\_type=='Find All':  
 for file in data.keys():  
 for tag in tags:  
 if len(re.findall(reg\_ex,str(data[file][tag]),re.IGNORECASE)):   
 docs[file]=data[file]  
 break  
 if len(docs)==0 and flag<1:  
 data = json.loads(open("dataset.json").read())  
 flag+=1  
 else: flag=2  
 return docs

### **Exact Match**

import re  
import json  
from reg\_ex import regular\_exp\_matcher  
from tokenizec import NlpTokenizer  
from matcher import matcher  
from collections import Counter  
from nltk.corpus import stopwords  
from check\_search import searcher  
import pandas as pd  
import numpy as np  
data = json.loads(open("dataset\_lower.json").read())  
def exact\_match\_fun(query,flags,zone=None):  
 docs={}  
 tk = NlpTokenizer()  
 x = (re.findall(r'"(.\*?)"', query.lower()))  
 input\_match = x[0]  
 input\_token = tk.tokenizec(input\_match)  
 tokenized = tk.tokenizec(query.lower())  
 tokens = [item for item in tokenized if item not in input\_token]  
 final\_tokens=[]  
 if flags['stop\_words'][0] == 1:   
 Stopwords=stopwords.words('english')  
 for t in tokens:  
 if t not in Stopwords:  
 final\_tokens.append(t)  
 if flags['stemmer'][0] == 'Porter Stemmer':   
 final\_tokens=searcher().Porter\_Stemmer(final\_tokens)  
 elif flags['stemmer'][0] == 'Snowball Stemmer':   
 final\_tokens=searcher().Snowball\_Stemmer(final\_tokens)  
 elif flags['stemmer'][0] == 'Lancaster Stemmer':  
 final\_tokens=searcher().Lancaster\_Stemmer(final\_tokens)  
 elif flags['stemmer'][0] == 'Customized Stemmer':  
 final\_tokens=searcher().Customized\_Stemmer(final\_tokens)  
   
 if zone != None:  
 files\_exact =regular\_exp\_matcher().reg\_ex('Find All',input\_match,tags=[zone.upper()])  
 freq\_t, files\_tokens= matcher(final\_tokens,flags,tags=[zone.upper()])  
 docs.update(files\_exact)  
 docs.update(files\_tokens)  
 else:  
 files\_exact =regular\_exp\_matcher().reg\_ex('Find All',input\_match)  
 freq\_t, files\_tokens= matcher(final\_tokens,flags)  
 docs.update(files\_exact)  
 docs.update(files\_tokens)  
 f1=list(files\_exact.keys())  
 freq=Counter(f1)  
 freq=pd.DataFrame(zip(freq.keys(),freq.values()),columns=['docs','freq']).sort\_values(by='freq',ascending=False,ignore\_index=True)  
 freq=freq.append(freq\_t,ignore\_index=True)  
 freq=freq.groupby(by='docs')['freq'].sum().reset\_index().sort\_values(by='freq',ascending=False,ignore\_index=True)  
 return freq, docs

# **Conclusions and Future Work**

This project implements many NLP techniques that enables the user to perform different types of queries. The project has the following advantages:

1. Simple UX design
2. Lean processing (processes designed to minimize errors)
3. Agile Development
4. High performance (Offline Indexing)
5. Simple result visualization
6. Responsive interface

On the other hand, the project has some disadvantages:

1. Offline data is stored locally, which result in performance issues in terms of storage and system requirements
2. Yago data is huge and stored locally, which affects performance
3. Yago performance is low since it works in a predefined set of words
4. Cosine matching vectors are generated at each query, which result in delay in the system performance.
5. Lemmatization is not fully developed in the custom stemmer

However, we look at the disadvantages as future improvements on the system, and other improvements as:

1. Offline data to be stored on a database. This would increase the performance of data retrieval
2. Yago dataset must be on a database such as Mongo DB
3. Yago recommendations should be implemented
4. Cosine matching vectors must be stored offline using npy or pickle
5. Developing more ontologies like mesh ontology
6. Developing tooltips for each button describing the features of the interface

The system is developed in a way that can be extended to add more features due to its agility.