ML BASED SOLICITATION IN FEDERAL TRANSCRIPTS

BITS SSZG628T: Dissertation

By

**DILIP PRASAD J**

2020MT12208

Dissertation work carried out at

**DXC TECHNOLOGY, CHENNAI**

****

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE**

**PILANI (RAJASTHAN)**

**APR 2022**

ML BASED SOLICITATION IN FEDERAL TRANSCRIPTS

BITS SSZG628T: Dissertation

By

**DILIP PRASAD J**

2020MT12208

Submitted in partial fulfillment of M.Tech. Software Systems

Dissertation work carried out at

**DXC TECHNOLOGY, CHENNAI**

Under the supervision of

**Dr Hans Beck**

**DXC Technology, Germany**

****

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE**

**PILANI (RAJASTHAN)**

**APR 2022**

#### CERTIFICATE

This is to certify that the Dissertation entitled “ML BASED SOLICITATION IN FEDERAL TRANSCRIPTS” and submitted by Dilip prasad Jayakumar having ID-No. 2020MT12208 for the partial fulfillment of the requirements of Master of Technology in software systems degree of BITS, embodies the bonafide work done by him/her under my supervision.

**Shape

Description automatically generated with medium confidence**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature of the Supervisor

Dr Hans Beck

Advisor, Data Science

Germany

Place : \_Germany\_\_\_\_\_\_\_\_\_\_\_

Date : \_\_20/04/2022 \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name, Designation & Organization &Location

# Acknowledgements

I would like to express my deep gratitude to Dr. Hans Beck my research supervisor, for their patient guidance, enthusiastic encouragement, and useful critiques of this research work. I would also like to thank Mrs. Hema Priya, for her advice and assistance in keeping my progress on schedule.

Finally, I wish to thank my wife and parents for their support and encouragement throughout my study.  
  
Dilip prasad Jayakumar

# ABSTRACT

Federal archive has evolved from Middle Ages to modern period, where archives are retained proofs of political and genealogical claims based on the authenticity. Archives do not neutrally store documents; rather, objects captured through archival practices are transformed into knowledge. In the creation phase of archives, records growth is expounded by modern electronic systems. Records will continue to be created and captured by the organization at an explosive rate as it conducts the business of the organization.

To perform search any specific transcripts without any pointers from the humongous set of files one can consider as cumbersome or almost impossible task. However, this have been the case for long where people are dedicatedly assigned to perform this as a full-time job. With the advent of technology and cheaper hardware has paved way to innovative and resource intensive computations executing complex algorithms to achieve what was once unimaginable.

Currently, the search operation on the German transcripts is done using Regex to perform a word-by-word search in all the files linearly. Where the time taken to complete the search increases based on the number for files which is inefficient. Additionally, the search only works with straight forward approach and will not be able to list mentions for related synonyms.

So, in this dissertation we will implement Natural Processing Techniques to not only understand the transcripts provided in German language but also perform innovative search mechanism. Where we could search by specific speaker and list all their mentions in all the archival documents and search with related synonyms that could be a potential match for that mention.

We will go a step ahead and review at Deep learning approaches to enhance the current NLP abilities to provide better search results.

For this Dissertation we will use nested transcripts from the below URL.

<https://www.bundesarchiv.de/cocoon/barch/0000/index.html>

Contents

[Acknowledgements 4](#_Toc102955060)

[ABSTRACT 5](#_Toc102955061)

[High level project architecture 8](#_Toc102955062)

[High level details 9](#_Toc102955063)

[Web Data Scraping 9](#_Toc102955064)

[CSV Data 9](#_Toc102955065)

[JSON 9](#_Toc102955066)

[JSON Data 9](#_Toc102955067)

[Queue 10](#_Toc102955068)

[Overall Code explained 10](#_Toc102955069)

[Queue Storage 10](#_Toc102955070)

[Module overview 11](#_Toc102955071)

[1. Limited Web Crawling file 12](#_Toc102955072)

[2. Extract Web Data file 15](#_Toc102955073)

[3. NLP Processing and training 16](#_Toc102955074)

[NLP phases 17](#_Toc102955075)

[Data preprocessing 18](#_Toc102955076)

[Tokenization 19](#_Toc102955077)

[Stop words 19](#_Toc102955078)

[Stemming 20](#_Toc102955079)

[Lemmatization 23](#_Toc102955080)

[Stemming vs Lemmatization 24](#_Toc102955081)

[Additional cleanup 25](#_Toc102955082)

[Lowercasing 25](#_Toc102955083)

[Removing stop words and punctuation 25](#_Toc102955084)

[Data Cleaning - Multi step process in single method 25](#_Toc102955085)

[Creating a Corpus of Data 26](#_Toc102955086)

[NLP Pipeline 26](#_Toc102955087)

[Exploratory Data Analysis (Visualization) 27](#_Toc102955088)

[Frequency Distribution 27](#_Toc102955089)

[Word cloud 29](#_Toc102955090)

[Gender Recognition 29](#_Toc102955091)

[Information Extraction Architecture 30](#_Toc102955092)

[Named Entity Recognition 30](#_Toc102955093)

[TF-IDF 31](#_Toc102955094)

[Model Creation 31](#_Toc102955095)

[WEB UI 31](#_Toc102955096)

[Search 32](#_Toc102955097)

[API 32](#_Toc102955098)

[API Repository 33](#_Toc102955099)

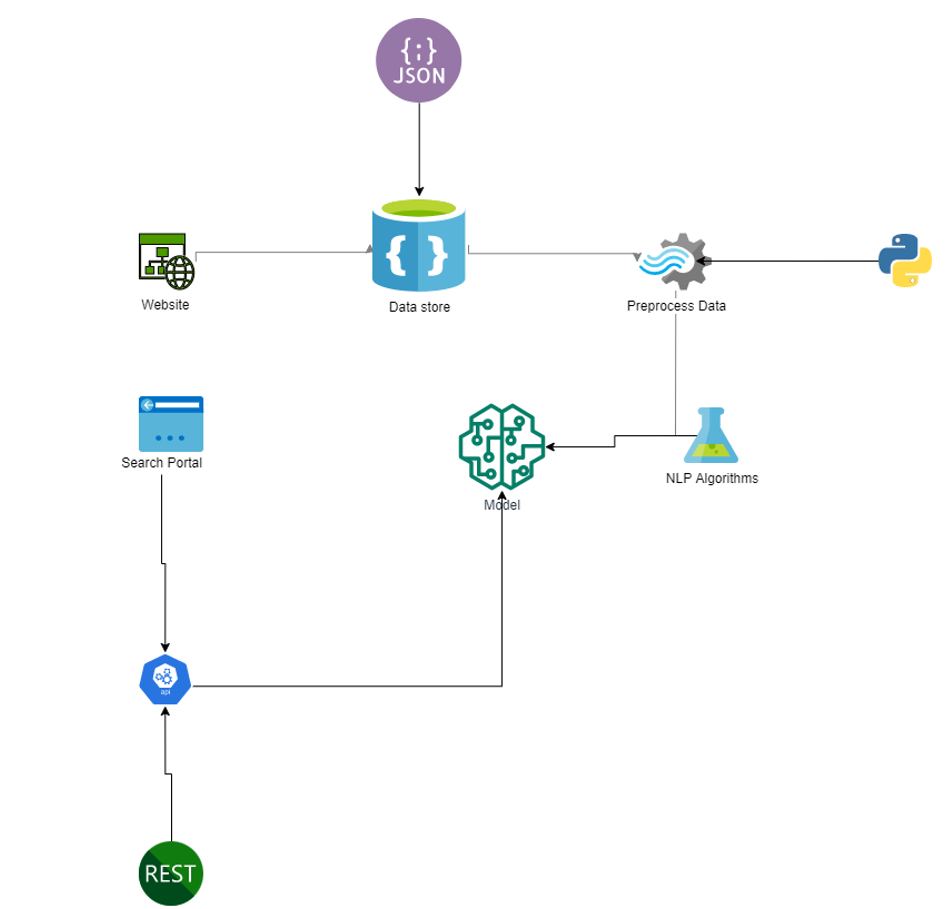
[Project Structure 34](#_Toc102955100)

[Required Libraries 34](#_Toc102955101)

[Conclusion 35](#_Toc102955102)

[References 36](#_Toc102955103)

# High level project architecture



1. Based on our requirements we will approach the solution in a segment-by-segment approach. We will initially crawl the entire website from the given root URL, the intention is to find all the sub-URL or sibling pages where we could find the relevant data. This helps to filter out unwanted data. The crawled URL is queued to azure queue for data extraction.
2. Once we have url’s queued to the azure queue, we could trigger the data extraction. Where the data from the given link is capture into a panda’s data frame initially and then transferred to a CSV file. The data extracted is normally plain text instead of rich text with images. Even there are multimedia data, we will ignore the same or adapt to recognize in the future releases/ versions. Which could be used based on our convenience at any point in time. This comes handy in case of unplanned downtime or data corruption.
3. After fetching the data into csv, we will do the necessary pre-processing steps required to clean up the text to extract the necessary details. All the preprocessing steps are done using Python, with the help of open-source libraries readily available on the internet. This process could be triggered independently of the previous step. With the help of NLP algorithms, we could perform semantic analysis on the given text, which helps us extract vital information which will be used during our search.

# High level details

## Web Data Scraping

Web scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. The web scraping software may directly access the World Wide Web using the Hypertext Transfer Protocol or a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis [4].

We will scrape the html data from the source website with readily available libraries. One such library is the Beautiful-Soup, with python we could easily extract required text from a complex HTML.

For the Sake for processing the data extracted will be converted to JSON format (JavaScript Object Notation)

Scraping will be done on German Federal Archives - [Link](https://www.bundesarchiv.de/cocoon/barch/0000/k/index.html)

## CSV Data

Though storing the data in a CSV file is generally a good idea, we are making use of azure queue storage predominantly to store and fetch the information required. However, for sharing the data frame we are using csv file format to store and retrieve to perform NLP based search on it.

## JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

## JSON Data

Some of the data is extracted and stored in the JSON format, which is used by the python code for fast and easier processing being light weight.

**Sample json stored in the azure queue**

{

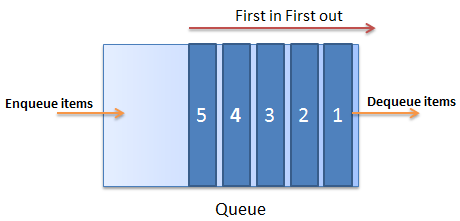
"Url":"https://www.bundesarchiv.de/cocoon/barch/0000/k/k1980k/index.html",

"TextInfo":"Die Kabinettsprotokolle der Bundesregierung 1980"

}

## Queue

Queue: A queue is a linear data structure in which elements can be inserted only from one side of the list called rear, and the elements can be deleted only from the other side called the front. The queue data structure follows the FIFO (First In First Out) principle, i.e. the element inserted at first in the list, is the first element to be removed from the list. The insertion of an element in a queue is called an enqueue operation and the deletion of an element is called a dequeue operation. In queue we always maintain two pointers, one pointing to the element which was inserted at the first and still present in the list with the front pointer and the second pointer pointing to the element inserted at the last with the rear pointer.

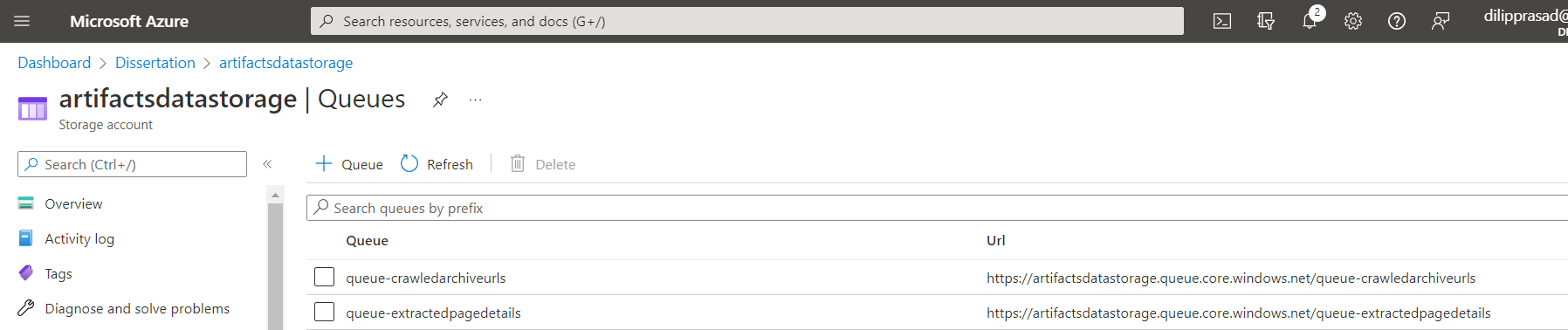


# Overall Code explained

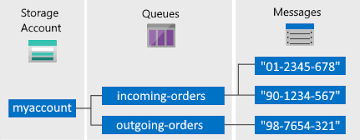
## Queue Storage

Below Snapshot displays the 2 different queue storage mechanism used from Microsoft’s Azure portal. These queue mechanism enables us to store and fetch the data. After fetching the messages, it could be either deleted from queue or left as it is for future processing. It is to be noted that, although it’s called as a storage – the data expires after a specific point of time.

*Snapshot of queues used in azure:*



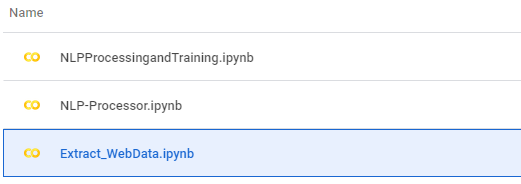
*Sample azure queue flow:*



## Module overview

We have 3 python files to handle the core NLP processes

1. Limited Web Crawling
2. Extract Web Data
3. NLP Process and Training
4. Perform Search





## Limited Web Crawling file

* With the give initial link for the German federal archive repository/ URL, we are using a recursive looping mechanism to fill all the related links that we could find. If the related link is found to be relevant, then we are going to use that for further Crawling.
* However, unlike traditional web search we do not want to crawl information other than the archive itself.
* Although we could store the crawled links in local csv, for a scalable and enterprise level app we are making use of Azure Queue storage [3].

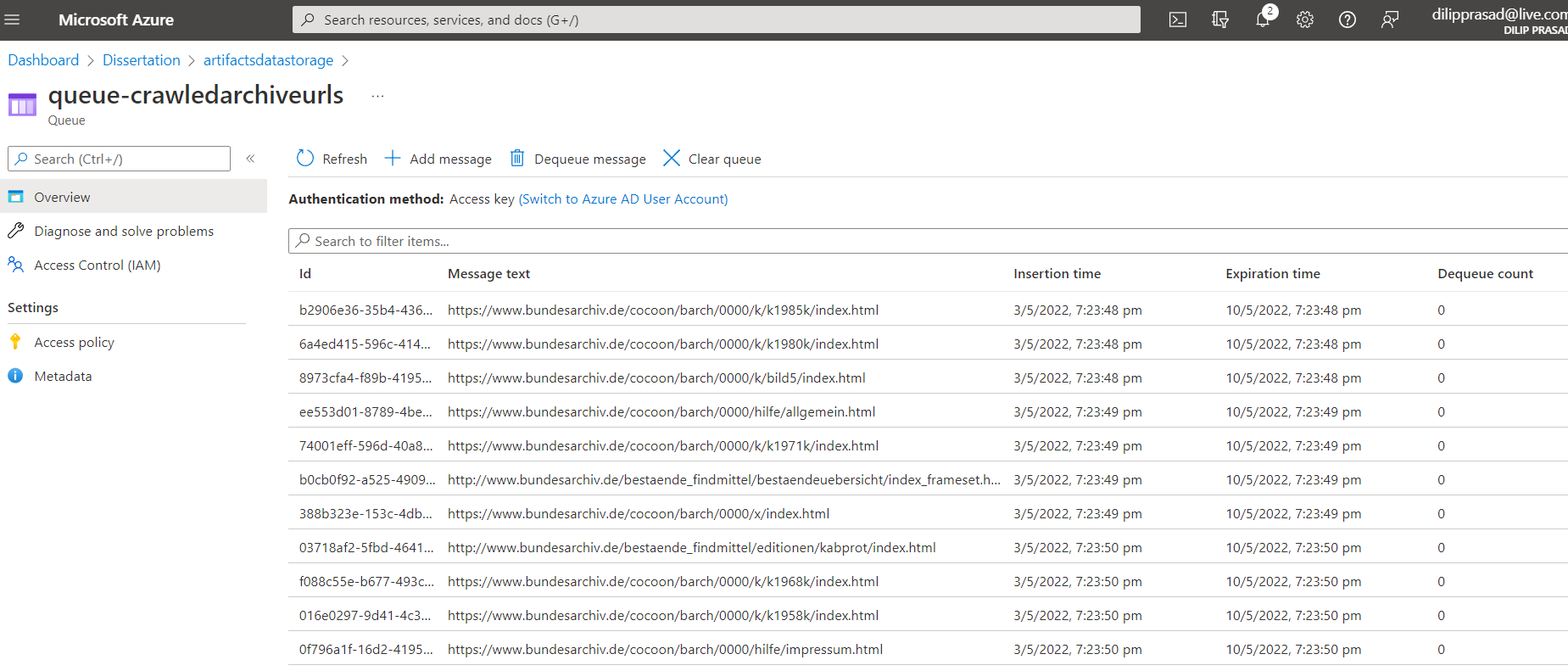
We are making use of the python’s Multi processor method to crawl in parallel making use of the multiple cores.

* We are making use of the python library beautiful soap to parse the html and find relevant links.
* Found url are queued to the azure queue name “**queue-crawledarchiveurls**”

*Source could be found at the below URL*

**Github Link:** <https://github.com/dilipprasad/Dissertation/blob/main/Limited_WebCrawling.ipynb>

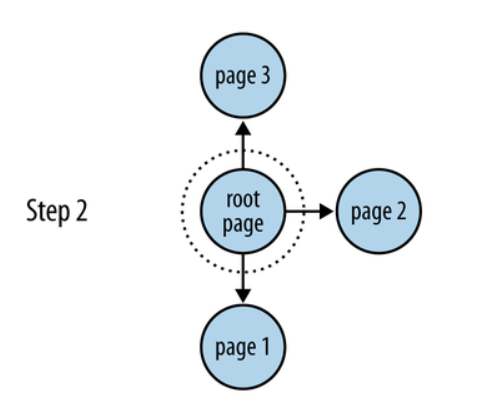
*Sample of the stored data of the queue “queue-crawledarchiveurls” under Message Text column*

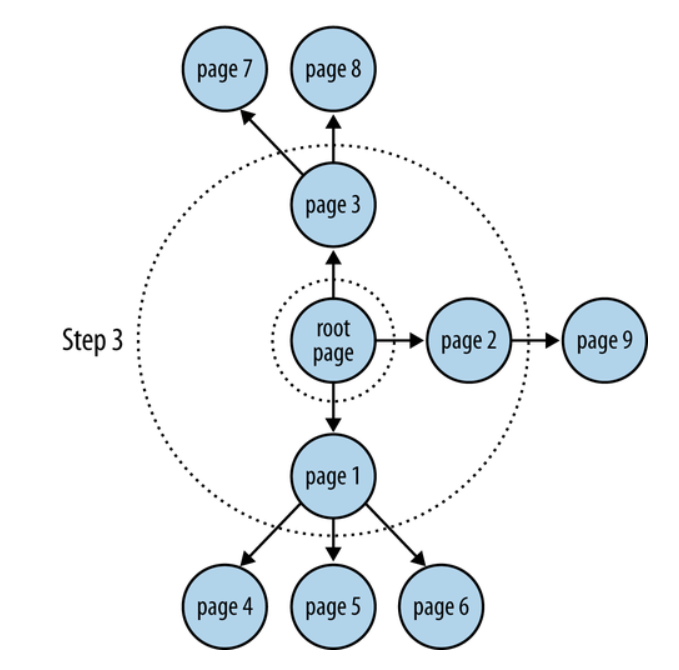


Starting point: <https://www.bundesarchiv.de//cocoon/barch/0000/k/index.html>

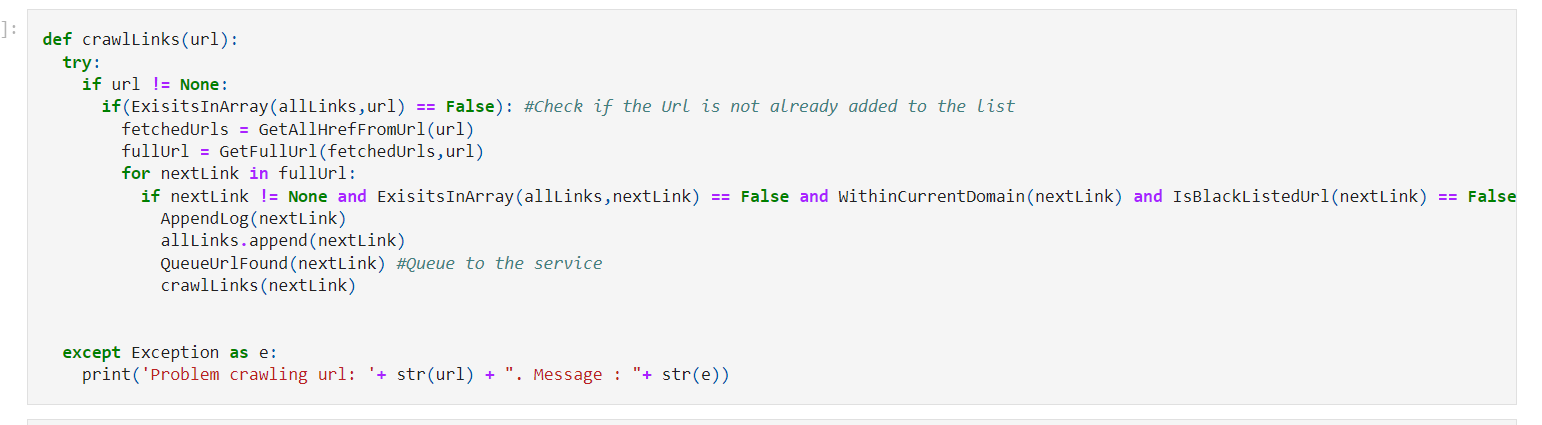
**Crawling Graph**







**Crawling code to find Url:**

****



## Extract Web Data file

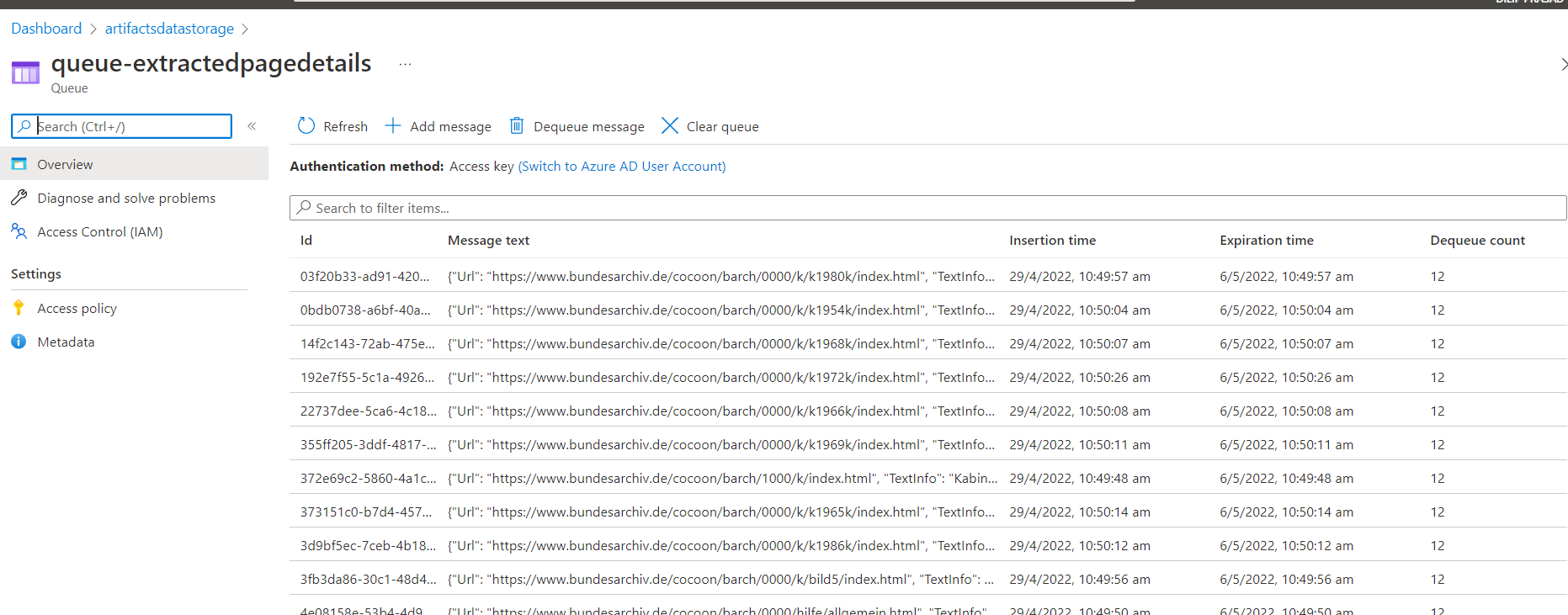
* As we have the list of URLS being queued to azure queue - “**queue-crawledarchiveurls**”, we could process those links
* Extract web data python module does fetch the URL from queue and extracts the required text from the web page
* Instead of grabbing the entire html we are fetching only the text
* With the retrieved text we are creating a json with the URL and its fetched text and send it to the final queue “**queue-extractedpagedetails**”
* Sample json data as below

{

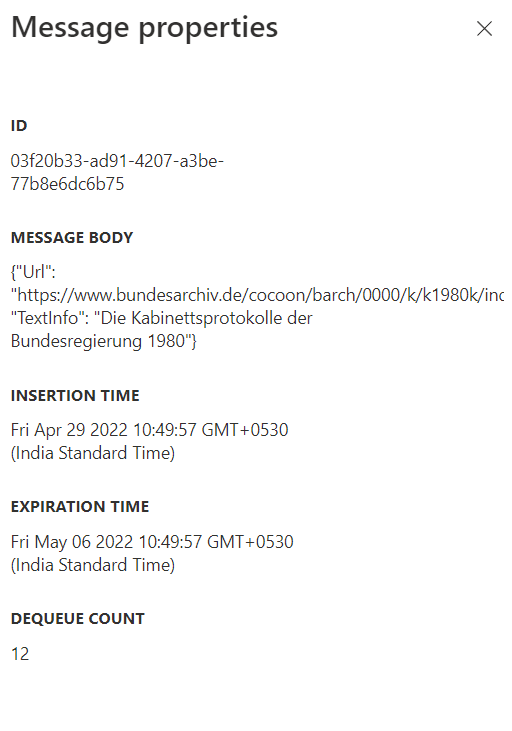
"Url":"https://www.bundesarchiv.de/cocoon/barch/0000/k/k1980k/index.html",

"TextInfo":"Die Kabinettsprotokolle der Bundesregierung 1980"

}



Sample message format:



**Github Url**: <https://github.com/dilipprasad/Dissertation/blob/main/Extract_WebData.ipynb>

## NLP Processing and training

* Finally, connect to the azure queue “**queue-extractedpagedetails**” and create a data frame of URL and text information
* We also cleanup the data and apply necessary algorithms

**Github link:** [**https://github.com/dilipprasad/Dissertation/blob/main/NLPProcessingandTraining.ipynb**](https://github.com/dilipprasad/Dissertation/blob/main/NLPProcessingandTraining.ipynb)

1. Perform search

This file specifically helps in loading the CSV to a panda’s data frame and also load the saved model created for our purpose.

**Github Link**: <https://github.com/dilipprasad/Dissertation/blob/main/PerformSearch.ipynb>

# NLP phases

There are 5 phases on any given NLP projects and the same will be performed here

**1. Lexical Analysis and Morphological**

The first phase of NLP is the Lexical Analysis. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. It divides the whole text into paragraphs, sentences, and words.

**2. Syntactic Analysis (Parsing)**

Syntactic Analysis is used to check grammar, word arrangements, and shows the relationship among the words.

**Example:** Agra goes to the Poonam

In the real world, Agra goes to the Poonam, does not make any sense, so this sentence is rejected by the Syntactic analyzer.

**3. Semantic Analysis**

Semantic analysis is concerned with the meaning representation. It mainly focuses on the literal meaning of words, phrases, and sentences.

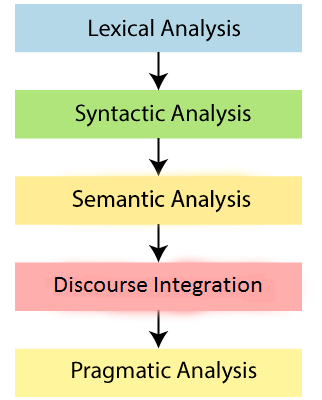
**4. Discourse Integration**

Discourse Integration depends upon the sentences that proceeds it and also invokes the meaning of the sentences that follow it.

**5. Pragmatic Analysis**

Pragmatic is the fifth and last phase of NLP. It helps you to discover the intended effect by applying a set of rules that characterize cooperative dialogues.

**For Example:** "Open the door" is interpreted as a request instead of an order.

[7]

# Data preprocessing

Data preprocessing is one of the critical steps in any machine learning project. It includes cleaning and formatting the data before feeding into a machine learning algorithm. For NLP, the preprocessing steps are comprised of the following tasks:

1. Tokenizing the string
2. Lowercasing
3. Removing stop words and punctuation
4. Stemming
5. Or Lemmatization

## Tokenization

Tokenization is the process of extracting all the words in a given sentence.



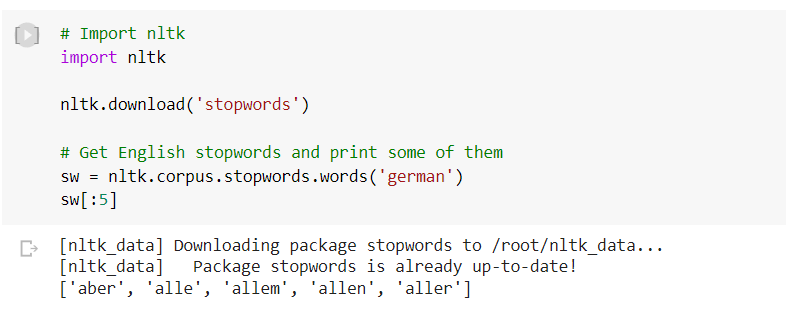
For the given sentence “Allgemeine HinweiseHier finden Sie Informationen und Hilfe zur Benutzung der Internetpräsentation der Kabinettsprotokolle sowie das Impressum.” We got the list of words

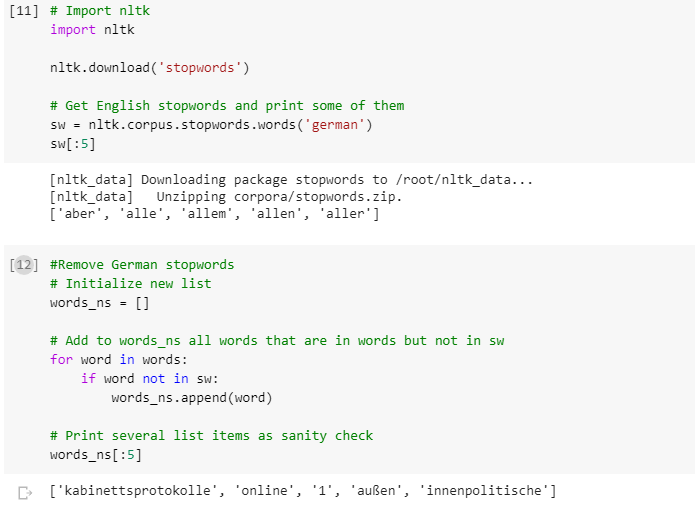
['Allgemeine', 'HinweiseHier', 'finden', 'Sie', 'Informationen', 'und', 'Hilfe', 'zur', 'Benutzung',

'der', 'Internetpräsentation', 'der', 'Kabinettsprotokolle', 'sowie', 'das', 'Impressum', '.']

## Stop words

Use the NLTK libraries pre-built stop words to ignore





For any given sentence, we will remove the stop words as its not required

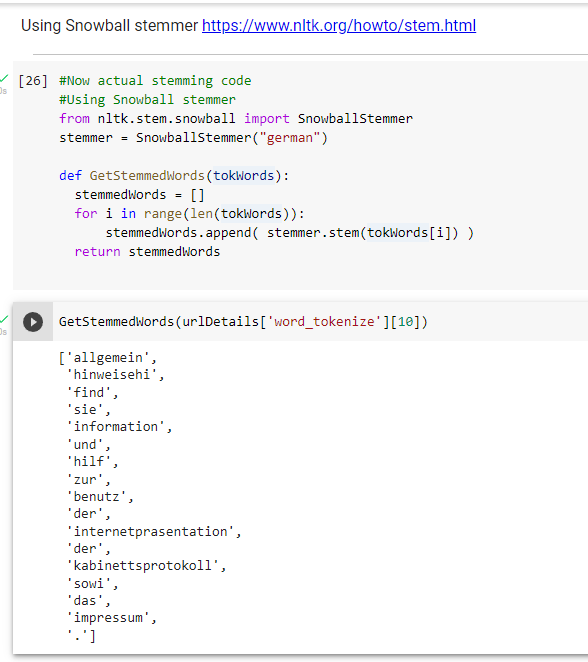
## Stemming

Stemming is a natural language processing technique that lowers inflection in words to their root forms, hence aiding in the preprocessing of text, words, and documents for text normalization. we employ stemming to reduce words to their basic form or stem, which may or may not be a legitimate word in the language.

For instance, the stem of these three words, connections, connected, connects, is “connect”. [5]

Out of Porter stemmer and Lancaster stemmer, Porter stemmer which primarily helps in search in alternative forms of words. Porter stemmer correctly handles the word lying (mapping it to lie), while the Lancaster stemmer does not.

We will more precisely use the latest version of the Porter stemmer known as “Snowball stemmer” which supports German and multiple other languages.



*Below snapshot is the google translation of the text from German to English- original vs stemmed text*

Text

Description automatically generated with medium confidence

## Lemmatization

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meanings to one word.[6]

* Used in comprehensive retrieval systems like search engines.

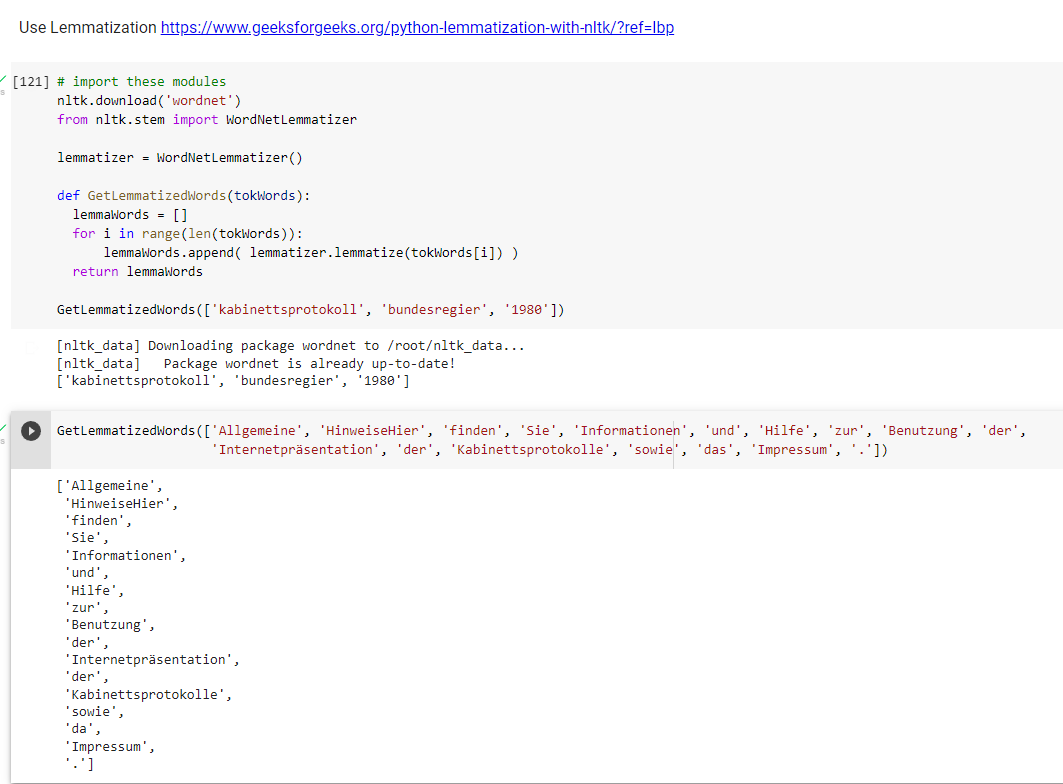
Used in compact indexing

We would be performing lemmatization as it does not convert lying to lie but converts women to woman. This is useful only if we are compiling a vocabulary of texts and require a valid list of lemmas.

*Below snapshot is the google translation of the text from German to English- original vs lemmatized text.*

A picture containing calendar

Description automatically generated

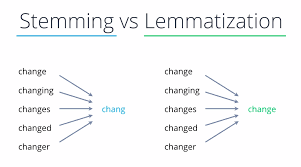


## Stemming vs Lemmatization

[5] The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is => be

car, cars, car's, cars' => car



# Additional cleanup

## Lowercasing

Converting upper cases to all lower case

## Removing stop words and punctuation

Remove all the unwanted special characters

## Data Cleaning - Multi step process in single method

The below code helps us to perform the necessary clean up, retaining only the vital information required



As we notice, the text “Die Kabinettsprotokolle der Bundesregierung 1980 !.” is converted to python list “ ['kabinettsprotokolle', 'bundesregierung', '1980'] ”

# Creating a Corpus of Data

With the multiple conversational / legislative transcripts we could create our own corpus using NLTK (Natural Language toolkit)

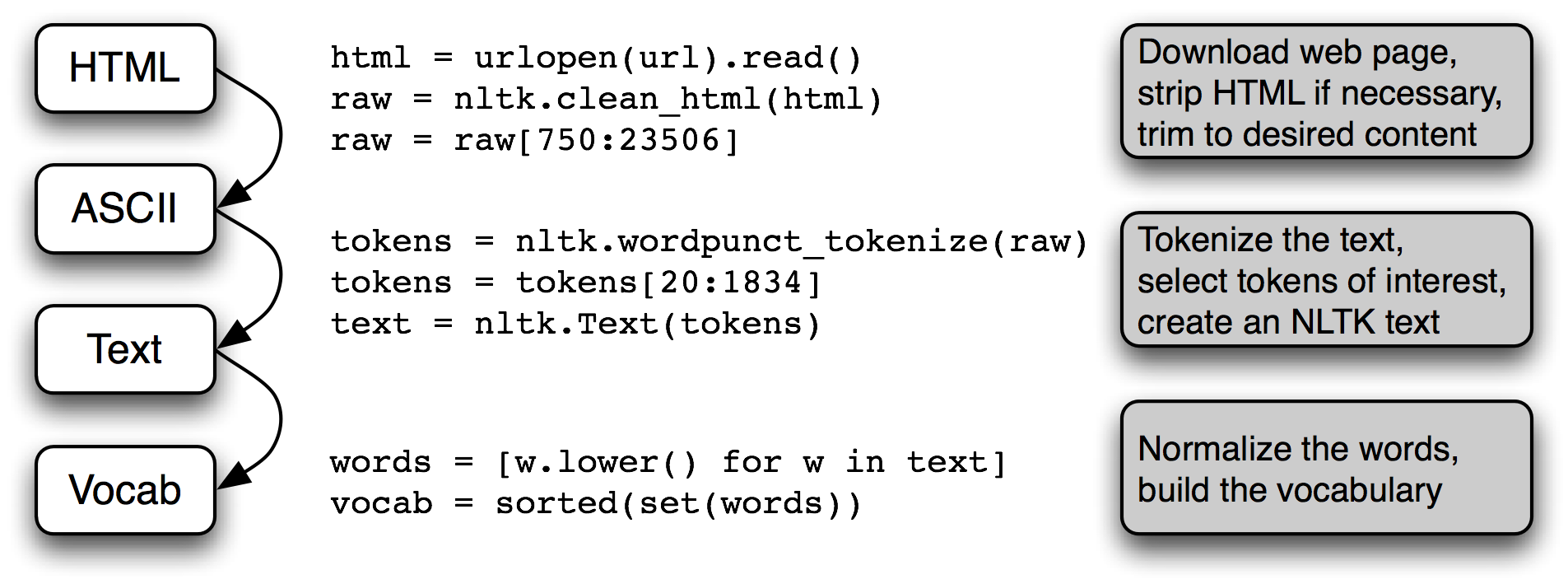
Which will also help us project the Frequency distribution of words, to facilitate the validation of the data and we must find if its Balanced.

# NLP Pipeline

NLP pipeline generally means that the multiple data preprocessing techniques are applied in a single step, which often gets reused multiple times. There are multiple steps present in an NLP Pipeline

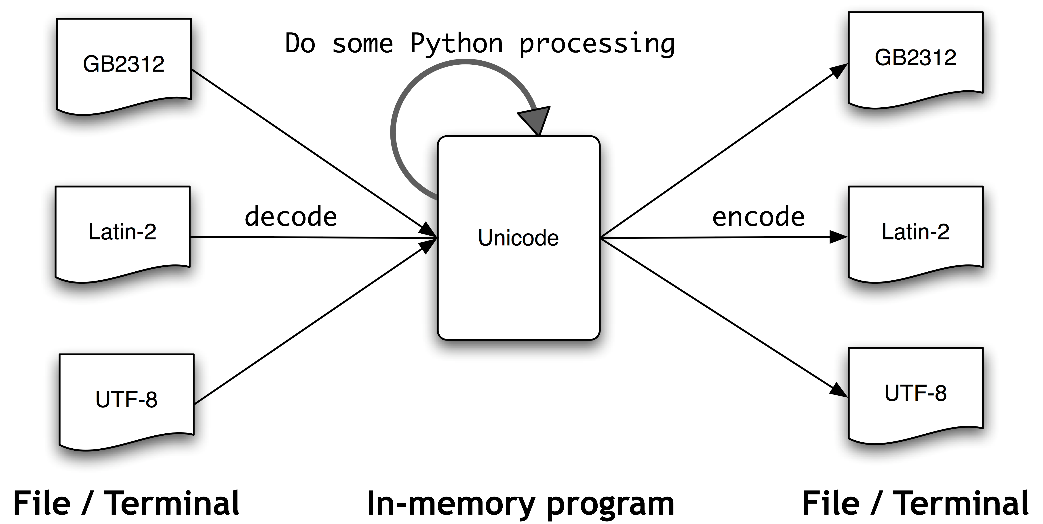
* Grab the HTML contents of the given URL
* then strip of the html tags /markup
* Get the data in text
* Build a Vocabulary with the words

Below is a typical example for a Webpage in English



However, ASCII is normally supported only for English, since we are using German, we need to find different encoding standards. So, we could use Unicode in short UTF standards. Each character is assigned a number, called a code point. In Python, code points are written in the form \uXXXX, where XXXX is the number in 4-digit hexadecimal form.

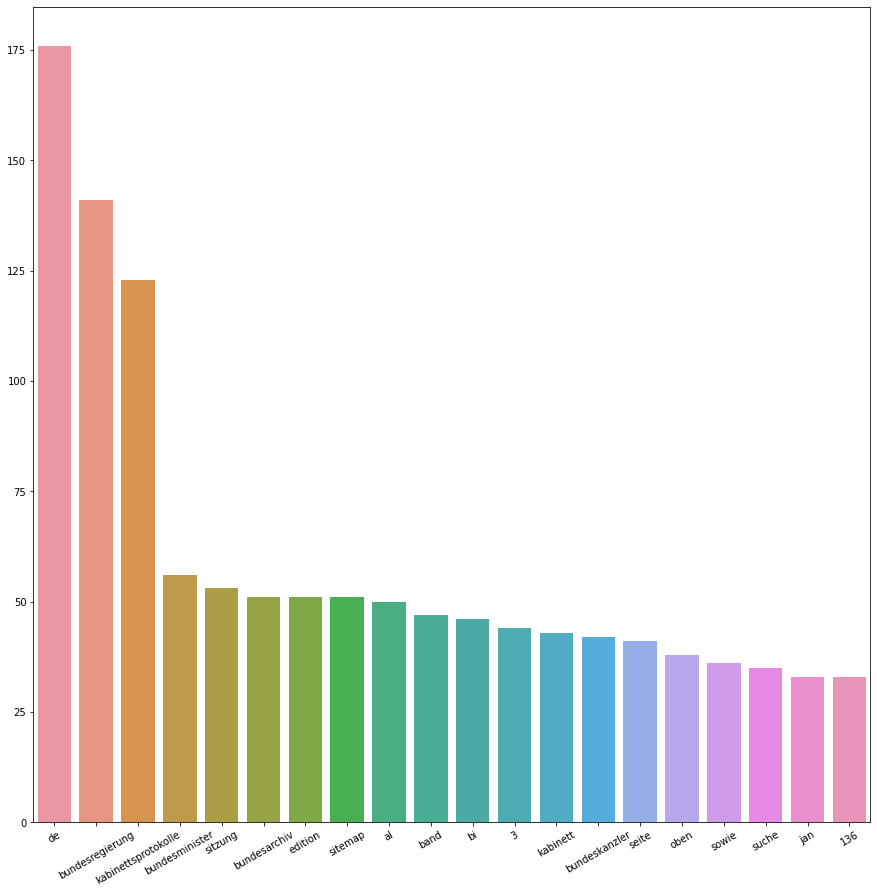
Since UTF supports only a limited set of characters, we could use UTF-8 which supports multiple bytes that can represent full range of Unicode characters.



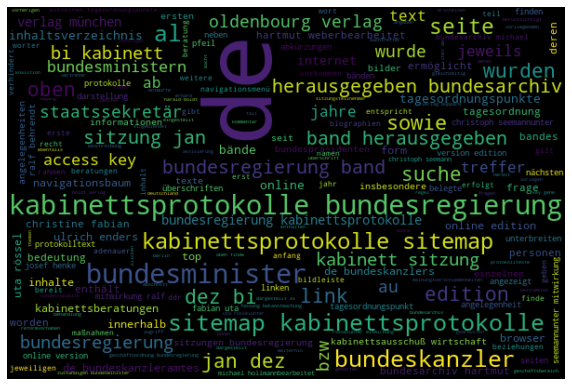
# Exploratory Data Analysis (Visualization)

## Frequency Distribution

Let’s look at a distribution of work frequency for a subset of data



## Word cloud

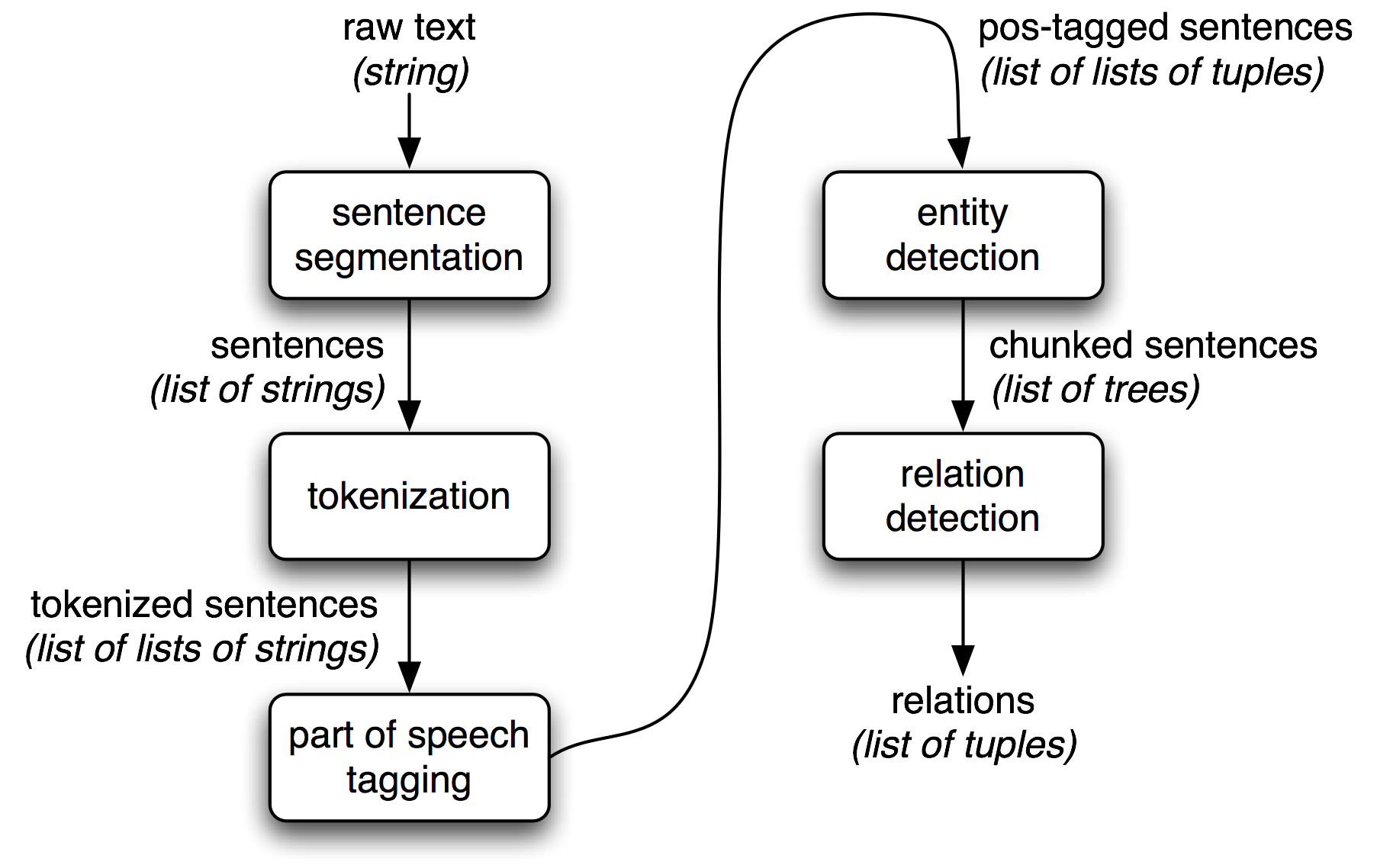


# Gender Recognition

Male and female names have some distinctive characteristics. Names ending in a, e and I are likely to be female, while names ending in k, o, r, s and t are likely to be male. As a part of this we will create a classifier to model these differences more precisely.

# Information Extraction Architecture

We will process using procedures discussed as above, raw text of the document is split into sentences using a sentence segmented, and each sentence is further subdivided into words using a tokenizer. Next, each sentence is tagged with part-of-speech tags, which will prove very helpful in the next steps



# Named Entity Recognition

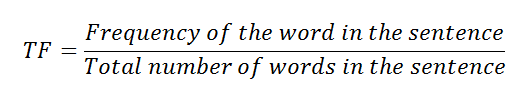
The goal of a named entity recognition (NER) system is to identify all textual mentions of the named entities. This can be broken down into two sub-tasks: identifying the boundaries of the NE, and identifying its type

Named entity recognition is a task that is well-suited to the type of classifier-based approach that we saw for noun phrase chunking. We will build a tagger that labels each word in a sentence using the IOB format, where chunks are labeled by their appropriate type.

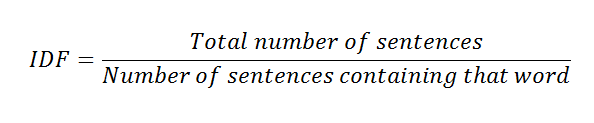
# TF-IDF

Term Frequency- Inverse Document frequency is a measure used in information retrieval. This score is used to determine importance of a term. When it comes to search engine TF-IDF plays a vital role to get a nearest match or a meaningful match (based on synonym)

Term Frequency – How frequent a word is occurring. If a word appears multiple times, then we could consider it has higher importance /score. This approach does not work always, as sometimes the words may not be frequent or many words could be frequent.

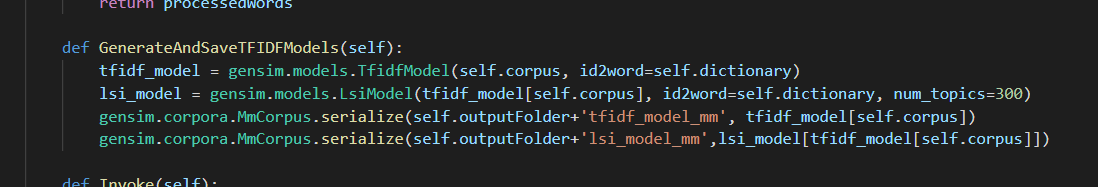


IDF – Inverse Document Frequency- measures the informativeness of the term t



# Model Creation

Finally, after the model is trained based on the collected data, we will readily save the model physical and ship it. This enables us to consume the model and use it to get relevant occurrence in the search results.

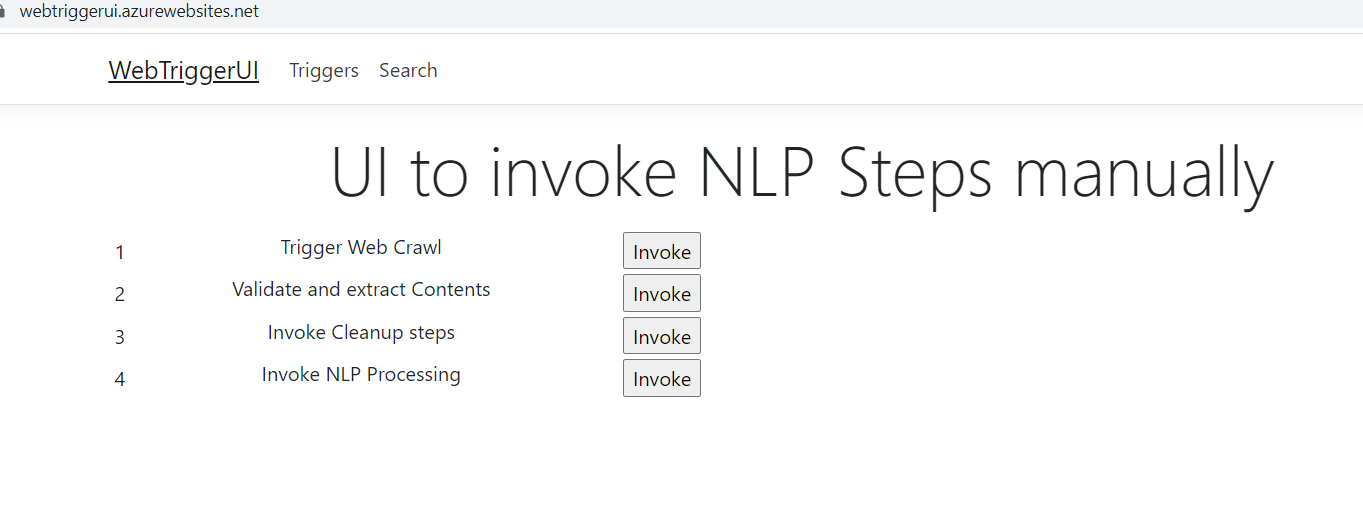


# WEB UI

We have created an UI which invokes the python scripts to trigger the process. We could individually process one step after the other.

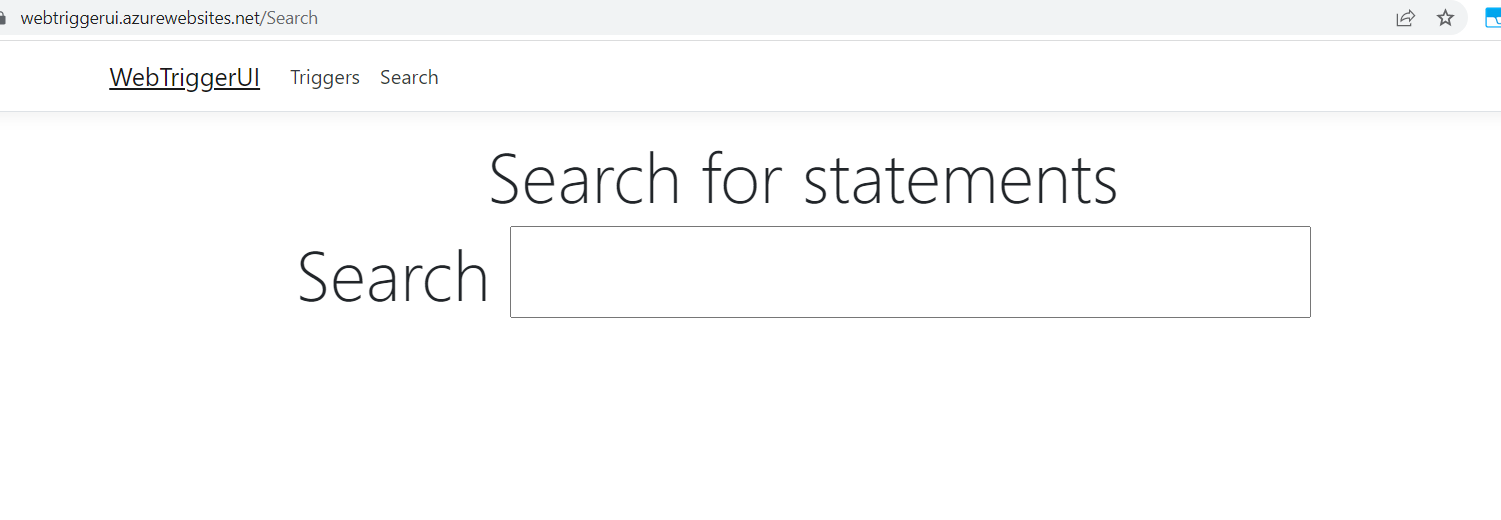
1. Triggers Web Crawling
2. Extracts and validates the web page contents
3. Invokes cleanup steps
4. Invokes NLP processing steps

GitHub link: <https://github.com/dilipprasad/WebTriggerUI>



# Search

Once the above-mentioned triggered process is completed, we could perform search and find results of our query



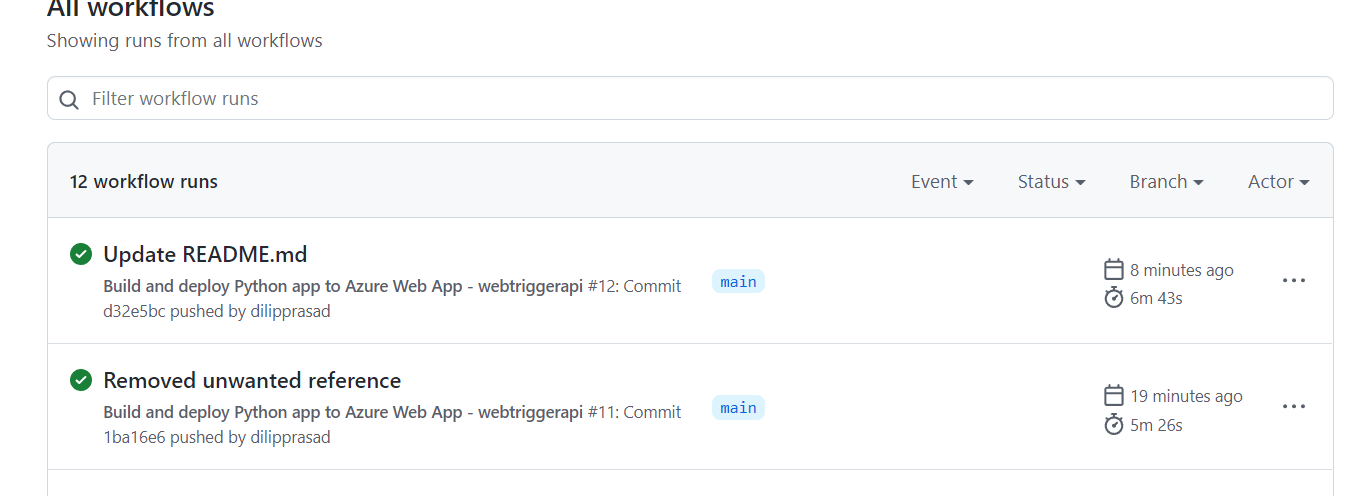
# API

The above search functionality is power by the API, though we have been going through the notebooks, for the sake of API we have moved all those to python files accessed by Fast API. Fast API is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

## API Repository

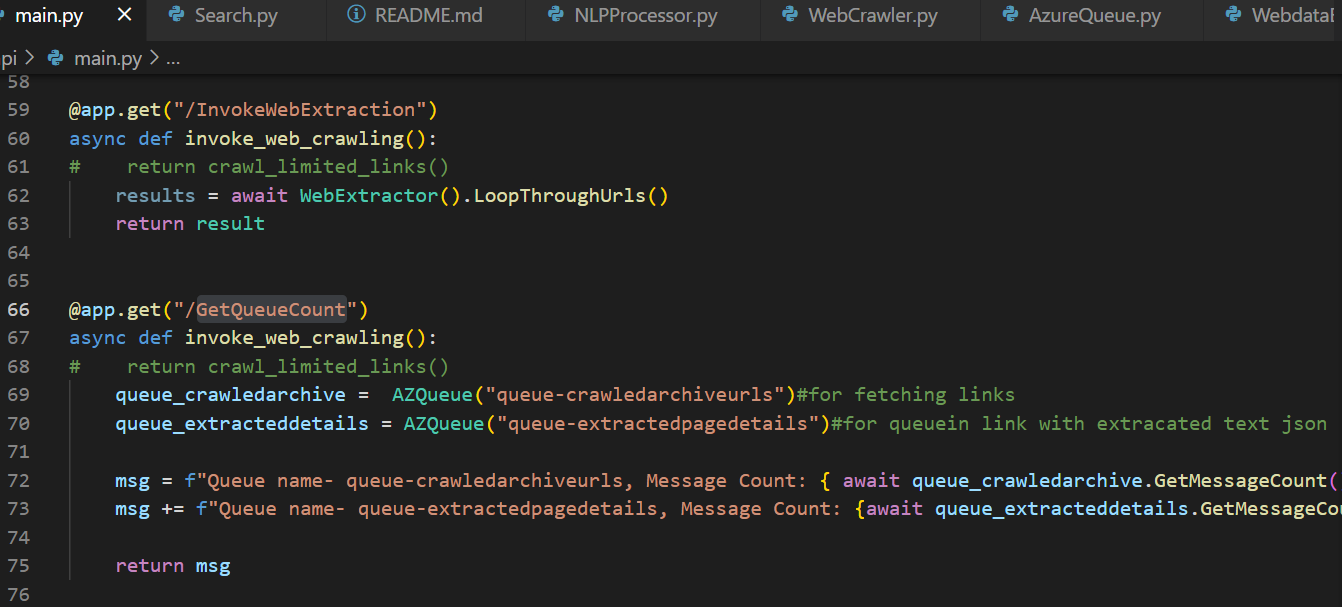
<https://github.com/dilipprasad/WebTriggerAPI>

Enabled CI/CD for each check in- allowing us to deploy the latest code when available

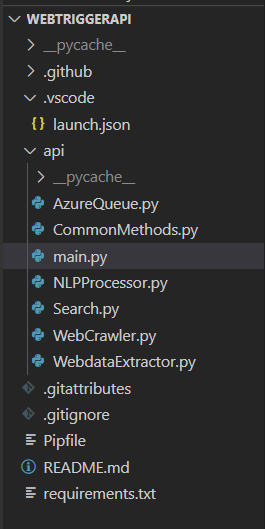


Some of the API Functionalities

1. Ping – Check status
2. Invoke Crawling – Long running crawl process
3. Invoke Web Extraction – Extract the web page text
4. Get Queue Count- For investigating the total queue count
5. Ge All Queue Messages – To Extract messages from azure queue
6. Process NLP – Apply NLP algorithms and save model
7. Search – Perform search and return results

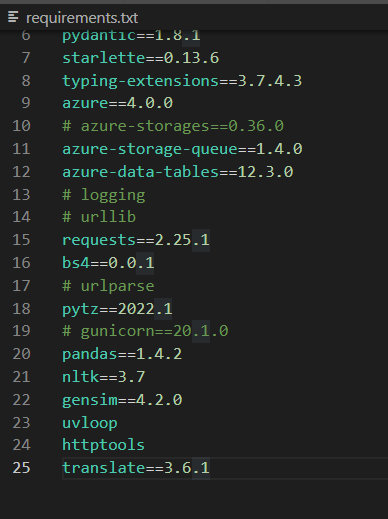


## Project Structure



## Required Libraries

The list of libraries such as pandas, numpy are listed in this file, which gets installed automatically in azure.



# Conclusion

We can crawl a web page in German language and use various data cleanup steps to extract the pivotal data then use NLP techniques to train. Finally, all these steps have paved way to help us search for results in an UI which fetches the relevant details (*which does not have to be a word-by-word match*)

Scope for improvement

* Use no server methodologies by making use of azure on demand or AWS functionless offerings
* Communicate between serverless functions to create a cascading experience
* Create a MLOPS pipeline to continuously train and deploy model for enhancements

# References

1. <https://en.wikipedia.org/wiki/National_archives>
2. <https://en.wikipedia.org/wiki/Records_management>
3. <https://docs.microsoft.com/en-us/azure/storage/queues/storage-queues-introduction>
4. <https://en.wikipedia.org/wiki/Web_scraping>
5. <https://www.analyticsvidhya.com/blog/2021/11/an-introduction-to-stemming-in-natural-language-processing/>
6. <https://www.geeksforgeeks.org/python-lemmatization-with-nltk/?ref=lbp>
7. <https://www.javatpoint.com/nlp>