Functional Document

A Web Scraping Tool to get articles data from web

Table of Contents

Introduction:

This is a python-based web scraping tool which fetches the information of an article or to get a list of articles based on the searched keywords. Tool lookup into the various search engines which are repositories for academic papers, research papers, journals etc.

Web scraping is the process of processing a web document and extracting information out of it in an automated manner.

Purpose:

It is an academic research tool for educational purposes.

Attributes:

It provides details of below listed features:

1. DOI
2. Title
3. URL
4. Authors
5. Publication Name
6. ISSN
7. Cited count
8. Affiliation
9. Type
10. Published date
11. Abstract

Value of these attributes depends on the result object provided by the search engine or the api used.

Search Engines Used:

We have used most widely and largest search engines for our total. These are:

1. **PubMed**- Primarily the MEDLINE database of references and abstracts on life sciences and biomedical topics

2. **PLOS One**- Peer-reviewed open access scientific journal published by the Public Library of Science
3. Academia- commercial social networking website for academics

4. **Google Scholar**- web search engine that indexes the full text or metadata of scholarly literature

5. **Microsoft Academic**- web search engine for academic publications and literature, developed by Microsoft Research

6. **ScienceDirect**- provides access to a large bibliographic database of scientific and medical publications of the Dutch publisher Elsevier.

7. **Elsevier SCOPUS**- Elsevier's abstract and citation database

8. **CORE**- academic search engine dedicated to open access research papers

9. **Springer**- Repository for books, e-books and peer-reviewed journals in science, humanities, technical and medical publishing

```
### uncomment the search engine baesd your requiremnt
 search_pubMed(query)
 search_PlosOne(query)
 search_academia(query)
 search_msAcademic(query)
 search_googleScholar(query)
 search_sciDirect(query)
 search_scopus(query)
 search_core(query)
 search_springer(query)
```

Functionality:

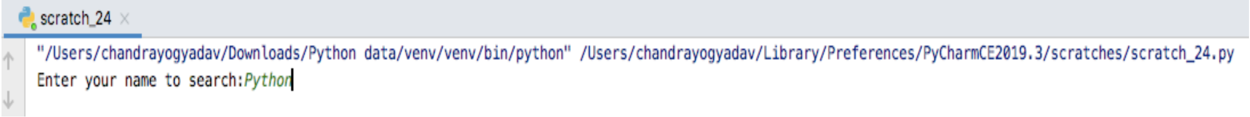
The tool has been scripted in python language and using its libraries.

Libraires used:

- Pandas
- Numpy
- requests,
- json,
- bs4,
- openpyxl,
- urllib3
- re

Input Operations-

- Tool can take any string input.
- Console will require user input once application started.
- Input can be:
  - Article keyword
  - Name
  - Author Name
  - Abstract etc.



```
"/Users/chandrayogyadav/Downloads/Python data/venv/venv/bin/python" /Users/chandrayogyadav/Library/Preferences/PyCharmCE2019.3/scratches/scratch_24.py
Enter your name to search:Python
```

Output Format:

- We have used dictionary to initially hold the response object

```
# append dict object data
data.append(resp_obj)
```

- Then converted final output into JSON format

```
# print the dict output
print("JOSN format:", data)
```

- Also, to further analysis we have transformed the output into DataFrames by segregating the output attributes into columns and values into rows
- After converting the output into dataframes, we are also saving it into excel format

```
#####-----creating final output------#####
# drop nested columns and keep 1st attribute
df.drop(["entities.items"], axis=1, inplace=True)
```

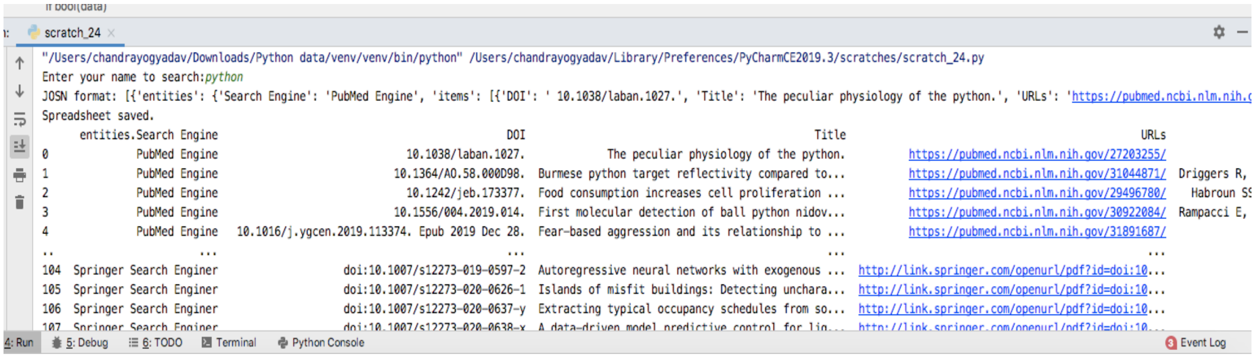
```
# create required temp objets
d1 = pd.DataFrame([])
result = pd.DataFrame([])

# split nested attributes into separate columns and stored output in a temp
object d1
i = 0
for i in range(0, len(data)):
    d = pd.json_normalize(data[i]['entities']['items'])
    d1 = d1.append(d, True)

    # concatenate both dataframes into one
    result = pd.concat([df, d1], axis=1)

# save final output to csv
result.to_excel('search_results.xlsx', index=False)
print('Spreadsheet saved.')
print(result)
```

Sample Output:



```
"/Users/chandrayogyadav/Downloads/Python data/venv/venv/bin/python" /Users/chandrayogyadav/Library/Preferences/PyCharmCE2019.3/scratches/scratch_24.py
Enter your name to search:python
JOSN format: [{'entities': {'Search Engine': 'PubMed Engine', 'items': [{'DOI': ' 10.1038/laban.1027.', 'Title': 'The peculiar physiology of the python.', 'URLs': 'https://pubmed.ncbi.nlm.nih.g
Spreadsheet saved.
      entities.Search Engine                             DOI                                          Title                                      URLs
0              PubMed Engine              10.1038/laban.1027.            The peculiar physiology of the python.        https://pubmed.ncbi.nlm.nih.gov/27203255/
1              PubMed Engine            10.1364/AO.58.000D98.  Burmese python target reflectivity compared to...        https://pubmed.ncbi.nlm.nih.gov/31044871/  Driggers R,
2              PubMed Engine              10.1242/jeb.173377.  Food consumption increases cell proliferation ...        https://pubmed.ncbi.nlm.nih.gov/29496780/    Habroun SS
3              PubMed Engine            10.1556/004.2019.014.  First molecular detection of ball python nidov...        https://pubmed.ncbi.nlm.nih.gov/30922084/  Rampacci E,
4              PubMed Engine  10.1016/j.ygcen.2019.113374. Epub 2019 Dec 28.  Fear-based aggression and its relationship to ...        https://pubmed.ncbi.nlm.nih.gov/31891687/
..                       ...                             ...                                            ...                                        ...
104  Springer Search Enginer       doi:10.1007/s12273-019-0597-2  Autoregressive neural networks with exogenous ...  http://link.springer.com/openurl/pdf?id=doi:10...
105  Springer Search Enginer       doi:10.1007/s12273-020-0626-1  Islands of misfit buildings: Detecting unchara...  http://link.springer.com/openurl/pdf?id=doi:10...
106  Springer Search Enginer       doi:10.1007/s12273-020-0637-y  Extracting typical occupancy schedules from so...  http://link.springer.com/openurl/pdf?id=doi:10...
107  Springer Search Enginer       doi:10.1007/s12273-020-0638-x  A data-driven model predictive control for lig...  http://link.springer.com/openurl/pdf?id=doi:10...
```

Methodology:

We have used 2 approaches to scrap the data from a search engine:

a) Web scrap the html page
b) Use the public API of the search engine

1. Web Scrap the HTML Page:

This methodology applied because either the engine doesn't provide any public api or the api doesn't have the required attributes.

Below are the search engines used under this approach:

1. Google Scholar

2. PubMed
3. Academia

Usage:

For example, in google scholar we have did below steps-

- Created a dictionary object to hold the output data

```
data = []
```

- Then fetched the html page output using request method of python and saved it response object.

```
response = requests.get(url, headers=headers, timeout=30)
```

- Parsed the response object into soup object using Beautiful Soup library from python. *(Beautiful Soup is a Python library for getting data out of HTML, XML, and other markup languages)*

```
soup = BeautifulSoup(response.content, 'lxml')
```

- Implemented the For loop to iterated required tags and save the text into the dictionary object data[].

```
for item in soup.select('[data-lid]'):

resp_obj = {"entities":    }


data.append(resp_obj)
```

2. API Call

We have consumed the public apis of the following search engines to generate the results:

1. PLOS One
2. Microsoft Academy
3. ScienceDirect
4. Elsevier SCOPUS
5. CORE
6. Springer

Usage:

We will show one example of CORE search engine-

- Get the api key by registering on the official site of the ScienceDirect.

```
# CORE API key
core_api = 'TSYp9xWZK7dm3XBz6r8RnalOGyAvjEFg'
```

- Get the URL and parameters details from the exmaples on the site

- Created URL by addng the api key and other pararemters

```
url = 'https://core.ac.uk:443/api-v2/search/' + query +
'?page=1&pageSize=10&apiKey=' + core_api
```

- Created response object by all the api using requests method and parse into soup

```
response = requests.get(url, headers={'User-agent': 'your bot 0.1'})
soup = BeautifulSoup(response.content, 'lxml')
```

- Converted the soup object into JSON format to iterate the keys and values

```
obj = json.loads(soup.text)
```

- Applied For loop to get the values on json object

```
for item in obj['data']:
    try:
        resp_obj = {"entities": {"Search Engine": "CORE Search Engine",
                                 "items": [{"DOI": item['_source']['doi'],
                                            "Title": item['_source']['title'],
                                            "URLs": item['_source']['urls'],
                                            "Authors":
item['_source']['authors'],
                                            "Publication Name":
item['_source']['publisher'],
                                            "ISSN": item['_source']['issn'],
                                            "Cited count":
item['_source']['citationCount'],
                                            "Affiliation": ['No Information'],
                                            "Type": item['_type'],
                                            # "Keywords": item['topics'],
                                            "Published Date":
item['_source']['datePublished'],
                                            "Abstract":
item['_source']['description']
                                            }]}}

        # append dict object data
        data.append(resp_obj)
    except Exception as e:  # raise e
        pass
        # print('error core:', e)
```

List of APIs:

PLOS One

Site- http://api.plos.org/
Source- http://api.plos.org/solr/examples/

Example-
http://api.plos.org/search?q=title:%22Drosophila%22%20and%20body:%22RNA%22&fl=id,abstract

Microsoft Academy

Register and Get API key- https://msr-apis.portal.azure-api.net/
APIs- https://msr-apis.portal.azure-api.net/docs/services/academic-search-api/operations/565d753be597ed16ac3ffc03/console
API attributes- https://docs.microsoft.com/en-us/academic-services/project-academic-knowledge/reference-evaluate-method
Example-
https://api.labs.cognitive.microsoft.com/academic/v1.0/evaluate?expr={expr}&model=latest&count=10&offset=0&attributes=Id

ScienceDirect

Register and Get API key- https://dev.elsevier.com/
API- https://dev.elsevier.com/sciencedirect.html
Parameters- https://dev.elsevier.com/documentation/SCOPUSSearchAPI.wadl
Live APIs-
https://dev.elsevier.com/sciencedirect.html#!/ScienceDirect_Search_V2/ScienceDirectSearchV2

Example-
https://api.elsevier.com/content/search/sciencedirect?query=gene&apiKey=7f59af901d2d86f78a1fd60c1bf9426a

Elsevier SCOPUS

Register and Get API key- https://dev.elsevier.com/
APIs- https://dev.elsevier.com/scopus.html
Parameters- https://dev.elsevier.com/documentation/SCOPUSSearchAPI.wadl
Live APIs- https://dev.elsevier.com/scopus.html#!/Scopus_Search/ScopusSearch

Example- https://api.elsevier.com/content/search/scopus?query=all(gene)&apiKey=7f59af901d2d86f78a1fd60c1bf9426a

CORE

Register and Get API Key- https://core.ac.uk/services/api/
API Parameters- https://www.elastic.co/guide/en/elasticsearch/reference/1.4/query-dsl-query-string-query.html#query-string-syntax
Live APIs- https://core.ac.uk/searchAssets/docs/

Example- https://core.ac.uk:443/api-v2/search/python?page=1&pageSize=10&apiKey=TSYp9xWZK7dm3XBz6r8RnalOGyAvjEFg

Springer

Register and Get API Key- https://dev.springernature.com/
API Parameters- https://dev.springernature.com/querystring-parameters
Live APIs- https://dev.springernature.com/example-metadata-response

Example- http://api.springernature.com/metadata/json?q=name:hughes year:2014&s=1&p=5&api_key=yourKeyHere