Course Project:
Animal-related articles Search System

Present to
Asst. Prof. Dr. Charnyote Pluempitiwiriyawej

By

Mr. Ekaphat	Seamthong	6188039
Mr. Ariya	Sontrapornpol	6188041
Mr. Ekkawit	Sangreungkit	6188122

ITCS414 – Information Retrieval and Storage
Faculty of Information and Communication Technology
Mahidol University 2020

Introduction

Animals have been walking among us since we couldn't ever remember. For some people, animals or pets are a part of their life. For some people, they work with animals daily. But we have never known enough about them, ever. There are over 8.7 million species of animals living in the world on the ground, under the water, or even in the sky. It's impossible for us to learn all about them and know all the species. But we can catch up with their move and how they affect our life. That is why we created this animal-related article search system. We created for those people who want to or catch up with the latest news about animals and people who want to study or know more about animals. We retrieved the articles from www.livescience.com which we considered as a very high reliable source for many reasons. We created a search engine with Elasticsearch which is a really powerful open-source search engine and implemented it on the web application with the Django framework which is one of the best web frameworks for Python.

Problems we are trying to solve

The problem we are trying to solve is many websites that integrate the animal article are having the non-animal related articles too. That could make the search engine not efficient and show a lot of animal-unrelated articles or information. Moreover, some websites show the information from unreliable sources and sometimes those articles are outdated. Therefore, we created the search engine that works specifically for animal-related articles which are published by reliable websites and authors. We retrieved more than 900+ articles from 2017 to November, 2020 which is to make sure our search engine has only the latest and fresh articles.

Existing relevant systems

There are not many relevant systems. These are what we can find.

1) DK FIND OUT : https://www.dkfindout.com/us/animals-and-nature/
   DK Find Out is a general interest site with fun facts, quizzes, games, and activities on a wide variety of topics. There are many articles for users to search, especially focusing on nature, including animal articles.
2) Popular Science : https://www.popsci.com/animals/
   Popular science is an interpretation of science intended for a general audience. While science journalism focuses on recent scientific developments, popular science is more broad-ranging. It may be written by professional science journalists or by scientists themselves. It is presented in many forms, including books, film and television documentaries, magazine articles, and web pages.

Methodology

1) Find reliable resources with a big amount of animal-related articles. In our case, we use www.livescience.com as a resource because their references for each article are reliable and professional. Moreover, they have over 1.2 million followers on Facebook fanpage. This can boost their credibility.
2) Study the html component from the url of each article and think roughly about how we can extract data from this url perfectly.
3) Trial and error with many Python libraries to find the best way to do the web scraping.
4) Do the web scraping, process, and save all retrieved data. In our case, we store it in a variable in the code and do the web scraping and storing data in one go.
5) Use Elasticsearch to index the documents from the data that we saved and store them in Elasticsearch's database.
6) Create the GUI to let users input search queries and show all of the related results. In our case, we decided to create a web application for this.

Implementation

1) We chose Jupyter Notebook to be our development environment to do web scraping and indexing.
2) We decided to do web scraping first because this is the main data that we needed to use for our search system. So, we started by importing all of the modules that we needed to use for scraping websites which are json for converting json string to python dictionaries, re for doing regular expressions, urllib.request for requesting for the components of the websites, and bs4 or BeautifulSoup for doing web scraping.

   Web Scraping part

   ```
   import json
   import re
   import urllib.request
   from bs4 import BeautifulSoup
   ```

   *note that firstly we tried with the "inscriptis" module and it didn't work well, so we tried bs4 and then it worked.

3) We retrieved the website components from the input variable “url” and created a BeautifulSoup object with that parameter. Basically, variable soup will store the whole html components from that link in the nice and ready-to-use format.

```
def get_all_web_data(url):
    try:
        html = urllib.request.urlopen(url)
    except:
        return None
    soup = BeautifulSoup(html)
```

4) After we get the website components we use the findAll method from BeautifulSoup object to find all of the html tags named “ script type = “application/ld+json” ”. Basically, this came from our experiment and we found out that this tag contained all of the metadata of the article. Then, we converted it into a string and did some cleanup. But they are all in json string format, so we cannot access each element inside. We had to convert the json string to a Python dictionary first. After that we can access each element and save the metadata that we want, such as a starting word (we will explain in the next step why we want this), author, headline, date, and link, into the variables.

```
try:
#find where the metadata is
    content = soup.findAll('script',type='application/ld+json')
    formatted_paragraph = content[0].string.replace('\n','').replace('&apos;','\'').replace('  ','')
    doc_data = json.loads(formatted_paragraph)

#store article metadata
    start_word = doc_data['articleBody'][:doc_data['articleBody'].find(' ')]
    author = doc_data['author']['name']
    headline = doc_data['headline']
    date = doc_data['datePublished']
    original_url = doc_data['url'] #re.search("https://www.livescience.com/\S*\.html",doc_data['url']).group()
except:
    return None
```

5) For now, we had already got the metadata of the article. So, it’s time to scrape the full article itself, which turns out to be a very challenging task. We firstly got all of the text by using findAll with “p” tags because the article has to be in the paragraph format, therefore it needs to be stored inside “p” tags. After we got the list of all of the documents we found that the last 6 of “p” tags are ads, so we deleted them by slicing [:-6] and stored the rest in the list. We turned the list to the full long string for us to be easy to process further. The point here is that we even got deleted ads from the bottom of the “p” tags, but there is still a lot of unwanted text before the main article itself. So, we had to find the way to slice the full long string to start at the first word of the article. Luckily, in the metadata that we extracted, there is a field that stores the short version of the article. So, we can extract the first word in the article from there and use it here as the starting point to slice the long string. That’s why we store a variable named “start_word” from the previous step. Then we did another string cleanup one last time and returned all of the data that we wanted from the website out as a dictionary.

```python
    #find and store the full article
    full_content = soup.findAll('p')
    full_paragraph = list()
    for element in full_content[:-6]:
        full_paragraph.append(str(element))
    full_long_paragraph = ''.join([element for element in full_paragraph])
    cleantext = BeautifulSoup(full_long_paragraph, 'lxml').text
    completed_article = cleantext[cleantext.find(start_word):].replace('\xa0','')

    return {'headline':headline, 'author':author, 'date':date[:10], 'link':original_url, 'content':completed_article}
```

This is the output example from this function.

*note that you can still see some \' which come from the pure word that the web developers typed in when we created the pages. For some pages, we can clean that, but it didn't work for some pages.

```
In [215]: #for test
          test = get_all_web_data('https://www.livescience.com/57965-fossils-of-extinct-giant-rodents-found.html')
          test
```

```
Out[215]: {'headline': "Extinct Giant Rodents' Family Tree Rewritten by New Fossil Finds",
 'author': 'Mindy Weisberger',
 'date': '2017-02-22',
 'link': 'https://www.livescience.com/57965-fossils-of-extinct-giant-rodents-found.html',
 'content': 'Scientists have found a near-complete skull and a jaw from a pair of giant rodents belonging to a group that lived
millions of years ago in South America, and they say the fossils show that the extinct creatures weighed as much as 1 ton when
fully grown.These are the best-preserved fossils to date of this extinct group, which was previously known only by skull fragme
nts and individual teeth, the scientists reported in a new study.The new fossils of the two rodents – an adult and a juvenile –
paint a more complete picture of the extinct and massive rat-like animals, the researchers said. For instance, the finds raise
questions about how these giant rodents were classified within their genus, and hint that several species that were thought to
be related may instead be a single species, the researchers wrote in the new study. [In Images: \'Field Guide\' Showcases Bizar
re and Magnificent Prehistoric Mammals]A number of oversize rodent species roamed South America during the Miocene epoch, which
lasted from about 23 million years ago to 5.3 million years ago, and some were downright gigantic. The largest rodent ever desc
ribed, the enormous Josephoartigasia monesi, was roughly the size of a buffalo and had a bite force as powerful as a tiger\'s,
according to a study published in February 2016 in the Journal of Anatomy.However, most of these large-rodent lineages went ext
inct long ago, except for the capybara, a water-loving, web-footed rodent that can weigh as much as 174 lbs. (79 kilograms). Al
so known as "water hogs" and "masters of the grasses," capybaras are found in Central and South America – with the exception of
one rogue individual that recently appeared in central California. (After several sightings, this capybara still remains at lar
ge.)Fossils from the giant-rodent genus Isostylomys date back to the early 20th century, but the new finds from Uruguay\'s Cama
cho Formation, a site from the late Miocene epoch – about 12 million to 5 million years ago – are the most complete to date.Sci
entists uncovered a nearly intact adult skull and jawbone, as well as a juvenile jawbone containing all of its teeth. Both indi
viduals represent the species Isostylomys laurillardi, which is thought to be nearly as large as J. monesi. The fossils\' excep
tional condition allowed scientists to compare tooth development between the adult and the juvenile, thus providing a new persp
ective on all other species in this genus, which had been described from more fragmentary fossil evidence.The study authors fou
nd that the adult-tooth shape emerged fairly early in the rodent\'s development, growing larger as the animal matured. Then, th
ey evaluated prior fossil finds by considering three possible tooth forms for I. laurillardi – prenatal, juvenile and adult – r
ecognizing that adult-tooth forms could vary in size. The researchers\' analysis determined that three known Isostylomys specie
s were, in fact, one species – I. laurillardi."Our study shows how the world\'s largest fossil rodents grow," study lead author
Andres Rinderknecht, a researcher in the Department of Paleontology at Uruguay\'s National Museum of Natural History, said in a
statement.The researchers concluded that, from a very young age, the giant rodents were very similar to the adults, Rinderknech
t said. That conclusion led the research team to deduce that the vast majority of the prior hypotheses were wrong, he said.The
findings were published online Tuesday (Feb. 21) in the Journal of Systematic Palaeontology.'}
```

This is the example of unwanted "p" tags in the string that we got which is why we have to cleanup and get the string that starts at the first word of the article.

```
'<p class="byline">\nBy\n<span class="no-wrap by-author"><a href="https://www.livescience.com/author/brandon-speckter" rel="aut
hor"><span style="white-space: nowrap">Brandon Speckter</span> - <span>Senior Writer</span></a></span>\n<time class="no-wrap re
lative-date chunk" datetime="2020-08-04T19:42:32Z" itemprop="datePublished">04 August 2020</time>\n</p><p class="strapline">One
woman is being treated for fractured ribs and internal bleeding, news outlets reported on Tuesday.</p><p>Two snorkelers were ho
spitalized after being smacked by the fins of an angry mama <a href="https://www.livescience.com/58464-humpback-whale-facts.htm
l"><u>humpback whale</u></a> off the coast of Australia, news outlets reported on Tuesday (Aug. 4).</p><p>The snorkelers were s
wimming with a whale-watching tour group on Saturday (Aug. 1) near Ningaloo Reef in Western Australia. The reef is a hotspot fo
r marine animal migrations, including elusive<a href="https://www.livescience.com/55412-whale-sharks.html"> <u>whale sharks</u>
</a>, which flock to the area every spring.</p><p class="mid__article"></p><p>The snorkelers were watching a humpback whale mot
her and calf swim by when the mother began exhibiting defensive behaviors, Australian news site<a data-original-href="https://w
ww.perthnow.com.au/news/wildlife/moment-protective-mother-humpback-whale-struck-swimmer-on-ningaloo-reef-tour-captured-ng-b8816
28677z" href="https://www.perthnow.com.au/news/wildlife/moment-protective-mother-humpback-whale-struck-swimmer-on-ningaloo-reef
-tour-captured-ng-b881628677z"> <u>Perth Now reported</u></a>. The 50-foot-long (15 meters) whale swam at the snorkelers, lashi
ng at the water with her tail. One of the snorkelers, a 29-year-old woman, was struck by the whale\'s tail, which fractured her
ribs and caused internal bleeding,<a data-original-href="https://www.bbc.com/news/world-australia-53632975" href="https://www.b
bc.com/news/world-australia-53632975"> <u>paramedics told the BBC</u></a>. She was flown to a Perth hospital, where she remains
in serious but stable condition as of Tuesday.</p><p>A second woman was slapped by the same whale\'s pectoral fin, which tore h
er hamstring.</p><p><strong>Related: </strong><a href="https://www.livescience.com/64969-whale-almost-swallows-man.html"><stron
g>What a fluke! Man ends up in whale\'s mouth</strong></a></p><p>Chartered whale-watching tours of Ningaloo are currently in th
e midst of a seven-year trial program monitored by the<a data-original-href="https://www.dpaw.wa.gov.au/management/marine/marin
e-wildlife/552-swimming-with-humpback-whales" href="https://www.dpaw.wa.gov.au/management/marine/marine-wildlife/552-swimming-w
ith-humpback-whales"> <u>Australian Department of Biodiversity, Conservation and Attractions</u></a> (DBCA). Normally, swimmers
are not allowed within 330 feet (100 m) of whales in Australia, according to the DBCA website. However, tourists participating
in licensed whale-swimming tours are exempt from these rules.</p><p>—<a href="https://www.livescience.com/28054-whales-giants-o
f-the-deep.html">Whale album: Giants of the deep</a></p><p>—<a href="https://www.livescience.com/bubble-net-whales-video.html">
Watch rare footage of whales blowing \'bubble nets\' to capture prey in a vortex of doom</a>\xa0</p><p>—<a data-original-href
="https://www.youtube.com/watch?v=JIF7XWe_3Sw" href="https://www.youtube.com/watch?v=JIF7XWe_3Sw">Video: Humpbacks block orca
\'s feeding frenzy</a></p><p>Like most animals, humpback whales have strong defensive instincts when it comes to protecting the
ir young. Mother whales are known to put themselves in harm\'s way to shield calves from hungry sharks – and occasionally, they
\'ve even been spotted<a href="https://www.livescience.com/61380-humpback-whale-saves-diver-video.html"> <u>trying to shield hu
mans</u></a> or seals from nearby predators. This "unintentional altruism" likely arises from a behavioral rule to protect calv
es from nearby threats at all costs, marine biologist Robert Pitman<a href="https://www.livescience.com/55639-humpbacks-protect
-when-killer-whales-attack.html"> <u>previously told Live Science</u></a>.</p><p>As such, swimming in close proximity to humpba
ck whales – which can weigh more than 36 tons (33 metric tons) – "involves some inherent risk," the DBCA said in a statement fo
llowing the incident.</p>'
```

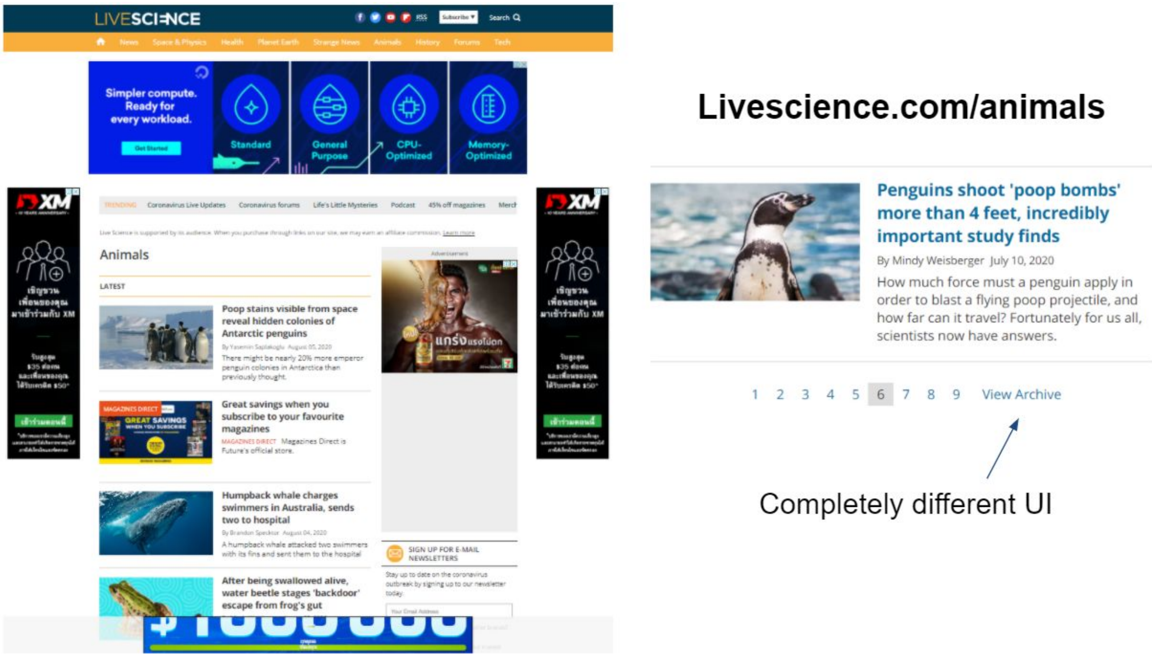
6) Up to now, we had completely implemented the function that can extract the data from a url. But we want more than 500 articles and if we had to paste the link one-by-one it would be such a time consuming task. So, we had the idea to extract the link from the main page which contains many article links in that page. We used almost the same method, but this time after we got a long string from “a class = “article-link” ” tags which are the tags that contain all of the article links on the web page, we used the regular expressions to extract the links. We can do this because we already know the format of what we want to extract unlike in the previous step that we have no idea what the starting words should be and the starting word differs from a webpage to a web page. That's why we had to use the method that we had shown.

```
In [79]: #Get all links from the main browse page of the website
         def get_all_links_current_version(main_url):
             try:
                 html = urllib.request.urlopen(main_url)
             except:
                 return None
             soup = BeautifulSoup(html)

             #Get Links from the current version of the website
             find_article_links = soup.findAll('a',class_='article-link')
             all_links = re.findall("https://www.livescience.com/\S*\.html",str(find_article_links))
             return all_links
```

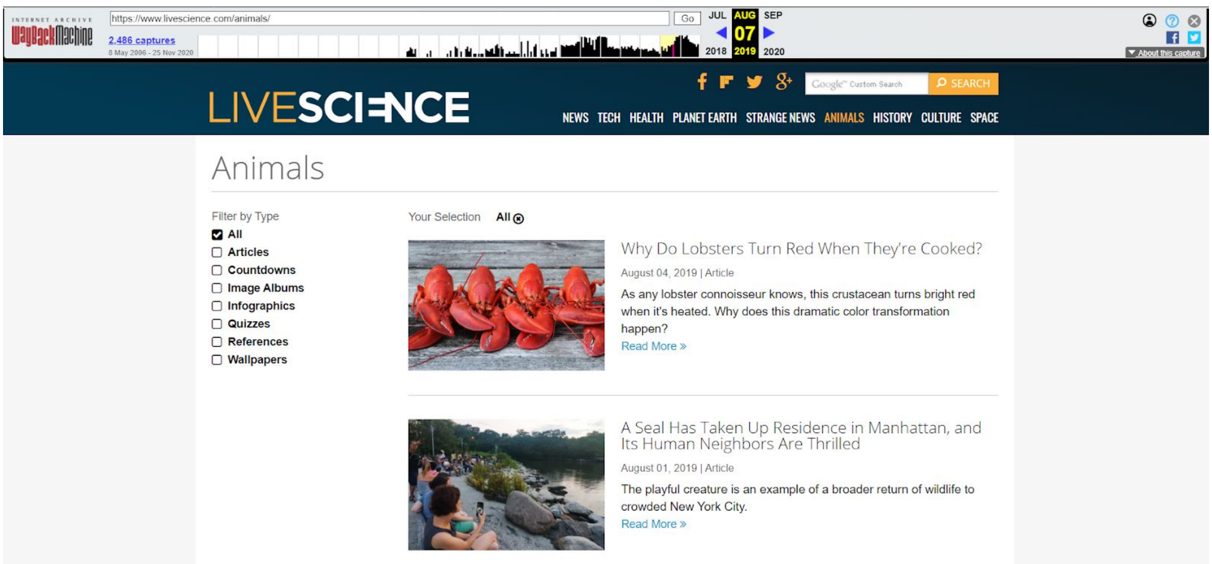
This is the output example of this function.
*note that we named it “current_version” because there is another function similar to this, but works for the different versions of this website in the different time which we will show it later.

```
In [111]: #for test
          test = get_all_links_current_version('https://web.archive.org/web/20200320223035/https://www.livescience.com/animals/')
          test

Out[111]: ['https://www.livescience.com/ancient-fish-fingers.html',
           'https://www.livescience.com/macaque-fight-thailand-temple-coronavirus.html',
           'https://www.livescience.com/earth-shorter-days-millions-years-ago.html',
           'https://www.livescience.com/white-giraffes-slaughtered-by-poachers.html',
           'https://www.livescience.com/smallest-dinosaur-of-mesozoic.html',
           'https://www.livescience.com/swamp-wallaby-always-pregnant.html',
           'https://www.livescience.com/oldest-cave-dwelling-animal-cockroaches.html',
           'https://www.livescience.com/deep-sea-sponges-sneeze-underwater.html',
           'https://www.livescience.com/llm-podcast-8-dinosaurs.html',
           'https://www.livescience.com/why-cats-have-white-socks-on-paws.html',
           'https://www.livescience.com/polar-bears-photos.html',
           'https://www.livescience.com/coconut-crab-clicking.html',
           'https://www.livescience.com/parasitic-worms-in-lizard-embryos.html',
           'https://www.livescience.com/first-non-breathing-animal.html',
           'https://www.livescience.com/ice-age-bird-permafrost.html',
           'https://www.livescience.com/dinosaur-tumor-tail.html',
           'https://www.livescience.com/snake-orgy-florida.html',
           'https://www.livescience.com/monkey-brains-have-engine-of-consciousness.html',
           'https://www.livescience.com/pink-manta-ray-spotted.html',
           'https://www.livescience.com/aye-aye-six-fingers-discovered.html']
```

7) Now, we had implemented 2 main functions to do web scraping and get all data that we want from many links at the same time from just the main link as an input. But we found some problems with the current version of the website as shown in the picture below.



As you can see, the current versions of the website let us access only 9 main pages and each page contains 20 links of articles that's totally 180 articles and that's not enough for us. And when we tried to click on the "View archive" button, it showed all of the articles in this website which most of them are not related to animals at all and it has a completely different UI. Therefore, it didn't work for us. We came up with an idea of what if we could go back in time and do the web scraping from the previous version of this website so that we can use the same function to do scraping. Then, we found https://web.archive.org/ it is the website that keeps track of the whole internet for a very long time. And that worked for us, so we put the main link there and we can do scraping as we expected. But that's not all the problems, after we used https://web.archive.org/ to go back in time, once, we went to older version than 13 August, 2019, we found that the website had changed the UI as we show below.



So we need to create another function to do web scraping from this version of the website. And luckily Livescience used this UI from 2016 to 2019 and that's enough for us to extract more than 500 articles. Another luck we got, the new function is very similar to the old version. Only difference is that we find all of the tags from class “read-url” instead of “article-link”.

```
In [80]: def get_all_links_older_version(main_url):
             try:
                 html = urllib.request.urlopen(main_url)
             except:
                 return None
             soup = BeautifulSoup(html)

             #Get links from the older version of the website (same method)
             #Start working since May 5, 2016 to august 7, 2019
             find_article_links = soup.findAll('a',class_='read-url')
             all_links = re.findall("https://www.livescience.com/\S*\.html",str(find_article_links))
             return all_links
```

This is the output example from the new function for scraping the older version of the website.

*note that with the older version of UI, each page contained only 10 links per page (the current version contains 20 links per page) But that’s not a big problem for us. So it does work fine.

```
In [85]: test2 = get_all_links_older_version('https://web.archive.org/web/20180102215036/https://www.livescience.com/animals/')
         test2

Out[85]: ['https://www.livescience.com/61308-do-animals-get-jealous.html',
          'https://www.livescience.com/61299-sea-stars-making-comeback.html',
          'https://www.livescience.com/61287-new-book-asks-does-it-fart.html',
          'https://www.livescience.com/61292-does-it-fart-10-fascinating-facts-about-animal-toots.html',
          'https://www.livescience.com/61272-new-worm-species-no-anus-discovery.html',
          'https://www.livescience.com/61269-cambrian-sea-monster.html',
          'https://www.livescience.com/32115-bison-vs-buffalo-whats-the-difference.html',
          'https://www.livescience.com/61248-dog-chocolate-poisonings-spike-at-christmas.html',
          'https://www.livescience.com/61241-how-often-do-dogs-maul-owners.html',
          'https://www.livescience.com/61238-spiders-build-sandcastles-underground.html']
```

8) After we implement the new function to do web scraping for the older version of the website, that means we got everything done with web scraping. Next we put all the links that we want to scrap and put it in a single list variable.
   *note that this is just a part of the total number of links. If you want to see the full list, you can look in the real code. We use a total of 77 main links. And that equals 1,000 links for the article.

```
In [118]: #Use when everything is finished
          all_article_links = list()
          main_links = {'https://www.livescience.com/animals/',
                        'https://web.archive.org/web/20200320223035/https://www.livescience.com/animals/', #Current #March 20, 2020
                        'https://web.archive.org/web/20200216223856/https://www.livescience.com/animals/', #Current #Feb 16, 2020
                        'https://web.archive.org/web/20200127213043/https://www.livescience.com/animals/', #Current #Jan 27, 2020
                        'https://web.archive.org/web/20200127213045/https://www.livescience.com/animals/2', #Current #Jan 27, 2020
                        'https://web.archive.org/web/20200127214447/https://www.livescience.com/animals/3', #Current #Jan 27, 2020
                        'https://web.archive.org/web/20200104041205/https://www.livescience.com/animals/4', #Current #Jan 04, 2020
                        'https://web.archive.org/web/20190731024418/https://www.livescience.com/animals/', #Older #July 31, 2019
                        'https://web.archive.org/web/20190723220159if_/https://www.livescience.com/animals/2', #Older #July 31, 2019
                        'https://web.archive.org/web/20190709215801/https://www.livescience.com/animals/', #Older #Jul 09, 2019
                        'https://web.archive.org/web/20190709203939/https://www.livescience.com/animals/2', #Older #Jul 09, 2019
                        'https://web.archive.org/web/20190617213319/https://www.livescience.com/animals/', #Older #June 17, 2019
                        'https://web.archive.org/web/20190617213320if_/https://www.livescience.com/animals/2', #Older #June 17, 2019
                        'https://web.archive.org/web/20190525180136/https://www.livescience.com/animals/', #Older #May 25, 2019
                        'https://web.archive.org/web/20190513020621/https://www.livescience.com/animals/', #Older #May 13, 2019
                        'https://web.archive.org/web/20190513020623/https://www.livescience.com/animals/2', #Older #May 13, 2019
                        'https://web.archive.org/web/20190418160125/https://www.livescience.com/animals/', #Older #Apr 18, 2019
                        'https://web.archive.org/web/20190402194936/https://www.livescience.com/animals/2', #Older #Apr 02, 2019
                        'https://web.archive.org/web/20190319212821/https://www.livescience.com/animals/', #Older #March 19, 2019
                        'https://web.archive.org/web/20190312075436/https://www.livescience.com/animals/', #Older #March 12, 2019
                        'https://web.archive.org/web/20190301174738/https://www.livescience.com/animals/', #Older #March 01, 2019
                        'https://web.archive.org/web/20190221182843/https://www.livescience.com/animals/', #Older #Feb 21, 2019
```

9) We iterated through the list of main links and create some conditions that if the link is the current version, so use the current_version function and if not, use older_version function. Moreover, with the first link and condition that we wrote, we can iterate

from page 1-9 in 1 go. That means we get 180 links from just only the first main link. And we print to see the process at the runtime (as you can see some of them in the output).

```
In [119]: main_links_amount = len(main_links)
for i,main_link in enumerate(main_links):
    if i <= 6:
        if main_link.startswith('https://www.livescience.com'):
            for j in range(1,10):
                links_in_web = get_all_links_current_version(main_link+str(j))
                if links_in_web != None:
                    all_article_links.extend(links_in_web)
            print("Done", i, "out of", main_links_amount)
        else:
            links_in_web = get_all_links_current_version(main_link)
            if links_in_web != None:
                all_article_links.extend(links_in_web)
                print("Done", i, "out of", main_links_amount)
    else:
        links_in_web = get_all_links_older_version(main_link)
        if links_in_web != None:
            all_article_links.extend(links_in_web)
            print("Done", i, "out of", main_links_amount)
all_article_links = set(all_article_links)
len(all_article_links)
```

```
Done 0 out of 77
Done 1 out of 77
Done 2 out of 77
Done 3 out of 77
Done 4 out of 77
Done 5 out of 77
Done 6 out of 77
Done 7 out of 77
Done 8 out of 77
```

10) After we finished scraping the links from 77 main links we got 944 links. Some of them didn't work, but that's totally fine for us. We got enough links. Then we call the function get_all_web_data that we implemented at the very beginning to get all of the data from 944 links and store in a list variable. (this process took about 1 hour and 15 mins due to the fact that the code have to access the website at runtime, so it takes time)

```
In [121]: article_data_list = list()
all_article_links_amount = len(all_article_links)
for i,link in enumerate(all_article_links):
    return_data = get_all_web_data(link)
    if return_data != None:
        article_data_list.append(return_data)
        print("Done", i, "out of", all_article_links_amount)
```

```
Done 0 out of 944
Done 1 out of 944
Done 2 out of 944
Done 3 out of 944
Done 4 out of 944
Done 5 out of 944
Done 6 out of 944
Done 7 out of 944
Done 8 out of 944
Done 9 out of 944
Done 10 out of 944
Done 11 out of 944
```

11) It's time to do indexing for all of the data that we got from the links. We got only 904 from 944 article links because again some of them didn't work. But it's still just fine. First, we imported the elasticsearch module. And then we iterated through 904 article data and use the iterate number to get the document id which is ranged between 0-903. And set the body to the article data.

Search Engine (Main part)

```
In [1]: from elasticsearch import Elasticsearch
```

```
In [2]: es = Elasticsearch()
```

```
In [124]: all_usable_article_data_amount = len(article_data_list)
for i,article_data in enumerate(article_data_list):
    es.index(index='article',id=i,body=article_data)
    print("Done", i, "out of", all_usable_article_data_amount)
```

```
Done 0 out of 904
Done 1 out of 904
Done 2 out of 904
Done 3 out of 904
Done 4 out of 904
Done 5 out of 904
Done 6 out of 904
Done 7 out of 904
Done 8 out of 904
Done 9 out of 904
```

12) In Jupyter Notebook, we also tried the search function of elasticsearch and it worked pretty fine. But in the end, we didn't use that because we have to move to the real .py file to work with the django framework that we will implement the web application for our search system.

This is our search prototype that we tried on Jupyter Notebook. As you can see, it worked pretty well. (We will explain the search query in the last step)

```
In [5]: search_query = input('Search: ')
size = input('Number of docs: ')
not_include = input('Words not to include (optional): ')
body = {
    "from":0,
    "size":int(size),
    "query": {
        "bool":{
            "should":[
                { "match": {"content":{"query": search_query}} },
                { "match": {"content":{ "query": search_query, "operator": "and" }} },
                { "match_phrase": {"content":{"query": search_query, "boost": 2}} }
            ],
            "must_not":[
                { "match": {"content":{"query": not_include}}}
            ]
        }
    }
}

res = es.search(index="article", body=body)
res
#res['hits'].keys()
#print(f"Number to show {size} \nMatched Query: {res['hits']['hits'][0]['_source'].get('content')}")
```

```
In [12]: #print matched results
for i,result in enumerate(res['hits']['hits']):
    print("Ranking:",i+1)
    print("Score:", result['_score'])
    print(f"Headline: {result['_source']['headline']}")
    print(f"Author: {result['_source']['author']} || Date: {result['_source']['date']}")
    print("\n\t",result['_source']['content'])
    print("\nOriginal article:",result['_source']['link'])
    print("------------------------------------------------------------------------------------------------------------------")
```

```
Ranking: 1
Score: 0.6904231
Headline: Why do so many cats have white 'socks' on their paws?
Author: Grant Currin || Date: 2020-02-29

         If you see a house cat, the odds are high that it will have white paws, a look that many owners affectionately call
"socks." But socks are rarely seen in wildcats, the elusive and undomesticated cousin of the house cat, so why do so many pet
cats sport furry white feet?As it turns out, this story started about 10,000 years ago, when humans and cats decided life was
better together.This domestication eventually led to über-prevalent socks on cats, as well as other well-known coat patterns,
said Leslie Lyons, professor emerita and head of the Feline Genetics Laboratory at the University of Missouri College of Vete
rinary Medicine.Related: Why do cats wiggle their butts before they pounce?"As humans became farmers and started staying in o
ne place, they had grain stores and refuse piles" that attracted rodents, Lyons said. It was a mutually beneficial arrangemen
t: the humans had fewer rodents to deal with and the cats got an easy meal.The wild, undomesticated progenitor species of hou
se cats, Felis silvestris, lives in Africa and Eurasia. One population of them even lives on Mount Etna, an active volcano in
Sicily. These felines are tasty snacks as kittens and stealthy predators as adults, so individuals born with a coat that offe
rs camouflage have tended to survive and reproduce.But not every F. silvestris is born with a coat that blends into its habit
at."Genetic mutations are occurring all the time," Lyons said.There isn't much evidence to indicate why early cat people chos
e the individuals they did, but Lyons said the range of coats seen on modern domestic cats shows that our agrarian ancestors
favored cats with markings that would have interfered with their camouflage. In its native mixed forest or scrub desert envir
onment, a cat with stark white paws would have stood out to predators and prey.When humans started taking an interest in cat
```

13) After moving to PyCharm IDE and django framework, we have to set up many things. But it's not the point, so we are not going to show everything on how the framework is set up here. Basically, when we enter the website it will call the file urls.py in "searchsite" folder and that file again will call another urls.py in "esearch" folder. And right after that this will call the function named "search_index" from views.py file.

This is the body of the file urls.py in "esearch" folder.

```
from django.urls import path
from . import views

app_name = 'esearch'
urlpatterns = [path('', views.search_index)]
```

14) search_index function is basically the function to receive the input from the search boxes from the html page and do the searching with function "esearch" from es_call.py file (we will go into detail in the next step) and then get the result as a response object from esearch function and send the result to main.html page to display the search result.

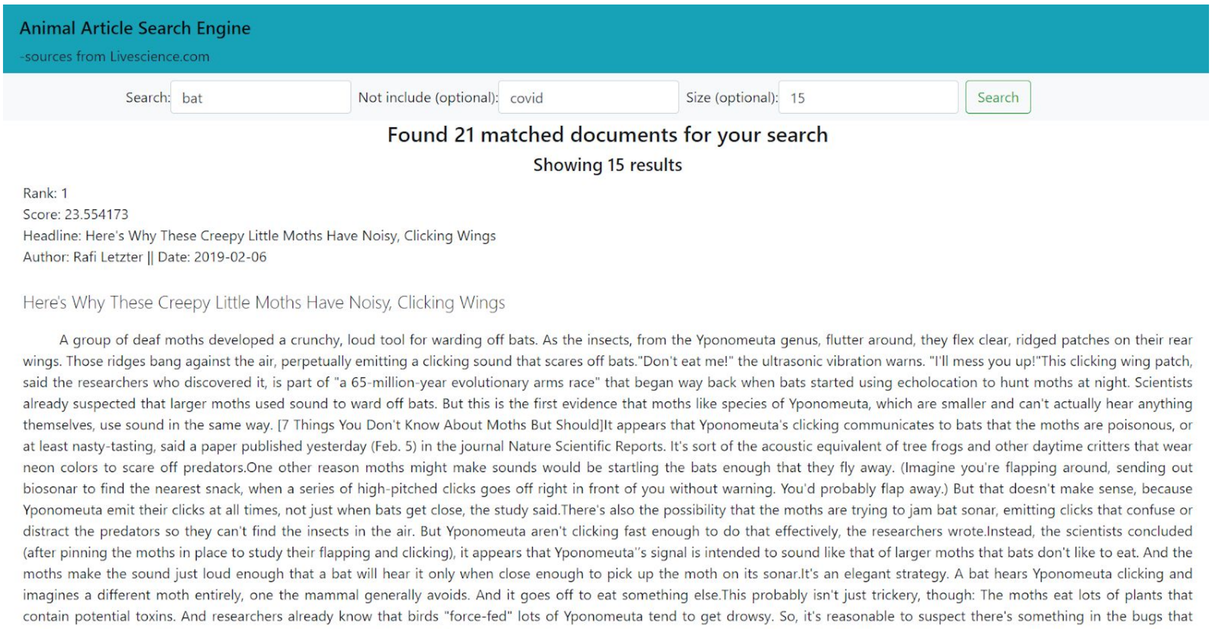
This is the body of search_index function in views.py

```python
from django.shortcuts import render
from django.http import HttpResponse
from .es_call import esearch


def search_index(request):
    results = []
    search_query_term = ""
    not_include_term = ""
    size_term = 10
    if request.GET.get('search_query') and request.GET.get('not_include') and request.GET.get('size'):
        search_query_term = request.GET['search_query']
        not_include_term = request.GET['not_include']
        size_term = int(request.GET['size'])
    elif request.GET.get('search_query') and request.GET.get('not_include'):
        search_query_term = request.GET['search_query']
        not_include_term = request.GET['not_include']
    elif request.GET.get('search_query') and request.GET.get('size'):
        search_query_term = request.GET['search_query']
        size_term = int(request.GET['size'])
    elif request.GET.get('search_query'):
        search_query_term = request.GET['search_query']
    elif request.GET.get('not_include'):
        not_include_term = request.GET['not_include']
    elif request.GET.get('size'):
        size_term = int(request.GET['size'])
    search_term = search_query_term or not_include_term or size_term
    (results, total_hits) = esearch(size_term, search_query=search_query_term, not_include=not_include_term)
    print(results)
    context = {'results': results, 'matched': total_hits, 'showing': len(results), 'search_term': search_term}
    return render(request, 'esearch/main.html', context)
```

15) esearch function in the es_call.py is the main function that does all of the search work and returns the response object. We need to import elasticsearch_dsl because the default module of elasticsearch return the result as python dictionary, so that didn't work with the website which require response object and elasticsearch_dsl is the module that returns the search result as a response object. You can see that we use "bool" queries to combine "should" and "must_not" queries to work together. Inside the "should" query we use 2 "match" query and 1 "match_phrase" query to optimize the search result. Basically, the first "match" query is the basic search one. The second "match" query, it contains "operator": "and" this means that if the document contains all of the words in the search query, the documents will have higher score (by default, if you didn't specify operator, elasticsearch us "OR"). The last "match_phrase" query is for the document that contains exactly the words in the order will gain a higher score and we boost this "match_phrase" to 2. That means we prioritize this phrase search. And lastly, "must_not" query works with the "not_include" search box on the website. And the get_results function is just for getting the results and storing them in variables.

```python
from elasticsearch import Elasticsearch
from elasticsearch_dsl import Search, Q


def esearch(size, search_query="", not_include=""):
    es = Elasticsearch()
    q = Q("bool", should=[Q("match", content=search_query),
                          Q({"match": {"content": {"query": search_query, "operator": "and"}}}),
                          Q({"match_phrase": {"content": {"query": search_query, "boost": 2}}})],
          must_not=[Q("match", content=not_include)])
    s = Search(using=es, index="article").query(q)[:size]
    response = s.execute()
    print('Total', response.hits.total, ' hits found.')
    results = get_results(response)
    return (results, response.hits.total['value'])


def get_results(response):
    results = []
    for i, result in enumerate(response):
        result_tuple = (i+1, result.meta.score, result.headline,
                        result.author, result.date,
                        result.content, result.link)
        results.append(result_tuple)
    return results
```

16) With everything working out well together, we got the perfect web application for search engines with 3 search boxes here.



Results and Discussion

With all the time and effort that we spend, we get the result that we want it to be. We completely started learning most things from scratch especially Elasticsearch and Django. At the end we managed to get things done as we expected. Even though there is something we wish we could have done more such as optimizing the search engine even more, auto-completing the search boxes, and words suggestion, with the time limit and a lot of projects from others subjects to be done, this is the best we can get. Maybe in the future or in our free time, we might pick this project up and try to further the capabilities of this project again.

Conclusion

In conclusion, we have successfully implemented the animal-related articles search system with Elasticsearch Python API and Django web framework. We did web scraping on Jupyter Notebook to get the data that we wanted from www.livescience.com. We solved the problem of too little article links to get by using https://web.archive.org/ wayback machine with the idea to go back in time and get the link from the website in the past with the same code function and also write a new function to work with the older version of the website. Lastly, we've done indexing with Elasticsearch and created a website as a GUI for our search engine with Django. We have learned like a lot from this project. It teaches us to work as a team, to come up with the idea, to solve the problem, and a lot more. It has been a great experience for us.