Design and Implementation of Web Scrapper for Fact-Checking Website

Makhan Kumbhkar[1*] · Shraddha Masih[2] and
Savita Kolhe[3]
[1]Research Scholar, School of Computer Science, DAVV, Indore MP, India
[2] Professor, School of Computer Science, DAVV, Indore MP, India
[3] Principal Scientist, Computer Application ICAR-Indian Institutes of Soybean Research, Indore, MP, India
[*1]makhan@christianeminent.com

Abstract. Web scraping software automates the extraction of vast amounts of data from websites, streamlining the tedious manual process of copying and pasting information into spreadsheets or other storage formats. For instance, collecting content from websites like Politifact.com manually would require hiring multiple people to visit each page, copy information such as titles, authors, statements, dates, and sources, and then input it into a database. This manual approach could take days or even months to complete. However, web scraping tools can perform this task programmatically, visiting every page and parsing HTML to extract the required data efficiently. In our research, we utilized web scraping to collect and organize information from Politifact.com, specifically targeting key details like the title, author, statement, date, and source, significantly reducing the time and effort needed for data preparation.

Keywords: HTML, Beautiful Soup, Parsing, Social Media, PolitiFact.com.

1 Introduction

In our research, we utilized web scraping to collect and analyze information from Politifact.com, a reputable fact-checking website operated by the nonprofit Poynter Institute in St. Louis. Politifact.com is widely recognized for its commitment to separating fact from fiction in the political arena. Notably, the site earned the 2009 Pulitzer Prize for National Reporting due to its innovative fact-checking initiative during the 2008 presidential campaign, where it used in-depth reporting and the reach of the World Wide Web to examine over 750 political claims. This initiative played a crucial role in helping voters distinguish between political rhetoric and factual information. To facilitate our research, we employed web scraping techniques to systematically extract key details from Politifact.com. Specifically, we targeted data fields such as the title of the article the author, the statement being fact-checked, the date of publication, and the original source of the in-formation. By automating this data collection process, we were able to efficiently gather large volumes of structured data that would have been time-consuming to collect manually. This method not only saved significant time and resources but also ensured that our data was comprehensive and up-to-date, allowing for a more thorough analysis of the information presented on Politifact.com. The use of this novel website for fact extraction enabled us to delve deeper into the accuracy of political claims,

S. Bhalerao et al. (eds.), *Proceedings of the International Conference on Recent Advancement and Modernization in Sustainable Intelligent Technologies & Applications (RAMSITA-2025),*Advances in Intelligent Systems Research 192,
https://doi.org/10.2991/978-94-6463-716-8_4

contributing to a better understanding of the dynamics of truth and misinformation in political discourse.

2 Literature Review

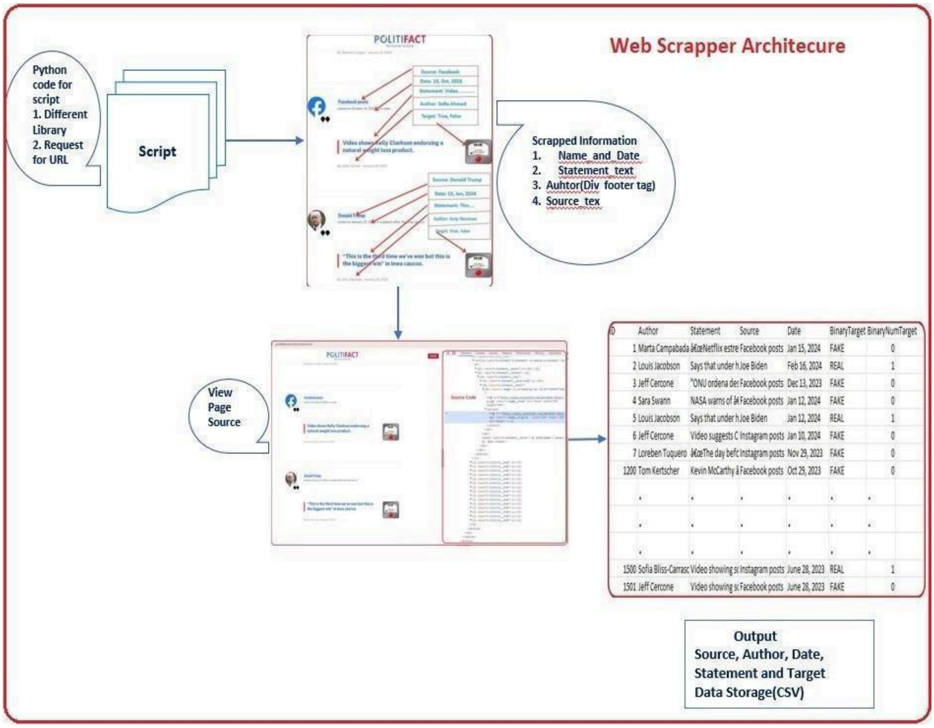
Web scraping is a technique for extracting data from the World Wide Web (WWW) and storing it in a file system or database for analysis or retrieval, as described in[3]. Various studies highlight its utility in automating data collection for diverse applications. In [2], a system was proposed to scrape cybercrime-related news articles, which were classified into respective crime categories. Similarly, web scraping has been used to extract structured information efficiently, bypassing the time-consuming process of manual data collection [5]. Python is often preferred for its robust libraries that simplify scraping [6].

The legal and ethical implications of web scraping have also been explored, particularly regarding its legality and situational appropriateness [4]. Innovative approaches, like UzunExt [7], optimize scraping by extracting content directly with string methods, bypassing the creation of a DOM tree. Cloud-based solutions, leveraging platforms like AWS, have also been studied to enhance scalability and computational efficiency [14].

In [8], an overview of traditional web scraping solutions was presented, emphasizing their functional characteristics and success in various domains. Another unique concept, "demos scraping," was introduced in [9], focusing on gathering citizen-related information. These works underline web scraping's versatility in handling large-scale data extraction tasks across different fields.

3 Architecture of Web Scrapper

In this architecture, we proposed data collection and how to store data in a particular file, shown in Figure-1.



D	Author	Statement	Source	Date	BinaryTarget	BinaryNumTarget
1	Marta Campabada	â€œNetflix estre	Facebook posts	Jan 15, 2024	FAKE	0
2	Louis Jacobson	Says that under h	Joe Biden	Feb 16, 2024	REAL	1
3	Jeff Cercone	"ONU ordena de:	Facebook posts	Dec 13, 2023	FAKE	0
4	Sara Swann	NASA warns of â€	Facebook posts	Jan 12, 2024	FAKE	0
5	Louis Jacobson	Says that under h	Joe Biden	Jan 12, 2024	REAL	1
6	Jeff Cercone	Video suggests C	Instagram posts	Jan 10, 2024	FAKE	0
7	Loreben Tuquero	â€œThe day befc	Instagram posts	Nov 29, 2023	FAKE	0
1200	Tom Kertscher	Kevin McCarthy â	Facebook posts	Oct 29, 2023	FAKE	0
	.	.	.	.	.	.
	.	.	.	.	.	.
	.	.	.	.	.	.
1500	Sofia Bliss-Carrasc	Video showing sc	Instagram posts	June 28, 2023	REAL	1
1501	Jeff Cercone	Video showing sc	Facebook posts	June 28, 2023	FAKE	0

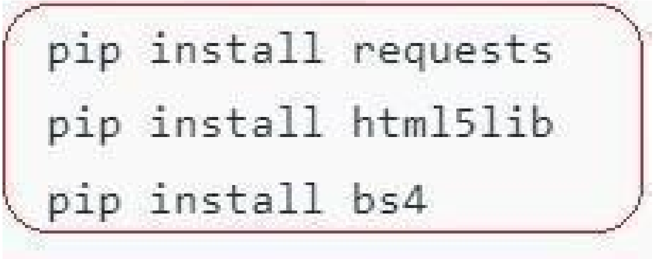
Fig. 1. Web Scrapper Architecture.

3.1 Implementation of Web Scrapper

Installing the required libraries using pip is the simplest method for installing external libraries in Python. It allows for easy management and installation of Python software packages. The Key libraries that are often essential include requests for handling HTTP requests, html5lib for parsing HTML documents, and BeautifulSoup (bs4) for web scraping and parsing HTML/XML content.These libraries, shown in Figure-2, are fundamental for tasks involving web data retrieval and processing in Python.

Beautiful Soup: A package called BeautifulSoup makes it simple to extract data from websites. It provides different HTML paradigms for iterating, searching, and altering the parse tree on top of an HTML or XML parser.

html5lib: html5lib is a Python package that implements the HTML5 parsing algorithm which is heavily influenced by current browsers and based on the WHATWG HTML5 specification. The lxml can benefit from the parsing capabilities of html5lib through the lxml Extension.



```
pip install requests
pip install html5lib
pip install bs4
```

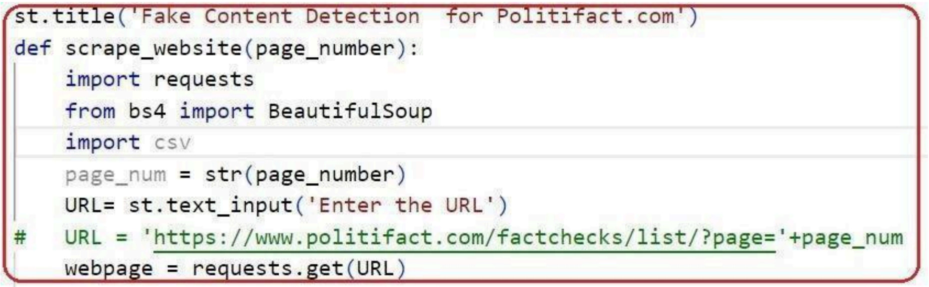
Fig. 2. Important Libraries

3.2Accessing the HTML Content From PolitiFact.com

There are two main ways to extract data from a website: by using API or by using a website (Politifact.com) for accessing information.

We followed some basic steps for accessing the HTML content from Politifact.com [17].

1. First of all, import the requests library.
2. Then, specify the URL of the webpage (Politifact.com) which want to scrape.
3. Send an HTTP request to the specified URL ((Politifact.com)) and save the response from the server in a response object called r.
4. Now, as print r.content to get the raw HTML content of the webpage.
5. We have shown the code in Figure-3.



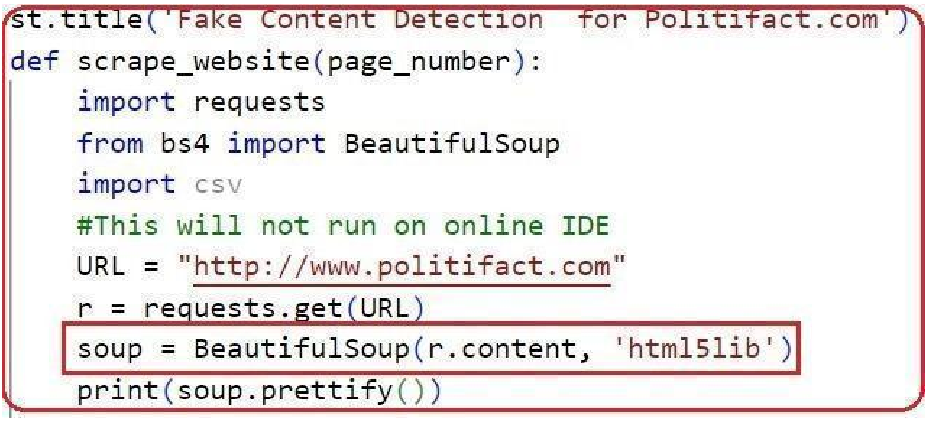
```
st.title('Fake Content Detection  for Politifact.com')
def scrape_website(page_number):
    import requests
    from bs4 import BeautifulSoup
    import csv
    page_num = str(page_number)
    URL= st.text_input('Enter the URL')
#   URL = 'https://www.politifact.com/factchecks/list/?page='+page_num
    webpage = requests.get(URL)
```

Fig. 3. Accessing the HTML content from webpage

Parsing the HTML content

To parse the HTML content retrieved from a website, the r.content method in Python is often used to obtain the raw response data. After retrieving the content, the html5lib parser can be employed to process and structure the HTML, making it easier to navigate and extract specific information from the webpage. In the steps, we created a BeautifulSoup object by passing two arguments:

1. **r.content:** It is the raw HTML content.
2. **html5lib:** Specifying the HTML parser which we want to use. Which is shown in Figure-4.

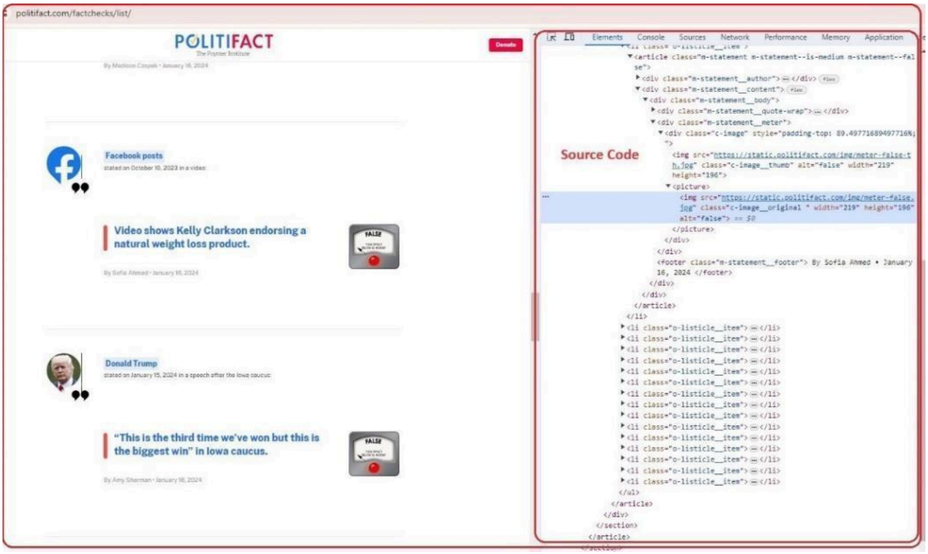


```
st.title('Fake Content Detection  for Politifact.com')
def scrape_website(page_number):
    import requests
    from bs4 import BeautifulSoup
    import csv
    #This will not run on online IDE
    URL = "http://www.politifact.com"
    r = requests.get(URL)
    soup = BeautifulSoup(r.content, 'html5lib')
    print(soup.prettify())
```

Fig. 4. Parsing the HTML content

Searching and Navigating Through the HTML Structure

We want to take some important information out of the HTML source code. Here basic source code of Politifact.com. All hierarchical structure data that might be programmatically extracted is contained in the soup object from the PolitiFact website. We are scraping a webpage with some quotes statement_footer, statement_quote,statement_meta and target statement from the HTML page. Therefore, we would like to develop a program to save those quotations statement_footer,statement_quote,statement_meta and target statement into the data frame from the HTML structure of websites. We can save data into the CSV file using pandas below figure-5.



By Madison Czopek • January 16, 2024

Facebook posts
stated on October 10, 2023 in a video:

Video shows Kelly Clarkson endorsing a natural weight loss product.

By Sofia Ahmed • January 16, 2024

Donald Trump
stated on January 15, 2024 in a speech after the Iowa caucus:

“This is the third time we’ve won but this is the biggest win” in Iowa caucus.

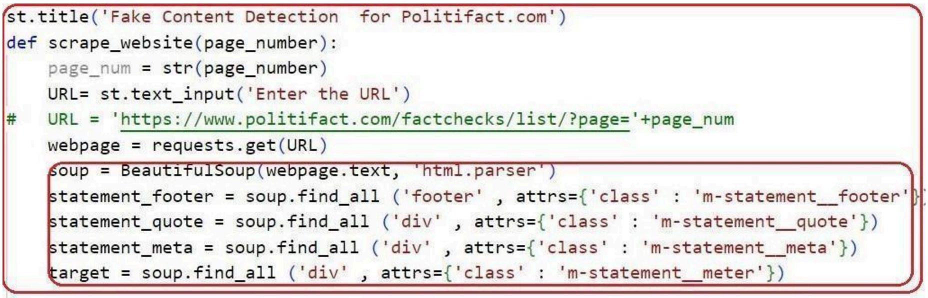
By Amy Sharman • January 16, 2024

```
▼<article class="m-statement m-statement--is-medium m-statement--fal
se">
  ▶<div class="m-statement__author">⋯</div> flex
  ▼<div class="m-statement__content"> flex
    ▼<div class="m-statement__body">
      ▶<div class="m-statement__quote-wrap">⋯</div>
      ▼<div class="m-statement__meter">
        ▼<div class="c-image" style="padding-top: 89.49771689497716%;
        ">
          <img src="https://static.politifact.com/img/meter-false-t
          h.jpg" class="c-image__thumb" alt="false" width="219"
          height="196">
         ▼<picture>
...         <img src="https://static.politifact.com/img/meter-false.
            jpg" class="c-image__original " width="219" height="196"
            alt="false"> == $0
          </picture>
        </div>
      </div>
      <footer class="m-statement__footer"> By Sofia Ahmed • January
      16, 2024 </footer>
    </div>
  </div>
</article>
</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
▶<li class="o-listicle__item">⋯</li>
</ul>
</article>
</div>
</section>
</article>
```

Fig. 5. HTML Structure of PolitiFact

4 Dataset Collection Using Web Scrapper

Data collection is the act of obtaining and examining precise information from PolitiFact to assess potential outcomes, trends, and probability, among Author, Statement, Source, Date and Target. We collected Author, Statement, Source, Date and Target unlabeled and real-time data from PolitiFact.



```
st.title('Fake Content Detection  for Politifact.com')
def scrape_website(page_number):
    page_num = str(page_number)
    URL= st.text_input('Enter the URL')
#   URL = 'https://www.politifact.com/factchecks/list/?page='+page_num
    webpage = requests.get(URL)
    soup = BeautifulSoup(webpage.text, 'html.parser')
    statement_footer = soup.find_all ('footer' , attrs={'class' : 'm-statement__footer'}
    statement_quote = soup.find_all ('div' , attrs={'class' : 'm-statement__quote'})
    statement_meta = soup.find_all ('div' , attrs={'class' : 'm-statement__meta'})
    target = soup.find_all ('div' , attrs={'class' : 'm-statement__meter'})
```

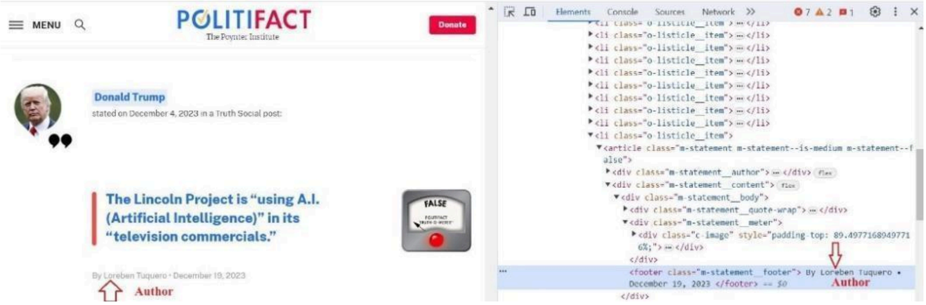
Fig. 6. Parsing the HTML Content of PolitiFact

Now, in the next steps we scrapped different types of attributes from the PolitiFact.com

Scrapping Author

Python's BeautifulSoup module allows us to scrape information from local HTML files. Website pages may be stored locally (offline), and it may be necessary to retrieve data from them. Using BeautifulSoup, we can scrape contents, such as tags like h1, h2, p, and div, and extract key information like the author, which is crucial for our research.

The figure-7 shows a web page from Politifact.com, where the author name, Loreben Tuquero, is found in the footer tag, within the class m-state-ment_footer. This class contains the information related to the author.



stated on December 4, 2023 in a Truth Social post:

The Lincoln Project is "using A.I. (Artificial Intelligence)" in its "television commercials."

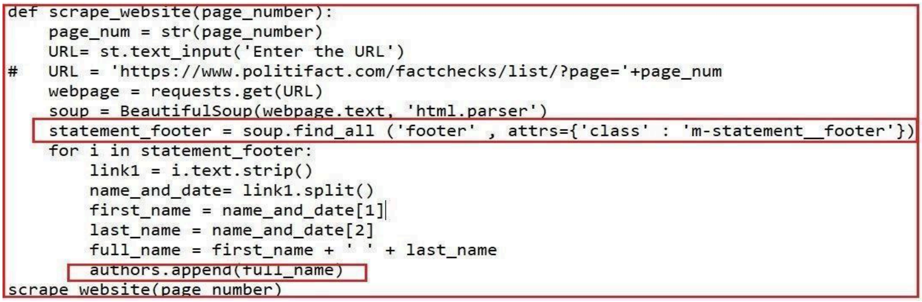
By Loreben Tuquero · December 19, 2023

```
▸<li class="o-listicle__item"> ⋯ </li>
▸<li class="o-listicle__item"> ⋯ </li>
▸<li class="o-listicle__item"> ⋯ </li>
▸<li class="o-listicle__item"> ⋯ </li>
▸<li class="o-listicle__item"> ⋯ </li>
▸<li class="o-listicle__item"> ⋯ </li>
▸<li class="o-listicle__item"> ⋯ </li>
▸<li class="o-listicle__item"> ⋯ </li>
▸<li class="o-listicle__item"> ⋯ </li>
▾<li class="o-listicle__item">
  ▾<article class="m-statement m-statement--is-medium m-statement--f
   alse">
    ▸<div class="m-statement__author"> ⋯ </div> flex
    ▾<div class="m-statement__content"> flex
      ▾<div class="m-statement__body">
        ▸<div class="m-statement__quote-wrap"> ⋯ </div>
        ▾<div class="m-statement__meter">
          ▸<div class="c-image" style="padding-top: 89.4977168949771
           6%;"> ⋯ </div>
         </div>
⋯        <footer class="m-statement__footer"> By Loreben Tuquero •
         December 19, 2023 </footer> == $0
       </div>
```

Fig. 7. Scrapping Author

Retrieving Author Name From m Statement Footer Tag:

We created a **scrape_website()** function for extracting authors from web pages. It takes one argument page number for information extraction and we passed the page URL into requests.get(URL) method. We used the for loop for extracting all the authors from a particular page and appended all authors in the list variable. Which is shown in the below Figure-8.



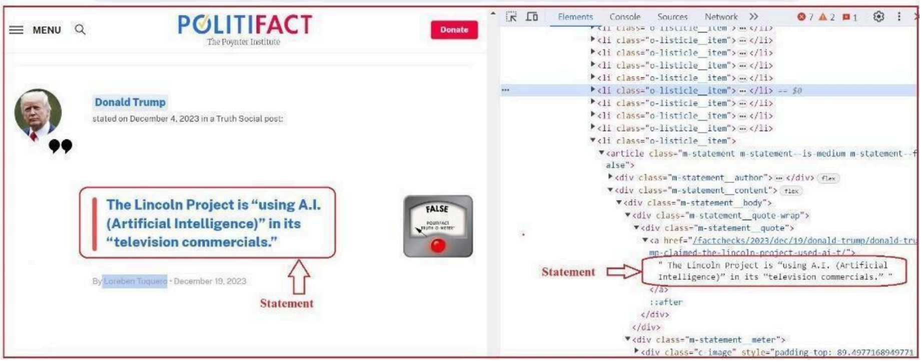
```
def scrape_website(page_number):
    page_num = str(page_number)
    URL= st.text_input('Enter the URL')
#   URL = 'https://www.politifact.com/factchecks/list/?page='+page_num
    webpage = requests.get(URL)
    soup = BeautifulSoup(webpage.text, 'html.parser')
    statement_footer = soup.find_all ('footer' , attrs={'class' : 'm-statement__footer'})
    for i in statement_footer:
        link1 = i.text.strip()
        name_and_date= link1.split()
        first_name = name_and_date[1]
        last_name = name_and_date[2]
        full_name = first_name + ' ' + last_name
        authors.append(full_name)
scrape_website(page_number)
```

Fig. 8. Implementation of Extracting Author

Scrapping Statement

The statement is another feature of our research. We collect it fromthe Ploliti-fact.com [3]. Looking at this figure-9, we can see that the post statement on the PolitiFact is stored within the anchor tag a with an attribute class m-statement quote and the Anchor tag resides into the DIV tag of the HTML page.Our main aim is to access statements from the HTML structure.

With the help of the statement, we can predict which statement is true or false in our research.



Donald Trump

stated on December 4, 2023 in a Truth Social post:

The Lincoln Project is "using A.I. (Artificial Intelligence)" in its "television commercials."

By Loreben Tuquero • December 19, 2023

```
<li class="o-listicle__item"> … </li>
<li class="o-listicle__item"> … </li>
<li class="o-listicle__item"> … </li>
<li class="o-listicle__item"> … </li>
<li class="o-listicle__item"> … </li> == $0
<li class="o-listicle__item"> … </li>
<li class="o-listicle__item"> … </li>
<li class="o-listicle__item"> … </li>
<li class="o-listicle__item">
 <article class="m-statement m-statement--is-medium m-statement--f
 alse">
  <div class="m-statement__author"> … </div>
  <div class="m-statement__content">
   <div class="m-statement__body">
    <div class="m-statement__quote-wrap">
     <div class="m-statement__quote">
      <a href="/factchecks/2023/dec/19/donald-trump/donald-tru
      mp-claimed-the-lincoln-project-used-ai-t/">
       " The Lincoln Project is "using A.I. (Artificial
       Intelligence)" in its "television commercials." "
      </a>
      ::after
     </div>
    </div>
    <div class="m-statement__meter">
     <div class="c-image" style="padding-top: 89.4977168949771
```

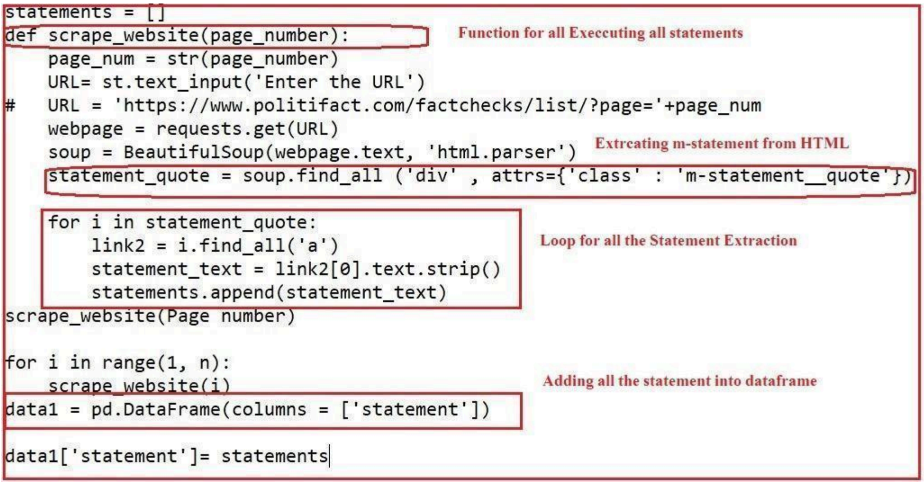
Fig. 9. Scrapping Statement

Retrieving Statement From m-statement Quote

We created the **scrape_website()** function, which takes page numbers as arguments for extracting statements from the particular page of PolitiFact.comgiven in figure-10. Next, we can assign the result of a request of that page to the variable with the request.get () method.

To work with web data, we are going to access the text-based content of web files. We can read the content of the server's response with the page.text.

Then, all the statements are appended into one dictionary object using for loop. Finally, convert all the dictionary statements into a data frame for further processing.



```
statements = []
def scrape_website(page_number):
    page_num = str(page_number)
    URL= st.text_input('Enter the URL')
#    URL = 'https://www.politifact.com/factchecks/list/?page='+page_num
    webpage = requests.get(URL)
    soup = BeautifulSoup(webpage.text, 'html.parser')
    statement_quote = soup.find_all ('div' , attrs={'class' : 'm-statement__quote'})

    for i in statement_quote:
        link2 = i.find_all('a')
        statement_text = link2[0].text.strip()
        statements.append(statement_text)
scrape_website(Page number)

for i in range(1, n):
    scrape_website(i)
data1 = pd.DataFrame(columns = ['statement'])

data1['statement']= statements
```

Function for all Execcuting all statements

Extrcating m-statement from HTML

Loop for all the Statement Extraction

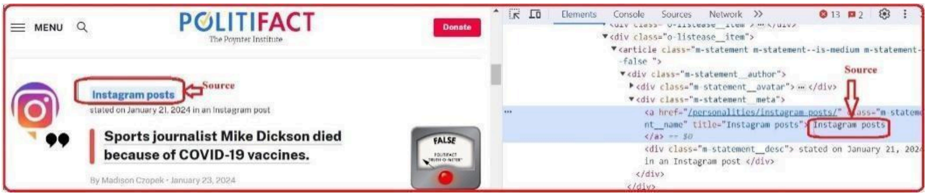
Adding all the statement into dataframe

Fig. 10. Implementation of Extracting Statement

Scrapping Source

The source is our third feature for predicting fake or real content . We stored Facebook posts, Instagram posts, Tim Scott, Social Media, Joe Biden, TikTok posts, Bloggers, and many more for our research.

The source is our third feature for predicting fake or authentic content. We stored Facebook, Instagram, Tim Scott, Social Media, Joe Biden, TikTok posts, bloggers, and more for our research. Figure-11 below shows the HTML design and source code of that page. By inspecting the source, we can extract information using the BeautifulSoup object. We mapped the desired information on the inspected HTML source code. The find method locates the first occurrence of a tag, while the find_all method scans the entire HTML code for all instances.



Instagram posts

stated on January 21, 2024 in an Instagram post

Sports journalist Mike Dickson died because of COVID-19 vaccines.

By Madison Czopek • January 23, 2024

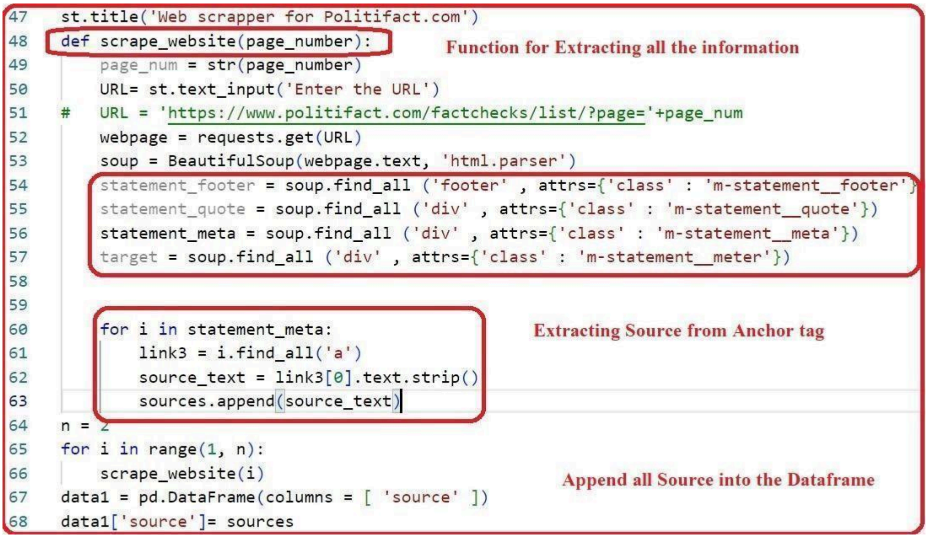
```
▼<div class="o-listease__item">
  ▼<article class="m-statement m-statement--is-medium m-statement-
  -false ">
    ▼<div class="m-statement__author">
      ▶<div class="m-statement__avatar"> ⋯ </div>
      ▼<div class="m-statement__meta">
          <a href="/personalities/instagram-posts/" class="m-stateme
          nt__name" title="Instagram posts">Instagram posts
          </a> == $0
          <div class="m-statement__desc"> stated on January 21, 2024
          in an Instagram post </div>
        </div>
      </div>
```

Fig. 11. Scrapping Source

Retrieving Source from HTML Page

We can extract sources from the HTML page of politifact.com using the **scrape_website()** function. That function takes one input as a page number and returns all the sources to the List object [5]. List object kept that information in the

Dataframe for further operations. That all the operations shown in the below Figure -12.



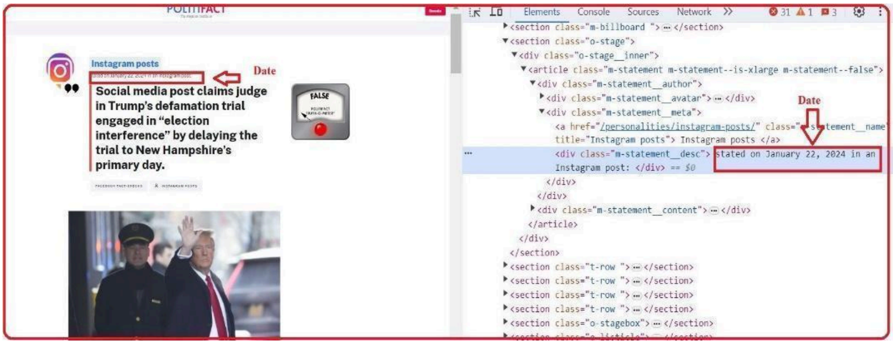
```
st.title('Web scrapper for Politifact.com')
def scrape_website(page_number):
    page_num = str(page_number)
    URL= st.text_input('Enter the URL')
#   URL = 'https://www.politifact.com/factchecks/list/?page='+page_num
    webpage = requests.get(URL)
    soup = BeautifulSoup(webpage.text, 'html.parser')
    statement_footer = soup.find_all ('footer' , attrs={'class' : 'm-statement__footer'})
    statement_quote = soup.find_all ('div' , attrs={'class' : 'm-statement__quote'})
    statement_meta = soup.find_all ('div' , attrs={'class' : 'm-statement__meta'})
    target = soup.find_all ('div' , attrs={'class' : 'm-statement__meter'})


    for i in statement_meta:
        link3 = i.find_all('a')
        source_text = link3[0].text.strip()
        sources.append(source_text)
n = 2
for i in range(1, n):
    scrape_website(i)
data1 = pd.DataFrame(columns = [ 'source' ])
data1['source']= sources
```

Function for Extracting all the information
Extracting Source from Anchor tag
Append all Source into the Dataframe

Fig. 12. Implementation of Extracting Source

Scrapping Date

As the name suggests these features one can trace the pattern of the spreading con-tent on the web. We can scrape the Date features from the Politi-fact.com website with the help of inspecting the source code. The Date Features resides in the m-statement desc class inside the div tag. We can extract these features using BeatifulSoup library as in the previous section we discussed about BeatifulSoup. That feature provides the propagation dates for the author when it circulates the fake or real content. The Scrapping dates from HTML pages are shown on below Figure-13.



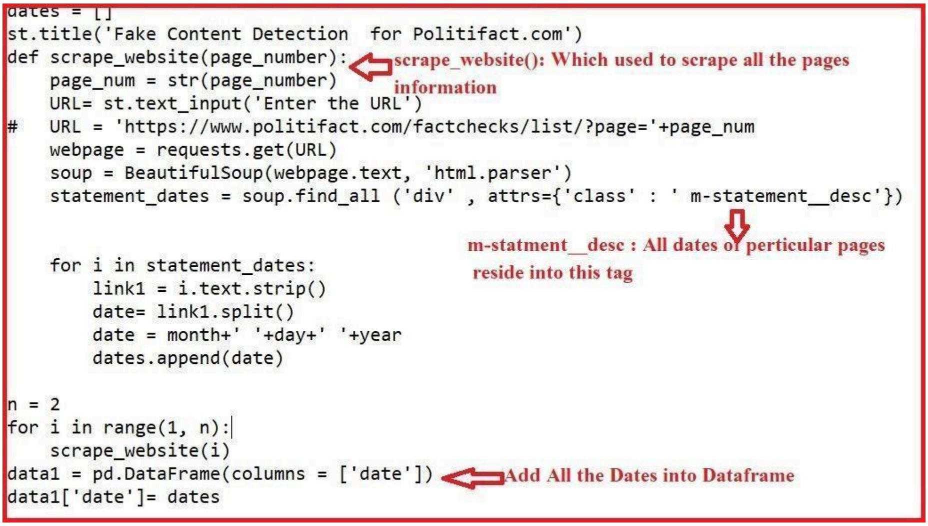
Social media post claims judge in Trump's defamation trial engaged in "election interference" by delaying the trial to New Hampshire's primary day.

```
▸<section class="m-billboard "> ⋯ </section>
▾<section class="o-stage">
  ▾<div class="o-stage__inner">
    ▾<article class="m-statement m-statement--is-xlarge m-statement--false">
      ▾<div class="m-statement__author">
        ▸<div class="m-statement__avatar"> ⋯ </div>
        ▾<div class="m-statement__meta">
           <a href="/personalities/instagram-posts/" class=" s tement__name
           title="Instagram posts"> Instagram posts </a>
           <div class="m-statement__desc"> stated on January 22, 2024 in an
           Instagram post: </div> == $0
         </div>
       </div>
      ▸<div class="m-statement__content"> ⋯ </div>
     </article>
   </div>
 </section>
▸<section class="t-row "> ⋯ </section>
▸<section class="t-row "> ⋯ </section>
▸<section class="t-row "> ⋯ </section>
▸<section class="t-row "> ⋯ </section>
▸<section class="o-stagebox"> ⋯ </section>
```

Fig. 13. Scrapping Date

Retrieving Date from HTML Page

We created a blank dictionary for storing all dates from Politifact.com. For that purpose, we created one **scrape_website ()** function that takes one input as a particular page number for scrapping all the dates from that page. Our dates stored into the m-statment desc which is classed into the div tag of that page. Then we used for loop to collect all the dates into a date object and append all the dates into the data frame. Which is shown below in Figure-14.



```
dates = []
st.title('Fake Content Detection  for Politifact.com')
def scrape_website(page_number):
    page_num = str(page_number)
    URL= st.text_input('Enter the URL')
#   URL = 'https://www.politifact.com/factchecks/list/?page='+page_num
    webpage = requests.get(URL)
    soup = BeautifulSoup(webpage.text, 'html.parser')
    statement_dates = soup.find_all ('div' , attrs={'class' : ' m-statement__desc'})


    for i in statement_dates:
        link1 = i.text.strip()
        date= link1.split()
        date = month+' '+day+' '+year
        dates.append(date)

n = 2
for i in range(1, n):
    scrape_website(i)
data1 = pd.DataFrame(columns = ['date'])
data1['date']= dates
```

scrape_website(): Which used to scrape all the pages information

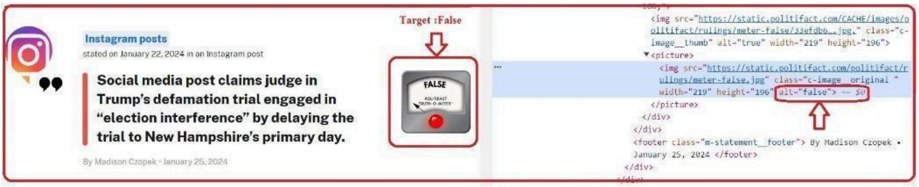
m-statment__desc : All dates of perticular pages reside into this tag

Add All the Dates into Dataframe

Fig. 14. Implementation of Extracting Date

Scrapping Target

To determine whether a sentence is true or untrue, we employ the target feature. We followed the same procedure as in the previous section used for different features. Figure-15 below shows scrapping the target variables from the HTML page.



Instagram posts

stated on January 22, 2024 in an Instagram post

Social media post claims judge in Trump's defamation trial engaged in "election interference" by delaying the trial to New Hampshire's primary day.

By Madison Czopek · January 25, 2024

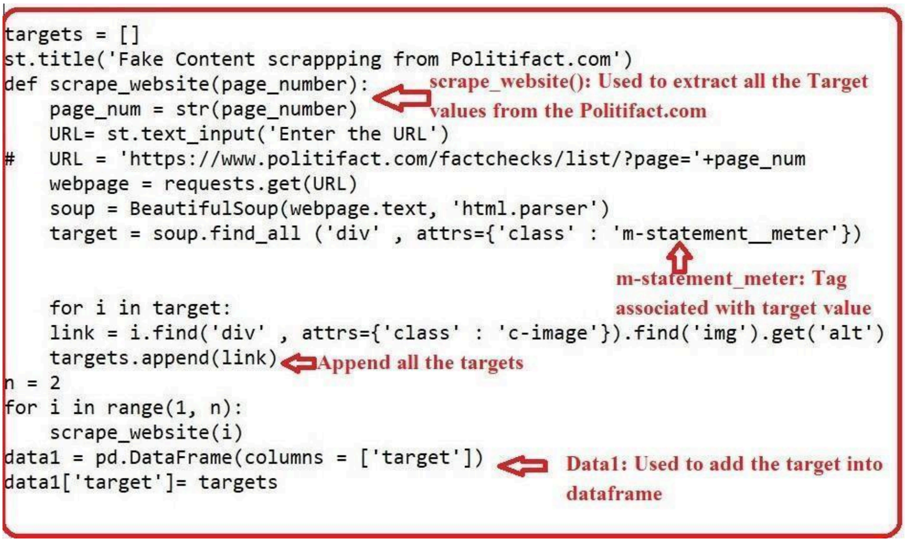
Target :False

```
    <img src="https://static.politifact.com/CACHE/images/politifact/rulings/meter-false/33efdb6….jpg." class="c-image__thumb" alt="true" width="219" height="196">
   ▼<picture>
      <img src="https://static.politifact.com/politifact/rulings/meter-false.jpg" class="c-image__original " width="219" height="196" alt="false"> == $0
    </picture>
  </div>
</div>
<footer class="m-statement__footer"> By Madison Czopek • January 25, 2024 </footer>
</div>
```

Fig. 15. Scrapping Target

Retrieving Target attribute from HTML Page

We created a blank dictionary for storing all dates from Politifact.com. For that purpose, we created one **scrape_website()** function that takes one input as a particular page number for scrapping all the dates from that page. Our dates stored into the m-statment desc which is classed into the div tag of that page and then we used loop to collect all the dates into a date object and append all the dates into the data frame which is shown below in Figure-16.



```
targets = []
st.title('Fake Content scrappping from Politifact.com')
def scrape_website(page_number):
    page_num = str(page_number)
    URL= st.text_input('Enter the URL')
#   URL = 'https://www.politifact.com/factchecks/list/?page='+page_num
    webpage = requests.get(URL)
    soup = BeautifulSoup(webpage.text, 'html.parser')
    target = soup.find_all ('div' , attrs={'class' : 'm-statement__meter'})


    for i in target:
    link = i.find('div' , attrs={'class' : 'c-image'}).find('img').get('alt')
    targets.append(link)
n = 2
for i in range(1, n):
    scrape_website(i)
data1 = pd.DataFrame(columns = ['target'])
data1['target']= targets
```

scrape_website(): Used to extract all the Target values from the Politifact.com

m-statement_meter: Tag associated with target value

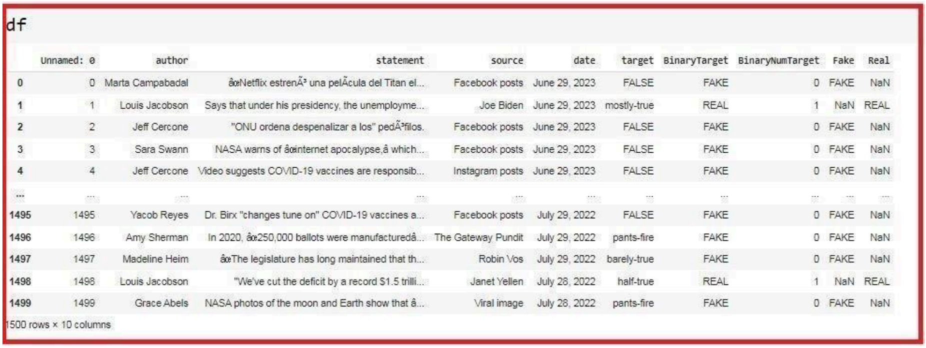
Append all the targets

Data1: Used to add the target into dataframe

Fig. 16. Implementation of Extracting Target

5 Results

We scrapped the different source, date, target, author, and statement attributes. The scrapped data stored into the csv file which shown in the Figure-17.



```
df
```

	Unnamed: 0	author	statement	source	date	target	BinaryTarget	BinaryNumTarget	Fake	Real
0	0	Marta Campabadal	âœNetflix estrenÃ³ una pelÃcula del Titan el...	Facebook posts	June 29, 2023	FALSE	FAKE	0	FAKE	NaN
1	1	Louis Jacobson	Says that under his presidency, the unemployme...	Joe Biden	June 29, 2023	mostly-true	REAL	1	NaN	REAL
2	2	Jeff Cercone	"ONU ordena despenalizar a los" pedÃ³filos.	Facebook posts	June 29, 2023	FALSE	FAKE	0	FAKE	NaN
3	3	Sara Swann	NASA warns of âœinternet apocalypse,â which...	Facebook posts	June 29, 2023	FALSE	FAKE	0	FAKE	NaN
4	4	Jeff Cercone	Video suggests COVID-19 vaccines are responsib...	Instagram posts	June 29, 2023	FALSE	FAKE	0	FAKE	NaN
...	...	...	...	...	...	...	...	...	...	...
1495	1495	Yacob Reyes	Dr. Birx "changes tune on" COVID-19 vaccines a...	Facebook posts	July 29, 2022	FALSE	FAKE	0	FAKE	NaN
1496	1496	Amy Sherman	In 2020, âœ250,000 ballots were manufacturedâ...	The Gateway Pundit	July 29, 2022	pants-fire	FAKE	0	FAKE	NaN
1497	1497	Madeline Heim	âœThe legislature has long maintained that th...	Robin Vos	July 29, 2022	barely-true	FAKE	0	FAKE	NaN
1498	1498	Louis Jacobson	"We've cut the deficit by a record $1.5 trilli...	Janet Yellen	July 28, 2022	half-true	REAL	1	NaN	REAL
1499	1499	Grace Abels	NASA photos of the moon and Earth show that â...	Viral image	July 28, 2022	pants-fire	FAKE	0	FAKE	NaN

1500 rows × 10 columns

Fig. 17. Visualization of the Dataset

6 Conclusion

We designed and implemented a web scraper for PolitiFact.com to automate the collection of data essential for detecting fake content. The scraper was meticulously crafted to extract key information, including the author, statement, source, date, and verdict of each fact-checked claim. We began by outlining a modular and scalable architecture with components for web crawling, data parsing, and storage. Implemented using Python with libraries like BeautifulSoup and requests, the scraper navigates the site, handles pagination, and adapts to structural changes, ensuring compliance with web scraping best practices. This tool provides a robust pipeline for gathering data, integral to training and validating our machine learning models for fake content detection, enhancing both efficiency and accuracy.

7 Future Scope

In the future, we aim to work with different types of platforms for scrapping the information from fact-checking websites.

References

1. Joel Christian, Sharada Valiveti, and Swati Jain. “Profiling Cyber Crimes from News Portals Using Web Scraping”. In: Futuristic Trends in Networks and Computing Technologies: Select Proceedings of Fourth International Conference on FTNCT 2021. Springer. 2022, pp. 1007–1016(2021).
2. Vaishnavi Deshmane et al. “Web Scraping for E-Commerce Website”. In: International Journal for Innovative Engineering & Management Research 13.4 (2024).
3. Zhao, Bo. "Web scraping." Encyclopedia of big data. Cham: Springer International Publishing, 2022. 951-953.
4. Lawson, Richard. Web scraping with Python. Packt Publishing Ltd, 2015.
5. Diouf, Rabiyatou, et al. "Web scraping: state-of-the-art and areas of application." 2019 IEEE International Conference on Big Data (Big Data). IEEE, 2019.
6. Thomas, David Mathew, and Sandeep Mathur. "Data analysis by web scraping using python." 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA). IEEE, 2019.
7. Uzun, Erdinç. "A novel web scraping approach using the additional information obtained from web pages." IEEE Access 8: 61726-61740(2020).
8. Milev, Plamen. "Conceptual approach for development of web scraping application for tracking information." Economic Alternatives 3 : 475-485(2017).
9. Ulbricht, Lena. "Scraping the demos. Digitalization, web scraping and the democratic project." Democratization 27.3 : 426-442(2020).
10. MS El Asikri, S Knit, and H Chaib. “Using web scraping in a knowledge environment to build ontologies using python and scrapy”. In: European Journal of Molecular & Clinical Medicine 7.03 (2020), p. 2020.

11. Moaiad Ahmad Khder. “Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application.” In: International Journal of Ad- vances in Soft Computing & Its Applications 13.3 (2021).
12. Sumit Kumar and Uponika Barman Roy. “A technique of data collection: web scraping with python”. In: Statistical Modeling in Machine Learning. Elsevier, 2023, pp. 23–36(2023).
13. Deepak Mangal and Dilip Kumar Sharma. “A Framework for Detection and Validation of Fake News via authorize source matching”. In: Micro- Electronics and Telecommunication Engineering: Proceedings of 4th ICMETE 2020. Springer. 2021, pp. 577–586(2020).
14. Chaulagain, Ram Sharan, "Cloud based web scraping for big data applications." 2017 IEEE International Conference on Smart Cloud (smartcloud). IEEE, 2017.
15. Kumbhkar Makhan, Masih Shraddha. “Comparative analysis of machine learning models for fake content detection”. In: Journal of Data Acquisition and Processing 38.1 (2023), p. 4255.
16. Kumbhkar Makhan, Masih Shraddha. Detecting false informa- tion in medical and health care domains. Https://ipindia. Gov.in/journal.htm. Patent no:202211052606, Indian Government Patent, journal No.-39/2022, 09,2022.
17. Kumbhkar Makhan Masih Shraddha. Detection of fake news text classification on covid-19 using deep learning approaches. Https://ipindia.gov.in/journal.htm**.** Patent no:202211053246,Indian Government Patent ,journal No.-38/2022,09,2022.
18. Kumbhkar Makhan Masih Shraddha. Detection of false medical reports generated by mri machine of x-ray using machine learning techniques. Https : / / ipindia . Gov . In / journal.htm. Patent no:202211053949,Indian Government Patent, journal No.-39/2022, 09,2022.
19. Kumbhkar Makhan Masih Shraddha. Healthcare misinformation detection and fact-checking: a novel approach. Https: //ipindia.gov.in/journal.htm. Patent no:202211055173,Indian Government Patent, journal No.-39/2022, 09,2022.
20. Kumbhkar Makhan Masih Shraddha. Machine learning model to identify fake news for covid-19 in health care system. Https://ipindia.gov.in/journal.htm**.** Patent no: 202211053232, Indian Government Patent ,journal No.-38/2022,09,2022.
21. Ryan Mitchell. Web scraping with Python: Collecting more data from the modern web. " O’Reilly Media, Inc.", 2018.
22. Laxmi B Rananavare et al. “Fake News Identification for Web Scrapped Data.” In: Journal of Advanced Zoology 44 (2023).
23. Kulvinder Singh, Mayank Pathak, and Aryan Jaswal Thakur. “Python- Powered Web Scraping for Data Extraction”. In: Emerging Trends in iot and Computing Technologies. CRC Press, 2024, pp. 321–329.
24. Vidhi Singrodia, Anirban Mitra, and Subrata Paul. “A review on web scrapping and its applications”. In: IEEE International conference on com- puter communication and informatics (ICCCI), pp. 1–6(2019).
25. Dinesh Kumar Vishwakarma, Deepika Varshney, and Ashima Yadav. “Detection and veracity analysis of fake news via scrapping and authenticating the web search”. In: Cognitive Systems Research 58 , pp. 217–229(2019).
26. Ferry Wahyu Wibowo, Akhmad Dahlan, et al. “Detection of Fake News and Hoaxes on Information from Web Scraping using Classifier Methods”. In: IEEE 4th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), pp. 178–183(2021).