Web Scraping of COVID-19 News Stories to Create Datasets for Sentiment and Emotion Analysis

Poojitha Thota
Department of Computer Science and Engineering,
University of Texas at Arlington
poojitha.thota@mavs.uta.edu

Ramez Elmasri
PhD, Department of Computer Science and Engineering,
University of Texas at Arlington
elmasri@uta.edu

ABSTRACT

Over the past few years, the ubiquitous usage of internet to broadcast information worldwide, has proved to be one of the best methods in making people aware about their surroundings. This has also led towards storage of vast amount of data in user interactive websites. Several news channels apart from live streaming, are using internet in such ways to convey their information. And these methods have not only benefited people to acquire regular updates but have also impacted their lives in many ways during the world awakening pandemic like COVID-19. Currently, in this pandemic situation, many leaders including federal and state government officials have shown great concern towards society by providing statements such as preventive measures, which were regularly recorded and elaborated in the form of stories by the news channels in their own websites. But, in general, the type of statement given by a leader will always affect people's opinion and behavior, which also happened during this pandemic. There were variations in number of cases of COVID-19 based upon a region leader's statements as per timeline and the people's modulating lifestyle there. This kind of relationship has motivated us towards analyzing the COVID-19 data based on sentiment and emotion involved in leader's statements, giving rise to an idea of web scraping to obtain their statements and stories from news channels. The main aim of this paper is to enlighten about different techniques and libraries used for web scraping, some challenges while designing a web scraper and finally arrive at an efficient methodology using them to extract news headlines and stories and create a new dataset that can be used in sentiment and emotion analysis. In future work, we will present the results of our experiments with the downloaded datasets.

CCS CONCEPTS

• **Information systems**; • **World Wide Web**; • **Web searching and information discovery**; • **Web search engines**; • **Web**; • **Web mining**; • **Data extraction and integration**; • **Surfacing**;

KEYWORDS

Pandemic, Sentiment and Emotion Analysis, Web Scraping

PETRA 2021, June 29–July 02, 2021, Corfu, Greece

ACM ISBN 978-1-4503-8792-7/21/06…$15.00
https://doi.org/10.1145/3453892.3461333

ACM Reference Format:
Poojitha Thota and Ramez Elmasri. 2021. Web Scraping of COVID-19 News Stories to Create Datasets for Sentiment and Emotion Analysis. In *The 14th PErvasive Technologies Related to Assistive Environments Conference (PETRA 2021), June 29–July 02, 2021, Corfu, Greece.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3453892.3461333

1 INTRODUCTION AND MOTIVATION

Data is the crucial component for any type of study or analysis on a particular context. There are huge numbers of datasets everywhere on the internet, that can help programmers in solving problems in several applications involving machine learning, deep learning, and data mining. But requirements for each project is never the same. Every application has its own target and different strategies should be applied to have a reliable output at the end. And such need leads to the idea of creating new datasets that can be used for multiple purposes later. Creating a large dataset is a tedious task if done manually. But methods like web scraping and web crawling can automate data collection and make it easier to create datasets for analysis.

The main motivation to create a new dataset arrived from the idea of analyzing how leader's statements during the period of COVID-19, has impacted on the number of COVID-19 cases on daily basis. In general, government rules the state or country, and the leaders like federal and state government officials are the ones who pass major orders and statements that the public listens to. The recent pandemic COVID-19 is a shock to the whole world and immediate decisions had to be taken by higher officials and implemented at many regions, to reduce its intensity. Many safety measures like mask mandate, six feet social distancing, staying sanitized, etc. have been introduced to prevent the spread of the virus.

But there were many areas where such measures have not been followed by the public, as there were no strict orders from the government, that lead to growth in the number of cases. Looking at such scenarios has given us an idea that there might be some relationship existing between the type of leader statement or order, its implementation, and the number of cases in that region. To judge how the leader speaks out, sentiment and emotion analysis can be performed. Based on the output, we can conclude how concerned the state official was towards the pandemic and checking the variations in number of cases at the same area can show the impact of the statements. This project mainly focuses on states present in the USA.

To perform a good analysis, the first requirement is acquiring state-wise data of leader's statements during the pandemic. This led to an idea of performing web scraping to collect such dataset

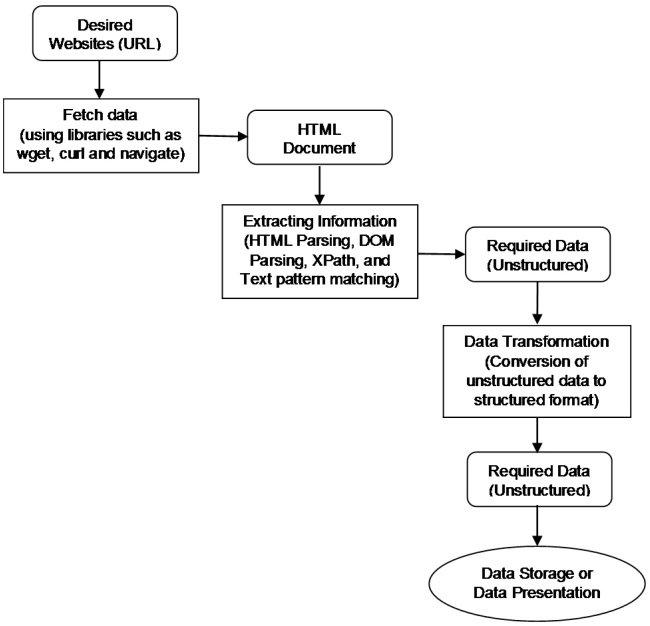

Figure 1: Overview of Web Scraping Process

from popular news sites like CNN. There are many in-built web scrapers, but most were not completely fulfilling our requirements. So, this gave us a motivation to build a different methodology for creating a web scraper with existing parsers. This will be clearly discussed in sections 3 and 4.

This paper is organized as follows, Firstly, we give an overview of web scraping process in section 2. It explains the different phases like fetching, extraction, and transformation of data. With this, we can know how a web scraper basically works.

The sections 3 and 4 give an overview of several techniques and libraries that are used for web scraping. These sections provide us with clarity about choosing libraries for certain type of operations. For building a better scraper, we have used programming language R and web scraping libraries in it. Detailed explanation about choosing these options will be provided in these sections.

There are some challenges that will be faced by any web scraper due to several factors that will be discussed in section 5. We will provide some generalized solutions embedding within the methodology that we have implemented, which we will be discussing in section 6. Finally, our results have been shown in section 6. Before concluding, the final section shows a short use cases of how the dataset can be used further.

2 OVERVIEW OF WEB SCRAPING

Web scraping is a process that involves extraction of information from various websites. It is also known as web data extraction, web data scraping, web harvesting or screen scraping [1]. In general, websites contain unstructured data, where web scraping can be used to fetch and extract the information, transform it into an understandable format like csv and finally store it into an external database. This technique can be performed manually by applying traditional methods of copy-paste, but in most of the cases, it is automatically performed to achieve efficiency. The whole process of web scraping can be split into different phases and explained in brief, as follows.

Phase-1: Fetch data

Firstly, in this step, desired websites must be selected where the data is accessed from. Then fetching can be done using HTTP protocol, an Internet protocol used to send and receive requests from a web server. In this phase, libraries such as wget, curl and navigate can be used to send a HTTP GET request to the URL of a website (desired location), from which HTML document is sent back as a response [2].

Phase-2: Extracting Information

After fetching the desired HTML documents, the next step would be extracting our required information from the website. This can be performed using several techniques such as HTML Parsing, DOM Parsing, XPath, and Text pattern matching, which will be discussed in detail in the next section.

Phase-3: Data Transformation

Once the required information is extracted from the desired locations (URL), the data will be in unstructured format. It can then be transformed into structured format such as CSV, spreadsheet, or pdf, either for presentation or storage.

The phases described above in the process of web scraping can be summarized in Figure 1

3 TECHNIQUES FOR WEB SCRAPING

To implement a web scraper, there are several approaches among which the best one can be selected based on a programmer's requirement. From the above, it can be observed that these techniques are used during the extraction phase, and they are mainly divided into

two categories Manual web scraping and Automated web scraping. This section discusses about some of the major techniques in each category and gives a scope for selecting the suitable technique among them.

3.1 Manual Web Scraping

3.1.1 Traditional copy-pasting. Traditional copy-pasting is a manual approach where data that has to be extracted from a website is manually copied as a group and pasted into a document. Then the required data would be collected from the group and arranged into a structured form. Occasionally, for less amount of data, this can be the best technique. But in the process of creation of large dataset, this technique could be tiresome and error prone as it involves lot of manual work [1, 3].

3.2 Automated Web Scraping

3.2.1 HTML Parsing. In general, parsing is the process of analyzing a string of symbols either in natural language or in computer languages or data structures conforming to the rules of grammar. The result of parsing a document is usually a tree of nodes that represent the structure of it. In HTML Parsing, after the HTML document is fetched, a tree of nodes is formed during parsing, from where the relevant information like headings in the page, title of the page, paragraphs in the page can be extracted by detecting the html nodes. Some semi-structured data query languages, such as XQuery and the HTQL, can be used to parse HTML pages and to retrieve and transform page content [1].

3.2.2 DOM Parsing. The DOM is a common platform for managing document structures. It works on XML documents and like HTML parsers, when an XML document is fetched and applied with a DOM parser, a tree structure containing all elements of the document is formed. With the help of DOM, the contents and structure of the document can be examined and used for extraction [4].

3.2.3 XPath. XPath stands for XML Path Language [4]. It is a technology that takes path expressions into account, to access nodes in a document. It can be used on XML documents to access different elements in their structure and content. XPath can also be used to navigate HTML documents. In general, after DOM parsing, XPath can be used as web scraper to scrape the website's data. It is not a language but comes in a form of expressions, that requires its own syntax.

3.2.4 Text Pattern Matching. Text pattern matching is a matching technique where regular expressions are used to match HTML tags and extract data from HTML documents [5]. Regular expressions, in general, are sequence of characters that give rise to a search pattern. As HTML is virtually composed of many strings, regular expressions can come into action here, by matching different strings. But regular expressions might not be the first choice in parsing HTML, as there is a chance of encountering mistakes such as missing tags.

Among popular techniques discussed above, based on the flexibility, and knowing about difficulties of their usage, it can be observed that HTML parsing is the most convenient technique. As there are numerous libraries and inbuilt tools available for performing this technique, it had been chosen for this project to start with scraping the news stories. The following section gives a detailed picture about different libraries containing HTML parser in python and R.

4 LIBRARIES FOR WEB SCRAPING

There are several libraries for web scraping. But being most popular languages in data science, python and R have bagged wide variety of built-in libraries that made the process easier for extracting relevant information from websites.

4.1 Libraries in Python

4.1.1 Requests. Requests is the most basic python library for web scraping. Through this, HTML requests can be made to a website server to retrieve data on the web page. This library is used in fetch phase of web scraping process, as discussed above in section-2, overview of web scraping. This python library gives high flexibility to users by providing various types of HTTP requests such as GET, POST, etc [9]. As this is basic library that can be used only for fetching web pages, it cannot be individually used to scrape data. It must be combined with other libraries like LXML and Beautiful Soup to obtain a reliable output.

LXML, a high performance, fast, HTML and XML parsing python library [10] can be used along with Requests to start parsing and extracting data from webpages. But it does not work well with poorly designed HTML documents. This makes LXML less flexible for usage when compared to other libraries, that would be discussed later in this section.

Beautiful soup is the most widely used python library for web scraping [4, 10]. For parsing HTML and XML documents, it creates a parse tree, and this uses HTML parsing technique discussed above in section 3.2. Beautiful soup has gained its popularity due to its ease of usage, but it is slower when compared to LXML. One of the major advantages is that it is suitable to any type of website. And this can be used along with requests to perform fetch and extraction phases successfully. To speed up the process, it can be combined with LXML parser.

4.1.2 Selenium. The libraries mentioned above has a limitation that they cannot work with websites designed with JavaScript. This makes them difficult to work while scraping dynamic webpages and reduces chances of automating scraping. But selenium is one of the python libraries that can overcome this difficulty.

Initially, it was made for automated testing of web applications, but later its ability to run JavaScript has extended its scope for using it as web scraping library. It consists of a web driver for rendering web pages, that can be used to perform clicks, highlights, scrolls, etc. automatically [10]. These features made it flexible for scraping dynamically populated web pages. But one of the major limitations of this library is that it loads and runs JavaScript for every page, which makes the processes slower and not suitable for large scale projects.

4.1.3 Scrapy. Scrapy is one of the most popular libraries in python. But it is not just a library, instead it can be considered as an efficient web scraping framework, which is a complete package consisting of functionalities suitable for fetching and extraction phases, discussed above in section 2. It can be used to crawl websites as it provides spider bots and gives users a facility to create their own spider bot

[11]. The fundamental points of interest of scrapy are that demands are booked and handled non concurrently, which implies that scrapy doesn't have to trust that a solicitation will be done and prepared, it can send another solicitation or do different things meanwhile [8] which speeds up the process and increases the efficiency. One of the limitations of scrapy is that it cannot handle JavaScript like selenium drivers. To be able to automate the process with scrapy, it must be combined again with other libraries like Splash, to extract data from dynamic websites.

As we can observe from the libraries mentioned above, each has its own advantages. But none of the libraries mentioned above can be used individually for complete process. At least two or three of them must be combined to obtain the complete functionality of a web scraper. And if the requirement is not satisfied, a greater number of libraries must be explored and used in python, to acquire a reliable output of extraction. This limitation gave us a scope for designing a scraper using the programming language R.

4.2 Libraries in R

There are numerous libraries in R, that were designed using statistical methods. Some of the major libraries in R have been studied for this project and proved to be sufficient for designing an efficient web scraper embedding all the functionalities.

4.2.1 Rvest. Rvest is popular package that is used for scraping data from html web pages [6]. It is inspired by libraries like beautiful soup in python. It is used along with the package magrittr in R, to have the piping functionality. Working with rvest is easier when compared to other libraries. This can be shown through an example.

Scraping data from "The Lego movie" from IMDB is used as an example here. While using rvest, first step would be downloading and parsing the file with the help of method html().

```
library(rvest)
lego_movie <- html("http://www.imdb.com/title/tt1490017
/")
```

Then pipelining can be done to extract data from css selectors that exist in the web page. Methods such as html_node() can be used to find all the nodes that match with the selector and then contents from the location can be extracted using method html_text(). The code snippet below shows the extraction of IMDB rating of the movie using these methods.

```
rating <- lego_movie %>%
html_nodes("strong span") %>%
html_text() %>%
as.numeric()
rating
#> 7.7
```

From the code snippet, it can be observed that the rating is obtained as 7.7 for "The Lego movie" in IMDB website.

As this library contains fetching and extraction phases of a web scraping process, discussed above in section 2, most of the extraction can be completed using this. But the goal to create a huge dataset requires the process to be automated. And, as Rvest does not provide this facility, it must be combined with a driver that helps in automating web scraping. R does contain such powerful library named as "RSelenium", which will be elaborated in the following section.

4.2.2 RSelenium. Rselenium is a combination of R and selenium, where it establishes a connection to Selenium Server/ Remote Selenium Server from within R [7]. Selenium as discussed above in section 4.1.4, works with JavaScript, and focuses on automating web browsers. Integrating the Rvest library along with RSelenium gives a facility for the users to access a web driver in a web scraper. Usage of this library involves few steps that would be discussed in detail in later sections of methodology implemented.

4.2.3 Stringr. Stringr is another popular package present in R, which contain wide range of set of functions that can be used for data cleaning or for string manipulations [12]. As there is text cleaning process in the final stage of web scraping process, that is Data Transformation phase, this package can be of great use for dealing such operations.

From the packages discussed above, we can observe that packages in R have more features when compared to that present in python. Instead of using several libraries to achieve functionality, R provides a facility for users to complete the operations with minimal usage of packages and methods. These are some of the reasons with which we can decide the languages and packages to be used. Based on all the advantages mentioned above, R had been selected and the packages rvest and RSelenium were used to achieve web scraper for this project. Their detailed integration and working will be shown in the methodology later in section 6.

5 CHALLENGES WHILE DESIGNING WEB SCRAPERS

Websites are designed in different ways and the goal of creation would include many factors such as user interactive, way of presenting large datasets, including latest features for styling, etc. And as the number of features increase, the complexity of the website increases, and this in turn challenges the web scrapers in many ways. But our goal is to design a web scraper that is suitable to any type of website. This section describes about some of the major challenges faced while designing an efficient web scraper, and brief description of solutions that can be used to overcome these challenges. Detailed description of solutions using code snippets will be given in the later section consisting of implementation.

5.1 Infinite Scrolling Webpages

Infinite scrolling is a technique used in websites where users scroll without ever reaching the end of the page. In such scenarios, new content gets loaded as user scrolls down. Several websites like Facebook, Twitter, etc. have included infinite scrolling as their main feature. The inclusion of this feature totally depends on the nature of the website and the requirement on which it is designed. For example, websites such as blogs, or others that give information about an organization are built towards a motive to provide information to people. Such websites would not require this feature. But the websites that intend to be browsed and explored by users tries to engage users with their information. This kind of websites embed this feature into the program, so that user invests time in the page. Our project mainly uses CNN website for scraping data. For news

sites like this, the purpose is to display the latest and most relevant stories first, here infinite scrolling can ensure users to have a look at latest stories, instead of all headlines and stories at once. One of the solutions can be to scroll the page multiple times, but that involves re-loading each time, the page gets scrolled. But this can be resolved using RSelenium driver which will be clearly shown in later sections of implementation.

5.2 Data Repository

In general, data extraction at large scale can give rise to huge amount of information. If data repositories are not designed properly, filtering, and organizing this kind of data could become cumbersome and time-consuming task.

5.3 Data Quality Assurance

Sometimes, extracted data can have missing information and may not match with a pre-defined template. In such cases, data accuracy could be reduced. To resolve such difficulties, quality assurance test should run on the data and each field must be validated. These tests can be done automatically using selenium drivers, but in some cases, manual testing could provide reliable results.

As we can see, the above-mentioned challenges are regularly faced during the process of web scraping. And having a target to scrape news sites like CNN, one of the major challenges like infinite scrolling webpages, has been faced in this project. But it has been resolved again with effective solution while programming the web scraper. The next section includes the methodology and way of implementation of the web scraper by overcoming such challenges.

6 METHODOLOGY, IMPLEMENTATION AND RESULTS

There are many web scrapers widely available, but every project has its own requirements, and scrapers should be designed accordingly. This project deals with scraping news stories and news headlines, which has its own set of challenges. Based on these, a methodology is implemented using programming language R and its interface RStudio. The whole process is explained through steps in detailed, as follows.

6.1 Selecting Packages

First step of the web scraping process includes selection of relevant packages. As we have discussed above in section 4 about libraries, we have chosen the programming language to be R and suitable packages to be “rvest”, “RSelenium”. And the package “stringr” can be used to manipulate and clean data. To use the packages into scraper program, they should be installed and downloaded before moving forward. This can be done with following commands,

```
install.packages("rvest")
install.packages("RSelenium")
install.packages("stringr")
```

To add them into our scraper program, we can use the following commands,

```
library(RSelenium)
library(rvest)
library(stringr)
```

After adding the packages, they are ready to be used and all the features, methods under them can be used.

6.2 Connecting to Selenium Driver

In R, RSelenium is used to establish a connection between Selenium Server and R. This allows us to automate the web scraping process. But firstly, to drive a browser on same machine that RSelenium is running on, we will need to have a Selenium server running on that machine. To run a selenium driver through R, the function “rsDriver” can be used, which manages the binaries required for running a Selenium server [7].

```
rD <- rsDriver(browser=c("firefox"))
```

The above command indicates that selenium driver and browser “Mozilla Firefox” has started. The value of the driver would be a list containing server and client. Now the client, which is indirectly the browser can be stored in a variable and used further.

```
driver <- rD$client
```

To connect and disconnect from the server, the methods open and close can be used, as shown below.

```
driver$open()
driver$close()
```

6.3 Navigating URL’s

Recollecting the phases of web scraping process from section 2, the first phase involves finding a URL and fetching data from it. This paper aims to collect news stories and news headlines during COVID-19, i.e., starting from March 2020. As it focuses mainly on CNN news sites, the relevant URL’s can be found, which contains international news related to COVID-19. The URL can be shown as follows,
https://www.cnn.com/world/live-news/coronavirus-pandemic-intl-04-17-20/index.html

From the above URL, we have observed that the modulating part is the date of the pandemic. Each day in period of pandemic has its own webpage, which made the process of web scraping easier. As we know the start date of pandemic, we can set end date and create a sequence of dates. This can be shown as follows,

```
dates=format(seq(as.Date("2020-03-25"),
as.Date("2020-12-10"), by="days"),
format="%m-%d-%y")
```

As we have a set of dates now, we can iterate the process to all webpages. We can navigate through the webpages using the method “navigate” by taking URL as argument. The commands can be shown as follows,

```
url=paste(c("https://www.cnn.com/world/
live-news/coronavirus-pandemic-",
dates[d],"-intl/index.html"),collapse="")
driver$navigate(url)
```

The navigate function opens the webpages on our selected browser and waits for the next action input from the user or from the program.

6.4 Data Extraction from Webpages

As the relevant URL's have been collected, data can be extracted from these sources. Here, the HTML parsers come into play. The method that acts as parser is getPageSource(), which gives the complete set of html nodes present in the webpage. The command to request from client is shown as follows,

```
page_source<-driver$getPageSource()
```

Now from the obtained page source, we can acquire the desired content such as dates, headlines, and stories under a headline. This can be done using the function "read_html". The whole command is shown as follows,

```
headlines<-read_html(page_source[[1]])%>%
html_nodes("h2") %>% html_text()
stories<-read_html(page_source[[1]])%>%
html_nodes("article") %>% html_text()
headlines<-read_html(page_source[[1]])%>%
html_nodes(".date") %>% html_text()
```

From the commands above, we can observe that headlines, stories and their corresponding dates have been obtained using the html nodes, h2, article and .date respectively. This extraction can be iterated to all the pages using a for loop.

6.5 Handling Infinite Scrolling Webpages

The extraction of data can be done successfully. But the CNN news site that is being used in this scenario is an infinite scrolling webpage, as discussed above in section 5.1. This should be handled in order to get the complete information existing in the page.

As we know, for infinite scrolling pages, from section 5.1, more information is loaded when page is scrolled down. One way to handle it is by scrolling down with the help of method "executeScript()", calculating the scroll height and finally updating it in a variable. Then the newly updated height can be compared to the initial height and start scrolling operation to that extent. This process can be stored in a function or it can be iterated using loops, and the complete loop has been shown as a code snippet,

```
last_height = 0
  repeat {    driver$executeScript("window.scrollTo
(0,document.body.scrollHeight);")
    Sys.sleep(10)
    updated_height = driver$executeScript
    ("return document.body.scrollHeight")
    if(unlist(last_height)==
unlist(updated_height))
    {break}
else {last_height=updated_height}
```

In the code snippet above, "sleep" function is used, which gives time for a webpage to be loaded completely in the new scroll area.

The above-mentioned method can be embedded into any project in common, where the scenario of infinite scrolling webpages exists.

6.6 Data Transformation

From the overview of web scraping, discussed above in section 2, the last phase of the process is "Data Transformation", where the extracted data will be transformed into desired format like csv or spreadsheet.

In this project, a csv is desired. The method "write.csv()" can be used to complete the operation and store the data into a csv file.

6.7 Closing the Client

Finally, after completion of scrapping, we can close the driver using "close()" method, which closes the connection established between R, selenium, and web browser.

```
driver$close()
```

6.8 Results and Accuracy

From the methodology discussed above, we can observe that it includes all phases of web scraping process. But the result desired in web scraping should be transformed and structured data. After transforming into csv file, the output of top 5 rows of news stories and headlines collected on August 13th, 2020, can be shown in the Table 1 as follows.

As the news headlines and stories have been collected from such websites where only COVID-19 related news exists, all the headlines had to be scraped to get complete set of headlines on that day. Therefore, the accuracy is 100% as all the targeted news stories and headlines were obtained for each day.

7 EXAMPLE APPLICATION OF THE DATASET

The dataset obtained in the previous section, can be used in many ways as we mentioned in section 1. This section provides a glance about how this dataset can be used further in sentiment and emotion analysis, by showing an example.

As we mentioned in section 1, after obtaining the data, the next process is to analyze the relationship between leader's statements and COVID-19 data. To show this, we have filtered a small sample of data related to the state of Georgia and statements provided by its leaders like the Governor of the state, Mayors of several cities, etc. The existing dataset is in the form of Table 2, and stores attributes such as Headline Region, Type of Leader, and Leader Name.

After preparing the dataset, the sentiments and emotions can be found for each headline and each story and can be stored as attributes into the dataset. These can be then used for analysis. Table 3 shows headlines and stories along with additional attributes for sentiments and emotions, which have been already predicted using a machine learning model.

Once a dataset is ready, we can start analyzing several aspects like most unconcerned statements given by leaders, types of emotions in their positive and negative statements, graphical analysis of both news statements and COVID-19 data in time series. Some of these outputs are displayed as follows as examples.

```
Most Unconcerned and negative statements of
Governor-State of Georgia:
Statement Date: 07-01-2020
Headline: Georgia's governor said they
could reopen. More than 50 restaurateurs
said in a newspaper ad they're not ready.
Statement Date: 07-16-2020
Headline: Georgia has no plan to
order people to wear masks, governor says
```

Table 1: Output showing top 5 news stories and headlines on August 13th, 2020

corona_dates	Headlines	Stories
08-13-20	Through violence, the pandemic. . .	Chicago's West Englewood neighborhood. . .
08-13-20	Venezuela reports highest. . .	Venezuela reported 1,281 new Covid-19 cases. . .
08-13-20	Japan reports 1,177 new. . .	Japan reported 1,177 new cases of Covid-19 and. . .
08-13-20	New Zealand reports 12 new cor. . .	New Zealand reported 12 new coronavirus cases. . .
08-13-20	Trump coronavirus adviser claim. . .	Dr. Scott Atlas listens as President Trump speaks. . .

Table 2: Sample of dataset showing statements provided by leaders in state of Georgia

Statement_Dates	Headlines	Stories	Headline Region	Type of Leader	Leader Name
07-01-20	Georgia has no plan to order people to wear. . .	Gov. Brian Kemp speaks at a press conference in. . .	Georgia	Governor	Brian Kemp
07-08-20	Atlanta mayor says she is signing an order to. . .	Atlanta Mayor Keisha Lance Bottoms said she. . .	Georgia	Mayor-Atlanta	Keisha Lance Bottoms
07-23-20	Atlanta mayor says she's working with. . .	Atlanta Mayor Keisha Lance Bottoms, speakin. . .	Georgia	Mayor-Atlanta	Keisha Lance Bottoms
07-31-20	Georgia governor extends public hea. . .	Gov. Brian Kemp signed two executive orders ext. . .	Georgia	Governor	Brian Kemp

Table 3: Sample of dataset showing statements provided by leaders in state of Georgia with additional attributes sentiments and emotions

Statement Dates	Headlines	Stories	Headline Region	Type of Leader	Leader Name	Sentiment Headline	Emotion Headline	Sentiment Story	Emotion Story
07-01-20	Georgia has no plan to order. . .	Gov. Brian Kemp speaks at a. . .	Georgia	Governor	Brian Kemp	Negative	Unconc. . .	Negative	Sad, Unc. . .
07-08-20	Atlanta mayor says she's si. . .	Atlanta Mayor Keisha Lance. . .	Georgia	Mayor-Atlanta	Keisha Lance. . .	Positive	Sad, Unconc. . .	Positive	Angry, Unc. . .
07-23-20	Atlanta mayor says she's wo. . .	Atlanta Mayor Keisha Lance. . .	Georgia	Mayor-Atlanta	Keisha Lance. . .	Positive	Safety	Positive	Sad, Unc. . .
07-31-20	Georgia governor exte. . .	Gov. Brian Kemp signed two exe. . .	Georgia	Governor	Brian Kemp	Positive	Safety	Positive	Safety, Unc...

The above output shows the most unconcerned statements and negative statements given by Governor of State of Georgia.

As an example, output showing different emotions in positive statements given by governor of state of Georgia is given as follows in Figure 2

As an example, output showing different emotions in positive statements given by governor of state of Georgia is given as follows in Figure 3

The above figures are provided as an example in analysis to see what type of emotions the leader had during the statements, split into main categories of sentiment as positive and negative.

In this way, the acquired dataset can be used to analyze several aspects in these types of scenarios.

8 CONCLUSION

There is a requirement for large datasets everywhere. Based on the features to be extracted, either new data must be collected, or existing data must be upgraded regularly. In this paper we have presented a unique methodology to scrape data from popular news sites like CNN, aiming for its future application in sentiment and emotion analysis. There are many challenges that are to be faced in the world of websites during the process of web scraping. This paper also gives a glance about them and includes generalized solutions that can be embedded into the main method itself. We hope that our methodology and information outlined in this paper helps many users in creating new datasets.

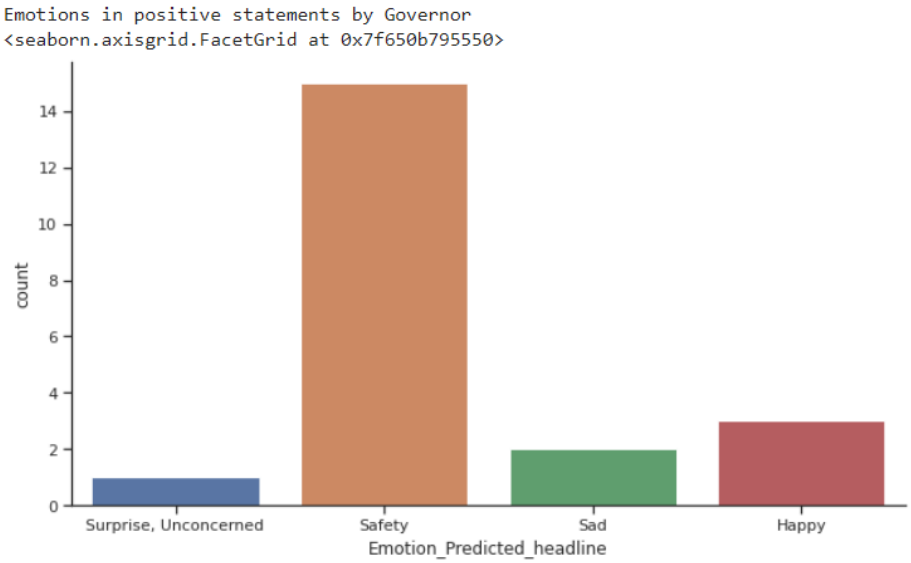

Figure 2: Figure showing different Emotions in Positive Statements given by Governor, State of Georgia

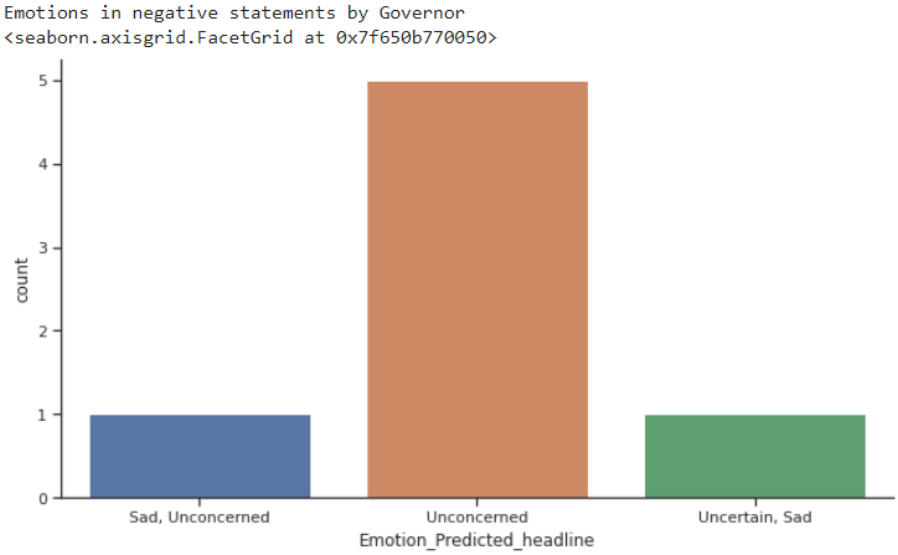

Figure 3: Figure showing different Emotions in Negative Statements given by Governor, State of Georgia

REFERENCES

[1] SCM de S Sirisuriya. 2015. A Comparative Study on Web Scraping. In Proceedings of 8th International Research Conference, KDU., http://ir.kdu.ac.lk/handle/345/1051

[2] Daniel Glez-Pena., Analia Lourenco., Hugo Lopez-Fernandez., Miguel Reboiro-Jato., Florentino Fdez-Riverola., 2013. Web scraping technologies in an API world. Briefings in Bioinformatics, Volume 15, Issue 5, September 2014, Pages 788–797., https://doi.org/10.1093/bib/bbt026

[3] Saurkar., Anand V., Kedar G. Pathare., and Shweta A. Gode., 2018. An Overview on Web Scraping Techniques and Tools. International Journal on Future Revolution in Computer Science & Communication Engineering 4.4 (2018): 363-367., http://www.ijfrcsce.org/download/browse/Volume_4/April_18_Volume_4_Issue_4/1524638955_25-04-2018.pdf

[4] Emil Persson., 2019. Evaluating tools and techniques for web scraping. Master's Thesis, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology

[5] Hamza moh., Salem and Manuel Mazzara., 2020. Pattern Matching-based scraping of news websites. Journal of Physics: Conference Series., https://iopscience.iop.org/volume/1742-6596/1694

[6] Hadley Wickham., RStudio. 2015. Easily Harvest (Scrape) Web Pages. https://rvest.tidyverse.org/ ., https://github.com/tidyverse/rvest

[7] John Harrison., Ju Yeong Kim., 2020. R Bindings for 'Selenium WebDriver'. http://docs.ropensci.org/RSelenium

[8] D. M. Thomas and S. Mathur., 2019. Data Analysis by Web Scraping using Python. In 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2019, pp. 450-454, doi: 10.1109/ICECA.2019.8822022

[9] Kenneth Reitz., 2015. Requests: HTTP for Humans., https://requests.readthedocs.io/en/master/

[10] Ryan Mitchell., Revised in 2018. Web Scraping with Python, 2nd Edition. Book published by O'Reilly, Inc. https://www.oreilly.com/library/view/web-scraping-with/9781491985564/
[11] Scrapy developers. Revised in 2020. Scrapy Tutorial. https://docs.scrapy.org/en/latest/intro/tutorial.html
[12] Hadley Wickham, RStudio. 2019. stringr, Simple, Consistent Wrappers for Common String Operations. https://www.rdocumentation.org/packages/stringr/versions/1.4.0